ARM® DS-5™

Version 5

Using the Debug Hardware Configuration Utilities

ARM

Copyright © 2010-2012, 2015 ARM. All rights reserved.
ARM DUI0498J

ARM® DS-5™

Using the Debug Hardware Configuration Utilities

Copyright © 2010-2012, 2015 ARM. All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
A	May 2010	Non-Confidential	First release
B	November 2010	Non-Confidential	Second Release
C	30 April 2011	Non-Confidential	DSTREAM and RVI v4.2.1 Release
D	29 July 2011	Non-Confidential	Update 1 for DSTREAM and RVI v4.2.1
E	30 September 2011	Non-Confidential	DSTREAM and RVI v4.4 Release
F	29 February 2012	Non-Confidential	Update for DS-5 version 5.9
G	29 July 2012	Non-Confidential	Update for DS-5 version 5.11
H	12 October 2012	Non-Confidential	Update for DS-5 version 5.12
I	13 July 2015	Non-Confidential	Update for DS-5 version 5.22
J	15 October 2015	Non-Confidential	Update for DS-5 version 5.23

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of ARM. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, ARM makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to ARM’s customers is not intended to create or refer to any partnership relationship with any other company. ARM may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any signed written agreement covering this document with ARM, then the signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited or its affiliates in the EU and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow ARM's trademark usage guidelines at *http://www.arm.com/about/trademark-usage-guidelines.php*

Copyright © [2010-2012, 2015], ARM Limited or its affiliates. All rights reserved.

ARM Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

http://www.arm.com

Conformance Notices

This section contains conformance notices.

Federal Communications Commission Notice

This device is test equipment and consequently is exempt from part 15 of the FCC Rules under section 15.103 (c).

Class A

Important: This is a Class A device. In residential areas, this device may cause radio interference. The user should take the necessary precautions, if appropriate.

CE Declaration of Conformity

CE

The system should be powered down when not in use.

It is recommended that ESD precautions be taken when handling DSTREAM, RVI, and RVT equipment.

The DSTREAM, RVI, and RVT modules generate, use, and can radiate radio frequency energy and may cause harmful interference to radio communications. There is no guarantee that interference will not occur in a particular installation. If this equipment causes harmful interference to radio or television reception, which can be determined by turning the equipment off or on, you are encouraged to try to correct the interference by one or more of the following measures:

- ensure attached cables do not lie across the target board
- reorient the receiving antenna
- increase the distance between the equipment and the receiver
- connect the equipment into an outlet on a circuit different from that to which the receiver is connected
- consult the dealer or an experienced radio/TV technician for help

Note

It is recommended that wherever possible shielded interface cables be used.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Contents

ARM® DS-5™ Using the Debug Hardware Configuration Utilities

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Preface

This preface introduces the *ARM® DS-5™ Using the Debug Hardware Configuration Utilities*.

It contains the following:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

About this book

This book describes how to configure a DSTREAM or RVI unit for use with an ARM® software development product, such as DS-5.

Using this book

This book is organized into the following chapters:

Chapter 1 Getting Started with the Debug Hardware Configuration Utilities
This chapter describes how to get started with using the debug hardware configuration utilities.

Chapter 2 Configuring Network Settings for your Debug Hardware Unit
This chapter describes how to configure the network settings for your debug hardware unit.

Chapter 3 Managing the Firmware on your Debug Hardware Unit
This chapter describes how to manage and update the software that is installed on the debug hardware unit, using the RVI Update utility.

Chapter 4 Creating Debug Hardware Target Configurations
This chapter describes how to create a debug hardware configuration file with the Debug Hardware Config utility for use by your debugger.

Chapter 5 Configuring CoreSight™ Systems
This chapter describes CoreSight systems and how to use Debug Hardware Config to configure them.

Chapter 6 Using Trace
This chapter describes the trace features supported by your trace hardware.

Chapter 7 Debugging with your Debug Hardware Unit
This chapter provides information about debugging with debug hardware.

Chapter 8 Troubleshooting your Debug Hardware Unit
This chapter describes what to do when you encounter problems when attempting to connect to a debug hardware unit or upgrade the firmware.

Glossary

The ARM Glossary is a list of terms used in ARM documentation, together with definitions for those terms. The ARM Glossary does not contain terms that are industry standard unless the ARM meaning differs from the generally accepted meaning.

See the *ARM Glossary* for more information.

Typographic conventions

italic
Introduces special terminology, denotes cross-references, and citations.

bold
Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

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

`mono`space
Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic`
Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold`
Denotes language keywords when used outside example code.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

<and>
: Encloses replaceable terms for assembler syntax where they appear in code or code fragments. For example:

```
MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
```

SMALL CAPITALS
: Used in body text for a few terms that have specific technical meanings, that are defined in the *ARM glossary*. For example, IMPLEMENTATION DEFINED, IMPLEMENTATION SPECIFIC, UNKNOWN, and UNPREDICTABLE.

Feedback

Feedback on this product

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

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

Feedback on content

If you have comments on content then send an e-mail to *errata@arm.com*. Give:

- The title *ARM® DS-5™ Using the Debug Hardware Configuration Utilities*.
- The number ARM DUI0498J.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

ARM tests the PDF only in Adobe Acrobat and Acrobat Reader, and cannot guarantee the quality of the represented document when used with any other PDF reader.

Other information

- *ARM Information Center*.
- *ARM Technical Support Knowledge Articles*.
- *Support and Maintenance*.
- *ARM Glossary*.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 1
Getting Started with the Debug Hardware Configuration Utilities

This chapter describes how to get started with using the debug hardware configuration utilities.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.1 About the debug hardware configuration utilities

The debug hardware configuration utilities enable you to connect to the debug hardware unit that provides the interface between your development platform and your PC.

The following utilities are provided:

RVI Config IP utility

Use this utility to configure the IP address on a debug hardware unit. This enables you to access the unit over Ethernet.

Debug Hardware Config utility

Use this utility to configure a debug hardware unit. This enables you to:

- Identify the target devices on your development platform. These devices can be one or more processors, and optional trace devices or CoreSight™ devices.
- Configure debug hardware and target-related features that are appropriate to correctly debug your development platform.
- Save the configuration to a device configuration file. The device configuration file is used by your debugger to connect to each target processor on your development platform.

RVI Update utility

Use this utility to update the firmware and devices on a debug hardware unit and probe.

Note

This document applies to the ARM® DSTREAM debug and trace unit and the ARM RVI debug unit. The term trace hardware refers to the built-in trace hardware of a DSTREAM unit.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.
1.8 Connecting to a debug hardware unit on page 1-19.

Related references

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.
Chapter 3 Managing the Firmware on your Debug Hardware Unit on page 3-35.
Chapter 4 Creating Debug Hardware Target Configurations on page 4-47.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.2 About starting the debug hardware configuration utilities

How you start the debug hardware configuration utilities depends on your host operating system.

Starting the utilities also depends on:

- Whether you installed the host software with an ARM software development product.
- Whether you are running a debugger session.

Note

For information about debugging with your own debugger, see your debugger documentation.

Related concepts

1.5 Accessing the Debug Hardware Config utility from your Debugger on page 1-15.

Related tasks

1.3 Starting the Debug Hardware Config utility on Windows on page 1-13.

1.4 Starting the Debug Hardware Config utility on Red Hat Linux on page 1-14.

Related information

Using the Debugger.

Debugger Command Reference.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.3 Starting the Debug Hardware Config utility on Windows

You can start any debug hardware configuration utility on a Windows platform.

Procedure

1. Select **Start** > **All Programs** > **ARM DS-5** > **Debug Hardware**.
2. Select the utility you want to use:
   - Select **Debug Hardware Config IP** to start the RVI Config IP utility.
   - Select **Debug Hardware Configuration** to start the Debug Hardware Config utility.
   - Select **Debug Hardware Update** to start the RVI Update utility.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.
1.8 Connecting to a debug hardware unit on page 1-19.

Related references

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.
Chapter 3 Managing the Firmware on your Debug Hardware Unit on page 3-35.
Chapter 4 Creating Debug Hardware Target Configurations on page 4-47.

Related information

Using the Debugger.
Debugger Command Reference.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.4 Starting the Debug Hardware Config utility on Red Hat Linux

To start a debug hardware configuration utility on a Red Hat® Linux platform, select the appropriate shortcut. The shortcut depends on the version of Red Hat Linux and the desktop environment that you are using.

You can start the debug hardware configuration utility from a desktop shortcut or from the command-line. If no desktop shortcut is available, at the command-line:

Procedure

1. Make sure that the paths to the utilities are configured. See *Getting Started for DS-5* for more details.
2. Enter the command for the utility you want to use:
   - Enter **rviconfigip** to start the RVI Config IP utility.
   - Enter **dbghwconfig** to start the Debug Hardware Config utility.
   - Enter **rviupdate** to start the RVI Update utility.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.
1.8 Connecting to a debug hardware unit on page 1-19.

Related references

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.
Chapter 3 Managing the Firmware on your Debug Hardware Unit on page 3-35.
Chapter 4 Creating Debug Hardware Target Configurations on page 4-47.

Related information

Using the Debugger.
Debugger Command Reference.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.5 Accessing the Debug Hardware Config utility from your Debugger

You can access the Debug Hardware Config utility directly from your debugger. See the DS-5 Debugger documentation for more details.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

Related information

Using the Debugger.

Debugger Command Reference.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.6 Scanning for available debug hardware units

You can use the Debug Hardware Config utility to search for debug hardware units that are connected to your local network or USB ports on your PC.

Procedure

1. Start the required configuration utility, for example Debug Hardware Config.
2.  Click the **Scan** button to scan for debug hardware units that are connected to your local network or to a USB port on your PC. The **Scan** button becomes animated to indicate that a scan is in progress.

Note

If some hardware units do not appear in the list, check your network and debug hardware setup.

When the configuration utility finds a unit, it adds it to the list of available units. The following figure shows an example:

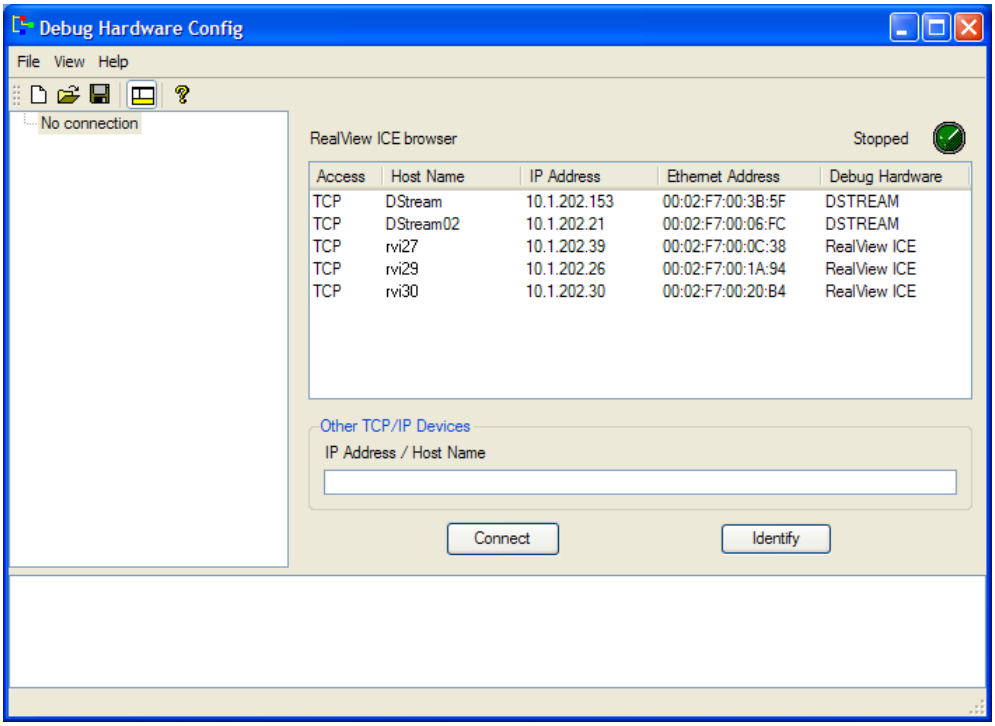

Figure 1-1 Debug Hardware Config utility

Note

Any unit that is shown in light gray is one that has responded to browse requests but does not have a valid IP address. You cannot connect to that unit by TCP/IP until you have configured it for use on your network.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

The scan tool searches for debug hardware units that are connected to your local network or USB ports on your PC. The units found are listed in the browser on the right of the window.

Note

Units that are connected to different networks do not appear in the configuration utility. Therefore, if you want to connect to a debug hardware unit on a separate network, you must know the IP address of that unit.

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

3. Select **File** > **Exit**. This disconnects from any connected unit, and exits the configuration utility.

Related concepts

8.5 Reasons why a debug hardware unit is not listed on page 8-135.

1.2 About starting the debug hardware configuration utilities on page 1-12.

Related tasks

1.7 Identifying a debug hardware unit on page 1-18.

1.8 Connecting to a debug hardware unit on page 1-19.

Related references

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.7 Identifying a debug hardware unit

If you have multiple debug hardware units on a network, you can identify the unit that you want to access from the configuration utility.

Procedure

1. Select a unit from the list, or enter an IP address of a unit if it is on a different network.
2. Click **Identify**.

   The identification indicators on the selected debug hardware unit flash for 5 seconds. If you have selected the wrong unit, select another unit from the list and repeat this step.

   ——— **Note** ———

   On RVI, all LEDs on the front panel flash during identification.

   On DSTREAM, the DSTREAM logo flashes during identification.

   ———————————

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.
1.8 Connecting to a debug hardware unit on page 1-19.

Related references

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.

Related information

The DSTREAM unit.
The RVI debug unit.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

1.8 Connecting to a debug hardware unit

You must connect to your debug hardware unit to create a configuration file that contains details of your target hardware and the debug hardware. The configuration file is required by your debugger to connect to your target development platform and debug your software.

Prerequisites

1. Set up, or have access to, the debug hardware unit that interfaces with your development platform.
2. Installed the correct version of the host software on your PC for your debug hardware unit.

Procedure

1. Start the required configuration utility, for example the Debug Hardware Config utility.
2.  If you do not see any debug hardware units listed for your local network, click the **Scan** button. The **Scan** button becomes animated to indicate that a scan is in progress. When the utility finds a debug hardware unit on your local network, it is added to the list of available units. The following figure shows an example:

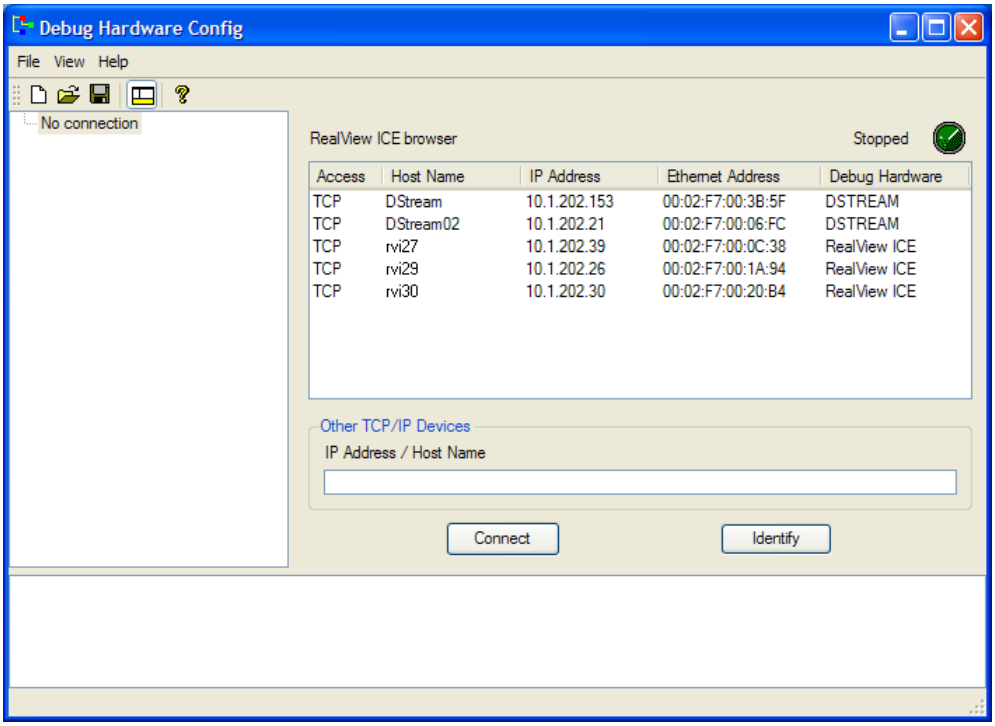

Figure 1-2 Debug Hardware Config utility showing debug hardware units

Note

The scan tool only searches for debug hardware units that are connected to your local network or USB ports on your PC. Therefore, units that are connected to a different network do not appear in the configuration utility. Therefore, if you want to connect to a debug hardware unit that is not accessible on your local network, ensure that you know the IP address of that debug hardware unit.

Any unit that is shown in light gray is one that has responded to browse requests but does not have a valid IP address. You cannot connect to that unit by TCP/IP until you have configured it for use on your network.

Alternatively, connect the debug hardware unit directly to your PC using a USB cable.

3. If multiple units are listed, and you are unsure about the debug hardware unit you want to use:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

a. Select a unit in the list, or enter an IP address of a unit on a different network.
b. Click **Identify**. The identification indicators on the unit flash for five seconds.

Note

On RVI, all LEDs on the front panel flash during identification.

On DSTREAM, the DSTREAM logo flashes during identification.

4. To connect to your required unit, select the unit and click **Connect**. Alternatively, do one of the following:
   - Double-click on the unit you want to connect to.
   - In the IP Address/Host Name field, enter either the IP address or host name of the device you want to connect to and click **Connect**.

   When a connection has been established, the configuration utility display changes to show the configuration features provided by that utility.

   If you have problems connecting to a debug hardware unit, you must troubleshoot the debug hardware connections.
5. Select **File** > **Exit**. This disconnects from any connected unit, and exits the configuration utility.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.

Related references

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.
Chapter 8 Troubleshooting your Debug Hardware Unit on page 8-130.

Related information

The DSTREAM unit.
The RVI debug unit.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 2
Configuring Network Settings for your Debug Hardware Unit

This chapter describes how to configure the network settings for your debug hardware unit.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.1 About configuring network settings

The configuration process depends on how the debug hardware unit is connected to the host computer, and whether your network uses *Dynamic Host Configuration Protocol* (DHCP).

If you have connected your debug hardware 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.

The following connections are possible:

- Your debug hardware unit is connected to your local network that uses DHCP. In this situation, you do not have to know the Ethernet address of the unit, but you must enable DHCP.
- Your debug hardware unit is connected to your local network that does not use DHCP. In this situation, you must assign a static IP address to the debug hardware unit.

Note

If you have connected your debug hardware 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.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
2.5 Debug hardware unit network settings on page 2-28.

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.

Related references

2.2 Debug hardware network settings on page 2-23.
Chapter 8 Troubleshooting your Debug Hardware Unit on page 8-130.

Related information

Connecting the DSTREAM unit.
Connecting the RVI hardware.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.2 Debug hardware network settings

Before you can configure the network settings, you must first determine the correct network settings for your debug hardware unit. To do this, you must consult with the system administrator for your network.

The information that you require depends on whether your network uses *Dynamic Host Configuration Protocol* (DHCP):

Table 2-1 Required debug hardware network settings

Information	Using DHCP	Not using DHCP
Host Name	Yes	Yes
IP Address	-	Yes
Default Gateway	-	Yes
Subnet Mask	-	Yes
Ethernet Address	Yes	Yes
Ethernet Type	Yes	Yes

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
2.3 The Configure Debug Hardware unit dialog box on page 2-24.
2.4 The Configure new Debug Hardware unit dialog box on page 2-26.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.3 The Configure Debug Hardware unit dialog box

The Configure Debug Hardware unit dialog box enables you to modify the network settings on a debug hardware unit that has previously been configured.

The following figure shows an example:

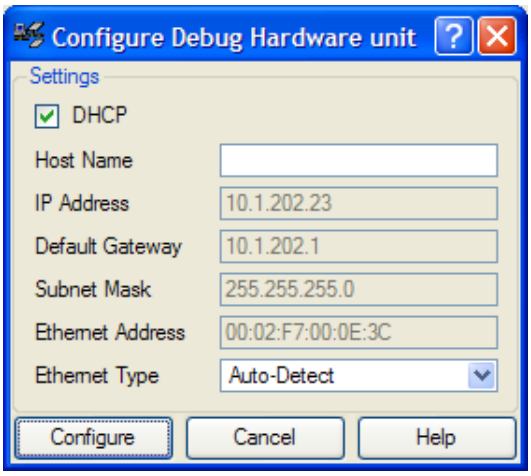

Figure 2-1 The Configure Debug Hardware unit dialog box

Note

You can modify the settings only for a debug hardware unit that is on your local network or that is connected to a USB port on your PC.

The network settings available depend on whether your network uses *Dynamic Host Configuration Protocol* (DHCP).

If your network uses DHCP, you must know:

- The hostname that you want to use for your unit (if any).
- The Ethernet type of your network.

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

- The hostname that you want to use for your unit (if any).
- The IP address that you want to use for your unit.
- The default gateway for your network (if it has one).
- The subnet mask for your network.
- The Ethernet type of your network.

Note

The Ethernet Address field is read-only.

After setting up the network settings, click **Configure** to write the values to the unit.

Click **Exit** to close the Configure Debug Hardware unit dialog box.

Related concepts

2.1 About configuring network settings on page 2-22.
2.9 Troubleshooting network settings for a debug hardware unit on page 2-34.
2.4 The Configure new Debug Hardware unit dialog box on page 2-26.
2.5 Debug hardware unit network settings on page 2-28.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.

2.7 Modifying the network settings for a debug hardware unit on page 2-31.

Related references

2.2 Debug hardware network settings on page 2-23.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.4 The Configure new Debug Hardware unit dialog box

The Configure new Debug Hardware unit dialog box enables you to configure the network settings for a debug hardware unit.

You can configure the settings of:

- A unit that has not been previously configured.
- A unit that is on a different subnet.

The following figure shows an example:

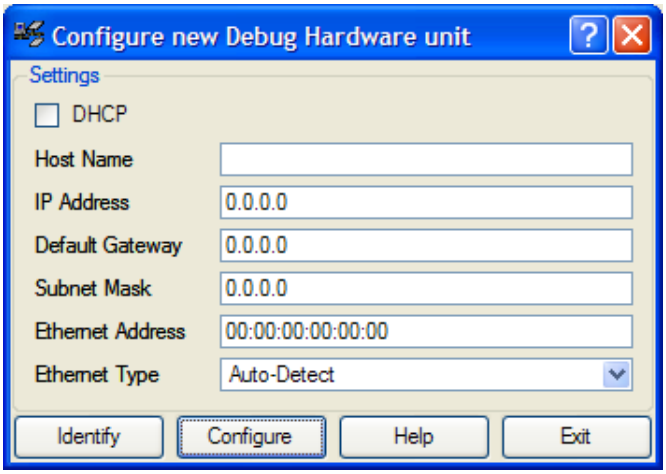

Figure 2-2 The Configure new Debug Hardware unit dialog box

The network settings available depend on whether your network uses *Dynamic Host Configuration Protocol* (DHCP).

If your network uses DHCP, you must know:

- The hostname that you want to use for your unit (if any).
- The Ethernet address of unit.
- The Ethernet type of your network.

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

- The hostname that you want to use for your unit (if any).
- The IP address that you want to use for your unit.
- The default gateway for your network (if it has one).
- The subnet mask for your network.
- The Ethernet address of unit.
- The Ethernet type of your network.

If more than one unit is listed in the browser, click **Identify** to identify your unit. The Identification LEDs on the selected unit flash for five seconds.

After setting up the network settings, click **Configure** to write the values to the unit.

Click **Exit** to close the Configure new Debug Hardware unit dialog box.

Related concepts

2.1 About configuring network settings on page 2-22.
2.5 Debug hardware unit network settings on page 2-28.

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related references

2.2 Debug hardware network settings on page 2-23.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.5 Debug hardware unit network settings

Several network settings are available for a debug hardware unit such as, DHCP, hostname, and IP address.

