RVDS 3.1 RealView ICE & RealView Trace Quickstart Tutorial



250v02

Contents

2
3
4
3
3
5
20
24
24
0
0
7

Introduction

Aim

This tutorial will get you started using RealView ICE (RVI) and RealView Trace (RVT) hardware with RealView Debugger (RVD) software. The versions used are:

- RVI host software v3.1
- RVI firmware v3.1
- RVD v3.1

It contains sections covering the essentials of installing and using:

- RealView ICE
- RealView Trace

Pre-requisites

The example code used in this tutorial will work with the memory map of an ARM926EJ-S Platform Baseboard (PB926EJ-S), but is designed in such a way that it can be ported to the memory map of your system with minimal effort. This platform has 128MB of RAM at address 0x0 once the Boot Monitor has run, but the examples should work with any target with at least 8KB of programmable memory at this location.

Additional information

This tutorial does not provide detailed documentation of RVD, RVT and RVI. Full documentation is provided with the products.



References in the subsequent sections of the tutorial are identified by these boxes.

Further help can be accessed by pressing F1 from within RVD, or from the help menu.

Full documentation is available in PDF format. This can be found by going to *Start* \rightarrow *Programs* \rightarrow *ARM* \rightarrow *RealView Development Suite* v3.1 \rightarrow *RVDS* v3.1 *Documentation Suite*.

When working through this tutorial, of particular interest are:

- RealView Debugger v3.1 User Guide
- RealView ICE and RealView Trace v3.1 User Guide
- RealView Debugger v3.1 Trace User Guide
- RealView Compiler Tools v3.1 Linker and Utilities Guide

Other useful information can be found on the www.arm.com web site:

- Application Note 168: Tracing with RVD (http://www.arm.com/documentation/Application_Notes/index.html)
 Contains a more advanced tutorial for RVD Trace
- FAQs (http://www.arm.com/support/devfaqsindex.html)

• Answer common RVD, RVI and trace questions

Section 1: RVDS and RVI Installation

This section describes how to install the required software on your host PC, and configure your RVI and RVT.

A	
<u> </u>	

RedHat Enterprise Linux 4 is the only Linux platform supported by RVDS 3.1. References to *Linux* in this tutorial apply to RHE4.

Install the RealView Development Suite (RVDS) software

Follow the instructions on the RealView Development Suite (RVDS) v3.1 CD for installation. It is recommended that you install to the default location. If you are installing on RedHat Enterprise Linux 4, you should use the setuplinux.bin installer. On Linux you will need to edit your .bashrc file to include the line source /<path>/RVDS31env.posh where <path> is your installation path.

🟐 ARM RealView Software Wizard	
Select Destination Please provide the destination	Realview [.]
Please select the destination to which you would like this product to be installed	
© Default destination	
C:\Program Files\ARM	
C Custom destination	
C:\Program Files\ARM	<u></u>
	Browse
Cancel	Back

After installing RVDS, RVD can be launched by:

- On Windows, navigating to *Start* → *Programs* → *ARM* → *RealView Development Suite* v3.1 → RealView Debugger v3.1
- On Linux, by typing **rvdebug** at a command shell prompt (note that this requires .bashrc to be updated as detailed above)

RVD configuration files are stored at:

- On Windows, C:\Documents and Settings\<username>\Application Data\ARM\rvdebug\3.1\
- On Linux, /home/<username>/.rvdebug/3.1/



You can revert to the default configuration by starting RVD with the **--cleanstart** switch. At a command prompt, type **rvdebug --cleanstart** and press enter.

You can revert to the default configuration by starting RVD with the **--cleanstart** switch (from a command line, type **rvdebug --cleanstart** and press enter).

Install the RealView ICE host software

Follow the instructions on the RealView ICE v3.1 CD for installation. Make sure that you install the software in the folder you are using for all of your ARM RealView tools. If you are installing on RedHat Enterprise Linux 4, you should use the setuplinux.bin installer.

Connecting to your RealView ICE unit

You can connect to your RealView ICE unit using one of 3 methods:

- via a USB cable (Windows only)
- via a Local Area Network (LAN) with or without DHCP
- via an Ethernet cross-over cable

By default, RVI is preconfigured to work correctly via USB or a DHCP-enabled network without additional configuration.

This document will briefly explain how to set up a connection via a USB cable.

If you are using Linux, you should connect to your RVI via Ethernet.

To set up an ethernet connection on a DHCP-enabled network:

- Follow step 1 below

- Connect the RVI to your network using the supplied Ethernet cable - Follow on from step 5 below, noting that your RVI will appear under *TCP/IP*

For a more detailed explanation, or for instructions for connecting via Ethernet, refer to Chapters 2 and 3 ('*Getting Started*' and '*Configuring RealView ICE Networking*') of the *RealView ICE and RealView Trace* v3.1 User Guide.

Connection via USB

1. Plug the power supply for the RealView ICE into the unit.

A

A

ARM

- 2. Connect the RealView ICE to your computer's USB port using the supplied cable.
- 3. Windows should automatically start the Found New Hardware Wizard. When prompted, browse to the driver stored in the C:\Program Files\ARM\RVI\ Drivers\usb_driver\1.2\6\win_32-pentium folder.

	Found New Hardware Wizard		Found New Hardware Wizard
Found New Hardware Wizard Please choose your search and installation options. Image: Search for the best driver in these locations. Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed. Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) Image: Search removable greated (inppy, CD-RDM) <th>Welcome t Hardware Windows Up Read our privacy Can Windows co column Can Windows co column Can Windows co Can Windows</th> <th>to the Found New Wizard Inch for current and updated software by omputer, on the hardware installation CD, or on date Web aite (with your permission). policy nnect to Windows Update to search for me only and givery time I connect a device <u>is time</u></th> <th>This wizard helps you install software for: Read/iew-ICE If your hardware came with an installation CD of floppy disk, insert it now. What do you want the wizard to do? Install the software automatically (Recommended) Install from a list or specific location (Advanced) Click Next to continue.</th>	Welcome t Hardware Windows Up Read our privacy Can Windows co column Can Windows co column Can Windows co Can Windows	to the Found New Wizard Inch for current and updated software by omputer, on the hardware installation CD, or on date Web aite (with your permission). policy nnect to Windows Update to search for me only and givery time I connect a device <u>is time</u>	This wizard helps you install software for: Read/iew-ICE If your hardware came with an installation CD of floppy disk, insert it now. What do you want the wizard to do? Install the software automatically (Recommended) Install from a list or specific location (Advanced) Click Next to continue.
Found New Hardware Wizard Please choose your search and installation options. Image: Search for the best driver in these locations. Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed. Image: Search removable media The best driver found will be installed. Image: Search removable media The best driver found will be installed. Image: Search removable media The best driver found will be installed. Image: Search removable media The best driver found will be installed. Image: Search removable media The search: Image: Cryptogram Files/ARM/RVI/Drivers/usb_drivers/1.2(Image: Search removable media The search: Image: Default search.		< Back Next > Cancel	< Back Next > Cancel
 Search for the best driver in these locations. Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed. Search removable media (floppy, CD-ROM) Include this location in the search: C:\Program Files\ARIMXPV\Drivers\usb_driver\1.2\v Browse Don't search. I will choose the driver to install. Choose this option to select the device driver form a list. Windows does not guarantee that the driver you choose will be the best match for your hardware. 		Found New Hardware Wizard Please choose your search and installa	tion options.
		 C Search for the best driver in these local Use the check boxes below to finit or erpaths and removable media. The best d Search removable media. The best d C Include this location in the search (C:\Program Files\ARM\RVI\Driv C Don't search. I will choose the driver to Choose this option to select the device the driver you choose will be the best m 	tions. xpand the default search, which includes local tiver found will be installed. .CD-ROM) h: rersYusb_driver\1.2\ Browse install. driver from a list. Windows does not guarantee that atch for your hardware.

- 4. Restart your computer and the RealView ICE unit.
- 5. Confirm that your RVI can be detected using the RVI Config IP utility. This utility can be found at *Start* \rightarrow *Programs* \rightarrow *ARM* \rightarrow *RealView ICE* v3.1 \rightarrow *RealView ICE Config IP*.



In RVI Config IP, select $RVI \rightarrow Start Scan$ to scan for ICEs.

