Trace Debug Tools

Version 1.2

User Guide

ARM

ARM DUI 0118C

Trace Debug Tools
User Guide

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Change
April 2000	A	First release
May 2001	B	Updated for TDT 1.1
March 2002	C	Updated for TDT 1.2

Proprietary Notice

Web Address

`http://www.arm.com`

Contents

Trace Debug Tools User Guide

Preface

This preface introduces the User Guide for the *Trace Debug Tools* (TDT). It contains the following sections:

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

About this book

This book describes how to use the version 1.2 *Trace Debug Tools* (TDT) extension to the *ARM Debugger Suite* (ADS) version 1.2, especially the *ARM eXtended Debugger* (AXD). This book documents only TDT version 1.2. It does not describe the core AXD functionality or any other installed extensions.

The main part of this book describes TDT for the ARM ETM. TDT also supports the use of an XScale target. If you are using TDT with an XScale, some elements of the functionality and user interface are different. Chapter 7 *Using TDT with XScale* describes these differences.

Intended audience

This book is written for all users of the trace extensions to ADS. It assumes that you are familiar with both the general operation of Windows programs, and the operation and basic functionality of AXD (see *ARM publications* on page x).

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to TDT.

Chapter 2 *Getting Started*

Read this chapter to locate the set-up instructions for your trace hardware, and to learn how to configure AXD to recognize the hardware. This chapter also describes how to configure AXD to allow nonstop debugging.

Chapter 3 *Trace Facilities*

Read this chapter to learn how to gather, generate, and configure real-time trace information, and how to use the trace facilities, step-by-step, in a typical trace session.

Note

It is suggested that you read this chapter before you perform any tracing.

Chapter 4 *Configuring the ETM*

Read this chapter to learn how to configure the *Embedded Trace Macrocell* (ETM). Using the Configure ETM dialog box, you can define the overall configuration of the ETM hardware.

Chapter 5 ***Configuring Trace Capture***

Read this chapter to learn how to use the Configure Capture dialog box, the filter facility that allows you to set conditions for controlling the output of trace information. The Configure Capture facility is not available with XScale trace, and this dialog box is therefore not present if you are using TDT with XScale.

Chapter 6 ***The Trace Window Display***

Read this chapter for a complete description of the Trace window, where trace output is displayed.

Chapter 7 ***Using TDT with XScale***

Read this chapter for a description of the main ways in which XScale Trace differs from ETM Trace, including setting up, the trace procedure, and the trace display window.

Chapter 8 ***Examples***

This chapter contains a number of examples detailing the use of TDT in specific circumstances.

Appendix A ***Setting up the Trace Hardware***

This appendix contains details about setting up your trace hardware for each of the supported hardware configurations.

Appendix B ***Setting up the Trace Software***

This appendix contains details about setting up your trace software for each of the supported configurations.

Appendix C ***Using RealMonitor with TDT***

This appendix describes how to configure RealMonitor to work with a TDT system.

Appendix D ***ETM Versions***

This appendix describes how the version numbers of the ETM implementations are written and their relationship to the ETM protocol numbers. Refer to the ETM specifications for more information on the ETM hardware. See *ARM publications* on page x.

Typographical conventions

The following typographical conventions are used in this book:

bold	Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate.
`italic`	Highlights special terminology, denotes internal cross-references and citations.
`monospace`	Denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`monospace` (`mono` underlined)	Denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.

Further reading

This section lists publications from both ARM Limited and third parties that provide additional information on developing code for TDT.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

This book contains information that is specific to TDT. Refer to the following books for related information:

- *Embedded Trace Macrocell Specification* (ARM IHI 0014)
- *Multi-ICE User Guide* (ARM DUI 0048)
- *Multi-ICE Installation Guide* (ARM IHI 0009)
- *MultiTrace User Guide* (ARM DUI 0150)
- *RMHost User Guide* (ARM DUI 0137)
- *CodeWarrior IDE Guide* (ARM DUI 0065)
- *AXD and armsd Debuggers Guide* (ARM DUI 0066)

- *ARM Architecture Reference Manual* (ARM DDI 0100).

Other publications

Refer to the following publications for additional information about the Agilent analyzers described in this manual:

- E5903-97002, *Trace Port Analysis for ARM ETM User's Guide*, Agilent Technologies, 2000.
- E3459-97005, *Emulation for the ARM7/ARM9 User's Guide*, Agilent Technologies, 2000.
- E5904-97000, *Agilent Technologies E5904B Option 300 Trace Port Analyzer for ARM User's Guide*, Agilent Technologies, 2000.

To access these documents, see the website `http://www.agilent.com`.

Refer to the following publication for additional information about the Tektronix Trace controller software described in this manual:

- *Tektronix TLA Logic Analyzer ARM ETM Support Package Instructions*, Dragonfly Software LLC, 2000.

Refer to the following publications for additional information about the Tektronix analyzers described in this manual:

- The Tektronix web site, `http://www.tek.com`
- The Dragonfly Software web site, `http://www.dfsw.com`.

Refer to the following publication for additional information about the external architecture of the Intel® XScale™ core:

- I*ntel® XScale™ Core Developer's Manual*, Intel Corporation, 2000.

To access this document, see the website `http://developer.intel.com.`

Feedback

ARM Limited welcomes feedback on both TDT and its documentation.

Feedback on Trace Debug Tools

If you have any problems with TDT, contact your supplier. To help them provide a rapid and useful response, please give:

- details of the release you are using
- details of the host and target you are running on
- a small standalone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem
- the version string of the tool, including the version number and date.

Feedback on this book

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

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

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter introduces *Trace Debug Tools* (TDT), part of the trace extensions to the *ARM Developer Suite* (ADS). It contains the following sections:

1.1 About the Trace Debug Tools

TDT is an add-on product to ADS, extending the debugging capability with the addition of real-time program and data tracing. Installing TDT 1.2 upgrades the ADS 1.2 AXD to include the extra functionality required by TDT.

Note

- If you are using ADS version 1.0, 1.0.1, or 1.1, you must obtain and install ADS version 1.2 before installing TDT 1.2. Contact your supplier to obtain this upgrade.
- You cannot use TDT 1.2 with the *ARM Debugger for Windows* (ADW). This includes the version of ADW supplied with TDT 1.0.
- TDT does not work with the *ARM Software Development Toolkit* (SDT).

1.2 The Real-Time Trace requirement

This section describes the benefits real-time trace can offer in a debugging environment. In a traditional debugging environment, breakpointing and single-stepping enable you to run your application code to a given point, and then stop the processor. You can then examine or change memory or register contents, step, or restart the application. In real-time systems, however, the processor often cannot be stopped because if it is, a system crash occurs. The trace tools enable you to debug these systems.

The most difficult bugs to find are those that occur where there is an unforeseen, and therefore unpredictable, interaction between the application software and hardware. These bugs can be intermittent, and usually occur while the system is running at full clock speed. Starting, stopping, or stepping the processor does not always reveal the problem. A historical, non-intrusive trace of instruction flow and data accesses can provide the extra information you need to identify the bug.

For example, assume that an application crashes during an interrupt routine. As a result of the crash, a memory protection fault occurs, the cause of which cannot be found using breakpoints and single-stepping methods. Using TDT, you can set up the trace filter facility to collect trace data only during the interrupt routine, and use a trigger to stop tracing when the protection fault occurs. In this case:

- the filter facility limits the amount of information that needs to be traced and analyzed
- the trigger ensures that the trace information around the bug has been captured and not overwritten.

Because trace buffer depths are finite, these features are important to ensure the buffer is filled only with relevant information. They also save time by limiting the information that must be analyzed to find the bug. If you are using the ARM ETM, trigger and filter conditions can be changed using the TDT Configure capture dialog box to refine what trace data is captured, and when it is captured (see Chapter 5 *Configuring Trace Capture*).

Other examples of where tracing can be more helpful than using traditional debugging methods are:

- getting precise information about how an intermittent program crash occurs
- finding when, and why, a particular value is written to a particular location in memory.

The debugging functionality required to provide a solution to these problems can be summarized as:

- a trace of execution to debug random or erratic problems that can only be detected after the problem has occurred (known as historic trace)
- visibility of the code executed and data processed while the processor is running
- non-intrusive trace of the program flow.

This functionality is provided by TDT when used in the *RealTrace* environment.

1.3 The ARM trace solution

ARM provides a trace solution for its cores when they are embedded within an *Application-Specific Integrated Circuit* (ASIC). Figure 1-1 shows this overall system and the interaction between the components. See *Setting up the trace hardware* on page 2-2 for details on setting up your hardware.

The ARM trace solution also supports XScale targets. See *The ARM trace solution for XScale targets* on page 7-2 for information on the overall system when an XScale target is used.

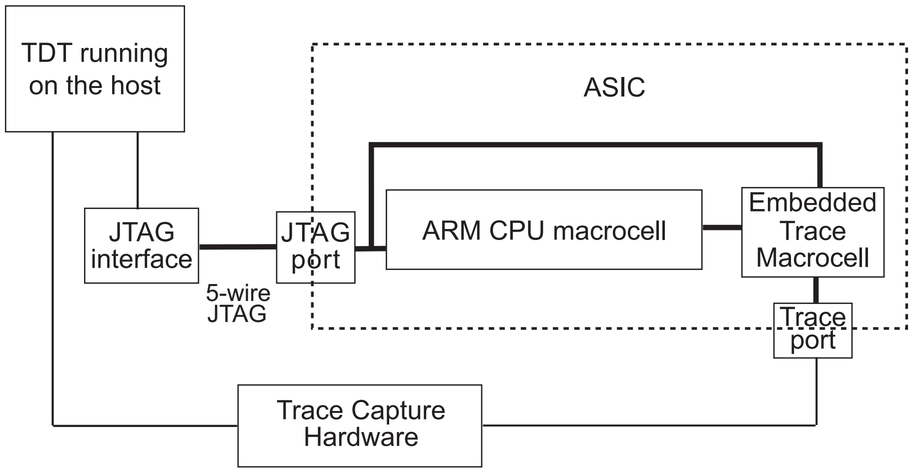


Figure 1-1 Real-Time Trace solution

The ARM trace solution comprises three elements that provide the capability to trace instructions and data accesses in real time:

Embedded Trace Macrocell

This monitors the ARM core buses, and passes compressed information through the trace port to the *Trace capture hardware*. To detect sequences of events performed by the processor, the on-chip *Embedded Trace Macrocell* (ETM) contains a configurable number of resources that are selected when the ASIC is designed. These resources comprise the trigger and filter logic you utilize when programming with TDT. See *The Embedded Trace Macrocell (ETM)* on page 1-6 for more details.

Trace capture hardware

This is an external device that stores the information from the trace port. See *The Trace capture hardware* on page 1-8 for more details.

Trace Debug Tools

This sets up the trigger and filter logic, retrieves the data from the analyzer, and reconstructs a historical view of processor activity. See *The Trace Debug Tools (TDT)* on page 1-8 for more details.

Other hardware components shown in Figure 1-1 on page 1-5 comprise:

ARM CPU macrocell

An ARM7-family or ARM9-family CPU that contains EmbeddedICE logic. The CPU must be ETM-enabled.

JTAG interface

A protocol converter, such as Multi-ICE®, that converts low-level commands (such as configuring an address comparator) from AXD into JTAG messages for the EmbeddedICE logic and the ETM.

1.3.1 The Embedded Trace Macrocell (ETM)

To trace processor instruction flow, the ETM broadcasts branch addresses and status information through the trace port. Later in the tracing process, the complete instruction flow is reconstructed by the debug tools using a binary image of the code to fill in the sequential instructions that were executed.

Note

It is not possible to reconstruct self-modifying code.

You can also program the ETM to broadcast the address and data value of data reads and data writes. The ETM only broadcasts address bits that have changed from the time of the last broadcast. Full data tracing of applications, excluding data-intensive functions such as block copy, can be achieved through a 16-bit data port with a 45-byte, on-chip FIFO.

If you do not want to trace all information, you can use the ETM to control the specific information that is traced (see *Defining resources* on page 5-17). For example, you can place the following restrictions on the amount of data that is traced:

- only accesses inside (or outside) of selected data regions
- only data accesses by certain routines.

Note

The instruction is always broadcast with the data trace. The bandwidth required for instruction trace is low.

You can use the same resources to turn tracing on and off, and therefore monitor a particular piece of code over a longer elapsed time within the finite trace buffer of the TPA. Resources are also used to generate a complex trigger event around which the analyzer collects the trace data.

The designer of the ASIC configures the number and size of resources. There are three standard configurations defined, shown in Table 1-1. Each configuration provides a different trade-off between silicon area, pin count and tracing features.

Table 1-1 Standard ETM configurations

Resource type	Small	Medium	Large
Pairs of address comparators	1	4 (8)	8 (16)
Data comparators	2	2	8
ASIC memory map decoders	4	8	16
Counters	2	2	4
State machine present	No	Yes	Yes
External inputs	2	4	4
External outputs	0	1	4
`FIFOFULL` present	Yes	Yes	Yes
FIFO depth	9 (ETM7) 10 (ETM9)	18 (ETM7) 20 (ETM9)	45
Trace packet width	4/8	4/8/16	4/8/16

The maximum width of the ETM data port is configured by the device manufacturer. The following sizes are available:

4-bit A 4-bit ETM data port using 9 output signals on the device being traced.

8-bit An 8-bit ETM data port using 13 output signals on the device being traced.

16-bit A 16-bit ETM data port using 21 output signals on the device being traced.

The standard five JTAG interface pins are also required to set up the ETM.

TDT interrogates your particular implementation of the ETM, and only provides GUI access to the resources that have been implemented.

Note

TDT is not able to detect whether all the optional features of the ETM (such as `FIFOFULL`) have been connected within the device containing the ETM. To determine this, refer to your device documentation.

1.3.2 The Trace capture hardware

The trace capture hardware is an external device that stores the information from the trace port. The trace information is compressed so that the analyzer does not need to capture data at the same bandwidth as that of an analyzer monitoring the core buses directly.

1.3.3 The Trace Debug Tools (TDT)

TDT retrieves compressed trace data from the trace capture hardware and reconstructs a historical view of processor activity using a stored copy of the binary image that was loaded into the target. The Trace window displays a disassembly of code executed with full symbol information, interleaved with data accesses to memory (see Chapter 6 *The Trace Window Display*).

In addition to displaying the basic trace information, you can either interleave the captured trace with your source code (C, C++, or Assembly language), or you can synchronize a memory, source, or disassembly window to the Trace window cursor. This enables a rapid understanding of the trace data. See *Viewing trace output with source code* on page 3-13.

TDT provides a configuration wizard to enable you to set up the trigger and filter logic of the ETM in a manner that does not require a detailed understanding of the ETM logic (see *Using the wizard* on page 5-48). With the wizard, you can:

- Use trace filters to restrict the trace stream that is captured and sent to the trace port. This prevents the buffer being filled with uninteresting and unwanted trace information (see *Defining rules* on page 5-15 and *Defining resources* on page 5-17).
- Set a trigger point to instruct the TPA to stop the trace capture process (see *Trigger* on page 5-7).
- Collect trace information before, after, or about the trigger point (see *The Configure ETM dialog box* on page 4-2 and *Trigger* on page 5-7).
- Set up triggers while the target system is running, and examine trace output without halting the process. This is referred to throughout this guide as *nonstop debugging*.

Note

You can perform traditional stop and start debugging, which halts the processor when a breakpoint or watchpoint is hit, by using **Go** and **Stop** from the **Execute** menu.

1.4 Code and trace capture examples

Several examples and screenshots in this manual use the example program Dhrystone, provided with ADS as an ANSI C version called `dhryansi` and as a K&R C version called `dhry`. All the ADS examples are in the `.../examples` subdirectory. They are supplied as source only so must be built before use with TDT.

Examples of trace configuration files (with the extension `.tcf`) are supplied with TDT. The accompanying source code is provided in the `Data_abort` example project. When the software is installed, these files are contained in folders in the ADS `…\examples\Trace` subdirectory. The examples are:

Data_abort Demonstrates how you can use TDT to discover where a problem (a data abort in this case) occurs in some example code. Before using this example, you must build the accompanying `.mcp` file using CodeWarrior to obtain the necessary `.axf` image (see the *CodeWarrior IDE Guide*). You must load this image into AXD to use with the `.tcf` file.

In this example, the problem area of code results from the use of an invalid pointer. That is, an `sString` is set to `0xFFFFFFFF` on returning from the function `ReverseFind()`.

`Data_abort.tcf` contains a set of rules that enable you to trace the image after execution enters the `ReverseFind()` function. A trigger is set when execution enters the address `0x10` (the data abort vector). The trace information generated from this example shows the flow of execution up to, and including, the data abort.

Data_access Demonstrates how you can use TDT to gather information about data accesses (read/writes) that occur in a specific function. Prior to using this example, you must load the `dhry.axf` image (found in `…\examples\dhry`) into AXD.

In this example, trace information is collected for all data accesses within `Func_2()`. A counter and a trigger are used to capture information at the 50th run of this function.

Note

- After you load an image into AXD, you might need to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware. See the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*.
- Because a `.tcf` file overwrites your existing ETM configuration when loaded, you might need to change these settings to conform to your hardware before starting the example trace capture (see Chapter 4 *Configuring the ETM*).

For step-by-step instructions on performing trace captures, refer to *Tracing procedure* on page 3-4. For these examples, you do not have to set any conditions because the conditions are already contained in the `.tcf` files (see *Loading from a file* on page 5-41). After you load the relevant `.tcf` file, you must enable the rules by checking them in the Configure capture dialog box (see *Enabling, disabling, and deleting rules* on page 5-49).

Rule definition example on page 5-44 provides step-by-step instructions for creating a set of rules.

Note

No example code is provided for tracing an XScale target.

1.5 New features in TDT 1.2

TDT 1.2 contains the following enhancements to the previous release, TDT 1.1:

- TDT can now be used with an XScale target if you are using Multi-ICE as the JTAG. The XScale target has its own trace hardware, so you do not need separate trace capture hardware. Because the XScale target has its own trace buffer, many of the configuration options used with the ARM ETM are invalid, and some elements of the user interface are different when you are using an XScale target. See Chapter 7 *Using TDT with XScale* for XScale-specific information.
- TDT 1.2 includes support for ETM 7 rev 1a and ETM9 rev 2a.
- TDT 1.2 can detect when the processor enters Java state and suppress trace. See *Warnings and other messages* on page 6-10 for information on the error message that TDT displays when this happens.
- You can stop saving trace to a log file at any time while tracing. See *Trace-specific GUI* on page 6-6 for information on saving trace to a log file.
- A new Refresh option in the View ETM State dialog box allows you to refresh the counter and state machine values at any time. See *View ETM State* on page 3-8 for more information.
- Output from the trace port can be suppressed when two ETMs are in use.
- TDT 1.2 is managed by the FLEX*lm* license management system that controls the use of the *ARM Developer Suite* (ADS) applications software.

Chapter 2
Getting Started

This chapter describes how to set up your trace hardware, and how to configure AXD to recognize the existence of the trace hardware. It also describes how to configure AXD to allow nonstop debugging.

For information on how to set up your system for XScale trace, see *The tracing procedure with XScale* on page 7-4.

This chapter contains the following sections:

- *Setting up the trace hardware* on page 2-2
- *Setting up the trace software* on page 2-4
- *Configuring AXD for nonstop debugging* on page 2-5
- *Tracing multiprocessor systems* on page 2-7.

2.1 Setting up the trace hardware

This section describes the requirements and procedure for setting up the trace system for ARM ETM. See *Setting up the trace hardware for XScale* on page 7-3 for information on setting up the trace system for XScale.

You can:

- use an emulated environment (see *ARMulator ETM emulation* on page A-14)
- directly trace your target hardware, described below.

To trace your target hardware, you must set up a hardware trace system, assembling the following components:

- an *ARM target board*, containing an ARM7-family or ARM9-family processor linked to an ETM
- trace capture hardware that is capable of storing trace information generated by the ETM
- a JTAG interface that can set up triggering and tracing in the ETM
- a workstation that is able to run TDT.

When choosing the target hardware and setting up the trace capture system, you must take care not to exceed the timing specifications of the target ETM signals or of the trace capture hardware. Exceeding the timing specifications results in synchronization and data errors that render the trace useless. More information can be found in the chapter on designing the target in the *ARM Multi-Trace Version 1.0 User Guide*.

You must decide between the available trace capture systems that you can use to connect the ETM to the debugger. The following trace capture systems are described in this manual and are directly supported by TDT:

- ARM MultiTrace
- Agilent 16600 or 16700 logic analyzer
- Agilent Trace Port Analyzer
- Tektronix TLA 600 or TLA 700 logic analyzer.

The following JTAG interfaces are supported by TDT:

- ARM Multi-ICE
- Agilent Emulation Module
- Agilent Emulation Probe.

The following sections describe how to set up the trace capture hardware for use with TDT:

- *ARM MultiTrace and ARM Multi-ICE* on page A-2

- *Agilent 16600 or 16700 logic analyzer and Emulation module* on page A-5
- *Agilent 16600 or 16700 logic analyzer and Multi-ICE* on page A-9
- *Tektronix TLA 600 or TLA 700 logic analyzer and Multi-ICE* on page A-12.

See *The ARM trace solution* on page 1-5 for a description of how your hardware interacts to complete the real-time trace solution.

2.2 Setting up the trace software

After you have set up your hardware (see *Setting up the trace hardware* on page 2-2), you must configure the software components. This section describes the procedure for setting up the trace system for ARM ETM. See *Setting up the trace software for XScale* on page 7-3 for information on setting up the trace software for XScale.

1. Install ADS 1.2 on your workstation, as described in the ADS installation instructions.
2. Install TDT 1.2 on your workstation, as described in the TDT installation instructions.
3. The software setup instructions are in Appendix B *Setting up the Trace Software*. Refer to one of the following sections for a description of the setup procedure for your system:
   - *ARMulator ETM emulation* on page A-14
   - *ARM MultiTrace and ARM Multi-ICE* on page B-2
   - *Agilent 16600 or 16700 Logic Analyzer and Emulation Probe* on page B-10
   - *Agilent 16600 or 16700 Logic Analyzer and ARM Multi-ICE* on page B-17
   - *Agilent Trace Port Analyzer and Agilent Emulation Probe* on page B-22
   - *Tektronix TLA 600 or TLA700 and ARM Multi-ICE* on page B-28.

2.3 Configuring AXD for nonstop debugging

When used with trace *Remote Debug Interface* (RDI) drivers, the TDT 1.2 capable version of AXD can connect to, and debug, running targets without stopping the target (see *Setting up the trace software* on page 2-4). Other RDI drivers might not support this feature.

This feature affects debugger functionality regarding:

- *AXD startup*
- *AXD configuration* on page 2-6
- *AXD behavior during target program execution* on page 2-6.

Note

You cannot perform nonstop debugging with an XScale target.

2.3.1 AXD startup

If you have appropriate hardware support, RDI drivers conforming to the RDI 1.5.1 rt specification (such as `multi-ice.dll`) can connect to a target already running a program. If you select this type of driver, it is not necessary to stop the target program (if it is already running) when the debugger connects to the target. If the debugger is appropriately configured, it enters execution state immediately when it connects to a running target. To stop the program, you must either select **Stop** from the **Execute** menu, or click the **Stop** toolbar button.

To ensure that the target is not stopped when AXD connects to it, use the `-nohalt` command-line option when starting AXD:

```
AXD -nohalt
```

If you specify this option, and the currently selected RDI driver does not support connection to a running target program, you are prompted to select a new RDI driver before the debugger attempts to connect to the target.

As an alternative to selecting the `-nohalt` option, you can use the **Configure Interface** dialog box (see *AXD configuration* on page 2-6) to save a session configuration that does not stop the target when AXD starts up.

You can start up AXD with a complete description of the AXD, TDT and RDI driver state. This is called a session. You save an AXD session to a file using the AXD **File → Save Session...** menu item. Use the following command line to start up AXD from a session file:

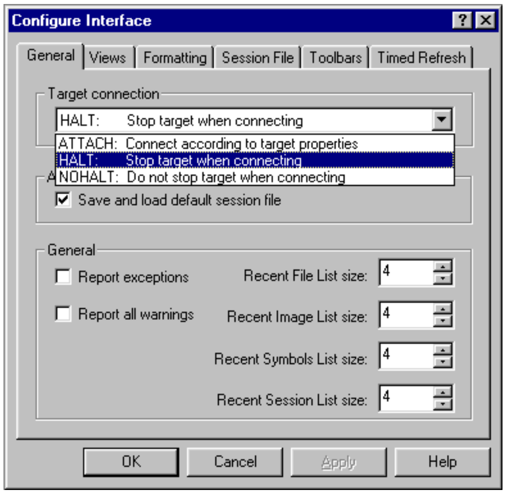
```
AXD -session path\sessionfile.ses
```

Refer to the *AXD and armsd Debuggers Guide* for more information on starting AXD and the arguments it uses.

2.3.2 AXD configuration

You can use the **Target connection** list control in the **Configure Interface** dialog box (see Figure 2-1). Select **NOHALT: Do not stop target when connecting** to prevent AXD from stopping the target program executing when it first connects.

If the selected debug agent is unable to do this, AXD prompts you to change the configuration.



Target connection
HALT: Stop target when connecting
ATTACH: Connect according to target properties
HALT: Stop target when connecting
NOHALT: Do not stop target when connecting

☑ Save and load default session file

General
☐ Report exceptions
☐ Report all warnings
Recent File List size: 4
Recent Image List size: 4
Recent Symbols List size: 4
Recent Session List size: 4

Figure 2-1 The AXD Configure Interface dialog box

2.3.3 AXD behavior during target program execution

With the version of AXD supplied with TDT, you can view source code and disassembly of your program while it is running.

——— **Note** ———

While the target program is running, the disassembly view is based on a copy of the image held by the debugger, and therefore might be out of date if you view memory that is written to. You can only inspect memory that is not contained in the image file if the debug agent, for example RealMonitor, can read memory while the target program is running.

———————————

2.4 Tracing multiprocessor systems

TDT enables you to trace multiprocessor systems. You require one instance of a debugger per processor, and the ability to connect to several ETMs or XScales at once, because the ETM/XScale interface can only be used by one processor at a time. The general principles of using TDT in a multiprocessor system are:

- there is one debugger, for example a TDT-enabled AXD, per processor
- there is one trace capture unit per processor
- there is a trace control connection per processor, which can if needed share a JTAG port and interface

The rest of this section is an example of how you can connect a single Multi-ICE interface unit and two MultiTrace interface units to debug a multiprocessor target.

2.4.1 Multiprocessor TDT example

The example is based on a configuration of a development board and two ARM cores, with three workstations providing the user interface. This configuration is shown in Figure 2-2.