DHCP

Enables or disables *Dynamic Host Configuration Protocol* (DHCP):

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

  ——— **Note** ———

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

  ———————————

- If your network does not use DHCP, you must know:
  - The hostname to use for your debug hardware unit (if any).
  - The IP address to use for your debug hardware unit.
  - The default gateway for your network (if it has one).
  - The subnet mask for your network.

Host Name

The host name for the unit. This must contain only the alphanumeric characters (A to Z, a to z, and 0 to 9) and the - character, and must be no more than 39 characters long.

IP Address

The static IP address to use when DHCP is disabled.

Default Gateway

The default gateway to use when DHCP is disabled.

Subnet Mask

The subnet mask to use when DHCP is disabled.

Ethernet Address

The Ethernet address of the unit.

Ethernet Type

The type of network you are using:

- If you know the type of network, select the type. The options are:
  - **10-MBit, Half Duplex**
  - **10-MBit, Full Duplex**
  - **100-MBit, Half Duplex**
  - **100-MBit, Full Duplex**.
- Otherwise, select **Auto-Detect**.

Related concepts

2.1 About configuring network settings on page 2-22.

2.9 Troubleshooting network settings for a debug hardware unit on page 2-34.

2.3 The Configure Debug Hardware unit dialog box on page 2-24.

2.4 The Configure new Debug Hardware unit dialog box on page 2-26.

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.6 Configuring the network settings for a debug hardware unit

If you have a debug hardware unit that does not have a valid IP address, or is on a different network, you must manually enter the Ethernet address during configuration.

Prerequisites

Before you can configure the network settings, you must first determine the correct network settings for your debug hardware unit:

- If you do not want to use DHCP, then you must obtain an IP Address, Default Gateway, and Subnet Mask from your network administrator.
- If you want to use DHCP, you must inform your network administrator of the Ethernet Address of the unit, so that it can be added to the DHCP server.

Procedure

1. Open the RVI Config IP utility.
2. If the debug hardware unit is on your local network or connected to a USB port on your PC, continue at step 3.

   Otherwise, continue at step 6.
3.  Click the **Scan** tool to scan for debug hardware units.

   ———— **Note** ————

   Only debug hardware units that are on your local network or connected to a USB port on your PC are listed.

   ————————————
4. Select the debug hardware unit that you want to configure.
5.  Click the **Identify** tool to verify that the identification LEDs flash on the correct debug hardware unit.
6.  Click the **Config New** tool. The Configure new Debug Hardware unit dialog box appears, as shown in the following figure:

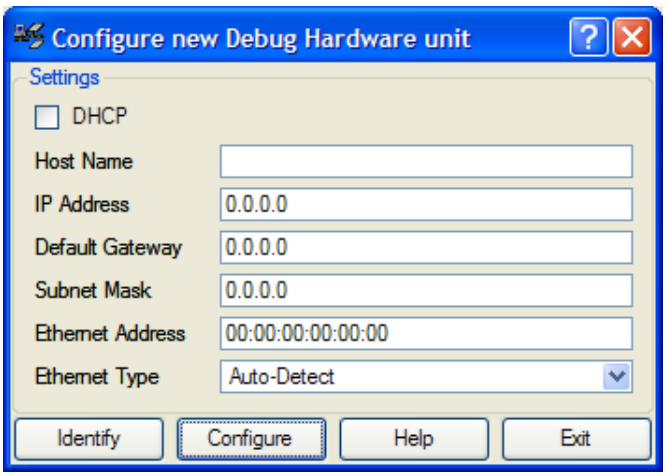

Figure 2-3 The Configure new Debug Hardware unit dialog box

7. Determine the Ethernet address of your debug hardware unit by reading the label on the side of the unit, and enter it into the Ethernet Address field.
8. If you are not using DHCP:
   a. Deselect **DHCP**.
   b. Enter the required details in the following fields:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

- IP Address
- Default Gateway
- Subnet Mask.

c. Continue at step 9.

9. If you are using DHCP, select **DHCP**.
10. 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.
11. Select the required Ethernet Type:
    - If you know the type of network that you are using, select that type.
    - Otherwise, select **Auto-Detect**.
12. Click **Configure**.

    The debug hardware unit restarts. During the restart, the unit is removed from the list of units. When the restart is complete, the unit re-appears in the list of units, with the new network settings.

    Note

    If the debug hardware unit is using DHCP, the list of units might display its **IP Address** as `127.0.0.2`. This is a dummy address that the debug hardware 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.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
2.9 Troubleshooting network settings for a debug hardware unit on page 2-34.
2.1 About configuring network settings on page 2-22.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.

Related references

2.2 Debug hardware network settings on page 2-23.
Chapter 8 Troubleshooting your Debug Hardware Unit on page 8-130.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.7 Modifying the network settings for a debug hardware unit

You can modify the network settings of a debug hardware unit only if that unit is on your local network or connected to a USB port on your PC. If the debug hardware unit is on a different network, you must use the Configure New Debug Hardware unit dialog box.

Prerequisites

Before you can configure the network settings, you must first determine the correct network settings for your debug hardware unit:

- If you do not want to use DHCP, then you must obtain an IP Address, Default Gateway, and Subnet Mask from your network administrator.
- If you want to use DHCP, you must inform your network administrator of the Ethernet Address of the unit, so that it can be added to the DHCP server.

Procedure

1. Open the RVI Config IP utility.
2.  Click the **Scan** tool to scan for debug hardware units.
3. Select the debug hardware unit that you want to modify.
4.  Click the **Identify** tool to verify that the identification LEDs flash on the correct debug hardware unit.
5. Click the **Configure** tool to display the Configure Debug Hardware unit dialog box. An example is shown in the following figure:

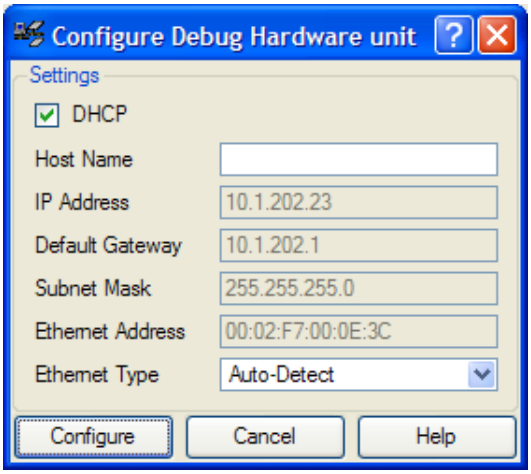

Figure 2-4 The Configure Debug Hardware unit dialog box

Note

The Ethernet Address field is read-only.

6. If you are not using DHCP:
   a. Deselect **DHCP**.
   b. Enter the required details in the following fields:
      - IP Address.
      - Default Gateway.
      - Subnet Mask.
   c. Continue at step 9.
7. If you are using DHCP, select **DHCP**.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8. 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.
9. Select the required Ethernet Type:
   - If you know the type of network that you are using, select that type.
   - Otherwise, select **Auto-Detect**.
10. Click **Configure**.

    The debug hardware unit restarts. During the restart, the unit is not present in the list. When the unit has restarted, it re-appears in the list of units with the new network settings.

    Note

    If the debug hardware unit is using DHCP, the list of units might display its **IP Address** as `127.0.0.2`. This is a dummy address that the debug hardware 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.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
2.9 Troubleshooting network settings for a debug hardware unit on page 2-34.
2.1 About configuring network settings on page 2-22.
2.3 The Configure Debug Hardware unit dialog box on page 2-24.
2.4 The Configure new Debug Hardware unit dialog box on page 2-26.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.
2.6 Configuring the network settings for a debug hardware unit on page 2-29.

Related references

2.2 Debug hardware network settings on page 2-23.
Chapter 8 Troubleshooting your Debug Hardware Unit on page 8-130.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.8 Restarting your debug hardware unit

The RVI Config IP utility restarts the networking software on the debug hardware unit whenever you change its settings.

If necessary, select **RVI** > **Restart** to force the networking software to restart.

You might want to restart the networking software on the debug hardware unit to get new network settings from the DHCP server. To do this, select **RVI** > **Restart**.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

2.9 Troubleshooting network settings for a debug hardware unit

There are solutions to common issues when configuring network settings for a debug hardware unit.

Why am I unable to see my DSTREAM or RVI unit on the network?

Seeing or browsing for DSTREAM or RVI units on the network relies on the local area network (LAN) allowing propagation of broadcast packets (UDP) on ports 30000 and 30001.

It is common to limit the propagation of these types of packets to a localized network region to prevent congestion, but it might be possible to allow propagation of packets on these specific ports. Contact your network administrator to request this modification.

Note

This issue is seen when a DSTREAM or RVI unit is behind any network component that filters network traffic, such as a firewall.

When is it appropriate to assign a fixed IP address to my DSTREAM or RVI unit?

If it is not possible to browse for the DSTREAM or RVI unit on the network using the tools, you can attempt to locate the unit by specifying the host name of the unit, for example `MyDSTREAM.local.example.com`.

If the host name cannot be resolved, you can use an IP address, for example `192.168.1.16`. In this case, you might want to assign a fixed IP address to the DSTREAM or RVI unit to prevent this IP address from changing. To request a fixed IP address, contact your network administrator. When the address is assigned to the DSTREAM or RVI unit, you can confirm its correct operation by using the ping command from a DOS or UNIX prompt before connecting the tools.

A fixed IP address is also appropriate when an Ethernet cross-over cable is used. In this case, a private network between the host PC and the unit is created, although this might not be necessary due to the availability of a USB connection.

Why does my debug connection fail when I connect the Mictor cable to my target?

Some target systems have their debug signals connected to both a Mictor trace connector and a separate debug-only connector. In this scenario, if you connect the Mictor cable alongside another debug cable, there is effectively a large unterminated stub on the debug signals. This can cause the debug interface to become unstable. To solve this problem, configure the DSTREAM or RVI software to use the Mictor cable for both the debug and trace signals, and disconnect any other debug cables.

Related concepts

2.3 The Configure Debug Hardware unit dialog box on page 2-24.
2.5 Debug hardware unit network settings on page 2-28.

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.
2.7 Modifying the network settings for a debug hardware unit on page 2-31.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 3
Managing the Firmware on your Debug Hardware Unit

This chapter describes how to manage and update the software that is installed on the debug hardware unit, using the RVI Update utility.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.1 About templates and firmware files

The debug hardware unit stores templates for each supported device. Each template defines how to communicate with the device and the settings that you can configure for that device. The templates are provided in a firmware file.

ARM periodically releases updates and patches to the firmware that is installed on a debug hardware unit. Each update or patch is released as a firmware file. The firmware filename has the following syntax:

`ARM-RVI-N.n.p-build-type.unit`

Where:

N.n.p is the version of the firmware. For example, 4.5.0 is the first release of firmware version 4.5.

build is a build number.

type is either:

`base`
: the first release of the firmware for version *N.n*

`patch`
: updates to the corresponding *N.n* release of the firmware.

unit identifies the debug hardware unit, and is one of:

- `dstream` for a DSTREAM debug and trace unit
- `rvi` for an RVI debug unit.

For example, patch 9 for DSTREAM firmware v4.5 is in the file:

`ARM-RVI-4.5.0-9-patch.dstream`

These might extend the capabilities of your debug hardware, or might fix an issue that has become apparent.

Related concepts

3.3 List of software version numbers on page 3-38.
3.2 Location of the firmware files in ARM® products on page 3-37.

Related tasks

3.5 Installing a firmware update or patch release on page 3-40.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.2 Location of the firmware files in ARM® products

You can obtain the firmware files from the ARM website or from the installed location at:
DS-5_install_directory\DS-5\sw\debughw\firmware

See *Getting Started with DS-5* for more information.

Related concepts

3.1 About templates and firmware files on page 3-36.

Related information

Getting Started with DS-5.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.3 List of software version numbers

Use the RVI Update utility to see the version numbers of the various debug software components.

To view the software version numbers in the RVI Update utility, select **RVI** > **Version Info...**. This shows the Version Info dialog box.