崎 RVI (Config IP							
File Vie	w RVI Help							
	🗲 🔳 🌮							
Access	Ethernet Address	Ethernet Type	DHCF	Host Name	I.P. Address	Default Gateway	Subnet Mask	Active Connection
USB								
\sim	00:02:F7:00:0B:FF	Auto-Detect	Yes	RVICE5	127.0.0.2	127.0.0.2	255.0.0.0	
TCP/IP								
-	00:02:F7:00:25:12	Auto-Detect	Yes	OtherICE1	10.1.72.34	10.1.72.1	255.255.255.0	0
-	00:02:F7:00:19:AC	Auto-Detect	Yes	OtherICE2	10.1.72.41	10.1.72.1	255.255.255.0	0
-	00:02:F7:00:05:C5	Auto-Detect	Yes	OtherICE3	10.1.72.190	10.1.72.1	255.255.255.0	0
L	00:02:F7:00:04:D0	Auto-Detect	Yes	OtherICE4	10.1.72.39	10.1.72.1	255.255.255.0	0
1								
								111

Your RVI should be listed under *USB*. There may be other ICEs on your network listed under *TCP/IP*.



Right-click on your ICE and select Identify.

This should cause the LEDs on the front of your RVI unit to flash.

If you want to name your RVI, you can do this by right-clicking on the RVI, selecting *Configure*, and entering a *Host Name*. Click *Configure* to confirm.

Install the RealView ICE firmware

You should update your RVI to the latest firmware, which can be found in your RVI installation folder.



Open the RVI Update utility. This utility can be found at *Start* \rightarrow *Programs* \rightarrow *ARM* \rightarrow *RealView ICE* v3.1 \rightarrow *RealView ICE Update*.



Click on your ICE and select Connect.

# RVI Update - <unconnected></unconnected>					
File View RVI Help					
2 2					
Connect to a RealView ICE to continue	RealView ICE	E browser			Scanning 🕢
	Access	Host Name	IP Address	Ethernet Address	
	Found5 Ė~USB				
		RVICE5	127.0.0.2	00:02:F7:00:0B:FF	
		OtherICE1 OtherICE2 OtherICE3 OtherICE4	10.1.72.34 10.1.72.41 10.1.72.190 10.1.72.39	00:02:F7:00:25:12 00:02:F7:00:19:AC 00:02:F7:00:05:C5 00:02:F7:00:04:D0	
	Other TCP/ IP Address	/IP Devices / Host Name	nnect	ldentiț	·



Click on the ICE (*Install Firmware*) button with a green arrow in the top left of the window.

💐 RVI Update - RVICE5	
File View RVI Help Install Fi	rmware
RealView ICE	Description Real/View ICE Updatable Software Version 3.1.1 build 763 Comments N/A Disconnect
	//



Browse to the C:\Program Files\ARM\RVI\Firmware\3.1\23 folder and select ARM-RVI-3.1.0-754-base.rvi. Click Open. Click Continue in the next dialog.



The firmware update will take around 2-3 minutes to complete.





As before, click on the ICE button with a green arrow, and browse to the .rvi file that you just extracted. Select *Open*, and then *Continue* in the following dialog.

Patching will again take around 2-3 minutes to complete. After completion, your RVI is now updated to the latest available RVI firmware revision.

Setting up RealView Trace

The RealView Trace unit should be securely mounted on the RVI unit. Note that the Trace unit requires no additional software in order to work.



Refer to section 6.4.2 (*'Connection Instructions'*) of the *RealView ICE* and *RealView Trace v3.1 User Guide*. for more information on mounting the RVT unit onto the RVI unit.

Connecting the RealView ICE and RealView Trace to your target hardware

Connect one end of the provided JTAG cable to the RealView ICE unit. Connect the other end of the cable to the socket marked *JTAG* on the target board.



Connect one end of the provided Trace cable to the small Trace 'T piece' adapter.

Plug the adapter into the MICTOR connector marked *TRACE* on your target. Connect the other end of the cable to your RealView Trace (RVT).



The RealView Trace unit does not need additional power; it can obtain power directly from the RealView ICE.

ARM



It is recommended that you use the LVDS JTAG probe in preference to the standard JTAG cable, as the LVDS probe:

- Lets you debug systems with a faster JTAG clock (as long as the target permits it). For TCK speeds of 20MHz or more you need to use the LVDS probe.
- Helps to avoid some issues related to weak JTAG signals, or JTAG signals with interference.
- Has a longer cable, enabling debugging when the ICE is further away from the target.



On some boards (including the PB926EJ-S), the JTAG and MICTOR connectors are too close together to plug both the LVDS JTAG and Trace probes into the board. In

ARM



this case, you can plug the LVDS JTAG connector into the side of the Trace connector, as shown below.

Probe

Section 2: Preparing for the Examples

This section prepares the debugger and the examples for the remainder of the tutorial.

2.1 – Building an Image (rebuilding the examples for your target)

The examples in this tutorial make use of 3 pieces of code:

- A Hello World example that outputs text, making use of RVD's STDIO tab
- A reset example, containing a vector table and some initialisation code, to demonstrate running a program at reset time
- A version of the classic Dhrystone benchmark modified to run indefinitely, to demonstrate trace capture

The code examples in this tutorial are provided with batch files that call the C compiler (*armcc*) and linker (*armlink*) to build the images. You will need to invoke the .bat batch files in each example folder in order to build an .axf image that can be loaded to your target. These examples will work without modification on the Versatile PB926EJ-S platform.



c:\rvds31_tutorial\PB926

These are the files for use with the PB926EJ-S board

The examples require 8KB of memory at address 0x8000. If you are working with a PB926EJ-S, or another target board that meets this requirement, then you should skip to the next section.

If you do not have 8KB of memory at address 0x8000, then you should follow the remainder of this section, which gives more information on rebuilding the examples to work with your target.

The address that *armlink* will link the image to execute from is specified by a scatter file. The scatter files used for the examples all have the file extension *.scat*.

Scatterfiles describe where code and data are stored at load time and at run time. In the example below, the scatter file is also used to locate the stack and heap.

```
LOAD 0x8000 \leftarrow Modify this value

{

RAM +0

{

*(+RO,+RW,+ZI)

}
```

ARM_LIB_STACKHEAP +0x1000 ALIGN 32 EMPTY 0x1000

{ }

}

This example scatter file creates one load region at 0x8000. Within this load region is an execution region called **RAM** at address +0, indicating that the address is at an offset of 0 from the load region address (i.e. 0x8000). This execution region contains all the code and data for the image.

A second execution region is called **ARM_LIB_STACKHEAP**, and is marked as $+0 \times 1000$ **ALIGN 32** so that it will be placed on the next 32 byte boundary that is $>= 0 \times 1000$ bytes from the end of the **RAM** region. It is 0×1000 bytes in size and is marked as **EMPTY** because it holds no code or data sections.

ARM_LIB_STACKHEAP is a key region name. In RVDS 3.0 and later, this execution region name tells the linker where you want to place the stack and heap, causing the linker to automatically link in all of the necessary code to set up the stack and heap accordingly.

In this example the heap will grow upwards from the beginning of the region **ARM_LIB_STACKHEAP** and the stack will grow downwards from the end of the region **ARM_LIB_STACKHEAP**. The absolute addresses that the heap and stack grow from will depend on the size of the **RAM** region.

In order to port this example to work with a target other than the PB926EJ-S, you would simply need to adjust the **LOAD** address (0x8000) to be an address in RAM on your target, allowing enough space above the chosen address to fit in the image including the stack/heap region (8KB is recommended).

0

Refer to Chapter 5 of the RVCT 3.1 Linker and Utilities Guide ('*Using Scatter-loading Description Files*') for more information on using scatterfiles.

After making changes to the scatterfile or to the tools' command lines, you must rebuild the image to implement these changes. This can be done by invoking the .bat build script again.

2.2 – Connecting to and Configuring your Target

Start RealView Debugger by going to Start \rightarrow Programs \rightarrow ARM \rightarrow RealView Development Suite v3.1 \rightarrow RealView Debugger v3.1

Select Target \rightarrow Connect to Target...



د..... المطلق

Click on the *Add* button to the right of the *RealView-ICE* entry in the *Connect to Target* window.

🌺 rvdebug.brd - Connect to Target				
<u>File View Connection H</u> elp				
📙 🙊 🔹 🗯 🐂 🐂 🚘 Grouped By 🛛 Target	•			
Name	Configuration	State		
			Add	
RealView ICE			Add	
L <not configured=""></not>	RealView-ICE	Not Configured	<u> </u>	
ABM Ltd. Direct Connection			Add	
Connection Modes				
Connect :				
Disconnect :				
				//

The *RVConfig* window appears. RVD should automatically detect your RealView ICE unit. If it does not, click on the green icon at the top-right hand corner of the *RVConfig* window to begin scanning.

