RealView® ICE and RealView Trace

Version 3.0

User Guide

ARM®

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.
ARM DUI 0155F

RealView ICE and RealView Trace
User Guide

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Change
December 2002	A	First release
January 2004	B	Updated for RealView ICE v1.1
August 2004	C	Updated for RealView ICE v1.2
May 2005	D	Updated for RealView ICE v1.4
November 2005	E	Updated for RealView ICE v1.5
June 2006	F	Updated for RealView ICE v3.0

Proprietary Notice

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

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

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

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

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`

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Contents
RealView ICE and RealView Trace User Guide

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

List of Tables
RealView ICE and RealView Trace User Guide

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

List of Figures
RealView ICE and RealView Trace User Guide

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Preface

This preface introduces the *RealView ICE and RealView Trace User Guide*. It explains the structure of the user guide and lists other sources of information that relate to:

- RealView® ICE v3.0 firmware and host software
- RealView Trace v1.0
- RealView Debugger.

This preface contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

About this document

This document describes the ARM® RealView ICE (RVI) run control unit, the RealView Trace data capture unit, and the software that enables you to use them.

Intended audience

This document is written for those who are using RealView ICE and RealView Trace with *RealView Development Suite* (RVDS) or RealView Debugger with the ARM Developer Suite™ (ADS). It is assumed that you are a software engineer with some experience of the ARM architecture, or a hardware engineer designing a product that is compatible with RealView ICE.

Parts of this document assume that you have some knowledge of *Joint Test Access Group* (JTAG) technology. If you require more information on JTAG, see *IEEE Standard 1149.1-2001*, available from the *Institute of Electrical and Electronic Engineers* (IEEE). For more information, see the IEEE website at `http://www.ieee.org/`.

Organization

This document is organized into the following chapters and appendices:

Chapter 1 *Introduction*

Read this chapter for a description of:

- what is provided in the RealView ICE and RealView Trace products
- the purpose of the EmbeddedICE logic within the CPU.

Chapter 2 *Getting Started*

This chapter provides information on how to start working with RealView ICE. It includes the hardware and software system requirements, and how to connect up the hardware.

Chapter 3 *Configuring RealView ICE Networking*

This chapter describes how to configure the network settings for your RealView ICE run control unit. If you are using a TCP/IP connection, you must configure the network settings before you can use the unit for debugging. If you are using a USB connection, you do not have to configure the network settings.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 4 ***Configuring a RealView ICE Connection***

This chapter describes how to configure a connection that is made using a RealView ICE run control unit.

Read this chapter in conjunction with the RealView Debugger user documentation.

Chapter 5 ***Debugging with RealView ICE***

This chapter describes how to:

- change the behavior of RealView ICE using internal variables
- implement breakpoints
- access the EmbeddedICE logic directly.

You must read this chapter in conjunction with the RealView Debugger documentation suite.

Chapter 6 ***Using RealView Trace***

This chapter describes RealView Trace and tells you how to connect the parts of RealView Trace and RealView ICE together. It also tells you where to find information on using RealView Trace with RealView Debugger.

Chapter 7 ***Managing the RealView ICE Software***

This chapter describes how to manage and update the software that is installed on the RealView ICE run control unit.

Chapter 8 ***Configuring RealView ICE for GDB***

This chapter provides information on the basic steps required to configure the RealView ICE unit to a state where you can begin debugging your image using the *GNU Debugger* (GDB).

Chapter 9 ***System Design Guidelines***

This chapter provides information about designing ARM architecture-based ASICs and PCBs that can be debugged using RealView ICE.

It includes:

- suggested clocking and reset circuit diagrams
- information on how to chain *Test Access Port* (TAP) controllers
- suggested physical connector types and pinouts
- a description of logic voltage level adaptation
- information on how power consumption varies with supply voltage.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Appendix A *JTAG Interface Connections*

This appendix describes and illustrates the JTAG pin connections.

Appendix B *User I/O Connections*

This appendix describes and illustrates the additional input and output connections provided in RealView ICE.

Appendix C *RealView Trace Interface Connections*

This appendix describes and illustrates the RealView Trace pin connections.

Appendix D *Designing the Target Board for Tracing*

This appendix describes the properties of a target board that can be connected to RealView Trace.

Appendix E *Hardware Variants*

This appendix describes the differences between the RealView ICE v3.0 hardware unit and its predecessors.

Typographical conventions

The following typographical conventions are used in this document:

bold — Highlights ARM processor signal names within text, and interface elements such as menu names. Can also be used for emphasis in descriptive lists where appropriate.

italic — Highlights special terminology, cross-references, and citations.

`monospace` — Denotes text that can be entered at the keyboard, such as commands, file names and program names, and source code.

`mono`space — Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.

`monospace italic`

Denotes arguments to commands or functions where the argument is to be replaced by a specific value.

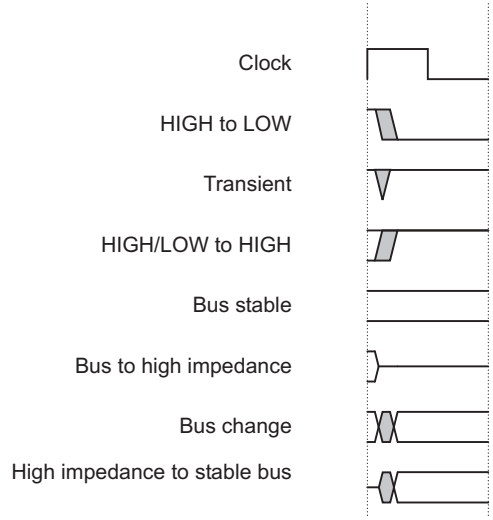
`monospace bold`

Denotes language keywords when used outside example code.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Timing diagram conventions

The following diagram shows the components used in the timing diagrams contained in this manual. Any variations are clearly labeled when they occur.



Key to timing diagram conventions

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

Further reading

This section lists publications by ARM® Limited, and by third parties, that are related to this product.

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

ARM publications

This document contains information that is specific to RealView ICE. The following documents also relate specifically to RealView ICE:

- *ARM RealView ICE Installation Guide* (ARM DSI 0017)

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- The RealView ICE `readme.html` file, supplied on the RealView ICE distribution CD, and installed beneath the `Product` subdirectory of the host software.

Also see the following books in the RealView Debugger documentation suite:

- *RealView Debugger Essentials Guide* (ARM DUI 0181)
- *RealView Debugger User Guide* (ARM DUI 0153)
- *RealView Debugger Target Configuration Guide* (ARM DUI 0182)
- *RealView Debugger Command Line Reference Guide* (ARM DUI 0175)
- *RealView Debugger Extensions User Guide* (ARM DUI 0174).

The following documentation provides general information on the ARM architecture, processors, associated devices, and software interfaces:

- *ARM Reference Peripheral Specification* (ARM DDI 0062).

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

See the following book for a description of the *Debug Communications Channel* (DCC):

- *RealView Compilation Tools Developer Guide* (ARM DUI 0203).

Other publications

The following publications might also be useful to you:

- *IEEE Standard Test Access Port and Boundary Scan Architecture* (IEEE Std. 1149.1-2001) describes the JTAG ports with which RealView ICE communicates
- Steve Furber, *ARM system-on-chip architecture* (2nd edition, 2000). Addison Wesley, ISBN 0-201-67519-6.

For more information about CEVA-Oak, CEVA-TeakLite, and CEVA-Teak DSP processors from CEVA, Inc. see `http://www.ceva-dsp.com`.

For more information about the ZSP400 DSP processor from the ZSP division of LSI Logic, see `http://www.zsp.com`.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Feedback

ARM Limited welcomes feedback on RealView ICE and RealView Trace, and on the documentation.

Feedback on RealView ICE and RealView Trace

If you have any problems with RealView ICE and RealView Trace, contact your supplier. To help us provide a rapid and useful response, give:

- the versions of RealView ICE software and firmware that you are using
- details of the platforms you are using, including both the host and target hardware types and operating system
- where appropriate, a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- if possible, sample output illustrating the problem.

Feedback on this document

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

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

General suggestions for additions and improvements are also welcome.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 1
Introduction

This chapter introduces RealView® ICE v3.0, RealView Trace v1.0 and GDB debugging, and describes the software components. It contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.1 About RealView ICE and RealView Trace

RealView ICE v3.0 is an EmbeddedICE® logic debug solution from ARM® Limited. It enables you to debug software running on:

- ARM processor cores that include the EmbeddedICE logic.
- the following *Digital Signal Processor* (DSP) cores:
  - CEVA-Oak, CEVA-TeakLite (revisions B and C), and CEVA-Teak (revisions A and B) from CEVA, Inc. (previously known as DSP Group)
  - ZSP400 from the ZSP Division of LSI Logic.

RealView ICE provides the software and hardware interface between a debugger running on a Windows or Red Hat Linux host computer, and a *Joint Test Action Group* (JTAG) *IEEE Standard 1149.1-2001* port on the target hardware.

The RealView Trace unit works in conjunction with ARM RealView ICE to provide real-time trace functionality for software running on *System-on-Chip* (SoC) devices with deeply embedded processor cores that contain the *Embedded Trace Macrocell*™ (ETM™) logic.

Note

RealView Trace v1.0 is only available for Windows platforms.

You can use RealView ICE and RealView Trace with systems that contain one or more ARM processor cores. RealView ICE also supports the *Embedded Trace Buffer*™ (ETB™) for capturing small amounts of trace information at high core clock speeds. See Chapter 6 *Using RealView Trace* for more information.

1.1.1 RealView ICE product contents

The RealView ICE product comprises:

- A run control unit that connects to your target board over a JTAG interface and to your PC using either USB or Ethernet.
- Mains cables and a power supply that powers the run control unit.
- An Ethernet cable.
- A USB cable.
- Two alternative cables that connect the RealView ICE run control unit to the target JTAG connector:
  - a short 20-way ribbon cable

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

  - a long 40-way ribbon cable and a *Low Voltage Differential Signaling* (LVDS) 40-way to 20-way probe.

- A 20-way to 14-way adaptor, for targets that use a 14-way *Insulation Displacement Connector* (IDC) box header.

  Caution

  Before using this adaptor, see *Using nonstandard connectors* on page 2-9.

- Software on CD-ROM that enables a debugger to communicate with the run control unit, and to configure and manage the run control unit.
- Documentation, including:
  - a software Installation Guide, supplied as a CD insert
  - a printed copy of this User Guide
  - online versions of this User Guide in Dynatext (Windows only) and PDF formats
  - online help
  - a packing list
  - a registration card.

1.1.2 RealView Trace product contents

The RealView Trace product (purchased separately) additionally comprises:

- the RealView Trace capture unit
- a cable to connect the RealView Trace capture unit to a trace port
- a logic grabber cable
- spacers for attachment to the RealView ICE unit.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.2 Availability and compatibility

RealView ICE and RealView Trace are available from ARM Limited and its resellers.

Contact ARM Limited directly regarding OEM licenses.

RealView ICE v3.0 and RealView Trace v1.0 are compatible with RVDS v3.0.

The RealView ICE software for the host computer is compatible with the following operating systems:

- Windows 2000 (service pack 4 or later), or Windows XP Professional (service pack 2, or later)
- Red Hat Enterprise Linux 3 and Red Hat Enterprise Linux 4.

Note

RealView Trace is only available for Windows platforms.

RealView ICE provides:

- The ability to access the target.
- Tools to configure RealView Debugger so that it can connect to the target through RealView ICE. RealView Debugger provides the user interface items, such as register windows and disassemblers, that make it possible to debug your application.

For more information on compatibility with target hardware, see the documentation supplied with your hardware.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.3 Introduction to EmbeddedICE logic and debug extensions

The EmbeddedICE logic and the ARM processor debug extensions enable RealView ICE to debug software running on an ARM processor. This section describes the basic principles of this operation:

- *Debug extensions to the ARM core*
- *The EmbeddedICE logic* on page 1-6
- *How the EmbeddedICE debug architecture differs from a debug monitor* on page 1-6.

Note

To determine whether a specific ARM processor has support for JTAG debugging, see its datasheet or technical reference manual.

For information on RealView Trace, see Chapter 6 *Using RealView Trace*.

1.3.1 Debug extensions to the ARM core

The debug extensions consist of several scan chains around the processor core, and some additional signals that are used to control the behavior of the core for debug purposes. The most significant of these additional signals are:

BREAKPT This core signal enables external hardware to halt processor execution for debug purposes. When HIGH during an instruction fetch, the instruction is tagged as breakpointed, and the core stops if this instruction reaches the execute stage of the pipeline.

DBGRQ This core signal is a level-sensitive input that causes the CPU core to enter debug state when the current instruction has completed.

DBGACK This core signal is an output from the CPU core that goes HIGH when the core is in debug state so that external devices can determine the current state of the core.

RealView ICE uses these, and other signals, through the debug interface of the processor core, for example by writing to the control register of the EmbeddedICE logic. For more details, see the section that describes the debug interface support of the ARM datasheet or technical reference manual for your core (for example, the *ARM7TDMI (Rev 4) Technical Reference Manual*).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.3.2 The EmbeddedICE logic

The EmbeddedICE logic is the integrated on-chip logic that provides JTAG debug support for ARM cores. EmbeddedICE-RT is a superset of EmbeddedICE that includes extensions supporting real-time debug, including setting breakpoints on a running target.

The EmbeddedICE logic is accessed through the TAP controller on the ARM core using the JTAG interface. See Chapter 9 *System Design Guidelines* for details of designing this into your own target.

The EmbeddedICE logic consists of:

- two or more breakpoint units
- a control register
- a status register
- a set of registers implementing the DCC link.

You can program one or both of the breakpoint units to halt the execution of instructions by the ARM CPU core. Execution is halted when a match occurs between the values in the breakpoint registers and the values currently appearing on the address bus, data bus, and selected control signals.

You can mask any bit to prevent it from affecting the comparison. Both breakpoint units can be configured to be a data breakpoint (monitoring data accesses) or an instruction breakpoint (monitoring instruction fetches).

For more information, see the relevant section of the appropriate ARM datasheet or technical reference manual.

1.3.3 How the EmbeddedICE debug architecture differs from a debug monitor

A debug monitor is an application that runs on your target hardware in conjunction with your application, and requires target resources (for example, memory, access to exception vectors, and timers) to be available.

The EmbeddedICE debug architecture requires almost no resources. Rather than being an application on the board, it works by using:

- additional debug hardware within the core, to enable the host to communicate with the target
- an external run control unit that buffers and translates the core signals into something that is usable by a host computer.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

The EmbeddedICE debug architecture enables debugging to be as non-intrusive as possible:

- the target being debugged requires very little special hardware to support debugging
- in most cases you do not have to set aside memory for debugging in the system being debugged and you do not have to incorporate special software into the application
- execution of the system being debugged is only halted when a breakpoint unit is triggered, or you request that execution is halted.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.4 Introduction to the RealView ICE components

This section introduces the components of the RealView ICE product, and describes how they fit together. It contains the following sections:

- *The RealView ICE run control unit*
- *The RealView ICE firmware* on page 1-9
- *The RealView ICE host software* on page 1-10.

See Chapter 6 *Using RealView Trace* for information on the RealView Trace components.

1.4.1 The RealView ICE run control unit

The RealView ICE run control unit provides the hardware to enable a computer to control multiple JTAG capable devices. The unit has ports at one end for connecting to the host computer and to a power source. These ports are shown in Figure 1-1.

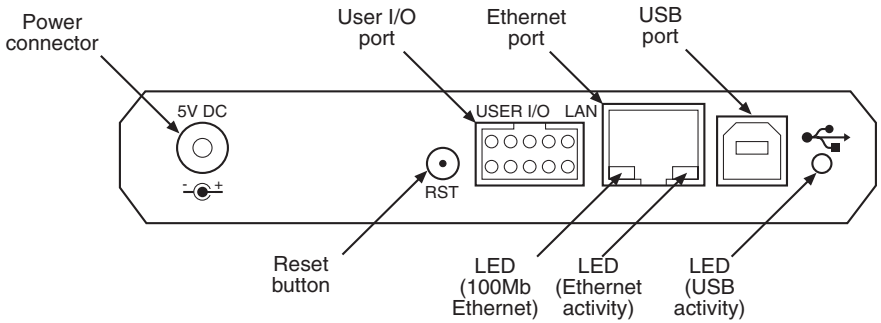


Figure 1-1 Ports for connecting to the host computer

The RST button is used to reset the RealView ICE unit when required, and returns RealView ICE to its power-up state. Using the RST button in this way does not reset the target. This button should not be confused with the Reset button mentioned in *Carrying out a real reset* on page 5-16, which is located on the target board itself.

The LEDs at the bottom of the Ethernet port display information about Ethernet speed and activity:

- The green LED shows the Ethernet speed. When Off, it indicates a speed of 10Mb/s, and when On indicates a speed of 100Mb/s.
- The yellow LED indicates that activity is taking place.

The ports at the other end of the unit connect to the target hardware. These ports are shown in Figure 1-2 on page 1-9.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

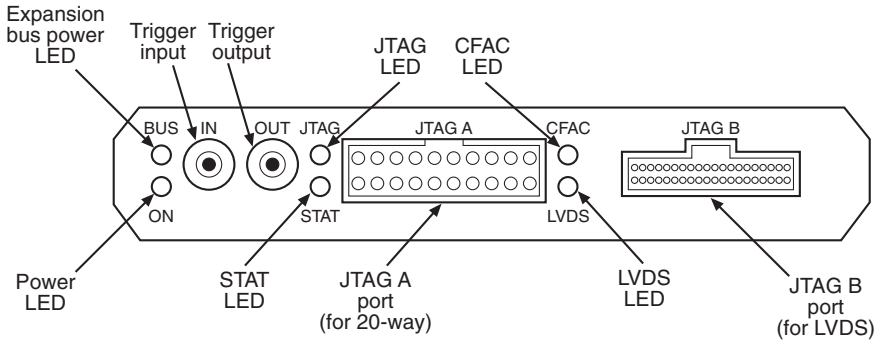


Figure 1-2 Ports for connecting to the target hardware

Cables are supplied to connect the run control unit to the host computer, and to the target hardware.

Note

RealView ICE v3.0 does not support the Trigger input and Trigger output signals.

If the RealView ICE unit detects an internal hardware or software failure from which it cannot recover, the four LEDs JTAG, STAT, CFAC and LVDS (shown in Figure 1-2) flash continuously. You must reboot RealView ICE before you can continue using it. To do this, press the RST button.

During installation of an update or a patch, the CFAC LED will light up, which denotes that Compact Flash Activity (CFAC) is taking place. While this is happening, you must not disconnect power from the run control unit, and should wait until this LED has extinguished. For further information on installing updates and patches, see *Procedure for installing an update or patch* on page 7-10. The CFAC LED also lights up when a debugger connects, and during DHCP lease renewal.

The RealView ICE run control unit contains an internal cooling fan that operates to control the internal temperature when necessary. The ventilation panels on the top and bottom of the RealView ICE run control unit and RealView Trace data capture unit must not be obscured.

1.4.2 The RealView ICE firmware

The RealView ICE firmware is located in the RealView ICE run control unit. It receives commands from the RealView ICE host software and translates them into JTAG accesses. The RealView ICE firmware contains specific sections of code for each ARM processor. These are called templates.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

You can update the RealView ICE firmware using the RealView ICE Update application in the following ways:

- for firmware fixes, you can obtain firmware patches from the ARM website
- for firmware updates that add new functionality, such as additional templates, you must obtain a new CD that also contains updates to the host software.

See Chapter 7 *Managing the RealView ICE Software* for information on using the RealView ICE Update application.

1.4.3 The RealView ICE host software

The RealView ICE host software fits between RealView Debugger and the RealView ICE hardware that controls the JTAG devices. It translates debugger commands, such as start, stop, and download, into JTAG control sequences for a particular processor. The RealView ICE software provides support for debugging on a wide range of ARM cores. To see a list of supported cores, open the RealView ICE Update application and expand the **JTAG Templates** and **ARM** trees. A list of templates for all supported cores is displayed. See Chapter 7 *Managing the RealView ICE Software* for more information on using the RealView ICE Update application.

The RealView ICE software can address each JTAG device individually, without affecting other devices on the board. It uses this ability to create virtual connections for each of the JTAG devices on the board. RealView Debugger can attach to one of these virtual connections, and perform debugging operations with no knowledge of the other devices on the board.

The RealView ICE software enables multiple concurrent connections. If you have licensed the multiprocessor extension to RealView Debugger, you can debug multiprocessor systems, as described in the *RealView Debugger User Guide*. The software can also perform a synchronized start or stop of processors, for debugging multiprocessor systems where the processors interact with each other.

The RealView ICE software also supports connections across a network, so that you can run the debugging software on several different computers.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.5 Introduction to GDB debugging with RealView ICE

RealView ICE enables you to debug applications running on ARM architecture-based cores using *GNU Debugger* (GDB). In addition to the basic debugging operations, such as setting breakpoints and stepping, RealView ICE provides extended debugging features—for example, you can connect to multiple cores.

RealView ICE enables you to debug applications using the following debugging modes:

- Halt-mode debugging, where the target stops while you examine it.
- Monitor-mode debugging, where the target continually runs and you monitor the target using DCC communications.

To connect to a target from GDB, you must configure RealView ICE to recognize your target devices. You do this by initiating an application to configure the scan chain, which in turn connects to RVI and connects to a core in a single operation. The procedure for configuring RealView ICE is described in Chapter 8 *Configuring RealView ICE for GDB*.

In addition to configuring RealView ICE, there are other requirements that you must consider when connecting to your target application:

- Your host workstation might require specific services to be running depending on the requirements of your target application, for example, *Dynamic Host Configuration Protocol* (DHCP).
- Your application might be running as a standalone application without an operating system, or as part of an operating system.

The methods you can use to connect to a target application, and the requirements for each method, are described in Chapter 8 *Configuring RealView ICE for GDB*.

1.5.1 GDB availability and compatibility

To find the latest information on GNU Debugger (GDB) compatibility with RealView ICE v3.0, refer to the ARM RealView ICE v3.0 Release Notes. Also see *The GNU toolchain for ARM architectures* on page 1-12.

For more information on compatibility with target hardware, see the documentation supplied with your hardware.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

1.5.2 Supported debug host platforms

RealView ICE supports remote GDB sessions running on the following operating systems:

- Windows 2000 (service pack 4 or later), or Windows XP Professional (service pack 2, or later)
- Red Hat Enterprise Linux 3 and Red Hat Enterprise Linux 4.

1.5.3 Recommended applications for debugging with GDB

The recommended applications for debugging with GDB are described in the following sections:

- *The GNU toolchain for ARM architectures*
- *Cygwin on Windows* on page 1-13
- *Using an IDE with a GNU toolchain for ARM architectures* on page 1-13.

Note

ARM Limited does not provide support for these applications. If you require support for these applications, see the related web site.

All examples in this chapter assume that you are using GDB from a Unix shell. Also, this chapter assumes that you are familiar with using:

- a GNU toolchain for ARM architectures (see *The GNU toolchain for ARM architectures*)
- Cygwin (see *Cygwin on Windows* on page 1-13)

It also assumes that you already have these applications installed. See the related web sites for details on how to obtain, install, and use them.

The GNU toolchain for ARM architectures

You must use a GNU toolchain built for ARM architecture support. You can obtain prebuilt versions of the toolchain from either:

- CodeSourcery GNU Toolchain for ARM processors, available from:

  `http://www.codesourcery.com`

  The CodeSourcery GNU Toolchain for ARM processors commands are prefixed with `arm-none-eabi-`, for example `arm-none-eabi-gcc`.

  This toolchain is compliant with the Application Binary Interface (ABI) for the ARM Architecture (base standard) [BSABI].

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- GNU ARM toolchain, available from:

  `http://www.gnuarm.com`

  The GNU ARM toolchain commands are prefixed with `arm-elf-` for Windows and `arm-linux-` for Red Hat Linux. For example, `arm-elf-gcc` and `arm-linux-gcc`.

  This toolchain is not compliant with the ABI for the ARM Architecture.

Cygwin on Windows

On Windows you can have Cygwin, obtainable from `http://www.cygwin.com`. Instructions for setting up Cygwin for use with RealView ICE are given in the document *Setting Up Cygwin on Windows*, in the file `CygwinSetup.pdf` that accompanies this document.

Using an IDE with a GNU toolchain for ARM architectures

Although GDB is a command-line debugger, there are *Integrated Development Environments* (IDEs) that use GDB as their backend. For further information on using IDEs in this way, refer to the Application Note that accompanies this release.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 2
Getting Started

This chapter describes the system requirements for RealView® ICE, and how to connect the RealView ICE hardware to your host computer and target system. It also describes how to use some common parts of the RealView ICE software. It contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

2.1 System requirements

This section describes the hardware and software requirements of RealView ICE:

- *Host software requirements*
- *Host hardware requirements* on page 2-3
- *Target hardware requirements* on page 2-3.

See Chapter 6 *Using RealView Trace* for information on the requirements for RealView Trace.

2.1.1 Host software requirements

The RealView ICE software for the host computer runs under the following operating systems:

- Windows 2000 (service pack 4 or later), or Windows XP Professional (service pack 2 or later)
- Red Hat Enterprise Linux 3 and Red Hat Enterprise Linux 4.

Note

RealView Trace is only available for Windows platforms.

RealView ICE v3.0 is compatible with RVDS v3.0.

Automatic dialup

Automatic dialup might be triggered when you use RealView ICE, because RealView ICE uses network facilities. You might want to prevent unnecessary dialups by disabling automatic dialup in the operating system for your host computer.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

2.1.2 Host hardware requirements

This section defines the minimum recommended hardware requirements for installing and running the RealView ICE software on a host computer.

Disk space

If you carry out a full installation of the software, up to 100MB of hard disk space is required.

Using the RealView ICE software on Windows

To use the RealView ICE software on Windows, you require the following:

- Pentium IBM-compatible machine
- CD-ROM drive (this can be a networked CD-ROM drive)
- an unused USB port, if direct connection to the run control unit is required
- a TCP/IP connection, if remote connection to the run control unit is required.

Using the RealView ICE software on Red Hat Linux

To use the RealView ICE software on Red Hat Linux, you require the following:

- Pentium IBM-compatible machine
- CD-ROM drive (this can be a networked CD-ROM drive)
- a TCP/IP connection.

Note

RealView Trace is only available for Windows platforms.

2.1.3 Target hardware requirements

RealView ICE has the following target hardware requirements:

- A device interface conforming to the IEEE Std. 1149.1-2001 (JTAG) specification.
- Electronic signals available to the interface, and within the limits of current and voltage specified in Chapter 9 *System Design Guidelines*.
- One of the following IDC box headers on the target hardware:
  - a 20-way header that conforms to the current ARM JTAG connection standard (as described in Appendix A *JTAG Interface Connections*)

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

  - a 14-way header that conforms to the previous ARM JTAG connection standard (as used by the ARM EmbeddedICE® run control unit).

  To use any other connectors, you must construct an appropriate adaptor or cable yourself.

- A maximum cable length between the target hardware and the RealView ICE run control unit of:
  - 30cm if you are using a 20-way ribbon cable
  - 3m if you are using a 40-way ribbon cable with the supplied LVDS probe.

  Otherwise, one or more of the modifications described in Chapter 9 *System Design Guidelines* must be used.

- One or more ARM architecture CPUs that has supporting debug logic linked into a JTAG scan chain. This includes most ARM7™, ARM9™, ARM10™, and ARM11™ cores. It does not include the StrongARM® and XScale processors.

You can use the RealView Update application to find out which processors are supported by the version of RealView ICE that you are using. To do this, start the RealView ICE Update application (see Chapter 7 *Managing the RealView ICE Software*) and select **RVI → Version Info...**.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

2.2 Connecting the RealView ICE hardware

This section explains how to set up the hardware for RealView ICE:

- *What you require*
- *Connection instructions* on page 2-6
- *Using nonstandard connectors* on page 2-9
- *Hot plugging and unplugging the JTAG cable* on page 2-9.

See Chapter 6 *Using RealView Trace* for information on setting up the RealView Trace hardware.

2.2.1 What you require

To set up the hardware you require the following items from the RealView ICE product kit:

- the RealView ICE run control unit
- the power adaptor for the run control unit
- the mains cable for the power adaptor that is appropriate for your region
- one of the following cables, to connect the run control unit to the PC or the network:
  - — the USB cable, to connect the run control unit directly to the PC using the USB port
  - — the RJ-45 Ethernet cable, to connect the run control unit to the network
  - — the Ethernet cross-over cable, to connect the run control unit directly to the PC using the Ethernet port.
- one of the following cables, to connect the run control unit to the target hardware:
  - — the *JTAG cable* (a short 20-way ribbon cable)
  - — the *LVDS cable and probe* (a long 40-way ribbon cable, and a small PCB with a 40-way and a 20-way IDC connector mounted on it)

You must also provide the following items:

- a host computer that conforms to the requirements given in *Host software requirements* on page 2-2, and in *Host hardware requirements* on page 2-3
- some target hardware containing a JTAG-capable device supported by RealView ICE (see *Target hardware requirements* on page 2-3).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Figure 2-1 shows connections using both the USB and Ethernet cables, and the JTAG 20-way ribbon cable.

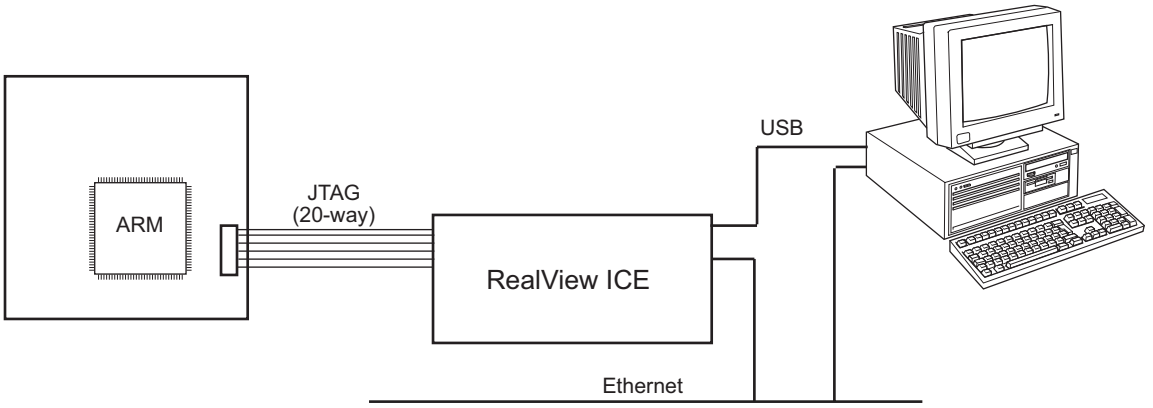


Figure 2-1 Connecting the RealView ICE hardware

2.2.2 Connection instructions

To connect the RealView ICE run control unit to your host computer and to the target hardware:

1. Ensure the RealView ICE software is installed on the host computer. See the *RealView ICE Installation Guide* for information on how to do this.
2. Connect the host computer to the RealView ICE run control unit, using either the USB port or a TCP/IP network connection, as required (see Figure 2-1):
   - If you are connecting using the USB port, connect one end of the supplied USB cable to a USB port on the host computer, and the other end of the cable to the USB port on the run control unit.

     Note

     The USB drivers are installed with the RealView ICE host software.
   - If you are connecting across an Ethernet network, connect the Ethernet port of the run control unit to a socket for the Ethernet network using the supplied RJ-45 Ethernet cable.
   - If you are using the cross-over cable, connect one end of the cross-over cable to the Ethernet port of the host computer, and the other end to the Ethernet port of the run control unit.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3. Connect the RealView ICE run control unit to the target hardware, using the appropriate cable:
   - If you want to use the highest JTAG clock speeds, or if you cannot position the run control unit close to the target hardware, use the LVDS cable and probe:
     — plug the supplied LVDS probe into the 20-way JTAG header on the target hardware

     Note

     If there is insufficient clearance to plug the probe directly into the header, you can use the supplied 20-way JTAG cable to connect the probe to the header.

     — connect one end of the supplied LVDS cable to the 40-way connector on the probe
     — connect the other end of the LVDS cable to the 40-way *JTAG B* socket on the RealView ICE run control unit.

     Note

     Cable selection is performed when the RealView ICE run control unit boots. If you change the cable, you must reboot the unit.

   - Otherwise, use the JTAG cable:
     — connect one end of the supplied JTAG cable to the 20-way JTAG header on the target hardware
     — connect the other end of the cable to the 20-way *JTAG A* socket on the RealView ICE run control unit.

   The IDC connectors used for these cables are keyed using a small protrusion that must be matched up with a slot in the header or socket.

   Caution

   If the target hardware does not have a 20-way IDC connector that conforms to the current ARM JTAG connection standard (as described in Appendix A *JTAG Interface Connections*), see *Using nonstandard connectors* on page 2-9.

4. If you are using RealView Trace, you have to connect the RealView Trace unit to the RealView ICE unit and to the target board. See *Connecting the RealView Trace hardware* on page 6-7 for information on how to do this.
5. Power up the target hardware.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6. Connect the external power supply to the RealView ICE run control unit, and to the mains electricity.

7. Switch on the power supply. The power LED and the expansion bus power LED both switch on.

8. The RealView ICE run control unit firmware is based on an embedded Linux kernel. Therefore, the unit takes a short time to boot up and establish either a network or USB connection. When the unit is booting:
   - The CFAC LED lights up, and the STAT LED flashes.

     As the RealView ICE run control unit boots, the STAT LED flash rate increases.
   - The unit detects which JTAG socket has a cable attached:
     - If RealView ICE detects that the LVDS cable and probe are connected to JTAG B socket, it uses them. It also switches on the LVDS LED.
     - If RealView ICE detects that a JTAG cable is connected to the JTAG A socket, the LVDS LED remains unlit.
   - When the STAT LED is permanently On, the RealView ICE run control unit has finished booting, and is ready to use.

9. If your RealView ICE unit is connected to a network, you must now run the Config IP application to configure the network settings, as described in Chapter 3 *Configuring RealView ICE Networking*.

   Note

   You have only to do the network configuration once.

   If the RealView ICE unit is powered up with only a USB connection, it uses an IP address of 127.0.0.0. However, if a network cable is also attached, the IP address associated with the USB connection is the IP address that you have assigned to the RealView ICE unit, or that it obtains from a DHCP server.