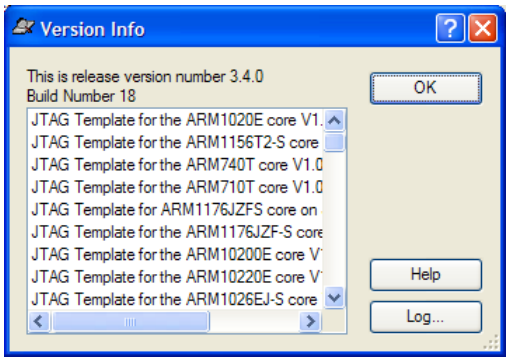

Figure 3-1 Example of version information

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
```

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.

Related tasks

3.5 Installing a firmware update or patch release on page 3-40.
3.4 Saving software version information to a file on page 3-39.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.4 Saving software version information to a file

You can save the software version information from the RVI Update utility to a file.

Procedure

1. Start the RVI Update utility.
2. Connect to the required debug hardware unit.
3. Select **RVI** > **Version Info...**.
4. Click **Log**. This displays the Select Log File Name dialog box.
5. Choose the location of the log file.

   The default filename is `rviversion`*`Nnnx`*`.log`, where *`Nnnx`* is the release version number. For example, 4100 is version 4.10.0.
6. Click **Save** to save the log file.
7. Click **OK** to close the Version Info dialog box.

Related concepts

3.3 List of software version numbers on page 3-38.

Related tasks

3.5 Installing a firmware update or patch release on page 3-40.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.5 Installing a firmware update or patch release

You can perform the same basic procedure for updating your firmware to a new version or applying a patch to the current version.

Prerequisites

If you are updating from a previous version of the firmware, for example from 4.4 to 4.5, you must upgrade to the base version first, then apply any patch that is related to that version. For example, if you have a DSTREAM unit:

1. Update your DSTREAM unit with `ARM-RVI-4.5.0-9-base.dstream`.
2. If a patch file is provided, such as `ARM-RVI-4.5.0-12-patch.dstream`, apply the patch to your DSTREAM unit.

If you want to restore the firmware to its original state after installing an upgrade, you can reinstall the original firmware file. This file is obtainable from the ARM website.

Note

If you have been running the Log Client utility, then make sure that you stop it before performing an update. Although the Log Client utility does not prevent the update from completing, it can take longer to complete.

Procedure

1. Select **Start** > **All Programs** > **ARM DS-5** > **Debug Hardware** > **Debug Hardware Update** to open the RVI Update utility.

   The RVI Update utility is displayed. An example is shown in the following figure:

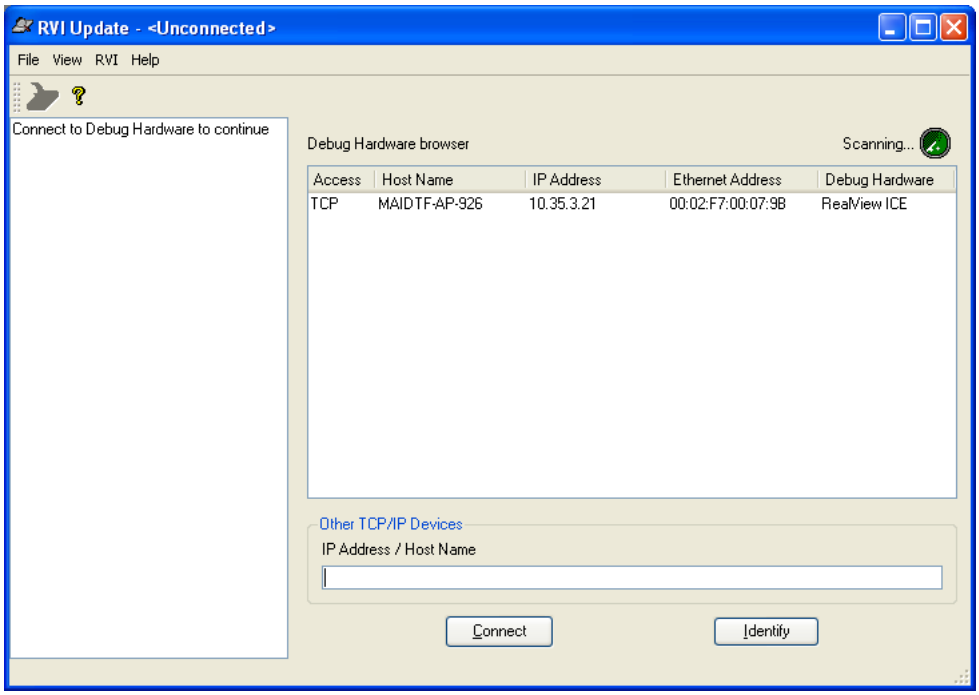

Figure 3-2 RVI Update utility

2. Either:
   - Select the debug hardware unit from the list.
   - Enter the IP address or host name of the debug hardware unit in the Other TCP/IP Devices field.
3. Click **Connect** to connect to the debug hardware unit. Details of the existing firmware on the unit are displayed. An example is shown in the following figure:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

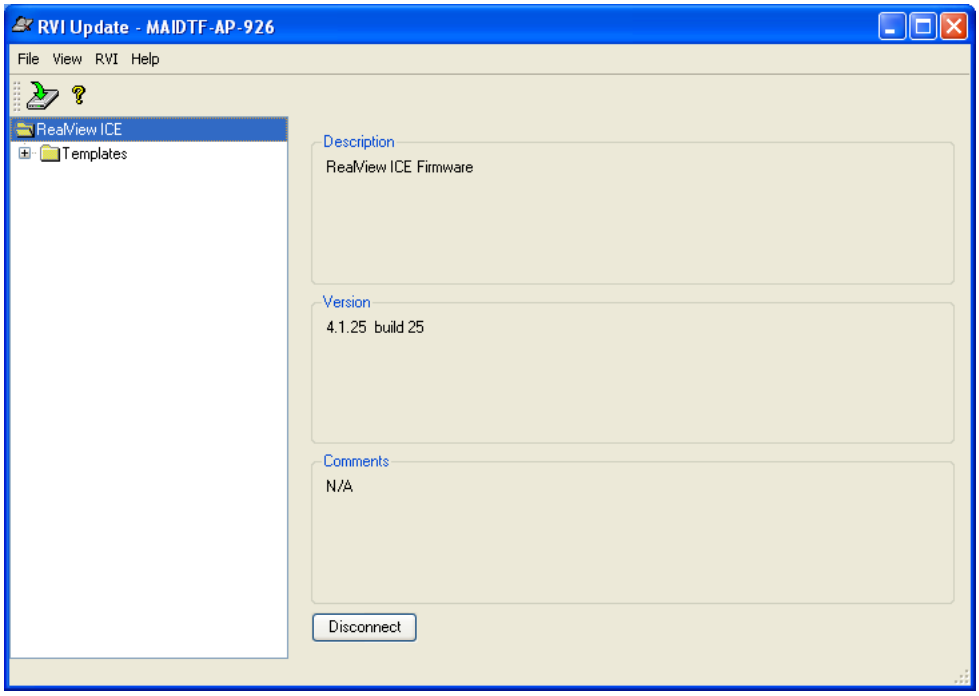

Figure 3-3 Firmware details

4.  In the RVI Update utility, click the **Install Firmware** tool. The Select Firmware Update to Install dialog box is displayed. An example is shown in the following figure:

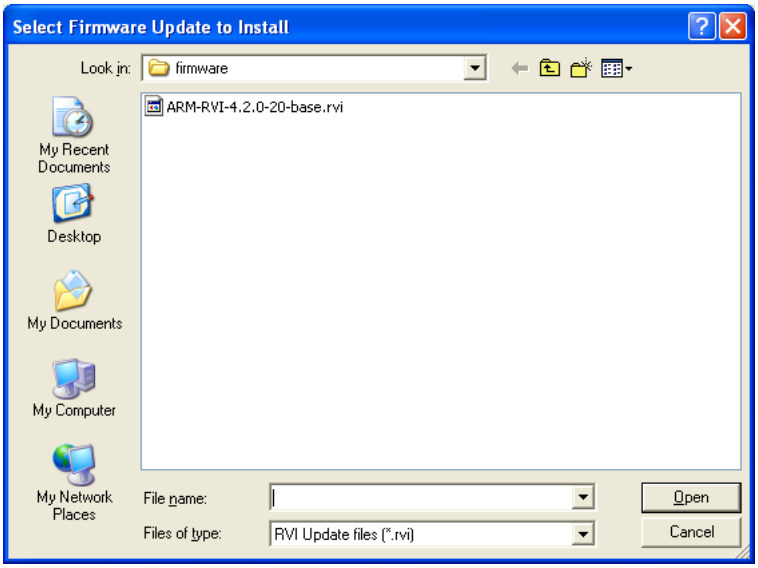

Figure 3-4 Selecting the component file to install

5. Navigate to the directory containing the component file for the update or patch that you want to install, and select the required file.
6. Click **Open**. After a short delay, a dialog box appears that describes what is in the component file, as shown in the following figure:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

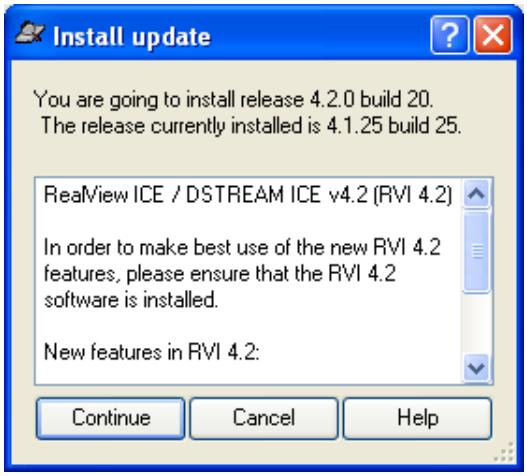

Figure 3-5 Confirming that you want to install the component file

When attempting to install a firmware update file, if you are using an older version of RVI Update, for example a pre-RVI v1.5 release, an error message appears as shown in the following figure:

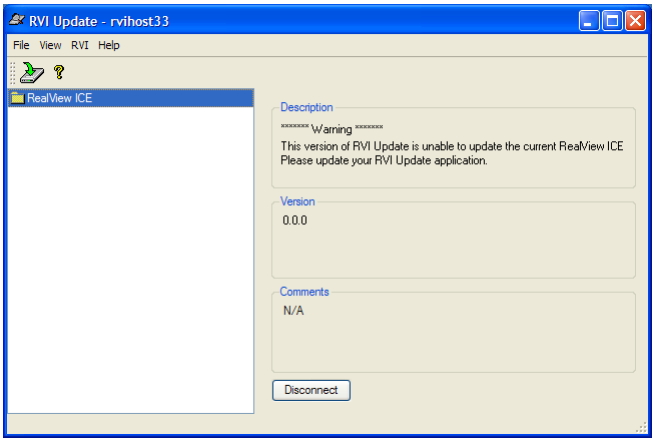

Figure 3-6 Warning message

Note

Before proceeding with your firmware update, you must upgrade your RVI Update utility to the latest software.

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 the following figure:

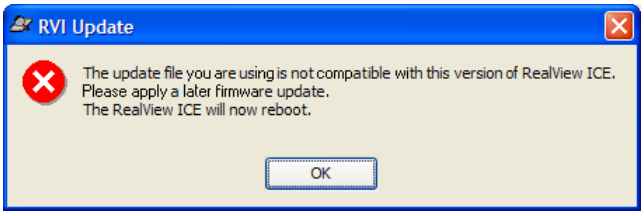

Figure 3-7 Error when using an incompatible version of hardware

7. In the Install update dialog box, click:
   - **Continue** to confirm that you want to install the components.
   - **Cancel** to make no change to the debug hardware unit.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

When you click **Continue**, the RVI Update utility uploads the component file to the debug hardware unit. The debug hardware unit unpacks the component file, and installs the update or patch that it contains. The progress of the installation is displayed as shown in the following figure:

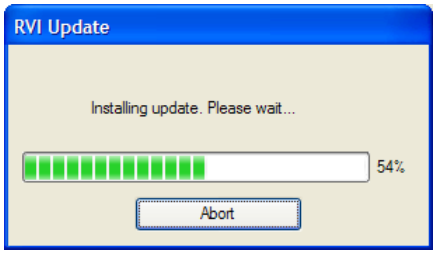

Figure 3-8 Progress during an installation

The debug hardware unit might automatically reboot itself as part of this procedure, depending on the patch or update that you are installing. The progress of the reboot is displayed as shown in the following figure:

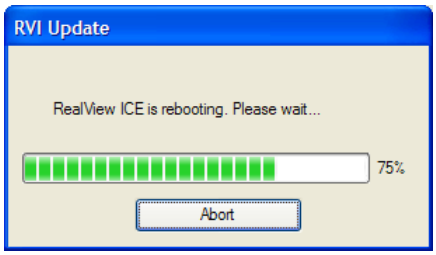

Figure 3-9 Progress when rebooting during an installation

During the installation, the FLASH LED lights up, showing that the unit is accessing its internal flash storage. During this time, do not disconnect power from the debug hardware unit. If a problem occurs during the installation, you must troubleshoot the firmware upgrade installation.

Note

During the installation, 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, you cannot stop the reboot.

When the installation is complete, a message is displayed as shown in the following figure:

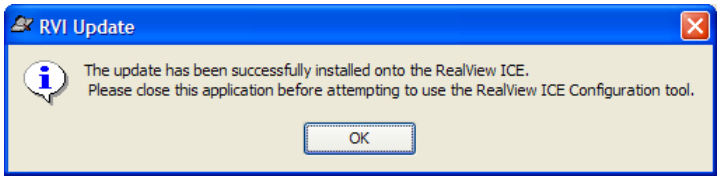

Figure 3-10 Message showing a successful installation

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
3.3 List of software version numbers on page 3-38.
3.1 About templates and firmware files on page 3-36.
3.2 Location of the firmware files in ARM® products on page 3-37.
8.8 Troubleshooting firmware upgrade installations on page 8-138.
8.10 Log Client Utility on page 8-141.

Related tasks

3.6 Upgrading an LVDS probe on page 3-45.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related information

ARM web site.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.6 Upgrading an LVDS probe

You can use the RVI Update utility to install an upgrade to your *Low Voltage Differential Signaling* (LVDS) probe. This upgrade procedure is necessary only if you want to use the *Serial Wire Debug* (SWD) feature.

This is a once-only upgrade that is required if your LVDS probe was released with RVI v3.0, because this type of probe is not capable of SWD.

To upgrade your LVDS probe:

Procedure

1. In the RVI Update utility, select **RVI** > **Upgrade LVDS Probe...**.
2. You are prompted to confirm your option to upgrade the probe. Select **Yes**, and the RVI Update utility begins the update process, during which you are reminded not to disconnect the probe, nor to power off your debug hardware unit, until the process is completed. This is shown in the following figure:

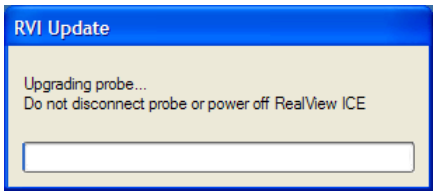

Figure 3-11 Progress during probe update

Note

To perform the upgrade, you must have v3.1 firmware or later installed on your debug hardware unit.

You must also have an LVDS probe that is at least at v2.

If you have a v1 probe (board number HPI-0090x), you must replace it with a later version. If so, contact ARM for more information.

Related concepts

8.8 Troubleshooting firmware upgrade installations on page 8-138.

Related tasks

3.5 Installing a firmware update or patch release on page 3-40.

Related information

ARM DSTREAM System and Interface Design Reference - Serial Wire Debug.

ARM RVI System and Interface Design Reference - Serial Wire Debug.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

3.7 Restarting the debug hardware unit in RVI Update

To restart the debug hardware unit, select **RVI** > **Restart**. RVI Update reboots the debug hardware unit, waits for the reboot to finish, then reconnects automatically. A message is displayed telling you that debug hardware is rebooting.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.

3.3 List of software version numbers on page 3-38.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.

3.5 Installing a firmware update or patch release on page 3-40.

3.6 Upgrading an LVDS probe on page 3-45.

Related references

Chapter 8 Troubleshooting your Debug Hardware Unit on page 8-130.

Chapter 2 Configuring Network Settings for your Debug Hardware Unit on page 2-21.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 4
Creating Debug Hardware Target Configurations

This chapter describes how to create a debug hardware configuration file with the Debug Hardware Config utility for use by your debugger.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.1 About creating debug hardware target configurations

A debug hardware target configuration enables your debugger to connect to the target devices and debug applications on your development platform.

You save your debug hardware target configuration in a configuration file. You reference this configuration file when you create target connections in your debugger.

This section contains the following subsections:

4.1.1 Creating a debug hardware configuration file

You can create a debug hardware configuration file using the Debug Hardware Config utility.

Procedure

1. Start the Debug Hardware Config utility. An example is shown in the following figure:

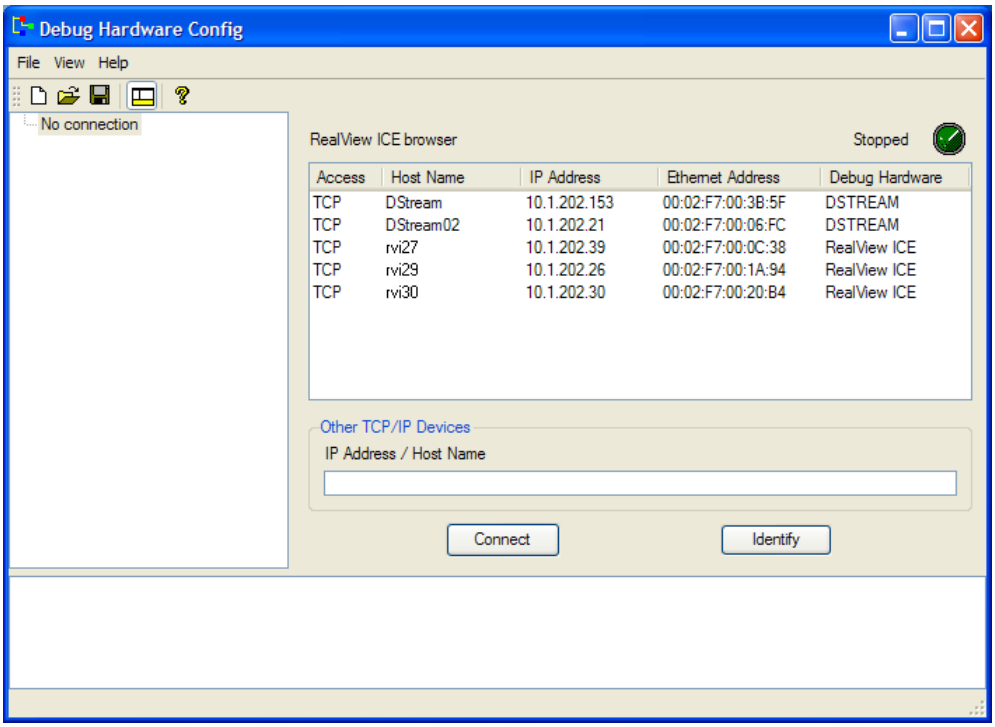

Figure 4-1 Debug Hardware Config utility

2. Connect to a debug hardware unit.
3. Click **Auto Configure** to identify the target devices on your development platform.
4. If a platform configuration exists for your development platform, you are prompted to select that platform configuration.

   Click **OK** to use the platform configuration. Otherwise, click **Cancel**.
5. Configure the following as required:
   - Device configuration settings.
   - Debug hardware Advanced settings.
   - Device properties, if appropriate.
   - Trace configuration settings (DSTREAM only).

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

6. Select **File** > **Save** to save your configuration. The Choose a filename to save as dialog box is displayed.
7. Locate a directory to save your configuration file and enter an appropriate filename. For example, **CoreSight_A8.rvc** for a CoreSight system containing a Cortex®-A8 processor.
8. Click **Save**.
9. Select **File** > **Exit** to close the Debug Hardware Config utility.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
4.2 About configuring a JTAG scan chain on page 4-51.
4.3 About configuring a device list on page 4-54.
5.3 About CoreSight™ autodetection on page 5-97.
4.11 About Adaptive clocking on page 4-70.
5.1 About CoreSight™ system configuration on page 5-95.
5.3 About CoreSight™ autodetection on page 5-97.
4.15 About platform detection and selection on page 4-82.
4.5 Select Platform dialog box on page 4-62.
4.8 About device properties on page 4-65.
4.12 About debug hardware device configuration settings on page 4-71.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.
4.9 Changing the properties of a device on page 4-67.
4.15.6 Configuring the debug hardware Advanced settings on page 4-87.

Related references

4.12.3 Debug hardware Advanced configuration settings on page 4-77.
4.12.4 Debug hardware Trace configuration settings on page 4-79.
Chapter 8 Troubleshooting your Debug Hardware Unit on page 8-130.

4.1.2 Opening an existing debug hardware configuration file in Debug Hardware Config

You can open a debug hardware configuration (`.rvc` file) in the Debug Hardware Config utility. After opening, the scan chain configuration is displayed.

Procedure

1. Select **File** > **Open**, and the Choose a file to open dialog box appears.
2. Locate the directory continuing your `.rvc` configuration files.
3. Select the appropriate `.rvc` file.
4. Click **Open**. The scan chain configuration is displayed, as shown in the following figure:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

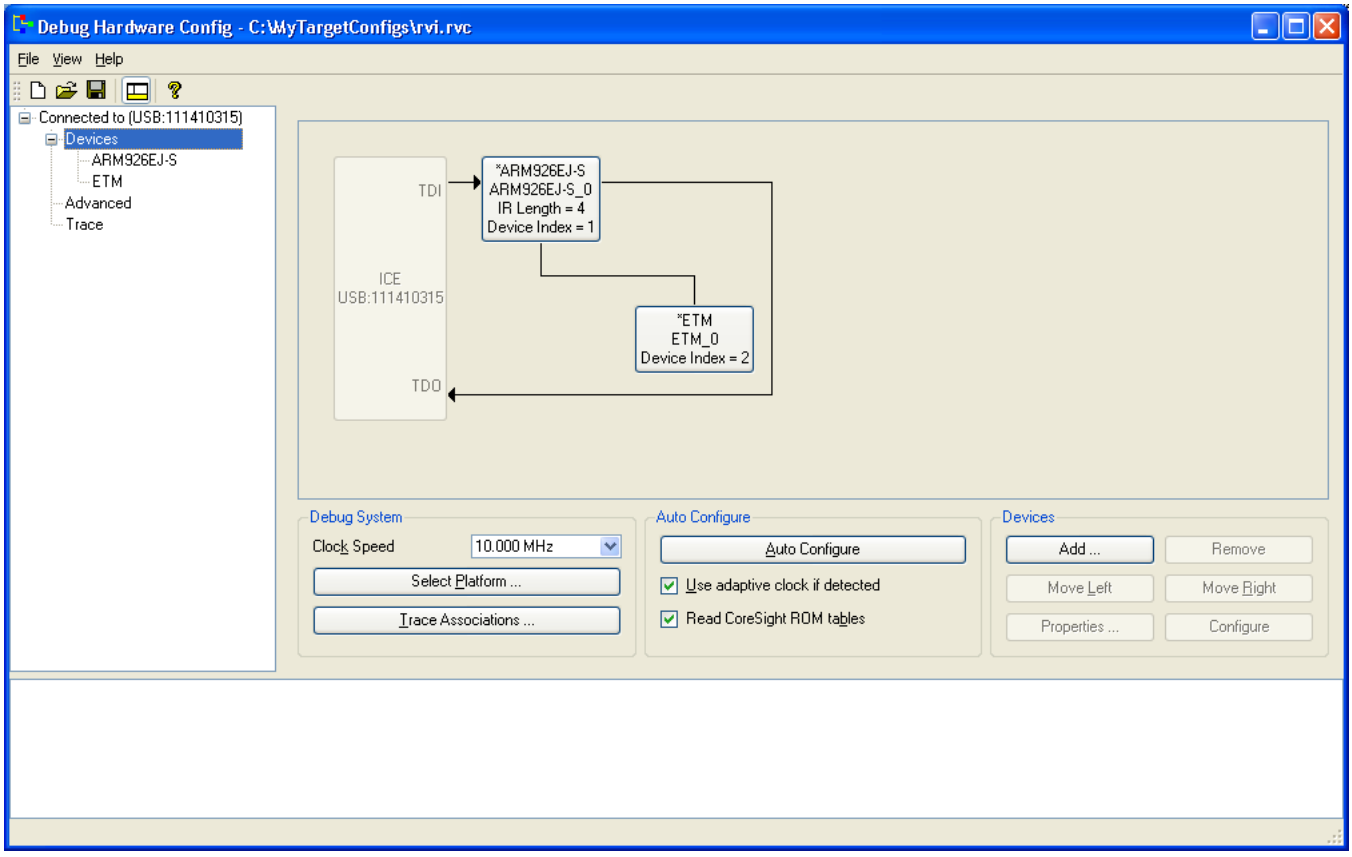

Figure 4-2 The scan chain controls

The title of the utility includes the full path to the configuration file. The path name might be different to that shown.

Note

The Trace Associations feature is deprecated.

5. Modify your configuration as required.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
4.1 About creating debug hardware target configurations on page 4-48.

Related concepts

5.1 About CoreSight™ system configuration on page 5-95.

Related tasks

4.1.1 Creating a debug hardware configuration file on page 4-48.

Related references

Chapter 5 Configuring CoreSight™ Systems on page 5-94.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.2 About configuring a JTAG scan chain

Use the scan chain controls to configure a scan chain for the currently connected debug hardware unit. As you add devices to the JTAG scan chain, a schematic diagram of the scan chain is created.

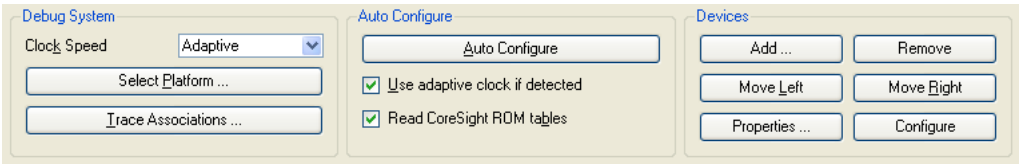

Figure 4-3 Scan chain controls

Note

The Trace Associations feature is deprecated.

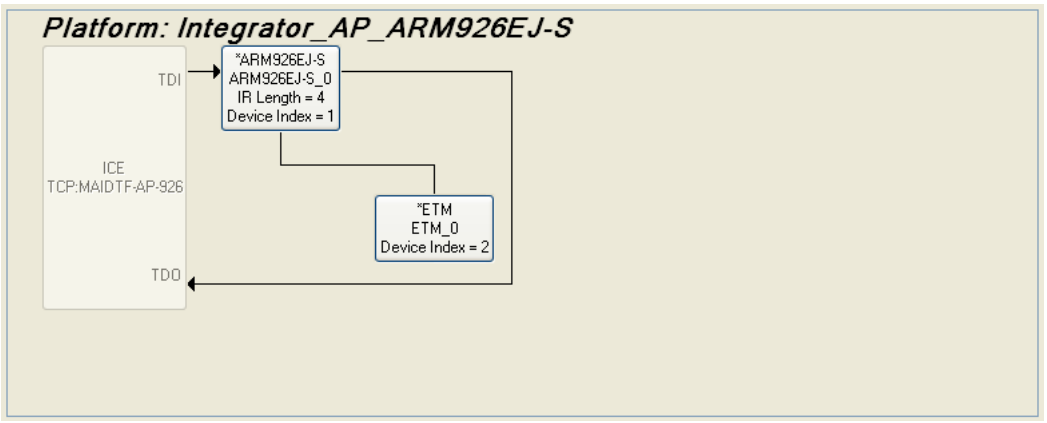

Figure 4-4 Scan chain schematic diagram

If a platform configuration file exists for your target, the Select Platform dialog box is displayed when you connect the debug hardware unit.

If you want to use the platform configuration file:

1. Select the platform.
2. Click **OK**. A label identifying the platform is included at the top of the schematic diagram.

This section contains the following subsections:

- *4.2.1 Controls available to manage devices* on page 4-51.
- *4.2.2 Device context menu controls* on page 4-52.
- *4.2.3 Managing a platform file* on page 4-52.

4.2.1 Controls available to manage devices

You can manage the devices in your configuration using the available controls.

- Click **Add...** to add a device to the scan chain.
- When a device is selected in the schematic diagram:
  - Click **Remove** to remove the selected device.
  - Click **Properties...** to update the properties for the selected device.
  - Click **Configure** to display the Device configuration settings.
- For a scan chain containing multiple devices, click:
  - **Move Left** to move the selected device to the left.
  - **Move Right** to move the selected device to the right.

Related concepts

4.4 About the scan chain on page 4-58.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.10 About setting the clock speed on page 4-68.
4.3 About configuring a device list on page 4-54.
4.5 Select Platform dialog box on page 4-62.
4.8 About device properties on page 4-65.
4.12 About debug hardware device configuration settings on page 4-71.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.
4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.
4.9 Changing the properties of a device on page 4-67.

4.2.2 Device context menu controls

To display the device context menu, right-click on a device in the scan chain schematic.

The options available for all devices are:

- Select **Properties...** to change the properties of the device.
- Select **Configuration...** to configure the device.
- Select **Remove Device** to remove the chosen device.

For a CoreSight system, the **Read CoreSight ROM table** option is available for the ARMCS-DP device. This enables your debug hardware to read the CoreSight ROM table.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.3 About configuring a device list on page 4-54.
4.5 Select Platform dialog box on page 4-62.
4.8 About device properties on page 4-65.
4.12 About debug hardware device configuration settings on page 4-71.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.
4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.
4.9 Changing the properties of a device on page 4-67.

4.2.3 Managing a platform file

You can select or clear a platform configuration file for your development board.

If a platform file exists, click **Select Platform...** to choose a platform that corresponds to your development board.

When a platform is assigned to your configuration, the **Select Platform...** button changes to **Clear Platform**.

Click **Clear Platform** to remove the platform assignment from your configuration.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.3 About configuring a device list on page 4-54.
4.5 Select Platform dialog box on page 4-62.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.8 About device properties on page 4-65.
4.12 About debug hardware device configuration settings on page 4-71.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.
4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.
4.9 Changing the properties of a device on page 4-67.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.3 About configuring a device list

You can either autoconfigure or manually configure a device list.

This section contains the following subsections:

4.3.1 About autoconfiguring a scan chain

When autoconfiguring a device list, debug hardware interrogates the scan chain and automatically selects the correct templates for supported ARM target devices, then adds them to the scan chain in the correct order.

This takes place at the current clock speed:

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

Warning

Autoconfiguring can be intrusive and stop your development platform from operating normally. If you want to connect to a target on your development platform without performing a reset and stop, you must manually add the devices to the scan chain.

Note

Autoconfiguration is disabled in Debug Hardware Config if the current debug hardware configuration has a platform that is assigned to it.

For *Serial Wire Debug* (SWD), autoconfiguring a system identifies the target devices on your development platform by reading appropriate SWD registers. The value of this register is set by the engineers that integrate the devices into a design. It is not set within the ARM devices themselves. For more information, see the ARM datasheet or technical reference manual for the processor that you are integrating.

Note

Before you autoconfigure a target that supports both JTAG and SWD, you must first enable **Use SWJ Switching** in the Advanced configuration settings.

Related concepts

4.4 About the scan chain on page 4-58.

4.10 About setting the clock speed on page 4-68.

4.2 About configuring a JTAG scan chain on page 4-51.

5.3 About CoreSight™ autodetection on page 5-97.

4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.

4.4.5 Changing the order of devices on the scan chain on page 4-61.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related information

Connecting the DSTREAM unit.
Connecting the RVI hardware.

4.3.2 About manually adding devices a scan chain

You might want to manually add devices to a scan chain when you do not want to reset and stop the targets on the development platform, or when autoconfiguration fails.

Other circumstances where you might want to manually configure a scan chain include:

- The Read ROM Table autoconfiguration phase for a CoreSight system fails to find any devices.
- Supported devices that are not detectable by the debug hardware unit.
- The device configuration on your development platform has changed since you created this debug hardware configuration, and the platform contains unsupported devices.

  For example, you might have added one or more devices to your development platform. In this case, you might also have to change the order of the devices, if the order on the development platform has changed.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.2 About configuring a JTAG scan chain on page 4-51.
5.3 About CoreSight™ autodetection on page 5-97.
4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.

Related information

Connecting the DSTREAM unit.
Connecting the RVI hardware.

4.3.3 About autoconfiguring a scan chain for CoreSight™ development platforms

You can autoconfigure the scan chain for CoreSight devices.

If your development platform contains CoreSight devices, then autoconfiguration involves:

- Adding the *ARM CoreSight Debug Port* (ARMCS-DP) device.
- Reading the CoreSight ROM Table. This table contains a list of the CoreSight devices included in your development platform.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.2 About configuring a JTAG scan chain on page 4-51.
5.3 About CoreSight™ autodetection on page 5-97.
4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related information

Connecting the DSTREAM unit.
Connecting the RVI hardware.

4.3.4 Autoconfiguring a scan chain

You can autoconfigure a scan chain using the tree diagram.

Procedure

1. Select the **Devices** node in the tree diagram.

   If a scan chain configuration is already set up and has a platform that is assigned to it, autoconfiguration is disabled in Debug Hardware Config. If you want to reconfigure the scan chain automatically, click **Clear Platform** before continuing.

   Note

   Before you autoconfigure a target that supports both JTAG and SWD, you must first enable **Use SWJ Switching** in the Advanced configuration settings.

2. Click **Auto Configure**.

   Each detected device is added to the scan chain configuration list in the control pane, and is also added to the tree diagram. Often, this is all that you have to do to configure the scan chain. You must then configure the devices themselves.

3. If a scan chain configuration is already set up, the following Auto Configure Scan Chain prompt might be displayed:

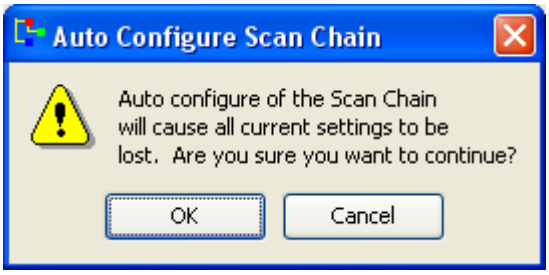

Figure 4-5 Auto Configure Scan Chain dialog box

Caution

If you click **OK**, your current scan chain configuration is lost.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
8.9 Troubleshooting autoconfiguration of a scan chain on page 8-140.
4.8 About device properties on page 4-65.
4.12 About debug hardware device configuration settings on page 4-71.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.

Related references

4.12.3 Debug hardware Advanced configuration settings on page 4-77.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related information

Connecting the DSTREAM unit.
Connecting the RVI hardware.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.4 About the scan chain

There are processes and considerations to be aware of when adding a device to the scan chain.

This section contains the following subsections:

4.4.1 Adding devices to the scan chain

You can add a device to the scan chain using the Add Device dialog box.

Procedure

1. Select the **Devices** node in the tree diagram.
2. Click **Add...**. The Add Device dialog box is displayed. The following figure shows an example:

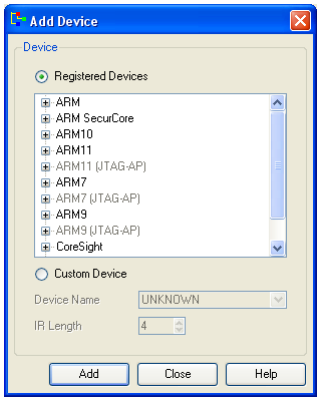

Figure 4-6 The Add Device dialog box

   The device availability depends on your firmware version.

   ——— **Note** ———