C-RVConfig - C:\Documents and Setting	gs\PSG\Applical	tion Data\ARM\rve	debug\3.1\RVI_0.rvc				
File View Help							
ⁱ RealView ICE (Not Connected)	RealView ICE b	RealView ICE browser Scanning					
	Access	Host Name	IP Address	Ethernet Address			
	Found1 È- TCP/IP	RVICE5	10.1.73.40	00:02:F7:00:07:43			
	Other TCP/IP IP Address / I	Devices Host Name Connect	įde	ntify			

Select your RealView ICE unit from the list and click Connect.

Additional options will appear:

-

-

RVConfig - C:\Documents and Setting	s\PSG\Application Data\ARM\rvdebug\3.1\RVI_0.rvc *	
<u>File V</u> iew <u>H</u> elp		
È-ReaView ICE: (TCP/IP RVICE5) ├-Devices -Advanced	ICE TDO	
	Auto Configure Scan Chain JTAG Clock Speed 10.000 MHz 🗾 Add Device	ve Device
	Vise Adaptive Clock if detected Move Left Move	e <u>R</u> ight
	Device Properties Config	uration
	1	1

Auto Configure Scan Chain causes the RVI to scan for devices in the target's scan chain. Each detected device is added to the tree diagram.



Your target should appear in the list:

- C:\Documents and Setting	s\PSG\Application Data\ARM\rvdebug\3.1\RVI_0.rvc *	
<u>File ⊻iew H</u> elp		
Exit Fixed	ARM926EJ-S IB Length = '4' ICE TDO Auto Configure Scan Chain JTAG Clock Speed Adaptive Y Add Device Remove	• Device
	Move Lett Move	
	Device Properties Configu	ation
Saves the file (Ctrl+S)		1

ARM

If you are using a target that is not recognized by *Auto Configure*, you should select *Add Device*... and navigate to the ARM core on which your target is based. If you are manually configuring your target in this way, you will need to fully populate the scan chain with *Custom UNKNOWN* (in *Add Device*...) entries if there are items other than the core on the scan chain (e.g. DSPs, FPGAs etc).



Select your target underneath the *RealView ICE* entry in the *Connection Control* window.

😪 rvdebug.brd - Connect to Target				
<u>F</u> ile ⊻iew <u>C</u> onnection <u>H</u> elp				
🗏 🍄 🦉 🗃 🖻 🌽 Grouped By 🛛 Target	-			
Name	Configuration	State		
⊕ - RealView Instruction Set Simulator (RVISS) ⊕ - Instruction Set System Model (ISSM) ⊖ - RealView ICE	l		Add Add Add	
	ReaNiew-ICE	Not Configured		
ARM926EJ-S_0	RVI	Disconnected		
Connection Modes				
Connect : Use Default 💌				
Disconnect :				
Double-click this Target or press return to con	nect to it.			li

Select *Connection* \rightarrow *Connect* from the menu to connect to your target.

Double-clicking on the connection name (**ARM926EJ-S_0**) will also cause RVD to connect to your target.

A

🥵 rydebug.brd - Connect to Target				
<u>File View Connection H</u> elp				
🛛 🛫 😳 🕮 🐂 💕 Grouped By 🗍 Target	-			
Name	Configuration	State		
	Post for ICE	Not Configured	Add Add Add	
ARM926EJ-S_0	RVI	Connected		
Connection Modes				
Connect : Use Default 💌				
Disconnect : As Is Without Debug 💌				
Double-click this Target or press return to disc	connect from it.			

2.3 – Setting up RealView Debugger

Line Numbers

Line numbers are used through this tutorial to identify specific source lines.



If you do not already have source line numbering enabled, select *Edit* \rightarrow *Advanced* \rightarrow *Show Line Numbers* to display the source file line numbers in the code window.

Advanced 🕨 🕨		Shift Lines Left	Alt+Shift+F8
		Shift Lines Right	Alt+F8
		Shift Width	
	•	Show Line Numbers	
		Show Original Line Numbe	ers

Workspaces

Workspaces are used to store personalized settings – for example the layout of individual windows within the main RVD window. You can dock/undock, resize and move these windows by dragging them. There are 'hot areas' to the left, right and bottom of the main RVD window. Dropping a window in one of these areas will cause the window to be docked.



- Drag the *Registers* window to the right hand edge of the screen so that it becomes docked.

<pre># W THE UND OF THE THE PLANE TH</pre>	M926EJ-5_0@RYI - RealView Debugger					_ 0
come to BealView Hokupper v3.1 d image ent Consections: image:setsions: ima	(at yew Target Debug Iools Help - →	Find	▼ Line	Scripts (None)	■ Sis Sis Sis	
Come of Paraview Hendinger Vill Hanger His Comments His Comments		1.00000		2		
d hage: HARDS E23-9_0RVF1 (connected) HARDS E23-9_0RVF1 (come to Realview Debugger VJ.1				Type Value	
ex (Source 1) HARD 267-5 (SHW1 (connect end)	ad Image				Image <none></none>	
Image: Second Action Image: Second Action <td< td=""><td>sat Connections: ATM25EL-S_O∦KVI (connected)</td><td></td><td></td><td></td><td></td><td></td></td<>	sat Connections: ATM25EL-S_O∦KVI (connected)					
by Description by Descr					Process Merray Max	
<pre></pre>					R0 0x47296337 R1	0x00007331
<pre></pre>					R2 0x1000000 R3	0x07C346BB
<pre>86 Descently 86 Descently</pre>					R4 0x101F1000 R5	0x0000000
<pre> a</pre>					R6 0x0000007 R7	0x07C0CB40
<pre>so taxes field bor control to be address field bor of the set in the set</pre>					R8 Ox07C4F5C8 R9	0x07C0CB40
so Connecto so Co					R10 0x00000040 R11	0x000000&
see Searces Job Lead 1000-2007 Searce Job Lead				· · · · · ·	R12 OxFFFCF331 SP	0x07EF90E8
An Cananda An Cananda An Cananda Conception of the set of the s					LR Dx07C0CC8C PC	0x07C09FD4
nge Commenty me Georgeorge Activities Contractions Guardrooccels - Challoon Locations Guardrooccels - Challoon Locations Guardrooccel					USR USR	
See a state Database Da	we Disassembly]		Core CP15 Cache Operations Cach	ne Lockdown TCM Regions TLB 0
Discrete Productions Discrete Productions	re Value		≚ Start addres	is Co	olumns Data sizes	Format
Digordocceb- dinknown Location>	0x07C09FD4> <unknown location=""></unknown>		No Address		Auto column 💽 4 bytes	Hexadecimal
Skak Skak	Dx07C0CCBC> <unknown location=""></unknown>			+0 +4	+8 +C +10 +14	
Address			No Adds	ress		
Seat View Trace FPGA Rev: 3 : PaalView Trace FPGA Rev: 3 : PaalV			No Adda	ress		
Seak J Weahl Weak2 Waak3			No Add	ress		
Ball/Lee Maddrees 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 31: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 32: Ball/Lee Trace FP6A Rev: 3 Watch Watch Watch 33: Ball/Lee Trace FP6A Rev: 3 Watch			No Add	ress		
Stati Weehi Weehi Modeline Of Paulities Trace FFGL Rev: 1 Modeline Modeline Of Security Trace FFGL Rev: 1 Trace FFGL Rev: 1 Trace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1 Strace FFGL Rev: 1 Strace FFGL Rev: 1 Trace FFGL Rev: 1<			No Add	ress		
0: Peallylew Trace FPGL Perr: 3 D: Peallylew Trace FPGL Perr: 3 pright LMB Linited 2001-2007 packed to stopped derice ped no Stop derected no isaget ped	Stack Watch1 Watch2 Watch3 Watch4		No Addres	#		
0; Pavlivie Trace FLD Ber; i yrgh: ME line 1000-1007 ached to stopped device ing: No stackberg to tog_dementy defined - setting tog_of_memory to 0x00020000 ped no NortOropits (dankores) pet no NortOropits (dankores) pet no NortOropits (dankores) pet no NortOropits (dankores)	0: RealView Trace FPG& Rev: 3					
yright 28% Linited 2002-2007 ached to stoppded derice ning: No stack/heap or top_dipempt defined - setting top_of_memory to 0x00020000 sped on Stop decided on target ppd at Conformation of the setting top_of_memory to 0x00020000 ppd at Conformation of the setting top_of_memory to 0x00020000 ppd at Conformation of the setting top_of_memory to 0x00020000 ppd at Conformation of the setting top_of_memory to 0x00020000	DO: MENTATEA IENCE NPA MEA: J					
pped at 0.007009794 : «Unknown» > 5400 Farfred Log	yright ARR Limited 2002-2007 sched to stopped davice ring: No stack/heap or top of memory defined - setting top_of_memory to 0x00020000 pped on Stop detected on target					
1921 1 300 FRFra Los	anned at 0x02009ED4: (Inknown)					
1 StdD FRFrid Loo						
				1		

- Select *Target* \rightarrow *Disconnect* to disconnect from the target.
- Select *File* \rightarrow *Workspace* \rightarrow *Save Workspace* to save the new layout.
 - Select File \rightarrow Workspace \rightarrow Save Settings on Exit to remove the tick

next to this option.