Warning

Do not obstruct the ventilation grills on the top and bottom of the RealView ICE unit, because doing so causes the unit to overheat.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

2.2.3 Using nonstandard connectors

RealView ICE is supplied with cables that each terminate in a 20-way IDC connector, wired to the current ARM JTAG connection standard (see Appendix A *JTAG Interface Connections*). Box headers suitable for this connector are fitted on all current ARM target hardware, and on several third-party targets.

Some target hardware is fitted instead with a 14-way IDC box header:

- Older ARM target hardware uses the previous ARM JTAG connection standard (as used by EmbeddedICE). This is signal-compatible with the current ARM standard. Use the supplied adaptor card to connect to these targets.
- Some other targets instead use the *Texas Instruments* (TI) JTAG connection standard. This has a different signal assignment to the ARM standards. An adaptor to enable RealView ICE to connect to these targets is available from ARM on request. Quote part number HBI 0068B.

Caution

If you use the wrong 20-way to 14-way adaptor, you might damage the target hardware.

If you are not certain of the connection standard that your target hardware uses, you *must* check the reference manual for the target *before* you connect it to the RealView ICE run control unit. This is especially important if you are using a target that has a 14-way IDC box header, or that is not manufactured by ARM Limited.

If the target that you are using does not use an ARM style connector, or if you are designing target hardware, contact ARM Limited for more information.

2.2.4 Hot plugging and unplugging the JTAG cable

You can plug and unplug the JTAG cable without affecting the target. This is because the RealView ICE run control unit includes power conditioning and switching circuitry.

You might want to do this if you have a target that is operating without a RealView ICE run control unit connected and you want to examine the target to find out why it is behaving in a particular way. To do this, you must power up the RealView ICE run control unit and configure the connection without disturbing the state of the target. This requires that the RealView ICE run control unit is powered before it is connected to the target.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

When unplugging the JTAG connector, you must be aware of the following:

- If you are using an RTCK system, make sure that no communication is taking place between the system and the RealView ICE run control unit. Otherwise, if the RealView ICE unit is waiting for a return clock, it might lock up. In this case, you must power down the RealView ICE unit, and power it back up again.
- If you are not using an RTCK system, the RealView ICE software can handle this situation. However, you must arrange to do a TAP reset using the debugger when you next plug the RealView ICE unit into a target. See *Advanced configuration* on page 4-22 for details.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

2.3 Using RealView ICE and RealView Trace

When you have connected RealView ICE to your host computer (see *Connecting the RealView ICE hardware* on page 2-5), you are ready to begin using RealView ICE (and RealView Trace if present) with RealView Debugger. See the *RealView Debugger* documentation suite for information on using RealView Debugger.

When you install the RealView ICE software, it adds capabilities to RealView Debugger to enable you to configure a RealView ICE connection using the RVConfig dialog box. This is described in full in Chapter 4 *Configuring a RealView ICE Connection*.

If you have to update the RealView ICE firmware at a later date, to extend the capabilities of the RealView ICE unit for example, you must use the RVI Update utility. This is described in Chapter 7 *Managing the RealView ICE Software*.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 3
Configuring RealView ICE Networking

This chapter describes how to configure the network settings for your RealView® ICE run control unit. If you have connected your run control unit to an Ethernet network or directly to the host computer using an Ethernet cross-over cable, you must configure the network settings before you can use the unit for debugging. You have only to configure the network settings once.

Note

If you have connected your run control unit directly to the host computer using a USB cable, and you do not intend to connect it to a network, you do not have to configure the network settings. See *Connection instructions* on page 2-6 for information on connecting up the components.

To configure your RealView ICE run control unit to use the correct network settings for your network:

1. *Determining the correct network settings* on page 3-2
2. *Starting the RealView ICE Config IP application* on page 3-3
3. *Configuring the network settings* on page 3-4
4. *Restarting your RealView ICE run control unit* on page 3-12
5. *Exiting the RealView ICE Config IP application* on page 3-3.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3.1 Determining the correct network settings

Before you can configure the network settings, you must first determine the correct network settings for your RealView ICE run control unit. To do this, you must consult with the system administrator for your network.

The information that you require depends on whether or not your network uses DHCP:

- if your network does not use DHCP, see *Not using DHCP*
- if your network uses DHCP, see *Using DHCP*.

3.1.1 Not using DHCP

If your network does not use DHCP, you must know:

- the hostname that you want to use for your run control unit (if any)
- the IP address that you want to use for your run control unit
- the default gateway for your network (if it has one)
- the subnet mask for your network.

3.1.2 Using DHCP

If your network uses DHCP, you must know the hostname that you want to use for your run control unit (if any).

Note

You do not have to know the IP address for your run control unit, or the default gateway and subnet mask for your network, because these settings are fetched from a DHCP server on your network.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3.2 Starting and exiting the RealView ICE Config IP application

This section describes how to start and exit the RealView ICE Config IP application:

- *Starting the RealView ICE Config IP application*
- *Exiting the RealView ICE Config IP application.*

3.2.1 Starting the RealView ICE Config IP application

To start the RealView ICE Config IP application:

- On Windows, select **Start → Programs → ARM → RealView ICE v3.0 → RealView ICE Config IP**.

  Note

  If you are using the default Windows XP settings, select **All Programs**.

- On Red Hat Linux, choose the appropriate shortcut. This depends on the version of Red Hat Linux and the desktop environment that you are using. If no desktop shortcut is available, enter the command `rviconfigip` at the command line.

The RealView ICE Config IP application opens, as shown in Figure 3-1.

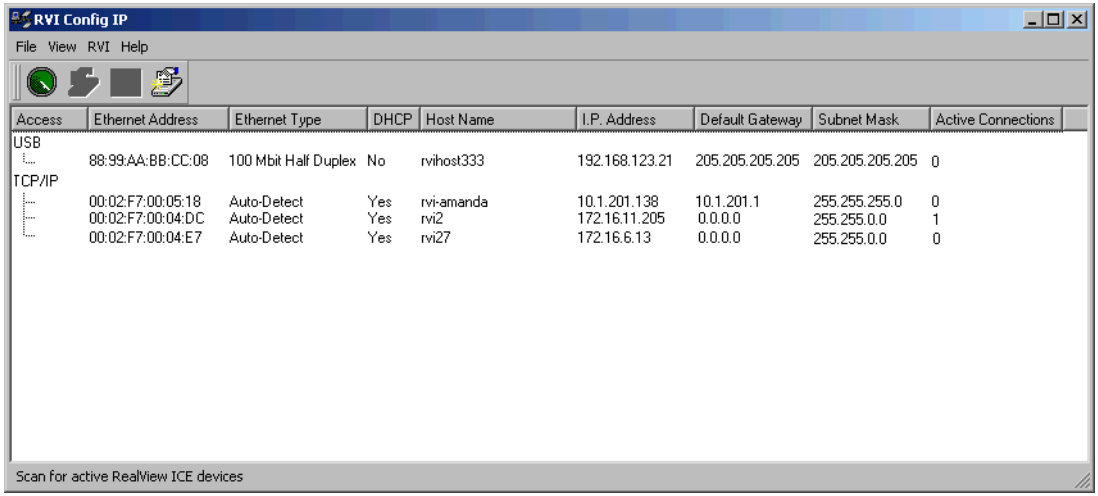


Figure 3-1 The RealView ICE Config IP application

3.2.2 Exiting the RealView ICE Config IP application

To exit the RealView ICE Config IP application, select **File → Exit**.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3.3 Configuring the network settings

The configuration process depends on the way in which the RealView ICE unit is connected to the host computer, and whether or not you know its Ethernet address:

- If your RealView ICE unit is connected to your local network, but you do not know its Ethernet address, you can select it from the list of available units. See *Configuring by scanning all run control units*
- If you know the Ethernet address of your RealView ICE unit, you can go straight to the Configuration dialog box and enter the address. See *Configuring using an Ethernet address* on page 3-9
- If you have connected your run control unit directly to the host computer using a cross-over cable, you must assign static IP addresses to the host computer and the run control unit. See *Configuring for connection with an Ethernet cross-over cable* on page 3-10.

Note

If you have connected your run control unit directly to the host computer using a USB cable, you do not have to configure the network settings.

Note

The toolbar buttons mentioned in the following sections also have equivalent options on the **RVI** menu.

3.3.1 Configuring by scanning all run control units

This section describes how to configure the network settings for your RealView ICE run control unit by scanning for all available run control units. You can also configure your run control unit by specifying its Ethernet address, as described in *Configuring using an Ethernet address* on page 3-9.

The configuration process consists of the following steps:

1. *Scanning for your RealView ICE run control unit*
2. *Identifying and selecting your RealView ICE run control unit* on page 3-7
3. *Configuring your RealView ICE run control unit* on page 3-7.

Scanning for your RealView ICE run control unit

Click **Scan** to scan for run control units that are connected to your local network. The Scan button becomes animated to indicate that a scan is in process. When RealView ICE finds a unit, it adds it to the list of available units, as shown in Figure 3-1 on page 3-3.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

If you want to stop scanning, click **Scan**. You can click **Scan** again at any time to force a rescan of available RealView ICE units and update the list.

Note

If you are using DHCP, RealView ICE scans for a DHCP server during the first minute after rebooting. During this period, the Scan tool cannot locate the unit. If the unit is unable to obtain its IP address from a DHCP server, it appears in the RealView ICE Config IP dialog box with the address 127.0.0.2.

If you want the RealView ICE run control unit to try again to obtain its IP settings from a DHCP server, you must first reboot the unit.

Note

The number of active connections shown in the RVI Config IP dialog box might be more than the number of active users, because each user might have multiple active connections. For example, the RealView ICE connection and the RealView Trace connection are both displayed as active connections, or a device might be listed under both USB and TCP/IP if it is accessible by both methods.

Troubleshooting

This section gives details of problems you might encounter when attempting to connect to a RealView ICE unit, and what you can do to solve them:

Multiple programs attempting to scan

Only one program can scan the TCP/IP network or USB ports for available RealView ICE units. If another program is scanning, for example the RVConfig dialog box in RealView Debugger (see *Using the RVConfig dialog* on page 4-3), the RVI Config IP application displays the error message shown in Figure 3-2.

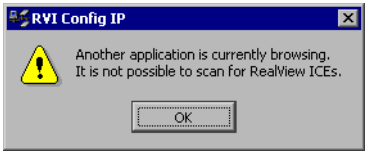


Figure 3-2 Error message when another program is browsing

You must stop one of the programs from scanning. To do this, click **Scan** or select **RVI → Stop Scan** from the menu in the application that you want to stop scanning.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

USB server not accessible

If the USB server is not accessible, the error message shown in Figure 3-3 appears:

Figure 3-3 Error message when no USB devices present

This indicates a problem with your RealView ICE installation. Click **OK**. If you do not want to connect to any devices over a USB connection, you can continue using RealView ICE over only TCP/IP connections. If you want to connect to a device using USB, you must reinstall RealView ICE. If the error persists, there might be a problem with your operating system.

Connection times out

The default timeout for establishing a TCP/IP connection is 5 seconds. If you repeatedly get timeouts when attempting to connect to a RealView ICE run control unit, you can change this setting. To do this:

1. If the environment variable `RVI_COMMS_CONNECT_TIMEOUT` does not already exist, then create it.
2. Set the value of this variable to the timeout that you want, in seconds. This must be an integer in the range 0-120.

For details of how to create and set an environment variable, see the documentation for the operating system that is supplied with your host computer.

Other active connections

If you connect to a RealView ICE run control unit that has other active connections, the RVI Config IP application displays the error message shown in Figure 3-4.

Figure 3-4 Error when other connections are active

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

If you continue, the changes that you make might interfere with the correct operation of these applications. Do one of the following:

- ensure that the other applications are disconnected, and then click **Yes** to continue using the RealView ICE Config IP application
- click **No** to stop using the RealView ICE Config IP application, and try again later.

Identifying and selecting your RealView ICE run control unit

To identify and select your RealView ICE run control unit from the list of units found, do one of the following:

- Determine the Ethernet address of your run control unit by reading the label on the side of the unit. Find the entry in the list that has the same Ethernet address, and select it.

- Select an entry in the list and click the **Identify** tool:
  - If the four LEDs JTAG, STAT, CFAC and LVDS on your interface (shown in Figure 3-5) flash for 5 seconds, you have selected its entry.

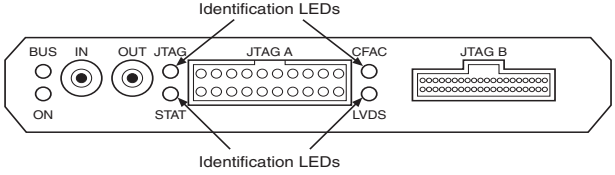


Figure 3-5 The identification LEDs

  - Otherwise, select another entry and try again.

Configuring your RealView ICE run control unit

When you have selected your RealView ICE run control unit, you must configure it to use the network settings that you previously determined (see *Determining the correct network settings* on page 3-2):

1. Click the **Configure** tool. The Configure RealView ICE device dialog box appears. See Figure 3-6 on page 3-8.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

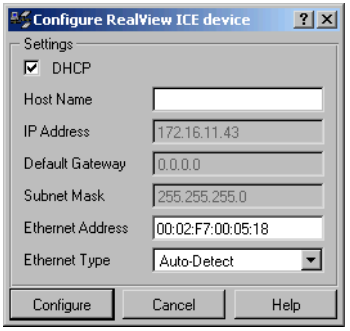


Figure 3-6 The Configure RealView ICE device dialog box

2. If you are using DHCP, select **DHCP**. Otherwise, deselect **DHCP**.
3. Enter the hostname in the Host Name field. This must contain only the alphanumeric characters (`A-Z`, `a-z`, and `0-9`) and the `-` character, and must be no more than 255 characters long.
4. If you are not using DHCP, enter the required details in the following fields:
   - IP Address
   - Default Gateway
   - Subnet Mask.

   Note

   If you are using DHCP, you do not have to type these settings, because they are allocated from a DHCP server on your network.
5. Set the required Ethernet Type:
   - if you know the type of network that you are using, select that type
   - otherwise, select **Auto-Detect**.
6. Click **Configure**.

   The RealView ICE run control unit restarts. While it is restarting, it is not present in the list of units. When it has restarted, it re-appears in the list of units, with its new network settings.

   Note

   If the RealView ICE run control unit is using DHCP, the list of units might display its **IP Address** as `127.0.0.2`. This is a dummy address, which the run control unit uses when it fails to obtain an IP address from the DHCP server.

   The list of units shows the correct address if the DHCP server has assigned it.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3.3.2 Configuring using an Ethernet address

If you have a RealView ICE run control unit that does not have a valid IP address or is on a different subnet, you must manually enter the Ethernet address during configuration.

To configure your RealView ICE run control unit by entering an Ethernet address:

1. Open the required configuration dialog box, which depends on whether or not your run control unit has a USB connection:

- If the device has a USB connection, select the device in the USB list and click the **Configure** tool. The Configure RealView ICE device dialog box appears, as shown in Figure 3-6 on page 3-8.

  Alternatively, double-click on the device in the USB list.

- If the device does not have a USB connection, click the **Config New** tool. The Configure new RealView ICE device dialog box appears, as shown in Figure 3-7.

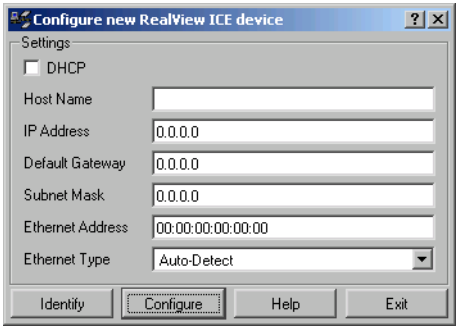


Figure 3-7 The Configure new RealView ICE device dialog box

2. Determine the Ethernet address of your run control unit by reading the label on the side of the unit, and enter it into the Ethernet Address field.

If you are using the Configure new RealView ICE device dialog box and you want to be certain that you are configuring the correct run control unit, click the **Identify** tool. Verify that the identification LEDs flash (see Figure 3-5 on page 3-7).

3. If you are using DHCP, select **DHCP**. Otherwise, deselect **DHCP**.

4. Enter the hostname in the Host Name field. This must contain only the alphanumeric characters (`A-Z`, `a-z`, and `0-9`) and the `-` character, and must be no more than 255 characters long.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5. If you are not using DHCP, enter the required details in the following fields:
   - IP Address
   - Default Gateway
   - Subnet Mask.

   Note

   If you are using DHCP, you do not have to type these settings, because they are fetched from a DHCP server on your network.

6. Select the required Ethernet Type:
   - if you know the type of network that you are using, select that type
   - otherwise, select **Auto-Detect**.

7. Click **Configure**.

   The RealView ICE run control unit restarts. While it is restarting, it is not present in the list of units. When it has restarted, it re-appears in the list of units, with its new network settings.

   Note

   If the RealView ICE run control unit is using DHCP, the list of units might display its **IP Address** as `127.0.0.2`. This is a dummy address, which the run control unit uses when it fails to obtain an IP address from the DHCP server.

   The list of units shows the correct address if the DHCP server has assigned it.

3.3.3 Configuring for connection with an Ethernet cross-over cable

If you have connected your run control unit directly to the host computer using a cross-over cable, you must assign static IP addresses to the host computer and the run control unit:

1. Assign a static IP address to your host computer. If your host computer was obtaining an IP address from a DHCP server you can use that address, but you must now assign it statically.

2. Select your RealView ICE unit, and click the **Configure** tool. Alternatively, double-click on the device in the list. The Configure RealView ICE device dialog box appears, as shown in Figure 3-6 on page 3-8.

3. Deselect **DHCP**.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

4. Enter a hostname in the Host Name field. This must contain only the alphanumeric characters (A-Z, a-z, and 0-9) and the - character, and must be no more than 255 characters long.

5. Enter the required details in the following fields:
   - Default Gateway
   - Subnet Mask.

   These must be the same as those for the host computer.

6. Enter an IP address of the run control unit in the IP Address field. Ensure that the host computer and the run control unit are in the same subnet. For example, if the subnet mask is set to 255.255.255.0, and the IP address of the host computer is 192.168.0.*x*, you must set the IP address of the run control unit to 192.168.0.*y*.

7. Select the required Ethernet Type. **Auto-Detect** is the recommended setting.

8. Click **Configure**.

   The RealView ICE run control unit restarts. While it is restarting, it is not present in the list of units. When it has restarted, it re-appears in the list of units, with its new network settings.

Note

Software such as firewall software might interfere with the communications between the computer and RealView ICE. You might have to temporarily disable any firewall when using RealView ICE and re-enable it after you finish.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3.4 Restarting your RealView ICE run control unit

The RealView ICE Config IP application normally restarts the networking software on the run control unit whenever you change its settings. If necessary, select **RVI** → **Restart** to force the networking software to restart.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 4
Configuring a RealView ICE Connection

When you install the RealView® ICE software, it adds various features to RealView Debugger. This chapter describes how to use these additional features to configure a RealView ICE connection, and how to connect RealView Debugger to a target using RealView ICE. You may also run RVConfig as a standalone feature, and this procedure is described.

This chapter contains the following sections:

Note

Certain aspects of this chapter assume that you are familiar with using RealView Debugger to connect to a target, and to configure a connection. For details, see the RealView Debugger documentation suite (see *ARM publications* on page xix), especially the *RealView Debugger Target Configuration Guide*

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

4.1 Changes to RealView Debugger

After the RealView ICE software installation, the following files are included in the RealView Debugger `\etc` directory:

- a default RealView ICE configuration file, `rvi.rvc`
- the RealView ICE board file, `RVI.brd`.

The first time you start RealView Debugger after the installation, it updates the `rvdebug.brd` file in your RealView Debugger home directory with the details from the `\etc\RVI.brd` file. After the update, the following capabilities are added to RealView Debugger:

- An RVConfig dialog box that you use to configure each RealView ICE run control unit. See *Using the RVConfig dialog* on page 4-3.
- New tabs in the Register pane of the Code window, in addition to the **Core** tab that is present for all targets. The additional tabs that appear depend on the target that you are debugging, but might include:
  - a **CP15** tab that displays and sets the values of registers in coprocessor 15 (the System Control coprocessor)
  - a **Cache Operations** tab that you can use to perform operations on the cache for the target
  - a **TLB Operations** tab that you can use to perform operations on the *translation look-aside buffer* (TLB) for the target
  - a **Debug** tab that controls various internal debugger settings, many of which are specific to RealView ICE.

  The **CP15**, **Cache Operations**, and **TLB Operations** tabs control features of the target hardware. These features are described in the ARM datasheet or technical reference manual for your core (see *ARM publications* on page xix).

  The **Debug** tab is described in *Using the Debug tab of the RealView Debugger Register pane* on page 4-28.

For further information, refer to the *RealView Debugger User Guide*.

For more specific information on target connection using RealView Debugger, see the *RealView Debugger Target Configuration Guide*.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

4.2 Using the RVConfig dialog

Before you can use a RealView ICE run control unit to connect RealView Debugger to a target, you must configure the run control unit using the RVConfig dialog box. This process involves the following steps:

1. *Opening the RVConfig dialog box via RealView Debugger*
2. *Connecting to a RealView ICE unit* on page 4-6
3. *Configuring a scan chain* on page 4-9
4. *Configuring devices* on page 4-18
5. *Advanced configuration* on page 4-22
6. *Saving your changes* on page 4-25
7. *Disconnecting from a RealView ICE unit* on page 4-25.

4.2.1 Opening the RVConfig dialog box via RealView Debugger

To open the RVConfig dialog box:

1. Display the Connection Control window in RealView Debugger. See the chapter that describes connecting to targets in the *RealView Debugger Target Configuration Guide*.
2. Right-click on the relevant `RealView-ICE` entry in the Connection Control window to display the context menu.
3. From this context menu, select the **Configure** option. The RVConfig dialog box appears, as shown in Figure 4-1 on page 4-4. The title of the dialog box includes the full path to the RealView ICE configuration file. The path name might be different to that shown if you have a different version of RealView Debugger installed.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

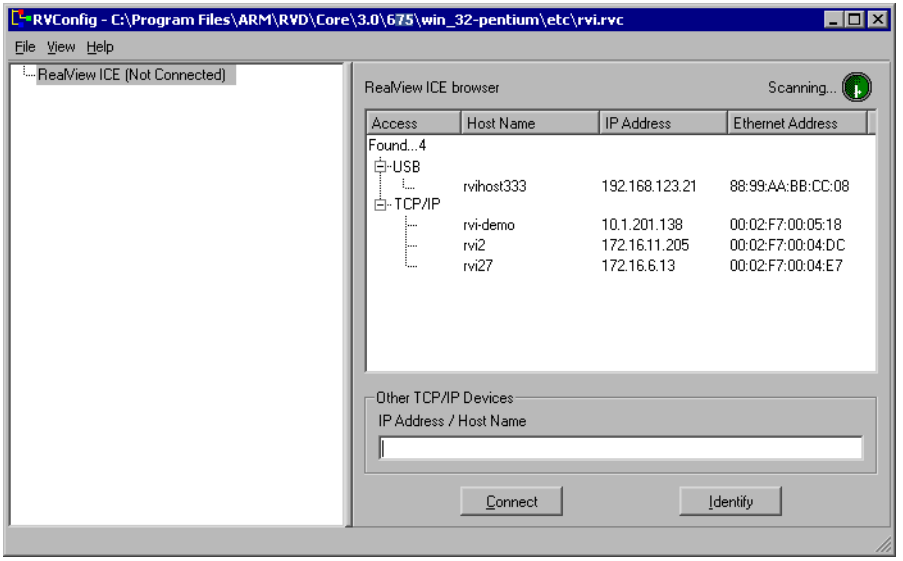


Figure 4-1 RVConfig dialog box

4.2.2 Opening the RVConfig dialog box — standalone method

You may also open the RVConfig dialog box to configure the run control unit, without first having to display the Connection Control window in RealView Debugger as described above.

To start the RVConfig application:

1. On Windows, select **Start → Programs → ARM → RealView ICE v3.0 → RealView ICE Configuration**.

   Note

   If you are using the default Windows XP settings, select **All Programs**.

   On Red Hat Linux, choose the appropriate shortcut. This depends on the version of Red Hat Linux and the desktop environment that you are using. If no desktop shortcut is available, enter the command `rviconfig` at the command line.

   The RVConfig application opens, as shown in Figure 4-2 on page 4-5.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

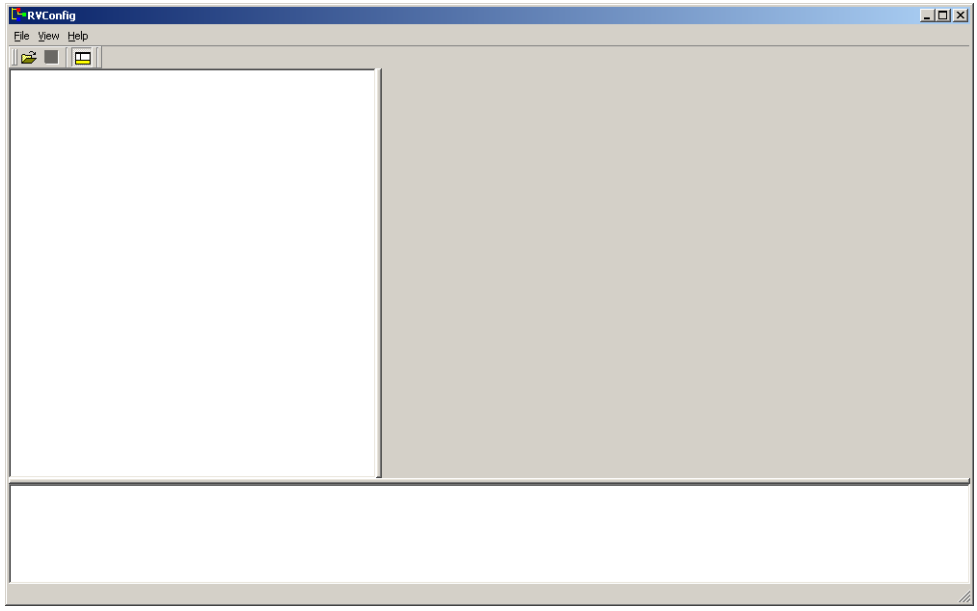


Figure 4-2 RVConfig application

2. Select **File** → **Open**.

   The Choose a file to open dialog appears, as shown in Figure 4-3:

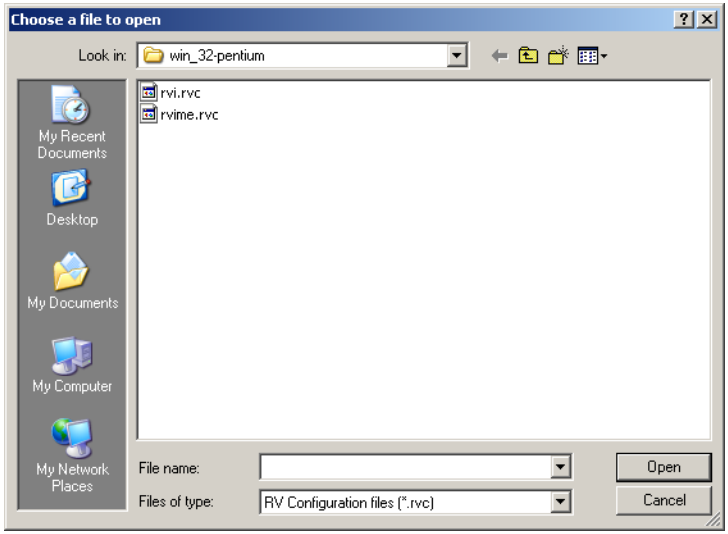


Figure 4-3 Choose a file to open dialog

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3. Browse the list, select the appropriate `.rvc` file, and click Open.

   The RVConfig dialog box appears, as shown in Figure 4-4. The title of the dialog box includes the full path to the RealView ICE configuration file. The path name might be different to that shown.

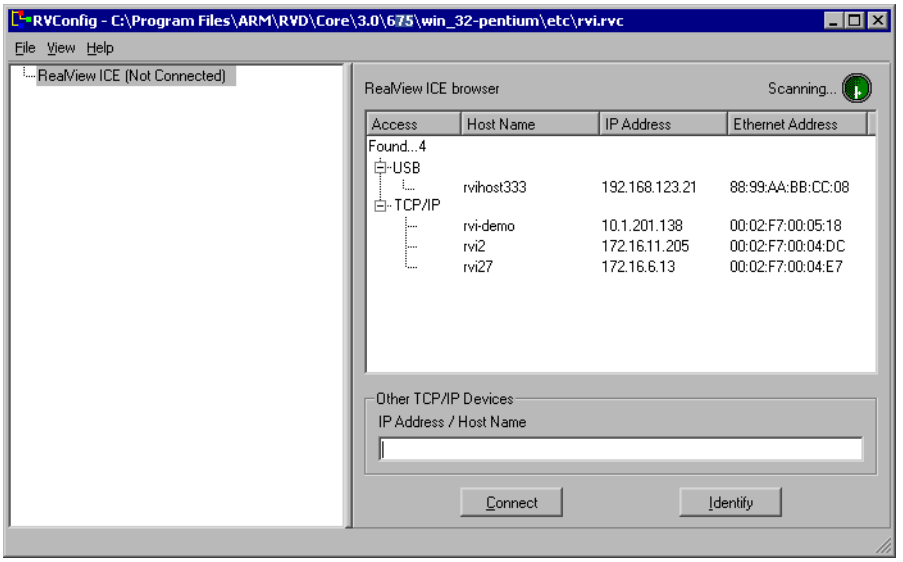


Figure 4-4 RVConfig dialog box

4.2.3 Connecting to a RealView ICE unit

When you start the RVConfig application, it scans for run control units that are connected to your local network. The Scan button becomes animated to indicate that a scan is in process. When RealView ICE finds a unit, it adds it to the list of available units, as shown in Figure 4-5 on page 4-7.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

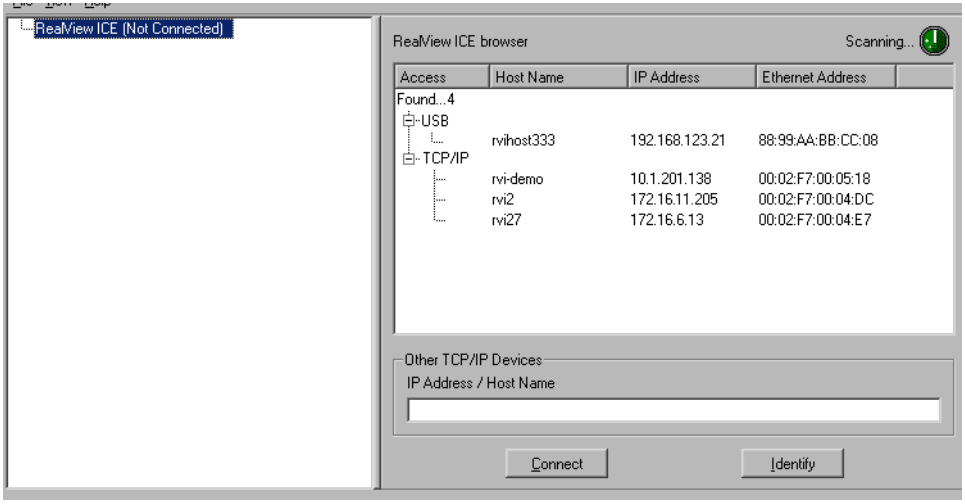


Figure 4-5 The RVConfig dialog box showing available units

If you want to stop scanning, click **Scan**. You can click **Scan** again at any time to force a rescan for available RealView ICE units and update the list.

The devices found are listed in the RealView ICE browser on the right of the dialog box. Select the unit you want to connect to and click **Connect**. Alternatively, do one of the following:

- double-click on the unit you want to connect to
- enter either the IP address or host name of the device you want to connect to in the IP Address/Host Name field and click **Connect**.

If you want to be certain that you are connecting to the correct run control unit, select an entry in the list, and click **Identify**:

- If the LEDs JTAG, STAT, CFAC and LVDS on your interface (see Figure 4-6) flash for 5 seconds, you have selected the correct entry for the unit in the list.

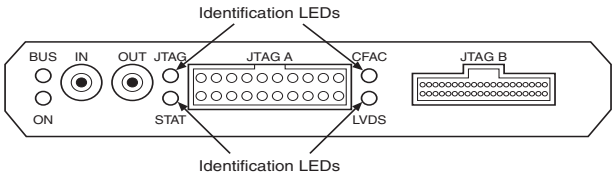


Figure 4-6 The Identification LEDs

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- Otherwise, select another entry and try again.

Note

Devices shown in light gray are those that have responded to browse requests but do not have a valid IP address. You cannot connect to these devices until you have configured the IP address. See Chapter 3 *Configuring RealView ICE Networking* for information on how to do this.

This adds a Devices node to the tree diagram on the left of the RVConfig dialog box, and selects that node. The control pane changes, ready for you to configure the scan chain for the connected RealView ICE unit. See *Configuring a scan chain* on page 4-9.

Troubleshooting

This section gives details of problems you might encounter when attempting to connect to a RealView ICE unit, and what you can do to solve them:

Multiple programs attempting to scan

Only one program can scan the TCP/IP network or USB ports for available RealView ICE units. If another program is scanning, for example the RVI Config IP dialog box (see *Scanning for your RealView ICE run control unit* on page 3-4), the RVConfig application displays the error message shown in Figure 4-7.

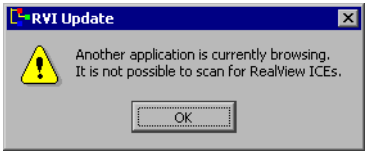


Figure 4-7 Error message when another program is browsing

You must stop one of the programs from scanning. To do this, click the **Scan** tool to terminate the scan in the program that you want to stop scanning.

USB server not accessible

If the USB server is not accessible, the error message shown in Figure 4-8 on page 4-9 appears:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Figure 4-8 Error message when no USB devices present

This indicates a problem with your RealView ICE installation. Click **OK**. If you do not want to connect to any devices over a USB connection, you can continue using RealView ICE over only TCP/IP connections. If you want to connect to a device using USB, you must reinstall RealView ICE.

Timeouts

The default timeout for establishing a TCP/IP connection is 5 seconds. If you repeatedly get timeouts when attempting to connect to a RealView ICE run control unit, you can change this setting. To do this:

1. If the environment variable `RVI_COMMS_CONNECT_TIMEOUT` does not already exist, then create it.
2. Set the value of this variable to the timeout that you want, in seconds. This must be an integer in the range 0-120.