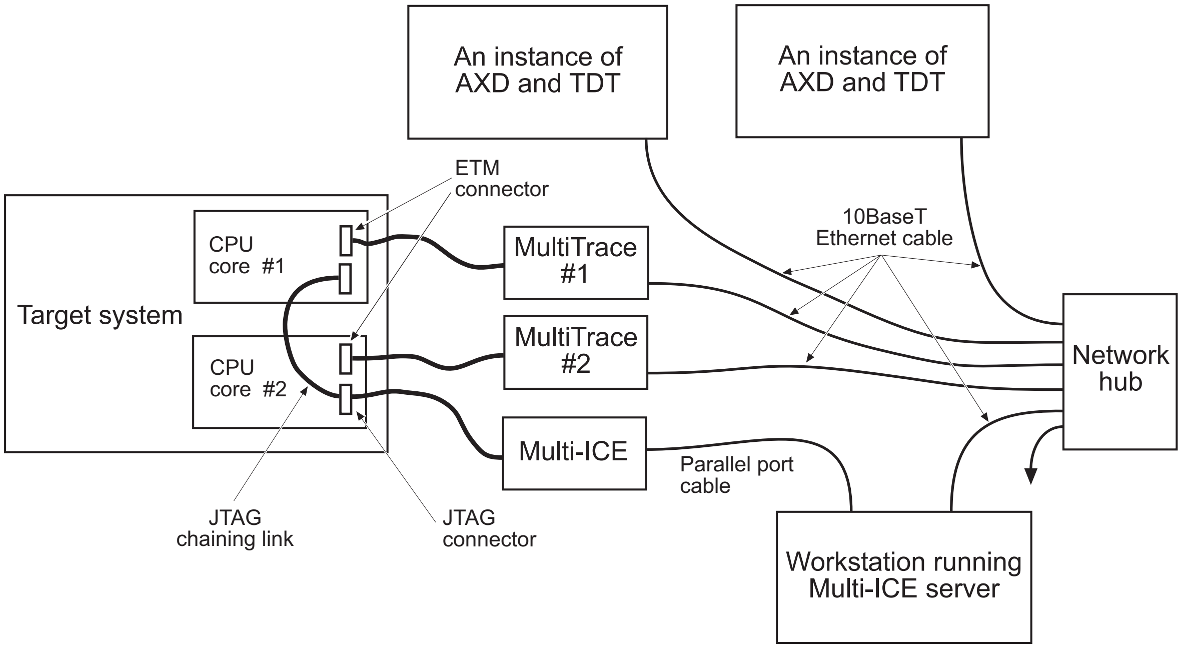


Figure 2-2 TDT Multiprocessor example

The target is set up with the JTAG lines of CPU cores connected serially. This means you only have to connect the Multi-ICE interface unit to the JTAG port on one core module. The ETM ports cannot be chained, and so you must connect a MultiTrace interface to each of the two ports individually.

You must connect Multi-ICE to a Windows workstation that can run the Multi-ICE server (it can be one of the machines running a debugger if required) and the MultiTrace units must be connected to the local network using the 10BaseT connector. When these connections have been made, you can power up the system.

You must install ADS 1.2, Multi-ICE Version 2.0, and TDT 1.2 on the workstations that you are using to debug, and Multi-ICE Version 2.0 on the workstation that you are using to drive the Multi-ICE interface unit. Configure the MultiTrace units using the `TPAConfig` utility, described in the *MultiTrace User Guide*, and configure the Multi-ICE server, described in the *Multi-ICE User Guide*.

You can now run AXD on the debug workstations and configure each to connect to their respective processors.

2.4.2 Multiprocessor TDT using other hardware

TDT can be used in multiprocessor configurations using other hardware for trace control and trace capture. The details of this vary between trace hardware manufacturers. For example, it is possible to set up the Tektronix TLA 700 Logic Analyzer to capture trace information from a second ETM port if the secondary capture cable is installed.

Chapter 3
Trace Facilities

This chapter introduces the TDT facilities used for gathering, generating, and configuring real-time trace information. It also provides a step-by-step procedure for using TDT in a typical trace session. For full details on using the specific trace configuration facilities, see Chapter 4 *Configuring the ETM* and Chapter 5 *Configuring Trace Capture*. For details on the procedure if you are using an XScale target, see *The tracing procedure with XScale* on page 7-4.

This chapter contains the following sections:

3.1 The Trace menu

When the AXD upgrade containing TDT is properly installed, the **Trace** menu is available in the AXD **Processor Views** menu, as shown in Figure 3-1.

——— **Note** ———

Some of the items in the **Trace** menu are disabled if you are connected to an XScale target. See *The XScale Trace menu* on page 7-8 for more information.

———

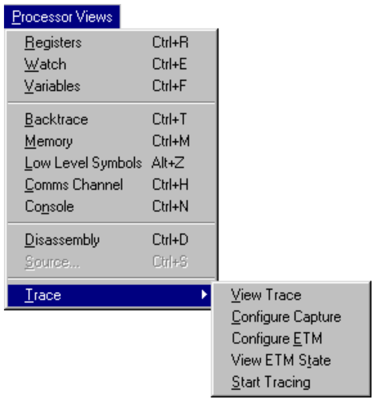


Figure 3-1 Trace menu

These menu items are also available on a context menu, by right-clicking on the ETM or XScale icon that appears under the processor containing the ETM or XScale in the target processors list, as shown in Figure 3-2.

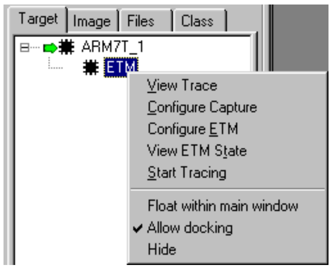


Figure 3-2 Trace Context menu

The other options are described in this chapter, in Chapter 4 *Configuring the ETM*, and in Chapter 5 *Configuring Trace Capture*.

Note

The **Start Tracing** option is replaced by **Stop Tracing** while tracing is underway.

3.1.1 Using help

The ADS manuals, including the TDT User Guide, are available as DynaText books. You can run the DynaText reader software by clicking **Programs** → **ADS v1.2** → **Online Books** from the **Start** menu or by selecting the AXD **Online Books** from the **Help** menu.

You can select **TDT Help** from the AXD **Help** menu at any time. This invokes the online help for TDT, with which you can navigate the complete help system. You can obtain help on individual topics as follows:

- For item-sensitive help, you can either:
  - click the ? symbol that appears in each dialog box, place the mouse pointer over the item for which you want information, and click again
  - click inside the field, or option, for which you want information, and press <SHIFT><F1>.
- For screen-sensitive help on individual dialog boxes, click on the **Help** dialog button, or press <F1> in the dialog box to invoke the appropriate TDT help information.

The exact version number of AXD is the same as for TDT, and is displayed by selecting **About AXD** from the AXD **Help** menu.

3.2 Tracing procedure

This section describes the steps you can use in a typical trace session using an ARM ETM. See *The tracing procedure with XScale* on page 7-4 for information on the equivalent procedure for XScale.

It is suggested that you follow these steps using one of the ADS sample programs, such as the `interwork` project found in the `…\examples\interwork` directory, before gathering trace information with your own executable file. You can use the ETM ARMulator as your first trace target (see *ARMulator ETM emulation* on page A-14), so you can become used to using TDT without the complexity of setting up real hardware.

For each session, you must perform certain operations in a specific order. However, you can select **TDT Help** from the AXD **Help** menu at any time (see *Using help* on page 3-3). Before proceeding with your first trace session, it is recommended that you first determine an area in your code that you will trace.

To trace a program:

1. Load the program into the debugger:
   - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.

     ——— **Note** ———

     After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

     ————————
   - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

   ——— **Note** ———

   If you do not load the program symbols into the debugger, you can still trace program execution and data values, but you cannot use symbolic names to refer to addresses, and the debugger cannot link instructions with source code. If you have not loaded program symbols into the debugger, keep the Trace window closed while the program is running, otherwise you will see a large number of error messages.

   ————————

2. Check the overall configuration of the ETM hardware by using the Configure ETM dialog box. The options configured in this dialog box are persistent across AXD sessions, and so apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

3. Display the Trace window by selecting **View Trace** from the **Trace** menu and arrange the columns as required.

4. View the ETM resources, such as the number of address comparators, that are available by selecting **View ETM State** from the **Trace** menu (see *View ETM State* on page 3-8).

5. Set conditions to either:
   - Ensure that all trace information is returned (see *Using the Enable Tracing action* on page 3-10).
   - Limit trace output to correlate to a specific area of execution. See Chapter 5 *Configuring Trace Capture* for complete details on setting conditions and triggers.

   The Start tracing now? or Restart tracing now? dialog box is displayed when you change the trace capture configuration. If you click **Yes**, the debugger enables tracing. This writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values (see *Controlling trace capture* on page 3-11).

   ——— **Note** ———

   If you are using breakpoints, all trace information prior to the breakpoint is lost when execution continues beyond the breakpoint.

   ————————

6. If your program is not already running, select **Go** from the **Execute** menu. Trace information is returned to the Trace window when:
   - the target processor stops executing the application program, for example if you click **Stop** from the **Execute** menu
   - a trigger action has fired and the trace buffer is full
   - you select **Stop Tracing** from the Trace window context menu.

   See Chapter 6 *The Trace Window Display* for full details of trace output.

7. You can compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13).

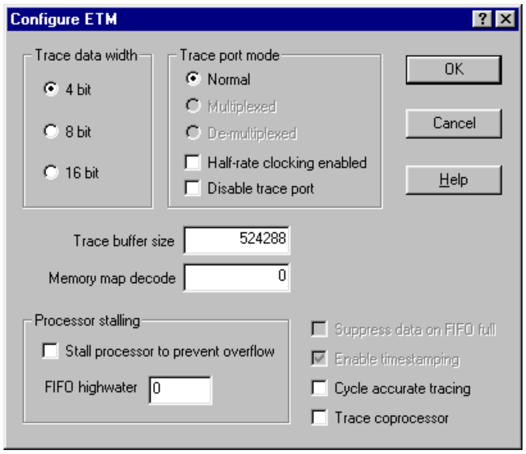
8. You can also link a row of trace output to its corresponding area of memory in the Memory window (see *Viewing memory at a location derived from trace output* on page 3-20). This window displays the values in target processor memory at the last point the debugger could read target memory, and might therefore differ from the data values shown in the trace display.

9. Optionally, you can select **Reload trace** from the Trace window context menu. This option is available when some form of information is visible in the Trace window. Use it if the displayed trace is incorrect because the debug symbol information is incorrect or incomplete. For example, if decoding the trace requires access to an RTOS kernel and the kernel image is not loaded (see *Reloading trace* on page 3-22 for details).

10. You can save the trace output to a file (see *Saving the trace output to file* on page 3-23).

3.3 Configure ETM

For every trace operation you perform with an ARM ETM, you can customize some general trace capture options using the Configure ETM dialog box. A typical display is shown in Figure 3-3.



Trace data width: [✓ 4 bit] 8 bit 16 bit

Trace port mode: [✓ Normal] Multiplexed De-multiplexed

[ ] Half-rate clocking enabled

[ ] Disable trace port

Trace buffer size 524288

Memory map decode 0

Processor stalling

[ ] Stall processor to prevent overflow

FIFO highwater 0

[ ] Suppress data on FIFO full

[✓] Enable timestamping

[ ] Cycle accurate tracing

[ ] Trace coprocessor

Figure 3-3 Configure ETM dialog box

For details on setting all the configurable options in the Configure ETM dialog box, see Chapter 4 *Configuring the ETM*. For details on configuring XScale trace, see *The tracing procedure with XScale* on page 7-4.

3.4 View ETM State

You can select **View ETM State** from the **Trace** menu to display the ETM State dialog box when using TDT with an ARM ETM (Figure 3-4).

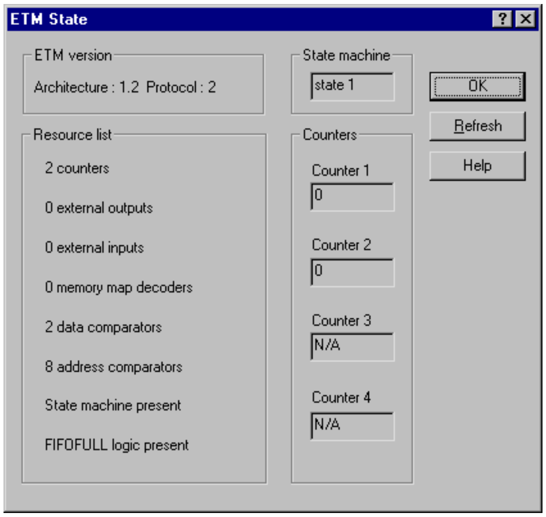


Figure 3-4 ETM State dialog box

The dialog box is split into the following sections:

ETM version

This field displays the architecture and protocol numbers of the ETM that you are connected to. Reference to protocol versions is deprecated in ETMs supporting architecture ETMv2 and above. For more information on protocol version numbers and architecture names, refer to Appendix D *ETM Versions*.

Resource list This area displays the resources that TDT has found in the ETM. Resources such as the `FIFOFULL` logic might be implemented by the ETM but not connected to the processor core, and so not operate. Refer to the hardware manual for your processor for more information.

State machine This area displays either:

- `N/A` (not applicable) if there is no state machine in the ETM
- the state number of the state machine at the time the dialog box is displayed.

The state machine is reset to state 1 when a breakpoint is hit.

Counters This area displays information for up to four counters. For each counter the field is:

- `N/A` (not applicable) if the counter is not implemented in the ETM
- the count at the time the dialog box is displayed.

The counter is reset to zero when a breakpoint is hit.

——— **Note** ———

On power-up of the board, the Counter values are random. If you are using a counter resource, the Counter values are set to the target value, and decrement whenever the counter is activated. See *Counter* on page 5-33 for details on setting this value.

————————

The buttons on the right of the dialog enable you to perform the following actions:

- Click **OK** to close the ETM State dialog box
- Click **Help** to get help on the ETM State dialog box
- Click **Refresh** to refresh the counter and state machine values.

——— **Note** ———

The **Refresh** button updates only the counter and state machine values. The list of available resources is not updated, but continues to show the resources that are built into the ETM. The number of resources that you have used is not reflected in this dialog.

If you have added new resources, for example as a result of changing your target, you will need to restart AXD before the new values are displayed in the ETM Stage dialog.

————————

3.5 Using the Enable Tracing action

Although you can limit the trace output in a variety of ways (see Chapter 5 *Configuring Trace Capture* for complete details on setting conditions), you can also set a *catch all* condition to ensure that every instruction executed generates trace information.

This section describes how to use the **Enable Tracing** action. You must ensure your ETM is configured appropriately for the hardware you are using (see Chapter 4 *Configuring the ETM* and Appendix B *Setting up the Trace Software*).

To use the **Enable Tracing** action:

1. Select **Configure capture** from the **Trace** menu. The Configure capture dialog box is displayed, as shown in Figure 3-5.

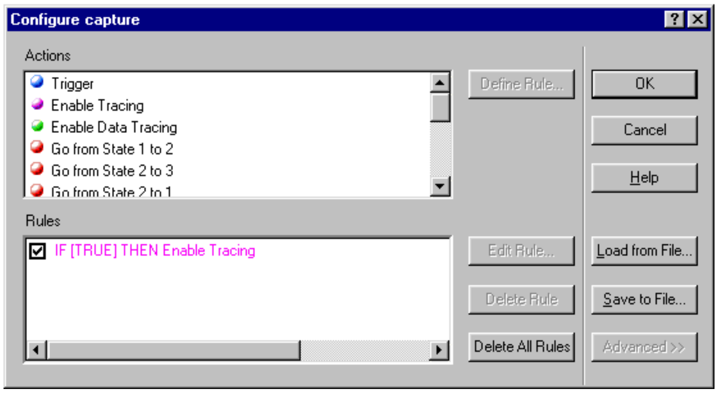


Figure 3-5 Configure capture dialog box

2. Double-click **Enable Tracing** in the left-hand Actions list. The Resource selection dialog box is displayed.

3. Select **TRUE** as your resource from the left-hand Resource drop-down list. Click **Next**. The Rule definition complete dialog box appears. Click **Finish**. The Configure capture dialog box is displayed again.

4. Click **OK** to accept the new rule. The Start tracing now? message box is displayed, asking whether you want to start tracing at this point. To start tracing, click **Yes**. Alternatively, click **No** and select **Start Tracing** from the **Trace** window context menu.

For complete details on this action, see *Enable Tracing* on page 5-9.

3.6 Controlling trace capture

This section describes the ways you can control the capture of trace data.

3.6.1 Start tracing

After completing the Configure ETM dialog box (see Chapter 4 *Configuring the ETM*) and setting the trace rules (see Chapter 5 *Configuring Trace Capture*), you must indicate that you want tracing to start. You can do this in either of the following ways:

- click **Yes** in the Start tracing now? message box that is displayed after clicking **OK** in the Configure ETM or Configure capture dialog box
- select **Start Tracing** from the **Trace** menu.

3.6.2 Stop tracing

To stop trace information from being returned to the Trace window, you can select **Stop Tracing** from the **Trace** window context menu after tracing has started. At this point, the trace capture hardware stops collecting trace data, and any trace data collected is displayed in the Trace window. This display reflects only the instructions executed prior to selecting **Stop Tracing**. You can restart tracing at any time by selecting **Start Tracing** from the **Trace** window context menu.

Note

If you restart tracing, any trace information that has already been captured by hardware trace capture buffers is lost and the Trace window is cleared of all contents.

3.6.3 Restart tracing

You can select **Configure ETM** or **Configure capture** from the **Trace** menu during tracing. You can then examine and change the set details in these dialog boxes. If you press **OK** (rather than **Cancel**) in either of these dialog boxes (regardless of whether you have changed any details), a Restart tracing now? message box is displayed. If you select **Yes** in this message box, tracing stops, and then restarts according to the new details you have set. If you select **No**, tracing continues according to the previously set details.

Note

If you restart tracing, any trace information that has already been captured by hardware trace capture buffers is lost and the Trace window is cleared of all contents.

3.7 Adjusting the displayed columns

After your program has executed, any captured trace information is directed to the Trace window. This section describes the methods you can use to add or remove columns, adjust column width, and change column order in the Trace window.

3.7.1 Adding or removing columns

As installed, all of the available columns are shown in the Trace window. To add or remove a column from the window display, right-click the mouse and select **Columns**. You can select or deselect the columns that you want to add or remove, respectively, from the display (see *Trace-specific GUI* on page 6-6).

3.7.2 Adjusting the column width

To adjust the width of a column, drag the heading divider of the desired column to the right (for a wider column) or to the left (for a narrower column). The columns to the right of the divider move to accommodate the new column width.

Note

If you reduce the width of a heading so much that it is no longer visible, you can make it visible again by dragging the heading divider again. The **Columns** entry on the Trace menu shows that column as displayed (checked).

3.7.3 Changing the column order

To change the order of the columns, drag the heading of the desired column to a new position. The other columns move to accommodate the new placement.

3.8 Viewing trace output with source code

When you have captured your trace data, you can view it in the Trace window. Trace decompression occurs only for the sections of trace data that are required by the currently displayed page of the Trace window.

Show the Trace window by selecting **View Trace** from the **Trace** menu. Trace information is shown in the Trace window when all of the following are true:

- there is an Enable Tracing rule in the trace capture rule set
- you answer **Yes** to the Start Tracing now? dialog box or the Restart tracing now? dialog box, or select **Start Tracing** from the **Trace** window context menu
- the debugger is connected to a target processor and the target has run code
- the Enable Tracing rule is TRUE after **Start Tracing** is enabled
- TDT receives and decompresses the trace information from the trace capture hardware.

You can choose to display:

Addresses and data

This shows the sequence of ARM or Thumb instructions captured by the ETM, with the oldest executed instructions at the top of the view. See *Displaying only traced instructions and data* on page 3-15.

Sync view

Do this by synchronizing the Trace window to the source file view. Activate a debugger window for the relevant source file, and select (checkmark) **Sync view** on the Trace window context menu. See *Displaying trace information in one window, source lines in another* on page 3-16.

Interleave source

Do this by clicking **Interleave source** in the Trace window context menu. See *Displaying trace information mixed with corresponding source lines* on page 3-18.

The Trace window context menu items relating to locating source and changing view options are disabled until trace information is available.

The Trace window shows a view of the decoded trace information. One of the lines in the Trace window is highlighted, and this is called the trace cursor. You can show different parts of the trace information in any of the following ways:

- using the mouse and vertical scroll bar
- use the up and down cursor keys to move by line
- use the page up and page down keys to move by approximately the Trace window height
- use the home key to go to the top of the buffer (the oldest instructions) and the end key to go to the bottom of the buffer (the youngest instructions)
- using **Goto index...** on the Trace window context menu to select a specific line.

The information shown in the Trace window can also provide a location for the debugger memory, source, or disassembly windows:

- Right click on the trace line to use as the location, and in the resulting Trace window context menu select one of the items on the **Locate to** menu. The debugger window used depends on the context:
  - If the line clicked is an instruction line, or a data line with an associated address, the requested window is opened and displayed.

    If the Address column on the selected data line is blank, all **Locate to** menu items are disabled. If there is no source available for the selected address then the **Source** menu item is disabled.
  - If the line clicked is a source line (because you have interleaved source enabled), then you can either display the same source line in its own view, or use the address of the first instruction for that source line to locate a disassembly or memory view.
- Double-click on a line in the Trace window to locate to a debugger window. The window used depends on the context:
  - If the line clicked is a data transfer (because you enabled data tracing), and the Address column is not blank, the memory window is opened and displayed.

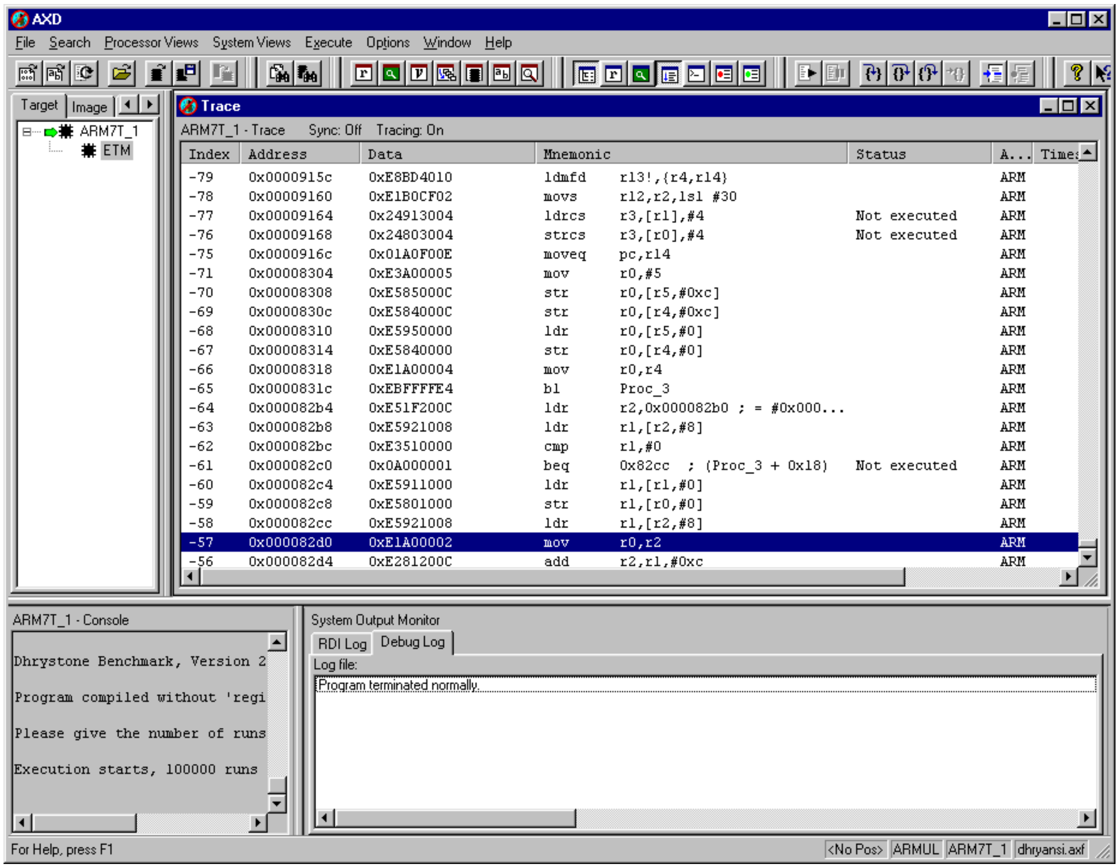
    If the Address column is blank then nothing happens.
  - If a source window is active and the line clicked is an instruction or an interleaved source line, then, if known, the source line corresponding to the selected line is displayed.

    If no source line is available, then nothing happens.
  - If a disassembly window is active, then the instructions in that region of memory are disassembled and displayed.

To keep a debugger window in step with the Trace window, so that when the Trace window cursor moves the debugger window contents change as well, checkmark the **Sync view** item on the Trace context menu. You cannot synchronize a source and a disassembly view to the trace cursor at the same time.

3.8.1 Displaying only traced instructions and data

The trace window displays only instructions and data by default. Show the Trace window by selecting **View Trace** from the **Trace** menu. Figure 3-6 shows the Trace window displaying instructions from a run of the Dhrystone example code.


AXD
File Search Processor Views System Views Execute Options Window Help
Target Image
ARM7T_1
ETM
Trace
ARM7T_1 - Trace Sync: Off Tracing: On
ARM7T_1 - Console
System Output Monitor
RDI Log Debug Log
Log file:
For Help, press F1
<No Pos> ARMUL ARM7T_1 dhryansi.axf

Index	Address	Data	Mnemonic		Status	A...	Time
-79	0x0000915c	0xE8BD4010	ldmfd	r13!,{r4,r14}		ARM	
-78	0x00009160	0xE1B0CF02	movs	r12,r2,lsl #30		ARM	
-77	0x00009164	0x24913004	ldrcs	r3,[r1],#4	Not executed	ARM	
-76	0x00009168	0x24803004	strcs	r3,[r0],#4	Not executed	ARM	
-75	0x0000916c	0x01A0F00E	moveq	pc,r14		ARM	
-71	0x00008304	0xE3A00005	mov	r0,#5		ARM	
-70	0x00008308	0xE585000C	str	r0,[r5,#0xc]		ARM	
-69	0x0000830c	0xE584000C	str	r0,[r4,#0xc]		ARM	
-68	0x00008310	0xE5950000	ldr	r0,[r5,#0]		ARM	
-67	0x00008314	0xE5840000	str	r0,[r4,#0]		ARM	
-66	0x00008318	0xE1A00004	mov	r0,r4		ARM	
-65	0x0000831c	0xEBFFFFE4	bl	Proc_3		ARM	
-64	0x000082b4	0xE51F200C	ldr	r2,0x000082b0 ; = #0x000...		ARM	
-63	0x000082b8	0xE5921008	ldr	r1,[r2,#8]		ARM	
-62	0x000082bc	0xE3510000	cmp	r1,#0		ARM	
-61	0x000082c0	0x0A000001	beq	0x82cc ; (Proc_3 + 0x18)	Not executed	ARM	
-60	0x000082c4	0xE5911000	ldr	r1,[r1,#0]		ARM	
-59	0x000082c8	0xE5801000	str	r1,[r0,#0]		ARM	
-58	0x000082cc	0xE5921008	ldr	r1,[r2,#8]		ARM	
-57	0x000082d0	0xE1A00002	mov	r0,r2		ARM	
-56	0x000082d4	0xE281200C	add	r2,r1,#0xc		ARM	

```
Dhrystone Benchmark, Version 2

Program compiled without 'regi

Please give the number of runs

Execution starts, 100000 runs
```

```
Program terminated normally.
```

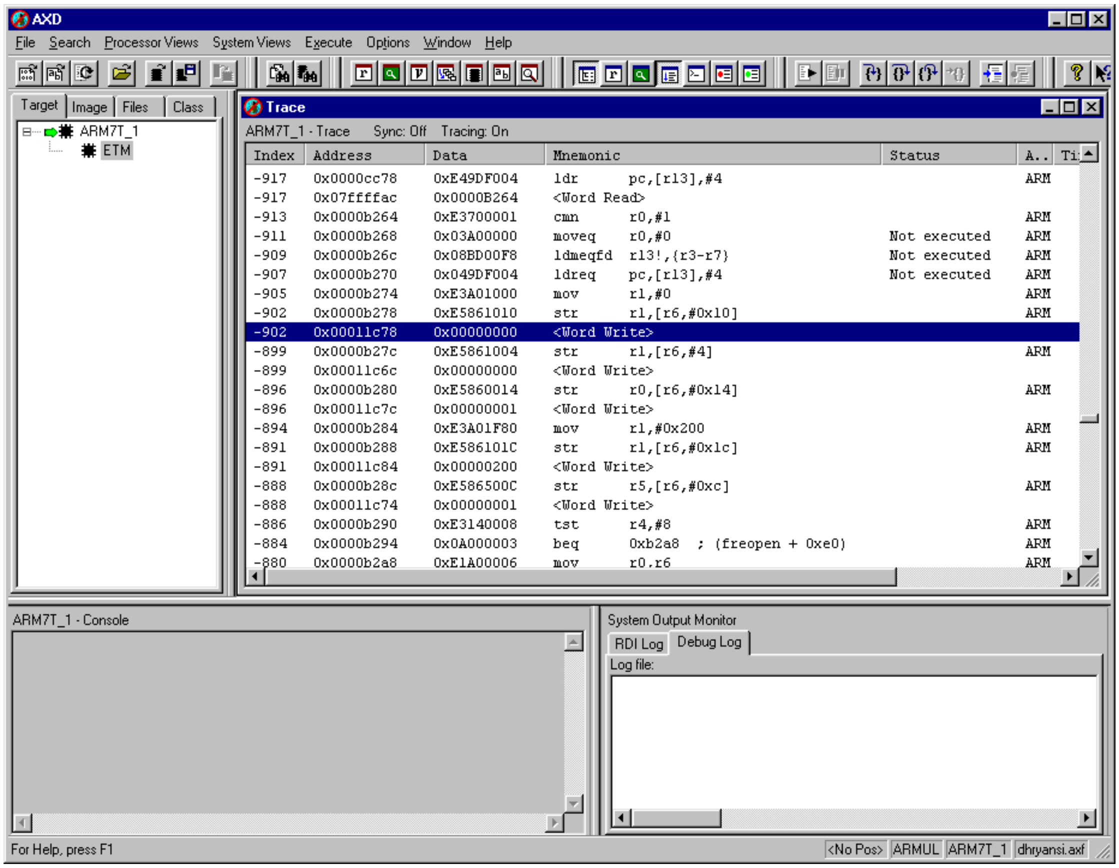
Figure 3-6 AXD with a Trace window

You can display traced data values transferred over the processor memory bus by defining an Enable Data Tracing rule. This is shown in Figure 3-7 on page 3-16, in this case with the Enable Data Tracing option **Addresses and data** selected.