RVD will now remember the window layout that you have just created, and will start up without trying to connect to your target.



See the FAQ '*How can I use workspaces to control RVD's GUI?*' on the ARM web site for more information on using Workspaces.

Semihosting

A

Semihosting is a mechanism that captures I/O requests made by code running on the target (typically library code), and communicates these to the host system for handling. For example, application *printfs* will by default appear within the debugger console window.

See Section 13.9 of the *RealView Debugger v3.1 User Guide* ('*Viewing semihosting controls for RealView ICE JTAG* connections') for more information on setting Semihosting options.

Vector catch is a mechanism that is used for catching exceptions that occur on the core. It is implemented using dedicated logic, or instruction breakpoints if the core that you are using does not implement this logic. This feature is particularly useful when debugging code for which you have not yet written exception handlers.

ARM

Semihosting and vector catch are controlled by RVD connection properties, and are enabled by default.

- Select Target \rightarrow Connection Properties from the RVDmenu.
- Browse to Advanced_information\Default\ARM_Config\
- Ensure that Vector catch is set to True
- In the Semihosting folder, ensure that Enabled is set to True
- Select *File* \rightarrow *Save and Close* to save any changes.



Changing settings in *Connection Properties* causes those settings to be applied to all subsequent connections of that type.

Changes made in *Connection Properties* do not affect any currently active connections. You must disconnect from your target before making changes and then reconnect afterwards.

Alternatively, you can enable/disable Semihosting in the *Debug* tab of the *Registers* window. Settings made in this window are temporary and are lost when you disconnect from the target.

A

Reg	jisters	×
	User Output6 and Coax	Low 🔽 🔺
	Allow ICE to perform	TRUE
	Allow ICE to latch S	TRUE
	Semihosting	
<	Semihosting Enabled	TRUE
	Top of Memory	0x00020000
	ARM SVC	0x00123456
	Thumb SVC	OxAB
	Vector	0x0000008
	Window	0x0015
		-
┫		
IS	Cache Lockdown TCM Regions 1	LB Operations Debug
	ARM926EJ-S_0@RVI_3	

-

Section 3: Using RealView ICE with RVD

This section provides an introduction to using the RealView ICE unit with RealView Debugger to debug an application running on your target hardware.

3.1 – Simple Hello World Project (RVD Basics)



Select Local File t	:o Load:				<u>?</u> ×
Look jn:	Hello_World		•	🗢 🗈 💣 🎹	•
My Recent Documents Desktop My Documents	hello.axf				
My Network Places	File name: Files of type:	hello.axf Absolute Files (*.axf;*.ou	ıt;*.elf;*.sym;	▼ *.a,*.eld)	<u>O</u> pen Cancel <u>H</u> elp
Symbols Unly	IV Replace E>	iisting File(s)			
Sections:			IM Auto-S	Det PL	
Arguments:			M Set Pl	L to Entry point	
Offset:					1

The main window now shows that the image is loaded. The red box indicates the current execution position.

Code Window Tabs

When RealView Debugger first loads an image, the main view contains a *Disassembly* tab that displays disassembled code with interleaved C/C++ source lines. The current location of the PC is shown (right click \rightarrow *Interleave Source* to toggle source interleaving).

If source code exists for the current location of the PC, you can toggle between disassembly and source code views using the toolbar button below.



By default, RealView Debugger will show the scope relevant to the current context of the PC.



Execution begins. The *Output* pane at the bottom of the RVD window shows the *StdIO* tab which allows console I/O operations for the current image. The program prints some text to the output window and ends.



Select *Target* \rightarrow *Reload Image to Target* from the menu.

RVD will reload the image ready for debugging. Again the current execution position is shown at main().

Registers Window

The RVD *Registers* window allows you to view and modify the contents of ARM registers.

Reg	gisters				X
	RO	0x0000018	R1	0x00020026	•
	R2	0x00000000	R3	0x00000000	
	R4	0x0000000	R5	0x00000000	
	R6	0x0000000	R7	0x0000A06F	
	R8	OxO7C3D3FO	R9	Ox07COCA8C	
	R10	0x0000A080	R11	0x0000A080	
	R12	0x00000000	SP	0x0000B2A0	
	LR	0x000091B8	PC	0x00008000	
	CPSR	nzcvqIFjtSVC			
Ð	USR				•
Co	ire CP1	15 Cache Operations	Cache L	.ockdown 🔤 TCM Regio 🕢	۲
	ARM926E	J-S_0@RealView-ICE			

Of most interest are the tabs for the *Core* registers (r0 - r15 and the CPSR), the *CP15* registers (e.g. enabling/disabling caches, mmu, branch prediction etc.), and the *Debug* registers (e.g. setting Top of Memory and enabling/disabling Semihosting).

The displays of registers r0-r15 can be used to modify the value of live variables (variables that are currently in use) when the core is stopped, for example when you are stepping through code or hit a breakpoint.

You can switch between different value formats by right-clicking on a register and selecting *Format* from the context menu. The most common choices are *Hex* and *Decimal*.

Right-click on *R0* and select *Format* \rightarrow *Decimal* to change the format of *R0* to decimal. Select the value next to *R0* and enter a new value (e.g. **526**).

Some of the registers are enumerated – for example the CP15 Control register.



In the Registers window *CP15* tab, double click on *Control*. Notice that by clicking on m/M you can enable/disable the MMU. Hover the mouse over each button to see what can be enabled/disabled.

CP15 Control	×
It rr v i r s b c a m	
lt_rr_v_ir_s_b_c_a_m	
0x00050078	
OK Cancel Help	

Another useful register that is enumerated is the *CPSR* (Current Program Status Register) in the *Core* tab. This register manages enabling/disabling of interrupts, core state (ARM or Thumb) and the current system mode.

PSR I	×
Condition N z c v q	
I F	
State	
SVC V	
Current value	1
NzcvqIFjtSVC	
0x800000D3	
OK Cancel Help	

Breakpoints

Breakpoints tell the debugger to stop the target when a particular event occurs. For example, you can tell the debugger to stop execution on the target before a particular instruction or C source code line is executed so that you can inspect and modify the current state of the core. This is implemented either by temporarily replacing the relevant instruction with a breakpoint instruction (a software breakpoint), or using watchpoint units within the core that monitor the address and data buses (a hardware breakpoint).

Software breakpoints can only be used on instructions that are in RAM, as they involve the temporary substitution of a breakpoint instruction for the original instruction. Hardware breakpoints can be used on any area of memory, including flash or ROM.

0

See the FAQ '*What is the difference between HW and SW breakpoints*?' on the ARM web site for more information on hardware and software breakpoints.

Hardware and software breakpoints can be conditional - the debugger will only stop the core under certain conditions. For example, a breakpoint could be set so that the debugger stops the core before a particular line of code is executed. A condition could be added to this breakpoint such that the breakpoint is only hit when a particular value is stored at a particular location in memory.

The simplest way to set a breakpoint is to double click in the grey margin to the left of a line in the *Disassembly* tab or within a source file. This will set an instruction breakpoint so that when the core is about to execute that assembly instruction or C source line the debugger will halt execution. RVD will use a software breakpoint where possible, otherwise a hardware breakpoint.



Set a breakpoint on **main()** by double clicking in the margin to the left hand side of the *Code* window. From the RVD main menu select $View \rightarrow Break/Tracepoints$ to view information about the breakpoints you have created.



Select $Debug \rightarrow Run$ from the menu (F5).

Execution now halts at **main()** after initialisation of the C library.

0

If the location where you want to set a breakpoint is in non-volatile memory (e.g. flash), you can explicitly set a hardware breakpoint.

Set a new hardware breakpoint on **subroutine()** by selecting $Debug \rightarrow Breakpoints \rightarrow Create Breakpoint... from the main menu.$

- Select Hardware Intsruction
- Type **subroutine** for the location and click *OK*.