For details of how to create and set an environment variable, see the documentation for the operating system that is supplied with your host computer.

4.2.4 Configuring a scan chain

This section explains how to configure a scan chain for the currently connected RealView ICE unit. This procedure consists of the following steps:

1. *Displaying the scan chain controls*
2. *Autoconfiguring a scan chain* on page 4-10
3. *Adding devices* on page 4-13
4. *Removing devices* on page 4-14
5. *Changing the order of devices* on page 4-15
6. *Changing the properties of a device* on page 4-15
7. *Setting the JTAG clock speed* on page 4-17.

Displaying the scan chain controls

Before you can configure a scan chain, you must ensure that the control pane displays the scan chain controls. To do this, select the **Devices** node in the tree diagram, as shown in Figure 4-9 on page 4-10.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

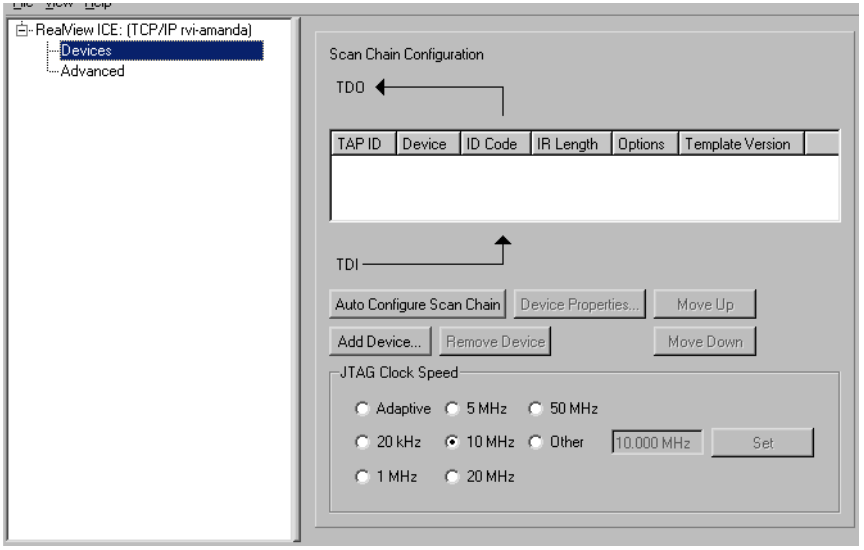


Figure 4-9 Displaying the scan chain controls

Autoconfiguring a scan chain

When autoconfiguring a scan chain, RealView ICE interrogates the scan chain and automatically selects the correct templates for supported ARM® target devices, and adds them to the scan chain in the correct order. This takes place at the current JTAG clock speed (see *Setting the JTAG clock speed* on page 4-17):

- If you are using a fixed JTAG clock speed, but RealView ICE detects one or more devices that require adaptive clocking, it automatically selects adaptive clocking.
- If you are using adaptive clocking, but RealView ICE does not detect any devices that support adaptive clocking, an error message is generated. Select a fixed JTAG clock speed.
- If the JTAG clock speed is too high, some devices on the scan chain might not be detected. If you suspect that this is happening, decrease the JTAG clock speed.

Note

If your target includes a DSP, you cannot autoconfigure the scan chain. Therefore, you must add the devices manually (see *Adding devices* on page 4-13).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Autoconfiguring identifies the target core by reading the JTAG TAPID register. The value of this register is usually set by the engineers that integrate the ARM core into a design. It is not set within the ARM core itself. For more information, see the ARM datasheet or technical reference manual for the core that you are integrating (see *ARM publications* on page xix).

Warning

Reading the JTAG TAPID register might not be sufficient for RealView ICE to uniquely identify the device. In these circumstances, auto-configuring might involve:

- resetting the core or the board
- stopping the core
- accessing registers
- accessing memory.

In extreme cases, these actions might cause physical damage to the system being debugged. If your system cannot tolerate this level of intrusion, you must instead add the device manually.

To auto-configure a scan chain:

1. Ensure that the scan chain controls are displayed, as described in *Displaying the scan chain controls* on page 4-9.
2. Click on **Auto Configure Scan Chain**. Each detected device is added to the Scan Chain Configuration list in the control pane, and is also added to the tree diagram. In many cases, this is all that you have to do to configure the scan chain. You must then configure the devices themselves, as described in *Configuring devices* on page 4-18.

You might see one of the following errors:

- If RealView ICE detects any unpowered devices, it displays the error shown in Figure 4-10.

Figure 4-10 Error shown when unpowered devices are detected

  If you see this error:

  — Check the JTAG connection between the RealView ICE run control unit and the target hardware. See *Connection instructions* on page 2-6.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

— Ensure that power is supplied to all your devices.

- If RealView ICE cannot identify any devices, it displays the error shown in Figure 4-11.

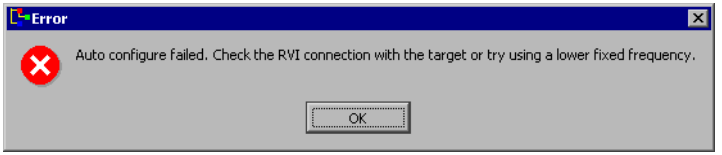


Figure 4-11 Error shown when no devices are detected

If you see this error, try auto-configuring again with a lower JTAG clock speed. See *Setting the JTAG clock speed* on page 4-17.

Note

You might have to power-cycle your target hardware when changing the JTAG clock speed.

- If communication cannot be made with the RealView ICE unit, it displays the error shown in Figure 4-12.

Figure 4-12 Error shown when there is no communication with RealView ICE

If you see this error, then check the network or USB cable to the RealView ICE unit.

You can use the other controls in the RVConfig dialog box to create or modify the scan chain configuration. See:

- *Adding devices* on page 4-13
- *Removing devices* on page 4-14
- *Changing the order of devices* on page 4-15
- *Changing the properties of a device* on page 4-15
- *Setting the JTAG clock speed* on page 4-17.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Adding devices

You can manually add devices to the scan chain, if required. You might want to do this in the following circumstances:

- You have previously autoconfigured the scan chain, and added extra devices to the scan chain. In this case, you might also have to change the order of the devices (see *Changing the order of devices* on page 4-15).
- The autoconfiguration fails. This might occur if your target includes a DSP device.

Note

If your target does not have a DSP, check the connection between your RealView ICE and the target, make sure your target is switched on, and then attempt the autoconfiguration again.

To add a device:

1. Ensure that the scan chain controls are displayed. See *Displaying the scan chain controls* on page 4-9.
2. Click on **Add Device**. The Add Device dialog box appears, as shown in Figure 4-13. Other devices might be available depending on your RealView ICE firmware.

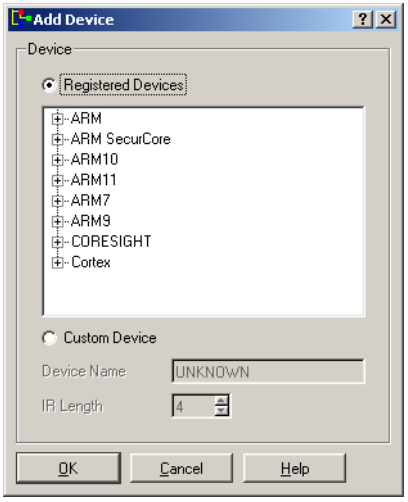


Figure 4-13 The Add Device dialog box

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3. If you have more than one device, check the order of the devices on the target. The device nearest to **TDO** is last on the chain.

   Note

   If you add the devices in the wrong order, you can later change the order (see *Changing the order of devices* on page 4-15).

4. Specify the device to add:
   - If the device appears in the list of **Registered Devices**:
     1. Select **Registered Devices**.
     2. Expand the relevant category. To do this, either double-click on its name, or click on the associated **+** button.
     3. Select the device that you want to add.
   - Otherwise:
     1. Select **Custom Device**.
     2. Enter the name of the device in the Device Name field. This is used as the name of the device node in the tree view, and can have any value.
     3. Enter the JTAG *Instruction Register length* (in bits) in the IR Length field.

        Note

        If you enter an incorrect value for the IR length, then any connections you attempt to make to the device fail.

5. Click on **OK**. The device is added to the scan chain.
6. If you have multiple devices, repeat step 4 and 5 to add each device in the order identified at step 3.

Removing devices

To remove an unwanted device from the scan chain:

1. Ensure that the scan chain controls are displayed. See *Displaying the scan chain controls* on page 4-9.
2. Select the device in the Scan Chain Configuration list.
3. Click **Remove Device**.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Changing the order of devices

The Scan Chain Configuration list shows the devices in ascending order of TAP ID, as indicated by the arrows in the Devices pane of the RVConfig dialog box (shown in Figure 4-9 on page 4-10). The device at the top of the list is last on the chain, nearest to **TDO**.

Note

If you have previously used Multi-ICE®, be aware that the list of devices on the scan chain is the opposite order to that used by Multi-ICE.

To change the position of a device in the scan chain:

1. Ensure that the scan chain controls are displayed (see *Displaying the scan chain controls* on page 4-9).
2. Select the device in the Scan Chain Configuration list.
3. Then do one of the following:
   - click **Move Up** to move the device up the scan chain
   - click **Move Down** to move the device down the scan chain.

Note

Changing the order of devices instructs RealView ICE of the actual order of the devices, but does not move the devices themselves.

Changing the properties of a device

To change the properties of a device:

1. Ensure that the scan chain controls are displayed (see *Displaying the scan chain controls* on page 4-9).
2. Select the device in the Scan Chain Configuration list.
3. Click **Device Properties**. The Device Properties dialog box appears, as shown in Figure 4-14 on page 4-16.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

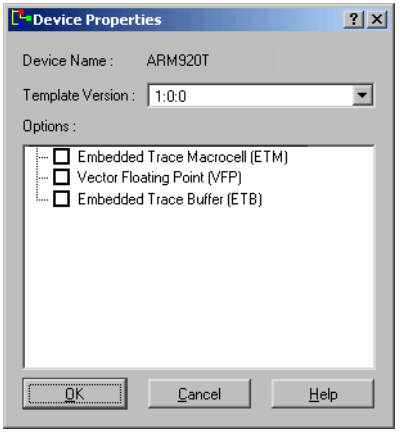


Figure 4-14 The Device Properties dialog box

4. Update the properties as necessary.
   - the Device Name field identifies the device that you are configuring
   - the Template Version sets the version of the device template that you are using. The RealView ICE unit stores templates for each supported device. These templates define how to communicate with the device, and the configuration options for that device.

     RealView ICE can store multiple versions of templates for each device. One version is used by default (typically the most recent), but you can use the other versions if necessary. For example, you might use the latest version of a template to debug a new project, but use an older version to perform a regression test.

     For details of how to change the Default Version of a template, see Chapter 7 *Managing the RealView ICE Software*.
   - the Options area specifies whether or not certain features are present:
     - **Embedded Trace Macrocell (ETM)**, when selected, indicates that the device has an ETM
     - **Vector Floating Point (VFP)**, when selected, indicates that the device has a VFP unit
     - **Embedded Trace Buffer (ETB)**, when selected, indicates that the device has an ETB.
5. Click on **OK**.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Setting the JTAG clock speed

It is important to select the best JTAG clock speed for your system. Higher clock speeds enable faster downloads, but setting the clock speed too high can result in intermittent faults and reliability problems. If you are experiencing such problems, try reducing the JTAG clock speed. If you are not sure which clock speed to use, try setting the default speed, 10MHz.

Warning

The standard JTAG cable must not be used for clock speeds exceeding 20MHz. For reliable operation at high clock speeds, you must use the LVDS cable.

To set the JTAG clock speed:

1. Ensure that the scan chain controls are displayed (see *Displaying the scan chain controls* on page 4-9).
2. Specify the clocking that you want using the JTAG Clock Speed controls, shown in Figure 4-15:

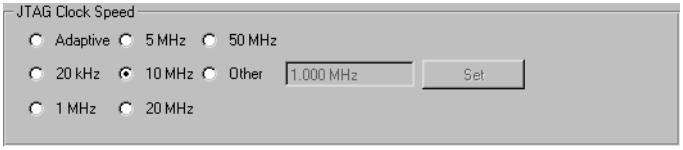


Figure 4-15 The scan chain JTAG Clock Speed controls

   - If you want to use fixed clocking, select the required speed. If the speed you want to use is not available as a preset:
     1. Select **Other**.
     2. Enter the required speed with the **Hz**, **kHz**, or **MHz** suffix as required.

        Note

        Although RealView ICE can support JTAG clock speeds down to 13Hz, your debugging environment might become unstable at speeds lower than 1kHz.

     3. Click on **Set**.
   - If you want to use adaptive clocking, select **Adaptive**. You can use adaptive clocking if the target provides the **RTCK** (Returned **TCK**) signal. This enables you to synchronize the JTAG clock to the processor clock outside the core, which ensures that there are no synchronization problems over the JTAG interface.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Note

Cores ending with -S must use adaptive clocking—that is, ARM926EJ-S™, ARM946E-S™, ARM966E-S™, ARM1026EJ-S™, ARM1136JF-S™, and ARM1136J-S™.

Note

If you use adaptive clocking, the maximum clock frequency is lower than with non-adaptive clocking, because of transmission delays, gate delays, and synchronization requirements. Do not use adaptive clocking unless the hardware design requires it.

For a full description of the concept of adaptive clocking, see Chapter 9 *System Design Guidelines*.

4.2.5 Configuring devices

Before you can configure a device on a particular scan chain, you must ensure that the RVConfig dialog box displays the controls for that device. To do this, select the node for the device in the tree diagram, as shown in Figure 4-16.

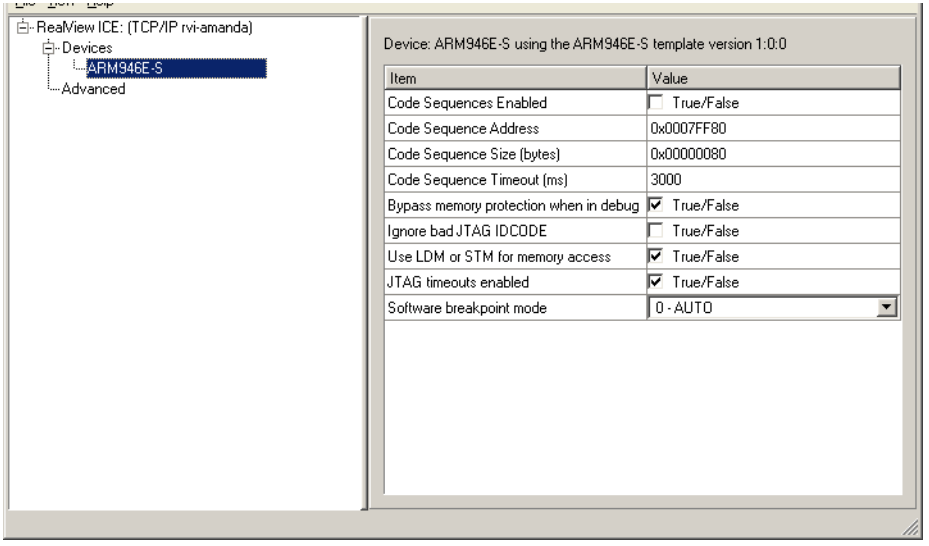


Figure 4-16 Displaying the device controls

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Depending on the device that you have selected, some or all of the following controls might appear:

- *The Code Sequence settings*
- *Bypass memory protection when in debug* on page 4-20
- *Ignore bad JTAG IDCODE* on page 4-20
- *Use LDM or STM for memory access* on page 4-20
- *Software breakpoint mode* on page 4-21
- *Unwind vector catch* on page 4-21
- *Clear break HW on connect* on page 4-22.

Note

If you wish to configure CoreSight systems, refer to the Application Note that accompanies this release.

The Code Sequence settings

The **Code Sequences Enabled** facility allows users to set code sequence default values and sizes without enabling them.

Note

Code sequences are disabled by default, so users must enable these as required.

The **Code Sequence Address** and the **Code Sequence Size (bytes)** values set the virtual address and the size of an area of memory on the target that the RealView ICE software can use. This area of memory must be:

- unused by the target
- readable
- writable
- non-cacheable (for cached targets)
- at least 128 bytes in size.

The RealView ICE software downloads code sequences to this area to perform various tasks, such as cleaning the cache (see *Cached data* on page 5-14) and accessing certain CP15 registers on targets such as the ARM920T™ and ARM1136JF-S. It does not preserve the contents of this area.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Note

You must ensure that the **Code Sequence Address** and the **Code Sequence Size (bytes)** values are correctly set up before you attempt to write to any of the Cache Operations or TLB Operations in the RealView Debugger Register pane. If you do not set these values correctly, RealView Debugger gives one or more of the following errors:

- `Error V28305 (Vehicle): Memory operation failed`
- `Warning: Code sequence memory area size error`
- `Unable to load code sequence into defined memory area.`

The **Code Sequence Timeout (ms)** value sets a timeout for execution of the uploaded code sequence. For most targets, a 500ms timeout is sufficient.

To change the code sequence settings, type the required value into the appropriate text box.

Bypass memory protection when in debug

If **Bypass memory protection when in debug** is selected, any memory protection provided by hardware (such as a memory management or protection unit) is bypassed whenever the target hardware enters debug state. This enables you to access protected memory so that you can set software breakpoints in it, or alter its contents.

Ignore bad JTAG IDCODE

By default, RealView ICE reads the device JTAG IDCODE to verify the integrity of the JTAG connection. The JTAG standard restricts the JTAG IDCODE value to be 32 bits long and requires the least significant bit to be a 1. If RealView ICE reads an invalid (bad) JTAG IDCODE, it assumes that the JTAG connection is not functioning properly, and fails the attempt to connect to the core.

If the device you want to connect to has an invalid JTAG IDCODE, set this option to `True` by checking the checkbox. This instructs RealView ICE to allow connection to the core even if it detects that the JTAG IDCODE is invalid.

Use LDM or STM for memory access

This option is available if you are using an ARM926EJ-S, ARM946E-S, or ARM966E-S processor. Set this option to `True` (checked) if you want to use *Load Multiple Instructions* (LDM) or *Store Multiple Instructions* (STM) to access target memory. You might have to set this option to `False` (unchecked) if you have a peripheral that is not fully compatible with the AMBA™ 2.0 standard, because in such cases LDM and STM might not be compatible.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

JTAG Timeouts Enabled

By default, JTAG timeouts are enabled. You must deselect this option to disable JTAG timeouts when RealView ICE is connected to a core using a low clock speed and adaptive clocking. This is because RealView ICE cannot detect the JTAG clock speed when adaptive clocking is used, and therefore cannot scale its internal timeouts. If a JTAG timeout occurs, the JTAG is left in an unknown state and RealView ICE cannot operate correctly without reconnecting to the core.

Software breakpoint mode

This option allows you to configure how RealView ICE handles software breakpoints. Select the required breakpoint mode:

AUTO This is the default mode for all templates:

- If the core being debugged supports `BKPT` instructions, RealView ICE automatically uses the `BKPT` instruction for software breakpoints.
- If the core being debugged does not support `BKPT` instructions, RealView ICE uses the watchpoint unit resource when you set a software breakpoint. In this case, RealView ICE automatically frees the watchpoint unit resource when all software breakpoints are cleared.

NONE When this mode is selected, you cannot set software breakpoints. If you attempt to set a software breakpoint, RealView ICE gives an error message telling you that there are no free resources to set the breakpoint.

WATCHPOINT

This option instructs RealView ICE to use one watchpoint unit to provide software breakpoint instructions, whether or not the core being debugged supports `BKPT` instructions. Select this option if the core supports `BKPT` instructions but you want to use a watchpoint unit.

BKPT This option instructs RealView ICE to use the `BKPT` instruction to provide software breakpoint instructions, whether or not the core supports this instruction. Select this option if you want to make sure that no watchpoints are used.

Unwind vector catch

This option is available if you are using an ARM10 processor. It instructs RealView ICE to unwind the vector if you have set a vector catch on a SVC, an Undefined instruction, a Prefetch Abort or a Data abort. Unwinding the vector sets the `PC` to the address of the

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

code that caused the exception instead of leaving it at the vector address. The LR and CPSR are restored, and RealView Debugger displays the code at this address. This enables you to more easily examine the code that caused the exception. If you want to run the exception handling code, you must leave this option unchecked.

Note

This option is only activated if a vector catch occurs. If a vector catch is not set, then the exception handler is run as normal.

Clear break HW on connect

This option is available if you are using an ARM11 processor. The ARM11 processor does not clear the breakpoint hardware on connection. Set this option to True to instruct RealView ICE to perform this operation each time you connect.

4.2.6 Advanced configuration

The Advanced settings enable you to change the reset behavior so that it meets the requirements of the target hardware that you are using. Before you can configure the advanced settings, you must ensure that the control pane displays the advanced controls. To do this, select the **Advanced** node in the tree diagram, as shown in Figure 4-17.

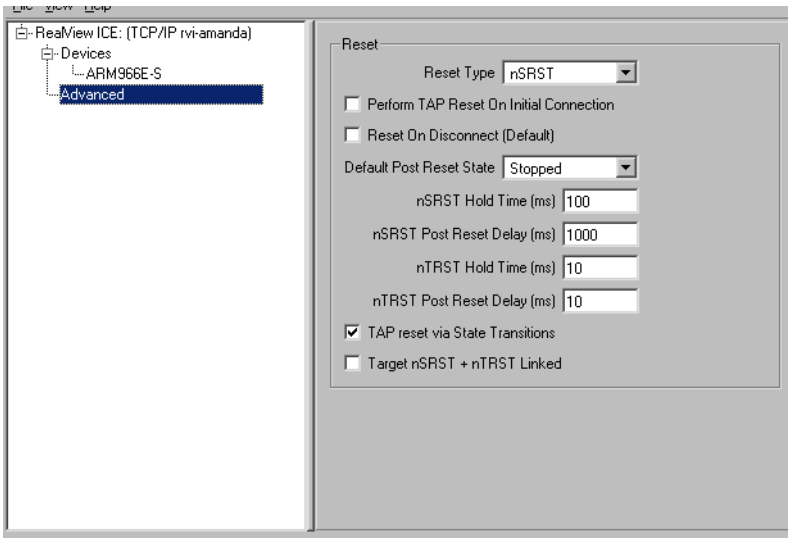


Figure 4-17 Displaying the advanced controls

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

To configure reset behavior:

- Set the **Reset Type** that the RealView ICE run control unit uses to reset the target hardware:

  nSRST Resets the hardware by holding the hardware **nSRST** system reset signal LOW. This is the default.

  nTRST Resets the target TAP by holding the **nTRST** TAP reset signal LOW.

  nSRST+nTRST Resets the hardware and the target TAP by holding both the hardware **nSRST** system reset signal and the **nTRST** TAP reset signal LOW.

  Fake Resets the system by entering supervisor mode, and setting the program counter to the address of the reset vector (known as a *soft reset*).

- Set the required default reset behavior:
  - Select **Perform TAP Reset on First Connect** to reset the target hardware whenever you connect.
  - Select **Reset On Disconnect** to reset the target hardware whenever you disconnect.

- Set **Default Post Reset State** to the required state for the target hardware:

  Running The target hardware is running.

  Stopped The target hardware is stopped.

  Note

  If you want to connect to a running target without performing a reset and without stopping the target, you must do both of the following:

  - In RealView ICE, set the Default Post Reset State to **Running**.
  - In RealView Debugger, connect using the **Connect (Defining Mode)** of **No Reset and No Stop**. For information on setting the connection mode, see the *RealView Debugger Target Configuration Guide*.

- Enter appropriate values in milliseconds for the reset hold times and delays:
  - **nSRST Hold Time (ms)** specifies how long the RealView ICE run control unit holds the hardware **nSRST** system reset signal LOW

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

— **nSRST Post Reset Delay (ms)** specifies how long after the hardware **nSRST** system reset before the RealView ICE run control unit enters the Post Reset State

— **nTRST Hold Time (ms)** specifies how long the RealView ICE run control unit holds the **nTRST** TAP reset signal LOW

— **nTRST Post Reset Delay (ms)** specifies how long after the **nTRST** TAP reset before the RealView ICE run control unit enters the Post Reset State.

- Select **TAP reset via State Transitions** if you want the JTAG logic in the target hardware to be reset by forcing transitions within its state machine. This is done in addition to holding the **nTRST** TAP reset signal LOW. Select this option if **nTRST** is not connected, or if the target hardware requires that you force a reset of the JTAG logic whenever resetting.
- Select **Target nSRST + nTRST Linked** if the target hardware has its **nSRST** and **nTRST** JTAG signals linked.

Note

These settings are sent to a RealView ICE run control unit whenever you connect to the unit from RealView Debugger. They are used as the default reset behavior for all target hardware that you debug with that run control unit. You can override these settings when you are debugging, as described in *Using the Debug tab of the RealView Debugger Register pane* on page 4-28.

Recommended settings for an ARM Integrator board

If you are using an ARM Integrator™ board, ARM Limited recommends that you use the default settings. These are shown in Figure 4-18.

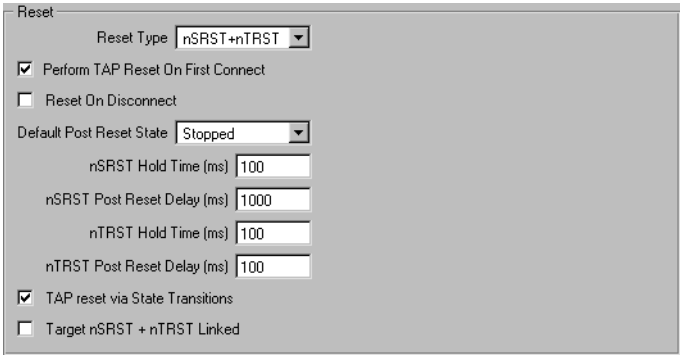


Figure 4-18 Recommended settings for an ARM Integrator board

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

If these operations complete, the FPGA OK LED on the core module is lit. If an Integrator AP board is present and its switches are set for the boot monitor to run, then the MISC LED on the core module, if present, is lit.

Note

The long **nSRST Post Reset Delay (ms)** is to allow the boot monitor to run and remap memory.

4.2.7 Saving your changes

To save any changes that you have made to the configuration, select **File → Save**.

Changes are stored by default in the file `rvi.rvc`, which is located by default in the RealView Debugger `etc` directory:

`install_directory``\RVD\Core\...\etc\rvi.rvc`

The *`install_directory`* is the location where you installed RVDS. The location of your `rvi.rvc` file is displayed in the title bar of the RVConfig dialog box.

You can change the location of the `rvi.rvc` file using the Connection Properties window, or save multiple copies of the file for different target configurations. For more details, see the *RealView Debugger Target Configuration Guide*.

4.2.8 Disconnecting from a RealView ICE unit

You might want to disconnect from a RealView ICE unit if you want to connect to another RealView ICE unit.

To disconnect from a RealView ICE unit:

1. Select the **RealView ICE** node in the tree diagram, to display the RVConfig dialog box as shown in Figure 4-19 on page 4-26.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

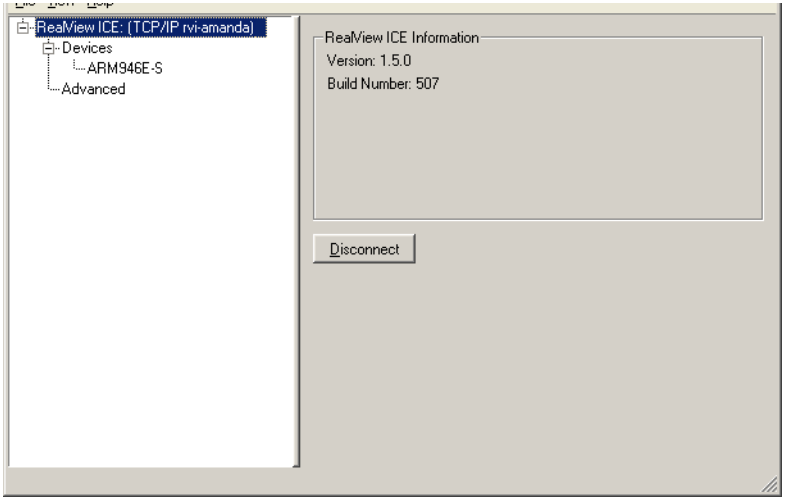


Figure 4-19 Displaying the connection controls

2. Click **Disconnect**.

   If you have unsaved configuration changes, a warning dialog box appears as shown in Figure 4-20.

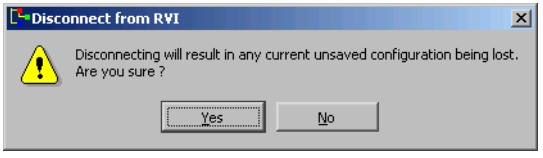


Figure 4-20 Warning when disconnecting with unsaved configuration changes

3. In this warning dialog box:
   - Click **Yes** to disconnect, losing any unsaved configuration data.
   - Click **No** to remain connected. Save your changes, then disconnect.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

4.3 Connecting RealView Debugger to a target using RealView ICE

To connect to your target hardware using a RealView ICE run control unit, use the same RealView Debugger features that you use for any other target. You must ensure that you use the RealView ICE target.

When connecting, an error may appear if the software detects that there is already a connection to the target. This might be because someone else is connected to the target, or because a connection has been left open by software that exited incorrectly. If the error message appears, you should:

- Click **Yes** if you are certain that nobody else is connected. This closes all open connections and then connects you.
- Otherwise click **No**. Determine who else is connected, and ask them to disconnect.

For more information about connecting RealView Debugger to targets, see the RealView Debugger documentation suite (see *ARM publications* on page xix).

See the chapter that describes configuring custom connections in the RealView Debugger Target Configuration Guide.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

4.4 Using the Debug tab of the RealView Debugger Register pane

When you install the RealView ICE software, it adds a **Debug** tab to the Register pane of the RealView Debugger Code window. This controls various internal debugger registers, many of which are specific to RealView ICE. To use this tab, you must first connect RealView Debugger to your target hardware, as described in *Connecting RealView Debugger to a target using RealView ICE* on page 4-27. For more specific information on target connection using RealView Debugger, see the *RealView Debugger Target Configuration Guide*.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 5
Debugging with RealView ICE

This chapter provides information about debugging with RealView® ICE. It contains the following sections:

Note

For more general information about debugging with RealView Debugger, see the RealView Debugger documentation suite (see *ARM publications* on page xix). For information about tracing with RealView ICE, see Chapter 6 *Using RealView Trace*.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.1 Post-mortem debugging

This section describes how to examine the state of a system that has previously been running but is currently not connected to RealView ICE.

Before you can examine a running target with RealView ICE, you must configure the RealView ICE run control unit for that target. If you have a target that is operating without a RealView ICE run control unit connected, and you want to examine it to find out why it is behaving in a particular way, you must power up the RealView ICE run control unit and configure the connection without disturbing the state of the target. This requires that the RealView ICE run control unit is powered before it is connected to the target.

The RealView ICE run control unit includes power conditioning and switching circuitry that enables you to plug and unplug the JTAG cable without affecting the target.

Note

The voltage reference used by the run control unit JTAG circuit is generated from the **VTref** signal present on the JTAG connector. If this signal is not connected at the target, you must modify the target or the JTAG cable to supply a suitable reference. Connecting **VTref** to **Vsupply** is usually sufficient.

To connect to a running target:

1. Ensure that the JTAG input lines **TDI**, **TMS**, **nSRST**, and **nTRST** have pull-up resistors (normal practice), and **TCK** has a pull-down resistor, so that when the adaptor is disconnected from the target these lines are in their quiescent state.
2. Plug the power jack into the run control unit.
3. Configure the RealView ICE connection (see *Using the RVConfig dialog* on page 4-3). You must do one of the following:
   - load a configuration that you have previously saved
   - manually configure the connection
   - autoconfigure using a separate test system.

   Note

   Do not use autoconfigure on the target to be debugged, because doing so might reset the processor.
4. If the target processor does not have any CP15 registers, you must explicitly configure the endianness, as described in the *RealView Debugger Target Configuration Guide*.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Note

Do not automatically detect the endianness of target processors that do not have a CP15 register. Doing so might disturb the state of the processor.

5. Plug the 20-way JTAG cable into the target.
6. Start the debugger, and connect to the running target.

   For RealView Debugger, use one of the following **Connect (Defining Mode)** options:

   - **No reset and no stop**
   - **Reset and no stop**

   See the *RealView Debugger Target Configuration Guide* for more details on connection modes.
7. To get a high-level (source code) view of the problem, you must load the symbol table for your target program into the debugger.

   For RealView Debugger:

   a. Open the Load File to Target dialog box.

   b. Locate the target program.

   c. Select **Symbols Only**.

   d. Click **Open**. The Load File to Target dialog box closes.

   See the *RealView Debugger User Guide* for more details on loading images.
8. When you have finished, unplug the JTAG connector to restart the system, if required, then exit the debugger.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.2 Semihosting

Semihosting enables the ARM® processor target to make I/O requests to the computer running the debugger. This means the target does not require a screen, keyboard, or disk during the development period. These requests are made as a result of calls to C library functions, for example, `printf()` and `getenv()`. Semihosting using RealView ICE is described in the following sections:

- *Enabling semihosting*
- *Adding an application SVC handler when using RealView ICE* on page 5-5.
- *M3 semihosting* on page 5-7

5.2.1 Enabling semihosting

When using RealView ICE, semihosting is handled by emulating a `SVC` exception handler using breakpoints. You can modify this semihosting mechanism using the following RealView Debugger internal registers:

`SEMIHOSTING_ENABLED`

By default, this variable is set to `1` to enable breakpoint semihosting but you can set it to the following values:

0 Disables semihosting.

1 Enables semihosting.

`SEMIHOST_VECTOR`

This variable controls the location of the breakpoint set by the RealView ICE software to detect a semihosted `SVC`. It is set to `8` by default, unless you have specified that high vectors are in use.

In RealView Debugger, these internal registers are accessed using the **Debug** tab in the Register pane of the Code window. See *Using the Debug tab of the RealView Debugger Register pane* on page 4-28 and the RealView Debugger documentation suite for more information (see *ARM publications* on page xix).

Semihosting

Semihosting involves setting a breakpoint either on the SVC vector or somewhere else in cooperation with your own SVC handler, depending on the value of `SEMIHOST_VECTOR`.