The Address column for data transfers displays the memory location read or written by the processor, if address tracing has been enabled in the Enable Data Tracing rule.

The Data column shows the value read, if data value tracing has been enabled in the Enable Data Tracing rule.

In the Mnemonic column of the Trace window, the text in the Mnemonic column for data transfer cycles indicates the size and direction of the transfer. For example, `<Word Write>` indicates a 32-bit write to memory.


AXD
File Search Processor Views System Views Execute Options Window Help
Target Image Files Class
ARM7T_1
ETM
Trace
ARM7T_1 - Trace Sync: Off Tracing: On
ARM7T_1 - Console
System Output Monitor
RDI Log Debug Log
Log file:
For Help, press F1
<No Pos> ARMUL ARM7T_1 dhryansi.axf

```
Index  Address     Data        Mnemonic                              Status        A..  Ti
-917   0x0000cc78  0xE49DF004  ldr      pc,[r13],#4                               ARM
-917   0x07ffffac  0x0000B264  <Word Read>
-913   0x0000b264  0xE3700001  cmn      r0,#1                                     ARM
-911   0x0000b268  0x03A00000  moveq    r0,#0                        Not executed ARM
-909   0x0000b26c  0x08BD00F8  ldmeqfd  r13!,{r3-r7}                 Not executed ARM
-907   0x0000b270  0x049DF004  ldreq    pc,[r13],#4                  Not executed ARM
-905   0x0000b274  0xE3A01000  mov      r1,#0                                     ARM
-902   0x0000b278  0xE5861010  str      r1,[r6,#0x10]                             ARM
-902   0x00011c78  0x00000000  <Word Write>
-899   0x0000b27c  0xE5861004  str      r1,[r6,#4]                                ARM
-899   0x00011c6c  0x00000000  <Word Write>
-896   0x0000b280  0xE5860014  str      r0,[r6,#0x14]                             ARM
-896   0x00011c7c  0x00000001  <Word Write>
-894   0x0000b284  0xE3A01F80  mov      r1,#0x200                                 ARM
-891   0x0000b288  0xE586101C  str      r1,[r6,#0x1c]                             ARM
-891   0x00011c84  0x00000200  <Word Write>
-888   0x0000b28c  0xE586500C  str      r5,[r6,#0xc]                              ARM
-888   0x00011c74  0x00000001  <Word Write>
-886   0x0000b290  0xE3140008  tst      r4,#8                                     ARM
-884   0x0000b294  0x0A000003  beq      0xb2a8  ; (freopen + 0xe0)                ARM
-880   0x0000b2a8  0xE1A00006  mov      r0,r6                                     ARM
```

Figure 3-7 Trace window with trace data and instructions

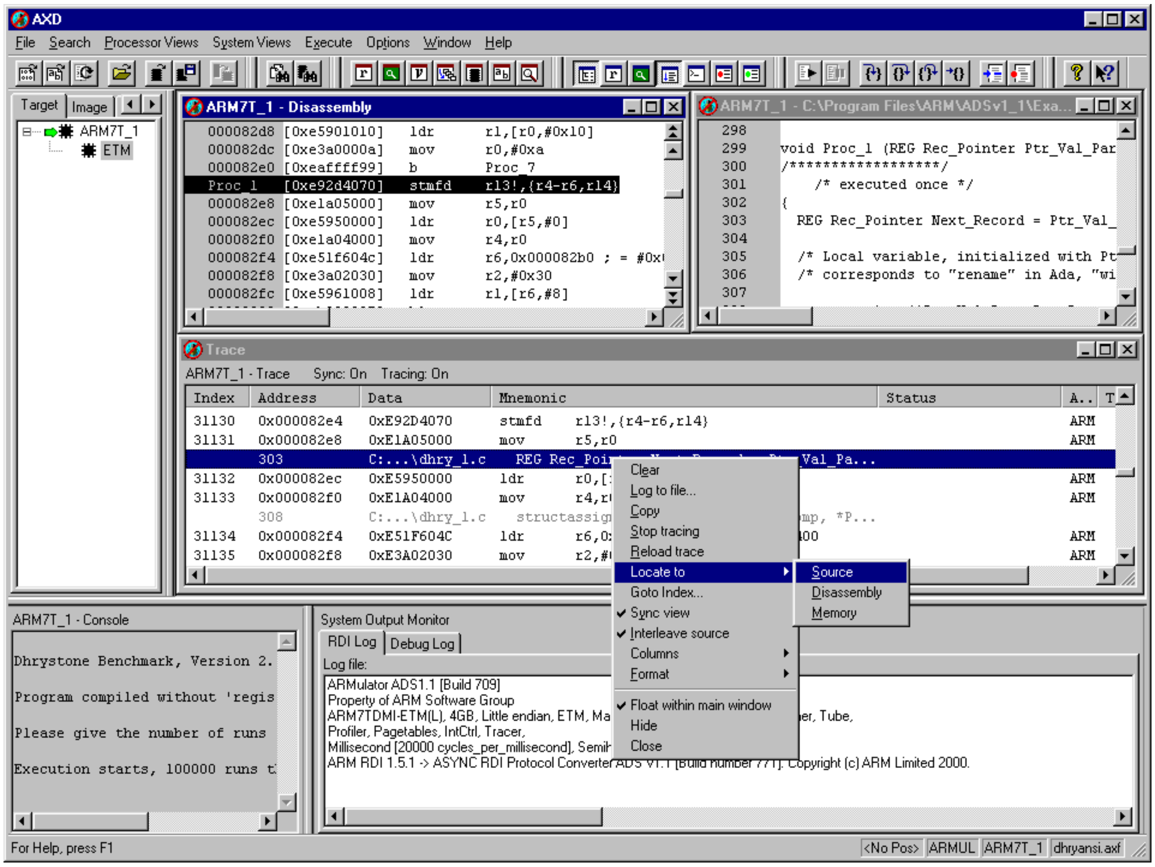
3.8.2 Displaying trace information in one window, source lines in another

To display trace information in one window and source lines in another:

1. Show the Trace window by selecting **View Trace** from the **Trace** menu.

2. Show the source view for the code you are inspecting in another window, by either:
   - placing the cursor over a source or disassembly line and selecting **Locate to** → **Source** on the Trace window context menu
   - selecting **Processor Views** → **Source...** on the AXD main menu and selecting the correct source file in the Open Source dialog box.
3. Select **Sync view** from the Trace window context menu.

Figure 3-8 shows the Trace window from a run of Dhrystone, with the context menu showing **Locate to** → **Source** on line `303`, and the source file `dhry_2.c` centred on line `303`.



```
000082d8 [0xe5901010]  ldr    r1,[r0,#0x10]
000082dc [0xe3a0000a]  mov    r0,#0xa
000082e0 [0xeaffff99]  b      Proc_7
Proc_1   [0xe92d4070]  stmfd  r13!,{r4-r6,r14}
000082e8 [0xe1a05000]  mov    r5,r0
000082ec [0xe5950000]  ldr    r0,[r5,#0]
000082f0 [0xe1a04000]  mov    r4,r0
000082f4 [0xe51f604c]  ldr    r6,0x000082b0 ; = #0x
000082f8 [0xe3a02030]  mov    r2,#0x30
000082fc [0xe5961008]  ldr    r1,[r6,#8]
```

```
298
299  void Proc_1 (REG Rec_Pointer Ptr_Val_Par
300  /******************/
301      /* executed once */
302  {
303    REG Rec_Pointer Next_Record = Ptr_Val_
304
305    /* Local variable, initialized with Pt
306    /* corresponds to "rename" in Ada, "wi
307
```

```
ARM7T_1 - Trace     Sync: On    Tracing: On

Index  Address     Data           Mnemonic                         Status   A..  T
31130  0x000082e4  0xE92D4070     stmfd    r13!,{r4-r6,r14}                 ARM
31131  0x000082e8  0xE1A05000     mov      r5,r0                            ARM
       303         C:...\dhry_1.c    REG Rec_Poi           Val_Pa...
31132  0x000082ec  0xE5950000     ldr      r0,[                             ARM
31133  0x000082f0  0xE1A04000     mov      r4,r                             ARM
       308         C:...\dhry_1.c    structassig           mp, *P...
31134  0x000082f4  0xE51F604C     ldr      r6,0             00              ARM
31135  0x000082f8  0xE3A02030     mov      r2,#                             ARM
```

```
Dhrystone Benchmark, Version 2.

Program compiled without 'regis

Please give the number of runs

Execution starts, 100000 runs t
```

```
ARMulator ADS1.1 [Build 709]
Property of ARM Software Group
ARM7TDMI-ETM(L), 4GB, Little endian, ETM, Ma                  er, Tube,
Profiler, Pagetables, IntCtrl, Tracer,
Millisecond [20000 cycles_per_millisecond], Semih
ARM RDI 1.5.1 -> ASYNC RDI Protocol Converter ADS v1.1 [Build number 771]. Copyright (c) ARM Limited 2000.
```

Figure 3-8 Synchronizing trace with source

In Figure 3-8 on page 3-17 there is a Trace window, a disassembly window, and a source window. If you:

- select the source window, then double-click on an instruction in the Trace window, the source window scrolls to show you the corresponding line of source (if it is available)
- select the disassembly window, then double-click on an instruction in the Trace window, the disassembly window scrolls to show you the corresponding section of disassembled code.

To use the context menu to locate the source for an instruction in the Trace window:

1. Right-click on the instruction to locate. The context menu appears, as shown at the bottom of Figure 3-8 on page 3-17.
2. Select **Locate to**. If the instruction:
   - corresponds to a source file line, and the debugger can access the file, **Source** is enabled in the submenu
   - does not correspond to a source file line, **Source** is grayed out.
3. Select **Source**. The source window for that file is opened and scrolled to the source line for that instruction.

You can use the **Sync View** menu item to synchronize the active source, disassembly, or memory view with the Trace window. When one of these debugger views is synchronized with the Trace window, moving the Trace window cursor also changes the contents of the synchronized debugger view.

3.8.3 Displaying trace information mixed with corresponding source lines

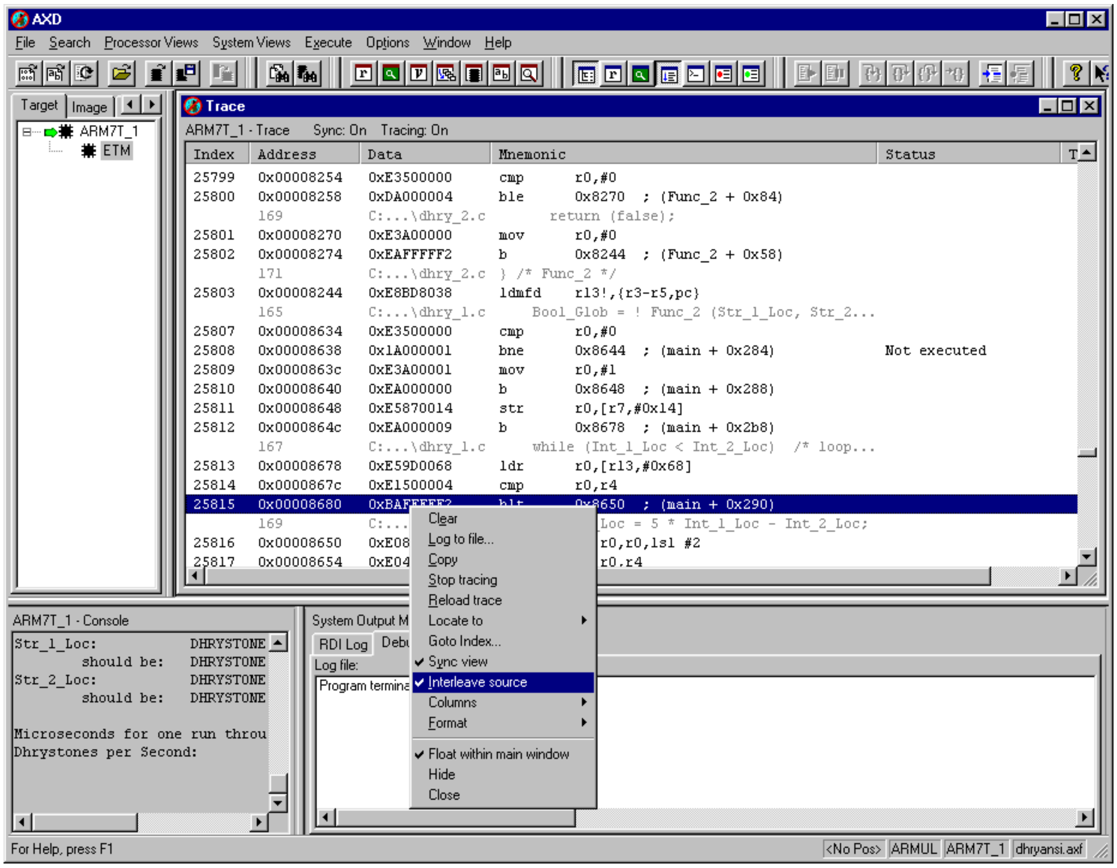
To display trace information mixed with corresponding source lines:

1. Show the Trace window by selecting **View Trace** from the **Trace** menu.
2. Select the menu item **Interleave source** on the Trace window context menu.

The Trace window is updated to show in gray the source lines corresponding to each group of instructions executed. If the traced instructions enter a memory region that has no available source, a message indicating this is included in gray just before the instruction:

```
-- No source available --
```

The next known source line is included in the listing with a file name and source file extract. Figure 3-9 on page 3-19 shows the Trace window from a run of Dhrystone, with the context menu showing **Interleave source**.



ARM7T_1 - Trace Sync: On Tracing: On

```
Index  Address     Data        Mnemonic                                   Status
25799  0x00008254  0xE3500000  cmp     r0,#0
25800  0x00008258  0xDA000004  ble     0x8270  ; (Func_2 + 0x84)
       169         C:...\dhry_2.c        return (false);
25801  0x00008270  0xE3A00000  mov     r0,#0
25802  0x00008274  0xEAFFFFF2  b       0x8244  ; (Func_2 + 0x58)
       171         C:...\dhry_2.c  } /* Func_2 */
25803  0x00008244  0xE8BD8038  ldmfd   r13!,{r3-r5,pc}
       165         C:...\dhry_1.c      Bool_Glob = ! Func_2 (Str_1_Loc, Str_2...
25807  0x00008634  0xE3500000  cmp     r0,#0
25808  0x00008638  0x1A000001  bne     0x8644  ; (main + 0x284)           Not executed
25809  0x0000863c  0xE3A00001  mov     r0,#1
25810  0x00008640  0xEA000000  b       0x8648  ; (main + 0x288)
25811  0x00008648  0xE5870014  str     r0,[r7,#0x14]
25812  0x0000864c  0xEA000009  b       0x8678  ; (main + 0x2b8)
       167         C:...\dhry_1.c      while (Int_1_Loc < Int_2_Loc)  /* loop...
25813  0x00008678  0xE59D0068  ldr     r0,[r13,#0x68]
25814  0x0000867c  0xE1500004  cmp     r0,r4
25815  0x00008680  0xBAFFFFF2  blt     0x8650  ; (main + 0x290)
       169         C:...                Loc = 5 * Int_1_Loc - Int_2_Loc;
25816  0x00008650  0xE08                r0,r0,lsl #2
25817  0x00008654  0xE04                r0,r4
```

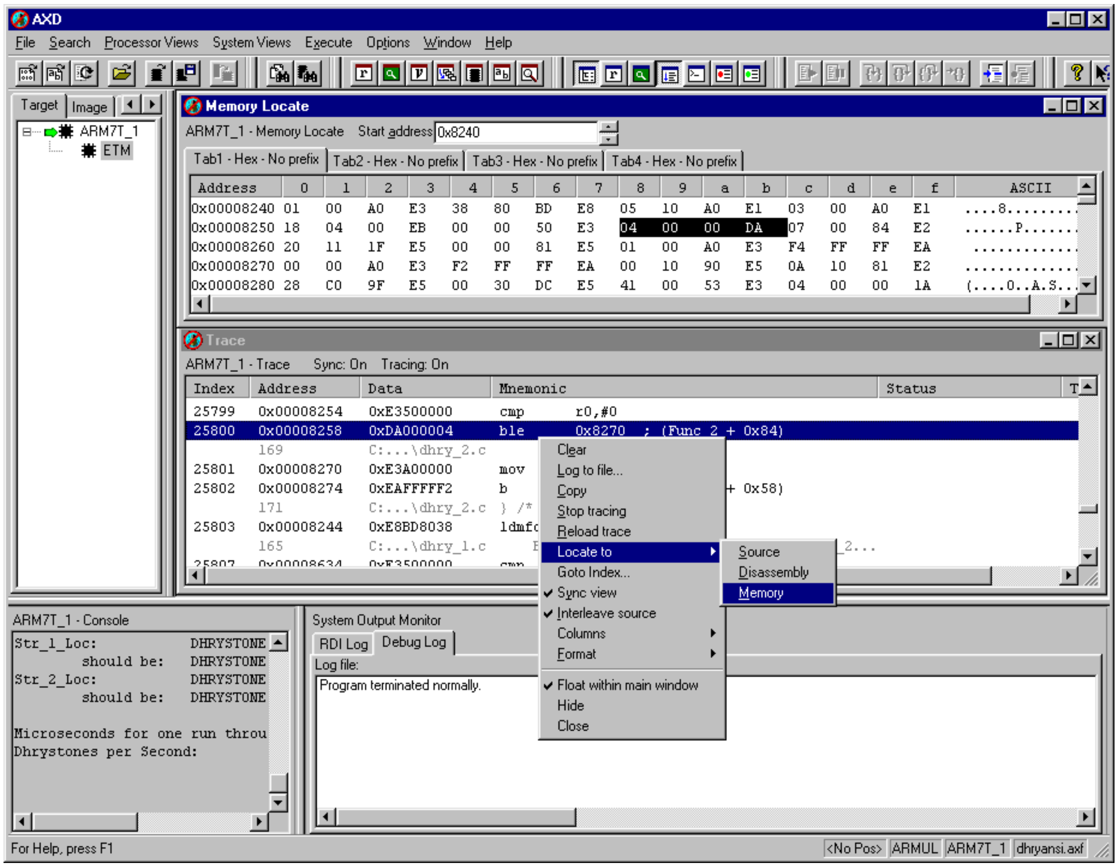
```
Str_1_Loc:           DHRYSTONE
        should be:   DHRYSTONE
Str_2_Loc:           DHRYSTONE
        should be:   DHRYSTONE

Microseconds for one run throu
Dhrystones per Second:
```

```
Program termina
```

Figure 3-9 Interleaving trace with source

3.9 Viewing memory at a location derived from trace output

You can use a Memory Locate window to display a range of addresses on either side of an address you select in the **Trace** window, as shown in Figure 3-10.



ARM7T_1 - Memory Locate Start address 0x8240

```
Address     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f   ASCII
0x00008240 01 00 A0 E3 38 80 BD E8 05 10 A0 E1 03 00 A0 E1  ....8........
0x00008250 18 04 00 EB 00 00 50 E3 04 00 00 DA 07 00 84 E2  ......P......
0x00008260 20 11 1F E5 00 00 81 E5 01 00 A0 E3 F4 FF FF EA   ............
0x00008270 00 00 A0 E3 F2 FF FF EA 00 10 90 E5 0A 10 81 E2  .............
0x00008280 28 C0 9F E5 00 30 DC E5 41 00 53 E3 04 00 00 1A  (....0..A.S...
```

ARM7T_1 - Trace Sync: On Tracing: On

```
Index  Address     Data        Mnemonic                              Status
25799  0x00008254  0xE3500000  cmp      r0,#0
25800  0x00008258  0xDA000004  ble      0x8270  ; (Func 2 + 0x84)
       169         C:...\dhry_2.c
25801  0x00008270  0xE3A00000  mov
25802  0x00008274  0xEAFFFFF2  b                       + 0x58)
       171         C:...\dhry_2.c  } /*
25803  0x00008244  0xE8BD8038  ldmfd
       165         C:...\dhry_1.c     F                _2...
25807  0x00008634  0xE3500000  cmp
```

ARM7T_1 - Console

```
Str_1_Loc:          DHRYSTONE
         should be: DHRYSTONE
Str_2_Loc:          DHRYSTONE
         should be: DHRYSTONE

Microseconds for one run throu
Dhrystones per Second:
```

System Output Monitor

Log file:

Program terminated normally.

Figure 3-10 Displaying the Memory Locate window

To view the Memory Locate window if it is not already open, either:

- In the Trace window, double-click in the address column of a traced memory access cycle. If multiple words are transferred by an instruction, for example by an `LDMIA`, you must double-click on the address for the first word transferred.

  Note

  You cannot link to a memory window by double-clicking on the address of an instruction access.

- In the Trace window, right-click on the desired address and select **Locate to → Memory** (see *Trace-specific GUI* on page 6-6).

In each case, the Memory window is displayed, showing a range of addresses before and after the chosen highlighted address.

——— **Note** ———

While the target application is running you can only link to the Memory window if you are using RealMonitor, because it is capable of reading target memory without stopping the target.

———————————

3.10 Reloading trace

After tracing has begun and your program has stopped executing, you can select **Reload trace** from the Trace window context menu to allow the decompression of all areas of memory for which it could not occur while the program was executing. Two reasons for using this option are:

Branching to a memory location outside the image

Due to an error in the source code, there might be a branch to a memory location that is not within the range of the image. In this case, TDT might fail to decompress the trace data because it cannot read the instructions at that address while the target is running. A symptom of this is the text `Synchronization lost` in the Trace window. To correct this, you can:

1. Do one of the following:
   - stop the target from running using AXD
   - wait for the target to stop at either a breakpoint or the end of the program.
2. Select **Reload trace** from the Trace window context menu. Because the target has stopped, all memory can now be read, and therefore TDT returns useful information for those areas outside the range of the image (see *Trace window output* on page 6-2 for details on interpreting trace information).

Note

If you are using RealMonitor, and the ETM has been programmed to exclude RealMonitor code from the trace, then this error does not occur, because the memory at the destination of the branch is read when required.

The wrong image was loaded

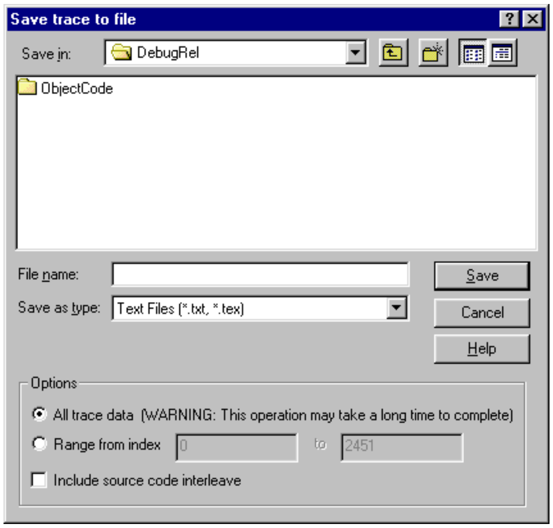
It is possible that the wrong image was loaded into AXD before or during program execution. To correct this, you can:

1. Load the correct image by selecting **Load symbols only…** from the **File** menu.
2. Select **Reload trace** from the Trace window context menu. This ensures that Trace information pertaining only to the desired image is displayed in the Trace window (see *Trace window output* on page 6-2 for details on interpreting trace information).

3.11 Saving the trace output to file

To save the complete decompressed trace output to a text file:

1. Select **Log to file** from the Trace window context menu. The Save trace to file dialog box is displayed (Figure 3-11).



All trace data (WARNING: This operation may take a long time to complete)
Range from index 0 to 2451
Include source code interleave

Figure 3-11 Save trace to file dialog box

2. Enter the filename, or select a previously existing file from the list or from another directory.
3. Click **All trace data** or **Range from index** to indicate the trace data to save.

   Warning

   If you select **All trace data**, the operation may take a long time to complete.

4. If you selected **Range from index**, enter the range in the range fields, in the format shown in the Index column of the Trace window. The default value for the range is taken from the Trace window selection area, or if there is no selection, is the whole buffer.
5. To include the interleaved source in the saved file, enable or disable **Include source code interleave**. This option is independent of the **Interleave source** item in the trace context menu, but the initial setting is the same as that in the Trace window.
6. Click on **Save**.

When trace is being saved, the **Log to file** option changes to read **Cancel log fil**e. Use this option to stop saving the trace at any time.

Chapter 4
Configuring the ETM

This chapter describes how to set the overall configuration of the ETM hardware . It contains the following section:

- *The Configure ETM dialog box* on page 4-2.

——— **Note** ———

The Configure ETM dialog is not enabled for XScale trace.

4.1 The Configure ETM dialog box

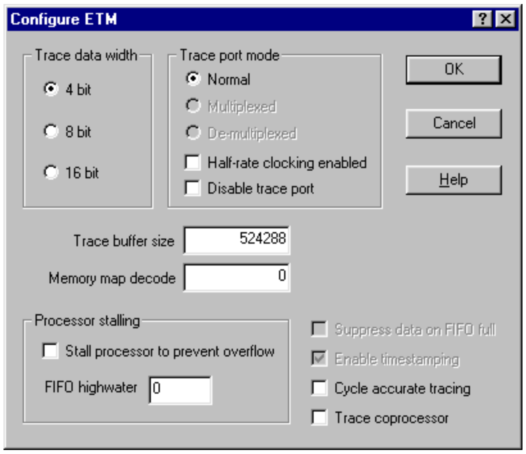
You can use this dialog box to configure the ETM behavior. To access this dialog box, select **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any tracing you perform, irrespective of the rules set up in the Configure capture dialog box (see Chapter 5 *Configuring Trace Capture*).