   Device groups that are shown in light gray indicates that the related devices are not available. These are seen when you autoconfigure a CoreSight development platform.

   ———————————

3. If the device appears in the list of registered devices:
   a. Select **Registered Devices**. A registered device is a one that the debug hardware unit can identify and for which some level of debug support exists. That is, a template exists for the device.
   b. Expand the relevant device group.
   c. Select the device that you want to add.
   d. Click **Add**. The device is added to the scan chain.

      ——— **Note** ———

      You can also add a device to the scan chain by double-clicking the device.

      ———————————

   e. If you have multiple devices, add each registered device.
   f. After you have added all your devices, click **Close** unless you want to add a custom device.
4. You might want to add a device that is not identified by the debug hardware, because it is a device your are currently developing. To do this:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

a. Select **Custom Device**.
b. 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.
c. Enter the JTAG *Instruction Register* (IR) length in bits in the IR Length field.

Note

If you enter an incorrect value for the IR length, any connections that you attempt to make to the device result in failure.

d. Click **Add**. The device is added to the scan chain.
e. If you have multiple devices, add each custom device.

Note

You cannot debug a custom device. However, adding the custom device in the correct order and with the correct length enables you to debug the supported devices in the same scan chain.

5. When you have finished, click **Close**.

Note

You can remove devices, or change their order in the scan chain, without first having to close the Add Device dialog box.

The following figure shows several devices that have been added to the scan chain in a hierarchical manner:

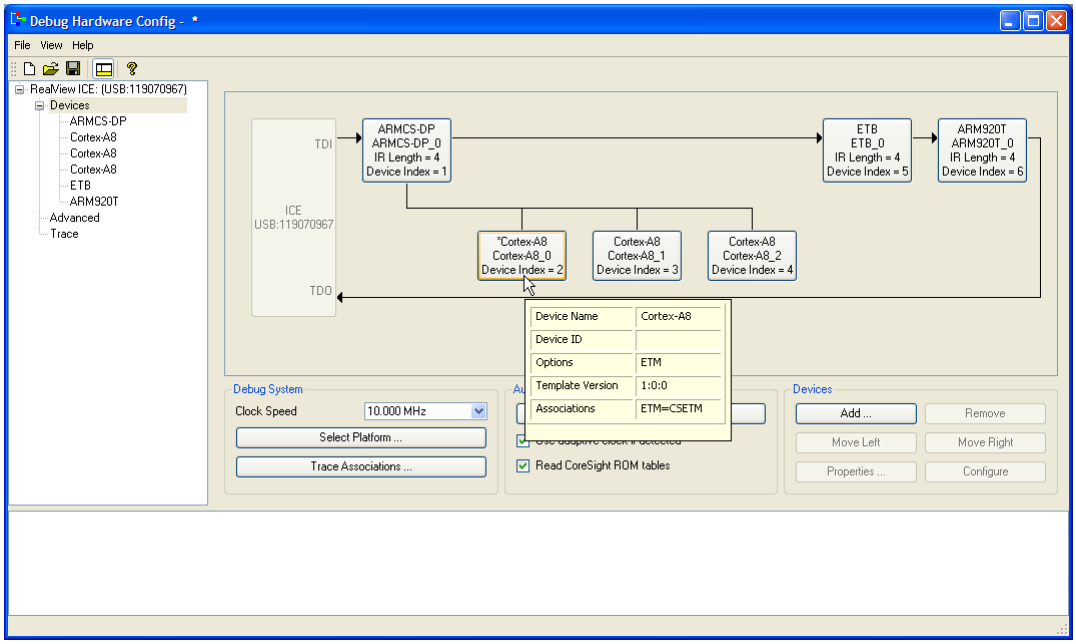

Figure 4-7 Scan chain devices with tooltip feature displayed

4.4.2 Considerations when adding devices to a scan chain

You must be aware of certain rules that apply to different types of configurations when you add devices to a scan chain.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

- In a traditional JTAG configuration, you must add devices in the correct order. 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.

- In a CoreSight configuration, you must first add the ARMCS-DP device. You can then add the remaining CoreSight devices in any order.
- Hierarchies are created automatically when a device is added as a CoreSight component, and enables you to manage component interactions easily.
- When the cursor is placed over a device, a tooltip displays details relating to that device.

4.4.3 Removing a device from the scan chain

You might want to remove a device if you have accidentally added the wrong device, or your target hardware configuration has changed.

To remove an unwanted device from the scan chain:

Procedure

1. Select the **Devices** node in the tree diagram.
2. Select the device in the scan chain configuration.
3. Click **Remove**.

   If a device has a child component, a confirmation prompt is displayed.

Related concepts

4.4 About the scan chain on page 4-58.

4.10 About setting the clock speed on page 4-68.

4.8 About device properties on page 4-65.

Related tasks

4.4.5 Changing the order of devices on the scan chain on page 4-61.

4.4.4 Removing a device from the scan chain when a platform is assigned

If a scan chain configuration is already set up and has a platform that is assigned to it, you must remove the assigned platform before you can remove a device.

To remove a device from the scan chain when a platform is assigned:

Procedure

1. Click **Clear Platform**.
2. Either:
   - Autoconfigure the scan chain again.
   - Manually add only the devices you want.

Related concepts

4.4 About the scan chain on page 4-58.

4.10 About setting the clock speed on page 4-68.

4.8 About device properties on page 4-65.

Related tasks

4.4.4 Removing a device from the scan chain when a platform is assigned on page 4-60.

4.4.5 Changing the order of devices on the scan chain on page 4-61.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.4.5 Changing the order of devices on the scan chain

For a traditional JTAG configuration, the devices must be added to the scan chain in the correct order in relation to debug hardware TDI and TDO. If they have been added in the wrong order, you can move them in the Device node in the tree diagram.

Procedure

1. Select the **Devices** node in the tree diagram.
2. In the scan chain schematic diagram, select the device that you want to move.
3. Click:
   - **Move Left** to move the device to the left
   - **Move Right** to move the device to the right.

Postrequisites

Note

The ordering of CoreSight devices on the same *Debug Access Port* (DAP) is not important.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.5 Select Platform dialog box

The Select Platform dialog box enables you to select a platform configuration that is suitable for your development platform.

The following figure shows an example:

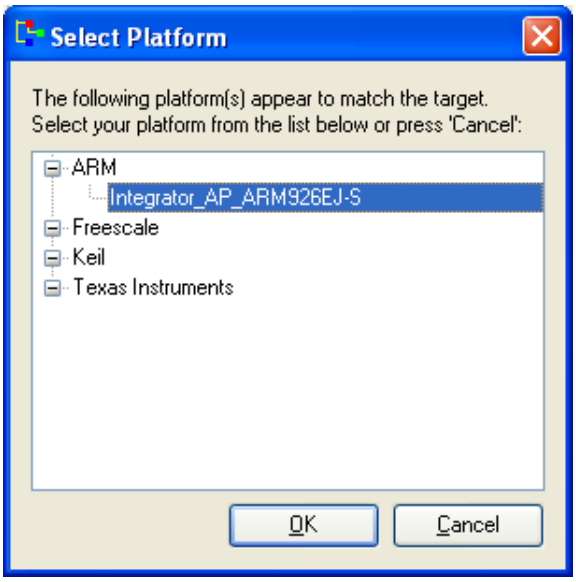

Figure 4-8 Select Platform dialog box

Click the **OK** button to select the chosen platform, in doing so the entire configuration for that platform is loaded.

Click the **Cancel** button to cancel any selected action.

Select **<None>** to put the debug hardware unit into a known state if you have incorrectly loaded a platform. This resets all settings to the default values.

Related concepts

4.15.5 About adding autoconfigure support for new platforms on page 4-86.
4.15 About platform detection and selection on page 4-82.

Related tasks

4.15.1 Autodetecting a platform on page 4-82.
4.15.2 Manually selecting a platform on page 4-84.
4.15.4 Adding new platforms on page 4-86.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.6 Export As Platform dialog box

The Export As Platform dialog box enables you to save the current configuration as a platform configuration.

The following figure shows an example:

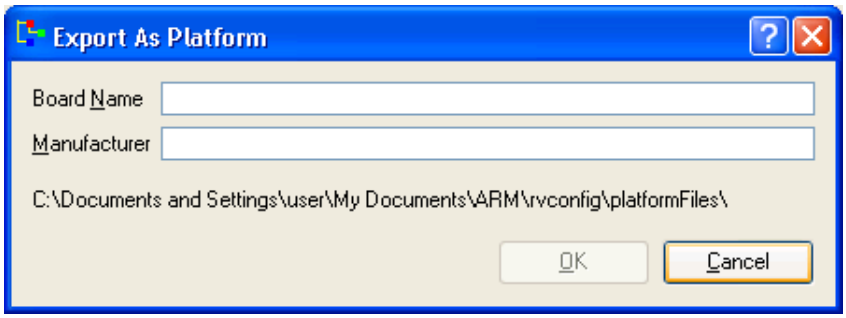

Figure 4-9 Export As Platform dialog box

Related concepts

4.15.5 About adding autoconfigure support for new platforms on page 4-86.

4.15 About platform detection and selection on page 4-82.

Related tasks

4.7 Exporting a configuration to a platform file on page 4-64.

4.15.1 Autodetecting a platform on page 4-82.

4.15.2 Manually selecting a platform on page 4-84.

4.15.4 Adding new platforms on page 4-86.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.7 Exporting a configuration to a platform file

You can export a configuration to a platform file using the Export as Platform dialog. Exported platform files are saved with the format *Manufacturer_BoardName*.rvc.

Procedure

1. Select **File** > **Export platform file...** to display the Export As Platform dialog box.
2. Specify a Board Name for the platform, for example **Integrator_AP_ARM926EJ-S**.
3. Specify a Manufacturer for the platform, for example, **ARM**.
4. Click **OK** to save the platform file and close the Export As Platform dialog box.

   The platform file is stored in:

   `C:\Documents and Settings\`*username*`\My Documents\ARM\rvconfig\platformFiles`

   The name of the platform file has the format:

   Manufacturer_BoardName.rvc
5. Select **File** > **Save** to save the configuration.

Related concepts

4.15.5 About adding autoconfigure support for new platforms on page 4-86.
4.15 About platform detection and selection on page 4-82.
4.6 Export As Platform dialog box on page 4-63.

Related tasks

4.15.1 Autodetecting a platform on page 4-82.
4.15.2 Manually selecting a platform on page 4-84.
4.15.4 Adding new platforms on page 4-86.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.8 About device properties

You can change the properties of the devices on the scan chain.

This section contains the following subsections:

4.8.1 Device Properties dialog box

The Device Properties dialog box enables you to change the properties of any device in the scan chain, if required.

The following figure shows an example:

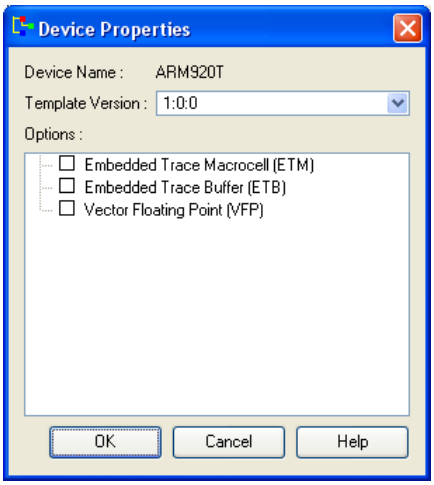

Figure 4-10 The Device Properties dialog box

You can:

- Select one or more options for the device. These options enable a debugger to determine the features that are supported by a device. For example, a Registers view might show the NEON™ registers when the `NEON SIMD Extensions (Neon)` option is selected.
- Set the template version for the device, if multiple versions are provided.

If no properties are available for a device, the following message is displayed in the Options list

4.8.2 Device properties

The device properties listed depends on the device being configured.

Embedded Trace Macrocell (ETM)
Select this if you want to capture trace from the *Embedded Trace Macrocell* (ETM).

Embedded Trace Buffer (ETB)
Select this if you want to capture trace from an *Embedded Trace Buffer* (ETB). You must also select **Embedded Trace Macrocell (ETM)**.

Vector Floating Point (VFP)
Select this to use VFP, if supported.

Vector Floating Point v3 (VFPv3)
Select this to use VFPv3, if supported.

Vector Floating Point v3-D16 (VFPv3-D16)
Select this to use VFPv3-D16, if supported.

NEON SIMD Extensions (Neon)
Select this to use NEON, if supported.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related tasks

4.9 Changing the properties of a device on page 4-67.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.9 Changing the properties of a device

You can change the properties of any device in the scan chain, if necessary. The properties available depend on the device you are configuring. For example, if you want to capture trace from an ETM, you must make sure the `Embedded Trace Macrocell (ETM)` option is selected.

Note

Capturing trace from an ETM is supported only with a DSTREAM unit.

Procedure

1. Select the **Devices** node in the tree diagram.
2. Select the device in the scan chain configuration list.
3. Click **Properties...** to display the Device Properties dialog box. The following figure shows an example:

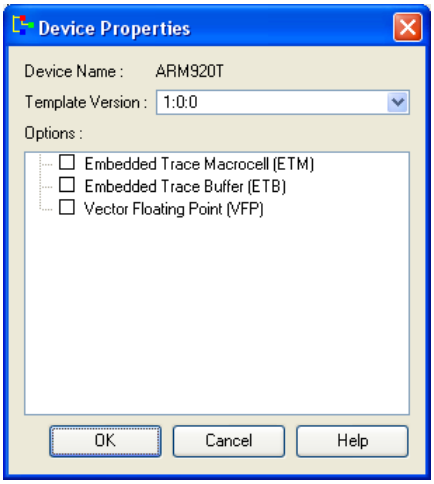

Figure 4-11 The Device Properties dialog box

Note

It is not possible to use an ETB without an ETM, so when you select ETB, ETM is selected automatically.

4. Select the options that you require.

   The options available depend on the device you are configuring.
5. Click **OK**.

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.

Related references

Chapter 3 Managing the Firmware on your Debug Hardware Unit on page 3-35.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.10 About setting the clock speed

It is important to select the best 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 manually reducing the clock speed. If you are not sure which clock speed to use, try setting the default speed, 10MHz.

Note

For RVI, for reliable operation at high clock speeds you must use the *Low Voltage Differential Signaling* (LVDS) cable.

This section contains the following subsections:

- *4.10.1 Setting a predefined clock speed* on page 4-68.
- *4.10.2 Setting a custom clock speed* on page 4-69.

4.10.1 Setting a predefined clock speed

You can select a predefined clock speed in the tree diagram.

Procedure

1. Select the **Devices** node in the tree diagram.
2. Select the clocking that you want to use from the Clock Speed drop-down list. The following figure shows an example:

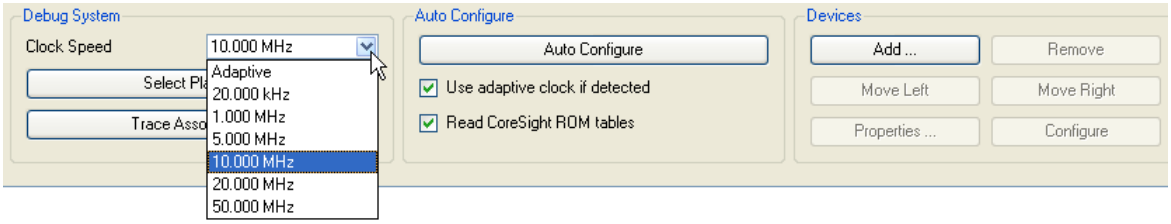

Figure 4-12 The scan chain speed controls

Note

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

Related concepts

4.4 About the scan chain on page 4-58.
4.10 About setting the clock speed on page 4-68.
4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.
4.4.5 Changing the order of devices on the scan chain on page 4-61.

Related information

ARM DSTREAM System Design Guidelines.
RVI Debug Unit System Design Guidelines.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.10.2 Setting a custom clock speed

If the clock speed you want to use is not available as a preset value, enter the required clock speed in the Clock Speed field with the **Hz**, **kHz**, or **MHz** suffix as required. For example, **40.0 kHz**.

Related concepts

4.4 About the scan chain on page 4-58.

4.10 About setting the clock speed on page 4-68.

4.8 About device properties on page 4-65.

Related tasks

4.4.3 Removing a device from the scan chain on page 4-60.

4.4.5 Changing the order of devices on the scan chain on page 4-61.

Related information

ARM DSTREAM System Design Guidelines.

RVI Debug Unit System Design Guidelines.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.11 About Adaptive clocking

Adaptive clocking enables your debug hardware unit to dynamically adjust the JTAG clock (**TCK**) following changes to the processor clock (**CLK**). This is useful when debugging system with low power modes and changing clocks in general.

Adaptive clocking is intended only for targets that are based on a synthesizable ARM processor implementing **RTCK**. When a fixed JTAG clock is used, the JTAG clock must be running at most 1/6 of the processor clock. Using a fixed JTAG clock does not support changes in the processor clock speed, which can corrupt the JTAG connection.

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.

Related information

ARM DSTREAM System Design Guidelines.

RVI Debug Unit System Design Guidelines.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.12 About debug hardware device configuration settings

You can configure settings that are specific to a target device on your development platform. The settings available depend on whether the device is a processor or a non-processor CoreSight device.

This section contains the following subsections:

4.12.1 Processor device settings

Depending on the processor that you have selected, a list of controls are available.

Allow execution with T-Bit Clear
: The Cortex-M series processors can only execute Thumb® code. However, their xPSR contains a bit to configure the instruction set state (ARM or Thumb), that is initialized from the reset vector. This enables you to test the exception handler that is associated with this exception type.

Allow PRCR DBGNOPWRDWN to be set
: Allow the DBGNOPWRDWN bit in the processor debug registers to be set. When set, the SoC power controller does not power down the processor.

Asynchronous abort handling on debug entry
: Ensure that all possible outstanding asynchronous Data Aborts are recognized before entry to the debug state.

Bypass memory protection when in debug
: This option enables the bypass of any memory protection provided by hardware (such as a memory management or protection unit) whenever the target hardware enters debug state. This means that you can access protected memory to set software breakpoints in it, or to alter its contents.

Bypass the device ID check on connect
: This option bypasses the check of the CoreSight component and peripheral ID registers, for a processor, on connection to the target.

Check PSCR dbgen bit can be set high
: Check whether the **DBGEN** signal can be set HIGH.

Clear breakpoint hardware on connect
: This control is available if you are using the ARM11™ family of processors. The debug logic of an ARM11 processor is not reset when a *Test Access Port* (TAP) reset is applied. Set this option to `True` to instruct the debug hardware unit to perform reset the debug logic each time you connect.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Code Sequence settings

Most systems store variables and the stack and heap in RAM. However, in some systems only Flash or ROM is mapped in memory at power-up and RAM must be enabled by software in the boot code.

To perform certain operations with some targets, the debug hardware unit must download a piece of code into memory and make the processor execute it. This code must be located in a writable area of memory (RAM) and must be accessed only by the JTAG tool.

You can set the address and size of this code using the **Code Sequence Address** and the **Code Sequence Size (bytes)** settings.

When you connect with a debugger to one of these targets, make sure that RAM is mapped in memory and that the cache clean code or code sequence address is mapped correctly. You can run a script from the command line to configure the target memory map straight after connecting to the target.

This area of memory must be:

- Unused by the target
- Readable
- Writable
- Non-cacheable (for cached targets)
- At least 128 bytes in size.

The debug software downloads code sequences to this area to perform various tasks, such as cleaning the cache and accessing certain system registers on targets such as ARM920T and ARM1136JF-S. It does not preserve the contents of this area.

——— **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 debugger Registers view. If you do not set these values correctly, your 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.

The **Use code sequence to clean cache** option enables you to configure how the caches are cleaned if you are using ARM 920T or ARM922T processors. Set this option when using the debugger to access IO memory, for example peripheral control registers for *Universal Asynchronous Receiver/Transmitters* (UARTs), when caches are enabled.

CoreSight AP index

The AP index of the device.

Available only for processors that are part of a CoreSight system.

CoreSight base address

The base address of the CoreSight registers for the processor on the *Advanced High-performance Bus* (AHB) or *Advanced Peripheral Bus* (APB).

Available only for processors that are part of a CoreSight system.

Debug acceleration level

This controls the level of acceleration used in debug operations.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Select one of the following options:

0 - Full This is the default. It enables full use of the performance features of RVI and the target processor.

1 - Partial This results in a lower performance than for the **Full** option.

2 - None Select the **None** option to use only basic operations. This results in the lowest performance available.

——— **Note** ———

In most instances, select the **Full** option unless advised otherwise by an ARM support engineer.

———————————

Default Gateway

Default gateway for the target when using virtual Ethernet. Used with the **IP Address** and **Network Mask** settings to enable access to your target from the network.

Enable continuous target state checking and DCC transfer

This option enables a background task to monitor the processor state and perform DCC transfers while the processor is executing.

Fast Memory Download

This control is available for those targets where the *Debug Access Port* (DAP) and the processor are running sufficiently fast enough to handle the data being sent to them by the debug unit. The debug unit does not have to check that each individual transaction with the DAP is successful.

Because the processor is behind the DAP, all processor accesses have to go through the DAP. As a guide, do not set this for FPGA-based targets, but only for real silicon.

If you set this option, then error checking is disabled and you are not informed of any errors that occur. If problems are encountered when downloading images, then uncheck this option to resolve them.

Set this to `False` if you experience problems using fast memory downloads.

This option only applies to JTAG-AP targets.

Ignore bad JTAG IDCODE

By default, debug hardware 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 debug hardware reads an invalid (bad) JTAG IDCODE, it assumes that the JTAG connection is not functioning properly, and fails the attempt to connect to the processor.

You must set the **Ignore bad JTAG IDCODE** option according to whether you want to instruct debug hardware to enable connection to the processor even if it detects that the JTAG IDCODE is invalid.

Ignore debug privilege errors when starting core

When the SPIDEN line is changed from HIGH to LOW, the following errors might be seen:

- `Insufficient debug privilege to restore core state for restart.`
- `Insufficient debug privilege to write software breakpoint to memory.`
- `Set Ignore debug privilege errors when starting core to suppress these errors.`

If set to `True`, debug hardware starts the processor running even though the breakpoints/ processor state is incorrect.

If set to `False` (the default), debug hardware refuses to start the processor and reports the errors.

IP Address

IP address of the target when using virtual Ethernet. Used with the **Default Gateway** and **Network Mask** settings to enable access to your target from the network.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

JTAG-AP Port index for core

For CoreSight systems with processors connected to JTAG-AP, the port index on the JTAG-AP to which the processor is connected.

JTAG timeouts enabled

JTAG timeouts are enabled by default. You must disable these when debug hardware is connected to a processor using a low clock speed and adaptive clocking, because debug hardware cannot detect the clock speed when adaptive clocking is used, so cannot scale its internal timeouts. If a JTAG timeout occurs, the JTAG is left in an unknown state, and debug hardware cannot operate correctly without reconnecting to the processor.

Network Mask

Net Mask of the target when using virtual Ethernet. Used with the **IP Address** and **Default Gateway** settings to enable access to your target from the network.

No error if step-instr can't stop

Controls generation of error messages if a debugger step instruction operation fails because a timeout attempts to stop the SecurCore® processor after a step is complete. This can occur on the SecurCore processor if an instruction execution results in the processor clock being disabled using CLKEN. The processor appears to be in a running state.

If set to `True` (the default), then no error message appears if an instruction step results in the processor running.

If set to `False`, then an error dialog box is displayed in your debugger.

Post Reset State

Set to the required state for the target hardware:

Running The target hardware is running.

Stopped Controls the state of a processor after a reset. It is only available for devices that are capable of running (such as ARM processors). Each device on the scan chain does not have to be set to the same value, so it is valid to have one processor running and another 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 debug hardware, set the Post Reset State to **Running**.
- In your debugger, connect using the **No Reset/No Stop** connection mode.

———————————

Soft TAP reset on JTAG sync loss