Software Instruction	Location:	subroutine	
Hardware Instruction	Value Match:	▼	
Hardware Read Hardware Write	Class:	Standard Breakpoint	_
Hardware Access	Force Bre	akpoint Size (bits) 16	

Sciect Debug / Kun nom the menu (15)

Execution will halt at **subroutine()**.

Tracepoints

Tracepoints are similar to breakpoints but instead of halting program execution will start/stop trace capture. Note that you need to have the Trace Analyzer connected to be able to create a tracepoint. Using the Trace Analyzer and the setting of tracepoints are discussed in more detail in the *Trace* section of this tutorial.

Symbols Window

The RVD *Symbols* window allows you to browse through symbols contained in the images currently loaded into the debugger.

Symbols	×
Filter K**	📕 🔲 🖸 🕞 🕞 🕞
Image 🛆 @Hello	
Images Modules Functions Variables	
ARM926EJ-5_0@RealView-ICE	

From the *Symbols* window you can conveniently locate a function, run to a function, add a variable to the watch window or set a breakpoint.



If the *Symbols* window is not already open, select $View \rightarrow Symbols$ from the RVD menu to display the *Symbols* window.

The format of the *Filter* at the top of the window is *Image\Module\Function*. You can manually edit the filter, or double click on an item in the *Images* or *Modules* tab to filter by that selection. Note that * is a wildcard to avoid filtering.

- The *Images* tab displays currently loaded images;
- The *Modules* tab displays modules contained in the currently filtered images;
- The *Functions* tab displays functions contained in the currently filtered modules;
- The *Variables* tab displays variables contained in the currently filtered functions.

Double clicking on a function name jumps to that function in the Disassembly tab.



Symbols				×
Filter XXX			<u>G</u> oto Va	riables .
Function Name \square	Address	Scope	Module	ln 🔺
malloc	0x000090A0	Public	HEAP1	@
setbuf	0x000099A4	Public	STDIO	@
setvbuf	0x00009900	Public	STDIO	@
strlen	0x00009D74	Public	STRING	@
subroutine	0x00008080	Public	HELLO	@-
•				•
Images Modules F	unctions Vari	ables		
ARM926EJ-S_0@RealV	iew-ICE			

Double clicking on a variable name displays the address and value of that variable in the *Cmd* tab. By default, the *Variables* tab does not display locals.

- Select the Variables tab in the Symbols window

- Right-click in the white space in the window and select Show Locals

Symbols				×
Filter XXX			🗖 Goto '	Variables
Variable Name \square	Address	Scope	Module	Image
main/greeting	0x00000000	Local	HELLO	@Hel
helloworldstr	0x0000A094	Public	<global></global>	@Hel
subroutine/message	0x01C91BB0	Local	HELLO	@Hel
MyInt	0x0000A080	Public	<global></global>	@Hel
•				▶
Images Modules	Functions Va	riables		
ARM926EJ-S_0@Rea	alView-ICE			



Double-click on **MyInt** to display its address and value in the *Cmd* tab.



See the FAQ '*How do I access the symbols in my image using RVD?*' on the ARM web site for more information on the *Symbols* window and its tabs.

A

Memory Window

The *Memory* window allows you to view the data in a particular area of memory, in a configurable layout and format.

- Find the address of **MyInt** from the *Address* column of the *Symbols* window *Variables* tab (0xA080 in the screenshots above in the *Symbols Window* section)



- Enter the address into the *Start address* field in the *Memory* window and press *enter*.

Memory							×
Start address	Colu	umns	Data	sizes		Format	
0x0000A080	💌 Au	ito column	💌 4 by	tes	•	Hexadecimal	-
	+0	+4	+8	+C			
0x0000A080	0x00000024	0x0000A094	Ox0000A0D8	0x0000A11C	\$	• • • • • •	 •••
0x0000A090	0x00000000	0x00000000	0x0000000	0x00000000			
0x0000A0A0	0x00000201	0x00000000	0x0000001	0x00000000			
0x0000A0B0	0x00000040	0x00000000	0x00000000	0x00000000	0		
0x0000A0C0	0x00000000	0x00000000	0x00000000	0x00000000			 •••
0x0000A080							
ARM926EJ-5_0	@RealView-ICE						

- Change the Format to Decimal

Memory						×
Start address	Colu	mns	Data	sizes	Format	
0x0000A080	🗾 Aul	o column	💌 🖌 4 by	tes	Decimal	_
	+0	+4	+8	+C		<u> </u>
0x0000A080	36	41108	41176	41244	1 \$	• • • •
0x0000A090	0	0	0	0)	
0x0000A0A0	513	0	1	0)	
0x0000A0B0	64	0	0	0) @	
0x0000A0C0	0	0	0	0)	••••
0x0000A080						
ARM926EJ-S_0	@RealView-ICE					

د..... اللکھیں - Step forward (press F11) until the yellow arrow points at line 15, (printf("%s from main\n", helloworldstr);).





Change the *Format* to *Hexadecimal* and *Data sizes* to *1 byte*.
Notice that the text that makes up the string helloworldstr is displayed on the right-hand side of the *Memory* window.

Memory																		×
Start address	Start address Columns							sizes				Form	at					
0x0000A094			Auto col	lumn		I byte			Hexadecimal				-					
	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9								
0x0000A094	0x48	0x65	0x6C	0x6C	Ox6F	0x20	0x57	Ox6F	0x72	0x6C	H e	1.	ιo	U	Γo	r l	L	
0x0000A09E	0x64	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	d.		• •		•	• •		
0x0000A0A8	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x01	0x02			• •		•	• •		
0x0000A0B2	0x00	0x00	0x00	0x00	0x00	0x00	OxOD	0x00	0x00	0x00			• •		•	• •		
OXOOOOAOBC	0x00	0x00	0x00	0x00	0x40	0x00	0x00	0x00	0x00	0x00	• •	·	0	• •	·	• •		-
0x0000A094																		
ARM926EJ-5_00	@RealVie	ew-ICE																

You can add new tabs to the *Memory* window by right-clicking on an existing tab and selecting *Duplicate View*. Tabs can be closed by selecting *Delete View*.





Select $Debug \rightarrow Set PC$ to Entry Point from the main menu, then press F5 to run to the breakpoint set earlier on main().

Watch Window

The Watch window allows you to keep track of specific variables.



If the *Watch* window is not already open, select $View \rightarrow Watch$ from the RVD menu to display the *Watch* window.

Watch				×				
Name		Value						
<u>ًا ا</u>				•				
Watch1	Watch2	Watch3	Watch4	J				
ARM926EJ-5_0@RealView-ICE								

Right-click on MyInt (on line 8 of Hello.c) in the main view and select *Add Watch*. Repeat for greeting and helloworldstr (lines 13 and 9 respectively).



Press F11 (Step Into) 3 times so that the yellow arrow points to line
15 (printf("%s from main\n", helloworldstr);).
Observe that as greeting and helloworldstr are initialised in the code, their values update in the *Watch* window.

Watch 🗵								
Name	Value 🔺							
MyInt	0x0000024							
greeting	Ox000080CC "Hello from subroutineD"							
helloworldstr	[0x0000A094] "Hello World"							
•								
Watch1 Watch2 Watch3 Watch4								
ARM926EJ-S_0@RealVie	W-ICE							

As well as adding a variable to the *Watch* window from the variable's context menu, you can manually enter a variable name into the *Watch* window *Name* column. Note that the core must be stopped for the values displayed to be updated.

ARM

When watching a pointer (including a string/array), a + symbol will be displayed to the left of the pointer name. Click on this to expand the display to show the contents of an array, or, for a pointer, to show the value pointed to by that pointer.



Click on the + next to the pointer **greeting**. Observe that the value pointed to by **greeting** is shown (the first character of the string, *H*).



Click on the + next to the array name helloworldstr. Observe that the characters that make up the array are shown. The text string ends with the first *NULL* character (0x0) at helloworldstr[11].

Watch	
Name	Value
MyInt	0x0000024
greeting	0x000080CC "Hello from subroutineD"
🔁 helloworldstr	[OxOOOOA094] "Hello World"
_ [0]	0x48 'H'
- [1]	Ox65 'e'
_ [2]	0x6C '1'
_ [3]	0x6C '1'
_ [4]	Ox6F 'o'
_ [5]	0x20 ' '
_ [6]	Ox57 '₩'
_ [7]	Ox6F 'o'
_ [8]	0x72 'r'
_ [9]	0x6C '1'
_ [10]	0x64 'd'
_ [11]	0x00 '\x00'
_ [12]	0x00 '\x00'
_ [13]	0x00 '\x00'
L [14]	0x00 '\x00'
•	
Watch1 Watch2 V	Vatch3 Watch4
ARM926EJ-S_0@RealV	iew-ICE

The *Watch* window contains 4 tabs so that you can have a different set of variables that you want to keep track of depending on which part of your image you are currently working with. This is particularly useful when moving up / down the Call Stack (see *Call Stack* below).