Many of the facilities provided by the ETM can be enabled or disabled in the hardware by the manufacturer, and some features used by TDT are dependent on support provided by the trace capture hardware. Because of this, mechanisms are provided to determine which facilities are actually implemented by the ETM and the trace capture hardware. However, there are facilities that might not be detected, and there are some optional features that cannot be autodetected. For example, if the FIFOFULL logic is present but not connected to the processor, the ETM does report that FIFOFULL logic is present but it does not operate. TDT enables or disables controls and resources to match the hardware using the following rules:

- if the feature is reported as being present, the controls are listed and enabled
- if the feature cannot be detected, but might be present in the ETM hardware being used, the controls are listed and enabled
- if the feature is not reported as present, then the controls are absent or disabled.

Before using a feature you must know that the combination of ETM and trace capture hardware that you are using supports it. Refer to the documentation for the device you are using and to the documentation for the trace capture hardware for more information.

This section describes the options available in the Configure ETM dialog box, as shown in Figure 4-1 on page 4-3. After you have configured the ETM, you must select **OK** to save the configuration, or **Cancel** to discard any changes.


Configure ETM

Trace data width: [✓ 4 bit] 8 bit 16 bit

Trace port mode: [✓ Normal] Multiplexed De-multiplexed

☐ Half-rate clocking enabled

☐ Disable trace port

Trace buffer size: 524288

Memory map decode: 0

Processor stalling:

☐ Stall processor to prevent overflow

FIFO highwater: 0

☐ Suppress data on FIFO full

✓ Enable timestamping

☐ Cycle accurate tracing

☐ Trace coprocessor

Figure 4-1 Configure ETM dialog box

The information displayed in this dialog box is stored in the AXD session file so that selections you make here are saved for future use. If you reload with the same session file, even if you do not display the ETM configuration dialog box, the saved settings are applied.

If you start AXD with a new session file, so that no saved information exists, the ETM configuration information is read from the hardware.

The options in the Configure ETM dialog box are:

Trace data width

The radio button selected shows the currently selected data port width. You can use this option to change the number of trace port pins used to broadcast trace information. This can be useful, for example, when trace port pins are multiplexed onto *General Purpose Input/Output* (GPIO) pins, and the hardware is configured to use these pins in their GPIO role. Also, some trace capture hardware, such as MultiTrace, can capture more cycles if a narrow port width is used.

Note

You must ensure that the width you select is supported by the hardware (both the target device and the trace capture hardware) you are using. If the width you select is not supported, no trace output appears in the Trace window.

Trace port mode

This control specifies the way the trace port is operated. It is set to one of the following values:

Normal	The normal mode. Trace data from the ETM is written to the output pins at the specified frequency.
Multiplexed	Use this to reduce the number of output pins used by the trace port. Two output signals are output on the same pin by clocking the signals at double the normal rate.
De-Multiplexed	Use this to reduce the signal switching frequency of the trace port signals. One output signal is output on two pins, so clocking the pins at half the normal rate.

Note

- Only **Normal** mode operation is possible when you are using ETM hardware implementing ETM version 1.0 or 1.1.
- If you use multiplexed or de-multiplexed clocks, you might have to alter the configuration of your logic analyzer or trace capture hardware.

Half-rate clocking enabled

Select this option if you want to set the ETM half-rate clocking signal. The effect of this signal is dependent on the implementation of your ASIC. For more details on half-rate clocking, see the *ETM control register* section of the *ARM Embedded Trace Macrocell Specification*.

Note

- This option is not available when you are using ETM hardware implementing ETM version 1.0. ETM hardware implementing ETM version 1.1 might support the option, but TDT cannot detect whether it does.
- If you enable half rate clocking, you might have to alter the configuration of your analyzer.
- The facility is supported by MultiTrace, but might not be supported by other trace capture hardware.

Disable trace port

Select this option if you want to suppress the output from the trace port. Use this to control trace output if your hardware has two or more ETMs sharing a single trace port.

TPA buffer size

This value represents the maximum number of cycles of trace that is stored by the trace capture hardware (see *The Trace capture hardware* on page 1-8). Typically, you set this to the maximum value that your trace capture hardware can support, but you can reduce it to speed up trace decompression. By default, this field shows the currently configured trace capture hardware buffer size.

With cycle-accurate tracing disabled, the trace capture hardware typically captures between one and two cycles for each instruction captured. If cycle-accurate tracing is enabled, it captures every cycle for which tracing is enabled.

Note

Some logic analyzer solutions might not support a variable buffer size, and instead always store the maximum number of cycles. Alternatively, the analyzer might only support certain set buffer sizes, and therefore selects a size closest to that requested. Refer to the manufacturer's handbook for details.

Memory map decode

This is an implementation-dependent value that can vary depending on the ASIC memory map decode logic (if any) present in your ASIC. This value is written to a control register, intended to configure the ASIC memory map decode hardware. For more details, refer to your ASIC documentation.

Processor stalling

The ETM contains a FIFO buffer that holds the traced data for transmission through the trace port. When this FIFO buffer becomes full, trace information is lost unless you have programmed the ETM to stall (temporarily stop) the processor.

The ASIC can include a facility to stall the processor until the FIFO buffer is empty. To enable this, select **Stall processor to prevent overflow**. When the number of bytes in the FIFO buffer is reduced to the number of bytes you set in **FIFO highwater**, the processor stalls as soon as possible. It restarts when the FIFO buffer is empty.

See *Enable Overflow Stalling* on page 5-13 for details on using this facility in a specific trace capture.

Note

Even if the **View ETM state** window indicates that FIFOFULL logic is present, processor stalling might not be supported. Please check your ASIC documentation to find out whether this feature is available.

Suppress data on FIFO full

Selecting this option suppresses the output from the trace port after a FIFO overflow occurs. Some versions of the ETM produce incorrect trace data following FIFO overflow. This only occurs on cached processors with slow memory systems, and happens when a cache miss occurs at the same time that the FIFO on the ETM overflows. If you select this option, the decompressor suppresses the data that the ETM might have traced incorrectly. However, some correctly traced data may also be suppressed.

Note

This option is disabled if your ETM does not generate incorrect trace data under these circumstances.

Enable Timestamping

Selecting this option enables the timestamp recording logic in the ETM. Timestamps are shown in nanoseconds, to one decimal place, and enable you to, for example, see when pauses have occurred in processor execution, and how long it takes between successive invocations of a particular section of code.

Note

The timestamp refers to the execution of instructions. This can be misleading in some cases. An example of this is when *Load Multiple (LDM)* and *Store Multiple (STM)* instructions are executed on processors in the ARM9 family. Instructions might leave the Execute stage of the pipeline after only one cycle, but still take many cycles to complete. In this case, the timestamp implies that an instruction has been executed in a single cycle when it has in fact taken many. Timestamps are therefore most useful for timing sections of code rather than individual instructions.

Transmitting the timestamps uses noticeable additional bandwidth on the trace capture hardware to host connection, so only enable this feature when you have to.

——— **Note** ———

This feature is not supported by all types of trace capture hardware. Refer to the documentation for your hardware for more information.

———————————

Cycle accurate tracing

Select this option if you want the ETM to operate in cycle-accurate mode. The two modes are:

Cycle-accurate mode

The ETM records the number of cycles executed while tracing is enabled. The **Index** column of the Trace window shows the cycle number of the cycle in which each instruction was executed (see *Trace window output* on page 6-2). The count does not include cycles executed during a trace discontinuity. You must use the **Timestamp** values in the **Trace** window to help you measure across discontinuities in the trace output (see **Timestamp** in *Trace window columns* on page 6-2).

Non cycle-accurate mode

The ETM does not record cycle counts. The Index column in the Trace window shows a row number relative to the trigger point (see *Trace window output* on page 6-2).

——— **Note** ———

Cycle-accurate tracing fills up the capture buffer faster than non cycle-accurate tracing because all wait cycles are captured.

———————————

Trace Coprocessor

Select this option if you want the data associated with *Move Coprocessor from ARM Register* (MCR) and *Move ARM Register from Coprocessor* (MRC) instructions to be traced.

Chapter 5
Configuring Trace Capture

This chapter describes the Configure capture dialog box. This allows you to set a wide variety of conditions that control the output of trace information.

——— **Note** ———

The Configure capture dialog box is not enabled for XScale trace.

———————————

This chapter contains the following sections:

5.1 About TDT capture configuration

Trace capture with the ETM uses a set of rules to determine the events you want to capture in the trace buffer. Within TDT these rules are set up using the **Configure Capture** dialog box, accessed from the **Trace** menu.

You are led through the process of creating capture actions using one of several capture wizards. These enable you to go back and modify previous choices as required until you are happy with your rule. The final page of each wizard summarizes the rule you have created.

To define or modify the configuration of the ETM, you must invoke the **Configure Capture** dialog box (see *The Configure capture dialog box* on page 5-5) from the **Trace** menu. You then double-click one or more of the actions from the **Actions** list to invoke the capture wizard for that action.

Note

- The rules associated with actions are evaluated in the ETM hardware in parallel on each cycle and so, unless you link them using the state machine or the counter, actions are completely independent of each other. Use the state machine to link rules together (see *State changes* on page 5-11 for more information).
- The implementation of the ETM might cause a short delay (for example, 1 cycle) between the condition for an action becoming TRUE and the action taking place.

Each action you select has a required condition and two optional conditions associated with it, as shown in Figure 5-1 on page 5-3.

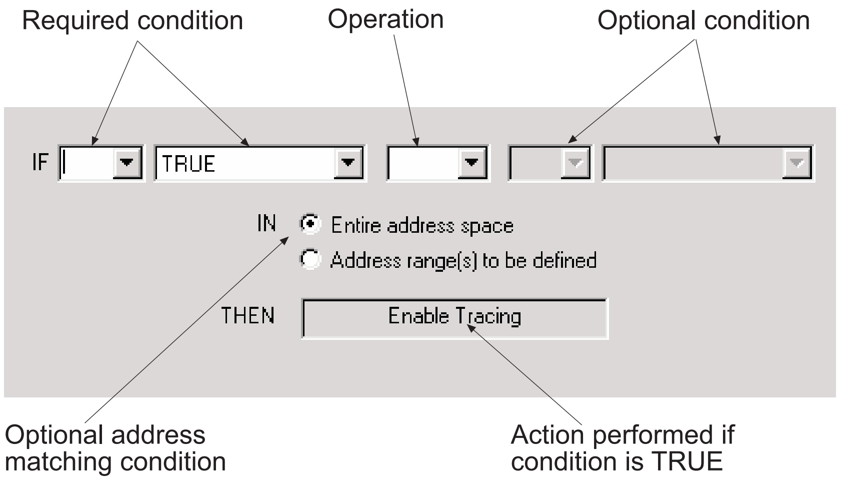

Required condition
Operation
Optional condition
IF
TRUE
IN
Entire address space
Address range(s) to be defined
THEN
Enable Tracing
Optional address matching condition
Action performed if condition is TRUE

Figure 5-1 Structure of wizard condition selection

These conditions, also called resources, form an expression that the ETM evaluates during the capture. The expression is also summarized in the **Rules** list of the **Configure Capture** dialog box. The conditions available are shown in drop-down lists and vary with the action being configured. An example showing the conditions available to the trigger action is shown in Figure 5-2.

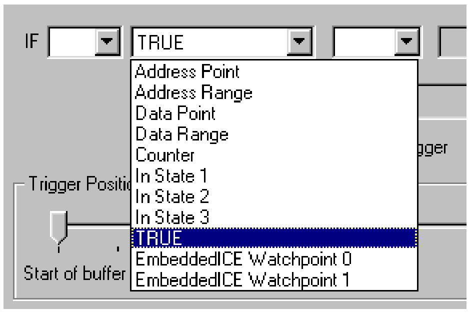


Figure 5-2 Conditions available to the Trigger action

Some conditions require more information than others (for example, the counter condition requires a count value) so when you define the expression, the wizard prompts for the additional information it requires in the following order:

1. The required condition.
2. The optional condition.
3. The address-matching condition.

The following are examples of rules you might use and how they can be expressed within TDT:

- You have a program that is behaving unpredictably, and you just want as complete an execution trace as possible.

  You must enable tracing, and set the trigger to capture all you can get. Triggering at **Start of buffer** for the event *address is _entry* ensures the execution trace fills the trace buffer with data after the program starts. The rule assumes the program entry point is called `_entry`.

  ```
  IF Address = %_entry THEN Trigger at Start of Buffer

  IF TRUE THEN Enable Tracing
  ```

- A program is failing after the 240th iteration of a loop, and real time constraints prevent the use of counted breakpoints.

  You must set up the ETM to trigger (that is, define a location around which the trace buffer is filled) when an address is reached for the 240th time. This requires a counter:

  ```
  IF Address = <address> THEN Decrement Counter

    IF Counted from = 240 THEN Trigger at Middle of Buffer
  ```

Additional examples are presented in Chapter 8 *Examples*.

5.2 The Configure capture dialog box

You can use this dialog box to select actions, such as enabling tracing or setting a trigger point, and to define the rules for each action. To access this dialog box, select **Configure capture** from the **Trace** menu.

The **Configure capture** dialog box is shown in Figure 5-3.

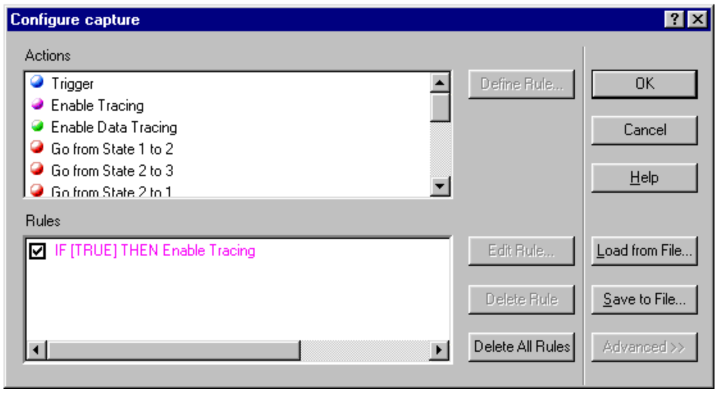


Figure 5-3 Configure capture dialog box

The areas of the dialog box are as follows:

- The **Actions** list contains each of the possible actions you can select to occur when the conditions you specify are met (see *Configure capture actions* on page 5-7 and *Defining rules* on page 5-15).
- The **Rules** list displays an abbreviated form of the rules you have already defined (see *Defining rules* on page 5-15 and *Using operators and negating conditions* on page 5-40). You can use the checkbox beside each rule to disable a rule and then to re-enable it again.

  The rules are color-coded to correlate to the specified action. Indenting is used to show conditions that use a counter (see *Counter* on page 5-33).
- The center panel contains the buttons used to manipulate rules:

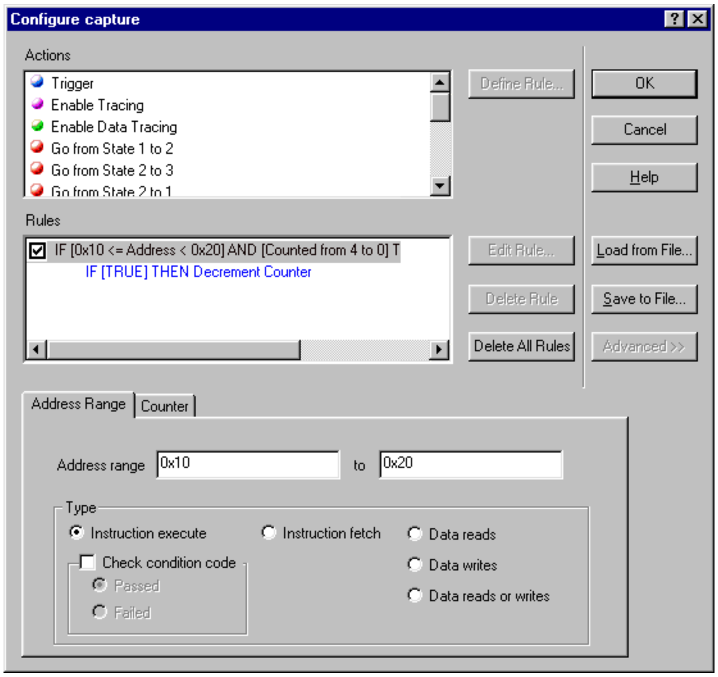
  Define Rule... This is enabled when you select one of the items in the **Actions** panel. Selecting it displays the rule creation wizard. An abbreviated form of the rule is shown in the **Rules** list.

  Edit Rule... This is enabled when you select one of the rules in the **Rules** panel. Clicking on this button displays the configuration wizard so that you can change the rule.

Delete Rule	This is enabled when you select one of the rules in the **Rules** panel. Clicking on this button deletes the rule from the list of active rules.
Delete All Rules	Clicking on this button deletes all the rules you have defined. You are prompted to confirm this.

- The right hand side panel contains standard **OK**, **Cancel**, and **Help** buttons, and the following:

Load from File	Click on this button to load a ruleset and the ETM configuration from a previously saved *Trace Configuration File* (TCF). The loaded rules replace the current ETM configuration and any existing rules.
Save to File	Click on this button to save the current ruleset and ETM configuration to a TCF.
Advanced>>	Click on this button to expand the dialog to show more details, as shown in Figure 5-4. The exact details shown in the extended part of the dialog box vary between different rules.



IF [0x10 <= Address < 0x20] AND [Counted from 4 to 0] T
IF [TRUE] THEN Decrement Counter

Address range 0x10 to 0x20

Figure 5-4 Configure capture dialog box showing Advanced section

5.3 Configure capture actions

This section describes each action you can select in the **Configure capture** dialog box. In *Rule definition example* on page 5-44, multiple actions (and rules) are combined to demonstrate a wider, more practical application of TDT functionality.

The possible actions are discussed in the following sections:

- *Trigger*
- *Enable Tracing* on page 5-9
- *Enable Data Tracing* on page 5-9
- *State changes* on page 5-11
- *External Out (1-4)* on page 5-12
- *Enable Overflow Stalling* on page 5-13.

To define a rule, you must select an action in the **Configure capture** dialog box and click **Define Rule**, or double-click the action. A wizard then guides you through the process of defining the rule(s) that must be satisfied for the action to occur.

5.3.1 Trigger

The trace capture hardware ensures that any trigger that occurs is kept within the trace buffer memory. If a trigger occurs, tracing stops as soon as the hardware buffer is full, although processor execution might continue. If you select this action, the **Resource selection** dialog box contains **Trigger control** options that allow you to configure where the trigger point appears in the trace data captured by the hardware (see Figure 5-5 on page 5-8).

The trigger point has a cycle number of zero (`0`), as shown in the **Index** column of the Trace window (see *Trace window output* on page 6-2). Cycles before the trigger point have negative cycle numbers. Cycles occurring after the trigger point have positive cycle numbers.

If you do not select a trigger, trace data is collected until either the program stops at a breakpoint, or you stop tracing manually by selecting **Stop** from the **Trace** menu. When tracing stops, up to one complete buffer-load of data is available, ending at the point that tracing stopped.

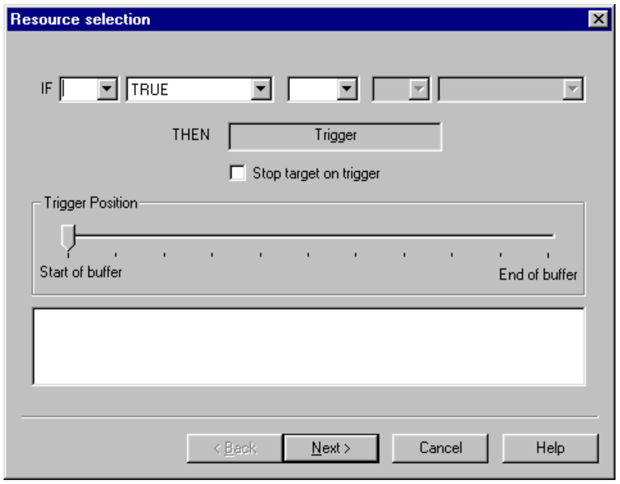


Figure 5-5 Resource selection for Trigger

Select **Stop target on trigger** if you want the ETM to halt the target when the trigger is hit.

Note

- This option only works if the **DBGREQ** signal from the ETM is connected to the **DBGREQ** signal of the ARM CPU. Refer to your hardware documentation for more information.
- Because of synchronization delays in the hardware, the processor might stop a few instructions after the trigger is hit. This delay varies depending on the trigger condition you select, and the detailed design of your target.

The **Trigger Position** control allows you to configure where the trigger event appears in the trace buffer. Setting it to the **Start of buffer** position causes the ETM to record only what happens after the trigger rule becomes true. Setting the control towards to **End of buffer** position causes the ETM to record more of the events before the trigger rule becomes true.

When decoding trace data, TDT must find a synchronization point that can be up to 1000 trace events into the buffer. This might mean that the oldest information in the buffer cannot be decoded. If you set the trigger position to Start of buffer, the instructions at and just after the trigger position might not be decoded and so might not appear in the Trace window.

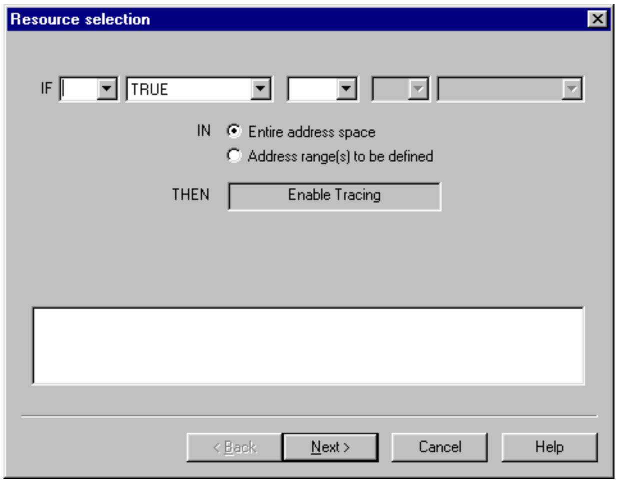
If the ETM is triggered more than once (for example, because an enable tracing rule becomes TRUE, then FALSE, then TRUE again) the trace buffer only contains events around the first FALSE to TRUE transition.

5.3.2 Enable Tracing

The **Enable Tracing** action enables the generation of trace information as shown in Figure 5-6. Tracing starts when the rule you define with this action evaluates to TRUE, and stops when the rule evaluates to FALSE. Using a rule, you can restrict this action to a range of addresses (see *Address range* on page 5-23).

Note

If you do not include an **Enable Tracing** action in a ruleset, no trace information is transmitted to TDT. You might do this if you are using TDT as an advanced breakpoint unit, or to generate external outputs.


Resource selection

IF TRUE

IN [✓ Entire address space] Address range(s) to be defined

THEN Enable Tracing

Figure 5-6 Resource selection for Enable Tracing

5.3.3 Enable Data Tracing

This action controls whether additional trace information is generated for data transfer instructions.

You configure the type of trace information that is retrieved by using the **Data tracing** options that appear in the **Resource selection** dialog box when you select this action, as shown in Figure 5-7 on page 5-10.

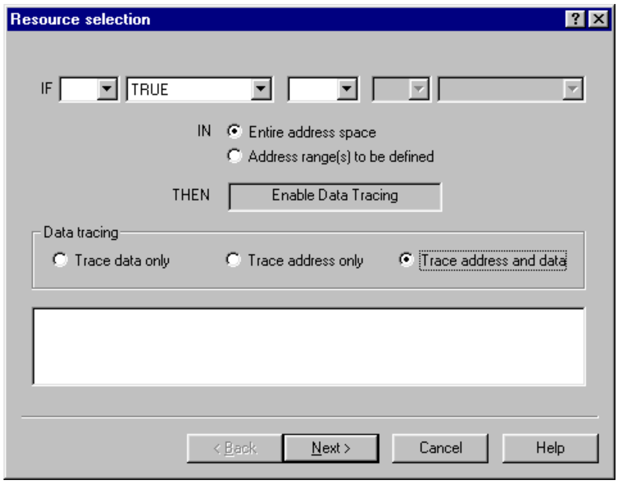


Figure 5-7 Resource selection for Enable Data Tracing

The **Data tracing** options specify the type of information the ETM traces for data transfer instructions. The options are:

Trace data only

Use this option when you want information about the value(s) transferred, but not the data transfer address.

Trace address only

Use this option when you want the data transfer address, but no information about the value(s) transferred.

Trace address and data

Use this option when you require both the data transfer address and information about the value(s) transferred.

Note

- For multiple transfer instructions, such as *Load Multiple* (LDM) and *Store Multiple* (STM), the address only appears for the first cycle. For details on LDM and STM, see the *Load and Store Multiple Instructions* section in the *ARM Architecture Reference Manual*.
- The ETM can only trace the data associated with instructions it is tracing. Because of this, data transfers are traced only when both **Enable Tracing** and **Enable Data Tracing** are TRUE for that instruction.

If you select either this action or the **Enable Tracing** action, you have the option to use address ranges in the configuration of your rule (see *Defining address ranges and ASIC memory maps* on page 5-37).

5.3.4 State changes

The state machine enables you to link rules together to form a sequence. For example, you can create a rule that only matches if execution at address A is followed by execution at address B.

These actions are only available if a state machine is present in your ETM hardware. If present, the state machine has three states, as shown in Figure 5-8.

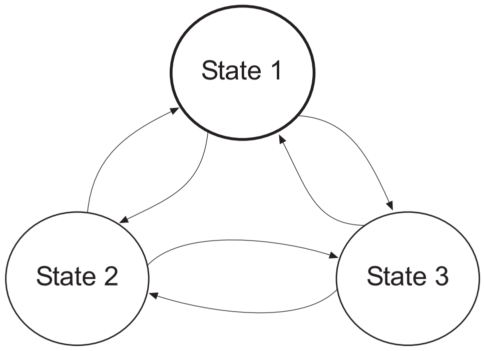


Figure 5-8 Changes of state

Initially, the state machine is always in State 1. There are six state-change actions, enabling you to change between any given pair of states:

- Go from State 1 to 2
- Go from State 1 to 3
- Go from State 2 to 1
- Go from State 2 to 3
- Go from State 3 to 1
- Go from State 3 to 2.

The **Resource selection** dialog box when you select this action is shown in Figure 5-9 on page 5-12.

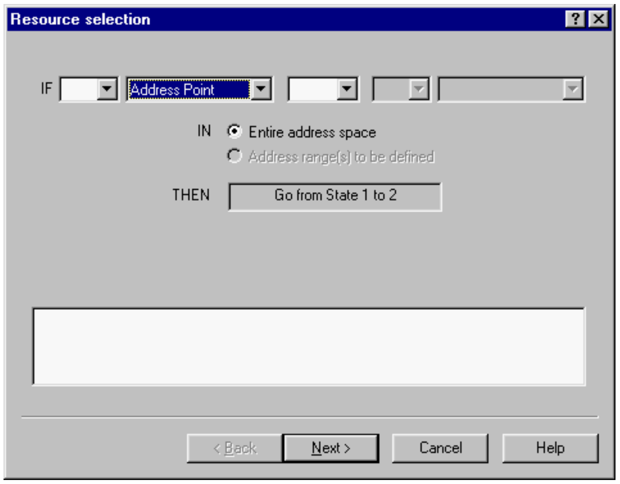


Figure 5-9 Resource selection for state changing

Note

- We recommend that you do not use the condition `IF TRUE`, because the rule executes as soon as the state machine enters the starting state.
- If you have defined rules for more than one state change from a particular state, and they are evaluated to true simultaneously, none of the desired state changes occur.
- The state machine is reset to State 1 whenever any of the following occurs:
  - tracing starts or restarts
  - the target stops at a breakpoint (unless you are using RealMonitor)
  - an image is loaded or reloaded (except when you use **Load symbols only…**).

See *States and sequencing* on page 5-35 for details on selecting states as a resource.