In some situations, such as a processor entering low power mode, the synchronization between the debug unit and the TAP Controller in the JTAG system can be lost. This can result in invalid values for the debug status register being read. To regain the synchronization, a soft TAP reset must be performed to get the TAP Controller into a known state.

If Soft TAP reset on JTAG sync loss is checked, a soft TAP reset is performed to get the TAP Controller into a known state if the debug unit reads invalid values for the debug status register.

Software breakpoint mode

If supported by your processors, this control enables you to configure how the debug hardware unit handles software breakpoints. Select the required breakpoint mode:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

AUTO This is the default mode for all templates:
- If the processor being debugged supports `BKPT` instructions, debug hardware automatically uses the `BKPT` instruction for software breakpoints.
- Only one watchpoint resource is used for multiple software breakpoints. Therefore, if the processor being debugged does not support `BKPT` instructions, debug hardware uses the watchpoint unit resource when you set a software breakpoint. The debug hardware unit 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, debug hardware gives an error message telling you that there are no free resources to set the breakpoint.

WATCHPOINT This mode instructs debug hardware to use one watchpoint unit to provide software breakpoint instructions, whether the processor being debugged supports `BKPT` instructions. Select this option if the processor supports `BKPT` instructions but you want to use a watchpoint unit.

BKPT This mode instructs debug hardware to use the `BKPT` instruction to provide software breakpoint instructions, whether the processor supports this instruction. Select this option if you want to make sure that no watchpoints are used.

Target byte order

The endianness of some processors cannot be determined from the processor registers. This option informs the debug unit what the endianness of the processor is.

0

Little endian. This is the default.

1

Big endian.

Use CTI for synchronized execution

This is set internally by DS-5 for synchronized execution. You are not expected to modify this.

Use LDM or STM for memory access

This options controls whether to use *Load Multiple Instructions* (LDM) or *Store Multiple Instructions* (STM) to access target memory. You might need to set this option if you have a peripheral that is not fully compatible with the AMBA® 2.0 standard, as in such cases LDM and STM might not be compatible.

Write-Through L2 Cache when in debug

This option is available if you are using an ARM11 processor.
Use this option with platforms that have a level 2 cache that interferes with debug operations. By default it is set to `False`, but the platform configuration files supplied for affected platforms set it to `True`.

Note

If you set this option for other platforms, unexpected behavior might result, and cause errors.

Note

ARM7xx-JTAG-AP, ARM9xx-JTAG-AP, or ARM11xx-JTAG-AP devices represent the JTAG-AP in a CoreSight system. To debug CoreSight systems that have processors that are connected to the DAP, the debug unit must know the pre-scan and post-scan bits for JTAG operations.

Multiple devices on the JTAG scan chain are connected in series, with data flowing serially from TDI to TDO. This means that debugging a given target in the chain requires that certain pre-scan and post-scan

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

bits are used. These bits ensure that the other devices are not affected by the data directed at the target device, and that the data is positioned correctly in the serial scan for the target device.

Related concepts

5.6 Configuration items available for ARM7™, ARM9™, and ARM11™ processors in a CoreSight system on page 5-100.

7.11 Strategies used by debug hardware to debug cached processors on page 7-123.

Related tasks

4.17 Configuring a target processor for virtual Ethernet on page 4-91.

Related references

5.5 Configuration items for processors in a CoreSight™ system on page 5-99.

4.12.2 Non-processor CoreSight™ device settings

Several device configuration settings are available for non-processor CoreSight devices.

Channel to pulse on synchronized execution start
This is set internally by DS-5 for synchronized execution. You are not expected to modify this.

CoreSight AP index
The AP index of the device.

Available for all devices, except the ARMCS-DP and ARMSW-DP devices.

CoreSight base address
The base address of the CoreSight registers for the device on the AHB or APB.

Available for all devices, except the ARMCS-DP and ARMSW-DP devices.

Enable synchronized execution start with CTI
This is set internally by DS-5 for synchronized execution. You are not expected to modify this.

Force ETM power up on connect
If the *CoreSight Embedded Trace Macrocell* (CSETM) is powered down when your debugger attempts to connection to it, then power up the device.

Memory Access AP index
The index number of the AHB-AP memory bus on the DAP. The AHB-AP bus is used for performing memory operations within the CoreSight system.

Available for the ARMCS-DP and ARMSW-DP devices only.

The ARMCS-DP device represents the *Debug Access Port* (DAP) in a CoreSight system. This device is automatically detected when you autoconfigure a CoreSight system. However, any devices that are connected to the DAP are not detected. Therefore, you must read the CoreSight ROM table of the ARMCS-DP to determine the devices that are connected to the DAP. If no CoreSight ROM table is present, then you must manually add devices to the scan chain.

Related concepts

5.6 Configuration items available for ARM7™, ARM9™, and ARM11™ processors in a CoreSight system on page 5-100.

7.11 Strategies used by debug hardware to debug cached processors on page 7-123.

Related tasks

4.17 Configuring a target processor for virtual Ethernet on page 4-91.

Related references

5.5 Configuration items for processors in a CoreSight™ system on page 5-99.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.12.3 Debug hardware Advanced configuration settings

Depending on your development platform configuration, several controls are available in the Advanced settings.

Allow ICE to perform TAP reset
Set this item to `True` to allow your debug hardware unit to hold the **nSRST** line active long enough to perform any post-reset setup that might be required after a target-initiated reset. This can extend the time that the target is held in reset. If this item is set to `False`, your debug hardware unit does not assert the reset line, but post-reset setup might not be complete before the target starts to run.

Allow ICE to latch System Reset
Set this item to `True` to allow your debug hardware unit to perform **nTRST** reset while holding **nSRST**. This ensures that the *Test Access Port* (TAP) state machine and associated debug logic is properly reset.

LVDS Debug Interface mode
This can be set either to JTAG or SWD. If set to SWD, this causes RVI to connect to the target using the SWD protocol instead of JTAG.

nSRST High mode
Selects the drive strength that is used when the reset signal is in the high, or inactive, state. Output can be driven as a strong or weak high, or not driven (tri-state).

nTRST High mode
Selects the drive strength that is used when the reset signal is in the high, or inactive, state. Output can be driven as a strong or weak high, or not driven (tri-state).

nSRST Low mode
Selects the drive strength that is used when the reset signal is in the low, or active, state. Output can be driven as a strong or weak low, or not driven (tri-state).

nTRST Low mode
Selects the drive strength that is used when the reset signal is in the low, or active, state. Output can be driven as a strong or weak low, or not driven (tri-state).

nSRST Hold Time (ms)
Specifies how long the debug hardware unit holds the hardware **nSRST** system reset signal LOW.

nTRST Hold Time (ms)
Specifies how long the debug hardware unit holds the **nTRST** TAP reset signal LOW.

nSRST Post Reset Delay (ms)
Specifies how long after the hardware **nSRST** system reset before the debug hardware unit enters the Post Reset State.

nTRST Post Reset Delay (ms)
Specifies how long after the **nTRST** TAP reset before the debug hardware unit enters the Post Reset State.

Perform TAP reset on first connect
Resets the target hardware whenever you connect.

Perform SYS reset on first connect
Resets the target hardware by asserting the **nSRST** signal when connecting to the first device in a debug session.

Reset Type
One of the following:

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.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

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*).

Ctrl_Reg
The Control register. This reset, in instances where processors have a reset register, enables you to reset the processor without using the external reset lines. If you set the reset type to `Ctrl_Reg`, then this control register is used.

SWO Mode
Set to Manchester or UART, depending on the target mode.

If the **SWO Mode** is set to UART, the debug hardware unit is able to detect the **SWO UART Baud rate**.

This setting has no effect in Manchester mode.

SWO UART Baud rate
For the frequency of the incoming data. If you set this to 0, the system attempts to autodetect the baud rate.

Note

UART mode in the SWO context also means *Non Return to Zero* (NRZ) mode.

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.

Target nSRST + nTRST linked
If the target hardware has its **nSRST** and **nTRST** JTAG signals linked.

Use SWJ Switching
If this is set, it causes the SWJ switching sequence to be sent before connecting to the target device. On devices that support SWJ switching, this causes the DAP to switch its interface to the selected protocol.

Note

If a target supports both JTAG and SWD, you must enable this setting before you autoconfigure that target.

Use deprecated SWJ Sequence
If this is set, it causes your debug hardware unit to use an alternative SWJ switching sequence, which is used on some older SWD-compatible targets. This option is normally clear, unless the processor requires the deprecated sequence.

Note

For RVI, the **LVDS Debug Interface mode**, **Use SWJ Switching**, and **Use deprecated SWJ Sequence** controls are not present in the control pane if you are not using a *Low Voltage Differential Signaling* (LVDS) probe.

User Output*n*
Sets the state of the **USER IO** pins on the rear of the RVI unit, or on the front of the DSTREAM unit.

Related concepts

7.14 About debugging with a reset register on page 7-127.

Related tasks

4.15.6 Configuring the debug hardware Advanced settings on page 4-87.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related information

DSTREAM reset signals.

RVI reset signals.

4.12.4 Debug hardware Trace configuration settings

The Trace configuration settings enable you to configure delays on the trace lines.

Delay Trace Clock, Delay Trace Signal *N*

Delay line *N* by a specified amount of time, expressed in intervals of 75 picosecond. Default delays are configured into the unit, and you are able to delay each signal by a specified amount relative to these defaults, to allow for variations in target hardware.

Invert Trace Clock

Invert the clock so that data is sampled on the falling edge, rather than on the rising edge, of the clock.

Related concepts

6.4 About configuring your debugger for trace capture on page 6-109.

6.1 About using trace hardware on page 6-105.

Related tasks

6.3 Configuring trace lines (DSTREAM unit only) on page 6-107.

4.12.5 Debug hardware Advanced configuration reset options

The debug hardware unit can be configured for system reset or for TAP reset.

Allow ICE to latch System Reset (AllowICELatchSysRst)

When enabled, this option enables the debug hardware unit to extend the time the target controller holds the target in reset. This enables the debug hardware unit to apply breakpoint settings before the processor starts execution. This is useful for debugging a target from reset, and allows the unit to stop the processor on the first instruction fetch after reset has been released by the unit.

When this option is disabled, the breakpoint settings might only take effect after the processor has already started execution, preventing debugging of the application reset handler.

The default setting is `True`.

Allow ICE to perform TAP Reset (AllowICETAPReset)

A *Test Access Port* (TAP) reset on early processors, such as the ARM7TDMI, also reset the debug registers associated with the JTAG device.

Later processor, such as the ARM1176, are not affected by a TAP reset. The main purpose of a TAP reset is to reset the JTAG state machine in the TAP Controller receiving commands from the debug hardware unit.

The default setting is `True`.

Related concepts

4.13 Configuring SecurCore® processor behavior if the processor clock stops when stepping instructions on page 4-80.

4.14 Errors when configuring TrustZone® enabled processor behavior when debug privileges are reduced on page 4-81.

Related references

4.12.3 Debug hardware Advanced configuration settings on page 4-77.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.13 Configuring SecurCore® processor behavior if the processor clock stops when stepping instructions

If a step instruction operation fails, you must decide whether to set the configuration setting **No error if step-instr can't stop (NO_ERROR_ON_STEPRUN)**. This configuration item controls the generation of error messages if a step instruction (`stepi`) operation fails because of a timeout attempting to stop the processor after a step is complete.

This can occur on the SecurCore processor if an instruction execution results in the processor clock being disabled through **CLKEN**. The processor appears to be in a running state. If you use the default setting of `True`, no error appears if an instruction step results in the processor running. If you set the item to `False`, an error dialog appears in your debugger.

Related concepts

4.14 Errors when configuring TrustZone® enabled processor behavior when debug privileges are reduced on page 4-81.

Related references

4.12.5 Debug hardware Advanced configuration reset options on page 4-79.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.14 Errors when configuring TrustZone® enabled processor behavior when debug privileges are reduced

The target does not allow invasive debug, that is when the processor enters debug state, while the execution environment is in the Secure World. Any attempt to do so might result in errors.

These errors are:

- Insufficient debug privilege to restore processor state for restart.
- Insufficient debug privilege to write software breakpoint to memory.

To suppress these errors, set the **Ignore debug privilege errors when starting core (IGNORE_START_DEBUG_PRIV_FAIL)** configuration item.

This option is useful if you are trying to debug an application in Normal World, while the current connection state of the target is in Secure World.

In this setup, the debug information and breakpoints are applied to the Normal World. However, the initial connection still happens while the target is in Secure World. The debug hardware unit has no control in this state, so debug control must be delayed until the target enters Normal World. Until that time, any errors arising from debug operations must be silently ignored.

Related concepts

4.13 Configuring SecurCore® processor behavior if the processor clock stops when stepping instructions on page 4-80.

Related references

4.12.5 Debug hardware Advanced configuration reset options on page 4-79.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.15 About platform detection and selection

Your debug hardware unit provides support for several development boards.

Two methods are available for platform detection and selection:

- Autodetection.
- Manual selection.

You can also create your own platform files and add them to the list of available platforms.

Note

The platform file contains information on the scan chain of a platform at the time the file was saved. However, if you create a scan chain configuration with the devices in a different order, then the saved platform file is not auto-detected. However, you can still manually select the platform configuration from the list of available platforms.

For more information on the various platforms that are supported, see also the Release Notes of the host software for your debug hardware unit.

This section contains the following subsections:

4.15.1 Autodetecting a platform

The debug hardware autodetection feature checks to see if a platform configuration file is available that matches the configuration of your development platform. If a platform configuration is available, the Select Platform dialog box is displayed.

An example is shown in the following figure:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

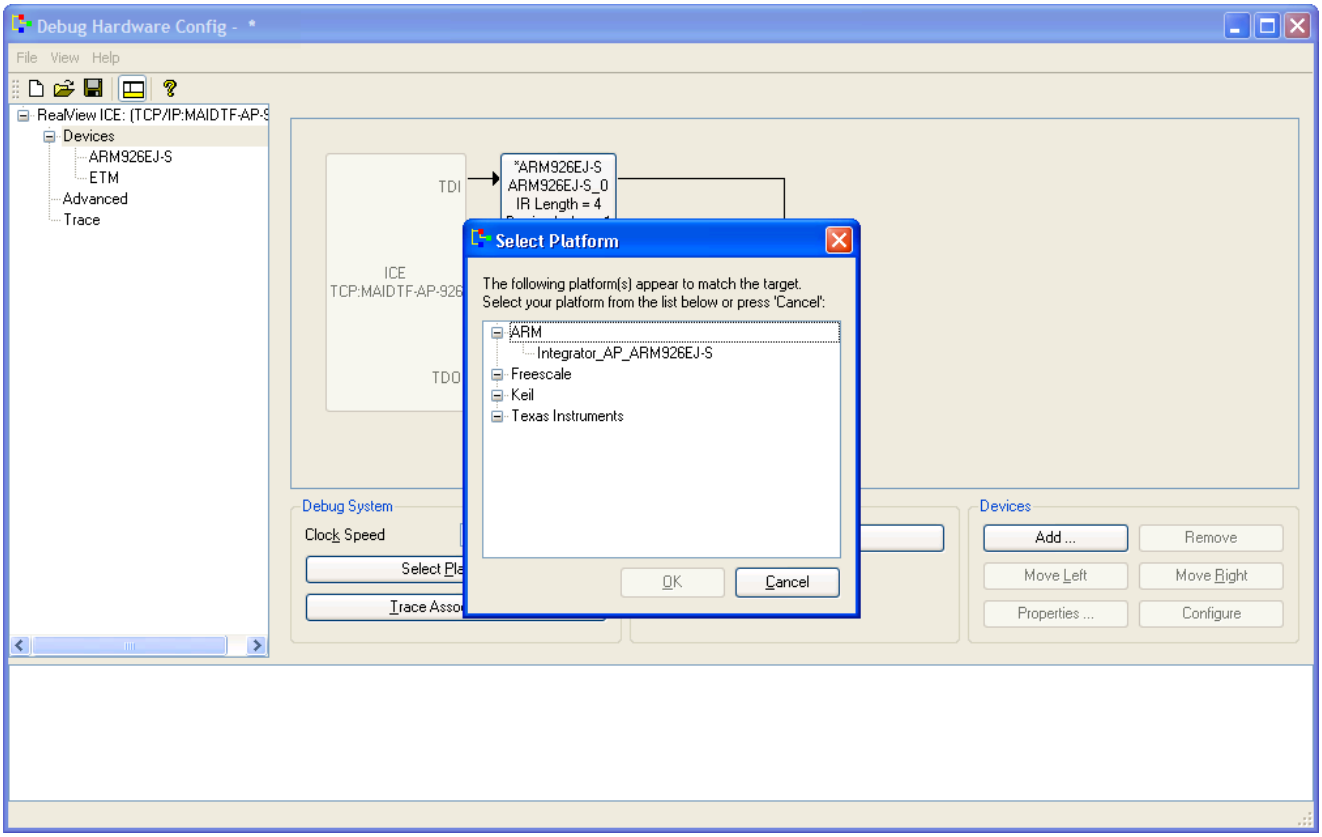

Figure 4-13 Automatic platform configuration

Select your platform from the list and click **OK**. This causes the entire platform configuration (that is, for the scan chain, Advanced settings and trace delays) to be loaded. The following figure shows an example:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

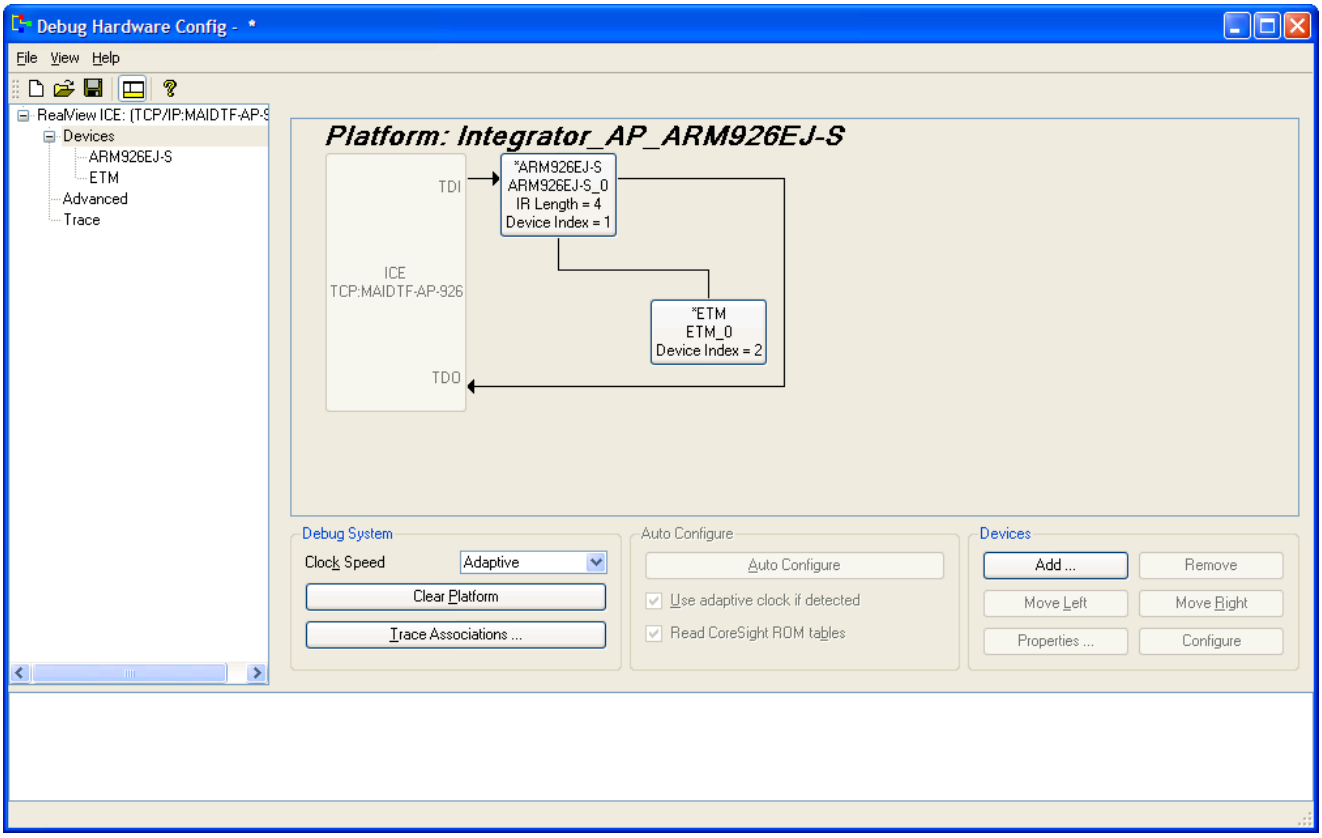

Figure 4-14 Platform configuration and identification

After the platform configuration is complete, the control pane shows the revised scan chain device/ platform configuration in use and the name of the loaded platform. The tree diagram also reflects the new configuration.

Note

You cannot add, move, or remove any devices when a platform configuration is in use.

During the configuration process, the label on the **Select Platform...** button changes to read as **Clear Platform**.

Related concepts

4.15 About platform detection and selection on page 4-82.

Related tasks

4.15.2 Manually selecting a platform on page 4-84.

4.15.3 Clearing a platform assignment from a debug hardware configuration on page 4-85.

4.15.4 Adding new platforms on page 4-86.

4.15.2 Manually selecting a platform

For platforms that cannot be detected automatically, you can perform a manual selection from a list of supported platforms. To manually select a platform, click the **Select Platform...** button from the Debug System pane of the Debug Hardware Config utility, and the Select Platform dialog box displays.

The following figure shows an example:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

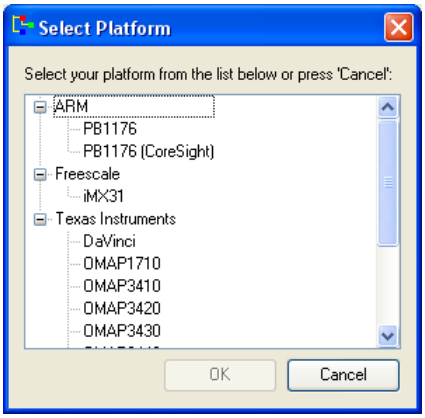

Figure 4-15 List of supported platforms

In the Select Platform dialog box, select your platform and click **OK**. This causes the entire platform configuration to be loaded, in a similar manner as when you are autodetecting a platform.

During the configuration process, the label on the **Select Platform...** button changes to read as **Clear Platform**.

You can also create your own platform files and add them to the list of available platforms.

Related concepts

4.15.5 About adding autoconfigure support for new platforms on page 4-86.
4.15 About platform detection and selection on page 4-82.

Related tasks

4.15.1 Autodetecting a platform on page 4-82.
4.15.3 Clearing a platform assignment from a debug hardware configuration on page 4-85.
4.15.4 Adding new platforms on page 4-86.

4.15.3 Clearing a platform assignment from a debug hardware configuration

You might want to clear a platform assignment from a debug hardware configuration because you want to reconfigure the scan chain for your development platform.

——— **Note** ———

This also clears the scan chain configuration. As an alternative, you might want to create a debug hardware configuration.

———————————

Procedure

1. Open the required debug hardware configuration file in the Debug Hardware Config utility.
2. Click **Clear Platform**.

   The scan chain configuration is deleted.
3. Either autoconfigure the scan chain or manually add devices to the scan chain as required.

Related concepts

4.4 About the scan chain on page 4-58.

Related tasks

4.15.4 Adding new platforms on page 4-86.
4.3.4 Autoconfiguring a scan chain on page 4-56.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.15.4 Adding new platforms

Debug Hardware Config is preconfigured to support several platforms, but you can add support for more platforms by creating your own platform files. You can also provide a name and the manufacturer details for the new platform.

Procedure

1. Configure a scan chain for the new platform.
2. Change the device settings as required.
3. Change the advanced settings as required.
4. Change the trace settings as required.
5. Specify a new platform name and its manufacturer in the Export As Platform dialog box. To do this, select **File** > **Export platform file...**.
6. In the Export As Platform dialog box, enter the new name and manufacturer details of the platform in the **Board Name** and **Manufacturer** fields, respectively. The following figure shows an example:

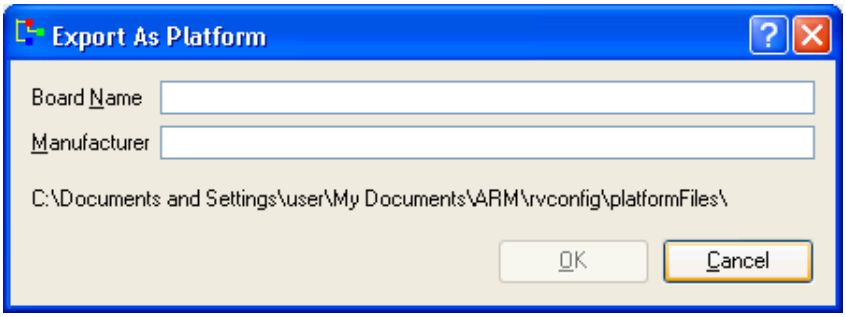

Figure 4-16 Export As platform dialog box

   The name of the new platform file is made up from the board name and manufacturer details of the platform. The board name and manufacturer are displayed in the Select Platform dialog box for the new platform.

   On Windows, the `platformFiles` directory, shown in this figure, is located in `My Documents\ARM\rvconfig\platformFiles`.

   On Linux, it is located in `~/.rvconfig/platformFiles`.
7. Click **OK**.

Related concepts

4.2 About configuring a JTAG scan chain on page 4-51.
4.3 About configuring a device list on page 4-54.
4.15.5 About adding autoconfigure support for new platforms on page 4-86.
4.15 About platform detection and selection on page 4-82.

Related tasks

4.15.1 Autodetecting a platform on page 4-82.
4.15.2 Manually selecting a platform on page 4-84.
4.15.6 Configuring the debug hardware Advanced settings on page 4-87.
6.3 Configuring trace lines (DSTREAM unit only) on page 6-107.

4.15.5 About adding autoconfigure support for new platforms

When adding platforms to the Select Platform dialog box, a platform detection file `.det` is created automatically after you have setup the board name and manufacturer. You can, however, add more `.det` files to the `platformFiles` directory, which allow different hardware versions to be recognized when you click the **Auto Configure** button.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

A `.det` file can contain information for one or more platforms. The platform description consists of a JTAG ID code and mask for each device on the JTAG scan chain of the target platform, and the name of the associated debug hardware configuration file `.rvc`.

For example:

```
0x0B73B02F,0xFFFFFFFF,  0x07926001,0xFFFFFFFF,  0,0,  0x2B900F0F,0xFFFFFFFF =
mycompany_eg1.rvc
```

In the above example, the platform is expected to contain four devices on its scan chain. The first device must have a JTAG ID code of `0x0B73B02F`, the second `0x07926001`, the third device can have any ID code, and the code of the fourth device must be `0x2B900F0F`.

Several `.det` files can be supplied, and each file can contain more than one line. For example:

```
0x22193024, 0xEFFFFFFF, 0,0, 0,0 = mycompany_eg2.rvc
0x08210024, 0xFFFFFFFF, 0,0, 0,0 = mycompany_eg2.rvc
0x05310024, 0xFFFFFFFF, 0,0, 0,0 = mycompany_eg3.rvc
```

In this example, a scan chain containing three devices, where the first device has an ID code of `0x22193024`, `0x32193024` or `0x08210024` is associated with `mycompany_eg2.rvc`. The mask value of `0xEFFFFFFF` means that both `0x22193024` and `0x32193024` are identified. If there are three devices, and the first has an ID code of `0x05310024`, then it is associated with `mycompany_eg3.rvc`.

When you click the **Auto Configure** button, the detected scan chain is compared against all the platforms described by `.det` files. If the connected target matches any of these platforms, a Select Platform dialog box is displayed, and asks the user to confirm that the platform has been correctly detected.

It is possible to create platform information that associates a given scan chain with several `.rvc` files. If this happens, the Select Platform dialog box lists all the platforms that match, and you will be asked to select the platform that matches your hardware.

Related tasks

4.15.4 Adding new platforms on page 4-86.

4.15.6 Configuring the debug hardware Advanced settings

The Advanced settings enable you to change the global configuration settings for the debug hardware unit. Such settings include debug connection mode settings and reset settings.

Note

These settings are sent to a debug hardware unit whenever you connect to the unit from a debugger. They are used as the default reset behavior for all target hardware that you debug with that debug hardware unit.

Procedure

1. Open the Debug Hardware Config utility.
2. Either:
   - Connect to and configure a debug hardware connection to create a configuration file.
   - Open an existing debug configuration file.
3. Select the Advanced settings group in the tree control to display the settings. The following figure shows an example:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

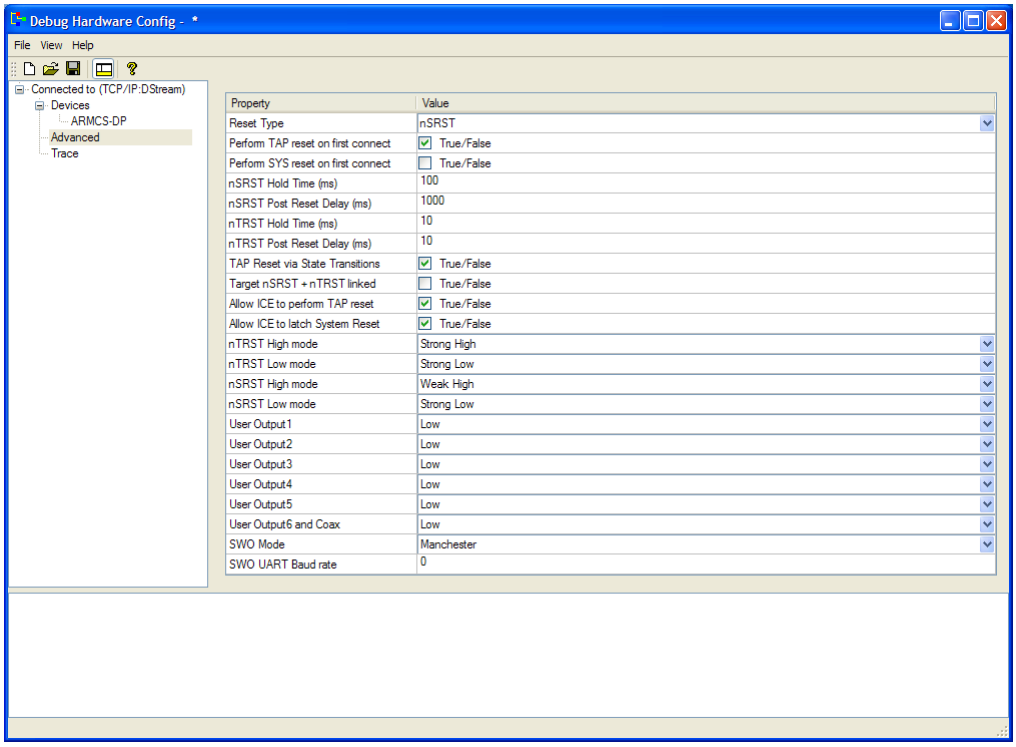

Figure 4-17 Displaying the advanced controls

4. Change the settings as required.
5. Select **File** > **Save** to save your changes.
6. Select **File** > **Exit** to close the Debug Hardware Config utility.

Related concepts

7.14 About debugging with a reset register on page 7-127.

Related references

Chapter 3 Managing the Firmware on your Debug Hardware Unit on page 3-35.
4.12.3 Debug hardware Advanced configuration settings on page 4-77.

Related information

ARM DSTREAM System and Interface Design Reference - Reset signals.
ARM DSTREAM System and Interface Design Reference - Serial Wire Debug.
ARM RVI System and Interface Design Reference - Reset signals.
ARM RVI System and Interface Design Reference - Serial Wire Debug.

4.15.7 Saving your changes

To save any changes that you have made to a configuration in the Debug Hardware Config utility, select **File** > **Save**.

Changes are stored in a debug hardware configuration file, `*.rvc`. See your debugger documentation for the location of this file. There can be several `*.rvc` files in this location, and the file names of the rvc files are related the connection name.

You can change the location of the `*.rvc` file, or save multiple copies of the file for different target configurations.

Related tasks

4.1.1 Creating a debug hardware configuration file on page 4-48.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.1.2 Opening an existing debug hardware configuration file in Debug Hardware Config on page 4-49.
4.16 Disconnecting from a debug hardware unit on page 4-90.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.16 Disconnecting from a debug hardware unit

You can disconnect from a debug hardware unit if you want to connect to another debug hardware unit using the Debug Hardware Config utility.

Procedure

1. Select the debug hardware node in the tree diagram to display the Debug Hardware Config utility. The following figure shows an example:

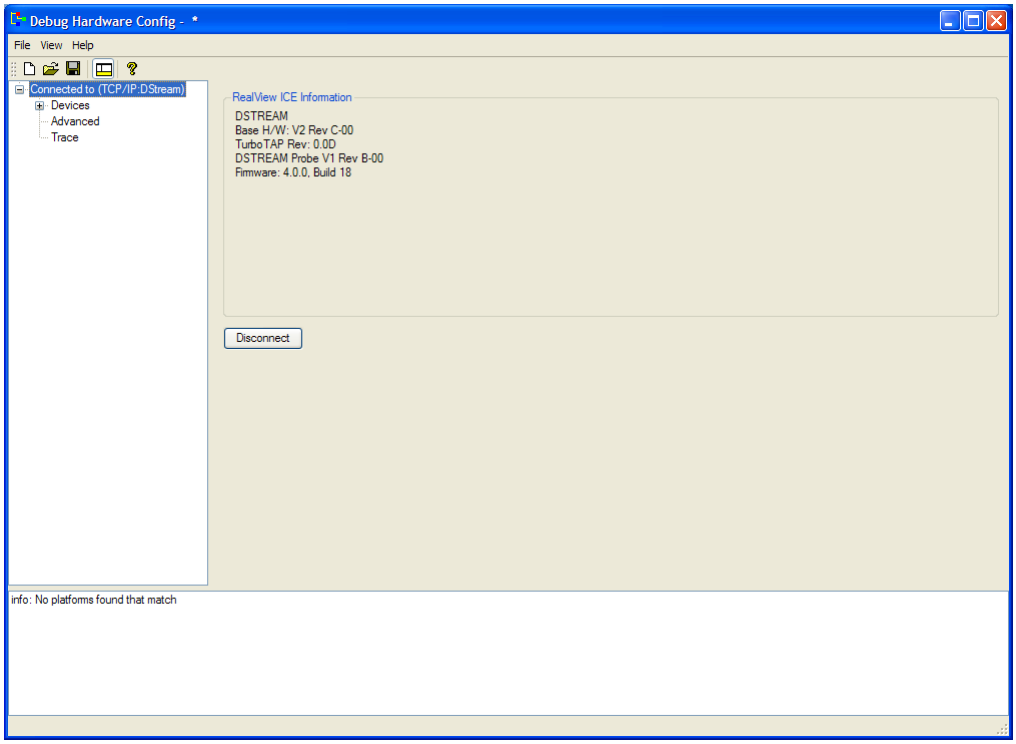

Figure 4-18 Displaying the connection controls

2. Click **Disconnect**.

   If you have unsaved configuration changes, a warning dialog box appears. The following figure shows an example:

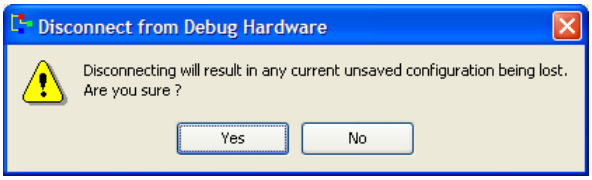

Figure 4-19 Warning when disconnecting with unsaved configuration changes

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.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.
4.1.1 Creating a debug hardware configuration file on page 4-48.
4.15.7 Saving your changes on page 4-88.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.17 Configuring a target processor for virtual Ethernet

For applications on your target to communicate over your network, you must configure virtual Ethernet on the target processor. This is required if the target does not have its own Ethernet hardware, or if its drivers have not yet been written.

Note

The network settings are supported only on ARM7™, ARM9™, ARM11, and ARM SecurCore processors using JTAG. They are not supported on CoreSight systems.

Procedure

1. Open the Debug Hardware Config utility.
2. Either:
   - Connect to and configure a debug hardware connection to create a debug hardware configuration file
   - Open an existing debug hardware configuration file.
3. Connect to the required debug hardware unit.
4. Select the **Devices** node in the tree diagram.
5. Select the processor that you want to configure for virtual Ethernet.
6. Set the network settings as required. The following table shows an example:

Table 4-1 Items for configuring static IP addresses

Device setting	Example
IP Address	`110.35.3.21`
Network Mask	`255.255.255.0`
Default Gateway	`110.35.3.254`

7. Select **File** > **Save** to save the changes.
8. Select **File** > **Exit** to close the Debug Hardware Config utility.

Related concepts

4.12 About debug hardware device configuration settings on page 4-71.

Related tasks

4.1.1 Creating a debug hardware configuration file on page 4-48.
1.8 Connecting to a debug hardware unit on page 1-19.
4.16 Disconnecting from a debug hardware unit on page 4-90.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

4.18 CoreSight™ device names and classes

The table shows the name and class of all CoreSight devices that are supported by the debug hardware.

Table 4-2 CoreSight device names and classes

Device name	Description	Device class
ARMCS-DP	ARM CoreSight Debug Port (supports both JTAG-DP and SW-DP)	Debug port
CSAUTHAP	CoreSight Authentication Access Port	
CSCTI	CoreSight *Cross Trigger Interface* (CTI)	
CSETB	CoreSight *Embedded Trace Buffer* (ETB)	Trace sink
CSETM	CoreSight *Embedded Trace Macrocell* (ETM)	Trace source
CSHTM	CoreSight *AHB Trace Macrocell* (HTM)	Trace source
CSITM	CoreSight *Instrumentation Trace Macrocell* (ITM)	Trace source
CSMTB	CoreSight *Micro Trace Buffer* (MTB)	Trace sink
CSPMU	CoreSight *Performance Monitor Unit* (PMU)	
CSPTM	CoreSight *Program Trace Macrocell* (PTM)	Trace source
CSREG	Provide generic support for your own CoreSight-compatible device that can be used by a debugger.	
CSSTM	CoreSight *System Trace Macrocell* (STM)	Trace source
CSSWO	CoreSight *Serial Wire Output* (SWO)	Trace sink
CSTFunnel	CoreSight *Trace Funnel*	Link
CSTMC	CoreSight *Trace Memory Controller* (TMC)	
CSTPIU	CoreSight *Trace Port Interface Unit* (TPIU)	Trace sink
ARM7EJ-S_JTAG-AP	ARM7EJ-S processor connected using JTAG-AP	Core
ARM7TDMI_JTAG-AP	ARM7TDMI processor connected using JTAG-AP	Core
ARM7TDMI_r4_JTAG-AP	ARM7TDMI revision 4 processor connected using JTAG-AP	Core
ARM926EJ-S_JTAG-AP	ARM926EJ-S processor connected using JTAG-AP	Core
ARM946E-S_JTAG-AP	ARM946E-S processor connected using JTAG-AP	Core
ARM966E-S_JTAG-AP	ARM966E-S processor connected using JTAG-AP	Core
ARM968E-S_JTAG-AP	ARM968E-S processor connected using JTAG-AP	Core
ARM9EJ-S_JTAG-AP	ARM9E-S processor connected using JTAG-AP	Core
ARM1136JFS-JTAG-AP	ARM1136JF-S processor connected using JTAG-AP	Core
ARM1156T2FS-JTAG-AP	ARM1156T2FS processor connected using JTAG-AP	Core
ARM1176JZF-JTAG-AP	ARM1156T2FS processor connected using JTAG-AP	Core
ARM11MPCore-JTAG-AP	ARM11MPCore processor connected using JTAG-AP	Core
Cortex-M*n*	Cortex-M processor	Core
Cortex-R*n*	Cortex-R processor	Core
Cortex-A*n*	Cortex-A processor	Core

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

The device names are the names used in debug hardware configuration files.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 5
Configuring CoreSight™ Systems

This chapter describes CoreSight systems and how to use Debug Hardware Config to configure them.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.1 About CoreSight™ system configuration

CoreSight systems consist of a *Debug Access Port* (DAP).

The DAP comprises of:

- A *debug port* (DP) that connects to the scan chain or *Serial Wire Debug* (SWD) interface, and provides the system interface to debug hardware.
- An *Advanced High-performance Bus Access Port* (AHB-AP for AHB access), or an *Advanced Peripheral Bus Access Port* (APB-AP for APB access).

Debug components are attached to the buses, and are accessed through the APs on the DAP. CoreSight debug components are configured with the index of the AP to which they are attached, and the base address on the bus.

The DAP can also contain a JTAG-AP that enables the connection of JTAG devices on internal scan chains, for example ARM11 processors. JTAG devices must be configured with the AP index of the JTAG-AP, the JTAG port on the AP, and the pre-bits and post-bits for both IR and DR scans for the particular device.

CoreSight components are associated with a DAP, so they are placed on the scan chain (or SWD connection) after the DAP with which they are associated. The DAP is represented on the scan chain by the ARMCS-DP device. CoreSight components are not on the JTAG scan chain, so they must be placed between the DAP and the next JTAG device.

Related concepts

1.2 About starting the debug hardware configuration utilities on page 1-12.
5.3 About CoreSight™ autodetection on page 5-97.
5.4 Autodetection in Serial Wire Debug mode on page 5-98.

Related tasks

5.2 Configuring CoreSight™ devices using a CoreSight ROM table on page 5-96.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.2 Configuring CoreSight™ devices using a CoreSight ROM table

If the target system contains a valid CoreSight ROM table, you can use this table to configure the AHB-AP and APB-AP devices behind the *Debug Access Port* (DAP).

To configure CoreSight devices using a CoreSight ROM table:

- If you are manually adding devices to the scan chain:
  1. Add the ARMCS-DP device.
  2. Right-click on the ARMCS-DP device, then select the **Read CoreSight ROM Table** option.
- If you are autoconfiguring scan chain, ensure that the **Read CoreSight ROM Tables** checkbox is selected before you auto configure. The checkbox is selected by default.

Related concepts

5.1 About CoreSight™ system configuration on page 5-95.

5.3 About CoreSight™ autodetection on page 5-97.

5.4 Autodetection in Serial Wire Debug mode on page 5-98.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.3 About CoreSight™ autodetection

Autodetecting a CoreSight system adds the ARMCS-DP device. The device represents the *Debug Access Port* (DAP) in a CoreSight system.

If the system contains a valid CoreSight ROM table, the devices that are listed in that table are also added by default.

The **Read CoreSight ROM Table** checkbox enables you to enable or disable the automatic reading of the table during an autoconfigration. The following figure shows the location of this checkbox:

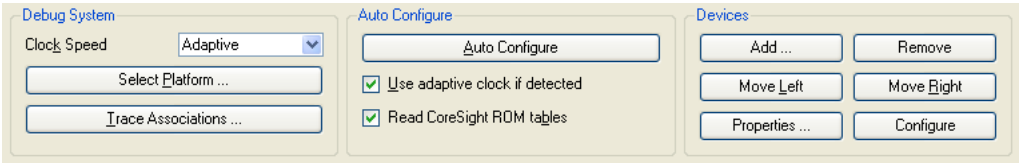

Figure 5-1 Read CoreSight ROM Table option

You can also force a read of the CoreSight ROM Table as follows:

1. Right-click on the ARMCS-DP device.
2. Select **Read CoreSight ROM tables** from the context menu.

Note

If you uncheck the **Read CoreSight ROM Table** checkbox, or the table read fails, you can manually add the CoreSight devices to the scan chain.

Related concepts

4.3 About configuring a device list on page 4-54.

5.1 About CoreSight™ system configuration on page 5-95.

5.4 Autodetection in Serial Wire Debug mode on page 5-98.

Related tasks

5.2 Configuring CoreSight™ devices using a CoreSight ROM table on page 5-96.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.4 Autodetection in Serial Wire Debug mode

Serial Wire Debug (SWD) does not support a scan chain in the same way that JTAG does.

When autoconfiguring in SWD mode, the debug hardware unit:

1. Adds the ARMCS-DP device to the scan chain configuration.
2. Optionally reads the CoreSight ROM Table to add the remaining devices. This is done by default, but you can override this.

Note

If your target supports both JTAG and SWD, then you must enable **Use SWJ Switching** in the Advanced configuration settings before autoconfiguring the scan chain.

Related concepts

5.1 About CoreSight™ system configuration on page 5-95.

5.3 About CoreSight™ autodetection on page 5-97.

Related tasks

5.2 Configuring CoreSight™ devices using a CoreSight ROM table on page 5-96.

Related references

4.12.3 Debug hardware Advanced configuration settings on page 4-77.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.5 Configuration items for processors in a CoreSight™ system

Figure showing an example of the CoreSight device settings for a processor and several configuration items.

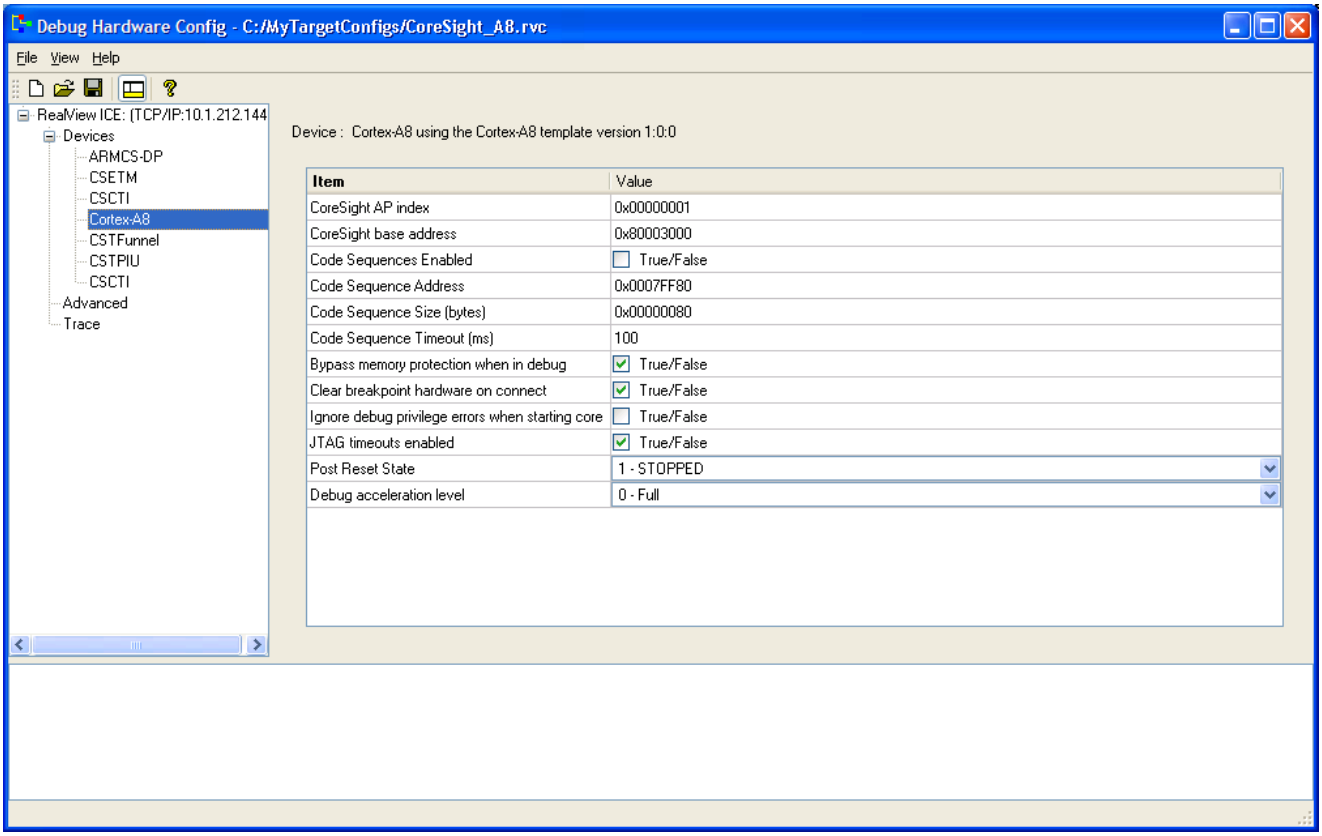

Figure 5-2 CoreSight device settings for a processor

The following additional configuration items are available when configuring CoreSight processors:

CoreSight AP index (CORESIGHT_AP_INDEX)
The index of the AP in the *Debug Access Port* (DAP) to be used for accessing the CoreSight debug registers for the CoreSight component.

CoreSight base address (CORESIGHT_BASE_ADDRESS)
For Cortex processors, this is the base address of the CoreSight debug registers on the *Advanced High-performance Bus* (AHB) or *Advanced Peripheral Bus* (APB) that is accessed through the AP as specified in the `CoreSight AP Index` configuration item.

Related concepts

5.6 Configuration items available for ARM7™, ARM9™, and ARM11™ processors in a CoreSight system on page 5-100.

5.7 Configuration items for CoreSight™ systems with multiple devices per JTAG-AP multiplexor port on page 5-102.

4.13 Configuring SecurCore® processor behavior if the processor clock stops when stepping instructions on page 4-80.

4.14 Errors when configuring TrustZone® enabled processor behavior when debug privileges are reduced on page 4-81.

Related references

4.12.5 Debug hardware Advanced configuration reset options on page 4-79.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.6 Configuration items available for ARM7™, ARM9™, and ARM11™ processors in a CoreSight system

Several configuration items are available when configuring these processors in a CoreSight system.

When using ARM7, ARM9, ARM11, or ARM11MPCore processors in a CoreSight system, you must add the corresponding *JTAG Access Port* (JTAG-AP) for the processor behind the ARMCS-DP component. For example, add the `ARM1176JZF-S_JTAG-AP` device if you have an ARM1176JZF-S processor in your CoreSight system.

List of configuration items:

CoreSight AP index (CORESIGHT_AP_INDEX)
The index of the JTAG-AP in the DAP to be used for accessing the CoreSight debug registers for the CoreSight component.

JTAG-AP Port index for core (JTAG_PORT_ID)
Each JTAG-AP implements eight JTAG port, each with its own TDI, TDO, TMS, and so on. The port index refers to the JTAG to which your CoreSight component is connected.

Fast memory download (FAST_MEM_WRITES)
The **Fast Memory Download** option is available for those targets where the DAP and the Core are running fast enough to handle the data being sent to them by the debug hardware unit without the debug hardware unit having to check that each individual transaction with the DAP has been successful. The processor is behind the DAP, so all processor accesses have to go through the DAP. As a guide, this setting must not be set for those targets that are FPGA-based.

——— **Note** ———

With this option set, error checking is disabled. If any errors occur, you are not informed. If problems are encountered when downloading images, uncheck this option to resolve them.

———————————

Pre-scan IR bits for Devices after the core on the JTAG-AP scanchain
The total length of the JTAG *Instruction Registers* (IRs) for devices appearing after the debugged target processor on the scan chain.

For example, if there are three devices after the processor with IR lengths of 5, 7, and 11, then set this to 23.

Post-scan IR bits for Devices before the core on the JTAG-AP scanchain
The total length of the JTAG IRs for devices appearing before the debugged target processor on the scan chain.

For example, if there are two devices before the processor with IR lengths of 2 and 3, then set this value to 5.

Pre-scan DR bits for Devices after the core on the JTAG-AP scanchain
The total number of devices appearing after the debugged target processor on the scan chain.

For example, if there are three devices after the processor, then set this value to 3.

Post-scan DR bits for Devices before the core on the JTAG-AP scanchain
The total number of devices appearing before the debugged target processor on the scan chain.

For example, if there are two devices before the processor, then set this value to 2.

——— **Note** ———

The ARM7xx-JTAG-AP, ARM9xx-JTAG-AP, or ARM11xx-JTAG-AP devices represent the JTAG-AP in a CoreSight system. To debug CoreSight systems that have processors that are connected to the DAP, the debug unit must know the pre-scan and post-scan bits for JTAG operations.

Multiple devices on the JTAG scan chain are connected in series, with data flowing serially from TDI to TDO. This means that debugging a given target in the chain requires that certain pre-scan and post-scan

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

bits are used. These bits ensure that the other devices are not affected by the data that is directed at the target device, and that the data is positioned correctly in the serial scan for the target device.

Related concepts

5.7 Configuration items for CoreSight™ systems with multiple devices per JTAG-AP multiplexor port on page 5-102.

4.13 Configuring SecurCore® processor behavior if the processor clock stops when stepping instructions on page 4-80.

4.14 Errors when configuring TrustZone® enabled processor behavior when debug privileges are reduced on page 4-81.

Related references

5.5 Configuration items for processors in a CoreSight™ system on page 5-99.

4.12.5 Debug hardware Advanced configuration reset options on page 4-79.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

5.7 Configuration items for CoreSight™ systems with multiple devices per JTAG-AP multiplexor port

The debug hardware unit does not support auto-detection of devices behind a JTAG-AP. Therefore, you must manually specify the JTAG scan chain attributes so that the unit can put the other devices into BYPASS.

To debug CoreSight systems that have processors connected to the *Debug Access Port* (DAP) through JTAG-AP, debug hardware must know the pre-bits and post-bits for JTAG operations. The following figure shows a hypothetical scan chain that could be connected to a JTAG-AP.

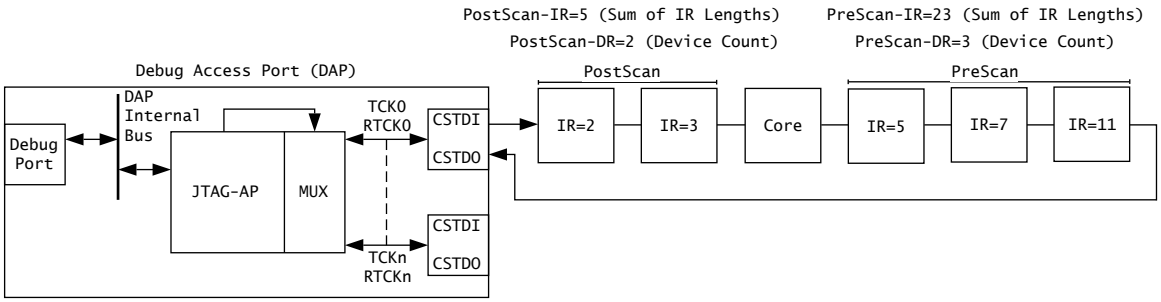

Figure 5-3 Scan chain that is connected to a JTAG-AP

Multiple devices on the scan chain are connected in series, with data flowing serially from TDI to TDO. This means that debugging a given target in the chain requires the use of certain pre-scan and post-scan bits to ensure that the other devices are not affected by the data directed at the target device, and that the data is positioned correctly in the serial scan for the target device.

To debug this system, you must set the following four configuration items:

- **Pre-scan IR bits for Devices after the core on the JTAG-AP scanchain (JTAG_AP_IR_PRE_BITS)**

  This is the total length of the JTAG *Instruction Registers* (IRs) for devices appearing between the processor being configured and the CSTDO input on the JTAG-AP port. In the figure above, the three devices that appear between the target processor and the CSTDO input on the JTAG-AP port have IR lengths 5, 7 and 11, respectively. Therefore, you must set this value to 23.
- **Post-scan IR bits for Devices before the core on the JTAG-AP scanchain (JTAG_AP_IR_POST_BITS)**

  This is the total length of the JTAG IRs for devices appearing between the CSTDI output on the JTAG-AP port and the processor being configured. In the figure above, the two devices that appear between the CSTDI output on the JTAG-AP port and the processor being configured have IR lengths 2 and 3, respectively. Therefore, you must set this value to 5.
- **Pre-scan DR bits for Devices after the core on the JTAG-AP scanchain (JTAG_AP_DR_PRE_BITS)**

  This is the total number of devices appearing between the processor being configured and the CSTDO input on the JTAG-AP port. In the figure above, there are three devices that appear between the processor being configured and the CSTDO input on the JTAG-AP port. Therefore, you must set this value to 3.
- **Post-scan DR bits for Devices before the core on the JTAG-AP scanchain (JTAG_AP_DR_POST_BITS)**

  This is the total number of devices appearing between the CSTDI output on the JTAG-AP port and the processor being configured. In the figure above, there are two devices that appear between the CSTDI output on the JTAG-AP port and the processor being configured. Therefore, you must set this value to 2.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related concepts

5.6 Configuration items available for ARM7™, ARM9™, and ARM11™ processors in a CoreSight system on page 5-100.

4.13 Configuring SecurCore® processor behavior if the processor clock stops when stepping instructions on page 4-80.

4.14 Errors when configuring TrustZone® enabled processor behavior when debug privileges are reduced on page 4-81.

Related references

5.5 Configuration items for processors in a CoreSight™ system on page 5-99.

4.12.5 Debug hardware Advanced configuration reset options on page 4-79.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 6
Using Trace

This chapter describes the trace features supported by your trace hardware.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

6.1 About using trace hardware

The trace data capture feature works with the debug run control feature in debug hardware. Together, they provide real-time trace functionality for software running in leading edge *System-on-Chip* (SoC) devices with deeply embedded processors that contain the *Embedded Trace Macrocell* (ETM) logic.

Trace hardware is included in a DSTREAM unit.

The trace functionality enables:

- Collection of trace information at clock speeds of up to 480MHz.
- Provision of a data streaming capability through a USB2 interface. This enables profiling directly from a hardware platform, with a debug hardware unit.

Related concepts

6.4 About configuring your debugger for trace capture on page 6-109.

Related tasks

6.3 Configuring trace lines (DSTREAM unit only) on page 6-107.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

6.2 Trace hardware capture rates

If a *Trace Port Interface Unit* (TPIU) in continuous mode is used, all port widths from 1 to 32 can be output by the target. DDR clocking enables data to be output from the ETM on both edges of **TRACECLK**. This effectively halves the clock frequency for the same data rate.

This section contains the following subsections:

6.2.1 DSTREAM trace hardware capture rates

The maximum capture rate of the DSTREAM unit is 600Mbps per trace signal using a 300MHz DDR clock signal. The capture buffer is 4GB in size.

Note

Some debuggers have limitations when tracing with DSTREAM. See the DS-5 Debugger documentation for details of the trace capabilities.

Related concepts

6.4 About configuring your debugger for trace capture on page 6-109.

6.1 About using trace hardware on page 6-105.

Related tasks

6.3 Configuring trace lines (DSTREAM unit only) on page 6-107.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

6.3 Configuring trace lines (DSTREAM unit only)

If you have a DSTREAM unit, you can configure delays on the trace lines using the Debug Hardware Config utility.

Procedure

1. Open the Debug Hardware Config utility.
2. If you have already configured a debug hardware unit, select **File** > **Open** to locate and open the corresponding configuration file.

   Otherwise:
   - Connect to the required debug hardware unit.
   - Create a debug hardware configuration.
3. Select the **Trace** node in the tree control. The following figure shows an example:

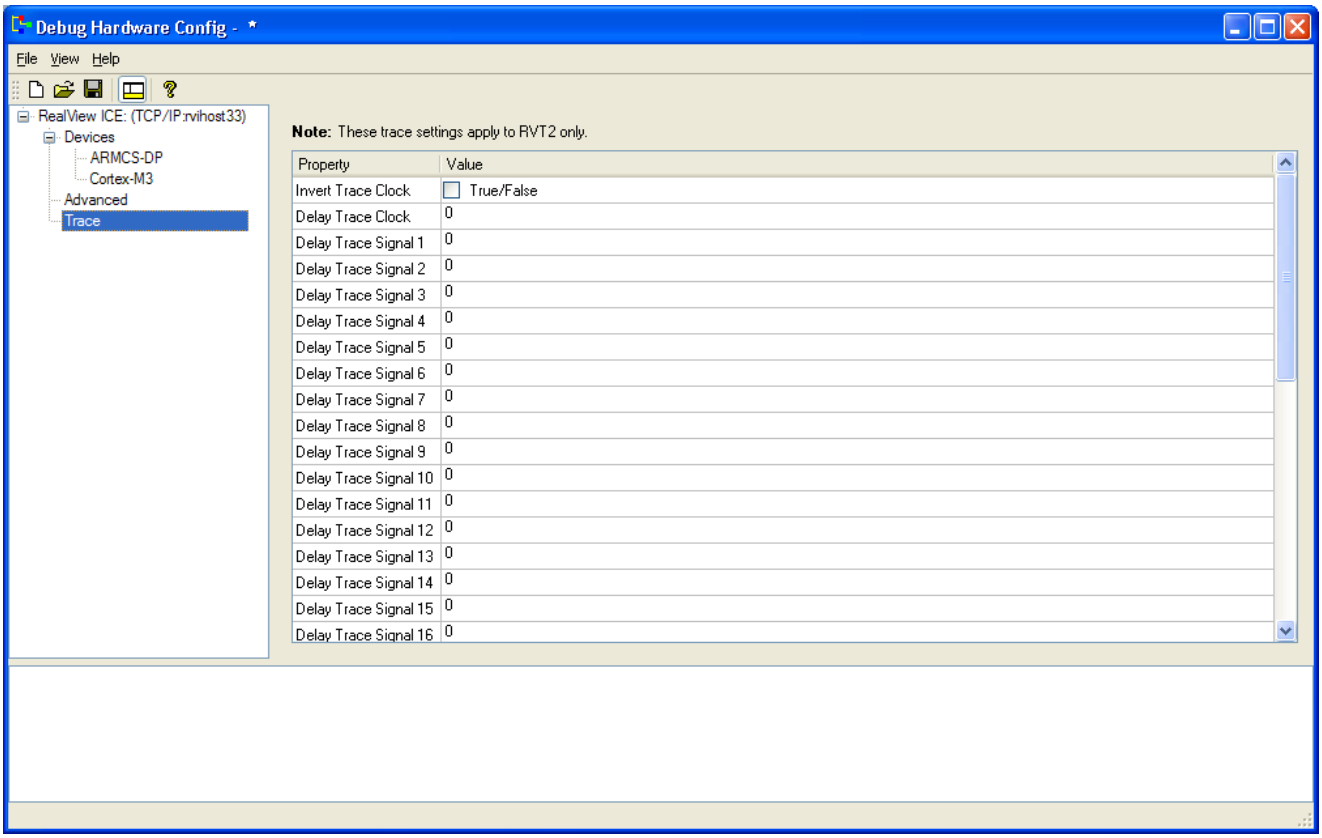

Figure 6-1 Trace node in Debug Hardware Config

You can delay each line by a specified amount of time (expressed in picoseconds, in 75ps intervals). Default delays are configured into the unit, and you are able to delay each signal by a specified amount relative to these defaults, allowing for variations in target hardware.

You can also invert the clock so that data is sampled on the falling edge (rather than on the rising edge) of the clock. To do this, select the **True/False** checkbox. The default is `False` (unchecked).

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Postrequisites

Note

Some debuggers have limitations when tracing with DSTREAM units. See the DS-5 Debugger documentation for details of the trace capabilities.

Related concepts

6.4 About configuring your debugger for trace capture on page 6-109.

6.1 About using trace hardware on page 6-105.

6.2 Trace hardware capture rates on page 6-106.

Related tasks

1.8 Connecting to a debug hardware unit on page 1-19.

4.1.1 Creating a debug hardware configuration file on page 4-48.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

6.4 About configuring your debugger for trace capture

When you have installed the host software and connected and configured the debug hardware unit, you must configure your debugger to use trace. For full details on how to capture trace with your debugger, see the documentation that accompanies your debugger.

Related concepts

6.1 About using trace hardware on page 6-105.
6.2 Trace hardware capture rates on page 6-106.

Related tasks

6.3 Configuring trace lines (DSTREAM unit only) on page 6-107.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 7
Debugging with your Debug Hardware Unit

This chapter provides information about debugging with debug hardware.

Note

For more general information about debugging, see the DS-5 Debugger documentation.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.1 Performing post-mortem debugging

Post-mortem debugging enables you to examine the state of a system that has previously been running but is not connected to debug hardware.

Prerequisites

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

The debug hardware 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 that is used by the debug hardware 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.

———————————

Procedure

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 debug hardware unit and wait for it to boot.
3. Configure the debug hardware connection. 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, such as an ARM7TDMI, does not have any system registers, you must explicitly configure the endianness.

   ——— **Note** ———

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

   ———————————

5. Plug the JTAG cable into the target.

   ——— **Caution** ———

   To prevent unwanted resets during connection of the debug hardware, you must ensure that either:
   - The target and debug hardware are properly earthed.
   - The ground pins of the debug connector make contact before the signal pins.

   ———————————

6. Start the debugger, and connect to the running target.

   In your debug hardware configuration, set the **Post Reset State** to **Running**.

   In your debugger, connect using the **Connect (Connection Modes)** of **No Reset / No Stop**.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7. To get a high-level (source code) view of the problem, load the symbol table for your target program into the debugger.
8. If the processor stopped during debugging, then in your debugger:
   - Clear any breakpoints that you have set.
   - Start the processor running.
   - Disconnect from all targets to which the debugger is connected on your development platform.

   If the processor is still running, then in your debugger disconnect from all targets to which the debugger is connected on your development platform.
9. Unplug the JTAG connector from the development platform.

Related concepts

7.2 Semihosting on page 7-113.

7.11 Strategies used by debug hardware to debug cached processors on page 7-123.

7.13 About debugging applications in ROM on page 7-125.

Related tasks

4.15.6 Configuring the debug hardware Advanced settings on page 4-87.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.2 Semihosting

Semihosting enables the ARM processor target to make I/O requests to the computer running the debugger. This means that 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()`.

Related concepts

7.3 About adding an application SVC handler when using debug hardware on page 7-114.

7.4 Cortex®-M3 semihosting on page 7-116.

7.11 Strategies used by debug hardware to debug cached processors on page 7-123.

7.13 About debugging applications in ROM on page 7-125.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.3 About adding an application SVC handler when using debug hardware

Many applications require their own SVC handlers in addition to semihosting SVCs. To ensure that the application SVC handler cooperates with the debug hardware semihosting mechanism, use your debugger.

You can:

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 an SVC and branch to an error handler. An example of a basic exception handler is shown in the following example.

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.
```