Note

M3 does not provide vector catch on SVC. For details on M3 semihosting see *M3 semihosting* on page 5-7.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

When the breakpoint is hit, RealView ICE interprets it as a semihosting request:

- the processor registers and memory are read as required to decode the request
- the request is executed on the host
- the return value is placed in register R0 and, when required, memory is modified
- the pc is modified so that the next instruction is the instruction following the SVC
- execution is resumed.

By default, the ARM C library code uses semihosting for I/O operations and for certain system-specific settings, such as SP value.

Note

Using RealView ICE semihosting with systems that include time-sensitive interrupt-driven software is not recommended. The processor is halted while a semihosting operation is performed, and interrupts are therefore missed.

The breakpoint on the SVC vector uses breakpoint resources that might be required for other purposes.

5.2.2 Adding an application SVC handler when using RealView ICE

Many applications require their own SVC handlers in addition to semihosting SVCs. To ensure that the application SVC handler cooperates with the RealView ICE semihosting mechanism:

1. Install the application SVC handler into the vector table.
2. Modify the value of `SEMIHOST_VECTOR` to point to a location that is only reached if your handler does not recognize the SVC, or recognizes it as a semihosting SVC.

For example, a particular SVC handler might detect if it has failed to handle a SVC and branch to an error handler. An example of a basic exception handler is shown in Example 5-1.

Example 5-1 Basic SVC handler

```
                          ; r0 = 1 if SVC handled
CMP r0, #1                ; Test if SVC has been handled.
BNE NoSuchSVC             ; Call unknown SVC handler.
LDMFD sp!, {r0}           ; Unstack SPSR...
MSR spsr_cf, r0           ; ...and restore it.
LDMFD sp!, {r0-r12, pc}^ ; Restore registers and return.
```


Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

You can modify this code for use in conjunction with RealView ICE semihosting as shown in Example 5-2.

Example 5-2 SVC handler with RealView ICE link

```
                        ; r0 = 1 if SVC handled
CMP r0, #1              ; Test if SVC has been handled.
LDMFD sp!, {r0}         ; Unstack SPSR...
MSR spsr_cf, r0         ; ...and restore it.
LDMFD sp!, {r0-r12, lr} ; Restore registers.
MOVEQS pc, lr           ; Return if SVC handled.
Semi_SVC
MOVS pc, lr
```

You must then set up the SEMIHOST_VECTOR with the address of Semi_SVC. The instruction at this address is never actually executed because the RealView ICE software returns directly to the application after processing the semihosted SVC. Using a normal SVC return instruction ensures that the application does not crash if the semihosting breakpoint is not set up.

If the application is linked with the semihosted ARM C library, and therefore uses the C library startup code, you must change the contents of SEMIHOST_VECTOR before the application installs its own handler, typically by setting a breakpoint in the main code. This is because, if SEMIHOST_VECTOR is set to the fall-through part of the application SVC handler before the application starts execution, the semihosted SVCs that are called by the library initialization can trigger an unknown breakpoint error. At this point, the SVC vector has not yet had the application handler written to it, and might still contain the software breakpoint bit pattern. This triggers a breakpoint that the RealView ICE software does not know about, because the SEMIHOST_VECTOR address has moved to a place that cannot currently be reached.

Note

If semihosting is not required by your application, including the startup code, you can simplify this process by setting SEMIHOST_ENABLED to zero.

You must take care when moving an application that previously ran in conjunction with the Angel debug monitor onto a RealView ICE system. On Angel debug monitor systems, application SVC handlers are typically added by moving and adjusting the contents of the Angel-installed SVC vector to another place, and installing the application SVC handler into the SVC vector. This method does not apply to the RealView ICE software because there is no instruction to move out of the SVC vector,

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

and no code to jump to. Therefore, when moving an application onto a RealView ICE-based system, you must convert to the RealView ICE way of installing the application and semihosted SVC handlers.

5.2.3 M3 semihosting

Since M3 does not provide vector catch on SVC, and the vector table contains jump addresses rather than instructions, semihosting cannot be supported using an SVC instruction.

As an alternative, semihosting is implemented using a software breakpoint which is recognized as a semihosting break by the debugger. The specific breakpoint instruction used is set by using the Thumb breakpoint config item.

When the semihosting break is executed, the semihosting call is processed in the normal way. After processing, execution continues from the instruction that follows the software breakpoint. The debugger does not stop on the breakpoint.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.3 Breakpoints

This section describes how RealView ICE implements breakpoints. It contains the following sections:

- *Hardware Breakpoints*
- *Software instruction breakpoints* on page 5-9
- *Processor exceptions* on page 5-10
- *Breakpoints and the program counter* on page 5-10
- *Interaction with RealView Debugger* on page 5-11
- *Problems setting breakpoints* on page 5-13.

5.3.1 Hardware Breakpoints

Some processors contain dedicated hardware resources, such as ARM EmbeddedICE® logic, for matching against specific hardware events. RealView Debugger enables you to configure these resources to implement instruction and data breakpoints.

Note

Data breakpoints are also sometimes referred to as watchpoints.

The resources available depend on the processor you are using. See the data sheet for your processor for information.

Hardware breakpoints might also provide additional matching capabilities. Examples of this include matching on an external signal, and distinguishing between privileged and non-privileged accesses. The Set Address/Data Breakpoint dialog box displays the capabilities of your hardware. See the chapter on breakpoints in the *RealView Debugger User Guide* for information on accessing and using this dialog box.

Hardware instruction breakpoints do not require the instruction in memory to be changed. This means that they can be used to debug code in Flash and ROM, and can be used with self-modifying code.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.3.2 Software instruction breakpoints

For processors that do not support hardware instruction breakpoints, or in cases where you have used up all the available hardware breakpoint resources, you can use software instruction breakpoints. Software breakpoints modify the instruction in memory to create a special value that causes the processor to enter debug state when executed. The value written to memory depends on the processor you are using. For ARM processors, one of the following schemes is used, depending on the architecture and processor revision:

- An undefined instruction is written to memory, and a hardware breakpoint resource is used to spot this instruction being executed. The processor enters debug state when the hardware breakpoint unit spots the undefined instruction entering the execute pipeline stage.
- An ARMv5 `BKPT` instruction is written to memory, and a hardware breakpoint resource is used to spot this instruction being executed. The processor enters debug state when the hardware breakpoint unit spots the `BKPT` instruction entering the execute pipeline stage.
- An ARMv5 `BKPT` instruction is written to memory. When this instruction is executed, the processor automatically enters debug state.

Where a hardware breakpoint unit is used to spot software instruction breakpoints, only a single hardware resource is used, no matter how many software instruction breakpoints are set. If you have difficulty setting software instruction breakpoints, you might have to free up a hardware breakpoint resource first.

Software breakpoints cannot be used to debug code in Flash or ROM, and can be unreliable in self-modifying code.

Note

When viewing memory or disassembly, RealView ICE reports the actual contents of memory. Prior to running, any software breakpoints are written to memory. When the core halts, the software breakpoints are removed from memory. On a number of cores, it is not possible to access memory while running, and means that if the user disconnects RealView-ICE from the core while the target is running, the breakpoints are left in memory. If the core subsequently executes one of the instructions, then (depending on the core architecture) the core will either stop at the software breakpoint or cause the core to take an undefined exception.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.3.3 Processor exceptions

Some processors provide dedicated hardware to enter debug state when a predetermined event occurs. Available processor events are displayed in the Processor Exceptions List Selection dialog box. See the chapter that describes breakpoints in the *RealView Debugger User Guide* for more information on this dialog box.

Most ARM cores provide hardware to enter debug state when an exception occurs. This is called *vector catch*. Some ARM cores, such as ARM7, do not provide vector catch hardware. For these cores, RealView ICE simulates vector catch using instruction breakpoints.

Note

If the exception vectors are in ROM, RealView ICE must use hardware breakpoints to simulate vector catch. This reduces the number of resources available for other purposes.

If RealView ICE uses an instruction breakpoint to simulate reset vector catch, the breakpoint might not be hit when a reset occurs. This is because most systems remap flash or ROM at the exception vectors during reset, and this displaces any instruction breakpoint that might be set. The following warning is output to the RealView Debugger console if RealView ICE simulates reset vector catch using an instruction breakpoint:

```
Warning: A software breakpoint is being used to simulate reset vector catch.
This may fail to be hit if the memory is remapped when a reset occurs.
```

The exact behavior of the ARM vector catch hardware depends on the core. ARM9 and ARM10 processors enter debug state only when the specified exception occurs. Other processors such as ARM11 enter debug state whenever the instruction at the exception vector is executed, regardless of whether the exception occurs or not.

5.3.4 Breakpoints and the program counter

This section describes the value of the program counter when a breakpoint is taken for the following events:

- *Hardware data breakpoints*
- *Hardware instruction breakpoints* on page 5-11
- *Software instruction breakpoints* on page 5-11
- *Processor events* on page 5-11.

Hardware data breakpoints

The address of the program counter after hitting a hardware data breakpoint depends on the processor being used.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

For ARM cores, a skid of either one or two instructions occurs after a data breakpoint is hit. This means that the instruction that generated the breakpoint, and possibly the one after that, are both executed. The program counter shown in RealView Debugger might not be the address of the instruction that generated the breakpoint.

Hardware instruction breakpoints

The address of the program counter after hitting a hardware instruction breakpoint depends on the processor being used.

For ARM cores, no skid occurs after hitting a hardware breakpoint. This means that the instruction that generated the breakpoint has not been executed, and the program counter is set to this address.

Software instruction breakpoints

The address of the program counter after hitting a software breakpoint is always the address of the breakpoint. Unless the instruction is a `BKPT` instruction, the instruction that generated the breakpoint is not yet executed.

Processor events

The address of the program counter after a processor event is hit depends on the processor being used. For ARM processors, vector catch hardware stops with the program counter on exception vector, before the instruction at that address is executed.

5.3.5 Interaction with RealView Debugger

This section describes how the breakpoint handling in RealView ICE interacts with the breakpoint handling in RealView Debugger. It contains the following sections:

- *Break details or break capabilities*
- *Memory maps* on page 5-12
- *Stepping* on page 5-12
- *Semihosting* on page 5-13
- *Resource allocation* on page 5-13.

Break details or break capabilities

You can find out what hardware breakpoint resources are available by viewing the break details or break capabilities in RealView Debugger (see the chapter that describes breakpoints in the *RealView Debugger User Guide* for more information). All ARM cores provide at least two hardware instruction breakpoint resources.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Memory maps

RealView Debugger enables you to define a memory map to describe the layout and type of memory in your system (see the chapter that describes memory mapping in the *RealView Debugger User Guide* for detailed information). When you set a breakpoint, areas of memory that are marked as read-only, such as Flash and ROM, automatically use hardware instruction breakpoints. All other types of memory use software instruction breakpoints by default.

Stepping

When you step through code, the debugger usually sets a temporary breakpoint on the destination address. If the code is in read-only memory, or if the software breakpoint implementation requires hardware assistance, a hardware breakpoint is used for this. If you are unable to step, you might have to free up a hardware breakpoint resource.

Some processors, such as ARM9, provide dedicated single-step hardware. RealView ICE uses this hardware if it is available, but steps larger than a single instruction might revert back to using breakpoints, in order to improve efficiency.

Note

For ARM7, ARM9, ARM11 or Cortex A8 processors, interrupts are disabled when single-stepping with RealView ICE. For the ARM10 processor, and for Cortex M3, interrupts are enabled when single-stepping with RealView ICE.

Interrupt behavior applies only to RVI single-instruction stepping. Higher-level stepping depends on the strategy in RealView Debugger, that is, whether you've used the "place Breakpoint and run" method, or the "multiple single-instruction steps" method.

Note

When hardware single-step is used, RealView ICE prevents the core from processing any pending interrupts.

For further information, see the chapter on controlling image execution in *RealView Debugger User Guide*.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Semihosting

RealView ICE provides an implementation of ARM semihosting (see *Semihosting* on page 5-4). If the semihosting vector is at the address of the SVC vector, RealView ICE uses SVC vector catch to detect semihosting operations (see *Processor exceptions* on page 5-10 for detailed information). In all other cases, RealView ICE uses a breakpoint at the semihosting vector to detect semihosting operations.

In both cases, the use of semihosting might use up hardware breakpoint resources, resulting in fewer resources available for your own use. Therefore, you might want to disable semihosting if your program does not use it.

Resource allocation

RealView ICE allocates hardware breakpoint resources as they are received, rather than allocating all the resources at the same time when the debugging session begins. Therefore, if you attempt to set a breakpoint when there are insufficient resources available, RealView ICE displays an error message as soon as you try to set the breakpoint, rather than waiting until debugging begins.

5.3.6 Problems setting breakpoints

If you have problems stepping or setting breakpoints, it might be because you have run out of hardware breakpoint resources. To work around this, you can try freeing some hardware breakpoint resources then repeating the action. Some examples of how you can free hardware breakpoint resources include:

- disable any breakpoints that you do not require
- change hardware breakpoints to software breakpoints where possible
- disable any processor events that you do not require
- disable semihosting if you are not using it.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.4 Cached data

When debugging a cached processor, RealView ICE uses the strategies described below.

1. On debug entry.
   - RVI will force Write-Through (WT) on cores which support this debug feature.
   - RVI will disable cache line fill on cores which support disabling of this feature in debug.
   - RVI will disable Translation Look-aside Buffer (TLB) loads on cores which support disabling of this feature in debug.
   - If data is read from cacheable memory, it will only be read into the caches if, and only if, disable linefill is not possible.
   - TLB matches and caches remain enabled.
2. On data write.
   - If WT is possible, nothing cache-related is performed.
   - If WT is not possible, strategy is core size- and data size-dependent:

   a. RVI may write to memory with caches enabled, and then write disabled, effectively simulating write through.

   b. RVI may clean and invalidate the D_{cache} and disable it. (The 940T requires that Code Sequences are enabled to do this.)
3. On restart into debug.
   - On cores which support the features, forced WT is removed, linefills are re-enabled, and TLB loads are enabled. If, and only if, data has been written, the I_{cache} is invalidated. If, and only if, D_{cache} has been disabled, then it is re-enabled.

   Data writes which could cause cache operations described above include user accesses via RealView Debugger, downloads, and any software breakpoints present in the system.

Note

For the ARM940T core you must configure the **Code Sequence...** settings in the **Debug** tab before attempting to debug with the cache(s) enabled. For more specific information on target connection using RealView Debugger, see the *RealView Debugger Target Configuration Guide*.

When the cache is enabled, the speed of semihosting decreases.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

5.5 Debugging applications in ROM

This section describes some of the issues involved with debugging applications in ROM using RealView ICE:

- *Debugging from reset*
- *Debugging systems with ROM at the exception vector* on page 5-16.

5.5.1 Debugging from reset

You can use the RealView ICE software to debug systems running in ROM. Typically, when target hardware has an application stored in ROM, and is powered up, it begins running the application. Therefore, when the debugger is started up on the host, the processor on the target is stopped. At this stage, the application can be at any point in its execution lifetime, depending on when the debugger was started.

This means that you can examine the state of the system and restart execution from the current place. In some cases, this is sufficient. However, in many cases it is preferable to restart execution of the application as if from power-on. There are two ways to do this:

- *Simulating a reset* on page 5-16
- *Carrying out a real reset* on page 5-16.

When you debug code that is running from ROM, you must ensure that at least one breakpoint unit remains available so that you can set breakpoints on code in ROM, because you cannot use software breakpoints for this purpose. On a processor without vector catch hardware, you must disable semihosting and vector catching as soon as possible after starting up the debugger. This can reduce the chances of the debugger taking these units for its own use.

You must set up any ROM breakpoints before any non-ROM breakpoints are set, so that the breakpoint units do not become full before you attempt to set the ROM breakpoint.

Another factor in debugging a system in ROM is that the ROM image does not contain any debug information. When debugging using the RealView ICE software, symbol or source code information is available by loading the relevant information into the debugger from a file on the host. For example, in RealView Debugger:

1. Open the Load File to Target dialog box.
2. Locate the ROM image.
3. Select **Symbols Only**.
4. Click **Open**. The Load File to Target dialog box closes.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Simulating a reset

You can often simulate a reset from within the debugger by setting:

- the pc to the address of the reset vector
- the CPSR to change into Supervisor mode with interrupts disabled.

This simulates the state of the ARM processor at power-on or reset, but it does not perform post-reset tasks such as resetting the memory map, or initializing any target-specific features such as peripheral registers. It is recommended that you modify these target-specific features to resemble their startup state before executing the application again, if possible. You can automate this procedure using the scripting facilities of RealView Debugger.

Carrying out a real reset

Depending on the design of the reset circuitry, you might be able to carry out a real reset of the board. Two forms of reset are required on the board:

- a full power-on reset that resets everything on the board
- a Reset button that resets everything on the board except the EmbeddedICE logic.

Note

The Reset button mentioned here should not be confused with the RST button located on the RVI unit itself, as described in *The RealView ICE run control unit* on page 1-8.

If your target implements a Reset button that drives **nTRST** in addition to **nSRST**, then the EmbeddedICE logic is reset along with the board, and the debugger might not be able to regain synchronization. This design is not recommended. See Chapter 9 *System Design Guidelines* for more information about the different forms of reset.

If a vector catch is set on the reset vector (or on the start address of the reset code) and the recommended reset circuit is used, when the target is reset, it halts on reset as required.

The ARM Integrator™ boards implement the required two levels of reset. The reset switch carries out the required initialization reset, thus enabling debug from reset. All that is required is to set the hardware breakpoint, and then press the Reset button.

5.5.2 Debugging systems with ROM at the exception vector

When debugging processors without vector catch hardware and with ROM rather than RAM at the exception vector, you must disable vector catching. This prevents RealView ICE from trying to set software breakpoints on the vector table.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 6
Using RealView Trace

This chapter describes RealView® Trace v1.0 and tells you how to connect the parts of RealView Trace and RealView ICE together. It also tells you where to find information on using RealView Trace with RealView Debugger. It contains the following sections:

Note

RealView Trace is only available for Windows platforms.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6.1 About RealView Trace

The RealView Trace data capture unit works in conjunction with the ARM® RealView ICE run control unit. Together, they provide real-time trace functionality for software running in leading edge *System-on-Chip* (SoC) devices with deeply embedded cores that contain the *Embedded Trace Macrocell* (ETM) logic.

RealView Trace forms one component in the ARM real-time trace debugging system, together with RealView ICE and RealView Debugger.

A typical system is shown in Figure 6-1.

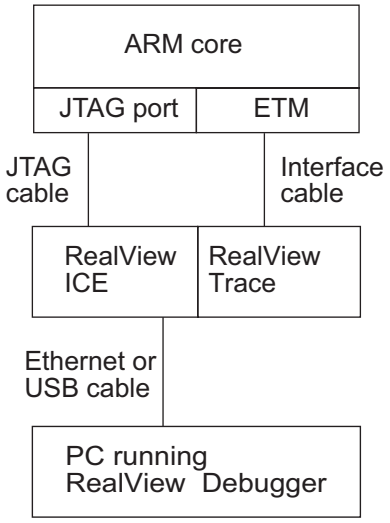


Figure 6-1 Trace system

RealView Trace has the following features:

- It passively collects information from an ARM architecture-based SoC containing an ETM. The ETM monitors the ARM instruction and data buses at full core speeds.
- It collects trace information at clock speeds of up to 250MHz.
- Uploading to RealView Debugger uses Ethernet 10/100baseT or USB.
- Data port widths of 4, 8, 16, and 32 bits are supported.
- It supports a half rate trace clock that captures data on both the rising and falling clock edges.
- The SoC voltage can be in the range of 0.9-5V.
- Trace information can be time stamped to a resolution of 10ns.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6.1.1 The RealView Trace product

The RealView Trace product comprises:

- the RealView Trace data capture unit, shown in Figure 6-2.

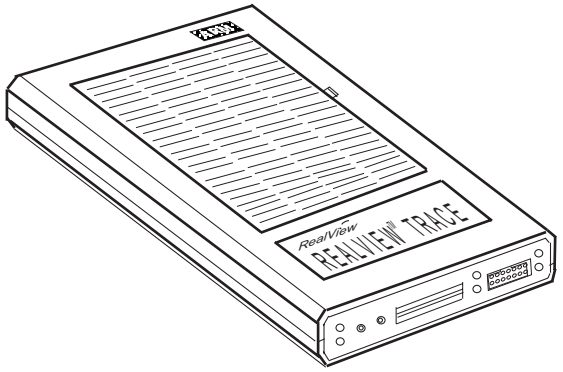


Figure 6-2 The RealView Trace data capture unit

- a cable to connect the run control unit to a trace port
- a printed copy of this User Guide.

6.1.2 Front panel layout

The layout of the RealView Trace front panel is shown in Figure 6-3.

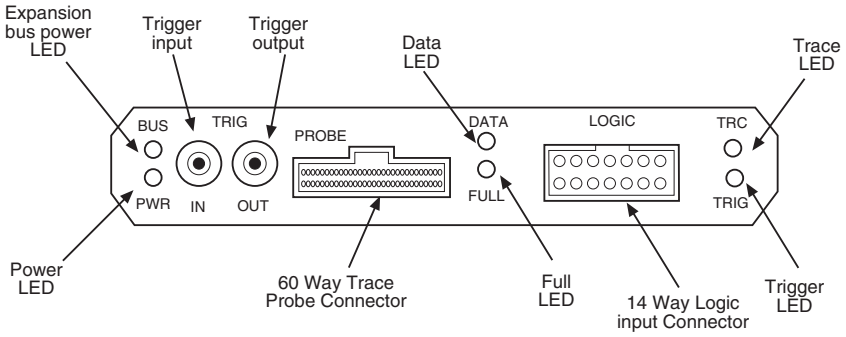


Figure 6-3 RealView Trace panel layout

See Appendix C *RealView Trace Interface Connections* for a detailed description of the interface connections.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Note

The Trigger input, Trigger output and Logic connectors are not supported by RealView ICE v3.0.

6.1.3 Capture rates

The ETM on the target board can output 4, 8, or 16 trace data bits. Half-rate clocking enables data to be output from the ETM on both edges of **TRACECLK**. This effectively halves the clock frequency.

Packing modes enable 2 or 4 consecutive trace samples to be written to the same memory location within RealView Trace. This increases the trace depth. It has the disadvantage of coarser time-stamping. Time-stamping can be disabled completely to increase trace depth even more. The system has the capability to set the port width automatically from the Configure ETM dialog box in RealView Debugger.

Half rate clocking and packing mode facilities provide correct operation at **TRACECLK** frequencies above 150MHz. Below 150MHz, the RealView Trace system operates without these facilities. With these facilities enabled, **TRACECLK** speeds of up to 250MHz are supported.

6.1.4 Availability and compatibility

RealView Trace is available from ARM Limited and its resellers. Contact ARM Limited directly regarding OEM licenses.

Debugger

RealView Trace can only be used with RVDS and RealView ICE.

Target board

The target board must have a device with any of the ARM-processor family, and an ETM (v1.x, v2.x, v3.0, v3.1, v3.2 and v3.3). The board connects to the RealView Trace unit using the connector described in Appendix C *RealView Trace Interface Connections*.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6.2 System requirements

This section describes the hardware and software requirements of RealView Trace:

- *Host software requirements*
- *Host hardware requirements*
- *Target hardware requirements*.

6.2.1 Host software requirements

The software component of RealView Trace is supplied with RealView ICE. RVDS is required to use RealView Trace.

Note

RealView Trace is only available for Windows platforms.

6.2.2 Host hardware requirements

To run RealView Trace, you must have a PC running RealView Debugger and RealView ICE, a RealView ICE run control unit, a RealView Trace data capture unit, and a network card if you are connecting using TCP/IP.

6.2.3 Target hardware requirements

RealView Trace supports processors containing any of the ARM processor family, and ETM v1.x, v2.x, v3.0, v3.1, v3.2 and v3.3. The board containing the processor must have a trace port connector.

Note

RealView ICE supports systems with multiple trace sources. In such systems, however, you can configure only one source to output trace during the lifetime of the debug session. For information on how to configure the trace source, refer to the Application Note that accompanies this release.

Caution

Target hardware running at high frequencies and not following the design guidelines specified in Appendix D *Designing the Target Board for Tracing* might exhibit irregularities in the trace data. Typical symptoms of this are missed triggers, trace data synchronization failures, and memory access failures. Ensure that your target hardware is capable of running at the selected frequency.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6.3 Installing RealView Trace

The RealView Trace software is automatically installed with the RealView ICE installation. If you are already using RealView ICE when you purchase RealView Trace, you receive a RealView ICE Update application along with the RealView Trace data capture unit. See Chapter 7 *Managing the RealView ICE Software* for information on installing the update.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6.4 Connecting the RealView Trace hardware

This section explains how to set up the hardware for RealView Trace.

6.4.1 What you require

To set up the hardware, you require the following from the RealView Trace product kit:

- The RealView Trace data capture unit, shown in *The RealView Trace data capture unit* on page 6-3.
- Eight plastic spacers (four 16mm and four 8mm), supplied with the RealView Trace data capture unit.
- The 60-way interface cable (a flat ribbon cable with a square *Insulation Displacement Connector* (IDC) socket at each end).
- The trace probe. This is a small PCB that contains the interface circuits that buffer the signals between the target board and the interface cable.
- A power supply unit is not supplied, because the RealView Trace unit is powered by the RealView ICE unit.

You must also provide some target hardware containing a device supported by RealView Trace (see *Target hardware requirements* on page 6-5).

6.4.2 Connection instructions

Before updating the software, you must connect up the RealView ICE and RealView Trace hardware. To do this:

1. Ensure that the RealView ICE unit is disconnected from the power supply and from the target board.

   Warning

   Failure to remove the power from the RealView ICE unit before connecting it to the RealView Trace unit can cause damage to the hardware and/or personal injury.

2. Remove the plastic ventilation grill from the top of the RealView ICE unit by unclipping it.
3. Screw the four 16mm plastic spacers onto the four 8mm plastic spacers exposed inside the RealView ICE unit, as shown in Figure 6-4 on page 6-8.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

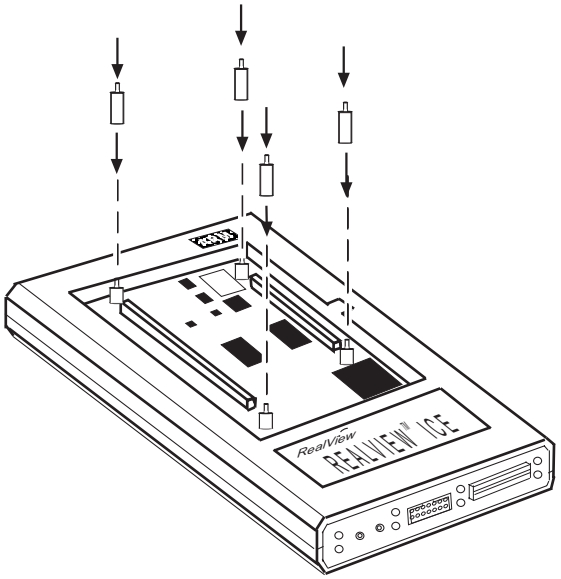


Figure 6-4 Positioning of 16mm plastic spacers

4. Using the four lengthened threaded inserts as guides, plug the RealView Trace unit into the RealView ICE unit. Apply downward pressure directly above the two connectors to ensure a positive fit.

5. Screw the four 8mm plastic spacers onto the four threaded inserts exposed through the PCB inside the RealView Trace unit. This fastens the two units together, making the assembly safe for transportation. Figure 6-5 shows a profile view of the connected units.

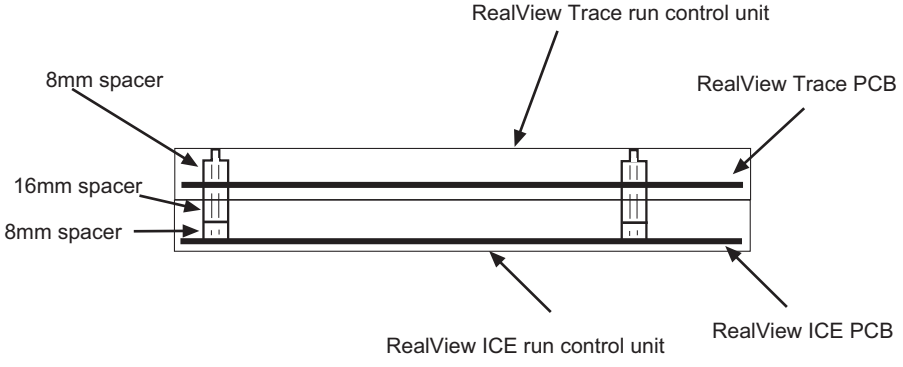


Figure 6-5 Profile view of connected units

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6. Clip the plastic ventilation grill that you removed from the RealView ICE unit onto the top of the RealView Trace unit.
7. Connect one end of the interface cable to the RealView Trace connector, and the other end of the cable to the trace probe.
8. Connect one end of the JTAG cable or LVDS probe to the respective JTAG socket on the RealView ICE run control unit. Connect the other end to the 20-way JTAG header on the target hardware.

   If your target board does not have separate trace and RealView ICE sockets, use the RealView ICE socket on the trace probe as shown in Figure 6-6.

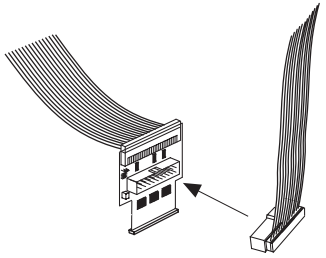


Figure 6-6 RealView ICE connector on probe

9. Plug the trace probe into the trace connector on the target board.
10. Connect the RealView ICE unit to a network shared by a PC running RealView Debugger.
11. Power up the RealView ICE unit and the target board. The RealView Trace unit is powered from the RealView ICE unit, and does not require a separate power supply.

Warning

Do not obstruct the ventilation grills on the top of the RealView Trace unit or the bottom of the RealView ICE unit, because doing so causes the units to overheat.

Figure 6-7 on page 6-10 shows a typical system.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

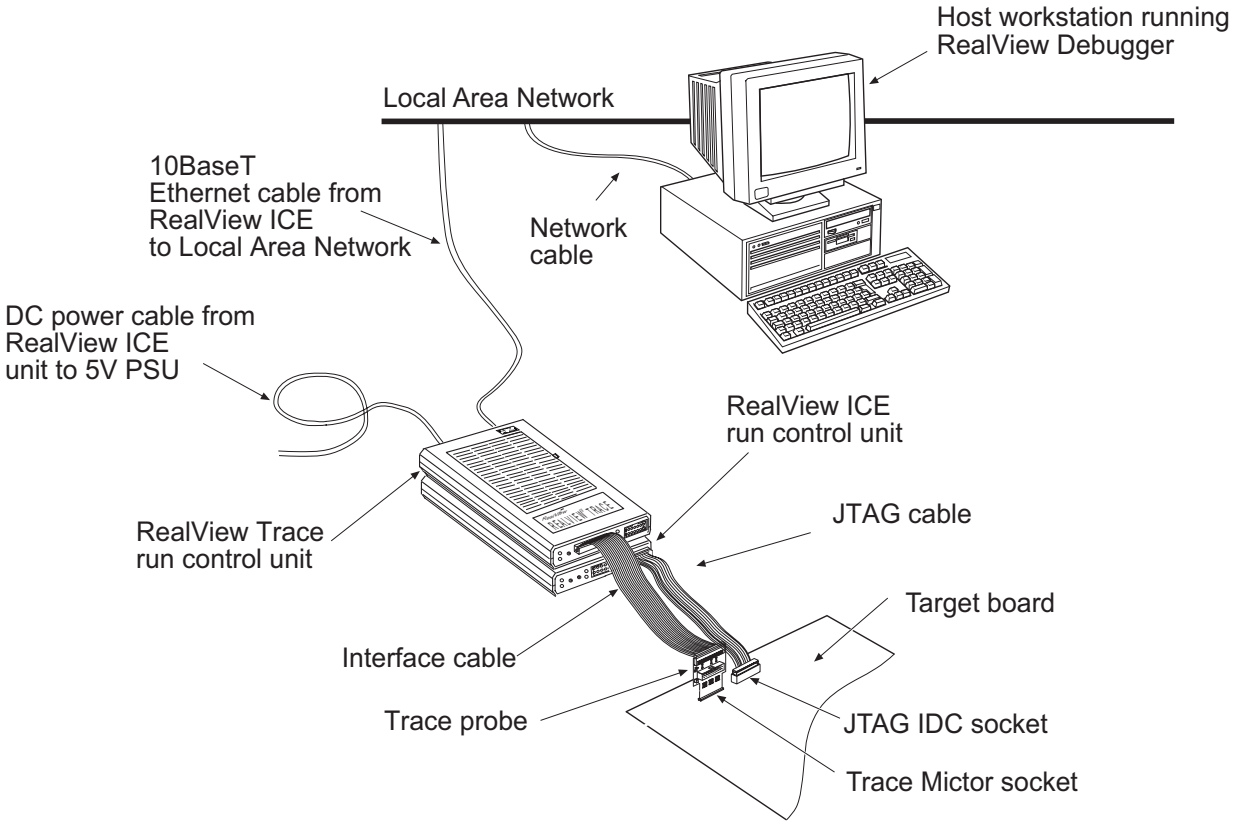


Figure 6-7 RealView Trace connections using an Ethernet cable

Note

You can also use the following alternative connections:

- use a USB cable to connect the RealView ICE run control unit directly to your host workstation
- connect one end of the JTAG cable or LVDS probe to the trace probe JTAG socket.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

6.5 Configuring RealView Debugger for trace capture

When you have installed RealView ICE and RealView Trace, connected up the hardware, and configured RealView ICE, you must configure RealView Debugger to use RealView Trace.

For full details on how to capture trace with RealView Debugger, see the *RealView Debugger Trace User Guide*.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 7
Managing the RealView ICE Software

This chapter tells you how to manage and update the software that is installed on the RealView® ICE run control unit, using the RealView ICE Update (RVI Update) application. It contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.1 Starting the RVI Update application

To start the RVI Update application:

- On Windows, select **Start → Programs → ARM → RealView ICE v3.0 → RealView ICE Update**.

  Note

  If you are using the default Windows XP settings, select **All Programs**.

- On Red Hat Linux, select the appropriate shortcut. This depends on the version of Red Hat Linux and the desktop environment that you are using. If no desktop shortcut is available, enter the command `rviupdate` at the command line.

The RVI Update application is displayed, as shown in Figure 7-1.