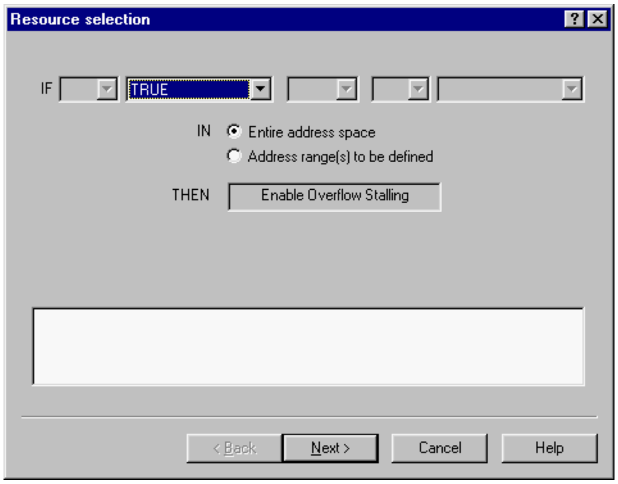
5.3.5 External Out (1-4)

These actions control single-bit output signals from the ETM. Up to four signals are available. The ASIC manufacturer determines the availability and usage of these output signals. Refer to your ASIC documentation for details.

5.3.6 Enable Overflow Stalling

This action controls whether processor stalling can take place when the FIFO buffer is full (when it exceeds the highwater value), and is shown in Figure 5-10. When the rule you define with this action is satisfied, and the FIFO buffer reaches its highwater value, the processor stalls until the FIFO buffer is cleared.

For this action to work, you must select the **Stall processor to prevent overflow** option in the Configure ETM dialog box (see the information on processor stalling in *The Configure ETM dialog box* on page 4-2).



IF TRUE IN [✓ Entire address space] THEN Enable Overflow Stalling

Figure 5-10 Resource selection for Enable Overflow Stalling

Note

The ASIC manufacturer determines the implementation of processor stalling.

You must use an address range to define the rules for this action. You must choose to enable overflow stalling over one of the following

- The complete target address range. Select the IN option **Entire address space** and click **Next>**. The returned trace is complete, but the processor might be stalled at a critical time.
- Specific parts of the address range. Select the IN option **Address range(s) to be defined** and click **Next>**. The trace might not be complete, but you can ensure that critical regions of code do not cause the processor to stall.

If you do not define an Enable Overflow Stalling rule, the effect is the same as defining a rule IN **Entire address space**.

See *Defining address ranges and ASIC memory maps* on page 5-37 for details on defining an address range.

5.4 Defining rules

This section describes how to set the rules which must evaluate to TRUE for a selected action to occur (see *Configure capture actions* on page 5-7). The main component of a rule is a resource such as an address range. A rule is a set of logic you create that can evaluate to either TRUE or FALSE, such as access to a given address range. Each rule you define can involve either:

- one condition
- two conditions joined by a logical operator.

You also have the option, for certain actions, to further limit the conditions by which a rule is evaluated by using an address range (see *Defining address ranges and ASIC memory maps* on page 5-37).

You can use the resource configuration wizard to help define a rule:

1. Double-click the desired action in the Configure capture dialog box (see *Configure capture actions* on page 5-7). The Resource selection dialog box is displayed.
2. Select a resource from the Resource drop-down list.

   To construct a complex rule, you can then select a logical operator, followed by a second resource (see *Using operators and negating conditions* on page 5-40 for details).

   Depending on the action you have selected, you might also have the option to select the IN option **Address range(s) to be defined** so that you can limit the action to particular address regions (see *Defining address ranges and ASIC memory maps* on page 5-37).
3. Click **Next>**. The next screen, depending on the resource or resources selected, is one of:
   - The Rule definition complete dialog box, indicating that no further details have to be provided. In this case, verify the information and click **Finish** to accept the rule and return to the Configure capture dialog box. Alternatively, you can click **<Back** to return to the previous screen, or click **Cancel** to return to the Configure capture dialog box without accepting the rule.
   - A resource-dependent, rule-definition dialog box.
   - The Define address range(s) dialog box. See *Defining address ranges and ASIC memory maps* on page 5-37 for details on how to use this dialog box.

4. Complete the required details for the resource (details for completing each resource are provided throughout this section). Click **Next>**. The next screen is one of the following:

   - The Rule definition complete dialog box, indicating that no further details have to be provided. In this case, verify the information and click **Finish** to accept the rule and return to the Configure capture dialog box. Alternatively, you can click **<Back** to return to the previous screen, or click **Cancel** to return to the Configure capture dialog box without accepting the rule.
   - A second rule-definition dialog box (resource-dependent), if completing an AND or OR condition. In this case:
     1. Complete the required details for the second resource.
     2. Click **Next>**. The Rule definition complete dialog box is displayed. Verify the information and click **Finish** to accept the rule and return to the Configure capture dialog box. Alternatively, you can click **<Back** to return to the previous screen, or **Cancel** to return to the Configure capture dialog box without accepting the rule.
   - A new Resource selection dialog box, if you have specified a counter resource and completed the counter details (see *Counter* on page 5-33). In this case:
     1. Select a resource, or resources, and fully define a rule which, when evaluated to true, increments the counter. Click **Next>**. If you enabled the counter reset option when defining the counter, an additional Resource selection dialog box is displayed, prompting you to define a rule which, when evaluated to true, resets the counter. Otherwise, the Rule definition complete dialog box is displayed.
     2. Verify the information and click **Finish** to accept the rule and return to the Configure capture dialog box. Alternatively, you can click **<Back** to return to the previous screen, or **Cancel** to return to the Configure capture dialog box without accepting the rule.
   - The Define address range(s) definition dialog box. See *Defining address ranges and ASIC memory maps* on page 5-37 for details on how to use this dialog box.

Note

For a description of the three areas of the Configure capture dialog box, which is displayed when a rule has been completely defined, see *The Configure capture dialog box* on page 5-5.

5.5 Defining resources

The ETM provides a wide range of resource types. This section describes each of the resources found in the Resource selection and Define address range(s) dialog boxes. The following sections describe the general behavior:

- *Resource conflicts* on page 5-18
- *Accessing the rule dialog boxes* on page 5-18
- *Entering Addresses* on page 5-19.

The resource dialog boxes are described in the following sections:

- *Address points* on page 5-20
- *Address range* on page 5-23
- *ASIC memory maps* on page 5-26
- *Data points* on page 5-27
- *Data range* on page 5-29
- *Counter* on page 5-33.
- *States and sequencing* on page 5-35, which describes:
  - In State 1
  - In State 2
  - In State 3.
- *Start and stop points* on page 5-31
- *External input (1-4)* on page 5-35
- *TRUE* on page 5-35
- *Watchpoint* on page 5-36.

Note

The resources that are available to you depend on the specification of the target device. It is recommended you refer to your hardware manual to determine which resources are available on your system. Table 1-1 on page 1-7 shows some of the possibilities.

TDT automatically detects many of the capabilities of your ETM, and resource configuration only shows those resources that are available.

5.5.1 Resource conflicts

When you define a new rule, the resources already used in other rules are discounted and the complete set of resources becomes available to you. This prevents TDT from running out of resources while you are in the process of defining a rule. If your new rule creates a problem with resources, TDT displays one of two error messages:

- If you define a rule that uses more resources than are actually available, TDT does not display an error message until you have finished defining the rule. When you try to enable the rule, TDT displays the following error message: `There are not enough ETM resources available to enable this rule.`
- If the new rule that you have defined is using resources that are currently being used by other existing rules, TDT displays the following error message: `Cannot enable this rule. All of the` *`<resource name>`* `resources required by this rule have already been used.  Check the ETM State for a list of available resources.`

To enable the new rule, you must first disable or delete any rules that are using the conflicting resources. See *Enabling, disabling, and deleting rules* on page 5-49 for information on how to disable or delete a rule.

5.5.2 Accessing the rule dialog boxes

TDT uses the features, or resources, provided by the ETM hardware to implement the rules that enable you to filter trace information. The ETM resources used by TDT rules implement expressions that are continuously evaluated to TRUE or FALSE.

The Configure Capture dialog box configures one or more rules. Actions occur when an expression containing one or more resources is TRUE. For example, Trigger is an action that defines a location within the trace buffer around which the trace data is stored. An address match resource can be linked to a trigger action with a rule, so that the location for a trigger is defined when the processor references a specific address.

There are two ways to access the resource selection dialog boxes:

- By selecting a resource from the resource drop-down list in a rule dialog box.

  Note

  Not all of the possible resources can be accessed from the resource drop-down list because:

  — start and stop points are never available

  — some actions cannot use some resource types.

- By selecting an item from the Define address range(s) dialog box.

Only data and address points and ranges, trace start and stop points, and ASIC memory maps, can be accessed from the Define address range(s) dialog box.

You can only access the Define address range(s) dialog box for the Enable Tracing, Enable Data Tracing, and Enable Overflow Stalling actions. These actions include a radio-button in the action dialog box, IN **Address range(s) to be defined**, that you select to access the Define address range(s) dialog box when you click **Next>**.

5.5.3 Entering Addresses

When you are asked to enter an address in a field you can use:

- any of the following forms of low-level address:
  - hexadecimal, for example:
    `0x8248` or `0x008248`
  - decimal, for example:
    `32768`
  - the low-level address of a function name, for example:
    `@Func_2`
  - hexadecimal address or low-level symbol, plus or minus an offset, for example:
    `@Func_2 + 0x20`
- the predefined low-level debugging symbols available in AXD, for example:
  `#r1`
- one of the following forms of high-level address:
  - function name, for example:
    `%Func_2`
  - function name and an executable line number, for example:
    `%Func_2:164`
  - the value of a global variable at the time the rule is entered, for example:
    `Int_Glob`
  - the address of a global variable, for example:
    `&Int_Glob`
  - the address of a member of a global array, for example:
    `&Arr_1_Glob[10]`
- for the special case of the **to** field in an address range, you can enter a function name, a colon, and the special symbol `$END` to signify the address just beyond the end of the function, for example:

```
%Func_2:$END
```

Note

You can use high level symbols to reference local variables that are in scope at the time the rule is entered into the Configure Capture wizard, although to do this the target must be stopped. Also, because local variables have limited scope, can be moved between registers and memory as the function executes, and, especially in optimized code, might not exist as distinct entities, you must take care to ensure that you are tracing what you expect.

TDT cannot adapt the rules you have defined to follow local variable movement, and does not automatically enable or disable rules as they enter or leave scope. If you have a state machine in the ETM you could use it to monitor the scope of a variable and enable another rule as required.

Refer to the *ADS Debuggers Guide*:

- for more details on low-level symbols, including the difference between low and high level addresses, see the *Low-level debugging* section
- for more details on the syntax used to define addresses, see the *Program locations* and *Low-level symbols* sections
- for more details on compound expressions, for example `@Func_2 + 0x20`, see the *AXD Facilities* section.

5.5.4 Address points

Address points are comparators that match with one pre-programmed address. Because an address point comparator becomes false as soon as the processor outputs another address, they must be linked with other ETM resources if you require an action to continue to occur after the program accesses a particular address. For example, you might use an address match to cause a state transition. The state transition records the fact that a condition became TRUE, in this case, an address was matched by an address point.

In addition to the address to match with, on ETM hardware implementing ETM version 1.2 you can define the instruction execution condition. Examples of using an address point in an action include:

- in a Trigger action, to define the trigger point
- in an Enable Tracing action, to produce a low bandwidth execution summary (you get a single instruction traced for each point you define).

Using an address point to create an execution summary is useful when all you need to know is whether a function is executed. Using several such points is a simple way of displaying the relative ordering of function calls. It has the benefit that even a small trace capture buffer can store a very large number of activations, so you can record execution history over a long period.

——— **Note** ———

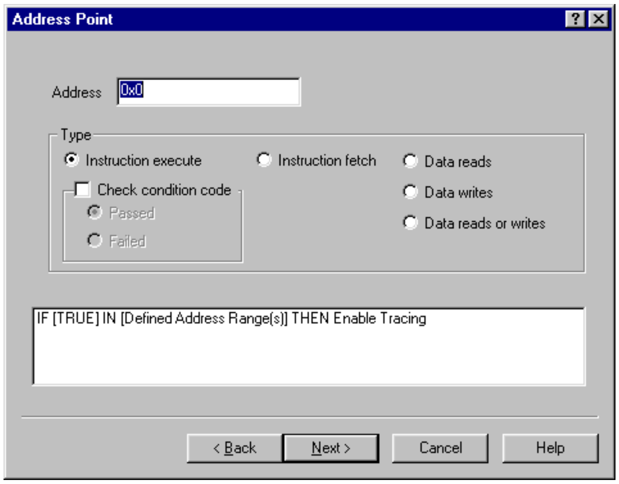
If you are using ETM hardware implementing ETM version 1.2 or later, you can also use the Start Address Point or Stop Address Point actions to simplify this process.

To display an address point dialog box for the Enable Tracing, Enable Data Tracing, or Enable Overflow Stalling actions:

1. Select **Address range(s) to be defined** in the Resource Selection dialog box.
2. In the Define address range(s) dialog box, select **Included Points** or **Excluded Points** from the Defined range(s) list.
3. Click **Add address point**.
4. Click **Next>**. The Include or Exclude Address Point dialog box is displayed, as shown in Figure 5-11 on page 5-22.

To display an address point dialog box for other actions:

1. Select **Address Point** in the Resource drop-down list of the Resource Selection dialog box.
2. Select **NOT** in the drop-down list to the left of the Resource drop-down list to define an excluded address point.
3. Click **Next>**. The Address Point dialog box is displayed. This looks and behaves like the Include or Exclude Address Point dialog box (Figure 5-11 on page 5-22).



Address: 0x0

Type: [✓ Instruction execute] Instruction fetch Data reads Data writes Data reads or writes

Check condition code (unchecked): [✓ Passed] Failed

IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 5-11 Include Address Point dialog box

To define an address match point

1. Provide an address to compare with in the **Address** field as described in *Entering Addresses* on page 5-19.

2. Select one of the following address comparison types:

   Instruction execute

   The address of each instruction that reaches the Execute stage of the pipeline is compared against the address you specify (it might not actually be executed if its condition code evaluates to false).

   To require a match only for instructions that are, or are not, executed, select **Check condition code** and click on one of:

Passed	Match only instructions that executed.
Failed	Match only instructions that did not execute.

   ——— Note ———

   If you are using ETM hardware implementing ETM version 1.0 or 1.1, **Check condition code**, **Passed** and **Failed** are all disabled, because the hardware does not support this feature.

   ———————————

   Instruction fetch

   The address of each instruction fetched is compared against the address you specify.

——— **Note** ———

Whether you select **Instruction execute** or **Instruction fetch**, the trace stream reflects processor activity at the Execute stage of the pipeline.

———

Data reads

The address comparison is made against the source address of a data transfer cycle. The address compared is the address of the data value read, and not the address of the load instruction.

Data writes

The address specified corresponds to the destination address of a data write instruction. The data write address is not the same as the address of the store instruction.

Data reads or writes

The address comparison is made against the source and destination address of load and store instructions.

——— **Note** ———

Selecting any of the three **Data** options above has the same effect as specifying a data point with a filter indicating that no bits should be tested.

———

5.5.5 Address range

To specify a range of addresses in a rule, use an address range, as shown in Figure 5-12 on page 5-24. Address ranges are used when you want something to happen in a particular region of memory. For example, you can Enable Tracing for some part of a function, or Enable Data Tracing for the memory occupied by a global array.

——— **Note** ———

When tracing multiple transfer instructions, such as *Load Multiple (LDM)* and *Store Multiple (STM)*, only the lowest address accessed is checked by the ETM. This means that even if such an instruction makes an access within a given address range, it might not be traced.

———

Address ranges are defined using a low address that is in the range and a high address that is beyond the range.

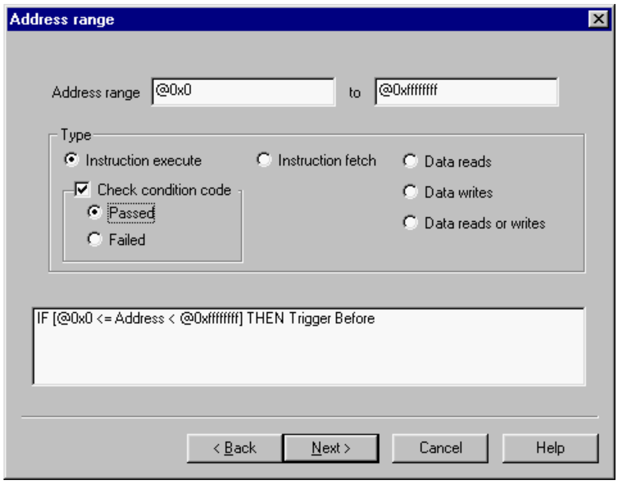
To display an address range dialog box for the Enable Tracing, Enable Data Tracing, or Enable Overflow Stalling actions:

1. Select **Address range(s) to be defined** in the Resource Selection dialog box.

2. In the Define address range(s) dialog box, select **Included Ranges** or **Excluded Ranges** from the Defined range(s) list.
3. Click **Add address range**.
4. Click **Next>**. The Include Address Range or Exclude Address Range dialog box is displayed. See Figure 5-12.

To display an address range dialog box for other actions:

1. Select **Address range** in the Resource drop-down list of the Resource Selection dialog box.
2. Select **NOT** in the drop-down list to the left of the Resource drop-down list to select an excluded address range.
3. Click **Next>**. The Address Range dialog box is displayed (Figure 5-12). This looks and behaves like the Include or Exclude Address Range dialog box.



Address range: @0x0 to @0xffffffff

Type: [✓ Instruction execute] [✓ Check condition code] [✓ Passed]

IF [@0x0 <= Address < @0xffffffff] THEN Trigger Before

Figure 5-12 Address range dialog box

To define an address range:

1. Provide a start and end address range in the **Address range** fields using the method described in *Entering Addresses* on page 5-19.
2. Select one of the following address comparison types:

   Instruction execute

   The address of each instruction that reaches the Execute stage of the pipeline is compared against the address range you specify (it might not actually be executed if its condition code evaluates to false).

To require a match only for instructions that are, or are not, executed, select **Check condition code** and click on one of:

Passed	Match only instructions that execute.
Failed	Match only instructions that did not execute.

——— **Note** ———

If you are using ETM hardware implementing ETM version 1.0 or 1.1, **Check condition code**, **Passed** and **Failed** are all disabled, because the hardware does not support this feature.

———————

Instruction fetch

The address of each instruction fetched is compared against the address range you specify.

——— **Note** ———

Whether you select **Instruction execute** or **Instruction fetch**, the trace stream reflects processor activity at the Execute stage of the pipeline.

———————

Data reads

The address comparison is made against the source address of a data transfer cycle. The address compared is the address of the data value read, and not the address of the load instruction.

Data writes

The address specified corresponds to the destination address of a data write instruction. As in the case of a data read, a data write does not relate to the address of the store instruction.

Data reads or writes

The address comparison is made against the source and destination address of load and store instructions.

——— **Note** ———

Selecting any of the three **Data** options above has the same effect as specifying a data range with a mask indicating that no bits are to be tested.

———————

——— **Note** ———

You can use more than one address range in the same rule (see *Defining address ranges and ASIC memory maps* on page 5-37 for details).

———————

5.5.6 ASIC memory maps

These resources are only available when you have ASIC memory map decode hardware installed in your ASIC. Each implementation of ASIC memory map decode hardware will have been defined by the ASIC designer to return true when instructions are being fetched from a particular address range. For example Memory Map 6 might correspond to addresses in a ROM.

Note

- TDT cannot tell whether an ASIC memory map input is connected and so always shows the ASIC memory map inputs that are provided by the ETM hardware as being available. You must check in the device documentation whether inputs are connected and to which addresses they map.
- ASIC memory map inputs might be TRUE for several distinct address ranges if the hardware address decoder ignores some high order address bits.

To define an ASIC memory map for the Enable Tracing, Enable Data Tracing, or Enable Overflow Stalling actions:

1. Select **Address range(s) to be defined** in the Resource Selection dialog box.
2. In the Define address range(s) dialog box select the number of the ASIC memory map you require, using the up and down arrow buttons next to **Add memmap *n*** , where *n* is the number of the ASIC memory map.

   Note

   - You can only add each ASIC memory map once to any rule, but you can use the same ASIC memory map in different rules. The **Add memmap *n*** button is grayed out for any map that you have added to the rule being defined.
   - The number of ASIC memory map resources available is dependent on your ETM configuration. If no ASIC memory maps are available, the selection arrows are disabled.

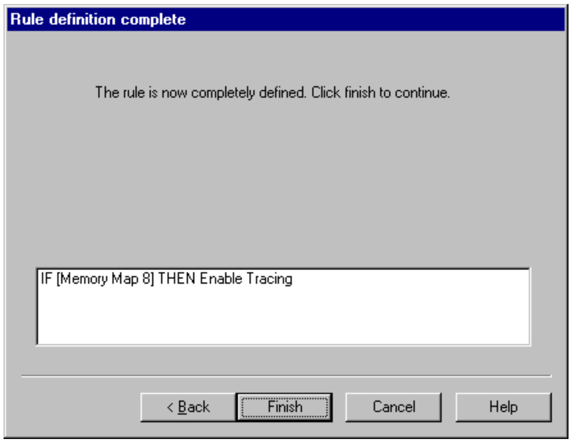
3. Click **Add memmap**.
4. Click **Next>**. The Rule definition complete dialog box is displayed straight away, because no more information is required. See Figure 5-13 on page 5-27.

To define an ASIC memory map for other actions:

1. Select **Memory map *n*** in the **Resource** drop-down list of the Resource Selection dialog box, where *n* is the number of the ASIC memory map to use.

2. Optionally, select **NOT** in the drop-down list to the left of the Resource drop-down list to select an excluded ASIC memory map.

3. Click **Next>**. The Rule definition complete dialog box is displayed straight away, because no more information is required. (Figure 5-13).


Rule definition complete

The rule is now completely defined. Click finish to continue.

IF [Memory Map 8] THEN Enable Tracing

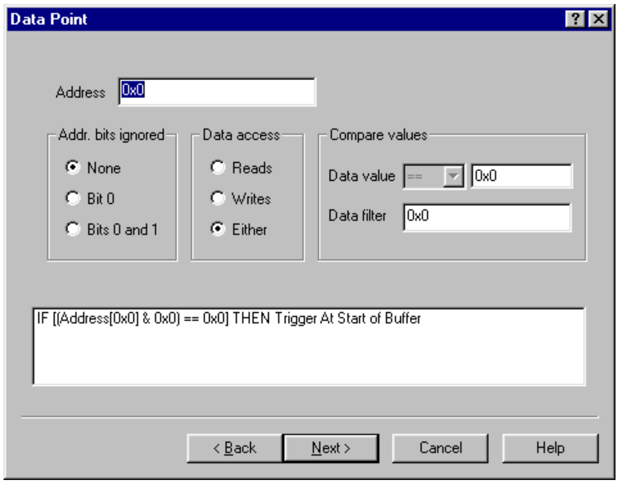
Figure 5-13 Defining an ASIC Memory map resource

5.5.7 Data points

A data point is used to test for a data transfer of a specific value, either to memory or from memory. The address you specify is the source or destination address of the data transfer. It does not specify the address of the load or store instruction.

To define a data point:

1. In the Resource selection dialog box, select **Data Range** from the Resource drop-down list. Click **Next**. The Data point dialog box is displayed (Figure 5-14 on page 5-28).



Address: 0x0

Addr. bits ignored: [✓ None] Bit 0 Bits 0 and 1

Data access: Reads Writes [✓ Either]

Data value: == 0x0

Data filter: 0x0

IF [(Address[0x0] & 0x0) == 0x0] THEN Trigger At Start of Buffer

Figure 5-14 Data Point dialog box

2. Provide a data point address in the **Address** field using the method described in *Entering Addresses* on page 5-19.

3. Set the following fields:

Addr. bits ignored

Select one of the following options to further specify what is checked and ignored:

None Only that exact address is checked.

Bit 0 Both addresses in the halfword are checked. This must be used, for example, if you are interested in byte accesses to either byte of a halfword.

Bit 0 and 1

All addresses in the same word are checked. This must be used, for example, if you are interested in byte accesses to any byte of a word.

Data access

The direction of the data transfer can be restricted to either **Reads**, **Writes**, or accesses in **Either** direction.

Compare values

This section allows you to specify transfers of particular values. You specify the value to compare data accesses with in the **Data value** field. You can either match on the value itself, using the **==** control, or anything except the value, using the **NOT=** control.

Note

The **NOT=** control is not available if you are using ETM hardware implementing ETM version 1.0 or 1.1.

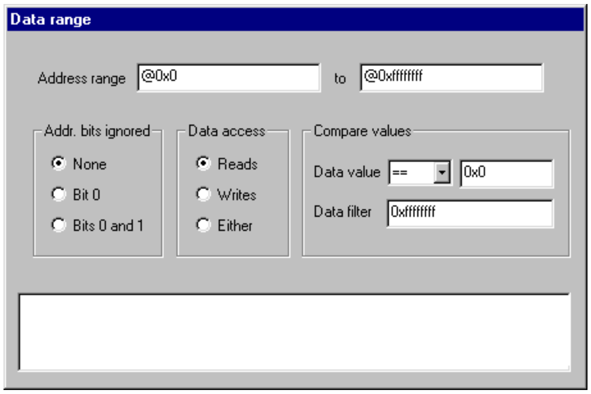
The **Data filter** allows you to specify individual bits to test when comparing values. Testing is performed on the following basis:

- a binary zero in the filter indicates that the bit is not compared
- a binary one in the filter indicates that the corresponding bit of the transfer is compared with the corresponding bit of the **Data value**.

4. Click **Next>**. If applicable, complete the second resource definition dialog box. Click **Finish** in the Rule definition complete dialog box.

5.5.8 Data range

You can use data ranges when you want to capture data accesses to a range of addresses. Most of the options you configure in the **Data range** dialog box, shown in Figure 5-15, are identical to those found in the **Data point** dialog box. When specifying a data range, the addresses you specify refer to the address of the data access, rather than the address of the instruction making the access.


Data range

Address range: @0x0 to @0xffffffff

Addr. bits ignored: [✓ None] Bit 0 Bits 0 and 1

Data access: [✓ Reads] Writes Either

Compare values

Data value: == 0x0

Data filter: 0xffffffff

Figure 5-15 Data range dialog box

To define a data range:

1. In the Resource selection dialog box, select **Data Range** from the Resource drop-down list. Click **Next>**. The Data range dialog box is displayed, as shown in Figure 5-15.

2. Provide a start and end data range in the **Address range** fields using the method described in *Entering Addresses* on page 5-19.

3. Set the following fields:

   Addr. bits ignored

   Select one of the following options to further specify what is checked and ignored:

   None Only that exact address is checked.

   Bit 0 Both addresses in the halfword are checked. This must be used, for example, if you are interested in byte accesses to either byte of a halfword.

   Bit 0 and 1

   All addresses in the same word are checked. This must be used, for example, if you are interested in byte accesses to any byte of a word.

   Data access

   The direction of the data transfer can be restricted to either **Reads**, **Writes**, or accesses in **Either** direction.

   Compare values

   These options allow you to specify transfers of particular values. You specify the value to compare data accesses with in the **Data value** field. You can either match on the value itself, using the **==** control, or anything except the value, using the **NOT=** control

   ——— **Note** ———

   The **NOT=** control is not available if you are using ETM hardware implementing ETM version 1.0 or 1.1.

   ————————

   The **Data filter** allows you to specify individual bits to test when comparing values. Testing is performed on the following basis:

   - a binary zero in the filter indicates that the bit is not tested
   - a binary one in the filter indicates that the corresponding bit of the transfer is compared with the corresponding bit of the **Data value**.

4. Click **Next>**. If applicable, complete the second resource definition dialog box. Click **Finish** in the Rule definition complete dialog box.

Note

- A data range resource evaluates to true when a data access that matches the condition(s) occurs. It remains true until a further data access that does not match the condition(s) occurs.
- If you specify an access type of **Reads** in your conditions, any write access sets the resource to false. If you specify **Writes**, any read access sets the resource to false.