You can modify this code for use with debug hardware semihosting as shown in the following example.

SVC handler with debug hardware 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 executed because the debug 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 debug 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 used 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 with the Angel debug monitor onto a debug hardware 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 debug software because there is no instruction to move out of the SVC vector, and no code to jump to. Therefore, when moving an application onto a debug hardware-based system, you must convert to the debug hardware way of installing the application and semihosted SVC handlers.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Related concepts

7.4 Cortex®-M3 semihosting on page 7-116.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.4 Cortex®-M3 semihosting

Because Cortex-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 specific software breakpoint that is recognized as a semihosting break by the debugger.

The breakpoint instruction opcode contains an immediate 8-bit value. The C library uses the `BKPT 0xAB` opcode for semihosting. The debugger can test for this opcode pattern to determine if the breakpoint was a semihosting request or not.

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.

Related concepts

7.3 About adding an application SVC handler when using debug hardware on page 7-114.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.5 Hardware breakpoints

Depending on implementation options, most ARM processors contain dedicated hardware resources, such as ARM EmbeddedICE logic, for matching against specific hardware events. Your 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 extra 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.

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

Related concepts

7.6 Software instruction breakpoints on page 7-118.
7.7 Processor exceptions on page 7-119.
7.8 Breakpoints and the program counter on page 7-120.
7.9 Interaction between breakpoint handling in the debug hardware and your debugger on page 7-121.
7.10 Problems setting breakpoints on page 7-122.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.6 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 that is 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:

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

When using a hardware breakpoint unit 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.

You cannot use software breakpoints to debug code in Flash or ROM, and can be unreliable in self-modifying code.

Note

When viewing memory or disassembly, debug hardware reports the actual contents of memory. Before running, any software breakpoints are written to memory. When the processor halts, the software breakpoints are removed from memory. On several processors, it is not possible to access memory while running. This means that if you disconnect debug hardware from the processor while the target is running, the breakpoints are left in memory. If the processor later executes one of the instructions, then (depending on the processor architecture) the processor either stops at the software breakpoint or causes the processor to take an undefined exception.

Related concepts

7.5 Hardware breakpoints on page 7-117.
7.7 Processor exceptions on page 7-119.
7.8 Breakpoints and the program counter on page 7-120.
7.9 Interaction between breakpoint handling in the debug hardware and your debugger on page 7-121.
7.10 Problems setting breakpoints on page 7-122.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.7 Processor exceptions

Depending on implementation options, most ARM processors provide dedicated hardware to enter debug state when a predetermined event occurs. Most recent ARM processors provide hardware for trapping exceptions.

When enabled, the effect is similar to placing a breakpoint on the selected vector table entry. This is called *vector catch*. However:

- Some ARM processors, such as ARM7 processors, do not provide vector catch hardware. For these processors, debug hardware simulates vector catch using instruction breakpoints.
- For Cortex-M3, this is equivalent to putting a breakpoint at the target of the vector. Cortex-M3 has a restricted set of vector catches available.
- If the exception vectors are in ROM, debug hardware must use hardware breakpoints to simulate vector catch. This reduces the number of resources available for other purposes, if the processor does not have vector catch support.

When the debug hardware simulates vector catch on earlier ARM processors that do not have vector catch support, it uses a software breakpoint when the vector table is located in RAM.

You must take care when debugging through a system reset. Some hardware targets alter the memory map after reset, so the location of in physical memory containing software breakpoint might not be in the correct reset position. The following warning is output to the debugger console if debug hardware 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 processor. ARM9 processors enter debug state only when the specified exception occurs. Other processors, such as ARM11 or older processors that use a breakpoint, enter debug state whenever the instruction at the exception vector is executed, regardless of whether the exception occurs or not.

Related concepts

7.5 Hardware breakpoints on page 7-117.
7.6 Software instruction breakpoints on page 7-118.
7.8 Breakpoints and the program counter on page 7-120.
7.9 Interaction between breakpoint handling in the debug hardware and your debugger on page 7-121.
7.10 Problems setting breakpoints on page 7-122.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.8 Breakpoints and the program counter

Several events describe the value of the program counter when a breakpoint is taken.

Hardware data breakpoints

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

For ARM processors, 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 your 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 processors, 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.

Related concepts

7.5 Hardware breakpoints on page 7-117.
7.6 Software instruction breakpoints on page 7-118.
7.7 Processor exceptions on page 7-119.
7.9 Interaction between breakpoint handling in the debug hardware and your debugger on page 7-121.
7.10 Problems setting breakpoints on page 7-122.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.9 Interaction between breakpoint handling in the debug hardware and your debugger

The following describe the interaction between breakpoint handling in the debug hardware and breakpoint handling in your debugger.

Break details or break capabilities

You can find out what hardware breakpoint resources are available by viewing the break details or break capabilities in your debugger.

Memory maps

Your debugger enables you to define a memory map to describe the layout and type of memory in your system. 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. Debug hardware uses this hardware if it is available, but steps larger than a single instruction might revert to using breakpoints, to improve efficiency.

——— **Note** ———

For ARM7, ARM9, ARM11, or Cortex-A8 processors, interrupts are disabled when single-stepping with debug hardware. For the Cortex-M3 processor, interrupts are enabled when single-stepping with debug hardware.

Interrupt behavior applies only to debug hardware single-instruction stepping. Higher level stepping depends on the strategy in your debugger, that is, whether you have used either of the:

- place breakpoint and run method.
- multiple single-instruction steps method.

———————————

——— **Note** ———

When hardware single-step is used, debug hardware prevents the processor from processing any pending interrupts.

———————————

Resource allocation

Debug hardware 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, debug hardware displays an error message when you try to set the breakpoint, rather than waiting until debugging begins.

Related concepts

7.2 Semihosting on page 7-113.
7.5 Hardware breakpoints on page 7-117.
7.6 Software instruction breakpoints on page 7-118.
7.7 Processor exceptions on page 7-119.
7.8 Breakpoints and the program counter on page 7-120.
7.10 Problems setting breakpoints on page 7-122.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.10 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 vector catch if you are debugging an early processor, such as the ARM7TDMI, and the vector table is in ROM.
- Disable semihosting if you are not using it.

Related concepts

7.5 Hardware breakpoints on page 7-117.

7.6 Software instruction breakpoints on page 7-118.

7.7 Processor exceptions on page 7-119.

7.8 Breakpoints and the program counter on page 7-120.

7.9 Interaction between breakpoint handling in the debug hardware and your debugger on page 7-121.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.11 Strategies used by debug hardware to debug cached processors

When debugging a cached processor, the debug hardware uses different strategies.

On debug entry

- Debug hardware forces *Write-Through* (WT) on processors that support this debug feature.
- Debug hardware disables cache line fill on processors that support disabling of this feature in debug.
- Debug hardware disables *Translation Lookaside Buffer* (TLB) loads on processors that support disabling of this feature in debug.
- If data is read from cacheable memory, it is only read into the caches if, and only if, disable linefill is not possible.
- TLB entries and caches remain enabled.

On data write

- If WT is possible, nothing cache-related is performed.
- If WT is not possible, the write depends on processor size and data size:
  1. Debug hardware can write to memory with caches enabled, and then write disabled, effectively simulating write through.
  2. Debug hardware can clean and invalidate the D_{cache} and disable it.