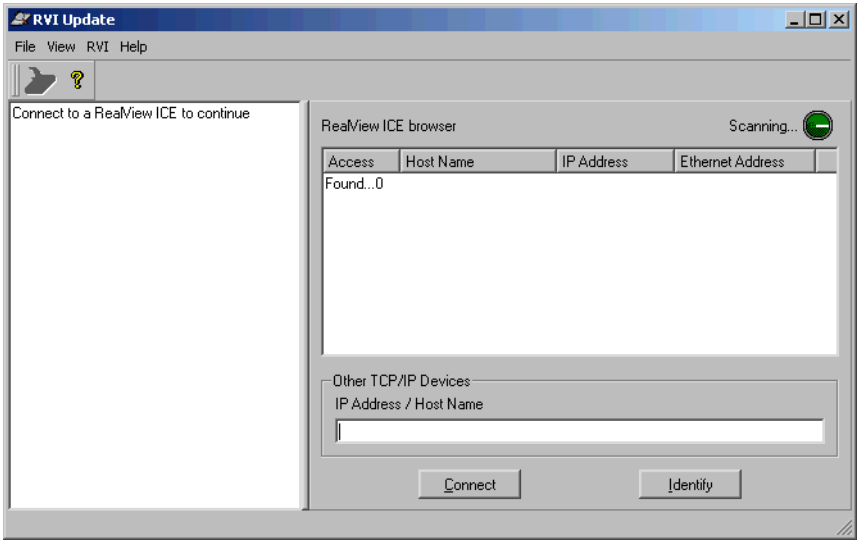


Figure 7-1 RVI Update application

Note

Prior to proceeding any further, you should ensure that you are using release v1.5 (or above) of the software. Using an earlier version may result in an error message appearing, as described in *Procedure for installing an update or patch* on page 7-10.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.2 Connecting to a RealView ICE unit

This section describes the RVI Update features available for connecting to a RealView ICE unit. It includes:

- *Scanning for RealView ICE units*
- *Identifying your RealView ICE unit* on page 7-4
- *Viewing the installed components* on page 7-5
- *RVI Update tasks* on page 7-5
- *Disconnecting from the RealView ICE unit* on page 7-5
- *Exiting the RVI Update application* on page 7-6
- *Troubleshooting RealView ICE connections* on page 7-6.

7.2.1 Scanning for RealView ICE units

When you start RVI Update, it scans for run control units that are connected to your local network. The **Scan** button becomes animated to indicate that a scan is in process. When RealView ICE finds a unit, it adds it to the list of available units, as shown in Figure 7-2.

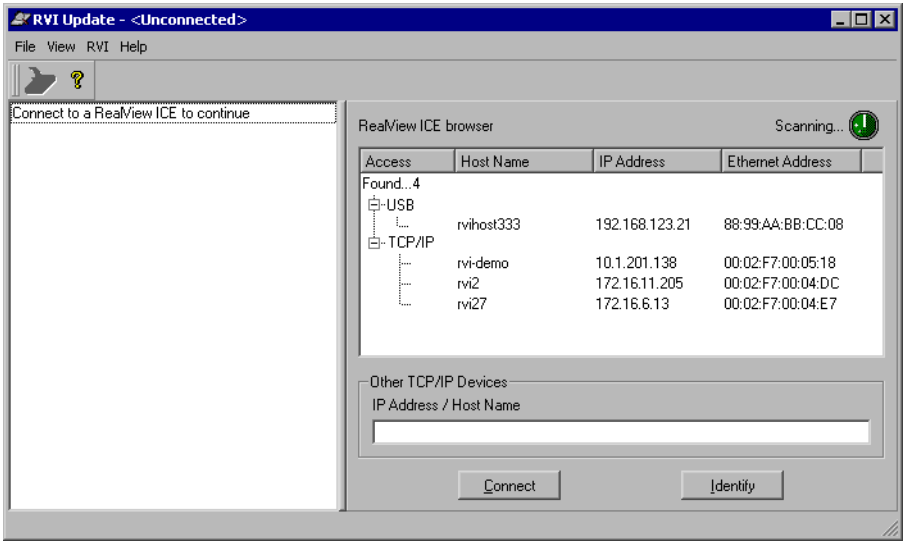


Figure 7-2 RVI Update application showing available units

If you want to stop scanning, click **Scan**. You can click **Scan** again at any time to force a rescan for available RealView ICE units and update the list.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

The devices found are listed in the RealView ICE browser on the right of the dialog box. Select the unit you want to connect to and click **Connect**. Alternatively, do one of the following:

- double-click on the unit you want to connect to
- enter either the IP address or host name of the device you want to connect to in the IP Address/Host Name field and click **Connect**.

If you have problems connecting to a RealView ICE unit, see *Troubleshooting RealView ICE connections* on page 7-6.

7.2.2 Identifying your RealView ICE unit

If you want to be certain that you are connecting to the correct run control unit, select an entry in the list, and click **Identify**:

- If the four LEDs JTAG, STAT, CFAC and LVDS on your interface (shown in Figure 7-3) flash for 5 seconds, you have selected its entry.

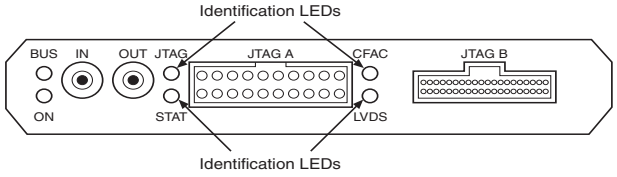


Figure 7-3 The identification LEDs

- Otherwise, select another entry and try again.

Note

For TCP/IP connections, devices shown in light gray are those that have responded to browse requests but do not have a valid IP address. You cannot connect to these devices until you have configured the IP address. See Chapter 3 *Configuring RealView ICE Networking* for information on how to do this.

Alternatively, connect using a USB cable.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.2.3 Viewing the installed components

When a connection has been established, the RVI Update application displays the components that are currently installed, as shown in Figure 7-4.

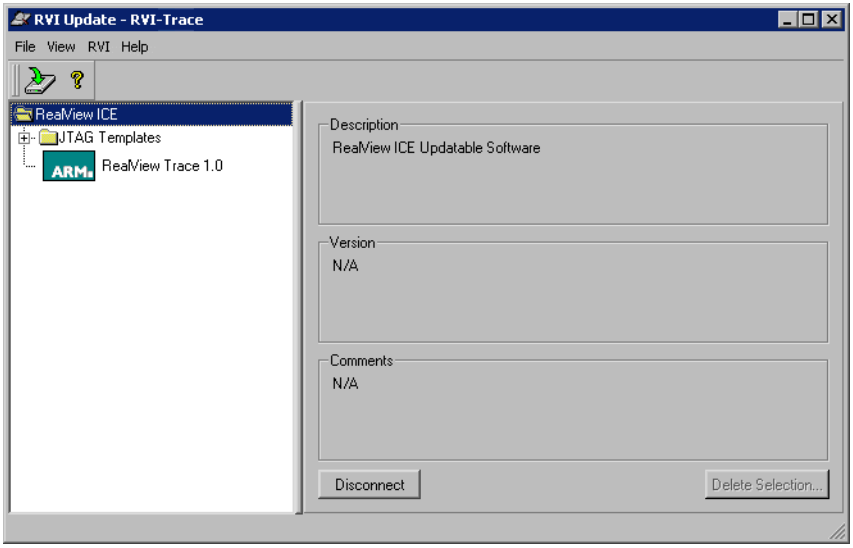


Figure 7-4 RVI Update application showing installed components

7.2.4 RVI Update tasks

When you are connected to a RealView ICE unit, you can use RVI Update to perform the tasks described in the following sections:

- *Viewing software version numbers* on page 7-8
- *Installing an update or patch* on page 7-10
- *Deleting a component* on page 7-15.

7.2.5 Disconnecting from the RealView ICE unit

When you have finished managing the RealView ICE software, click **Disconnect** in the RVI Update application to disconnect from the RealView ICE unit. You can then connect to and manage the software on another RealView ICE run control unit if required.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.2.6 Exiting the RVI Update application

When you have finished managing the software on all RealView ICE run control units, select **File** → **Exit** in the RVI Update application. This disconnects from any connected unit, and then exits the RVI Update application.

7.2.7 Troubleshooting RealView ICE connections

This section gives details of problems you might encounter when attempting to connect to a RealView ICE unit, and what you can do to solve them:

Multiple programs attempting to scan

Only one program can scan the TCP/IP network for available RealView ICE units. If another program is scanning the network, the RVI Update application displays the error message shown in Figure 7-5.

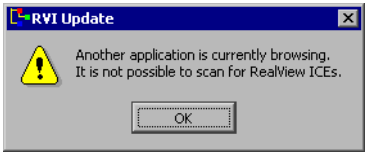


Figure 7-5 Error message when another program is browsing

You must stop one of the programs from scanning the network. To do this, click the **Scan** tool.

USB server not accessible

If the USB server is not accessible, the error message shown in Figure 7-6 appears:

Figure 7-6 Error message when no USB devices present

This indicates a problem with your RealView ICE installation. Click **OK**. If you do not want to connect to any devices over a USB connection, you can continue using RealView ICE over only TCP/IP connections. If you want to connect to a device using USB, you must reinstall RealView ICE. If the error persists, there might be a problem with your operating system.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Timeouts

The default timeout for establishing a TCP/IP connection is 5 seconds. If you repeatedly get timeouts when attempting to connect to a RealView ICE run control unit, you can change this setting. To do this:

1. If the environment variable `RVI_COMMS_CONNECT_TIMEOUT` does not already exist, then create it.
2. Set the value of this variable to the timeout that you want, in seconds. This must be an integer in the range 0-120.

For details of how to create and set an environment variable, see the documentation for the operating system that is supplied with your host computer.

Other active connections

If you connect to a RealView ICE run control unit that has other active connections, the RVI Update application displays the error message shown in Figure 7-7.

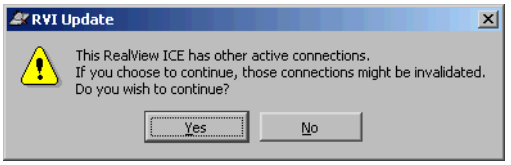


Figure 7-7 Error when other connections are active

If you continue, the changes that you make might interfere with the correct operation of these applications. Do one of the following:

- ensure that the other applications are disconnected, and then click **Yes** to continue using the RVI Update application
- click **No** to stop using the RVI Update application, and try again later.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.3 Viewing software version numbers

To view software version numbers, select **RVI → Version Info** in the RVI Update application. The RVI Update dialog box displays a window giving version information. An example is shown in Figure 7-8.

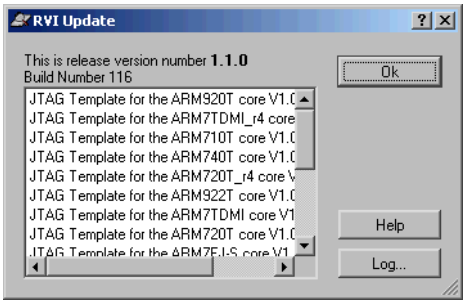


Figure 7-8 Version information

7.3.1 Version information dialog box components

In this dialog box:

- The text above the scrolling list shows the version number of the software release that is installed, in the format:

  `This is release version number` *`major.minor.patch`*

  where:

  `major` is the major release version number

  `minor` is the minor release version number

  `patch` is the patch level of the *`major.minor`* version.

- The scrolling list shows the version number of each component of the installed software.

- The **Log...** button enables you to save the version information (see *Saving the version information to a file*).

7.3.2 Saving the version information to a file

To save the information in the RVI Update dialog box to a file:

1. Click **Log**. The Select Log File Name dialog box is displayed.
2. Choose the location of the log file.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3. Click **Save** to save the log file.
4. Click **OK** to close the RVI Update version information dialog box.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.4 Installing an update or patch

ARM® Limited periodically releases updates and patches to the software that is installed on a RealView ICE run control unit. Each update or patch is released as a component file with the suffix `.rvi`. These might extend the capabilities of RealView ICE, or might fix an issue that has become apparent. You can obtain these files from the ARM website at `http://www.arm.com`.

If you want to restore the RealView ICE firmware to its original state after installing an upgrade, you can reinstall the original component file. You can obtain this from the ARM website, and you can also find it on the installation CD, located in the directory `RVIfirmware`.

This section includes:

- *Procedure for installing an update or patch*
- *Troubleshooting firmware upgrade installations* on page 7-13.

7.4.1 Procedure for installing an update or patch

To install an update or patch to the software on a RealView ICE run control unit:

1. In the RVI Update application, select **RVI → Install Component File**, or click the **Install Component File** tool. The Select Component File to Install dialog box is displayed, as shown in Figure 7-9.

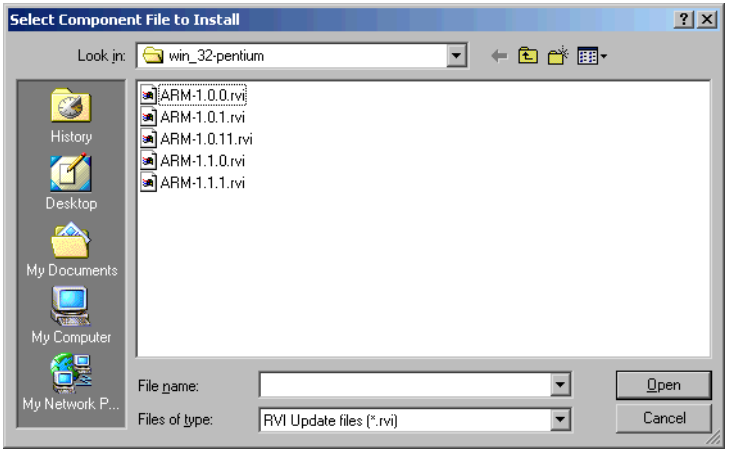


Figure 7-9 Selecting the component file to install

2. In this dialog box, navigate to the directory containing the component file for the update or patch that you want to install, and select the required file.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

3. Click **Open**. After a short delay, a dialog box appears that describes what is in the component file, as shown in Figure 7-10.

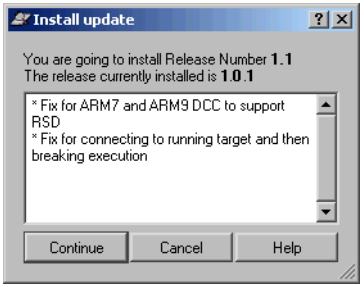


Figure 7-10 Confirming that you want to install the component file

If you are using an older version of RVI Update (for example, a pre-v1.5 release) when attempting to install an update file, an error message will appear as shown in Figure 7-11. Before proceeding with your intended installation, you should upgrade your RVI Update application to the latest software.

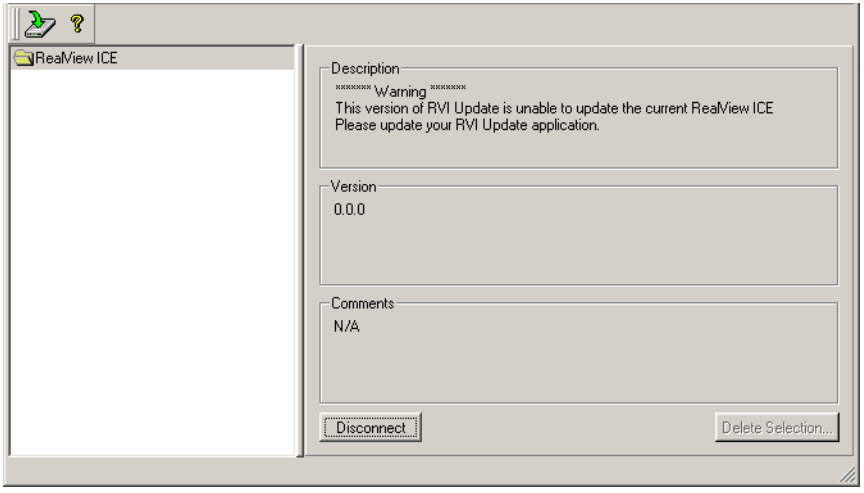


Figure 7-11 Error message

Similarly, if you are using a version of hardware that is incompatible with the firmware you are attempting to install, an error message similar to the one shown in Figure 7-12 on page 7-12 will appear.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Figure 7-12 Error when using an incompatible version of hardware

Note

For further information on hardware, see Appendix E *Hardware Variants*.

4. In the Install update dialog box, click **Continue** to confirm that you want to install the components, or **Cancel** to make no change to the run control unit. The RVI Update application then uploads the component file to the run control unit. The run control unit unpacks the component file, and installs the update or patch that it contains. The dialog box shown in Figure 7-13 appears, showing the progress of the installation.

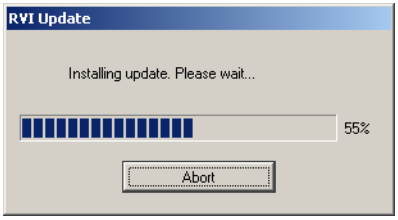


Figure 7-13 Progress during an installation

The run control unit might automatically reboot itself as part of this procedure, depending on the patch or update that you are installing. The dialog box shown in Figure 7-14 appears, showing the progress of the reboot.

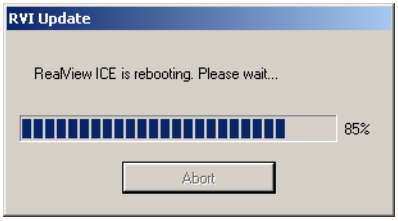


Figure 7-14 Progress when rebooting during an installation

During the installation process, the CFAC LED will light up, showing that CFAC activity is taking place. During this time, do not disconnect power from the run control unit. If a problem occurs during the installation, see *Troubleshooting firmware upgrade installations* on page 7-13.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Note

While an installation is taking place (see Figure 7-13 on page 7-12), the **Abort** button is enabled. This means that you can safely stop the installation from proceeding by clicking this button. If the **Abort** button is not enabled, for example during rebooting (see Figure 7-14 on page 7-12), you cannot stop the reboot.

When the installation is complete, a message appears to inform you of this, as shown in Figure 7-15.

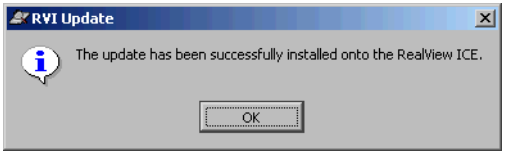


Figure 7-15 Message showing a successful installation

7.4.2 Troubleshooting firmware upgrade installations

There are two main types of error that might occur during installation of a patch:

- *Version problems*
- *Errors during file operation on the host* on page 7-14.

Version problems

A patch targets a particular *major.minor* release version of the software. It might contain:

- new components that are not in the targeted software
- updates to components that are already in the targeted software.

If there is a problem installing a patch, a dialog box appears to inform you of the problem:

- If the patch targets a version of the software that is not installed, the dialog box shown in Figure 7-16 appears. In this case the patch is not installed, and the software on the run control unit remains unchanged. Make sure that you have the patch for the version of the firmware that you have installed (see *Viewing software version numbers* on page 7-8).

Figure 7-16 Error when installing a patch to uninstalled software

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- If the patch does not contain any new or updated components (typically because a later patch has already been installed), the dialog box shown in Figure 7-17 appears.

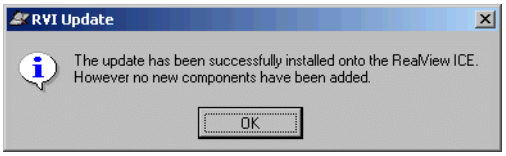


Figure 7-17 Message when installing a patch that has no new components

If you see one of these dialog boxes, click **OK**.

Errors during file operation on the host

If an error occurs during file operation on the host, a dialog box appears to inform you of the problem:

- If the error occurs before any data has been written to the compact flash, the dialog box shown in Figure 7-18 appears.

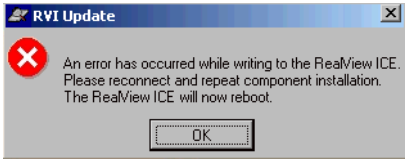


Figure 7-18 Error before data has been written to compact flash

  Click **OK** and begin the installation again.

- If some data has already been written to the compact flash when the error occurs, the RealView ICE run control unit must reboot to clean up the failed installation and revert to the backed up state. The dialog box shown in Figure 7-19 appears.

Figure 7-19 Error during writing to compact flash

  Click **OK**. The RealView ICE run control unit reboots. You can then begin the installation again.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.5 Deleting a component

To permanently delete some or all versions of a component from the run control unit:

1. In the RVI Update application, expand the tree diagram until you see the version of the component that you want to delete. For an example, see Figure 7-20.

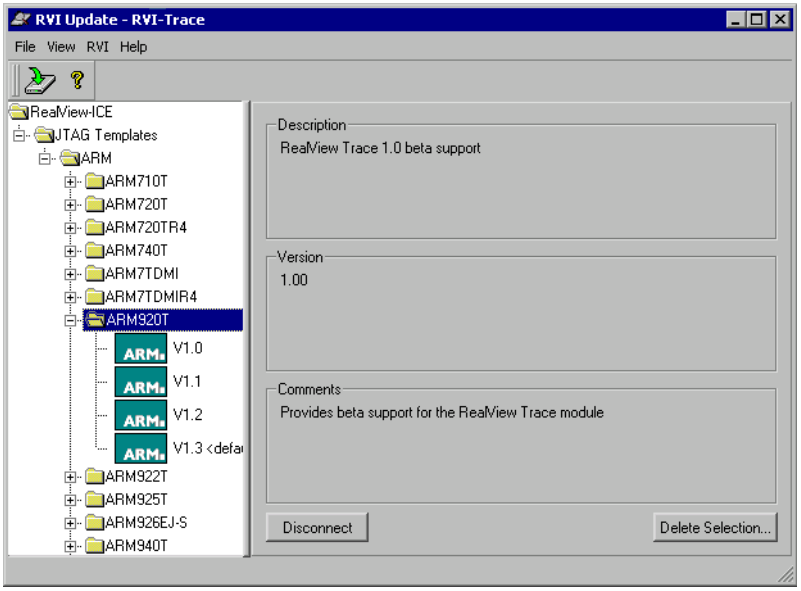


Figure 7-20 Tree showing the different versions of a component

2. Select the version of the component that you want to delete by clicking on it.

3. Click **Delete Selection**. A dialog box appears warning you that you cannot undo this action, as shown in Figure 7-21.

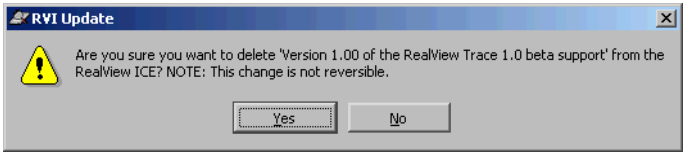


Figure 7-21 Warning that you cannot undo a deletion

4. Click **Yes**. The selected version of the component is deleted, and is no longer available, whatever default set you are using.

5. Repeat this process for any other versions of components that you want to delete.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

7.6 Restarting the RealView ICE run control unit

To restart the RealView ICE run control unit, select **RVI → Restart**. RealView ICE Update reboots the run control unit, waits for the reboot to finish, then reconnects automatically. A message is displayed telling you that RealView ICE is rebooting.

If you do not want to reconnect, click **Disconnect**. The browser pane is displayed and you can connect to a different RealView ICE unit if required.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 8
Configuring RealView ICE for GDB

This chapter describes the basic steps required to configure the RealView® ICE unit to a state where you can begin debugging your image using the *GNU Debugger* (GDB). It includes:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.1 About using RealView ICE for debugging with GDB

RealView ICE provides functionality that extends the debugging features available in GDB. This section includes:

- *Features supported when debugging with GDB*
- *Features not supported when debugging with GDB* on page 8-3
- *RealView ICE TCP/IP ports used* on page 8-3
- *Building for standalone target platforms* on page 8-3.

Note

To find the latest information on GNU Debugger (GDB) compatibility with RealView ICE v3.0, refer to the ARM RealView ICE v3.0 Release Notes.

For information on GDB and Command Monitor error codes, refer to the ARM website.

8.1.1 Features supported when debugging with GDB

RealView ICE supports connections from remote GDB sessions over TCP/IP. These GDB connections support all non-OS specific functionality, including:

- full memory and register access
- run and stop
- software and hardware breakpoints and watchpoints
- target reset (restart)
- binary program downloading
- step-over-range
- single stepping
- extended mode.

Debugging modes

You can use the following debugging modes with GDB:

- Halt-mode debugging, where the target stops while you examine it.
- Monitor-mode debugging, where the target continually runs and you monitor the target through processed DCC mode connections using DCC (see *DCC modes* on page 8-16).

See *Methods of connecting from remote GDB sessions* on page 8-5 for details of the connection methods you can use with these modes.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.1.2 Features not supported when debugging with GDB

When using GDB, RealView ICE does not provide support for:

- threads (in start-stop debugging)
- debugging over the RealView ICE USB port
- synchronized start/step on multi-core systems.

8.1.3 RealView ICE TCP/IP ports used

To use the RealView ICE Command Monitor and debug your target with GDB, RealView ICE uses the TCP/IP ports described in Table 8-1.

Table 8-1 RealView ICE TCP/IP ports

Port	Description
4000 series	The port range used (in a one-up series) to connect to a target from GDB, and to perform halt-mode debugging (see *Halt-mode debugging* on page 8-5).
5000 series	The port range used (in a one-up series) for Monitor-mode debugging and other processed DCC mode connections (see *Monitor-mode debugging* on page 8-6, and *How connections to multiple processors are allocated* on page 8-21). **Note** To use these ports you must set the DCC mode to a non-zero value (see *DCC modes* on page 8-16).

8.1.4 Building for standalone target platforms

If you are building for a standalone target platform (that is, without an operating system), the precompiled C library of the GNU toolchain for ARM architectures assumes that a debug monitor is resident in ROM.

If you are not using a debug monitor, Red Hat eCos/Redboot, or any other operating system, you must provide the following components:

- Your own I/O routines and optionally a target GDB stub.
- The `crt0.S` source file, which is mandatory. This source file provides the C startup procedure that is responsible for setting up the stack and heap, and for initializing C static and global variables.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Note

If you are using a debug monitor, Red Hat eCos/Redboot or other operating system, you must provide at least these components and possibly a gdbserver.

Documentation on how to do this is readily available from the Internet.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.2 Methods of connecting from remote GDB sessions

The method you use to connect to a target depends on:

- the resources required by the target application (for example, an IP stack)
- the debugging facilities available on the target (for example, a GDB stub)
- whether or not your target has an embedded OS, such as Linux, that is running gdbserver instances.

This section includes:

8.2.1 Summary of the connection methods for each debugging mode

How you connect to a target determines the debugging mode. The following sections describe the connection methods for each debugging mode.

Halt-mode debugging

Halt-mode debugging is the simplest method of debugging a target with GDB. You directly connect to RealView ICE, which then controls the starting and stopping of the processor. This method of connecting is subsequently referred to as an RVI-GDB connection.

See *Connections to a target without built-in GDB support (RVI-GDB)* on page 8-7 for more details.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Monitor-mode debugging

Monitor-mode debugging requires that your target application communicate with GDB using the *Debug Communications Channel* (DCC) of an ARM architecture-based processor. However, if your target application includes an Ethernet facility, you do not have to use DCC. Different DCC modes are available depending on the requirements of your target (see *DCC modes* on page 8-16 for more details).

The connection methods for Monitor-mode debugging are:

Target-GDB connections

Semi-transparent connections to GDB stubs. The GDB stubs run on individual CPU targets and communicate through the DCC of the target. The GDB stub must be compiled into the target application. See *Connections to a target with a GDB stub (Target-GDB)* on page 8-9 for more details.

Target-GDB-Virtual Ethernet connections

An extension to Target-GDB connections for standalone applications running an IP stack. The TCP/IP connections are bridged by the RealView ICE unit from the target through DCC to the Ethernet port of the RealView ICE unit. See *Connections to a target GDB stub using processed DCC mode (Target-GDB-Virtual Ethernet)* on page 8-11 for more details.

GDBserver connections

An alternative to Target-GDB-Virtual Ethernet connections where the target is running gdbserver running under an operating system (OS). See *Connections to a target OS using gdbserver (GDBserver)* on page 8-12 for more details.

GDB-NFS connections

Connections to the root filesystem on the target OS that is mounted over NFS. The RealView ICE unit acts as a bridge between the debug host and the target OS. See *Connections to a target OS using NFS (GDB-NFS)* on page 8-13 for more details.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.2.2 Connections to a target without built-in GDB support (RVI-GDB)

These are connections to targets where no GDB stub has been built into the target application, or when you want to perform halt-mode debugging. Connections of this type use the built-in GDB protocol interpreter of RealView ICE to control the CPU directly, and are referred to as RVI-GDB connections. When you want to examine the internal state of the CPU (such as registers, memory, and variables), the image on the target stops executing. After examining the required state, you must start the image again. Figure 8-1 shows the configuration.

Note

Because this method does not use the DCC semihosting mechanism, any prompts and messages that are output by the application cannot be displayed.

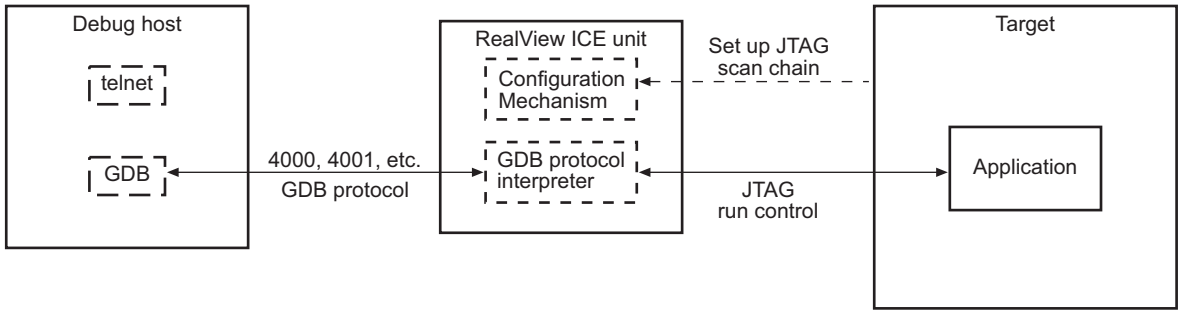


Figure 8-1 RVI-GDB connections

RVI-GDB Scenarios

Use the RVI-GDB connection method to:

- perform run and stop debugging of a single ARM processor
- perform run and stop debugging with GDB at the same time as debugging the application using the methods described in *Connections to a target with a GDB stub (Target-GDB)* on page 8-9 and *Connections to a target GDB stub using processed DCC mode (Target-GDB-Virtual Ethernet)* on page 8-11.

Note

When the image stops, so does the handling of interrupt routines, which might not always be desirable when debugging a real-time system.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

RVI-GDB Requirements

To use the RVI-GDB connection method, it is recommended that you compile your target application using a GNU toolchain for ARM architectures (see *The GNU toolchain for ARM architectures* on page 1-12).

Procedure for debugging applications through RVI-GDB connections

If your application does not have GDB support linked-in, you can use the GDB protocol built into the RealView ICE unit to debug your application. However, this controls the CPU directly, and the CPU stops whenever you want to examine its internal state.

To debug an application through an RVI-GDB connection:

1. Power up your target hardware and RealView ICE unit.
2. Configure the core using rvconfig, using either automatic or manual configuration. Save the rvc file in a convenient location.
3. Run rvigdbconfig, specifying the rvc file that was created above:
   `rvigdbconfig -f rvi.rvc`
4. Start GDB, load the symbols if required, and connect to the first core (using port 4000 of RealView ICE in this example):

   ```
   arm-elf-gdb
   (gdb) file demo.elf
   (gdb) target remote rvi5:4000
   Remote debugging using rvi5:4000
   0x00000000 in $a ()
   (gdb)
   ```

   GDB is now connected to the core, and an image can be loaded and debugged.

   For information on connecting to more than one core, see *Multicore debugging* on page 8-15.

   Note

   To load and boot a complete system, use the `rviload` utility (see *Loading and booting a complete system* on page 8-19).
5. Set up any breakpoints or other debugging features, then run the application. Debug your application in the usual way.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.2.3 Connections to a target with a GDB stub (Target-GDB)

These are connections to a target that is running an application with a GDB stub, and are referred to as Target-GDB connections. The GDB stub enables the target application to communicate with a host application through RealView ICE, using the DCC of an ARM architecture-based processor. The DCC carries the GDB protocol packets between the target and the remote GDB session over the TCP/IP ports 5000, 5001,... as shown in Figure 8-2. See also *RealView ICE TCP/IP ports used* on page 8-3.

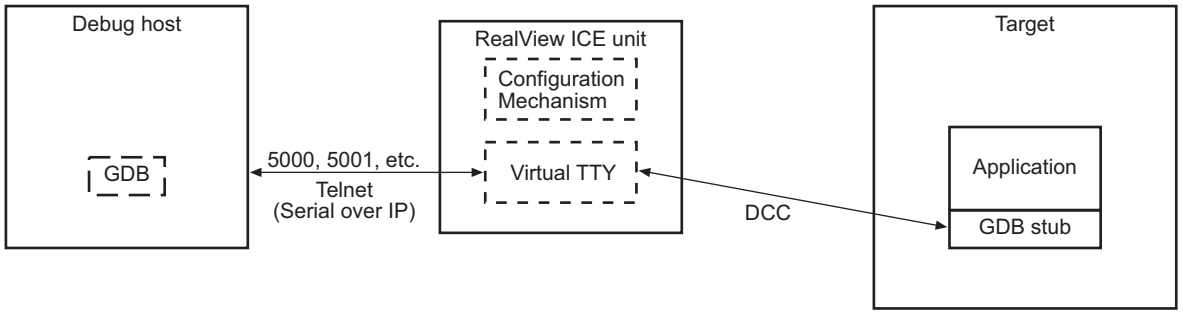


Figure 8-2 Target-GDB connections

Target-GDB Scenarios

Use the Target-GDB connection method to:

- debug a target system that does not have an OS
- debug a target system with an OS that supports GDB.

Target-GDB Requirements

To use the Target-GDB connection method, it is recommended that you compile the DCC driver and GDB stub into your target application using a GNU toolchain for ARM architectures (see *The GNU toolchain for ARM architectures* on page 1-12). You can either:

- link the example GDB stub into your target application or operating system
- port your existing serial GDB stub to use the DCC driver.