Call Stack

The execution scope or context determines the visibility of variables and functions. A variable or function is referred to as in scope if the name can be accessed at the current point of execution. The scope of a variable or function can be:

- the current source file, for global variables and functions;
- the current function, for local variables

When you load an image, scope is initially set to the value of the PC, which is usually the entry point of the image. As you step through the image and move into child functions, the scope updates to continue to show the current context.

\$	- Select <i>View</i> \rightarrow <i>Call Stack</i> from the RVD main menu	
	Call Stack	×
	Name	
	void subroutine(const char *) Line #25 Col 2	
	int main(void) Line #17 Col 2	
	unknownrt_entry(void)	
	Call Stack	
	ARM926EJ-S_0@RVI_1	

The Call Stack window shows the hierarchical flow of a program and enables you to trace back to the program's status at an earlier point. By moving up to a previous entry in the call stack, you can change the scope to be at the point where the child function will return to. This works by retrieving from the stack local variables that were active immediately before the function call took place.

Double-click in the margin to the left of the **subroutine()** function definition (line **23**) to set a breakpoint. Press F5 to run to this point in the program.

```
19 return 0;
20 }
21 
22 
23 void subroutine(const char *message)
24 {
25 printf(message);
26 }
```

Double-click on the int main (void) Line #17 Col 2 entry in the Call Stack window.
Observe the output Scoped at level 1: (0x000080AC): HELLO\main Line 17:2 in the Cmd tab.

A blue arrow and a blue box show the new scope:

~	10	
~	11	int main(void)
•	12	(
	13	<pre>const char *greeting = "Hello from subroutine\n";</pre>
	14	<pre>strcpy(helloworldstr, "Hello World");</pre>
	15	<pre>printf("%s from main\n", helloworldstr);</pre>
	16	<pre>subroutine(greeting);</pre>
-	17	<pre>printf("And Goodbye from main\n");</pre>
	18	<pre>printf("MyInt is %d\n\n", MyInt);</pre>
	19	return 0;
	20	}

0

A blue arrow and blue box indicates that the current scope is different from the location of the PC. A yellow arrow and red box indicates that the current scope is the location of the PC.

You can also move up (to parent functions) and down (to child functions) the call stack by entering the commands *up* and *down* in the RVD Command Line (see *Command Line* below).



Variables in the *Watch* window that are local to a particular function also become active / inactive as you move up / down the stack. You can make use of multiple *Watch* windows, or the multiple tabs within an individual *Watch* window to isolate variables that are relevant to a particular scope.

If the scope is at a location that corresponds to a source file then RVD automatically opens that source file if the *Home Page* tab or another source file currently has the focus.

Command Line

The *Cmd* tab in RVD displays the current status of the debugger. When you carry out a GUI action, you will usually see a command-line equivalent echoed to this tab. You can use these textual equivalents on the command-line (the grey bar at the bottom of the *Cmd* tab). This is particularly useful when creating a script (see below).

×	> step
	> go
	Stopped at 0x00008080 due to SW Instruction Breakpoint
	Stopped at 0x00008080: HELLO\subroutine Line 25:2
	> up
	Scoped at level 1: (OxOOOO80AC): HELLO\main Line 17:2
	Stop>
	4
	Cmd StdIO FileFind *Log

لیسی
#####

Use the *up* and *down* commands to move up and down the call stack (see *Call Stack* above).

Simple Scripting

You can gather together a sequence of command line instructions into a script. This is a plain text file that can be called automatically when you connect to a target, or on demand. An example of automatically calling a script at connect time is given in the *Configuring Trace* section of this tutorial.

To create and run a simple script:

- Carry out the GUI actions that you want to script
- After each GUI action, copy and paste the command echoed to the *Cmd* tab into a text file
- Save the text file (e.g. *MyScript.inc*)
- Click on the *Add Script* toolbar button and browse to the script that you created
- Click on the *Run Script* toolbar button to run the script



Scripts are most commonly used to perform target configuration on connection to a target, or to perform the connection itself followed by some subsequent steps. The example below shows the generation of a simple connection script.

- Disconnect from your target (*Target* \rightarrow *Disconnect*)
- Right-click in the *Cmd* tab and select *Clear* to clear the existing output.
- Connect to your target by double-clicking on the target in the Target
- \rightarrow *Connect to Target...* dialog

- Load the previous **Hello.axf** image via the *Target* \rightarrow *Load Image*... dialog.

ARM



See the FAQ '*RVD Scripting & Automation*' on the ARM web site for more information on scripting with RVD, including some example scripts.

Section 4: Using RealView Trace with RVD

This section provides an introduction to using the RealView Trace unit with RealView Debugger to perform trace capture.



Application Note 168 'Tracing with RVD' provides a more comprehensive walkthrough guide to tracing your target.

4.1 – Configuring Trace

When you have auto-configured (or manually configured) your target in RVD, you can configure whether or not your target has an ETM (Embedded Trace Macrocell) or an ETB (Embedded Trace Buffer). The PB926EJ-S contains an ETM, but no ETB.



Ensure that *ETM* is selected, and *ETB* is deselected. Click *OK*.

Device Properties	<u>? ×</u>
Device Name : ARM926EJ-S	
Template Version : 1:0:0	-
Options :	
Embedded Trace Macrocell (ETM) Embedded Trace Buffer (ETB)	
QK <u>C</u> ancel <u>H</u> el	p

If your target contains an ETB (for example, the CM1136JF-S), then you can also select *ETB* to use the on-chip buffer rather than the external RealView Trace unit.

Connect to your target, then open the RVD Analysis window.

Select V	′iew →Analysis Window	
Fil III	ARM926EJ-S_0@RVI - RealView Debugge e Edit View Target Debug Tools Help e Edit View Target Debug Tools Help Image: Constraint of the second se	r
File Edit View Find Filter	alysis Sort Trace Data Profiling Data Help	
<no analyzer="" connec<="" td=""><td>sted and No File Loaded></td><td></td></no>	sted and No File Loaded>	
Trace Source Profile		Not connected

From the Analysis window you can configure trace settings and view collected trace data.



Connect the analyzer to your target by selecting $Edit \rightarrow Connect / Disconnect Analyzer$

Select *Edit* \rightarrow *Automatic Tracing Mode* \rightarrow *Instructions and Data* from the menu in the *Analysis* window.



The debugger is now configured to automatically capture trace for both instructions & data.

Select *Edit* \rightarrow *Data Tracing Mode* \rightarrow *Data Only* from the menu in the *Analysis* window.





The debugger is now configured to capture both data values and addresses for data trace capture.

Trace settings can be configured via the *Configure ETM* dialog, by selecting *Edit* \rightarrow *Configure Analyzer Properties...* from the Analysis window menu. For this tutorial the default settings do not need to be changed.