     Note

     The ARM940T processor requires that Code Sequences are enabled to do this.

On restart into debug

- On processors that 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 that could cause the cache operations described include user accesses using your debugger, downloads, and any software breakpoints present in the system.

Note

For the ARM940T processor, you must configure the code sequence settings before attempting to debug with caches enabled.

When the cache is enabled, the speed of semihosting decreases, because of the additional cache maintenance overhead performed by the debugger.

Related concepts

7.2 Semihosting on page 7-113.
7.13 About debugging applications in ROM on page 7-125.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.12 Considerations when debugging processors with caches enabled

When debugging a processor with caches enabled, you might have to provide the address of an area of memory on the target that can be used exclusively by debug hardware.

On some targets, the debug software downloads code sequences to this area to perform various tasks, such as cleaning the cache, and accessing the system registers. Debug hardware does not preserve the contents of this area.

A code sequence area is only required for certain processors where the required operations cannot be performed directly over JTAG. If debug hardware requires a code sequence area, and one has not been enabled, errors are displayed within the debugger. For example:

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

Note

The code sequence area must be 128 bytes long and in a non-cacheable, readable, and writable area.

To set up a code sequence area, use the options for each specific processor in the Debug Hardware Config utility. This provides access to configuration items for each processor for:

- Enabling code sequences.
- Setting the address and size of the code sequence areas.

Note

These settings might also be available in your debugger Registers view. Any settings modified using the Registers view in your debugger are only modified during the debug session. Any settings modified using the Debug Hardware Config utility are persistent until modified again.

Related concepts

7.2 Semihosting on page 7-113.
7.11 Strategies used by debug hardware to debug cached processors on page 7-123.
7.13 About debugging applications in ROM on page 7-125.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.13 About debugging applications in ROM

Some of the issues involved with debugging applications in ROM using debug hardware are related to debugging from reset or debugging systems with ROM at the exception vector.

This section contains the following subsections:

7.13.1 About debugging from reset

You can debug systems running in ROM. A typical embedded system has the application programmed in non-volatile memory, such as ROM or Flash.

When target hardware is powered up, the application starts running. When connecting the debugger to a target already running the application, the application stops at an arbitrary point in the code. The default behavior of the debugger is to stop the target on connection. Loading the image symbols gives you source code view of the current location.

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, often it is preferable to restart execution of the application as if from power-on. There are three ways to do this:

- Debugging with a simulated a reset.
- Debugging with a reset register.
- Debugging with a target reset.

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.

On ARM processors that use a breakpoint resource to implement software breakpoints, such as the ARM7TDMI, you must remove all software breakpoints if you are out of breakpoint resources. This enables you to place a single hardware breakpoint in ROM.

Another factor in debugging a system in ROM is that the ROM image does not contain any debug information. When debugging, symbol or source code information is available by loading the relevant information into the debugger from the ELF image on the host.

Related concepts

7.13.2 About debugging with a simulated reset on page 7-125.

7.14 About debugging with a reset register on page 7-127.

7.15 About debugging with a target reset on page 7-128.

7.16 About debugging systems with ROM at the exception vector on page 7-129.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

7.13.2 About debugging with a simulated reset

You can, where supported, simulate a reset from within the debugger.

To simulate a reset from within the debugger:

- Set the PC to the address of the reset vector.
- Set 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

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

before executing the application again, if possible. You can automate this procedure using the scripting facilities of your debugger.

Related concepts

7.13.1 About debugging from reset on page 7-125.

7.14 About debugging with a reset register on page 7-127.

7.15 About debugging with a target reset on page 7-128.

7.16 About debugging systems with ROM at the exception vector on page 7-129.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Related concepts

7.2 Semihosting on page 7-113.

7.11 Strategies used by debug hardware to debug cached processors on page 7-123.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.14 About debugging with a reset register

Where supported, some processors, particularly CoreSight types, such as the Cortex-M3, include a reset register that can reset the processor without using the physical reset lines. In a multi-processor system, you can use this register to reset only the target processor and not the complete system.

To select this type of reset, set the reset type to `Ctrl_Reg`.

Related concepts

7.13.1 About debugging from reset on page 7-125.
7.13.2 About debugging with a simulated reset on page 7-125.
7.15 About debugging with a target reset on page 7-128.
7.16 About debugging systems with ROM at the exception vector on page 7-129.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.15 About debugging with a target reset

Depending on the design of the reset circuitry, you might be able to carry out a target reset of the board.

Two forms of reset are required on the board.

- A full power-on reset resets everything on the board.
- A Reset button resets your development platform depending on whether it is a CoreSight system:
  - For non-CoreSight systems, everything on the board is reset except the EmbeddedICE logic. The EmbeddedICE logic is the debug logic in the processor.
  - In a CoreSight system, the **nSRST** (system reset) resets the entire design, except for the debug subsystem and trace subsystem. This includes the debug logic from all devices, debug and trace bus, access ports, and *Debug Access Port* (DAP).

Note

The Reset button mentioned here must not be confused with the RESET button located on the debug hardware unit itself.

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.

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.

Related concepts

7.13.1 About debugging from reset on page 7-125.
7.13.2 About debugging with a simulated reset on page 7-125.
7.14 About debugging with a reset register on page 7-127.
7.16 About debugging systems with ROM at the exception vector on page 7-129.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Related information

The DSTREAM unit.
ARM DSTREAM System Design Guidelines.
RVI Debug Unit System Design Guidelines.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

7.16 About 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 debug hardware from trying to set hardware breakpoints on the vector table.

The default setting is to enable, exception trapping on reset, Prefetch Abort, and Data Abort. On a processor without vector catch hardware, three breakpoint resources are used in ROM. Therefore, if a processor has three or fewer resources, you cannot debug applications running on that processor.

Related concepts

7.13.2 About debugging with a simulated reset on page 7-125.
7.14 About debugging with a reset register on page 7-127.
7.15 About debugging with a target reset on page 7-128.

Related tasks

7.1 Performing post-mortem debugging on page 7-111.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Chapter 8
Troubleshooting your Debug Hardware Unit

This chapter describes what to do when you encounter problems when attempting to connect to a debug hardware unit or upgrade the firmware.

It contains the following sections:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.1 Situation when multiple programs are attempting to scan

Only one program on each host computer can scan the TCP/IP network or USB ports for available debug hardware units. If another configuration utility is scanning, an error message is displayed.

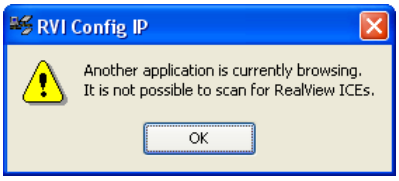

Figure 8-1 Error message when another program is scanning

You must stop the other configuration utility from scanning. To do this, click the **Scan** button, or select **RVI** > **Stop Scan** from the menu in the configuration utility that you want to stop scanning.

Related concepts

8.2 The USB server is not accessible on page 8-132.
8.4 The debug hardware unit has other active connections on page 8-134.

Related tasks

1.6 Scanning for available debug hardware units on page 1-16.
1.8 Connecting to a debug hardware unit on page 1-19.
8.3 Fixing connection time-out problems on page 8-133.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.2 The USB server is not accessible

An error message is displayed if a communication error occurs between the Debug Hardware Config utility and the USB server application.

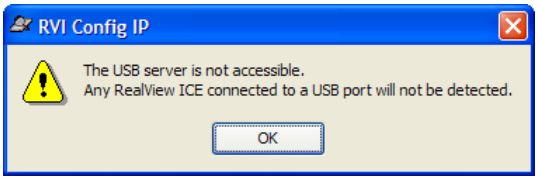

Figure 8-2 Error message when no USB devices present

The USB server might not be accessible because:

- The USB driver did not load. In this case, you might have to reinstall the host software.
- The USB server application has stopped working. In this case, kill the USB server process and restart the Debug Hardware Config utility.

Note

If you do not want to use a USB connection, then use the debug hardware unit over a TCP/IP connection. If you have not yet done so:

1. Connect the unit to your network.
2. Configure the network settings for the unit.

Related concepts

2.1 About configuring network settings on page 2-22.

8.1 Situation when multiple programs are attempting to scan on page 8-131.

8.4 The debug hardware unit has other active connections on page 8-134.

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.

8.3 Fixing connection time-out problems on page 8-133.

Related references

2.2 Debug hardware network settings on page 2-23.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.3 Fixing connection time-out problems

The default timeout for establishing a TCP/IP connection is five seconds. You can change this setting, if you repeatedly get timeouts when attempting to connect to a debug hardware unit.

Procedure

1. Create the environment variable `RVI_COMMS_CONNECT_TIMEOUT` if it does not already exist.
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.

Postrequisites

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.

Note

Be aware that TCP/IP latency might affect the response times and performance of your debugger.

Related concepts

8.1 Situation when multiple programs are attempting to scan on page 8-131.
8.2 The USB server is not accessible on page 8-132.
8.4 The debug hardware unit has other active connections on page 8-134.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.4 The debug hardware unit has other active connections

If you connect to a debug hardware unit that has other active connections, an error message is displayed.

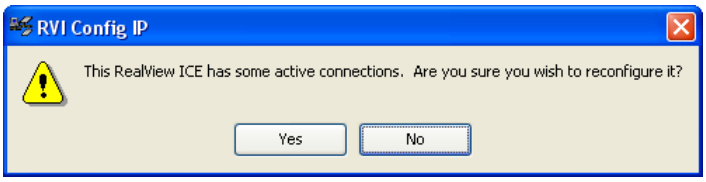

Figure 8-3 Error when other connections are active

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

- Ensure that the other configuration utilities are disconnected, then click **Yes** to continue using the configuration utility.
- Click **No** to stop using the configuration utility, and try again later.

Related concepts

8.1 Situation when multiple programs are attempting to scan on page 8-131.
8.2 The USB server is not accessible on page 8-132.

Related tasks

8.3 Fixing connection time-out problems on page 8-133.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.5 Reasons why a debug hardware unit is not listed

A debug hardware unit might not be listed in the debug hardware utility for several reasons. You can try different solutions to resolve the issues.

- If the debug hardware unit is connected to a USB port, install the USB drivers.
- The unit is on a different subnet to your PC. In this case, obtain either the IP address or the host name of the unit and enter the value in the IP Address / Host Name field of the utility you are using.
- The unit has not yet been configured for use on a network. Configure the network settings if you have privilege to do so. Otherwise, contact the person responsible for the unit.
- The unit did not boot correctly. Reboot the unit. If the unit is on the same subnet as your PC, it appears in the list of units when the reboot is successful.

Related tasks

2.6 Configuring the network settings for a debug hardware unit on page 2-29.

Related references

2.2 Debug hardware network settings on page 2-23.

Related information

Setting up the ARM DSTREAM Hardware - Installing the USB drivers for your debug hardware unit.

Setting up the ARM RVI Hardware - Installing the USB drivers for your debug hardware unit.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.6 Auto Configure button is disabled in the Debug Hardware Config utility

The **Auto Configure** button is disabled in the Debug Hardware Config utility when you have a platform that is assigned to your debug hardware configuration.

If you want to autoconfigure the scan chain again:

1. Click **Clear Platform**.
2. Click **Auto Configure**.

Related concepts

4.4 About the scan chain on page 4-58.

Related tasks

4.3.4 Autoconfiguring a scan chain on page 4-56.
4.15.3 Clearing a platform assignment from a debug hardware configuration on page 4-85.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.7 Remove button is disabled in the Debug Hardware Config utility

The **Remove** button is disabled in the Debug Hardware Config utility when you have a platform that is assigned to your debug hardware configuration.

If you want to remove a device from the scan chain:

1. Click **Clear Platform**.
2. Either:
   - Autoconfigure the scan chain again.
   - Manually add only those devices you want to keep.

Related concepts

4.4 About the scan chain on page 4-58.

Related tasks

4.3.4 Autoconfiguring a scan chain on page 4-56.
4.15.3 Clearing a platform assignment from a debug hardware configuration on page 4-85.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.8 Troubleshooting firmware upgrade installations

The main types of error that might occur during installation of a firmware patch are version problems and errors during file operation on the host.

This section contains the following subsections:

8.8.1 Troubleshooting version problems

A patch targets a particular `major.minor` release version of the software.

The patch 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, or if a patch has no new components, different dialog boxes appear to inform you of the situation.

- If the patch targets a version of the software that is not installed, a dialog box appears:

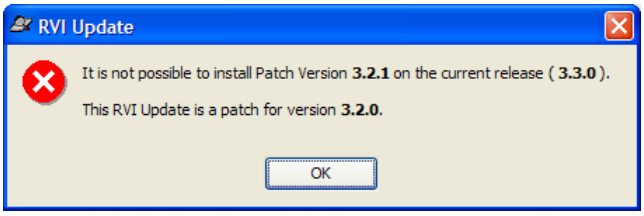

Figure 8-4 Error when installing a patch to uninstalled software

  In this case, the patch is not installed, and the software on the debug hardware unit remains unchanged. Make sure that you have the patch for the version of the firmware that you have installed.
- If the patch does not contain any new or updated components (typically because a later patch has already been installed), a dialog box appears:

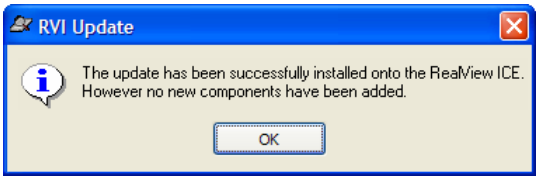

Figure 8-5 Message when installing a patch that has no new components

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

Related concepts

3.3 List of software version numbers on page 3-38.

3.1 About templates and firmware files on page 3-36.

8.8.2 Troubleshooting errors during file operation on the host

If an error occurs during file operation on the host, different dialog boxes appear to inform you of the specific problem.

- If the error occurs before any data has been written to the compact flash, the dialog box shown in the following figure appears:

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

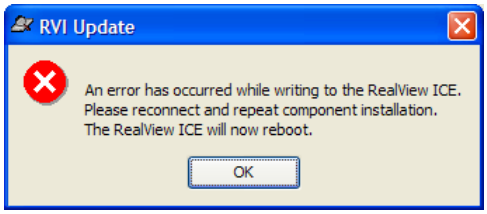

Figure 8-6 Error before data has been written to compact flash

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

- If some data has already been written to the compact flash when the error occurs, the debug hardware unit must reboot to clean up the failed installation and revert to the backed-up state. The dialog box shown in the following figure appears:

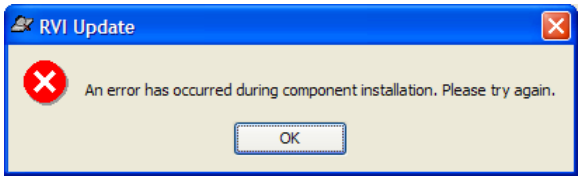

Figure 8-7 Error during writing to compact flash

Click **OK**. The debug hardware unit reboots. You can then begin the installation again.

Related concepts

3.3 List of software version numbers on page 3-38.
3.1 About templates and firmware files on page 3-36.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.9 Troubleshooting autoconfiguration of a scan chain

This describes the errors you might encounter when autoconfiguring a scan chain.

- If debug hardware detects any unpowered devices, it displays the error shown in the following figure:

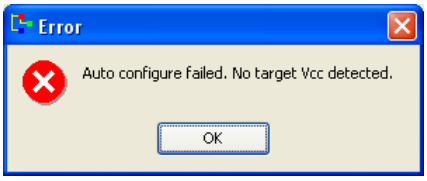

Figure 8-8 Error shown when unpowered devices are detected

  This error message can also display if the target is connected by the JTAG ribbon cable if debug hardware is started when the *Low Voltage Differential Signaling* (LVDS) probe is connected, or if the probe is connected and used after you started debug hardware.

  If you see this error:
  — Check the JTAG connection between the debug hardware unit and the target hardware.
  — Ensure that power is supplied to all your devices.
- If debug hardware cannot identify any devices, it displays the error shown in the following figure:

Figure 8-9 Error shown when no devices are detected

  If you see this error:
  — Manually configure the scan chain if your target has unsupported devices.
  — Try auto-configuring again with a lower clock speed.

  Note

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

- The Read ROM Table phase for a CoreSight system fails to find any devices. This might be because the ROM table is corrupt. Manually configure the scan chain.
- If communication cannot be made with the debug hardware unit in your current configuration, it displays the error shown in the following figure.

Figure 8-10 Error shown when there is no communication with debug hardware

  If you see this error, it might mean that the debug hardware unit in your configuration file no longer exists, or has been configured with different network settings.

Related concepts

4.4 About the scan chain on page 4-58.

Related tasks

4.3.4 Autoconfiguring a scan chain on page 4-56.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

8.10 Log Client Utility

The Log Client utility receives logging messages from a DSTREAM or RVI unit. The unit can be connected over a network or USB.

Output from all logging processes is displayed.

Syntax

```
dbghw_log_client unit_identifier
dbghw_log_client -help
```

Where *unit_identifier* is the identifier for the DSTREAM or RVI unit. It can be in any of the following formats:

IP Address Specify either:

xxx.xxx.xxx.xxx

TCP:*xxx.xxx.xxx.xxx*

For example, **TCP:100.16.89.12**

Hostname Specify either:

unit_name

TCP:*unit_name*

For example, **my-debug-unit**.

USB Specify either

USB

USB:*usb_port_number*

usb_port_number is the number that is shown in the tree control or the scan chain schematic diagram of the Debug Hardware Config utility, for example:

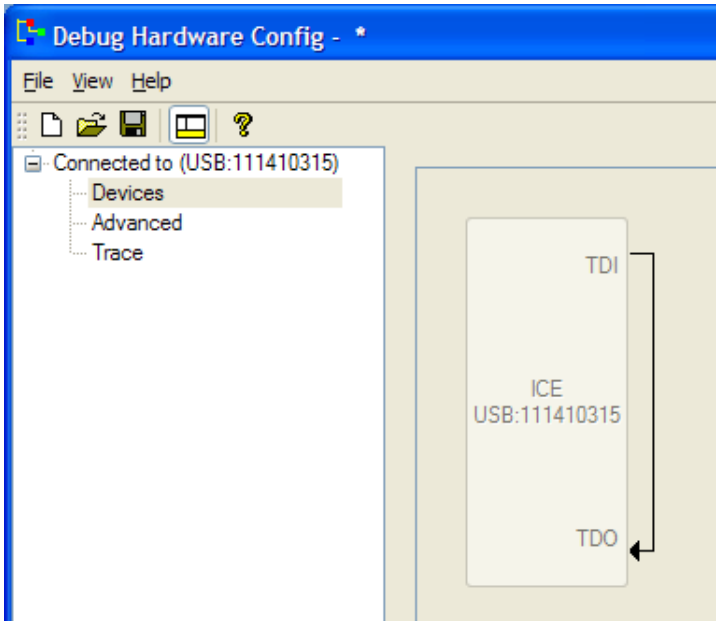

Figure 8-11 USB port number in the Debug Hardware Config utility

In this example, the port number is 111410315.

Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

Usage

Specify `-h` or `-help` to display the help message and usage information.

Press Ctrl+C to close the Log Client utility.

Note

When using RVI Update, make sure that you close the Log Client before trying to update. If you leave the Log Client running while the update is in progress, the update takes longer than normal. However, the update still completes.

Example

The following example shows the logging output when connecting to an RVI unit and autoconfiguring a scan chain that has a single ARM926EJ-S processor.

```
C:\Program Files\DS-5\bin>dbghw_log_client USB:111410315

ARM DSTREAM/RVI Log Client.
Copyright 2001-2015 ARM Limited. All rights reserved.
V1.3.0

Connected. Now building process list, please wait...
Process List built. Connected and waiting for logging.

usbd           info     CUsbClientCon: created udp listen port 30000
usbd           info     CUsbClientCon: created udp listen port 30000
serverd        error    CLVDSProbe::ReadEEPROM: Failed to read probe EEPROM (probably no
probe plugged in)
serverd        error    CLVDSProbe::ReadEEPROM: Failed to read probe EEPROM (probably no
probe plugged in)
rifsd          error    CRIFSCmdProc::CommandFILEOPEN open failed file(/real-ice/backup/
_version_) errno=2 {No such file or directory}
serverd        error    CLVDSProbe::ReadEEPROM: Failed to read probe EEPROM (probably no
probe plugged in)
cfgqueryd      info     CJTAGDriver::DetectRTCLK did detect RTCLK
cfgqueryd      info     CFPGA_RVIHW::SetJTAGClock @ 10000000 Hz, not using RTCK
cfgqueryd      info     CClkCtrl::SetJTAGClkFreq requested freq=10000000; using
freq=80000000 with FPGADiv=2
cfgqueryd      info     CJTAGDriver::DetectRTCLK did detect RTCLK
cfgqueryd      info     CFPGA_RVIHW::SetJTAGClock @ 50000000 Hz, using RTCK
cfgqueryd      info     CClkCtrl::SetJTAGClkFreq requested freq=50000000; using
freq=100000000 with FPGADiv=0
cfgqueryd      info     CFPGA_RVIHW::SetJTAGClock @ 50000000 Hz, using RTCK
cfgqueryd      info     CClkCtrl::SetJTAGClkFreq requested freq=50000000; using
freq=100000000 with FPGADiv=0
cfgqueryd      info     CRVIAutoDetect::MeasureDeviceCount measuring the length of DR chain
in BYPASS
cfgqueryd      info     DR Chain 0..01..1 filled[1024]:
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000011111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
0

cfgqueryd      info     CRVIAutoDetect::MeasureTotalDRLength Measured total DR length as 1
cfgqueryd      info     CRVIAutoDetect::MeasureDeviceCount Measured JTAG device count as 1
cfgqueryd      info     DR Chain after tap reset[32]: 00000111100100100110000001000001
cfgqueryd      info     Device 0 has IDCODE = 0x07926041
cfgqueryd      info     CRVIAutoDetect::deviceCountForIDCode device count for ID 0x07926041
is 1
cfgqueryd      info     CRVIAutoDetect::MeasureScanChainLength returned 1 devices
cfgqueryd      info     CFPGA_RVIHW::SetJTAGClock @ 50000000 Hz, using RTCK
cfgqueryd      info     CClkCtrl::SetJTAGClkFreq requested freq=50000000; using
freq=100000000 with FPGADiv=0
cfgqueryd      info     CRVIAutoDetect::autodetect_config measuring the total IR length
cfgqueryd      info     IR Chain 1..10..0 filled[1024]:
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```


Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential

```
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111110000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
1


cfgqueryd      info     CRVIAutoDetect::MeasureTotalIRLength Measured total IR length as 4
cfgqueryd      info     CRVIAutoDetect::autodetect_config measured the total IR length = 4
cfgqueryd      info     CRVIAutoDetect::autodetect_config measuring the number of taps
cfgqueryd      info     CRVIAutoDetect::MeasureDeviceCount measuring the lengthof DR chain
in BYPASS
cfgqueryd      info     DR Chain 0..01..1 filled[1024]:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000001111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
0

cfgqueryd      info     CRVIAutoDetect::MeasureTotalDRLength Measured total DR length as 1
cfgqueryd      info     CRVIAutoDetect::MeasureDeviceCount Measured JTAG device count as 1
cfgqueryd      info     DR Chain after tap reset[32]: 00000111100100100110000001000001
cfgqueryd      info     Device 0 has IDCODE = 0x07926041
cfgqueryd      info     AutoDetect: get_ir_data, Dev 0 IRLen=4
cfgqueryd      info     CRVIAutoDetect::autodetect_config measured the number of taps = 1
cfgqueryd      info     Device 0 detected as ARM926EJ-S from STMicroelectronics
cfgqueryd      info     DR Chain 0..01..1 filled[1024]:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000011111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111000
0
cfgqueryd      info     CRVIAutoDetect::MeasureTotalDRLength Measured total DR length as 5
cfgqueryd      info     CRVIAutoDetect::get_scan_chain_length Detecting length of scan chain
6
cfgqueryd      info     DR Chain 0..01..1 filled[1024]:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000011111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111100000000101000011001000000001000000000
1

cfgqueryd      info     CRVIAutoDetect::MeasureTotalDRLength Measured total DR length as 40
^C
C:\Program Files\DS-5\bin>
```


Copyright © 2010-2012, 2015 ARM. All rights reserved.
Non-Confidential