Note

On the GDB connection to the target, it is recommended that you enable DCC and processed DCC mode before starting the processor (see *Setting DCC parameters* on page 8-17).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Procedure for debugging applications through Target-GDB connections

If your application includes a target-resident GDB stub, it may communicate over DCC.

To debug an application using a Target-GDB connection:

1. Power up your target hardware and RealView ICE unit.
2. Configure the core using rvconfig, using either automatic or manual configuration. Save the rvc file in a convenient location.
3. Run rvigdbconfig, specifying the rvc file that was created above, and the appropriate DCC mode, for example mode 2:
   rvigdbconfig –f rvi.rvc -d 1:2
4. Start GDB, load the symbols if required, and connect to the second core (for example) to load and run your application in the usual way. This example uses port 4001 of RealView ICE:

```
arm-elf-gdb
(gdb) file demo.elf
(gdb) target remote rvi5:4001
Remote debugging using rvi5:4001
0x00000000 in $a ()
(gdb)
(gdb) load demo.elf
Loading section .vectors, size 0x30 lma 0x0
Loading section .text, size 0x1dbcc lma 0x8000
Loading section .rodata, size 0x1bcb4 lma 0x25bcc
Loading section .data, size 0xc84 lma 0x41980
Start address 0x8000, load size 238900
Transfer rate: 106177 bits/sec, 318 bytes/write.
(gdb)
(gdb) c
Continuing.
```

5. Start another GDB session to debug the image in the usual way, using (in this example) port 5001, the first available port of RealView ICE:

```
(gdb) set remotetimeout 10
(gdb) file myprogram
(gdb) target remote rvi5:5001
```

   See *How connections to multiple processors are allocated* on page 8-21.

Note

You only have to perform steps 1 to 3 once at the start. You can perform steps 4 and 5 as often as required.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.2.4 Connections to a target GDB stub using processed DCC mode (Target-GDB-Virtual Ethernet)

If your target application requires TCP/IP communication with the debug host, you can connect to the target using processed DCC mode. Connections of this type are referred to as Target-GDB-Virtual Ethernet connections. This method is an extension to that described in *Connections to a target with a GDB stub (Target-GDB)* on page 8-9, and is as shown in Figure 8-3. In this method, RVI provides a network bridging facility to targets, and allows a target with only a JTAG connection to RVI to have access to the same network resources available to RVI. This works by intercepting IP packets on the network and examining them, and those packets that are addressed to the target are then sent over DCC alongside the normal GDB protocol. A driver is required on the target to interface the DCC channel to the target's protocol stack, making the bridged network connection appear as an ethernet device on the target. IP is the only network layer protocol supported.

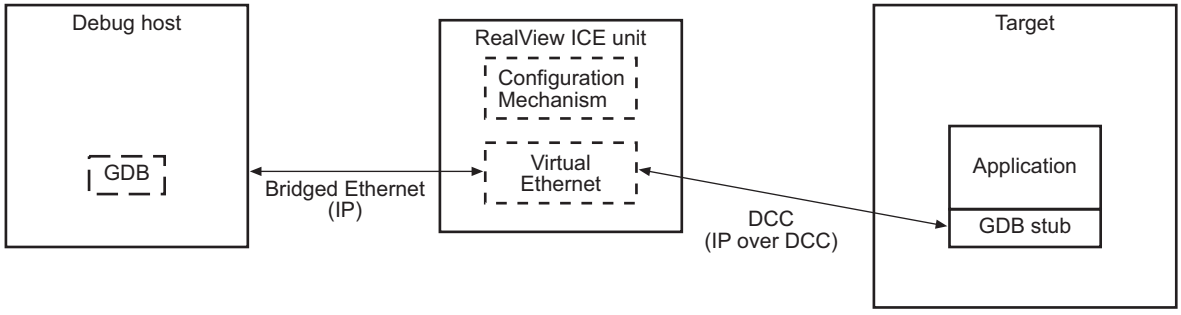


Figure 8-3 Target-GDB-Virtual Ethernet connections

Note

In order to reduce the load on the DCC and JTAG connection, broadcast packets are not sent to the target.

Target-GDB-Virtual Ethernet Scenario

Use the Target-GDB-Virtual Ethernet connection method to communicate with a standalone application that has a TCP/IP stack. For example, an application might provide a web server that serves web pages to the host.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Target-GDB-Virtual Ethernet Requirements

To use the Target-GDB-Virtual Ethernet connection method:

- It is recommended that you compile the DCC driver and GDB stub into your target application using a GNU toolchain for ARM architectures (see *The GNU toolchain for ARM architectures* on page 1-12).

  Note

  On the GDB connection to the target, you must enable DCC and processed DCC mode before starting the processor (see *Setting DCC parameters* on page 8-17).

- The target application must be running an TCP/IP stack.
- RealView ICE acts as a network bridge between the target processor and the host PC using a virtual Ethernet link. The target must have its own IP address that is either fixed or obtained from a DHCP server, and that appears on the virtual Ethernet as an independent host.

8.2.5 Connections to a target OS using gdbserver (GDBserver)

If your target application requires TCP/IP communication with the debug host, you must connect to the target using bridged Ethernet. Connections of this type are referred to as GDBserver connections. This method is an extension to that described in *Connections to a target with a GDB stub (Target-GDB)* on page 8-9, shown in Figure 8-4. In this method, IP packets can be carried over the same link alongside the normal GDB protocol.

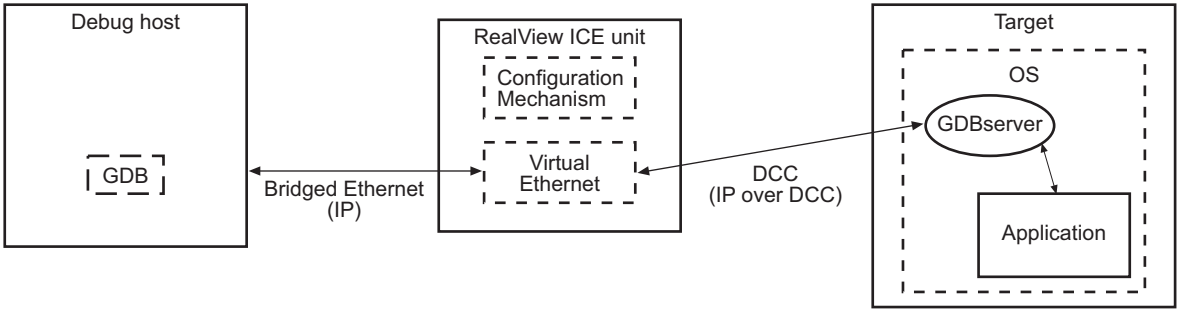


Figure 8-4 GDBserver connections

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

GDBserver Scenario

Use the GDBserver connection method to debug an application on a target that has an embedded OS, such as Linux, that supports independent processes. In this case you can run GDB server (gdbserver) instances. The GDB server can have TCP/IP connections to the debug host that is running GDB.

GDBserver Requirements

To use the GDBserver connection method:

- On the GDB connection to the target, you must enable DCC and processed DCC mode before starting the processor (see *Setting DCC parameters* on page 8-17).
- The target OS must be running an TCP/IP stack and gdbserver.
- RealView ICE acts as a network bridge between the target processor and the host PC using a virtual Ethernet link. The target must have its own IP address that is either fixed or obtained from a DHCP server, and that appears on the virtual Ethernet as an independent host.

Procedure for debugging an application using a GDBserver connection

If your application uses an IP stack, it may communicate over DCC through a bridged Ethernet connection.

To debug an application using a GDBserver connection:

1. Power up your target hardware and RealView ICE unit.
2. Download and boot the target using the `rviload` utility (see *Loading and booting a complete system* on page 8-19).
3. When the Linux kernel has finished booting, start the gdbserver as follows:

   ```
   ~ # gdbserver localhost:portnum filename
   ```

   The TCP/IP port number you specify here is the port number you must use with the GDB `target remote` command from subsequent GDB sessions. Also, make sure the port number is not in use by another service.

8.2.6 Connections to a target OS using NFS (GDB-NFS)

This is useful for developing software on deeply embedded systems, and also for debugging brand new targets where only the CPU and RAM are initially known to work. Connections of this type are referred to as GDB-NFS connections. The GDB-NFS connection method is shown in Figure 8-5 on page 8-14.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

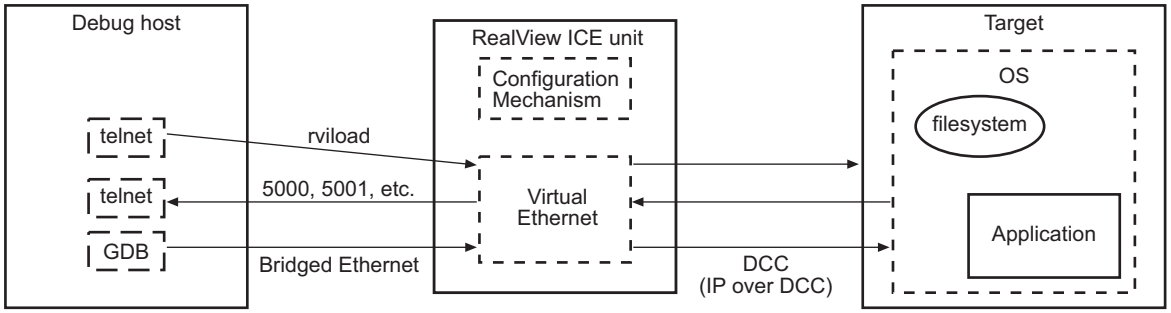


Figure 8-5 GDB-NFS connections

For example, the minimum I/O support that Linux requires is a system console and a root file system. You can connect the console to a GDB command-line console and, with the appropriate driver, you can mount the root file system over NFS with RealView ICE acting as a bridge. Alternatively, you might have a file system and kernel loaded into memory and booted, and an NFS file system mounted to be shared later (see *Loading and booting a complete system* on page 8-19).

Note

This network connection is not as fast as an office LAN, because of the limited bandwidth of DCC.

Procedure for debugging a target using a GDB-NFS connection

If your target has a complete operating system, you can mount a file system over NFS. In this case, RealView ICE acts as a bridge.

To debug a target using a GDB-NFS connection:

1. Power up your target hardware and RealView ICE unit.
2. Load the Linux kernel and uBoot image using rviload. For example:

   ```
   RVIv14 Utils> ./rviload --host=rvi5 -j1000000 -mVEC -a0 7fc0:uImage
   1000000:u-boot.bin
   ```

   Press Return. Messages appear showing u-boot loading and running the Linux kernel.
3. Mount the directory exported by NFS using the following command:

   ```
   ~ # mount -t nfs -n client_IP_address:/exported_directory /mnt
   ```

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.3 Preparing RealView ICE for remote GDB connections

This section describes how to prepare RealView ICE to accept remote GDB connections and be able to communicate with a GDB session. It includes:

- *About connecting to targets through RealView ICE*
- *DCC modes* on page 8-16.

8.3.1 About connecting to targets through RealView ICE

To connect to a target from GDB, you must configure RealView ICE to recognize your target devices. You do this by using rvigdbconfig to configure RVI with the scan chain details. GDB is then connected to the core.

The procedure is:

1. Power up your target hardware and RealView ICE unit.
2. Launch RVConfig, and configure the core by following the steps described in *Opening the RVConfig dialog box — standalone method* on page 4-4.
3. Run rvigdbconfig, specifying the appropriate `rvc` file:

   ```
   rvigdbconfig -f rvi.rvc
   ```

4. Start GDB, load the symbols if required, and connect to the first core (using port 4000 of RealView ICE in this example):

   ```
   arm-elf-gdb
   (gdb) file demo.elf
   (gdb) target remote rvi5:4000
   Remote debugging using rvi5:4000
   0x00000000 in $a ()
   (gdb)
   ```

   GDB is now connected to the core, and an image can be loaded and debugged.

8.3.2 Multicore debugging

In instances of multicore debugging, each device on the scan chain is allocated a GDB port to connect to the core. Ports are allocated in sequence, starting from port 4000—as shown, for example, in step 4 of *About connecting to targets through RealView ICE*. Port 4000 is connected to the first device in the scan chain, port 4001 to the second device, and so on.

Synchronized start/step are not supported.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.3.3 DCC modes

If your target application communicates using DCC, you must configure the DCC mode. The DCC modes you can set are:

Mode 1: raw DCC

In this mode, the data is fed over TCP/IP ports 5000, 5001,... . Data is sent from the host to the target 4 bytes at a time. If fewer than 4 bytes are available, the data is padded with 0 bytes until it is 4 bytes long. Data from the target to the host is received 4 bytes at a time, and no padding or trimming is performed.

Note

It is recommended that you use the processed DCC mode (mode 2). If the processed DCC mode is unsuitable for your application, then use mode 1 DCC. However, you must also implement a suitable communications protocol.

Mode 2: processed DCC mode

Processed DCC is used for providing virtual serial and Ethernet connections to the target.

Processed DCC mode

Processed DCC mode is a RealView ICE facility that provides (as far as the debug host and the target are concerned):

- A virtual Ethernet facility using the DCC channel and a collection of software tools in RealView ICE and the host PC. It allows TCP/IP to be used to the target as though the target has an Ethernet port of its own.
- A virtual serial port facility using the DCC channel and a collection of software tools in RealView ICE and the host PC.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Setting DCC parameters

Ethernet bridging works by examining incoming packets at RVI, then deciding which are destined for RVI itself and which are destined for the target. To do this, RVI must know the IP address, subnet mask and default gateway parameters for the target. These parameters are normally determined via DHCP, where the target asks for a configuration, and one is supplied by a server over the network. In this case, RVI is able to intercept the incoming DHCP packet containing the parameters and configure itself appropriately. It is, however, possible to configure a target with a static IP address. In this case there is no DHCP transaction to intercept, and RVI has no way of determining the target configuration. You must set these parameters in RVI for correct operation.

DCC ethernet bridging is configured using rvigdbconfig, and you must set the appropriate parameter when using DCC mode. For example:

```
rvigdbconfig -f rvi.rvc -d 0:2
```

will set device 0 (the first device on the scan chain) to DCC mode 2.

Additional devices are configured in a similar way. For example:

```
rvigdbconfig -f rvi.rvc -d 0:2 -d 1:2
```

will configure devices 0 and 1 to mode 2.

The IP parameters for static IP configurations are set up in the following way:

```
rvigdbconfig -f rvi.rvc -d 0:2 -s 0:10.0.0.10:255.255.255.0:10.0.0.1
```

This will configure device 0 to use DCC mode 2 with IP address `10.0.0.10`, subnet mask `255.255.255.0`, and default gateway `10.0.0.1`. This format can be used to configure multiple cores if required. For example:

```
rvigdbconfig -f rvi.rvc -d 0:2 -d 1:2 -s 0:10.0.0.10:255.255.255.0: 10.0.0.1 -s
1:10.0.0.11:255.255.255.0:10.0.0.1
```

For further details on communicating over DCC, see *Procedure for debugging applications through Target-GDB connections* on page 8-10.

If using rviload, you should set VEC or VEP. (See the allowed `MODE` values under *rviload command syntax* on page 8-19.)

Note

Once ethernet bridging is running, normal LAN services are accessible (including DHCP and NFS).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

DCC and interrupts

The use of DCC interrupts has significant speed implications when using processed DCC mode. If possible, you should tie DCC interrupts into the interrupt system of the target and be able to enable and disable the read and write interrupt individually.

RealView ICE uses JTAG to control debug operations, and JTAG is used to send and receive data over DCC. RealView ICE polls the target JTAG for status:

- If interrupts are used, the target is interrupted when data is written to the DCC register or read from it. This allows the target to deal quickly with the data, and continue normal processing.
- If interrupts are not available, the target must regularly poll the DCC register for any new data, which means that the target wastes time checking the register for data when none is present. Subsequent data will be discovered only at the next poll.

If RealView ICE finds that there is data to be transferred into or out of DCC, it attempts to transfer as many words as possible in one burst, up to a predefined limit. However, if the target has not sent more data or emptied its transfer register, RealView ICE breaks out of its burst and begins polling the execution status and DCC.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.4 Loading and booting a complete system

You can load and boot a complete system without having to run GDB, RealView Debugger, or any other supported debugger to control the RealView ICE unit. To do this, you use the `rviload` utility. This section includes:

- *Requirements for using the rviload utility*
- *rviload command syntax*
- *Using the rviload utility from a Cygwin bash or Red Hat Linux shell* on page 8-20

8.4.1 Requirements for using the rviload utility

Before you can use the `rviload` utility, there must be no other active connections to the target scan chain device from RealView Debugger, GDB, or any other active connection.

8.4.2 rviload command syntax

The `rviload` utility has the following command syntax:

`rviload [`*`options`*`]... `*`address`*`:`*`file`* `[`*`address`*`:`*`file`*`]...`

The options are:

`-c, --check` Check memory as it is being written.

`-a, --autoconfig=DELAY`

Autoconfigure the scan chain and then wait DELAY (s).

`-s, --jtagclock`

The JTAG clock speed in Hz, 0=='RTCK' (default 10MHz).

`-d, --devnum=DEVNUM`

The JTAG scan chain device number (default 1). In rviload, device numbers start from 1 whereas in rvigdbconfig, device numbers start from 0.

`-H, --host=HOST`

The host IP address/name of the RealView ICE.

`-j, --jump=JUMPTO`

Start executing from this (hex) address after loading.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

`-m, --dccmode=MODE`

Enable debug communications in the particular `MODE`, which must be one of the following:

DCC Raw DCC via client.

DCP Raw DCC via TCP/IP port range from 5000.

VEC Processed DCC with tty channel via client.

VEP Processed DCC with tty channel via port range from 5000.

The DCC mode for rviload is specified by using a three-letter mode name such as VEP or VEC, whereas the DCC mode for rvigdbconfig is specified by a mode number such as 0, 1 or 2.

`-p, --page=PAGE`

The target memory page number.

`-q, --quiet` Do not print any messages.

`-r, --rule=RULE`

Target rule code.

`-h, --help` Display this output.

8.4.3 Using the rviload utility from a Cygwin bash or Red Hat Linux shell

To use the `rviload` utility from a Cygwin bash or Red Hat Linux shell, enter:

```
$ rviload [option]... address:file [address:file]...
```

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

8.5 Multiprocessor debugging with GDB and RealView ICE

RealView ICE is capable of simultaneously and synchronously debugging multiple targets. However, GDB does not support multiprocessor debugging directly.

Note

Although multiprocessor debugging is possible using GDB, it is recommended that you debug multiple processors using higher level tools, such as RealView Debugger.

This section includes:

- *How connections to multiple processors are allocated*
- *Considerations when debugging multiple targets with GDB*

8.5.1 How connections to multiple processors are allocated

When you make connections to multiple target processors, the connections on ports 5000, 5001,... are allocated by RealView ICE—that is, redirected virtual DCC is allocated a port for each core in a similar way to GDB halt-mode debugging. The port range starts from port 5000, so virtual Ethernet or raw redirected DCC for the first core will appear on port 5000, the second core on port 5001, and so on.

8.5.2 Considerations when debugging multiple targets with GDB

Be aware of the following if you are debugging multiple targets with GDB:

- Multiprocessor debugging with GDB requires that you open multiple command windows (such as Xterms). You must have one GDB session for each target processor to which you want to connect.
- If you have multiple targets on the RealView ICE JTAG scan chain, then:
  - the target processors are numbered consecutively, starting at one
  - the available bandwidth over DCC is shared between all target processors
  - communications to all target processors are through a single JTAG chain.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Chapter 9
System Design Guidelines

This chapter provides information on developing ARM® architecture-based devices and *Printed Circuit Boards* (PCBs) that can be debugged using RealView® ICE. It contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

9.1 About the system design guidelines

This chapter describes the following:

- How to connect multiple TAP controllers in systems comprising more than one unit. For example, connecting an ARM core plus a *Digital Signal Processor* (DSP).
- Using the RealView ICE adaptive clocking feature to control the JTAG clock rate.
- Reset signals, providing examples of circuits.
- Compatibility with EmbeddedICE™ connectors.

See Appendix A *JTAG Interface Connections* for details of the JTAG interface connector pinout. See Appendix D *Designing the Target Board for Tracing* for information on designing a board that can be connected to RealView Trace.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

9.2 System design

This section describes how to design clocking and reset circuits that are compatible with RealView ICE. It contains the following sections:

- *Using adaptive clocking to synchronize the JTAG port*
- *Reset signals* on page 9-6.

9.2.1 Using adaptive clocking to synchronize the JTAG port

ARM architecture-based devices using only hard macrocells, for example ARM7TDMI® and ARM920T, use the standard five-wire JTAG interface (**TCK**, **TMS**, **TDI**, **TDO**, and **nTRST**). Some target systems, however, require that JTAG events are synchronized to a clock in the system. To handle this case, an extra signal (**RTCK**) is included on the JTAG port. For example, this synchronization is required in:

- an ASIC with single rising-edge D-type design rules, such as one based on an ARM7TDMI-S™ processor core
- a system where scan chains external to the ARM macrocell must meet single rising-edge D-type design rules.

The adaptive clocking feature of RealView ICE addresses this requirement. When adaptive clocking is enabled, RealView ICE issues a **TCK** signal and waits for the **RTCK** signal to come back. RealView ICE does not progress to the next **TCK** until **RTCK** is received.

Note

- If you use the adaptive clocking feature, transmission delays, gate delays, and synchronization requirements result in a lower maximum clock frequency than with non-adaptive clocking. Do not use adaptive clocking unless it is required by the hardware design.
- If, when autoconfiguring a target, the RealView ICE run control unit receives pulses on **RTCK** in response to **TCK** it assumes that adaptive clocking is required, and enables adaptive clocking in the target configuration. If the hardware does not require adaptive clocking, the target is driven slower than it could be. You can disable adaptive clocking using controls on the JTAG settings dialog box.
- If adaptive clocking is used, RealView ICE cannot detect the JTAG clock speed, and therefore cannot scale its internal timeouts. If the target clock frequency is very slow, a JTAG timeout might occur. This leaves the JTAG in an unknown state, and RealView ICE cannot operate correctly without reconnecting to the core. JTAG timeouts are enabled by default. You can disable JTAG timeouts by

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

deselecting the option JTAG Timeouts Enabled in the RVConfig dialog box. See Chapter 4 *Configuring a RealView ICE Connection* for information on configuring RealView ICE.

You can use adaptive clocking as an interface to targets with slow or widely varying clock frequency, such as battery-powered equipment that varies its clock speed according to processing demand. In this system, **TCK** might be hundreds of times faster than the system clock, and the debugger loses synchronization with the target system. Adaptive clocking ensures that the JTAG port speed automatically adapts to slow system speed.

Figure 9-1 illustrates a circuit for basic applications, with a partial timing diagram shown in Figure 9-2. The delay can be reduced by clocking the flip-flops from opposite edges of the system clock, because the second flip-flop only provides better immunity to metastability problems. Even a single flip-flop synchronizer never completely misses **TCK** events, because **RTCK** is part of a feedback loop controlling **TCK**.

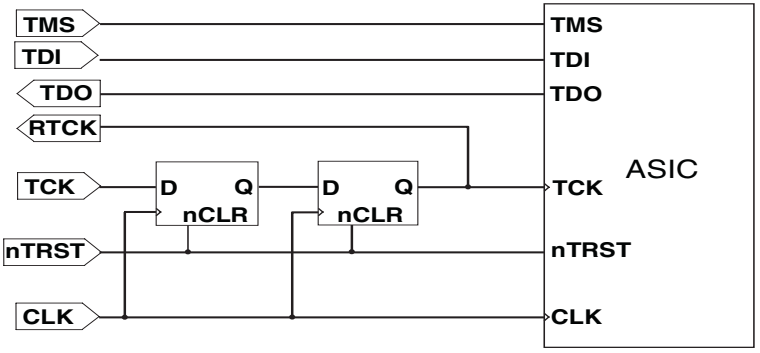


Figure 9-1 Basic JTAG port synchronizer

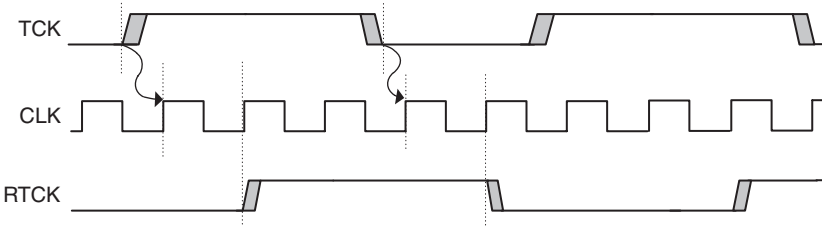


Figure 9-2 Timing diagram for the Basic JTAG synchronizer in Figure 9-1

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

It is common for an ASIC design flow and its design rules to impose a restriction that all flip-flops in a design are clocked by one edge of a single clock. To interface this to a JTAG port that is completely asynchronous to the system, it is necessary to convert the JTAG **TCK** events into clock enables for this single clock, and to ensure that the JTAG port cannot overrun this synchronization delay. Figure 9-3 shows one possible implementation of this circuit, and Figure 9-4 on page 9-6 shows a partial timing diagram, showing how **TCKFallingEn** and **TCKRisingEn** are each active for exactly one period of **CLK**. It also shows how these enable signals gate the **RTCK** and **TDO** signals so that they only change state at the edges of **TCK**.

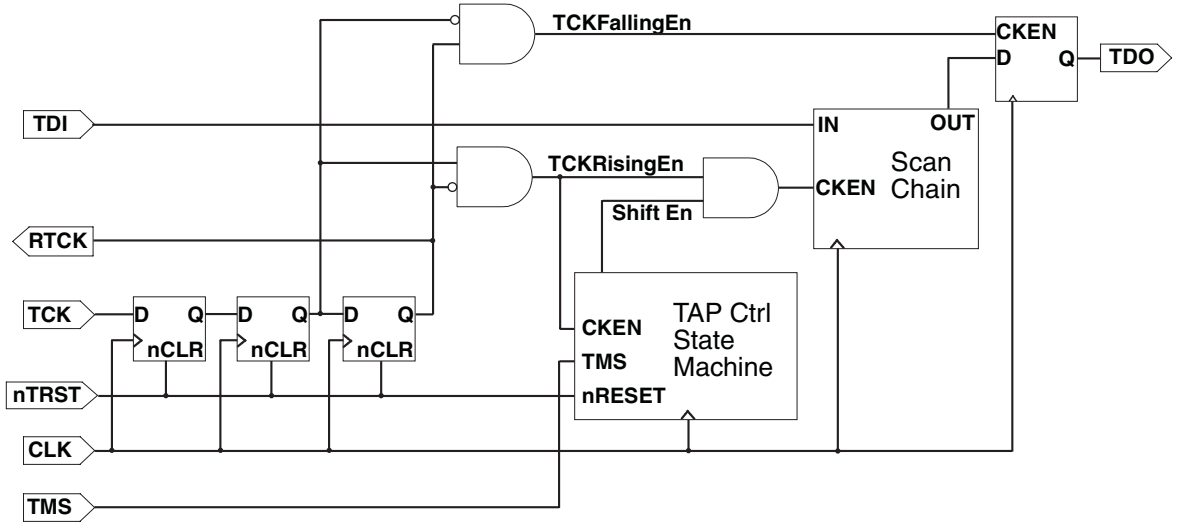


Figure 9-3 JTAG port synchronizer for single rising-edge D-type ASIC design rules

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

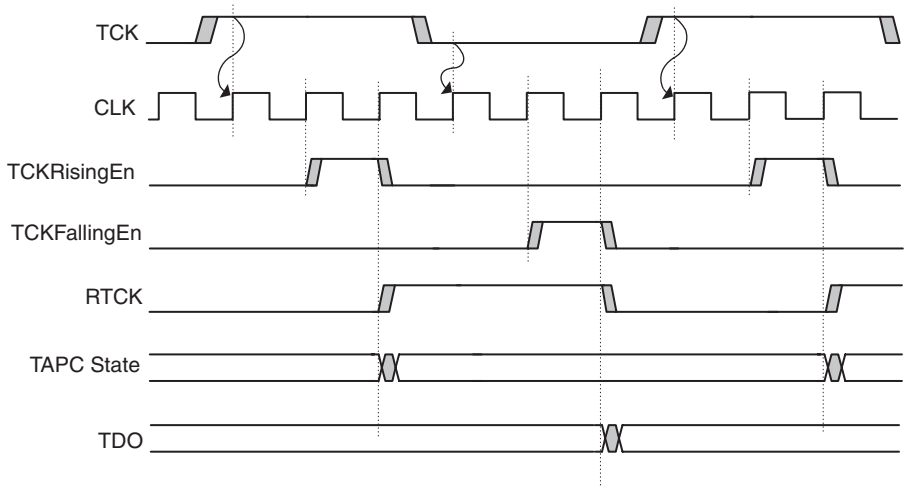


Figure 9-4 Timing diagram for the D-type JTAG synchronizer in Figure 9-3 on page 9-5

9.2.2 Reset signals

This section describes the reset signals that are available on ARM devices and how RealView ICE expects them to be wired. It is presented in the following sections:

- *ARM reset signals*
- *RealView ICE reset signals* on page 9-7
- *Example reset circuits* on page 9-7.

ARM reset signals

All ARM cores have a main processor reset that might be called **nRESET**, **BnRES**, or **HRESET**. This is asserted by one or more of these conditions:

- power on
- manual push button
- remote reset from the debugger (using RealView ICE)
- watchdog circuit (if appropriate to the application).

Any ARM processor core including the JTAG interface has a second reset input called **nTRST** (TAP Reset). This resets the EmbeddedICE logic, the TAP controller, and the boundary scan cells. It is activated by remote JTAG reset (from RealView ICE).

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

It is strongly recommended that both signals are separately available on the JTAG connector. If the **nRESET** and **nTRST** signals are linked together, resetting the system also resets the TAP controller. This means that:

- it is not possible to debug a system from reset, because any breakpoints previously set are lost
- you might have to start the debug session from the beginning, because RealView ICE does not recover when the TAP controller state is changed.

RealView ICE reset signals

The RealView ICE run control unit has two reset signals connected to the debug target hardware:

- **nTRST** drives the JTAG **nTRST** signal on the ARM processor core. It is an output that is activated whenever the RealView ICE software has to re-initialize the debug interface in the target system.
- **nSRST** is a bidirectional signal that both drives and senses the system reset signal on the target. The output is driven LOW by the debugger to re-initialize the target system.

The target hardware must include a pull-up resistor on both reset signals.

Example reset circuits

The circuits shown in Figure 9-5 on page 9-8 and Figure 9-6 on page 9-9 illustrate how the behavior described in *Reset signals* on page 9-6 can be achieved. The MAX823 used in Figure 9-6 on page 9-9 is a typical power supply supervisor. It has a current limited **nRESET** output that can be overdriven by the RealView ICE run control unit.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

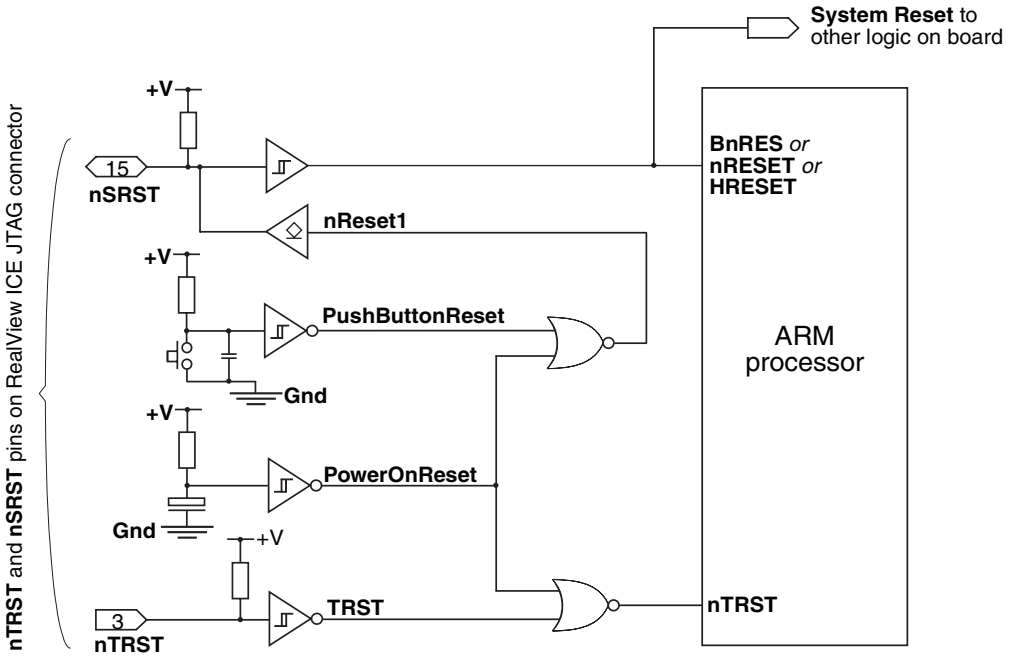


Figure 9-5 Example reset circuit logic

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

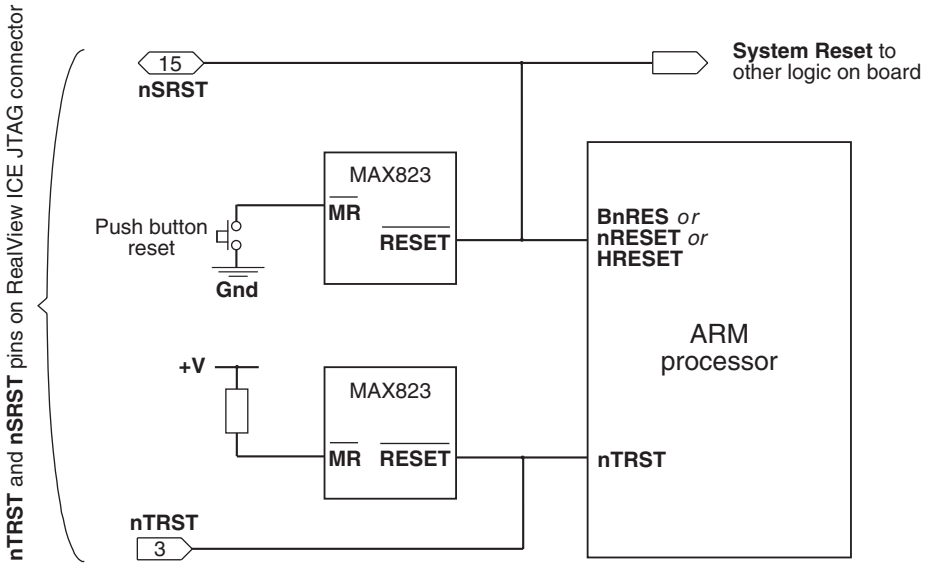


Figure 9-6 Example reset circuit using power supply monitor ICs

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

9.3 ASIC guidelines

This section describes:

- *ICs containing multiple devices*
- *Boundary scan test vectors* on page 9-11.