A	RM9	26EJ-9	5_0@	RVI - A	Analys	is											. 🗆	×
File	Edit	View	Find	Filter	Sort	Trace	e Data	Profiling	Data	Help 📘								
] 🖻	Ē	Сору					C	trl+C										
Elen	\$₩	Conne	ct/Dis	connect	: Analy:	zer							Addres:	s		Opco	de	
<n)< td=""><td>₽↓</td><td>Tracing</td><td>g Enat</td><td>oled</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></n)<>	₽ ↓	Tracing	g Enat	oled														
		Config	ure Ar	halyzer	Proper	ties												
		Set Tra	ace Bu	lffer Siz	e													
																		_
				/ Co	onfigur	'e ETM	М							×				
				Archit	ecture:	1.3	Protoc	ol: 0										
				Trac	e data '	width	Trac	e port mo	de —		Trac	e buft	fer packi	ing -				
					ont ont		ю N С M	ormal uttiplexed			O No	utomai ormal r	tic nacking					
				16	bit		ÕD	e-multiple>	ed		O Do	ouble p	packing					
				C 24	bit		Пн	alf-rate clo	ocking e	enabled	C Q	uad pa	acking					
				C 32	bit		🗖 Di	isable trac	eport									
				FIFO	overflo	ow pro	otectio	n ———	Tr	ace copr	oc regi	ster tr	ansfer -					
				O St	all proc	essor			Ö	All								
				C Da	ita supp	pressi	on		C	Only wh	en traci	ing da	ta					
				FIFO I	nighwa	ter [_									
				- Exte	nded e;	derna	l input	selection										
				Input :	1		Input 2	2	Input	3	Ini	put 4						
					Memor	y map	decod	le 0x00	<pre>c c c Address Opcode Address Opcode Address Opcode Addr</pre>									
	AktP26EJ-5_00eRVI - Analysis File Edit View Find Filter Sort Trace Data Profiling Data Help Connect/Disconnect Analyzer Address Opcode Configure Analyzer Properties Set Trace Buffer Size Set Trace Buffer Size Set Trace data width Trace port mode Trace buffer packing Architecture: 1.3 Protocol: 0 Trace data width Trace port mode Automatic Automatic Abit Normal Soft Trace data width Trace port mode Trace buffer packing Automatic Automatic Soft Soft Trace data width Trace port mode Trace buffer packing Automatic Automatic Soft Trace data width Trace port mode Trace buffer packing Automatic Normal packing Double packing Ouble packing Ouble packing Ouble packing Soft Double packing Ouble packing Soft Double packing Ouble packing Ouble packing Ouble packing Ouble packing Soft Double packing Ouble packing																	
					Enable	timest	tampin	а ,								Opcode		
					Cycle a	accura	ate trac	sing										
				Г	Data or	nly tra	ce (Do	not trace	instruc	tions)								
				Г	Suppre	ss da	ta on F	FIFO full										
				-ETM	Pairing													
				Pair E	TM with	n No	Pairing	3		-								
				🗖 Ma	aster E1	IM												
						ок		C	ancel	1	H	Help	1					

0

See Section 4.3 of the *RealView Debugger v3.1 Trace User Guide* ('*Configuring the ETM parameters*') for more information on ETM configuration options.

A reduced trace buffer size will reduce the time taken for the debugger to retrieve the trace data from the trace unit, but will limit the amount of data that can be captured. If you are performing lots of single steps or will be stopping your target regularly, you may want to set a small buffer size – perhaps as small as 1024 records. If you need to capture trace data for the entire execution of a larger piece of code, you may want to use the maximum buffer size. For this example we will set the buffer size to 65535 records. The maximum available buffer size for your trace hardware will be set automatically when you first connect to your RealView Trace unit (at least 1 million records).

Select *Edit* \rightarrow *Set Trace Buffer Size...* from the menu, and enter a buffer size of 65535 records. Click *Set*.

- A	RM9	26EJ-9	5_0@	RVI - A	inalys	is								_ 🗆	×
File	Edit	View	Find	Filter	Sort	Trace D	ata	Profiling	Data	Help					
] 🖻	6	Сору					Ctr	l+C							
Eler	₩.	Conne	ct/Disc	:onnect	Analy	zer						Address	0 Op	code	
< N	· 🚮	Tracing	g Enab	led											
		Config	ure An	alyzer	Proper	ties									
		Set Tra	ace Bul	ffer Siz	e										
		Shara (Contro	l Elour	Thomas	se Oslu									
ī		_	1-1												
	on o	incer v	raiue											즤	
		Q	Enter	r buffer	size t	o set:									
	65	5535													
											9	iet	 Cancel		

Trace analyzer connection and configuration can be carried out using a connection script that can be associated with a connection. When you perform the instructions above, notice that a command appears in the RVD *Cmd* tab for each step. You can copy these instructions into a text editor and save the resulting script as a text file.

×	inc> analyzer, connect // connect the Trace analyzer	
	Initialising Trace Support	
	Connecting Analyzer to ARM926EJ-S_0 ETM Architecture: 1.3.1	
	ARM Internal Use Only	
	Software supplied by: ARM Ltd	
	inc> analyzer,auto_both // inst & data auto tracing mode	
	inc> analyzer,dataonly // data only tracing for data	
	inc> etm config,packauto // select auto trace buffer packing	
	inc> analyzer,set_size=65535 // set trace buffer size to 65535 records	
	Stop>	
	<u>4</u>	
	Cmd StdIO FileFind *Log	

Open Notepad and paste in the following commands: analyzer, connect // connect the Trace analyzer



If you are using a target other than the PB926EJ-S, you may need to modify the above commands.

You can tell RVD to run the script that you have created whenever you connect to a target via RVI.





If you now disconnect from and reconnect to your target, you should find that the analyzer is automatically connected and configured for you.

When you no longer want to run this script at connect time you will need to remove the command you just created from *Connection Properties*. If you want to run the script a single time, you can type **inc** `**full_path_to_script**>' on the RVD command line.

NB: If you are working with a Versatile PB926EJ-S or AB926EJ-S development board, you will need to reduce the core clock speed from 210MHz (default) to 140MHz in order to perform trace capture in both normal and half-rate tracing modes. This can be done by running the following RVD script commands before you begin your trace, either manually from the RVD command line, or by adding them to the beginning of your trace configuration script (**TraceConfig.inc**):



// unlock system registers
setmem /32 0x10000020 =0x0000A05F
// modify SYS_OSC0
setmem /32 0x1000000C =0x00002C6C
// lock system registers
setmem /32 0x10000020 =0x00000000

Your system is now configured to the point where you can perform *Auto Trace* using the ETM without setting any tracepoints. Trace capture will begin immediately when your program begins execution and will continue until the target is stopped (e.g. at a breakpoint). In the following examples tracepoints are used to specify a specific region where tracing should be carried out, reducing the total amount of trace data captured.

-

4.2 – Performing Simple Trace Capture

Select $Target \rightarrow Load Image...$ from the RVD main menu. Browse to **Dhry_Inf.axf** in the **c:\rvds31_tutorial\PB926\Infinite_Dhrystone** directory.

Select Local File	to Load:				<u>?</u> ×
Look jn:	Contraction Infinite_Dhrys	tone	•	🗢 🗈 💣 匪] .
My Recent Documents Desktop My Documents	Dhry_Inf.axf				
S	File <u>n</u> ame:	Dhry_Inf.axf		•	<u>Open</u>
My Network Places	Files of type:	Absolute Files (*.axf;	".out;".elf;".sym;	*.a;*.eld) 🗾	
					<u>H</u> elp
🔲 Symbols Only	🔽 Replace Ex	isting File(s)	PC		
Sections:			Auto-9	Set PC	
Arguments:			🔽 Set P	C to Entry point	
Offset:			-		
					11.

In the *Process* tab of the *Process Control* window (*View* \rightarrow *Process Control*), expand *Sources* and double click on **dhry_1.c** to open that source file.

5

	X
Туре	Value 🔺
🛷 ARM926EJ-S	PC=0x0000B59C
占 📕 Image	Dhry_Inf.axf
– 🗹 Load	Image+Symbols
_ Sources	From Image
-8	armsys.c
-8	bigflt.c
-8	boardlib.s
-8	btod.s
-8	classify.c
-8	d2f.s
-8	dcheck.s
	dcheck1.s
	ddiv.s
	dflt.s
│	dhry.h
	dhry 1.c
	dhry_2.c
	division.s
	dmul.s
	n inth
Process Memory	Map

In this example we will run an *Automatic Trace*. This uses no trace points for specifying the trace range and so causes trace to be captured for the entire program execution. It is possible to specify *Instruction Only* or *Instruction and Data* trace (see previous section). If tracing data in addition to instructions, you may need to consider the size of your trace buffer and the bandwidth of your trace port.





5

Ensure that *Data Value in Decimal* is selected in the *Trace Data* menu, and *Code Window Tracking* is selected in the *View* menu.