5.5.9 Start and stop points

Start and stop points are used to switch on or switch off the Enable Tracing action. You do not have to match a stop point with every start point. For example, you can have several start points at each of the addresses that a function can be reached from, and a stop point within the function.

Using start and stop points, you can trace every instruction or data transfer that is executed by the target processor after an instruction at a particular address has been fetched, without having to specifically program the regions to trace as address ranges.

You can stop tracing after an instruction at a particular address has been fetched or after the value transferred to or from a particular memory location matches a given pattern.

Note

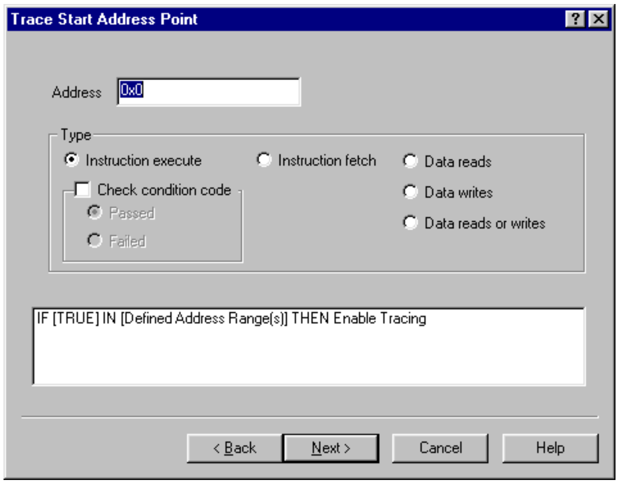
Start and stop points are not available if you are using ETM hardware implementing ETM version 1.0 or 1.1. Some uses of start and stop points can be simulated using the state machine. For details, see Chapter 8 *Examples*.

It is possible to create gaps in the trace generated using a start point by using another stop point and start point, or exclude address ranges, associated with the Enable Tracing rule. For example, if you require a trace of code running with active interrupts, but you require that the interrupt handler instructions do not appear in the trace, you must define an exclude range that includes the code of the interrupt handler.

There are two kinds of start and stop point:

Address Points

Start Address points and Stop Address points trigger when an instruction Fetch or Execute, or a data read or write, occurs at a specific address. The Start Address Point dialog box is shown in Figure 5-16 on page 5-32.



Address 0x0

Type: [✓ Instruction execute] Instruction fetch Data reads Data writes Data reads or writes

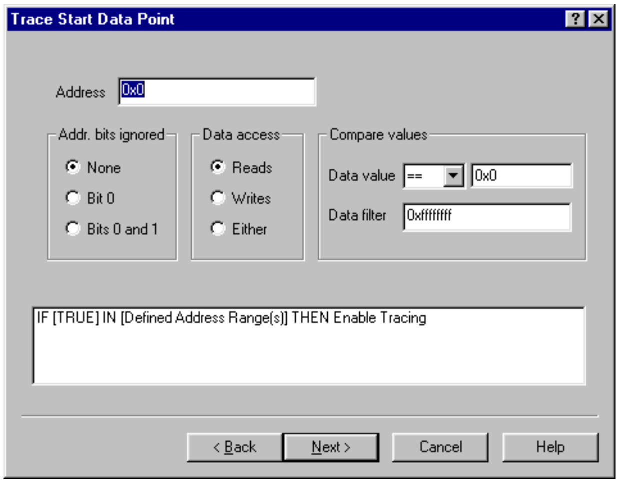
Check condition code: [✓ Passed] Failed

IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 5-16 Start Address Point dialog box

Other than the dialog box title, the Trace Stop Address Point dialog box looks the same as the Trace Start Address Point dialog box.

Data Points Start Data points and Stop Data points trigger when a data read or write occurs from or to a specific address, and the data value transferred matches a defined value in a specified way. The Trace Start Data Point dialog box is shown in Figure 5-17.



Address 0x0

Addr. bits ignored: [✓ None] Bit 0 Bits 0 and 1

Data access: [✓ Reads] Writes Either

Data value == 0x0

Data filter 0xffffffff

IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 5-17 Start Data Point dialog box

Other than the dialog box title, the Trace Stop Data Point dialog box looks the same as the Trace Start Data Point dialog box.

To define a start or stop point:

1. In the Defined Range(s) list dialog box, select **Trace Start Point** or **Trace Stop Point**.
2. To define an address point, click **Add Address Point**. To define a data point, click **Add Data Point**.
3. Click **Next**. The start or stop point dialog box is displayed.
4. Provide an address for the point in the **Address** field using the method described in *Entering Addresses* on page 5-19.
5. To define a start or stop address point, fill in the dialog box as described in *Address points* on page 5-20.

   To define a start or stop data point, fill in the dialog box as described in *Data points* on page 5-27.
6. Click **Next**. If applicable, complete the second resource definition dialog box. Click **Finish** in the Rule definition complete dialog box.

Note

For address and data points triggering on a data transfer the **Address** field contains the address of the data item, not the address of the instruction that caused the transfer.

5.5.10 Counter

You can use a counter resource to specify that an action should occur when a rule evaluates to true for a set number of instances.

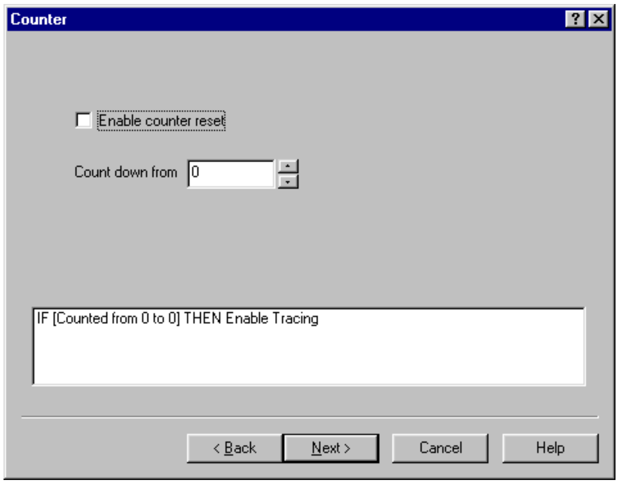
To fully define a counter rule:

1. Select an action such as **Trigger** from the Configure capture dialog box. This is the action that occurs when the counter reaches the value you select. The Resource selection dialog box is displayed.
2. Select **Counter** from the Resource drop-down list. Click **Next**. The **Counter** dialog box is displayed, as shown in Figure 5-18 on page 5-34.
3. Complete the counter details as follows:
   a. Select **Enable counter reset** if you want to restart the counter when a specific event occurs.

b. In the **Count down from** field, enter a value indicating the number of instances for which the rule must evaluate to true before the selected action occurs. This value is displayed in the ETM State window (see *View ETM State* on page 3-8).

4. Click **Next**. A Resource selection dialog box is displayed. Select a resource, or resources, and fully define a rule which, when evaluated to true, decrements the counter. Instructions for defining the rules are resource-dependent.

5. If you enabled the counter reset option, an additional Resource selection dialog box is displayed, prompting you to define a rule which, when evaluated to true, resets the counter. In the Configure capture dialog box, this rule appears indented, directly under the rule for decrementing the counter.



☐ Enable counter reset

Count down from 0

IF [Counted from 0 to 0] THEN Enable Tracing

Figure 5-18 Counter dialog box

Note

You can see the progress of your counter while execution is underway. To do this, select **View ETM State** from the **Trace** menu (see *View ETM State* on page 3-8). This window shows the value of the counter decreasing whenever it is activated. You can update the counter and state machine values at any time by clicking on the **Refresh** button.

Caution

Counters are reset to their initial value whenever any of the following occurs:

- a breakpoint is reached
- tracing is restarted

- an image is loaded or reloaded onto the target.

5.5.11 States and sequencing

These resources are only available if a state machine (see *State changes* on page 5-11) is present in your hardware. The states are predefined by your ASIC designer and vary according to your ETM configuration (see *State changes* on page 5-11 for more details).

To define a state resource:

1. In the Resource selection dialog box, select **In State 1**, **In State 2**, or **In State 3** from the Resource drop-down list. This causes the selected action to occur when the state machine is in the state you have selected.
2. Click **Next>**. The Rule definition complete dialog box is displayed.
3. Click **Finish**.

Note

- You can see the current state of your state machine while execution is underway. To do this, select **View ETM State** from the **Trace** menu (see *View ETM State* on page 3-8). You can update the counter and state machine values at any time by clicking on the **Refresh** button.
- The state machine is reset to State 1 whenever any of the following occurs:
  - tracing starts or restarts
  - the target stops at a breakpoint
  - an image is loaded or reloaded (except when you use the **Load symbols only…** option).

5.5.12 External input (1-4)

These resources are application-specific. Up to four **External Input** actions are available depending on your ETM configuration. The ASIC manufacturer determines the meaning of them. External inputs can be any combination of logic that evaluates to TRUE or FALSE.

5.5.13 TRUE

This is a special type of resource that requires no hardware support. The action always evaluates to TRUE when this resource is selected. This resource is always available when defining rules.

5.5.14 Watchpoint

The watchpoint resource(s) is application-specific (for example, EmbeddedICE Logic Watchpoints are specific to EmbeddedICE Logic). The ASIC manufacturer determines the meaning of them. Watchpoints can be any combination of logic that evaluates to TRUE or FALSE.

5.6 Defining address ranges and ASIC memory maps

You can use the address range dialog box to perform an action that is dependent on the processor accessing any of several address ranges and/or ASIC memory map locations. This interface is only available when defining rules for the **Enable Tracing**, **Enable Data Tracing**, or **Enable Overflow Stalling** action.

The Define address range(s) dialog box shown in Figure 5-19 enables you to combine several conditions in one rule. For example, you might want to enable tracing for several functions. A single rule using defined address ranges can contain each address range, or its inverse, that you want to include for evaluation.

Note

The following items in the Defined range(s) list are not available with ETM hardware implementing ETM version 1.0 or 1.1:

- for the Enable Tracing action, Included Points and Excluded Points
- Trace Start Points and Trace Stop Points.

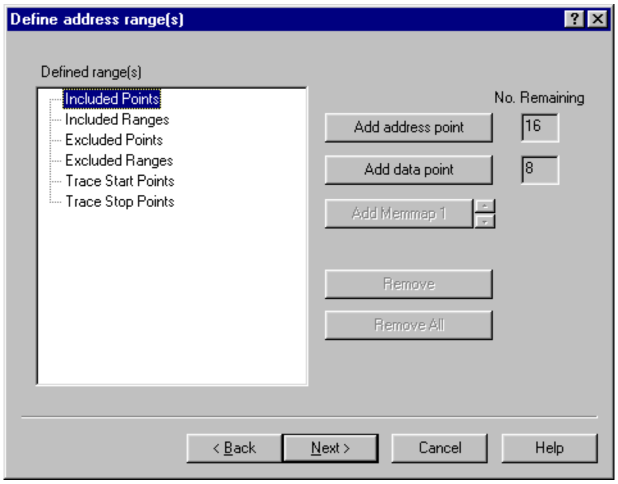


Figure 5-19 Define address range(s) interface example

To define address ranges:

1. Select the **Enable Tracing**, **Enable Data Tracing**, or **Enable Overflow Stalling** action in the Configure capture dialog box and define the rule conditions you require.
2. Select the **Address range(s) to be defined** radio button.

3. Click **Next>**. The dialog box that corresponds to the resource you have selected is displayed. If you have selected two resources in the Resource selection dialog box, the dialog box that correlates to the first selected resource is displayed.

4. Define the rule for the displayed resource. Click **Next>**. If you have a selected a second resource in the Resource selection dialog box, the dialog box corresponding to the second resource is displayed, and you must complete the second condition for the rule.

5. Click **Next>**. The Define address range(s) dialog box is displayed, as shown in Figure 5-19 on page 5-37. The number and type of resources available is dependent on your ETM configuration.

6. Select an included or excluded point, range, or ASIC memory map from the Defined range(s) list. Click on one of the **Add** buttons. (When you click on Next> you can define each of the items you have added.)

   Included or excluded ranges are defined as follows:

   - For items in the include range, the rule evaluates to true whenever the processor accesses memory within either the specified address range, or the selected predefined ASIC memory map area of execution. The addresses included in the range are:

     `address >= low_address && address < high_address`

   - An exclude range is the inverse of the include range. That is, the rule evaluates to true whenever the processor accesses either memory outside the address range, or areas of execution that are not contained within the predefined ASIC memory map area of execution. The addresses included in the range, and so excluded from the action, are:

     `address < low_address || address >= high_address`

   where:

`address`	is the memory address generated by the processor
`low_address`	is the value entered into the **from:** address text field in the dialog box
`high_address`	is the value entered into the **to:** address text field in the dialog box.

   ——— **Note** ———

   You cannot define both included points or ranges and excluded points or ranges for the same rule, other than for the **Enable Data Tracing** action.

7. Click **Next>**. The next dialog box that is displayed depends on what items you have placed in either list:
   - If you have placed one or more of the available address range resources in either list, the **Address range** dialog box is displayed. You must define each address range separately, using the same procedure for defining an individual address range (see *Address range* on page 5-23 for details). After you have finished defining each address range, the Rule definition complete dialog box is displayed.
   - If you have placed one or more ASIC memory map resources in either list, the Rule definition complete dialog box is displayed.
8. Click **Finish**. The Configure capture dialog box is displayed and your rule appears in the left column with any other rules you have defined.
9. If you no longer want to monitor, or exclude, a specific range of addresses or ASIC memory map, you must remove that item from the include or exclude list. To do this, you must modify the rule using the wizard (see *Using the wizard* on page 5-48), clicking **Next>** repeatedly until you arrive at the Define address range(s) dialog box. Select the item from the exclude or include list and click the appropriate left-arrow button to return it to the Range Resource List. That address range or ASIC memory map resource is now available for use in another rule.

Refer to Chapter 8 *Examples* for more information.

5.7 Using operators and negating conditions

When you select (double-click) an action from the Configure capture dialog box, the Resource selection dialog box is displayed, an example of which is shown in Figure 5-20.

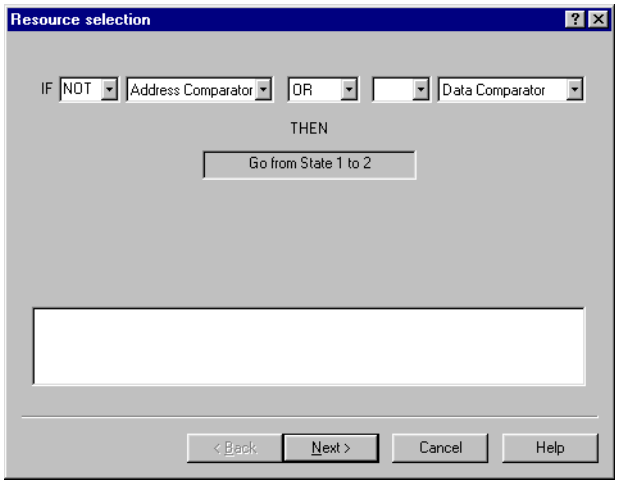


Figure 5-20 Resource selection dialog box

The rules described in *Defining rules* on page 5-15 typically involve a one-to-one, if-then logical construct. You can, however, use a logical operator to further specify a rule. You can select any of the following logical operators in a rule condition:

OR Where one of the rules, using either of the specified resources, must be met for the condition to be satisfied.

AND Where both of the rules, using both specified resources, must be met for the condition to be satisfied.

nothing (blank)
Where no operator is needed (only the first resource is defined).

To select a logical operator, use the drop-down list that appears between the two Resource drop-down lists. Figure 5-20 shows an example of selecting the **OR** condition. In this example, either the address point condition or the data point condition need to be met for the specified action (state transition) to occur.

To negate a condition, select **NOT** from the drop-down list directly to the left of the resource to be used.

5.8 File commands

This section describes the following file-command options available from the Configure capture dialog box:

- *Saving to a file*
- *Loading from a file*.

5.8.1 Saving to a file

To save the current set of rules, and the current ETM configuration, to a file:

1. Click **Save to File** in the Configure capture dialog box. The Save As dialog box is displayed.
2. Enter the desired filename, or select a previously existing file from the list or from another directory, and click **Save**. The default file extension is `.tcf`. If the specified filename already exists, you must confirm whether you want to overwrite the existing file.

Note

The file is saved in a format that TDT v1.0 cannot read.

5.8.2 Loading from a file

Note

Both the current set of rules and the current ETM configuration are overwritten when a configuration file is loaded.

To load an existing set of rules and ETM configuration:

1. Click **Load from File** in the Configure capture dialog box. A set of previously saved rules appears in a standard Open file dialog box (Figure 5-21 on page 5-42).

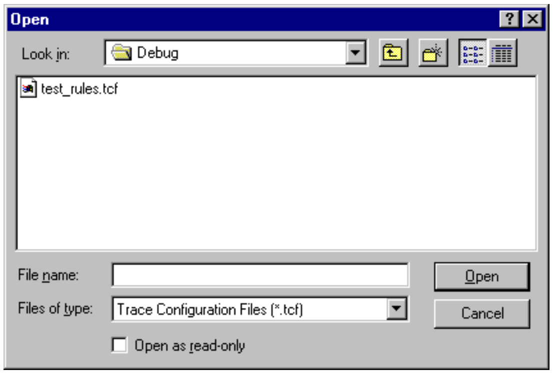


Figure 5-21 Loading from a file

2. Select the required file from the list, or select a file from another directory, and click **Open**. The set of rules from this loaded file appears in the Configure capture dialog box, overwriting any existing rules.

It is possible that rules are disabled as they are loaded because:

- symbols referenced in the rule definition are not known to the debugger
- resources required by the rule are not available.

There are two likely reasons for this:

- the debugger has no current image, or it has loaded a different image to that loaded when the rule was created
- the ETM resources available on the target differ from those available when the rule was created.

Loading a TDT 1.0 configuration file

If you attempt to load a `.tcf` file that was saved by TDT 1.0:

1. TDT prompts you with the message:

   ```
   This configuration file was generated by an older version of Trace Debug
   Tools and must be converted to the new format. Do you wish to convert the
   configuration file?
   ```

   Click on **Yes** to continue and convert the file, or **No** to abandon loading it.

2. The file is read and converted to the new format. If this fails, TDT abandons loading the file.

3. The original file is renamed by adding `.old` to the original name

4. A new file containing the converted configuration is written using the original name.

5.9 Rule definition example

The program `dhry.axf` in the `…\examples\dhry` directory is used here to demonstrate how to define rules. You must build `dhry` before trying this example. For details on building projects, refer to the ADS *CodeWarrior IDE Guide*. For the purpose of this example, assume that something unexpected is occurring just prior to the 50th run of the main application loop, and that the problem has resulted from code located somewhere on or near line 164 in `Func_2`.

You must first compile and load `dhry.axf` into the debugger and ensure that your ETM is configured appropriately (see Chapter 4 *Configuring the ETM*).

Note

After you load an image into AXD, you might need to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

To ensure that your trace output displays useful blocks of trace information, you must select **Configure capture** from the **Trace** menu, and set rules as follows:

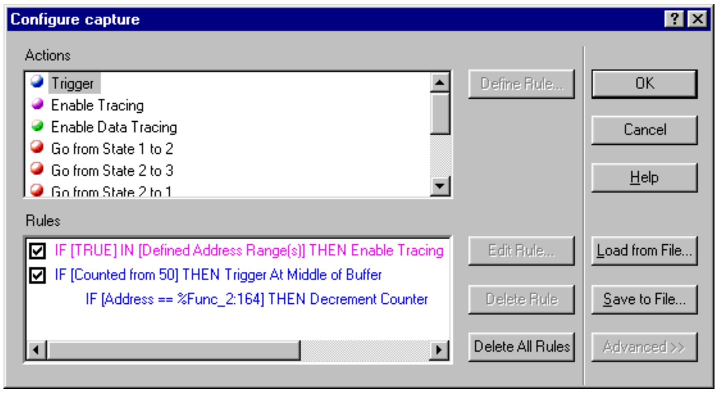
1. Select (double-click) **Enable Tracing** from the **Actions** menu. The Resource selection dialog box is displayed.
2. Ensure that TRUE is selected in the **Resource** drop-down list. Select **Address range(s) to be defined** and Click **Next**. The **Define address range(s)** dialog box is displayed.
3. Select **Included Ranges** in the tree control and click **Add Address Range**. Click **Next**. The **Include Address range** dialog box is displayed.
4. In the **Address range** fields, select the range `%Func_2` to `%Func_2:$END`. This ensures that trace information is returned only for `Func_2` instructions. Click **Next**. The **Rule definition complete** dialog box is displayed.
5. Click **Finish**. The **Configure capture** dialog box is displayed.
6. Select (double-click) **Trigger** from the **Actions** menu. The **Resource selection** dialog box is displayed.
7. Select **Counter** from the **Resource** drop-down list. Move the **Trigger Position** slider near to the centre of the track to ensure that the instruction for your selected line of code, line 164 of `Func_2`, appears near the middle of the captured buffer-load. Click **Next**. The **Counter** dialog box is displayed.

8. Enter `50` in the **Count from** field. This ensures that the program executes fifty times before the trigger is enabled. Click **Next**. The **Resource selection** dialog box is displayed.

9. Select **Address point** from the **Resource** drop-down list. Click **Next**. The **Address point** dialog box is displayed.

10. In the **Address** field, enter `%Func_2:164`, that is, the line of code around which you want to investigate. Click **Next**. The **Rule definition complete** dialog box is displayed.

11. Click **Finish**. The **Configure capture** dialog box is displayed (Figure 5-22).



IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing
IF [Counted from 50] THEN Trigger At Middle of Buffer
IF [Address == %Func_2:164] THEN Decrement Counter

Figure 5-22 Example of defined rules

These rules, described from the top downwards, produce the following sequence of events:

a. Enable tracing only for the area of code between the start of `Func_2` and the end of `Func_2`, that is, for the entire function.

b. Set the trigger to decrement when the counter is activated for the 50th time.

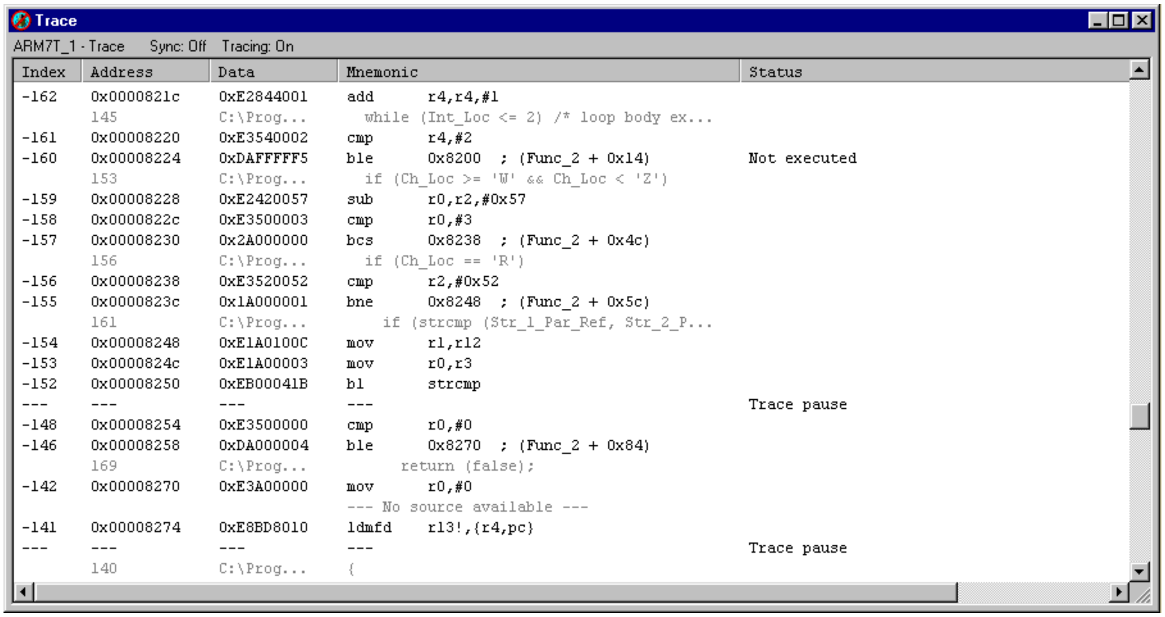
c. Decrement the counter every time the first command in line 164 in `Func_2` (`0x8254`) is executed.

12. Click **OK**. The **Start tracing now?** message box is displayed. Click **Yes** to start tracing.

13. Select **Go** from the **Execute** menu of AXD (or use the toolbar button). Execution stops at the breakpoint in the `Main()` function.

14. Select **Go** from the **Execute** menu of AXD (or use the toolbar button). The image continues execution of the image beyond the `main()` breakpoint.

15. In the **Console Window**, enter `1000` as your number of runs through the benchmark. Trace information is displayed in the Trace window, as shown in Figure 5-23. After the ETM triggers, the Trace window updates with the collected trace information.



```
Index  Address     Data        Mnemonic                                         Status
-162   0x0000821c  0xE2844001  add      r4,r4,#1
       145         C:\Prog...    while (Int_Loc <= 2) /* loop body ex...
-161   0x00008220  0xE3540002  cmp      r4,#2
-160   0x00008224  0xDAFFFFF5  ble      0x8200  ; (Func_2 + 0x14)               Not executed
       153         C:\Prog...    if (Ch_Loc >= 'W' && Ch_Loc < 'Z')
-159   0x00008228  0xE2420057  sub      r0,r2,#0x57
-158   0x0000822c  0xE3500003  cmp      r0,#3
-157   0x00008230  0x2A000000  bcs      0x8238  ; (Func_2 + 0x4c)
       156         C:\Prog...    if (Ch_Loc == 'R')
-156   0x00008238  0xE3520052  cmp      r2,#0x52
-155   0x0000823c  0x1A000001  bne      0x8248  ; (Func_2 + 0x5c)
       161         C:\Prog...      if (strcmp (Str_1_Par_Ref, Str_2_P...
-154   0x00008248  0xE1A0100C  mov      r1,r12
-153   0x0000824c  0xE1A00003  mov      r0,r3
-152   0x00008250  0xEB00041B  bl       strcmp
---    ---         ---         ---                                              Trace pause
-148   0x00008254  0xE3500000  cmp      r0,#0
-146   0x00008258  0xDA000004  ble      0x8270  ; (Func_2 + 0x84)
       169         C:\Prog...        return (false);
-142   0x00008270  0xE3A00000  mov      r0,#0
                               --- No source available ---
-141   0x00008274  0xE8BD8010  ldmfd    r13!,{r4,pc}
---    ---         ---         ---                                              Trace pause
       140         C:\Prog...  {
```

Figure 5-23 Example of Trace window output

Example `.tcf` files are provided in the ADS …`\examples\TDT` directory (see *Code and trace capture examples* on page 1-10).

5.10 Modifying existing rules

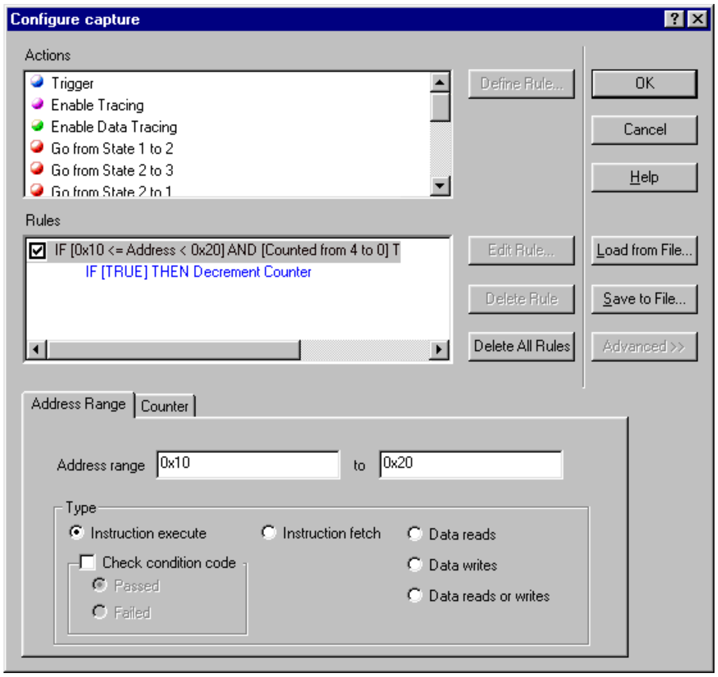
This section describes the two methods you can use for modifying existing rules:

- *Using the tabbed dialog box*
- *Using the wizard* on page 5-48.