9.3.1 ICs containing multiple devices

If your ASIC contains multiple devices that have a JTAG TAP controller, you must serially chain them so that RealView ICE can communicate with all of them simultaneously. The chaining can either be within the ASIC, or externally:

- *TAP controllers serially chained within the ASIC*
- *TAP controllers serially chained externally* on page 9-11.

Note

There is no support in RealView ICE for multiplexing **TCK**, **TMS**, **TDI**, **TDO**, and **RTCK** between a number of different processors.

TAP controllers serially chained within the ASIC

The JTAG standard originally described serially chaining multiple devices on a PCB. This concept can be extended to serially chaining multiple TAP controllers within an ASIC. This configuration does not increase the package pin count. It does increase JTAG propagation delays, but this impact can be small if you put unaddressed TAP controllers into bypass mode (as shown in Figure 9-7 on page 9-11).

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

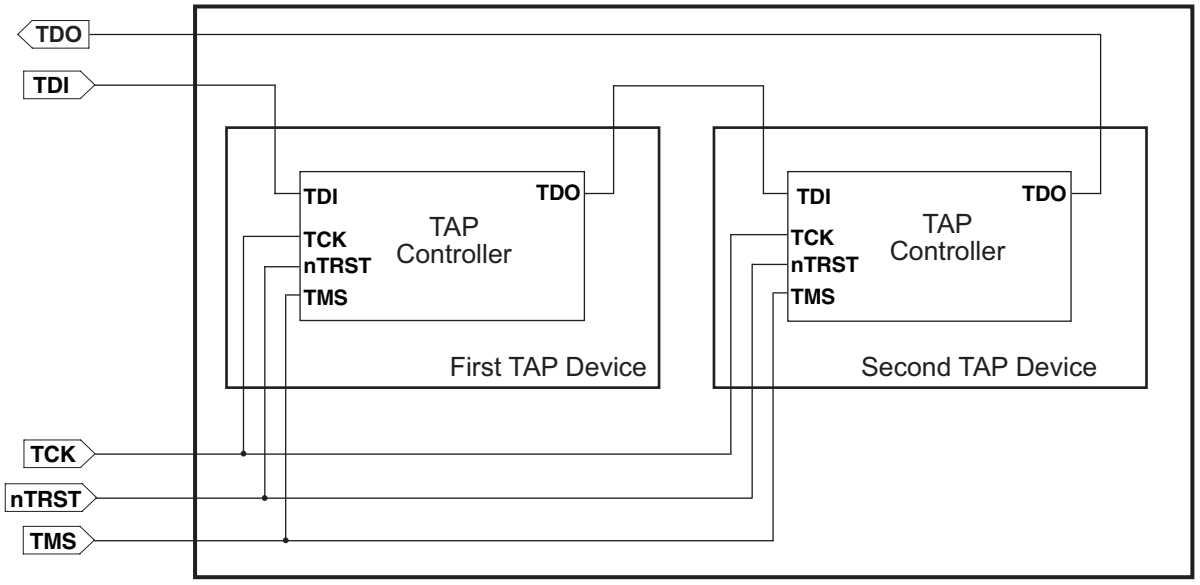


Figure 9-7 TAP Controllers serially chained within an ASIC

TAP controllers serially chained externally

You can use separate pins on the ASIC for each JTAG port, and serially chain them externally (for example on the PCB). This configuration can simplify device testing, and gives the greatest flexibility on the PCB. However, this is at the cost of many pins on the device package.

9.3.2 Boundary scan test vectors

If you use the JTAG boundary scan test methodology to apply production test vectors, you might want to have independent external access to each TAP controller. This avoids the requirement to merge test vectors for more than one block in the device. One solution to this is to adopt a hybrid, using a pin on the package that switches elements of the device into a test mode. This can be used to break the internal daisy chaining of **TDO** and **TDI** signals, and to multiplex out independent JTAG ports on pins that are used for another purpose during normal operation.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

9.4 PCB guidelines

This section contains guidelines on the physical and electrical connections present on the target board:

- *PCB connections*
- *Target interface logic levels* on page 9-13.

For RealView ICE JTAG header connectors see *JTAG interface pinouts* on page A-2.

9.4.1 PCB connections

It is recommended that you place the 20-way JTAG header as closely as possible to the target device, because this minimizes any possible signal degradation caused by long PCB tracks. The header must be a 0.1 inch pitch standard box header.

Figure 9-8 shows the layout of possible PCB connections.

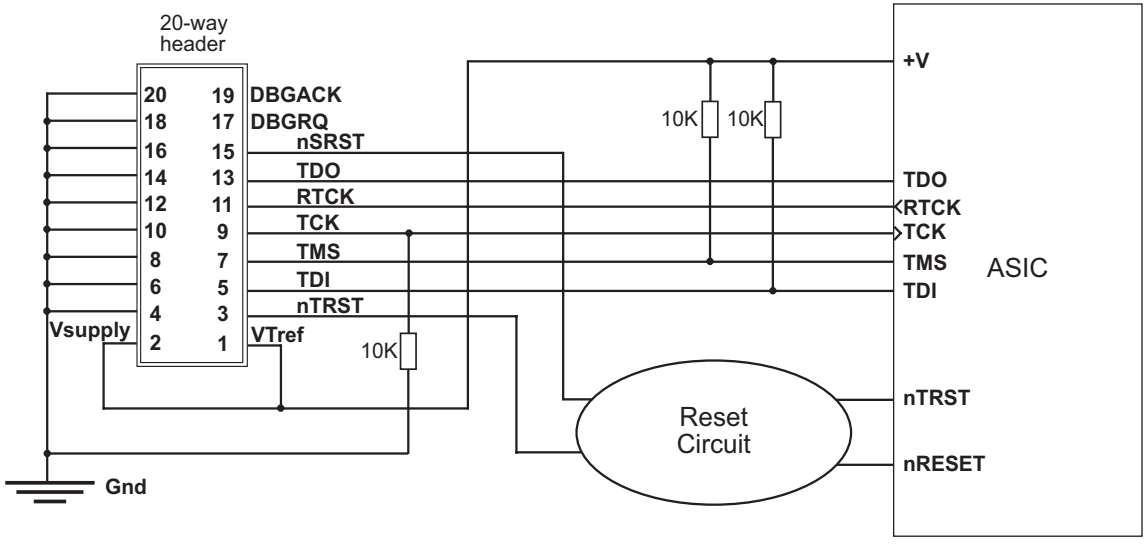


Figure 9-8 Typical PCB connections

Note

- The signals **TMS** and **TDI** must be pulled up on the PCB, as shown.
- **TCK** must be pulled down on the PCB, as shown.
- **DBGRQ** and **DBGACK** are not used by the ARM tools, but are used by some third party tools. If your tools do not use **DBGRQ** and **DBGACK**, and your device has a **DBGRQ** input, this must be pulled down on the PCB.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- **RTCK** is used by -S core variants, such as the ARM966E-S. If your device does not use **RTCK**, then **RTCK** must be pulled down on the PCB.

9.4.2 Target interface logic levels

RealView ICE is designed to interface with a wide range of target system logic levels. It does this by adapting its output drive and input threshold to a reference voltage supplied by the target system.

VTref (pin 1 on the JTAG header connector) feeds the reference voltage to the RealView ICE run control unit. This voltage, clipped at approximately 3.2V, is used as the output high voltage (**Voh**) for logic 1s (ones) on **TCK**, **TDI**, and **TMS**. 0V is used as the output low voltage for logic 0s (zeroes). The input logic threshold voltage (**Vi(th)**) for the **TDO**, **RTCK**, and **nSRST** inputs is 50% of the **Voh** level, and so is clipped to approximately 1.55V. The relationships of **Voh** and **Vi(th)** to **VTref** are shown in Figure 9-9.

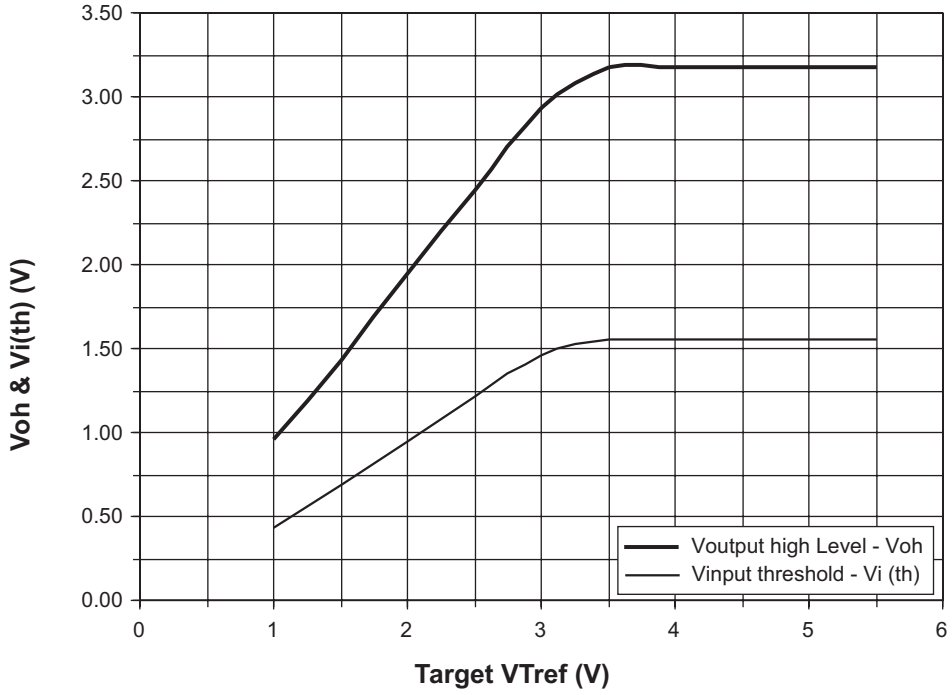


Figure 9-9 Target interface logic levels

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

RealView ICE can adapt interface levels down to **VTref** less than 1V. If, however, **VTref** becomes less than approximately 0.85V, RealView ICE interprets this condition as Target Not Present, and the software reports this as an error condition.

The **nTRST** output from RealView ICE is effectively driven as an active low signal, so it is actively pulled to 0V but relies on a 4.7kΩ pull-up resistor to end the reset state. This is because it is common to wire-OR this signal with another source of **nTRST**, such as power-on reset in the target system.

The **nSRST** output from RealView ICE is a similarly-driven active low signal, and must be pulled-up with a resistor in the target system. Because this signal is also an input to the RealView ICE run control unit, there is a 4.7kΩ internal pull-up resistor. This is to avoid spurious lows on the input when **nSRST** is not connected to the target system.

The input and output characteristics of the RealView ICE run control unit are compatible with logic levels from TTL-compatible, or CMOS logic in target systems. For information when assessing compatibility with other logic systems:

- the output impedance on the LVDS probe of the **TCK**, **TMS**, and **TDI** signals is approximately 47Ω
- the output impedance of all other signals is approximately 100Ω.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

9.5 JTAG signal integrity and maximum cable lengths

For JTAG-based debugging, you must have a very reliable connection between RealView ICE and the target, because there is no way to detect or correct errors. For this reason it is important to guarantee good signal integrity.

One factor that can limit the maximum cable length is propagation delays. Normally the RealView ICE run control unit samples data returning from the target using the same clock as for sending data, **TCK**. If the propagation delay gets too long then the RealView ICE run control unit samples the signal at the wrong time. This can be resolved by using *adaptive clocking*. In this mode the target returns a clock, **RTCK**, and RealView ICE does not sample data on **TDO**, or send more data on **TDI**, until clocked by this signal.

In an ASIC or ASSP (for example, in ARM processor based microcontrollers) the **TDO** and **RTCK** signals are not typically implemented with a stronger driver than other signals on the device. The strength of these drivers varies from device to device. An example specification is to sink or source 4mA. Many designs connect these pins on the device directly to the corresponding pins on the RealView ICE connector.

Over very short lengths of cable, such as the one supplied with RealView ICE, this type of weak driver is adequate. However, if longer cables are used then the cable becomes harder to drive as the capacitive load increases. When using longer cables it becomes essential to consider the cable as a transmission line and to provide appropriate impedance matching, otherwise reflections occur.

RealView ICE has much stronger drivers and they are connected through 100Ω series resistors to impedance match with the JTAG cable. The output circuitry of RealView ICE can easily sink or source over 40mA of current. This is very much better than the typical circuit used at the target end.

With the typical situation at the target end (weak drivers, no impedance matching resistors) you can only expect reliable operation over short cables (approximately 30cm). If operation over longer cables is required:

- For very long cables, a solution is to buffer the JTAG signals through differential drivers, such as the LVDS cable and probe supplied with RealView ICE. Reliable operation is possible over tens of metres using this technique.
- For intermediate lengths of cables, you can instead improve the circuitry used at the target end. The recommended solution is to add an external buffer with good current drive and a 100Ω series resistor for the **TDO** (and **RTCK** if used) signals on your target hardware. Using this technique you can debug over cable lengths up to several metres. Depending on cable length and propagation delays through your buffers and cables, it might still be necessary to use adaptive clocking.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

If you are not already using adaptive clocking in your design, you can generate **RTCK** at the target end by using the **TCK** signal fed through the same buffer and impedance matching circuit as used for **TDO**.

Reducing the JTAG clock speed used by RealView ICE avoids some, but not all, of the problems associated with long cables. If reducing the speed of downloading code and reading memory in the debugger is not a significant problem, try experimenting with lowering this clock speed.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

9.6 Compatibility with EmbeddedICE interface target connectors

The EmbeddedICE run control unit uses a 14-way connector for the interface to the target system. ARM Limited supplies an adaptor board with the RealView ICE run control unit, so that you can connect it to target hardware with 14-way connectors.

9.6.1 Adaptor to connect a RealView ICE run control unit to 14-way connectors

The 14-way socket on the adaptor board plugs into the box header on the target, and the RealView ICE ribbon cable is connected to the 20-way header on the adaptor board.

The three-pin header J3 has the following connections:

Pin 1 0V, **Gnd**.

Pin 2 RealView ICE connector pin 2, no connection.

Pin 3 Target connector pin 1, **SPU**.

Note

The J3 header is provided for use with Multi-ICE® run control units, and has no useful purpose with RealView ICE run control units.

The three-pin header J4 has the following connections:

Pin 1 0V.

Pin 2 RealView ICE connector pin 11, **RTCK**.

Pin 3 Resistor fed by RealView ICE connector pin 9, **TCK**.

The jumper link supplied on the adaptor board connects 0V back to the RealView ICE **RTCK** input. If the target system is to use adaptive clocking, **TCK** can be tapped off here, and the synchronized version used to clock the target can be fed back as **RTCK**.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Appendix A
JTAG Interface Connections

This appendix describes and illustrates the JTAG interface connection on the RealView® ICE unit. It contains the following sections:

- *JTAG interface pinouts* on page A-2
- *JTAG interface signals* on page A-3
- *JTAG port timing characteristics* on page A-6.

See Appendix C *RealView Trace Interface Connections* for information on the RealView Trace unit pin connections.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

A.1 JTAG interface pinouts

The RealView ICE run control unit is supplied with a short ribbon cable, and a longer ribbon cable and LVDS probe. These both terminate in a 20-way 2.54mm pitch IDC connector. You can use either cable to mate with a keyed box header on the target. The pinout is shown in Figure A-1.

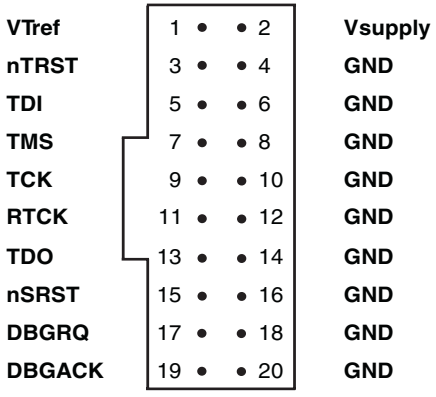


Figure A-1 JTAG interface pinout

Note

All **GND** pins must be connected to **0V** on the target hardware.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

A.2 JTAG interface signals

Table A-1 describes the signals on the JTAG interfaces.

Table A-1 JTAG signals

Signal	I/O	Description
DBGACK	-	This pin is connected in the RealView ICE run control unit, but is not supported in the current release of the software. It is reserved for compatibility with other equipment to be used as a debug acknowledge signal from the target system. It is recommended that this signal is pulled LOW on the target.
DBGRQ	-	This pin is connected in the RealView ICE run control unit, but is not supported in the current release of the software. It is reserved for compatibility with other equipment to be used as a debug request signal to the target system. The RealView ICE software maintains this signal as LOW. When applicable, RealView ICE uses the core's scan chain 2 to put the core in debug state. It is recommended that this signal is pulled LOW on the target.
GND	-	Ground.
nSRST	Input/output	Active Low output from RealView ICE to the target system reset, with a 4.7kΩ pull-up resistor for de-asserted state. This is also an input to RealView ICE so that a reset initiated on the target can be reported to the debugger. This pin must be pulled HIGH on the target to avoid unintentional resets when there is no connection.
nTRST	Output	Active Low output from RealView ICE to the Reset signal on the target JTAG port, with a 4.7kΩ pull-up resistor for de-asserted state. This pin must be pulled HIGH on the target to avoid unintentional resets when there is no connection.
RTCK	Input	Return Test Clock signal from the target JTAG port to RealView ICE. Some targets must synchronize the JTAG inputs to internal clocks. To assist in meeting this requirement, you can use a returned, and retimed, **TCK** to dynamically control the **TCK** rate. RealView ICE provides Adaptive Clock Timing, that waits for **TCK** changes to be echoed correctly before making more changes. Targets that do not have to process **TCK** can ground this pin.
TCK	Output	Test Clock signal from RealView ICE to the target JTAG port. It is recommended that this pin is pulled LOW on the target.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Table A-1 JTAG signals (continued)

Signal	I/O	Description
TDI	Output	Test Data In signal from RealView ICE to the target JTAG port. It is recommended that this pin is pulled HIGH on the target.
TDO	Input	Test Data Out from the target JTAG port to RealView ICE. It is recommended that this pin is pulled HIGH on the target.
TMS	Output	Test Mode signal from RealView ICE to the target JTAG port. This pin must be pulled HIGH on the target so that the effect of any spurious **TCK**s when there is no connection is benign.
Vsupply	Input	This pin is not connected in the RealView ICE run control unit. It is reserved for compatibility with other equipment to be used as a power feed from the target system.
VTref	Input	This is the target reference voltage. It indicates that the target has power, and It must be at least 0.628V. **VTref** is normally fed from $\mathbf{V_{dd}}$ on the target hardware and might have a series resistor (though this is not recommended). There is a 10kΩ pull-down resistor on **VTref** in RealView ICE.

A.2.1 JTAG interface signal details

VTref is used to create the logic-level reference for the input comparators on **TDO**, **RTCK** and **nSRST**. RealView ICE clips the logic-level reference to3.3V. RealView ICE inputs (**TDO**, **RTCK** and **nSRST**) are taken to high-impedance inputs of comparators. Each input is read as a logic 1 when it exceeds half the voltage reference.

VTref also controls the output logic levels to the target. RealView ICE uses analog switches to drive the output signals. The output is connected to ground for a logic 0 and to the JTAG interface voltage for a logic 1.

TDI, **TMS** and **TCK** have 47Ω series resistors on the LVDS probe. All other outputs from the LVDS probe and the RealView ICE 20-way connector have 100Ω series resistors.

nSRST and **nTRST** are both active low signals. When asserted, these signals are connected to ground for a logic 0, and when de-asserted use a 4.7kΩ pull-up for a logic 1.

You must ensure that your board has appropriate pull-up and pull-down resistors on the JTAG signals:

- **TMS**, **TDI**, **TDO**, **nSRST** and **nTRST** must have pull-ups.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- **TCK** must have a pull-down to enable hot swap and post-mortem debugging
- **RTCK** must have a pull-down to fix a stable value on that signal when debugging a non-synthesizable core.
- **DBGRQ** must have a pull-down. This ensures that the core doesn't enter debug state in an uncontrolled way.
- **DBGACK** must have a pull-down, so the default value that the debugger sees is core not in debug state.

The recommended value for pull-ups and pull-downs is 10kΩ, although the optimum value depends on the signal load. For example, pull-downs must be about 1kΩ when working with TTL logic.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

A.3 JTAG port timing characteristics

This section describes the timing characteristics of the RealView ICE unit:

- Figure A-2 shows the JTAG port timing and parameters
- Table A-2 on page A-7 gives the requirements for these parameters when you are using the JTAG A port
- Table A-3 on page A-7 gives the requirements for these parameters when you are using the JTAG B port.

These timing characteristics must be considered if you design a target device or board and want to be able to connect RealView ICE at a particular **TCK** frequency. The characteristics relate to the RealView ICE hardware. You must consider them in parallel with the characteristics of your target.

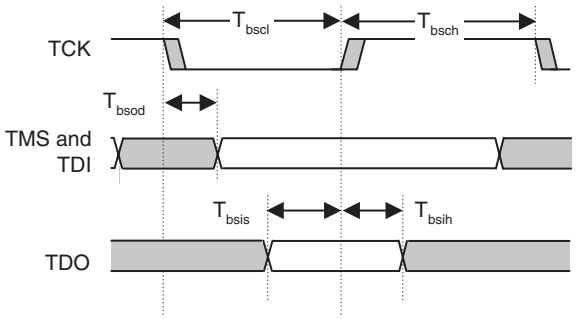


Figure A-2 JTAG port timing diagram

In a JTAG device that fully complies to IEEE1149.1, **TDI** and **TMS** are sampled on the rising edge of **TCK**, and **TDO** changes on the falling edge of **TCK**. To take advantage of these properties, RealView ICE samples **TDO** on the rising edge of **TCK** and changes its **TDI** and **TMS** signals on the falling edge of **TCK**. This means that with a fully compliant target, issues with minimum setup and hold times can always be resolved by decreasing the **TCK** frequency, because this increases the separation between signals changing and being sampled.

Note

There are no separate timing requirements for adaptive clocking mode, as the minimum T_{bsch} and T_{bscl} times are identical and are the same as for non-adaptive clocking. T_{bsis} and T_{bsih} are relative to RTCK rising, and not TCK rising, as RTCK is used to sample TDO in adaptive clocking mode.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

The only real timing difference is that in adaptive mode, RealView ICE samples **TDO** on the rising edge of **RTCK** and not **TCK**, so the **TDO** timing will be relative to **RTCK**.

Table A-2 shows the timing requirements for the JTAG A port, measured open circuit (no target connection, except for 3.3V reference on **VTref**) with the supplied JTAG cable connected.

Table A-2 RealView ICE JTAG A timing requirements

Parameter	Min	Max	Description
T_{bscl}	50ns	500µs	**TCK** LOW period
T_{bsch}	50ns	500µs	**TCK** HIGH period
T_{bsod}	-	6.0ns	**TDI** and **TMS** valid from **TCK** (falling)
T_{bsis}	15.0ns	-	**TDO** setup to **TCK** (rising)
T_{bsih}	6.0ns	-	**TDO** hold from **TCK** (rising)

Table A-3 shows the timing requirements for the JTAG B port, measured open circuit (no target connection, except for 3.3V reference on **VTref**) with no cable connected.

Table A-3 RealView ICE JTAG B timing requirements

Parameter	Min	Max	Description
T_{bscl}	10ns	500µs	**TCK** LOW period
T_{bsch}	10ns	500µs	**TCK** HIGH period
T_{bsod}	-	3.2ns	**TDI** and **TMS** valid from **TCK** (falling)
T_{bsis}	6.2ns	-	**TDO** setup to **TCK** (rising)
T_{bsih}	4.5ns	-	**TDO** hold from **TCK** (rising)

Note

- The RealView ICE software enables you to change the **TCK** frequency. The **TCK** LOW:HIGH mark-space ratio is always 50:50. The other parameters must be considered with the specific values of T_{bscl} and T_{bsch} that you have chosen. The default values for an autoconfigured single-TAP system are, nominally, T_{bscl}=50ns and T_{bsch}=50ns.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

- T_{bsod} is the maximum delay between the falling edge of **TCK** and valid levels on the **TDI** and **TMS** RealView ICE output signals. The target samples these signals on the following rising edge of **TCK** and so the minimum setup time for the target, relative to the rising edge of **TCK**, is $T_{bscl}-T_{bsod}$.
- T_{bsis} is the minimum setup time for the **TDO** input signal, relative to the rising edge of **TCK** when RealView ICE samples this signal. The target changes its **TDO** value on the previous falling edge of **TCK** and so the maximum time for the target **TDO** level to become valid, relative to the falling edge of **TCK**, is $T_{bscl}-T_{bsis}$.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Appendix B
User I/O Connections

This appendix describes and illustrates the additional input and output connections provided in RealView® ICE, and consists of:

- *The RealView ICE User I/O connector* on page B-2.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

B.1 The RealView ICE User I/O connector

This section describes the User *Input/Output* (I/O) connector.

The User I/O connector is situated on an end panel of the RealView ICE run control unit (see Figure 1-1 on page 1-8). The connector is a 10-way 2.54mm pitch IDC header that mates with IDC sockets mounted on a ribbon cable (see Figure B-1).

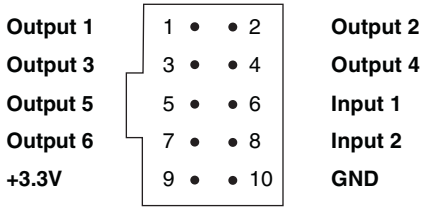


Figure B-1 User I/O pin connections

Warning

You must establish a common ground between the RealView ICE run control unit and the target hardware before you connect any of the User I/O signals.

Table B-1 shows the User I/O pin connections.

Table B-1 User I/O pin connections

Pin	Signal	I/O	Description
Pin 1	**Output 1**	Output	This is a user output bit. It operates at a 3.3V swing, with a 100Ω series resistance.
Pin 2	**Output 2**	Output	This is a user output bit. It operates at a 3.3V swing, with a 100Ω series resistance.
Pin 3	**Output 3**	Output	This is a user output bit. It operates at a 3.3V swing, with a 100Ω series resistance.
Pin 4	**Output 4**	Output	This is a user output bit. It operates at a 3.3V swing, with a 100Ω series resistance.
Pin 5	**Output 5**	Output	This is a user output bit. It operates at a 3.3V swing, with a 100Ω series resistance.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Table B-1 User I/O pin connections (continued)

Pin	Signal	I/O	Description
Pin 6	**Input 1**	Input	This is a user input bit. It has a 10kΩ weak pull-up to the unit internal +3.3V supply, and requires a $\mathbf{V_{ih(min)}}$ of 2.0V and a $\mathbf{V_{il(max)}}$ of 0.8V. It can safely be driven by 5V logic levels, and has *Electro Static Discharge* (ESD) protection greater than the 2kV human body model. This pin is not currently supported.
Pin 7	**Output 6**	Output	This is a copy of the trigger output on the end panel (see Figure 1-1 on page 1-8). It operates at a 3.3V swing, with a 100Ω series resistance.
Pin 8	**Input 2**	Input	This is a copy of the trigger input on the end panel (see Figure 1-1 on page 1-8). It has a 10kΩ weak pull-up to the unit internal +3.3V supply, and requires a $\mathbf{V_{ih(min)}}$ of 2.0V and a $\mathbf{V_{il(max)}}$ of 0.8V. It can safely be driven by 5V logic levels, and has *Electro Static Discharge* (ESD) protection greater than the 2kV human body model. This pin is not currently supported
Pin 9	**+3.3V**	Output	This is intended for powering external signal conditioning circuitry, to a maximum current of 100mA. Incorrect use of this output might cause the RealView ICE run control unit to enter current limit.
Pin 10	**GND**	-	-

Note

Input is not currently supported on the User I/O pin connections.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Appendix C
RealView Trace Interface Connections

This appendix describes and illustrates the RealView® Trace v1.0 pin connections. It contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

C.1 RealView Trace front panel components

The layout of the RealView Trace front panel is shown in Figure C-1.

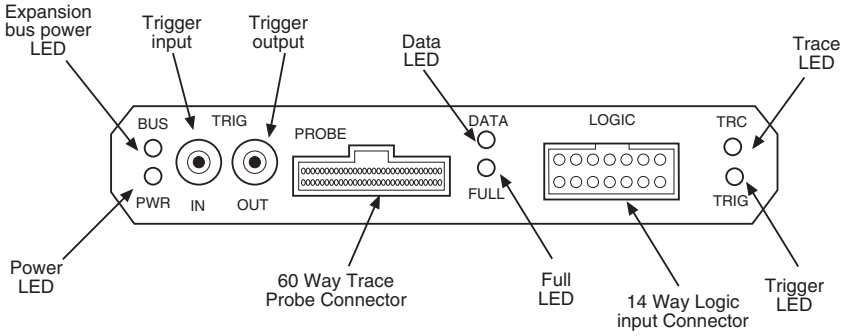


Figure C-1 RealView Trace panel layout

The components are:

- *LEDs*
- *Trigger input and output SMB connectors* on page C-3
- *RealView Trace probe connector* on page C-3
- *Logic input connector* on page C-7.

Note

The Trigger input, Trigger output and Logic connectors are not supported by RealView ICE v3.0.

C.1.1 LEDs

The LEDs on the RealView Trace front panel have the following functions:

PWR The PWR LED indicates that the RealView Trace unit is powered up. This is always lit when the unit is powered from its own external power supply. When RealView Trace is powered by RealView ICE, the Power LED is lit only when RealView ICE is initializing the RealView Trace unit.

BUS The BUS LED indicates module initialization and RealView ICE expansion bus power. It lights up when RealView ICE is initializing the RealView Trace unit, and remains lit while the RealView ICE expansion bus is powered.

DATA The DATA LED indicates that the RealView Trace module has data in its buffer

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

FULL The FULL LED indicates that the RealView Trace module data buffer is full.

TRACE The TRACE LED indicates that the RealView Trace module has enabled the buffer for capture.

TRIG the TRIG LED indicates that the RealView Trace module has triggered

C.1.2 Trigger input and output SMB connectors

The trigger input has 10K weak pull-up to the internal 3.3V supply and requires a $V_{ih(min)}$ of 2.0v and $V_{il(max)}$ of 0.8V. The input can be safely driven by 5V logic levels and is ESD protected greater than 2kV human body model.

The trigger output operates at 3.3 voltage swings with a 100Ω series resistance.

Note

The Trigger input and Trigger output connectors are not supported by RealView ICE v3.0.

C.1.3 RealView Trace probe connector

RealView Trace supports 4, 8, and 16-bit data port widths with the high density target connector described in *Trace high density connector*.

RealView Trace can capture the state of signals **PIPESTAT[2:0]**, **TRACESYNC** and **TRACEPKT[n:0]** at each rising edge of each **TRACECLK** or on each alternate rising or falling edge. For details on setting capture options, see the chapter that describes tracing in the *RealView Debugger Trace User Guide*.

Trace high density connector

RealView Trace provides a connection to the AMP Mictor connector. Figure C-2 on page C-4 shows the target connector (AMP 2-5767004-2 38-pin surface mount receptacles).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

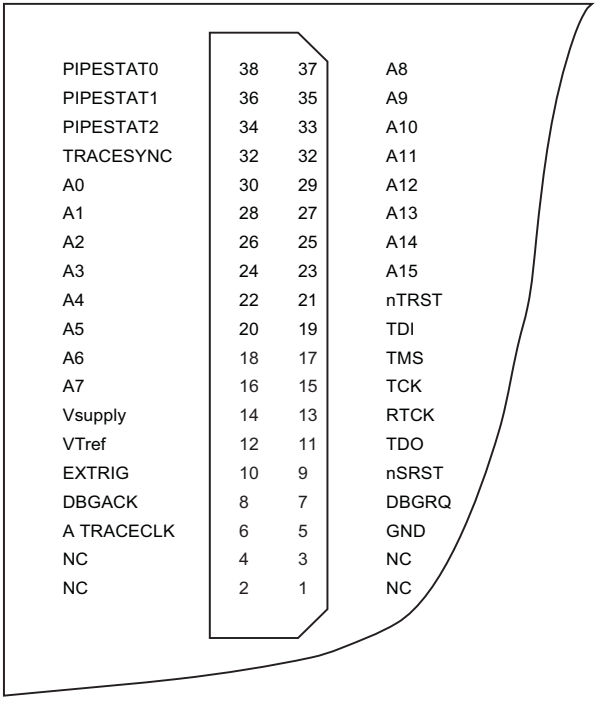


Figure C-2 Pin surface mount receptacles on trace probe connector

Table C-1 shows the pinouts in single *Embedded Trace Macrocell* (ETM) mode.