Т	ra	ce Data Profiling Data Help
][•	~	Position
ŧ,	~	Absolute Time
e		Relative Time
"	~	Access Type
ш Г ,	~	Address as Symbol/Line
e,	~	Address as Value
ш		Data Value in Hex

Data Value in Decimal



-ARM9	926EJ- S_ 0@R¥I_1 - <i>i</i>	Analysis				_ 🗆	×
<u>File</u> <u>E</u> dit	t <u>V</u> iew Fi <u>n</u> d Filter S	ort <u>T</u> race D	ata Profiling Data Help 📒				
] 🖻 日	🛯 🖻 📴 🗱 銔	4 0 ►					
Elem	Time/cycl	Туре	Symbolic	Address	Data/Dec	Opcode	
0	Warning: Tr	ace paus	ie				
0	3	Exec	Reset_Handler	0x0000B59C		OxEAFFF297	
1	4	Exec	main	0x00008000		0xEB000000	
2	5	Exec	scatterload_rt2	0x00008008		OxE28F002C	
3	7	Exec	scatterload_rt2	0x0000800C		0xE8900C00	
3		R Data	scatterload_null+0x20	0x0000803C	18048		
3		R Data			18064		
4	9	Exec	scatterload_rt2	0x00008010		OxEO8AA000	
5	12	Exec	scatterload_rt2	0x00008014		OxEO8BB000	
6	13	Exec	scatterload_rt2	0x00008018		OxE24A7001	
7	14	Exec	scatterload_null	0x0000801C		OxE15A000B	
8	15	Exec	scatterload_null	0x00008020		0x1A000000	
9	23	Exec	scatterload_null	0x00008028		OxE8BA000F	
9		R Data		0x0000C6BC	50940		
9		R Data			50940		
9		R Data			10492		
9		R Data			32836		
10	25	Exec	scatterload_null	0x0000802C		OxE24FE018	-
4							Þ
Trace	Source Profile						
					Tracing ena	bled	

The columns can be interpreted as follows:

Elem	The element number in the current trace buffer. If a trace trigger has
	been set then element 0 will appear at the trigger point. Otherwise
	element 0 will appear as the last element.
Time/cycl	The relative cycle number on which an element began execution.
Туре	Exec : An instruction that was executed
	NoExec: A conditional instruction that was not executed

	R Data: A data read
	W Data: A data write
Symbolic	Gives the module name and line number for the corresponding source
	code.
Address	Indicates the address an instruction was fetched from, or the address
	data was read from or written to.

Scroll through the captured trace data to see the source code and disassembly synchronized with each line of trace data.

The same example can be run with a defined trace range where tracing will take place, by placing markers, or tracepoints, in the code. The advantage of using tracepoints is that you can capture trace data just for specific areas of interest in your code, and avoid having that data overwritten by unwanted data when the trace buffer wraps.



5

Right-click in the margin to the left of line **150** in **dhry1.c** (within the main **for ()** loop in the body of **main ()**). Select *Insert Tracepoint*...to display the *New Tracepoint* dialog.



Select *Start of Trace Range (Instruction and Data)* from the list and click *OK* to set the tracepoint.

PNew Tracepoint		×
Select tracepoint to set:		
Set Trigger		
Trace Start Point		
Trace End Point		
Start of Trace Range (Instruction Only)		
Start of Trace Range (Instruction and Data)		
Start of Excluded Trace Range (Instruction and Data)		
Start of Excluded Trace Range (Data Only)		
Set ExternalOut1 Point		
	ОК	Cancel



Right-click in the margin to the left of line 189 (Proc_2 (&Int_1_Loc);). Select *Insert Tracepoint*...to display the Tracepoint *List Selection* dialog again.



Select *End of Trace Range (Instruction and Data)* and click *OK* to set the trace stop point.

PNew Tracepoint	1	×
Select tracepoint to set:		
Set Trigger		,
Trace Start Point		
Trace End Point		
End of Trace Range (Instruction and Data)		
Set ExternalOut1 Point		
	OK Cancel	

The source code display should now show the recently set trace and break points:

ARM

	148	<pre>for (Run_Index = 1; Run_Index <= 10000; ++Run_Index)</pre>
Start.	149	(
of	150	
_ (3) 151	Proc_5();
Trace V	152	Proc_4();
Range	153	/* Ch_1_Glob == 'A', Ch_2_Glob == 'B', Bool_Glob == true */
	154	<pre>Int_1_Loc = 2;</pre>
	155	Int_2_Loc = 3;
	156	strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 2'ND STRING");
	157	Enum_Loc = Ident_2;
	158	Bool_Glob = ! Func_2 (Str_1_Loc, Str_2_Loc);
	159	/* Bool_Glob == 1 */
	160	<pre>while (Int_1_Loc < Int_2_Loc) /* loop body executed once */</pre>
	161	(
	162	Int_3_Loc = 5 * Int_1_Loc - Int_2_Loc;
	163	/* Int_3_Loc == 7 */
	164	<pre>Proc_7 (Int_1_Loc, Int_2_Loc, ∬_3_Loc);</pre>
	165	/* Int_3_Loc == 7 */
	166	Int_1_Loc += 1;
	167) /* while */
	168	/* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
	169	<pre>Proc_8 (Arr_1_Glob, Arr_2_Glob, Int_1_Loc, Int_3_Loc);</pre>
	170	/* Int_Glob == 5 */
	171	<pre>Proc_1 (Ptr_Glob);</pre>
	172	<pre>for (Ch_Index = 'A'; Ch_Index <= Ch_2_Glob; ++Ch_Index)</pre>
	173	<pre>/* loop body executed twice */</pre>
	174	(
	175	<pre>if (Enum_Loc == Func_1 (Ch_Index, 'C'))</pre>
	176	/* then, not executed */
	177	(
	178	<pre>Proc_6 (Ident_1, &Enum_Loc);</pre>
	179	<pre>strcpy (Str_2_Loc, "DHRYSTONE PROGRAM, 3'RD STRING");</pre>
	180	<pre>Int_2_Loc = Run_Index;</pre>
	181	<pre>Int_Glob = Run_Index;</pre>
	182	}
	183)
	184	/* Int_1_Loc == 3, Int_2_Loc == 3, Int_3_Loc == 7 */
	185	<pre>Int_2_Loc = Int_2_Loc * Int_1_Loc;</pre>
	186	<pre>Int_1_Loc = Int_2_Loc / Int_3_Loc;</pre>
End	187	Int_2_Loc = 7 * (Int_2_Loc - Int_3_Loc) - Int_1_Loc;
of 🔪	1 88	/* Int_1_Loc == 1, Int_2_Loc == 13, Int_3_Loc == 7 */
π_{race} (2) 189	<pre>Proc_2 (∬_1_Loc);</pre>
	190	/* Int_1_Loc == 5 */
Range	191	// make loop infinite
(•) 192	<pre>if (Run_Index >= 10000)</pre>
Brack	193	Run_Index = 1;
Dreak	194	} /* loop "for Run_Index" */
Point		

5

5

Select *Debug* \rightarrow *Set PC to Entry Point* from the main menu, then press F5 to run to the breakpoint set earlier on line **192**.

Execution halts on the breakpoint. Select $View \rightarrow Analysis$ Window from the RVD menu to view the captured trace.

⊻iew	Target	<u>D</u> ebug	Tool					
i I	<u>N</u> ew Code Window							
-	<u>A</u> nalysis Window							
	Suctors W	lindowe						

F ARM92	ARM926EJ-S_0@RYI - Analysis											
Eile Edit View Find Filter Sort Irace Data Profiling Data Help												
Elem	Time/cycl	Type	Symbolic	Address	Data/Dec	Opcode						
-556	-556 Warning: Trace pause											
-530	-530	Exec	main\#158	0x00008290		0xE2700001						
-518	-518	NoExec	main\#158	0x00008294		0x33A00000						
-506	-506	Exec	main\#159#160	0x00008298		0xE5870014						
-506	-506	W Data	Bool_Glob	0x0000C900	1							
-494	-494	Exec	main\#160	0x0000829C		0xE59D0050						
-494	-494	R Data		0x000102D8	2							
-488	-488	Exec	main\#160	0x000082A0		0xEA000052						
-486	-486	Exec	main\#160	0x000083F0		0xE1500005						
-463	-463	Exec	main\#160	0x000083F4		0xBAFFFFF3						
-462	-462	Exec	main\#161#162	0x000083C8		0xE0800100						
-454	-454	Exec	main\#162	0x000083CC		0xE0400005						
-452	-452	Exec	main\#162	0x000083D0		0xE58D004C						
-452	-452	W Data		0x000102D4	7							
-450	-450	Exec	main\#163#164	0x000083D4		0xE59D0050						
-450	-450	R Data		0x000102D8	2							
-448	-448	Exec	main\#164	0x000083D8		0xE28D204C						
-446	-446	Exec	main\#164	0x000083DC		0xE1A01005						
-420	-420	Exec	main\#164	0x000083E0		0xEB000255						
Trace Source Profile												
						Tracing enabled						

Scroll through the captured trace data to see the source code and disassembly synchronized with each line of trace data.

0

-

For a more in-depth trace tutorial, refer to *Application Note 168* '*Tracing with RVD*' on the ARM web site.

ARM

Summary

In this tutorial, you have:

- Installed RVDS and the RVI software
- Connected your RVI and RVT to your computer and target board
- Updated your RVI's firmware
- Configured RVD

You have learnt:

- How to rebuild the example images
- Basic Scatterfile usage
- How to connect to your target
- How to debug a program using RVD
- How to use automatic trace
- How to define a trace range