5.10.1 Using the tabbed dialog box

To change the types of resources used to define an existing rule, you must use the wizard (see *Using the wizard* on page 5-48). However, to modify the details of any resources in a rule, you can use the tabbed dialog box:

1. Select **Configure capture** from the **Trace** menu.
2. Highlight (single-click) a rule in the rule list.
3. Click on **Advanced>>** to view its details. The details appear on one or more tabs at the bottom of the dialog box, as shown in Figure 5-24.



IF [0x10 <= Address < 0x20] AND [Counted from 4 to 0] T
IF [TRUE] THEN Decrement Counter

Address range 0x10 to 0x20

Type: [✓ Instruction execute]

Check condition code: ☐

Figure 5-24 Using the tabbed dialog box

4. To modify the rule, select the tab displaying the specific resource whose details you want to modify, and change any of the settings of these details (in Figure 5-24 on page 5-47, the rule pertaining to the Address range has been selected). Click **OK** to store the changes. Alternatively, click **Cancel** if you want to discard any changes made.

In Figure 5-24 on page 5-47, the first **Address range** tab relates to the first rule in the list. To modify settings for the decrement counter rule, select the **Counter** tab.

5.10.2 Using the wizard

To change the resources used to define an existing rule, you must use the wizard. You can also use the wizard to modify the details of any resources in the rule. To use the wizard to modify an existing rule:

1. Select **Configure capture** from the **Trace** menu.
2. Double-click on the rule you want to modify. Alternatively, highlight (single-click) the rule and select **Edit** using the right-click feature. The Resource selection dialog box appears.
3. Use the wizard in the same manner as for defining a rule (see *Defining rules* on page 5-15), making changes as necessary.
4. Click **Cancel** if you want to discard any changes made. This returns you to the Configure capture dialog box.

5.11 Enabling, disabling, and deleting rules

To ensure that the desired rules are written to the ETM hardware:

1. Select the rules you want to enable. Rules shown in the Configure capture dialog box are:
   - enabled if the checkbox next to the rule is selected
   - disabled if the checkbox next to the rule is not selected.

   To delete a rule from the list, select the rule and select **Delete Rule**.

   ——— **Note** ———

   When a rule, such as a counter, has supplementary rules, only the main rule has a checkbox. If you enable or disable the main rule, all of its supplementary rules are enabled or disabled, respectively.

   ————————

2. Start tracing in either of the following ways:
   - Click **Yes** in the Start tracing now? message box.
   - Select **Start tracing** from the **Trace** window context menu.

——— **Note** ———

- The ETM evaluates all enabled rules concurrently. There is no flow of control through the list of rules, and there is no significance to the order of rules in the list.
- You can enable rules only if there are sufficient hardware resources available to support them. If you try to enable a rule when there are insufficient resources, an error message that describes the conflict appears when you click **Finish** in the Rule definition complete dialog box, or when you enable a previously disabled rule. The rule you have just defined is disabled. You can only enable it by freeing the conflicting resource.

————————

Chapter 6
The Trace Window Display

This chapter describes the contents of the Trace window, where trace output is displayed. It also contains information on the Trace window *Graphical User Interface* (GUI), specifically the mouse right-click option.

Some elements of the Trace window are different if you are using an XScale target. See *The XScale trace display* on page 7-7 for details.

This chapter contains the following sections:

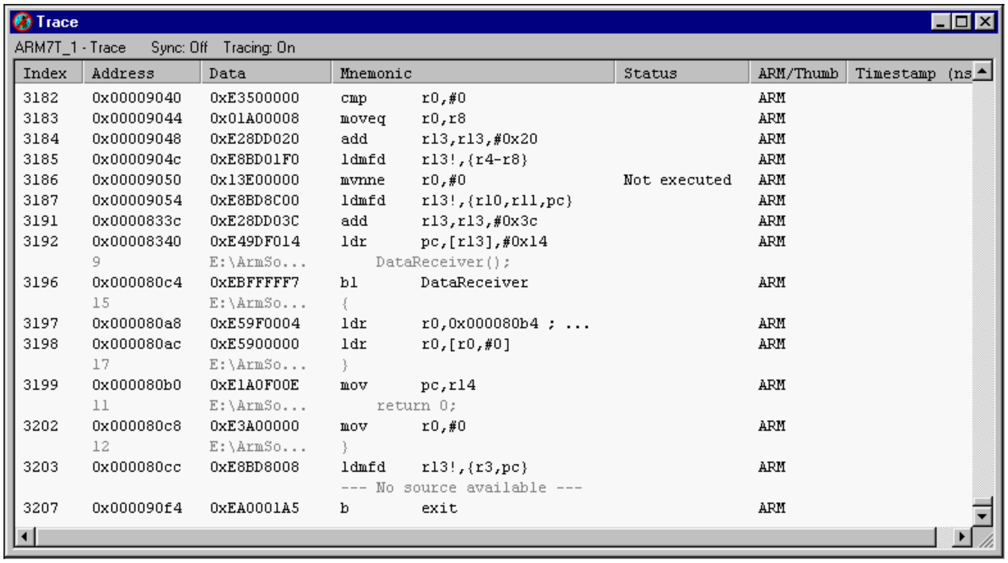
6.1 Trace window output

This section describes the contents of the Trace window:

- *Trace window columns*
- *Trace status information* on page 6-5.

6.1.1 Trace window columns

This section describes the columns displayed in Trace window output. The output information shown by the Trace window is dependent on the options you previously selected when configuring the ETM (see *The Configure ETM dialog box* on page 4-2). Figure 6-1 shows an example of Trace window output.



```
Trace
ARM7T_1 - Trace    Sync: Off   Tracing: On
Index  Address     Data         Mnemonic                          Status        ARM/Thumb  Timestamp (ns
3182   0x00009040  0xE3500000   cmp     r0,#0                                   ARM
3183   0x00009044  0x01A00008   moveq   r0,r8                                   ARM
3184   0x00009048  0xE28DD020   add     r13,r13,#0x20                           ARM
3185   0x0000904c  0xE8BD01F0   ldmfd   r13!,{r4-r8}                            ARM
3186   0x00009050  0x13E00000   mvnne   r0,#0                     Not executed  ARM
3187   0x00009054  0xE8BD8C00   ldmfd   r13!,{r10,r11,pc}                       ARM
3191   0x0000833c  0xE28DD03C   add     r13,r13,#0x3c                           ARM
3192   0x00008340  0xE49DF014   ldr     pc,[r13],#0x14                          ARM
       9           E:\ArmSo...      DataReceiver();
3196   0x000080c4  0xEBFFFFF7   bl      DataReceiver                            ARM
       15          E:\ArmSo...  {
3197   0x000080a8  0xE59F0004   ldr     r0,0x000080b4 ; ...                     ARM
3198   0x000080ac  0xE5900000   ldr     r0,[r0,#0]                              ARM
       17          E:\ArmSo...  }
3199   0x000080b0  0xE1A0F00E   mov     pc,r14                                  ARM
       11          E:\ArmSo...      return 0;
3202   0x000080c8  0xE3A00000   mov     r0,#0                                   ARM
       12          E:\ArmSo...  }
3203   0x000080cc  0xE8BD8008   ldmfd   r13!,{r3,pc}                            ARM
                                --- No source available ---
3207   0x000090f4  0xEA0001A5   b       exit                                    ARM
```

Figure 6-1 Example of Trace window output

The Trace window comprises the following columns:

Index/Cycle Displays one of two pieces of information, depending on the mode:

Cycle-accurate mode

Displays the index in cycles. The column heading reads Cycle in this mode. The difference between the two indexes for two consecutive instructions is equivalent to the number of cycles the second instruction took to execute.

Note

- When there is a discontinuity in the trace output, the difference between the indexes on either side of the discontinuity does not indicate the number of cycles missed.
- If the target hardware stops the processor clock to implement correct memory access timing you cannot convert clock cycles to precise time differences.

Non-cycle-accurate mode

Displays the index as a sequence of integers. The column heading reads Index in this mode. If the trigger has triggered, the index is the offset from the trigger. If there is no trigger point, the initial value of the index is arbitrary.

Note

Differences between consecutive indices do not indicate the number of cycles the instruction took to execute.

Note

No index cycle count is given on memory accesses.

Address Displays different address information, depending on the type of cycle:

Instruction cycles

Displays the address of the instruction executed.

Data transfer cycles

Displays the source or destination address of the transfer, depending on the direction of the transfer.

Data Displays different information, depending on the type of cycle:

Instruction cycles

Displays the opcode of the instruction.

Data transfer cycles

Displays the value of the data transferred.

Mnemonic Displays different information, depending on the cycle:

Instruction cycles

Displays the disassembled information.

Data transfer cycles
Indicates the direction of the transfer.

Status Displays any further information about the cycle, when applicable, which can include one of the following messages:

`Not Executed`
Indicates that a conditional instruction was not executed because the condition code failed.

`Reset` Indicates that a reset exception occurred.

`Undef Exception`
Indicates that an undefined exception occurred.

`Branch to <SWI>`
Indicates that a branch to a SWI occurred.

`Prefetch Abort`
Indicates that a Prefetch Abort occurred.

`Data Abort`
Indicates that a Data Abort occurred.

`Interrupted`
Indicates that an interrupt (IRQ) occurred.

`Fast Interrupt`
Indicates that a fast interrupt (FIQ) occurred.

`Triggered`
Indicates that the trigger condition was met.

ARM/Thumb

Indicates the processor state for the cycle:
- `ARM` indicates ARM state
- `Thumb` indicates Thumb state.

Timestamp (ns)

Displays the time, in nanoseconds, that an instruction was executed relative to the trigger. This column remains blank if your trace capture buffer, or its current configuration, does not support timestamping.

6.1.2 Trace status information

As shown in Figure 6-1 on page 6-2, there is status information at the top of the Trace window. This displays the following information:

Tracing	`On` when tracing is enabled, and `Off` when tracing is disabled (see *Controlling trace capture* on page 3-11). For trace capture to start, several other conditions must be TRUE as well (see *Viewing trace output with source code* on page 3-13).
Sync,	Indicates whether the Synchronize window option has been enabled in the Trace view context menu (see *Viewing trace output with source code* on page 3-13).

6.2 Trace-specific GUI

When navigating in the Trace window, you can right-click with the mouse to select Trace view specific options. Figure 6-2 shows these options.

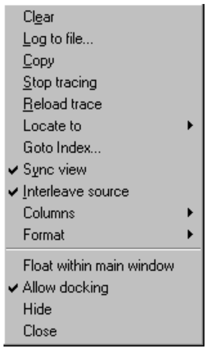


Figure 6-2 Right-click options in the Trace window

The Trace view right-click options are:

Clear Clears the current contents of both the Trace window and the trace capture buffer.

Log to file Saves the current trace details to a file. Toggles to **Cancel log file** while trace details are being saved (see *Saving the trace output to file* on page 3-23 for details).

Copy Copies the currently selected lines in the view to the clipboard. You can use the standard Windows extended selection techniques to select multiple lines.

Start tracing, Stop tracing

Starts (if not started) or stops (if already started) tracing using the currently defined rules. The menu item name changes to show the available action.

——— **Note** ———

You must also define a rule that includes the **Enable tracing** action and then execute code before any trace information is displayed in the view.

———————————

Reload trace Restart the current trace session, reloading all available information from the trace capture hardware.

Locate to

You can use the trace information to

- display a trace line with a specific index
- locate content in another AXD view based on an address taken from the selected trace line:
  - for data access trace lines, the address is the data memory address
  - for instruction trace lines, the address is the instruction address
  - for other lines (for example, trace interruption markers) there is no defined address.

The possible views are shown in Figure 6-3.

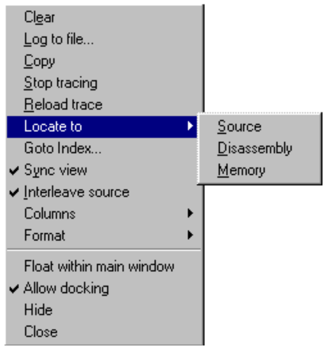


Figure 6-3 Trace view Locate to menu

Source	Displays the current image source window at the line of source code corresponding to the highlighted line in the Trace window.
Disassembly	Displays the current image disassembly window at the line of ARM assembly code corresponding to the highlighted line in the Trace window
Memory	Displays a memory window showing memory at the address provided from the traced line.

Goto Index...

Use this item to scroll the Trace window to show a line with a specified index number. Selecting this item displays the Goto index number dialog box, enabling you to enter the number of the line to show.

If you enter an index number that does not exist, because index numbers skip values, the window scrolls to show the line nearest the one you asked for.

Sync view You can select this to enable automatic scrolling in the currently active source or disassembly window (see *Viewing trace output with source code* on page 3-13).

Interleave source

You can interleave assembly instructions and high-level source code in the trace view by enabling this item.

Columns Specifies which columns are displayed in the Trace window. Select or deselect those columns which you want or do not want displayed, respectively:

The possible columns are shown in Figure 6-4.

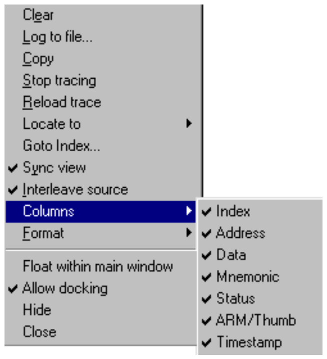


Figure 6-4 Trace view Columns menu

For a description of these columns, see *Trace window output* on page 6-2.

For details on adjusting the column width and column order, see *Adjusting the displayed columns* on page 3-12.

Format Specifies the numerical format in which values in the Data column appear.

The possible formats are shown in Figure 6-5 on page 6-9. Refer to the *ADS Debuggers Guide* for further information.

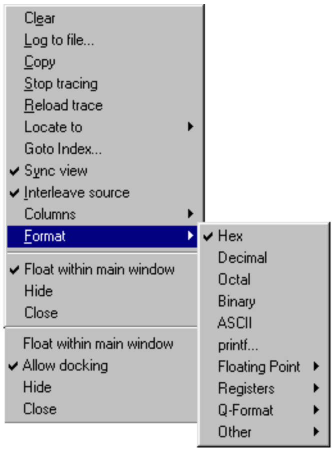


Figure 6-5 Trace view Format menu

6.3 Warnings and other messages

This section describes the various warnings and messages that might appear in the Trace window output for both ARM ETM and XScale.

Warnings indicate that there is an inconsistency between the trace output and the image you have loaded. This type of inconsistency can be caused by an incorrect image that you have loaded into the debugger, or by the program overwriting itself.

Other messages indicate problems TDT had with decoding the trace buffer contents.

Possible messages are:

`Warning: The next instruction was traced as a branch`

The instruction on the next line is not a branch, but was traced as a branch by the ETM. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying.

`Warning: The next instruction was not traced as an indirect branch`

The instruction on the next line is an indirect branch, but was not traced as such by the ETM. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying.

`Warning: The next instruction was traced as a memory access instruction`

The trace from the ETM indicated that the instruction on the next line read some data from memory, or wrote some data to memory, but the instruction is not a memory access instruction. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying. Decompression of data, and data address, tracing stop until appropriate synchronization points are found in the trace data.

`Warning: The next instruction should have been executed unconditionally`

The trace from the ETM indicated that the instruction on the next line failed its condition code test, so was not executed, but the instruction is one that should have been executed unconditionally. This usually indicates that the wrong image is loaded into the debugger, or the code is self-modifying.

`Warning: Corrupt address in trace data`

The trace data contains an impossible address. This only occurs as a result of a hardware problem (such as a faulty connector).

`Warning: The next instruction was not traced as a branch`

The current instruction is a branch but the trace does not indicate this. This message only occurs in XScale trace.

`The next instruction could not be read`

The memory containing the traced instruction could not be read. This might occur when the program attempts to execute a region of unreadable memory, in which case the instruction should abort with a Prefetch Abort. It might also occur if trace is decoded while the program is still running, and the program attempts to execute code outside the loaded image. In this case, after the program has stopped, you can select **Reload trace** from the Trace window context menu to obtain a complete trace (see *Reloading trace* on page 3-22).

`Memory Address Unknown: insufficient trace data`

This error only occurs near the beginning of the decoded trace when the trace buffer (not the FIFO) of the trace capture hardware has overflowed. It means that there has not yet been a complete memory access address in the trace data, and therefore the trace decoder cannot calculate the address of a data access. The ETM outputs a complete address on the first data access traced, and repeats this every 1024 cycles after this, if there are data accesses to be traced. To reduce buffer usage, other memory addresses are output relative to the last full memory address. If the buffer overflows, and the complete address is lost, the decoder cannot calculate data addresses that occur before the next full data address is emitted.

`Trace pause` Some untraced instructions (that is, code for which tracing was disabled) were executed at this point.

`Synchronization lost`

The received trace was corrupt, or did not correspond to the loaded image. The trace decompressor could not even calculate the addresses of the executed instructions. All trace output is suppressed until a new synchronization point is found.

`Trace FIFO overflows`

The FIFO overflowed and as a result some trace was lost.

`Debug state`

The processor entered debug state. Tracing stops while the processor is in debug state.

`Coprocessor data transfer of unknown size`

When tracing data, TDT executed an unrecognized coprocessor memory access instruction, and the decompressor could not deduce the amount of data transferred by the instruction. Decompression of data, and data address, tracing stop until appropriate synchronization points are found in the trace data.

`Data synchronization lost following FIFO overflow`

Some versions of the ARM ETM can cause corrupt data trace after a FIFO overflow has occurred. If the decompressor sees a case where this is likely to have happened, it outputs this message, and suppresses data and data address tracing until it can resynchronize.

`Data address synchronization restored`

This message occurs after a problem in which data address synchronization has been lost. It indicates that the decompressor has found a point at which it can resume decompressing data addresses.

`Data synchronization restored`

This message occurs after a problem in which data synchronization has been lost. It indicates that the decompressor has found a point at which it can resume decompressing data.

`Unable to trace Java state, trace data ignored`

The ETM detected the processor entering Java state. The decompressor is unable to decompress Java bytecode execution, so all trace output is suppressed until the processor leaves Java state.

`No data in trace buffer`

The trace buffer is composed entirely of zero. This error is very rare. This message only occurs in XScale trace.

`Traced branch address does not match instruction's branch address.`

The tools have branch addresses from both the trace and from the image loaded into the debugger, and these addresses do not match. This error is uncommon. This message only occurs in XScale trace.

Chapter 7
Using TDT with XScale

This chapter explains how to use TDT with XScale. Because the XScale target has its own trace buffer, many of the configuration options used with the ARM ETM are invalid, and some elements of the user interface are different for XScale trace.

This chapter contains the following sections:

7.1 The ARM trace solution for XScale targets

When TDT is used with an XScale target, the debug hardware and XScale core are embedded within an XScale *Application Specific Standard Product* (ASSP). Figure 7-1 shows the overall system and the interaction between the components when an XScale target is used. See *Setting up the trace hardware for XScale* on page 7-3 for details on setting up your hardware.

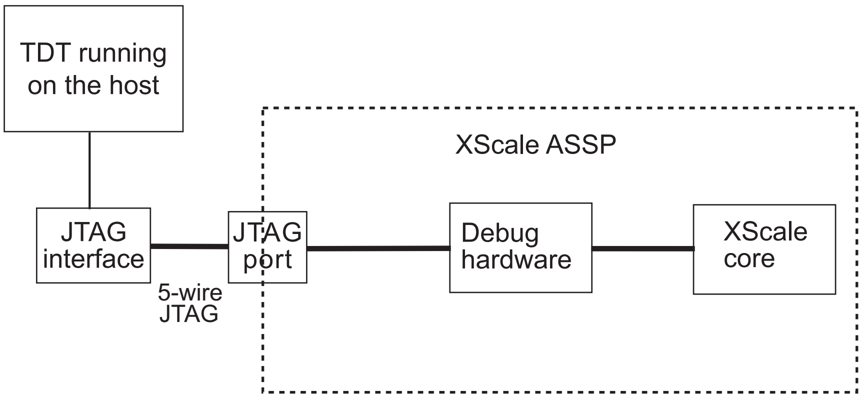


Figure 7-1 Real-time trace solution for XScale

The elements that provide the capability to trace instructions and/or data accesses in real time are:

XScale core This is a processor core based on the Intel® XScale™ Microarchitecture core. The XScale core cannot trace data, and only traces instructions.

Debug hardware

The debug hardware performs debug functions such as setting breakpoints and watchpoints. It also maintains a control flow history, which is used by the TDT software to create an instruction trace. No external trace capture hardware is therefore required, but the buffer is relatively small.

Trace Debug Tools

This retrieves the data from the debug hardware, and reconstructs a historical view of processor activity. See *The Trace Debug Tools (TDT)* on page 1-8 for more details.

The JTAG interface is a protocol converter, such as Multi-ICE, that converts low-level commands (such as programming a register) from AXD into JTAG messages for the XScale debug logic.

7.2 Setting up XScale trace

This section describes how to set up your trace hardware for use with an XScale target, and how to configure AXD to recognize the existence of the trace hardware.

7.2.1 Setting up the trace hardware for XScale

This section describes the requirements and procedure for setting up the trace system for XScale. XScale trace allows you to directly trace your target hardware. To do this, you must set up a hardware trace system, assembling the following components:

- a target board, containing an XScale processor
- a JTAG interface that can set up tracing in the XScale target and read the trace
- a workstation that is able to run TDT.

ARM Multi-ICE is the only JTAG interface supported by the TDT. See *ARM Multi-ICE for XScale* on page A-4 for information on how to set up the trace capture hardware for use with Multi-ICE and XScale.

7.2.2 Setting up the trace software for XScale

After you have set up your hardware (see *Setting up the trace hardware for XScale*), you must configure the software components:

1. Install ADS 1.2 on your workstation, described in the ADS installation instructions.

2. Install TDT 1.2 on your workstation, described in the TDT installation instructions.

3. The software setup instructions are in Appendix B *Setting up the Trace Software*. Refer to *ARM Multi-ICE for XScale* on page B-7 for a description of the setup procedure for your system.

7.3 The tracing procedure with XScale

Note

Before you can debug a target, the debug monitor must be installed on the target. If Multi-ICE fails to detect the debug monitor when it connects, Multi-ICE installs the debug monitor. This involves resetting the processor, so any code running out of ROM stops running. To enable the debug monitor to be installed without resetting the processor, you must implement a special reset handler in the program in ROM. See the *Multi-ICE User Guide* for information on how to do this.

To trace a program with an XScale target:

1. Load the program into the debugger:
   - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.

     Note

     After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

   - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.
2. Locate the area of your program for which you want to collect trace information.
3. Display the Trace window by selecting **View Trace** from the **Trace** menu and arrange the columns as required.
4. Select **Configure Capture** from the **Trace** menu. The Configure XScale Trace dialog box is displayed (Figure 7-2).

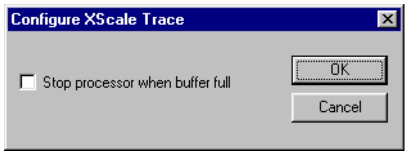


Figure 7-2 Configure XScale Trace dialog box

Select **Stop processor when buffer full** if required.

The option **Stop processor when buffer full** is deselected by default. When this option is set, irrespective of any breakpoints or watchpoints that are set in your code, one buffer-load of trace information is captured and returned to the Trace window whenever you instruct TDT to start tracing. In this case, a buffer-load of trace data is collected at the current execution point in your program. For example, if you want to analyze trace information at the start of the execution of the function `main()`, you must ensure that program execution commences at that function when you start tracing. You can do this, for example, by stepping through your program to the `main()` function before you begin tracing.

If you want tracing to stop when a breakpoint or watchpoint is reached, irrespective of whether the trace buffer is full, you must ensure that the option **Stop processor when buffer full** is deselected.

——— **Note** ———

If **Stop processor when buffer full** is selected, and the trace buffer becomes full, the debug hardware raises a debug exception. If this happens, the target stops executing the image and the trace is displayed immediately.

———————————

5. Select **Start Tracing** from the **Trace** menu.

——— **Note** ———

Unlike for ETM targets, you must start tracing before running the target, because you cannot start or stop tracing while the target is running.

———————————

6. Select **Go** from the **Execute** menu. Trace information is returned to the Trace window when:
   - the trace buffer is full and the target has stopped running due to a debug exception, if you have the **Stop processor when buffer full** option checked
   - tracing is enabled and you stop the target from running
   - tracing is enabled and the target has stopped for another reason.

   See Chapter 6 *The Trace Window Display* for full details of trace output. *The XScale trace display* on page 7-7 describes the ways that the XScale Trace window differs from the ARM ETM Trace window.

7.4 Controlling trace capture

This section describes the ways you can control the capture of trace data.

——— **Note** ———

You cannot start and stop tracing while the XScale target is running. You must select **Start tracing** or **Stop tracing** before running the program, because these menu items are grayed out during program execution. The menu items described below are grayed out while the XScale target is running.

———————————

7.4.1 Start tracing

To indicate that you want tracing to start, select **Start Tracing** from the **Trace** menu.

7.4.2 Stop tracing

To stop trace information from being returned to the Trace window, you can select **Stop Tracing** from the **Trace** window context menu after tracing has started. At this point, the trace capture hardware stops collecting trace data, and any trace data collected is displayed in the Trace window. This display reflects only the instructions executed prior to selecting **Stop Tracing**. You can restart tracing at any time by selecting **Start Tracing** from the **Trace** window context menu.

——— **Note** ———

If you restart tracing, any trace information that has already been captured by hardware trace capture buffers is lost and the Trace window is cleared of all contents.

———————————

7.5 The XScale trace display

This section describes the elements of the XScale trace display window that are different from the ARM ETM trace display:

- Instructions never have a `Not executed` status. Xscale trace does not tell you whether or not an instruction actually passed its condition code.
- XScale trace does not use triggers. Therefore instructions never have a `Triggered` status.
- Timestamps are not enabled. This is because timestamps are relative to the trigger point, and triggers are not used in XScale trace. The Timestamp column in the trace display window remains blank.
- XScale trace produces some warnings and other messages that are not found in ARM ETM trace. These are:
    - `Warning: The next instruction was not traced as a branch`
    - `No data in trace buffer`
    - `Traced branch address does not match instruction's branch address.`

  See *Warnings and other messages* on page 6-10 for a complete list of TDT warnings and other messages.

See Chapter 6 *The Trace Window Display* for more information on the trace display.

7.6 The XScale Trace menu

The **Trace** menu in the **Processor Views** menu for XScale trace contains the same items as the ARM ETM **Trace** menu, but the following items are disabled:

- Configure ETM
- View ETM State.

See *The Trace menu* on page 3-2 for information on the **Trace** menu.

Chapter 8
Examples

This chapter contains examples showing the use of TDT. It contains the following sections:

8.1 About the examples

These examples are provided to help you understand how you can use TDT to help in your program development process. The examples are presented in the form of solutions to hypothetical problems. You can apply similar techniques to other problems. Each example is described in a number of parts:

- a summary of the example
- a description of the situation, in the section Scenario
- one or more ways of using TDT to find out more, in the Solutions.

The examples are independent of each other, and you can work through a single example without having to refer to others.