Table C-1 Connector signals

Target board signal	Pin	Description
NC	1	No Connect
NC	3	No Connect
GND	5	Signal ground
DBGRQ	7	Debug request

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Table C-1 Connector signals (continued)

Target board signal	Pin	Description
nSRST	9	Active Low output from RealView ICE to the target system reset, with a 4.7kΩ pull-up resistor for de-asserted state. This is also an input to the run control unit so that a reset initiated on the target can be reported to the debugger.
TDO	11	Test data output from target JTAG port
RTCK	13	Return test clock from the target JTAG port
TCK	15	Test clock from the run control unit to the JTAG port
TMS	17	Test mode select from the run control unit to the JTAG port
TDI	19	Test data input from the run control unit to the JTAG port
nTRST	21	Active Low output from the run control unit to the target TAP reset line, with a 4.7kΩ pull-up resistor for de-asserted state.
Port A15 TRACEPKT	23	The trace packet port
Port A14 TRACEPKT	25	The trace packet port
Port A13 TRACEPKT	27	The trace packet port
Port A12 TRACEPKT	29	The trace packet port
Port A11 TRACEPKT	31	The trace packet port
Port A10 TRACEPKT	33	The trace packet port
Port A9 TRACEPKT	35	The trace packet port
Port A8 TRACEPKT	37	The trace packet port

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Table C-1 Connector signals (continued)

Target board signal	Pin	Description
NC	2	No Connect
NC	4	No Connect
Port A TRACECLK	6	Clocks trace data on rising edge or both edges
DBGACK	8	Debug acknowledge from the test chip, **High** when in debug state
EXTRIG	10	Optional external trigger signal to the ETM
VTref	12	Target reference voltage
Vsupply	14	Supply voltage (not used by RealView ICE)
Port A7 TRACEPKT	16	The trace packet port
Port A6 TRACEPKT	18	The trace packet port
Port A5 TRACEPKT	20	The trace packet port
Port A4 TRACEPKT	22	The trace packet port
Port A3 TRACEPKT	24	The trace packet port
Port A2 TRACEPKT	26	The trace packet port
Port A1 TRACEPKT	28	The trace packet port
Port A0 TRACEPKT	30	The trace packet port
Port A TRACESYNC	32	Start of branch sequence signal
Port A PIPESTAT2	34	RAM pipeline status
Port A PIPESTAT1	36	RAM pipeline status
Port A PIPESTAT0	38	RAM pipeline status

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

C.1.4 Logic input connector

The logic inputs have a 1M weak pull-down and require a $\mathbf{V_{ih(min)}}$of 2.0v and $\mathbf{V_{il(max)}}$ of 0.8V. These inputs can safely be driven by 5V logic levels and are ESD protected greater than 2kV human body model. All logic inputs are sampled by an internally generated clock derived from the target ETM clock.The 3.3V power output can be used to power external signal conditioning circuits to a maximum current of 100mA. Incorrect use of this output might cause the unit to enter current limit.

Note

The Logic input connector is not supported by RealView ICE v3.0.

Figure C-3 shows the pin surface mount receptacles on the logic input connector.

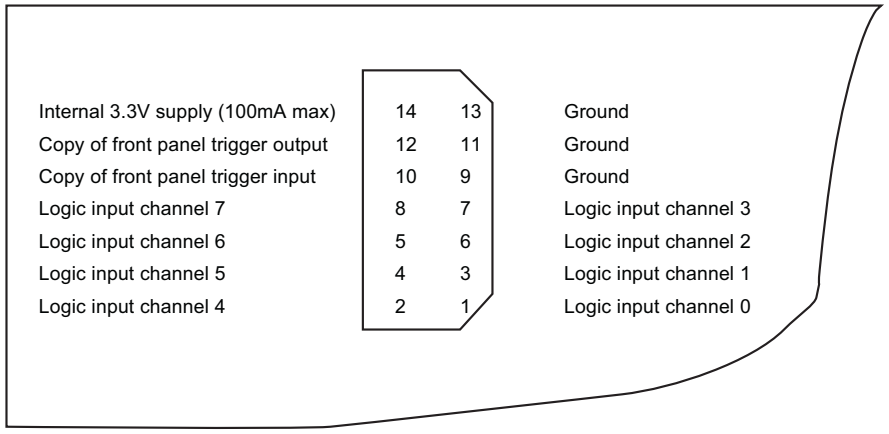


Figure C-3 Pin surface mount receptacles on logic input connector

Warning

Before you connect any of the logic signals, a common ground must be established between the RealView Trace data capture unit and the target hardware. This is usually achieved through the RealView Trace probe, but can also be achieved by the trigger SMB connectors or the logic input connector itself when not using the supplied logic grabbers.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

C.2 Trace signals

Data transfer is synchronized by the **TRACECLK** signal.

C.2.1 Signal levels

The maximum capacitance presented by RealView Trace at the trace port connector, including the connector and interfacing logic, is less than 6pF. The trace port lines have a matched impedance of 50Ω.

The RealView Trace unit operates with a target board that has a supply voltage range of 1.0V-5.0V.

C.2.2 Clock frequency

For capturing trace port signals synchronous to **TRACECLK**, RealView Trace supports a **TRACECLK** frequency of up to 250MHz. Figure C-4 and Table C-2 describe the timing for **TRACECLK**.

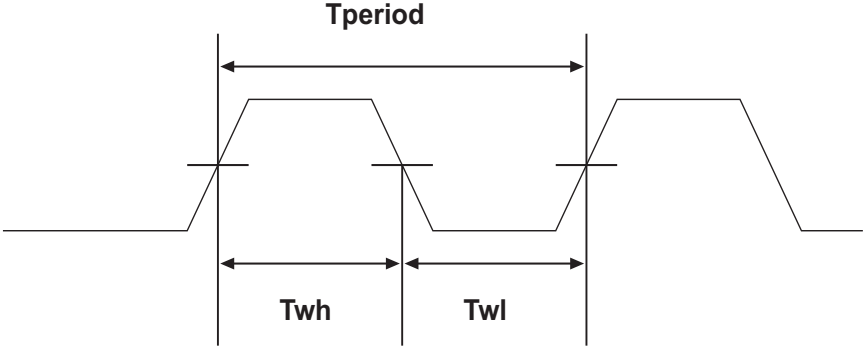


Figure C-4 Clock waveforms

Table C-2 TRACECLK frequencies

Parameter	Minimum	Period
Tperiod	4.0ns	Clock period
Twh	1.5ns	High pulse width
Twl	1.5ns	Low pulse width

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

C.2.3 Data setup and hold

Figure C-5 and Table C-3 show the setup and hold timing of the trace signals with respect to **TRACECLK**.

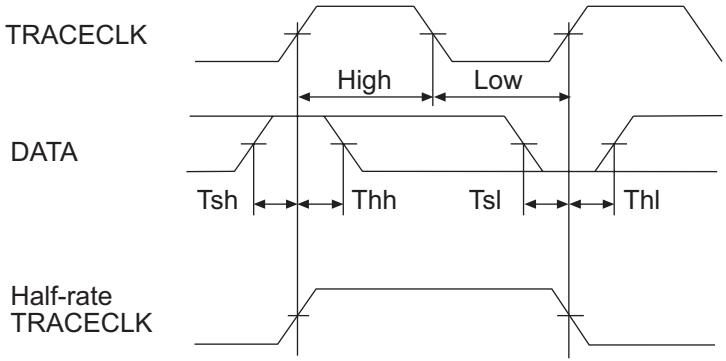


Figure C-5 Data waveforms

Table C-3 Data setup and hold

Parameter	Minimum	Period
Tsh	2.0ns	Data setup high
Thh	1.0ns	Data hold high
Tsl	2.0ns	Data setup low
Thl	1.0ns	Data hold low

Note

RealView Trace supports half-rate clocking mode. Data is output on each edge of the **TRACECLK** signal and **TRACECLK (max)** <= 125MHz. For half-rate clocking, the setup and hold times at the Mictor connector must be observed.

C.2.4 Switching thresholds

The RealView Trace probe detects the target signalling reference voltage (VTref) and automatically adjusts its switching thresholds to VTref/2. For example, on a 3.3 volt target system, the switching thresholds are set to 1.65 volts.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

C.2.5 Hot plugging

RealView Trace is not damaged if it is powered up when plugged into an unpowered target or if an unpowered RealView Trace unit is plugged into a powered target.

If both the RealView Trace unit and the target are powered, no damage occurs to the RealView Trace unit, but there might be damage to a (third-party) target system.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Appendix D
Designing the Target Board for Tracing

This appendix describes the properties of a target board that can be connected to RealView® Trace. It contains the following sections:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.1 Overview of high-speed design

Failure to observe high-speed design rules when designing a target system containing an ARM *Embedded Trace Macrocell* (ETM) trace port can result in incorrect data being captured by RealView Trace.You must give serious consideration to high-speed signals when designing the target system.

The signals coming from an ARM ETM trace port can have very fast rise and fall times, even at relatively low frequencies. For example, a signal with a rise time of 1ns has an effective knee frequency of 500MHz and a signal with a rise time of 500ps has an effective knee frequency of 1GHz ($f_{knee} = 0.5/Tr$).

Note

These principles apply to all of the trace port signals (**TRACEPKT[15:0]**, **PIPESTAT[0:2]**, **TRACESYNC**), but special care must taken with **TRACECLK**.

D.1.1 Avoid stubs

Stubs are short pieces of track that tee off from the main track carrying the signal to, for example, a test point or a connection to an intermediate device. Stubs cause impedance discontinuities that affect signal quality and must be avoided.

Special care must therefore be taken when ETM signals are multiplexed with other pin functions and where the PCB is designed to support both functions with differing tracking requirements.

D.1.2 Minimize signal skew (balancing PCB track lengths)

You must attempt to match the lengths of the PCB tracks carrying all of **TRACECLK**, **PIPESTAT**, **TRACESYNC,** and **TRACEPKT** from the ASIC to the mictor connector to within approximately 0.5 inches (12.5mm) of each other. Any greater differences directly impact the setup and hold time requirements.

D.1.3 Minimize crosstalk

Normal high-speed design rules must be observed. For example, do not run dynamic signals parallel to each other for any significant distance, keep them spaced well apart, and use a ground plane and so forth. Particular attention must be paid to the **TRACECLK** signal. If in any doubt, place grounds or static signals between the **TRACECLK** and any other dynamic signals.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.1.4 Use impedance matching and termination

Termination is almost certainly necessary, but there are some circumstances where it is not required. The decision is related to track length between the ASIC and the Mictor connector (see *Termination* on page D-4).

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.2 Termination

To calculate the maximum track length that can be used without termination, you must know the following about your ASIC and PCB:

- the rise time (`Tr`) of the signals coming off the ASIC
- the impedance of the output drivers on the ASIC for the ETM signals
- the propagation delay per inch of PCB track (`Tpdt`).

D.2.1 Example

The maximum track length without termination is given by:

$$\text{Length}_{(\text{inches})} < \frac{\text{Tr}_{(\text{ps})}}{5\ \text{Tpdt}_{(\text{ps})}}$$

That is, the signal propagation delay from the ASIC to the Mictor connector must be less than one fifth of the signal rise time. This calculation allows for the delay of the Mictor connector and the delay of the track from the Mictor to the input buffers on the probe.

For a case where the signal rise time (`Tr`) is 1ns (1000ps) and the propagation delay of the trace (`Tpdt`) is 160ps per inch (typical for a PCB made with FR4 laminate), `L` must be less than `1000/(5 * 160)`. That is, `L` must be less than 1.25 inches. If the PCB trace length from the ASIC to the Mictor connector is greater than 1.25 inches, you must use termination.

D.2.2 Termination options

There are four termination options:

Matched impedance

Where available, the best termination scheme is to have the ASIC manufacturer match the output impedance of the driver to the impedance of the PCB track on your board. This produces the best possible signal.

Series (source) termination

This method requires a resistor fitted in series with signal. The resistor value plus the output impedance of the driver must be equal to the PCB track impedance.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

DC parallel termination

This requires either a single resistor to ground or a pull-up/pull-down combination of resistors (Thevenin termination), fitted at the end of each signal and as close as possible to the Mictor connector. If a single resistor is used, its value must be set equal to the PCB track impedance. If the pull-up/pull-down combination is used, their resistance values must be selected so that their parallel combination equals the PCB track impedance.

Caution

At lower frequencies, parallel termination requires considerably more drive capability from the ASIC than series termination and so, in practice, DC parallel termination is rarely used.

AC parallel termination

This typically uses a resistor and capacitor in series to ground.

Caution

AC termination can only be used with signals with a 1:1 mark/space ratio (DC balanced) and is not suitable for asymmetric signals such as the **TRACEPKT** and **PIPESTAT** signals. For this reason, AC termination is not recommended.

D.2.3 Rules for series terminators

Series (source) termination is the most commonly used method. The basic rules are:

1. The series resistor must be placed as close as possible to the ASIC pin (less than 0.5 inches)
2. The value of the resistor must equal the impedance of the track minus the output impedance of the output driver. So for example, a 50Ω PCB track driven by an output with a 17Ω impedance, requires a resistor value of 33Ω.
3. A source terminated signal is only valid at the end of the signal path. At any point between the source and the end of the track, the signal appears distorted because of reflections. Any device connected between the source and the end of the signal path therefore sees the distorted signal and might not operate correctly. Care must be taken not to connect devices in this way, unless the distortion does not affect device operation.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.2.4 PCB track impedance

Use the following formula only for microstrips (track on outer layer over a ground plane):

$$\text{Impedance} = \frac{87}{\sqrt{(E_r + 1.41)}} \ln\left[\frac{5.98h}{(0.81w + t)}\right] \Omega$$

Where:

h	Height above ground plane (inches)
w	Trace width (inches), and $0.1 < w/h < 2$
t	Trace thickness (inches)
E_r	Relative permittivity of core/prepreg, and $1 < E_r < 15$

The dimensions h, w, and t are shown in Figure D-1.

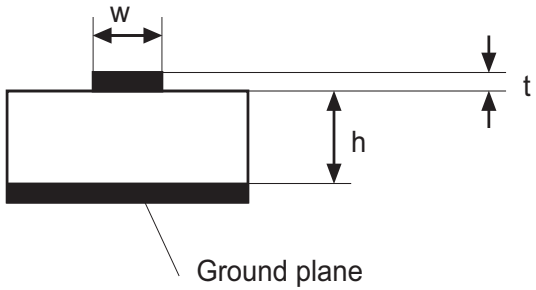


Figure D-1 Track impedance

As an example, the following track (in microstrip form) has an impedance of 51.96Ω:

h	0.005 inch height above ground
w	0.007 inch width track
t	0.0014 inch thickness (1 oz. finished weight)
E_r	4.5 (FR4 laminate)

Note

As the track width increases, the impedance decreases.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.3 Probe dimensions and keep out areas

Figure D-2 shows the probe attached to a target board.

Caution

The Mictor connector is not robust. It is recommended that the plastic shroud is fitted around the target connector. This part is not supplied as standard with RealView Trace.

The Mictor connector support shroud is available from Agilent as part number E5346 - 44701

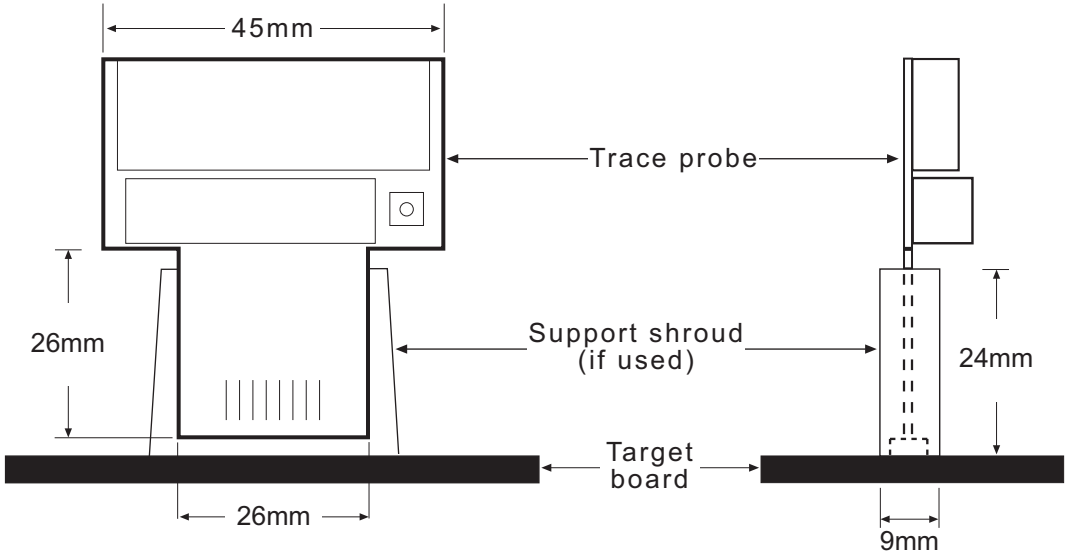


Figure D-2 Probe dimensions

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.4 Signal requirements

Table D-1 lists the specifications that apply to the signals as seen at the Mictor connector.

Table D-1 Signals seen at the Mictor connector

Signal	Value
Fmax	250MHz
Tsh and Tsl setup times (min.)	2.0ns
Thh and Thl hold times (min.)	1.0ns
TRACECLK high pulse width (min.)	1.5ns
TRACECLK low pulse width (min.)	1.5ns

The signal waveform is shown in Figure D-3.

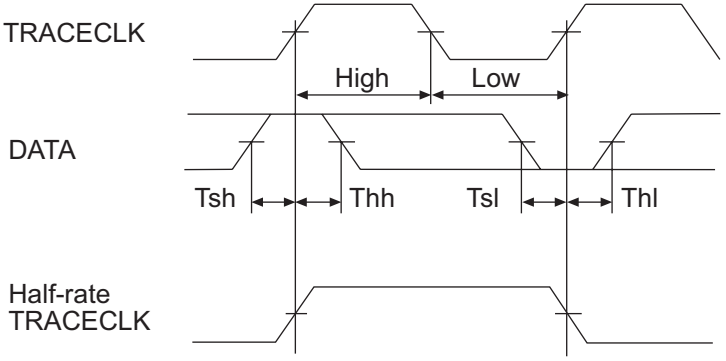


Figure D-3 Setup and hold

Note

RealView Trace supports half-rate clocking mode. Data is output on each edge of the **TRACECLK** signal and **TRACECLK (max)** <= 125MHz. For half-rate clocking, the setup and hold times at the Mictor connector must be observed.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.4.1 Switching Thresholds

The RealView Trace probe senses the target signalling reference voltage (VTref) and automatically adjusts its switching thresholds to VTref/2. For example, on a 3.3 volt target system, the switching thresholds are set to 1.65 volts.

D.4.2 Hot plugging

If both RealView Trace and the target are powered, plugging or unplugging the trace cable does not damage or crash the RealView Trace system. It is not possible, however, to guarantee similar immunity to any third party target system. You must, therefore, take precautions such as pulling inputs and driving or making high Z outputs when **Vsupply** is not present.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

D.5 Probe modeling

For **TRACECLK** frequencies above 100MHz, it is recommended that modeling is used. The characteristics for the RealView Trace probe are:

- The Mictor connector single line model can be downloaded directly from the AMP website at `http://www.amp.com/` (models are currently located at `http://www.amp.com/simulation/scripts/models.asp`). The closest available model is entitled *MICTOR, .025" PITCH, 2 ROW, VERTICAL PLUG TO VERTICAL RECEPTACLE, 0.260" [6.6 mm] HEIGHT* .

  The connector assembly (plug and receptacle together) can be viewed as a transmission line with an impedance of 45Ω and a propagation delay of 39ps. For more accurate modeling, multi-line models can be requested from AMP.

- All traces are microstrips (on outer layers over ground planes)
- Trace widths are 0.007 inch
- Trace thickness is 0.0014 inch (1oz finished weight)
- Distance from trace to ground plane is 0.005 inch
- Trace lengths from Mictor to input buffers are 0.329 inches maximum and 0.193 inches minimum
- Average trace separation is 0.010 inches
- `Er` = 4.5.
- The probe input buffers are a mixture of Fairchild FIN1104MTD and FIN1108MTD devices (4 channel & 8 channel devices respectively).

  The following signals are routed to FIN1104MTD inputs:
  - TRACEPKT4
  - TRACEPKT5
  - TRACEPKT6
  - TRACEPKT7
  - TRACECLK

  All other signals are routed to FIN1108MTD devices.

  The datasheets, `ibis` and `hspice` models for these devices are available on the Fairchilds web site at `http://www.fairchildsemi.com/`.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Appendix E
Hardware Variants

Depending on the hardware unit that you are using, it is possible that your unit's features may be different in appearance from those generally referred to in this document.

This appendix describes the differences between the RealView® ICE v3.0 hardware unit and its predecessors.

This appendix includes:

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

E.1 RealView ICE hardware

Depending on the RealView ICE hardware unit that you are using, your unit's features may well be different from those referred to elsewhere in this document.

This appendix describes the differences in appearance or functionality between the RealView ICE v3.0 hardware unit and its predecessors. It consists of the following sections:

- *End panel changes*
- *New LVDS probe* on page E-4

E.1.1 End panel changes

The RVI v3.0 hardware unit has been enhanced, involving changes to both end panels.

The host computer ports end panel now also comprises:

- a reset switch, RST
- a DC socket negative/positive symbol.

The host computer ports end panel details are shown in Figure E-1.

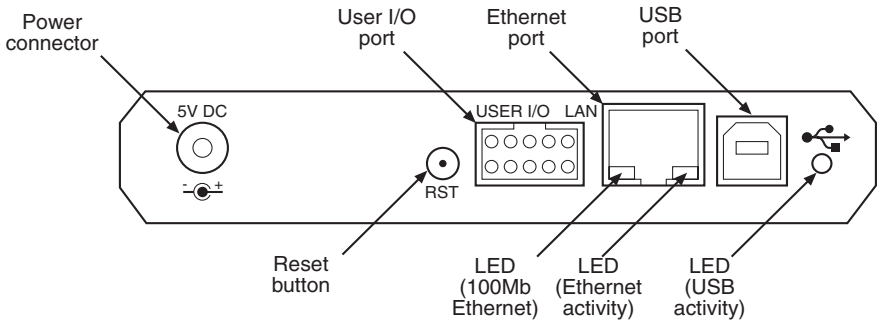


Figure E-1 RVI v3.0 host computer ports end panel

Compare this new arrangement with that of earlier units, an example of which is shown in Figure E-2 on page E-3.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

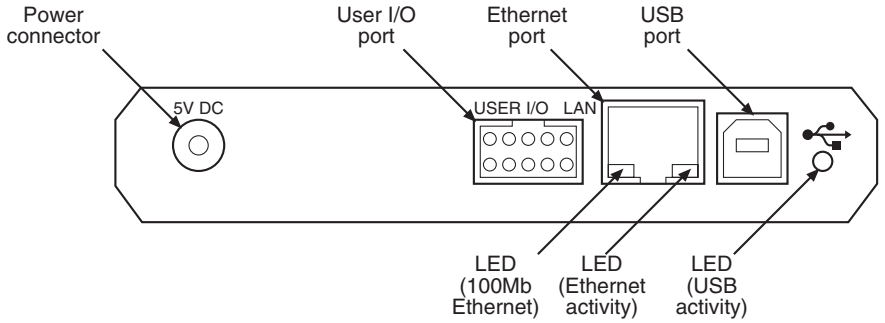


Figure E-2 Replaced host computer ports end panel

The main change in functionality between the v3.0 hardware and that of its predecessors is the introduction of the RST button, which allows users to reset the RVI to its default state without the requirement to disconnect the unit from the power supply and then reconnect. For further information on the functionality of each of the features shown in Figure E-1 on page E-2, refer to *The RealView ICE run control unit* on page 1-8.

In the case of the target hardware ports end panel, this now shows the name of each of the four identification LEDs. Refer to Figure E-3.

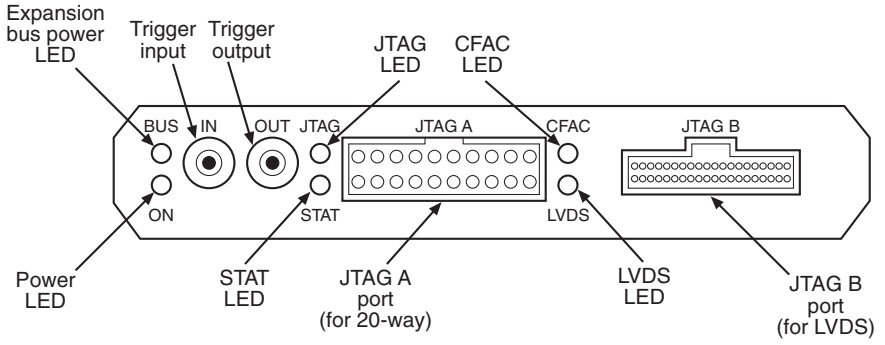


Figure E-3 RVI v3.0 target hardware ports end panel

Here, the identification LEDs JTAG, STAT, CFAC and LVDS replace the previous lettering system of A, B, C and D, respectively. An example of the replaced target hardware ports end panel is shown in Figure E-4 on page E-4.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

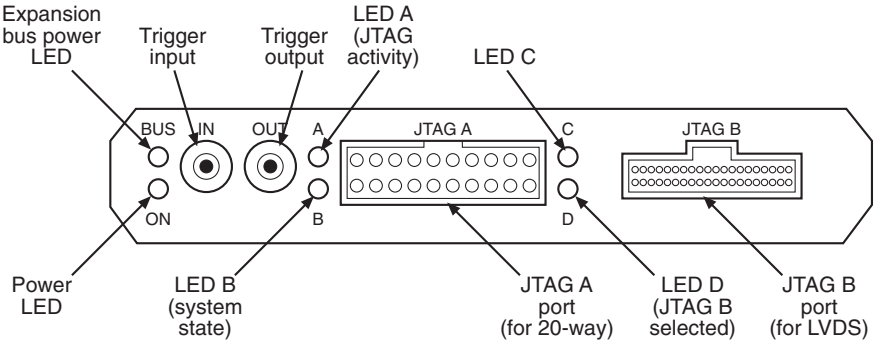


Figure E-4 Replaced target hardware ports end panel

E.1.2 New LVDS probe

An updated Low Voltage Differential Signaling (LVDS) probe has been introduced in RVI v3.0. The probe is visibly different to its predecessor by the presence of two colored LEDs, which signify the status of the activity taking place.

Figure E-5 shows the new LVDS probe's dimensions and its LED locations.

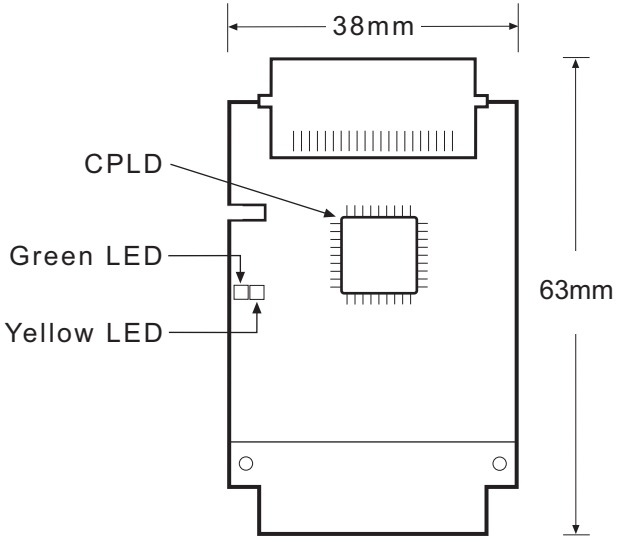


Figure E-5 LVDS probe LEDs and dimensions

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Table E-1 provides a diagnostic means to determine the type of activity taking place, according to the permutations of the green and yellow LEDs.

Table E-1 Diagnostic table

Green	Yellow	Power	CPLD OK	JTAG Activity
Off	Off	No	N/A	N/A
Dim	Dim	Yes	No	N/A
Off	On	Yes	Yes	No
Flicker	On	Yes	Yes	Yes

Note

The new LVDS probe will be supplied as standard with RealView ICE v3.0 units. The new probe can also be used with a RealView ICE v1.5 unit, updated with a firmware patch that provides support for the new probe. See *Installing an update or patch* on page 7-10.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

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.

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

Address breakpoint A type of breakpoint. *See* Breakpoint.

ADS *See* ARM Developer Suite.

Advanced High-performance Bus (AHB)
The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

See also Advanced Microcontroller Bus Architecture and AHB-Lite.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Advanced Microcontroller Bus Architecture (AMBA)
AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB conforms to this standard.

AFS ARM Firmware Suite.

AHB *See* Advanced High-performance Bus.

AHB-Lite
AHB-Lite is a subset of the full AHB specification. It is intended for use in designs where only a single AHB master is used. This can be a simple single AHB master system or a multi-layer AHB system where there is only one AHB master on a layer.

Angel Angel is a software debug monitor that runs on the target and enables you to debug applications running on ARM architecture-based hardware. Angel is commonly used where a JTAG emulator 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.

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.

See also Little-endian.

Breakpoint A user-defined point where execution stops so that a debugger can examine the state of memory and registers.

See also Hardware breakpoint and Software breakpoint.

Cache cleaning The process of writing *dirty data* in a cache to main memory.

See also Dirty data.

Complex Programmable Logic Device (CPLD)
A collection of PAL-type devices in a single package.

Context menu *See* Pop-up menu.

Coprocessor An additional processor that is used for certain operations, for example, for floating-point math calculations, signal processing, or memory management.

 Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

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

See also Integrator.

CoreSight Debug and real-time trace. The infrastructure for monitoring, tracing and debugging a complete system-on-chip.

CPLD *See* Complex Programmable Logic Device.

CPSR *See* Program Status Register.

CPU Central Processor Unit.

Current Program Status Register (CPSR)
See Program Status Register.

DAP *See* Debug Access Port.

Data breakpoint A location in the image that is monitored. If the value stored there is accessed in a specific way, the debugger halts execution of the image.

See also Instruction breakpoint.

DCache Data cache.

Debug Access Port (DAP)
A TAP block that acts as an AMBA (AHB or AHB-Lite) master for access to a system bus. The DAP is the term used to encompass a set of modular blocks that support system-wide debug. The DAP is a modular component, intended to be extendable to support optional access to multiple systems such as memory mapped AHB and CoreSight APB through a single debug interface.

Debugger An application that monitors and controls the execution of a second application. It is usually used to find errors in the application program flow.

Dirty data When referring to a processor data cache, data that has been written to the cache but has not been written to main memory. Only write-back caches can have dirty data, because a write-through cache writes data to the cache and to main memory simultaneously. The process of writing dirty data to main memory is called *cache cleaning*.

See also Cache cleaning.

DLL *See* Dynamic Linked Library.

Double word A 64-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Dynamic Linked Library
A collection of programs, any of which can be called when needed by an executing program. A small program that helps a larger program communicate with a device, such as a printer or keyboard, is often packaged as a DLL.

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.

See also IEEE1149.1-2001 and In-Circuit Emulator.

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

Embedded Trace Macrocell (ETM)
The Embedded Trace Macrocell is the logic inside the core that communicates details of program execution to the external trace port.

Endpoint connection
A debug target processor, normally accessed through an *access-provider connection*.

Environment The actual hardware and operating system that an application runs on.

ETB *See* Embedded Trace Buffer.

ETM *See* Embedded Trace Macrocell.

Flash memory Nonvolatile memory that is often used to hold application code.

Halfword A 16-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

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

See also Breakpoint and Software breakpoint.

Host A computer that provides data and other services to another computer. *Especially*, a computer providing debugging services to a target being debugged.

IC Integrated Circuit.

ICache Instruction cache.

ID Identifier.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

IEEE 1149.1-2001 The IEEE Standard that defines TAP. Commonly, but incorrectly, referred to as JTAG.

See also Test Access Port

Image An executable file that has been loaded onto a processor for execution.

Instruction breakpoint
A location in the image that is monitored. If execution reaches this location, the debugger halts execution of the image.

See also Data breakpoint.

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

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

IR *See* Instruction Register.

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 1149.1-2001 (JTAG).

JTAG *See* Joint Test Action Group.

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.

LSI Large Scale Integration.

LVDS Low Voltage Digital Signaling.

Memory Management Unit (MMU)
Hardware that controls caches and access permissions to blocks of memory, and translates virtual to physical addresses.

MMU *See* Memory Management Unit.

MPU Multi-Processor Unit.

Multi-ICE A JTAG-based tool for debugging embedded systems.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

nSRST Abbreviation of *System Reset*. The electronic signal that causes the target system other than the TAP controller to be reset. This signal is known as **nSYSRST** in some other manuals.

See also **nTRST**.

nTRST Abbreviation of *TAP Reset*. The electronic signal that causes the target system TAP controller to be reset. This signal is known as **nICERST** in some other manuals.

See also **nSRST**.

Open collector A signal that can be actively driven LOW by one or more drivers, and is otherwise passively pulled HIGH. Also known as a 'wired AND' signal.

PCB Printed Circuit Board

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.

Processor Status Register
See Program Status Register.

Program image *See* Image.

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), that records information about an alternate processor mode.

PSR *See* Program Status Register.

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

RealView ICE RealView EmbeddedICE interface.

RealView Trace Provides tracing functionality for RealView ICE.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

Remapping Changing the address of physical memory or devices after the application has started executing. This is typically done to enable RAM to replace ROM when the initialization has been done.

RTCK Returned **TCK**. The signal that enables Adaptive Clocking.

RTOS Real Time Operating System.

RVCT *See* RealView Compilation Tools.

RVD *See* RealView Debugger

Saved Program Status Register (SPSR)
See Program Status Register.

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.

Semihosting A mechanism where I/O requests made in the application code are communicated to the host system, rather than being executed on the target.

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

See also Breakpoint and Hardware breakpoint.

SPSR *See* Program Status Register.

Supervisor Call (SVC)
An instruction that interrupts the program being executed, and passes control to the supervisor.

SVC *See* Supervisor Call.

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 *See* Test Access Port.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.

TAP Controller Logic on a device that enables access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1-2001.

See also Test Access Port and IEEE1149.1-2001.

Target The target hardware, including processor, memory, and peripherals, real or simulated, on which the target application is running.

TCK The electronic clock signal that times data on the TAP data lines **TMS**, **TDI**, and **TDO**.

TDI The electronic signal input to a TAP controller from the data source (upstream). Usually this is seen connecting the RealView ICE run control unit to the first TAP controller.

TDO The electronic signal output from a TAP controller to the data sink (downstream). Usually this is seen connecting the last TAP controller to the RealView ICE run control unit.

Test Access Port (TAP)
The collection of four mandatory and one optional terminals that form the input/output and control interface to a JTAG boundary-scan architecture. The mandatory terminals are **TDI**, **TDO**, **TMS**, and **TCK**. The optional terminal is **nTRST**. This signal is mandatory in ARM cores because it is used to reset the debug logic.

TMS Test Mode Select.

TPA *See* Trace Port Analyzer.

Trace funnel A device that combines multiple trace sources onto a single bus.

Trace Port Analyzer (TPA)
A logic analyzer that can capture the details of program execution in real time. RealView Trace is the ARM trace port analyzer.

Trace Port Interface Unit (TPIU)
A trace sink used to drain trace data, and acts as a bridge between the on-chip trace data and the data stream captured by a TPA.

TTL Transistor-transistor logic. A type of logic design in which two bipolar transistors drive the logic output to one or zero. LSI and VLSI logic often used TTL with HIGH logic level approaching +5V and LOW approaching 0V.

VLSI Very Large Scale Integration.

Word A 32-bit unit of information. Contents are taken as being an unsigned integer unless otherwise stated.

Copyright © 2002, 2004-2006 ARM Limited. All rights reserved.