No source code for these examples is provided in this manual, although you might find source code provided in the ADS product, in the directory `.../examples/`, useful as a starting point.

No examples are provided for XScale targets.

The examples are:

Debugging a program that dies intermittently

This example shows you how to start to debug a system that crashes or misbehaves intermittently.

It demonstrates how to:

- set up a trace trigger at an address
- enable tracing and view the trace information.

Finding out what changes a global to an invalid value (simple)

This example shows you how to find out the code sequence that changes a global variable, when you know what the changed value is.

It demonstrates how to:

- trigger on specific data accesses to a particular address
- enable tracing and view the trace information.

Finding out what changes a global to an invalid value (complex)

This example shows you how to find out why a global variable changes, when the incorrect value that is written changes or cannot be determined in advance.

It demonstrates how to:

- set up a filtered trigger so that only specific sequences cause the trigger to match

- capture and view all instructions that perform a write to a given location in memory.

Debugging a system that must not be halted

This example shows you how to examine a target that cannot be stopped by using the ETM. You can produce an execution trace of a particular function, with or without the code inserted by exception handling.

It demonstrates how to:

- enable tracing based on the processor executing code within an address range
- enable tracing using stop and start points
- enable tracing using the ETM state machine
- use exclude and include address ranges
- use the Trace window without stopping the target.

8.2 Debugging a program that dies intermittently

Use this procedure to debug a system that crashes or misbehaves intermittently.

8.2.1 Scenario

You have a program running on an ARM target board that occasionally crashes, and you are reasonably sure that something in the program causes a Data Abort. However, you cannot determine why this is happening by considering the available output or by using the other debugging tools available to you.

8.2.2 Solution

You do not know very much about what is happening, so to obtain more information you can capture a trace of the instruction sequences leading up to the Data Aborts. Doing this involves:

- triggering on an instruction access at the Data Abort exception vector
- positioning the trigger at the end of the trace buffer (because you know when the system has failed, not when it starts to fail)
- enabling tracing.

The example requires you to perform the following procedures:

- *Loading the debugger*
- *Setting up trace capture* on page 8-5
- *Tracing the target* on page 8-9
- *Analyzing the trace data* on page 8-9.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
    - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
    - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

Note

After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up trace capture

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-1.

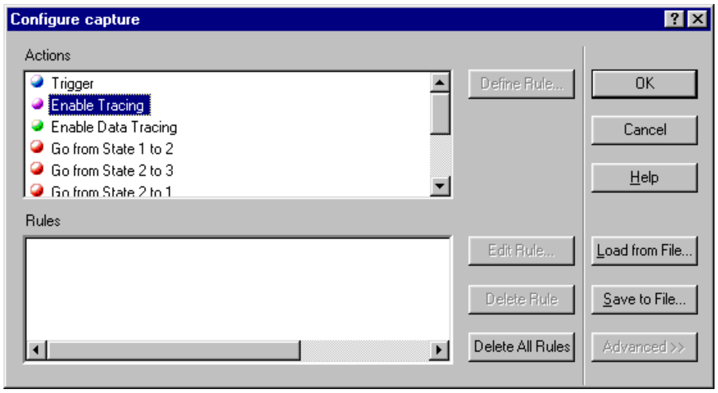


Figure 8-1 Configure capture dialog box

2. Double-click **Enable Tracing** from the Actions list to define a rule. The Resource selection dialog box shown in Figure 8-2 on page 8-6 is displayed.

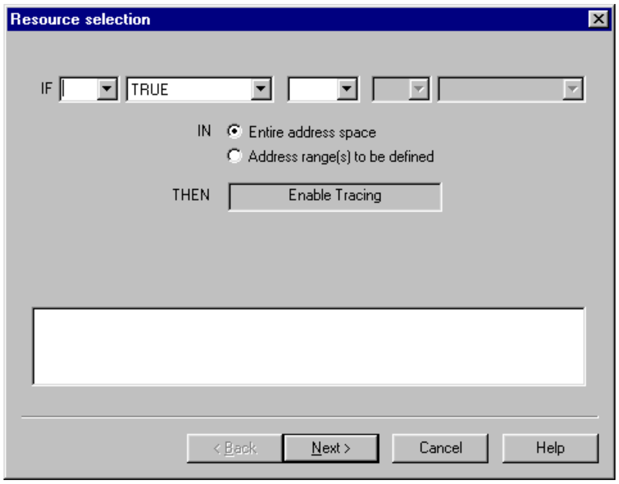
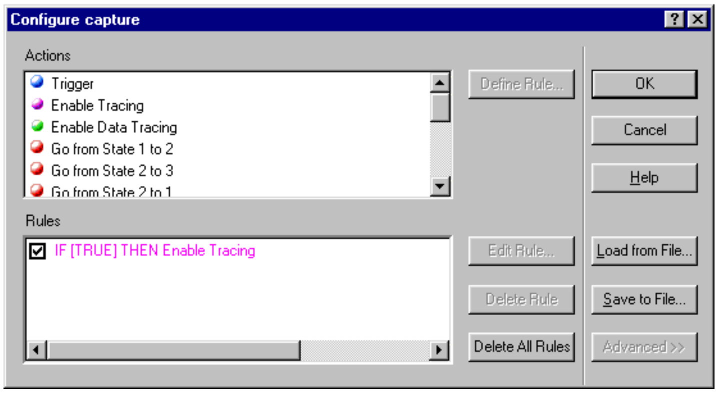


Figure 8-2 Resource selection for Enable Tracing

3. The dialog box settings enable tracing for the whole address space, which is what you require. Click **Next>** to display the end of rule dialog box.

4. Click **Finish** to create this rule. The Configure Capture dialog box now shows the new Enable Tracing rule in the Rules list, as shown in Figure 8-3.



IF [TRUE] THEN Enable Tracing

Figure 8-3 Configure capture dialog box

5. Double-click **Trigger** in the Actions list to define a rule. The Trigger resource selection dialog box shown in Figure 8-4 on page 8-7 is displayed.

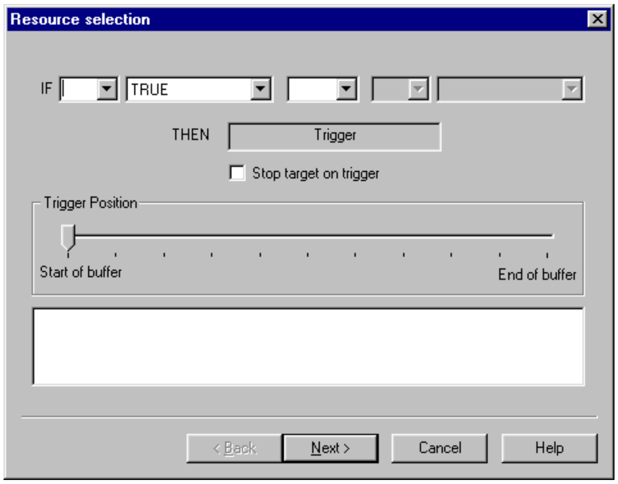


Figure 8-4 Resource selection for Trigger

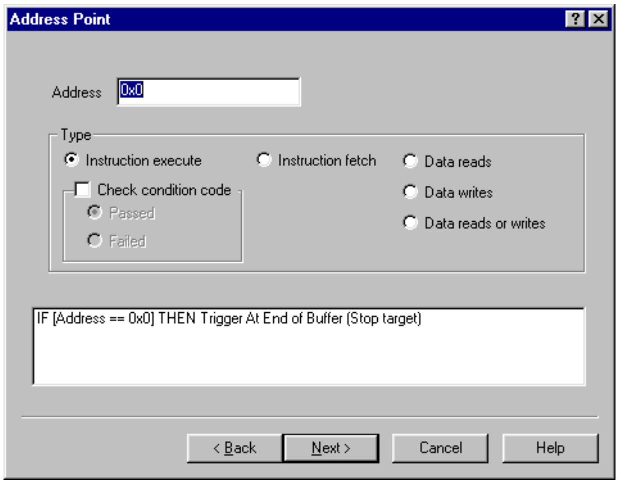
6. Click on the drop-down list showing TRUE and select **Address Point**.

7. Optionally, click on **Stop target on trigger**. Stopping the target makes it easier to debug because the debugger can read target memory when it has to, rather than relying on the program image file.

8. Move the Trigger Position slider to the End of buffer position.

   Setting up the trigger in this way causes the trace buffer to match on an address, which we will define next, and contain all the trace information up to the point the trigger matches.

9. Click on **Next>**. The Address Point dialog box shown in Figure 8-5 on page 8-8 is displayed.



Address: 0x0

Type: [✓ Instruction execute]

IF [Address == 0x0] THEN Trigger At End of Buffer (Stop target)

Figure 8-5 Address Point dialog box

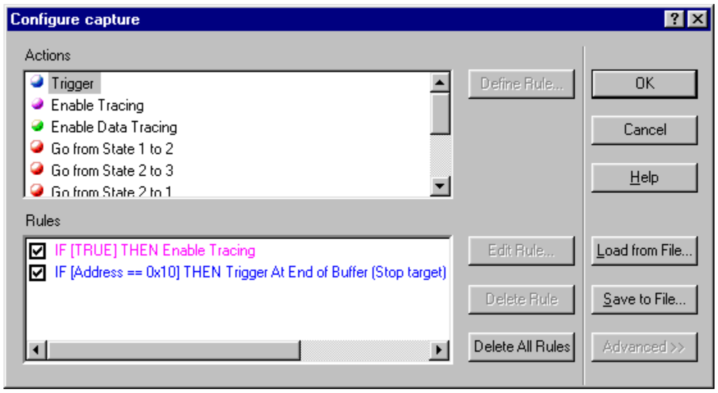
10. In the Address field, enter `@0x10`, the address of the Data Abort exception.

    Note

    Remember that a program can branch to the vectors as well as execute there because of an exception, so this address rule matches more than true Data Aborts.

11. Click **Next>** to display the end of rule dialog box.

12. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-6 now has two rules in the Rules list.



Rules:

- ☑ IF [TRUE] THEN Enable Tracing
- ☑ IF [Address == 0x10] THEN Trigger At End of Buffer (Stop target)

Figure 8-6 Completed Configure capture dialog box

13. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values.
2. Click on **View Trace** from the **Trace** menu. The trace output window appears.
3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).
2. By examining the trace output you can try to determine where the fault is.

8.3 Finding out what changes a global to an invalid value (simple)

Use this procedure when you know that a particular invalid value is being written to a global variable, but you have not been able to find the code that is changing it.

8.3.1 Scenario

You are running a program that uses interrupts and has a global data pool. The program consistently fails, but when you use breakpoint debugging there is no pattern to the failure except that a function picks up the value 255 from an enumerated global variable called `IndexMode` that has a maximum value of 8.

8.3.2 Solution

This problem can be solved by tracing writes to the address of the global. Because you know the incorrect value, and the location of the incorrect write, you can program a capture rule to trigger on this event.

The example requires you to perform the following procedures:

- *Loading the debugger*
- *Setting up the Enable Tracing trace capture rule* on page 8-11
- *Setting up the Trigger trace capture rule* on page 8-13
- *Tracing the target* on page 8-15
- *Analyzing the trace data* on page 8-15.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
    - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
    - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

    Note

    After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the Enable Tracing trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-7.

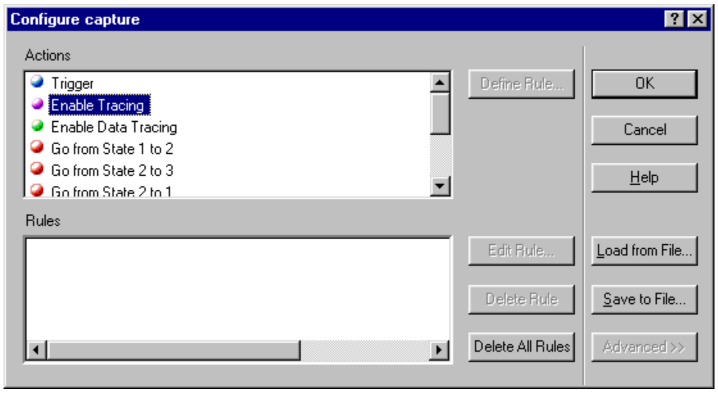


Figure 8-7 Configure capture dialog box

2. Double-click **Enable Tracing** from the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-8 on page 8-12 is displayed.

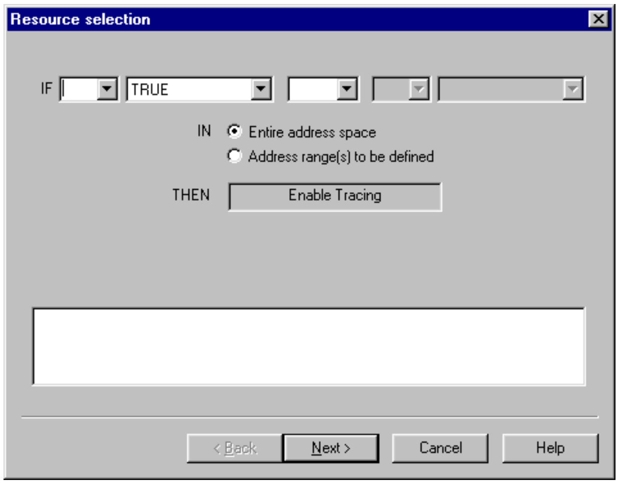


Figure 8-8 Resource selection for Enable Tracing

3. The dialog box settings enable tracing for the whole address space, which is what you require. Click **Next>** to display the end of rule dialog box.

4. Click **Finish** to create this rule. The Configure capture dialog box now shows the new Enable Tracing rule in the Rules list, as shown in Figure 8-9.

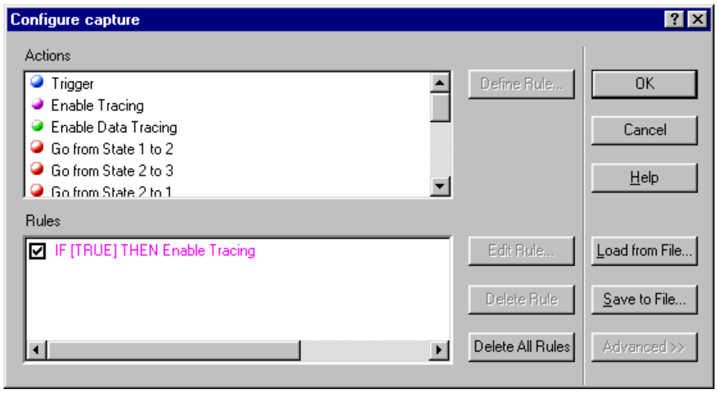
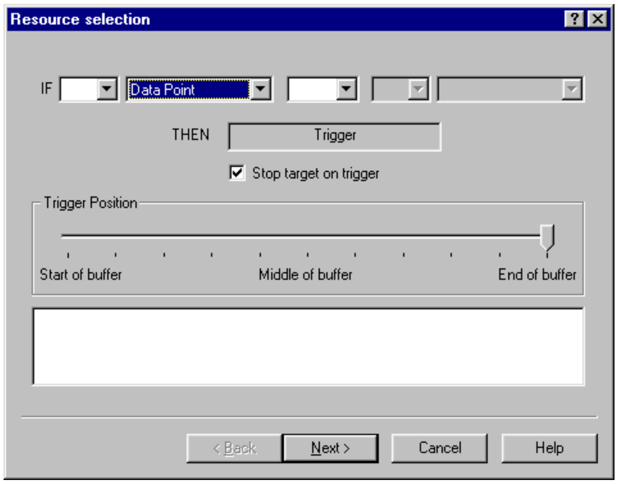


Figure 8-9 Configure capture dialog box

Setting up the Trigger trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Trigger** in the Actions list to define a rule. The Trigger Resource selection dialog box shown in Figure 8-10 is displayed.



IF Data Point

THEN Trigger

☑ Stop target on trigger

Figure 8-10 Resource selection for Trigger

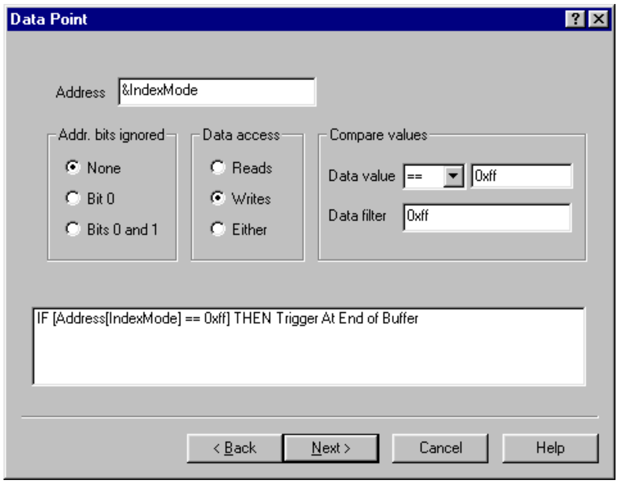
2. Click on the drop-down list showing TRUE and select **Data Point**.

3. Optionally, click on **Stop target on trigger**. Stopping the target makes it easier to debug because the debugger can read target memory when it needs to, rather than relying on the program image file.

4. Move the Trigger Position slider to the End of buffer position.

   Setting up the trigger in this way causes the trace buffer to match on an address, which we will define next, and contain all the trace information up to the point the trigger matches.

5. Click on **Next>**. The Data Point dialog box shown in Figure 8-11 on page 8-14 is displayed.



Address: &IndexMode

Addr. bits ignored: [✓ None] Bit 0 Bits 0 and 1

Data access: Reads [✓ Writes] Either

Data value: == 0xff

Data filter: 0xff

IF [Address[IndexMode] == 0xff] THEN Trigger At End of Buffer

Figure 8-11 Data Point dialog box

6. In the Address field, enter `&IndexMode`, the address of the global variable.

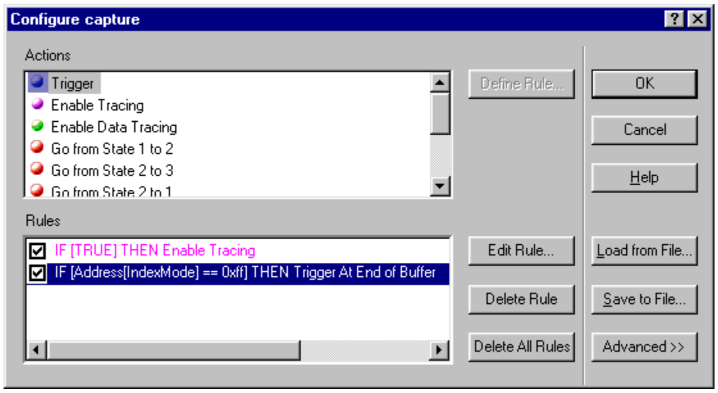
7. In the Data value field, enter `255`. If you are using ETM hardware implementing ETM version 1.0 or 1.1, the drop down list is disabled. Otherwise, it must be set to **==**.

8. In the Data access field, click on **Writes**.

9. In the Data filter field, enter `0xFF`, the mask for a single byte value.

10. Click **Next>** to display the end of rule dialog box.

11. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-12 on page 8-15 now has two rules in the Rules list.



IF [TRUE] THEN Enable Tracing
IF [Address[IndexMode] == 0xff] THEN Trigger At End of Buffer

Figure 8-12 Completed Configure capture dialog box

12. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values.

2. Click **View Trace** on the **Trace** menu. The trace output window appears.

3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).

2. By examining the trace output you can try to determine where the fault is.

8.4 Finding out what changes a global to an invalid value (complex)

Use this procedure when you know that a random invalid value is being written to a global variable, but you have not been able to find the code that is changing it.

8.4.1 Scenario

You have a program running on an ARM target board that uses interrupts and has a global data pool. The program intermittently fails, and the use of breakpoint debugging has revealed just that a function called `DataReceiver` reads an incorrect and seemingly random value from a global variable called `DataCounter`. Unfortunately, there are many valid writes to this variable, so the solution provided in *Finding out what changes a global to an invalid value (simple)* on page 8-10 is inadequate.

8.4.2 About the solutions

This problem can be solved by tracing writes to the address of the global, but to make this practical you must filter out at least most of the correct writes from the trace. You must find out the functions you expect to write to the global and exclude either the whole function, or just the region within the function that does the write, from the addresses that are traced.

If there is more than one function that you must filter out, and you have ETM hardware implementing ETM version 1.2, you can do one of the following:

- filter out a range of addresses that includes all the functions that perform the write, in the belief that the erroneous write is not included in that range. This is described in *Solution using a filtered trigger*.
- capture all instructions that write to the variable and postprocess the trace output. This is described in *Solution by capturing data writes* on page 8-23.

If you have ETM hardware implementing ETM version 1.0 or 1.1, the ETM is not capable of capturing all instructions writing to a specified address, and so you must use the filter, possibly using more than one run of the program to narrow down the address range.

8.4.3 Solution using a filtered trigger

This solution is based on filtering out a range of addresses that includes all the instructions that validly write to the variable, on the assumption that the erroneous write is not included in that range. It involves:

- *Loading the debugger* on page 8-17
- *Setting up the Enable Tracing trace capture rule* on page 8-17

- *Setting up the Trigger trace capture rule* on page 8-19
- *Tracing the target* on page 8-22
- *Analyzing the trace data* on page 8-22.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
    - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
    - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

    —— **Note** ——

    After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

    ————————

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the Enable Tracing trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-13 on page 8-18.

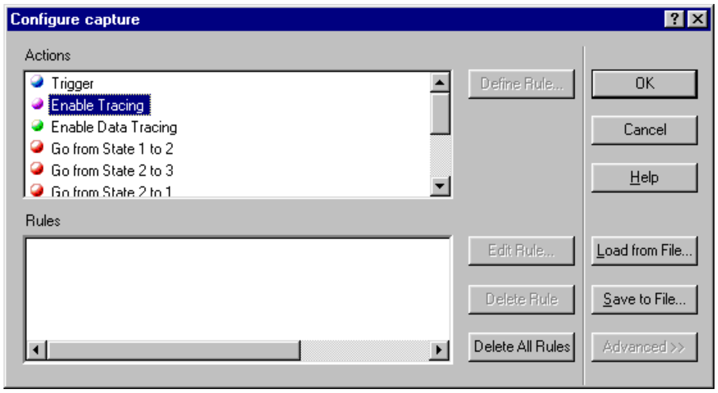


Figure 8-13 Configure capture dialog box

2. Double-click **Enable Tracing** in the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-14 is displayed.

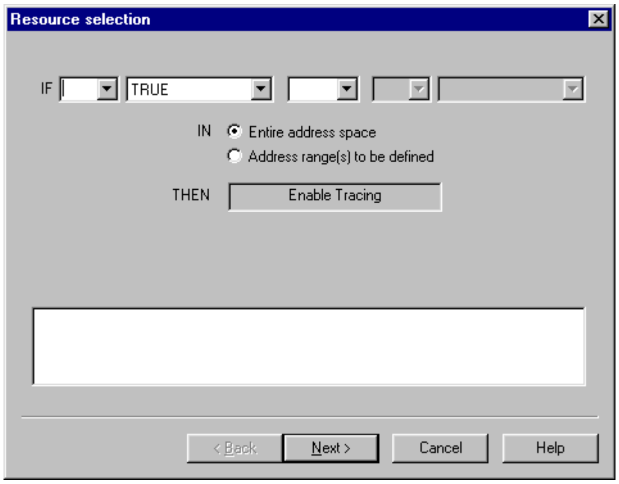
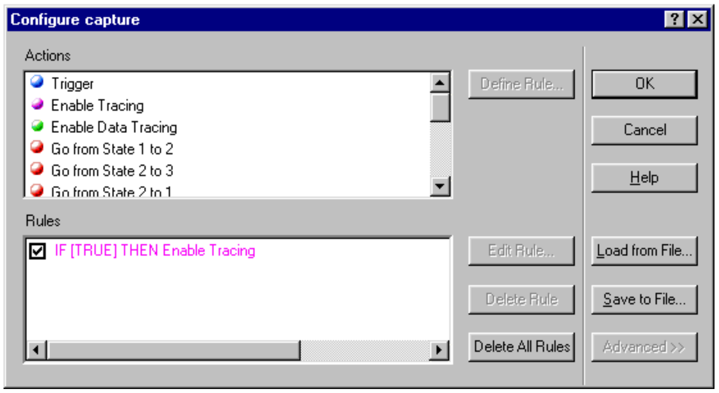


Figure 8-14 Resource selection for Enable Tracing

3. The dialog box settings enable tracing for the whole address space, which is what you require. Click **Next>** to display the end of rule dialog box.

4. Click **Finish** to create this rule. The Configure capture dialog box now shows the new Enable Tracing rule in the Rules list, as shown in Figure 8-15 on page 8-19.



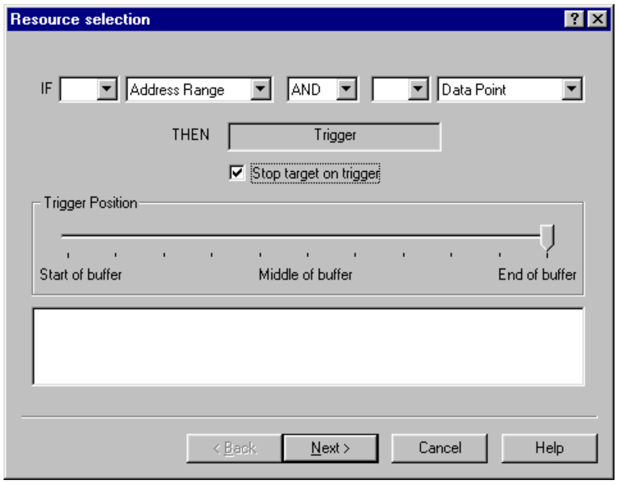
IF [TRUE] THEN Enable Tracing

Figure 8-15 Configure capture dialog box

Setting up the Trigger trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Trigger** in the Actions list to define a rule. The Trigger Resource selection dialog box shown in Figure 8-16 is displayed.



IF [ ] Address Range AND [ ] Data Point
THEN Trigger
☑ Stop target on trigger

Figure 8-16 Resource selection for Trigger

2. Click on drop-down list number 1 (counting to the right from the top left of the dialog box) and select **NOT**.

3. Click on drop-down list number 2 and select **Address Range**.

4. Click drop-down list number 3 and click **AND**. Leave drop-down list number 4 blank.

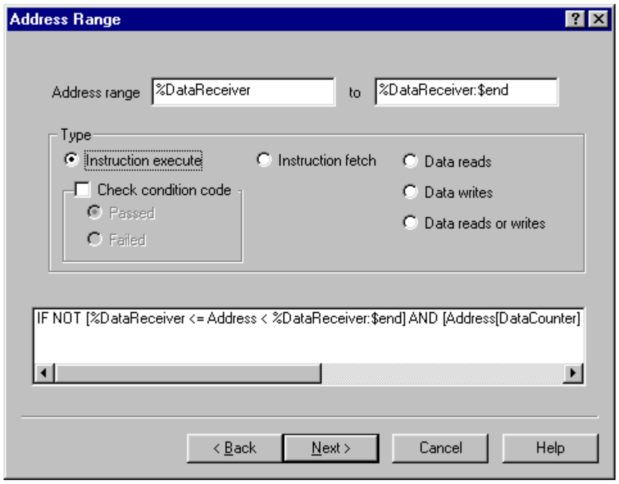
5. Click drop-down list number 5, and click **Data Point**.

6. Optionally, click on **Stop target on trigger**. Stopping the target makes it easier to debug because the debugger can read target memory when it needs to, rather than relying on the program image file.

7. Move the Trigger Position slider to the End of buffer position.

   Setting up the trigger in this way causes the trace buffer to match on an address, defined next, and so to contain all the trace information up to the point the trigger matches.

8. Click on **Next>**. The Address Range dialog box shown in Figure 8-17 is displayed.


Address range: %DataReceiver to %DataReceiver:$end

Type: (●) Instruction execute

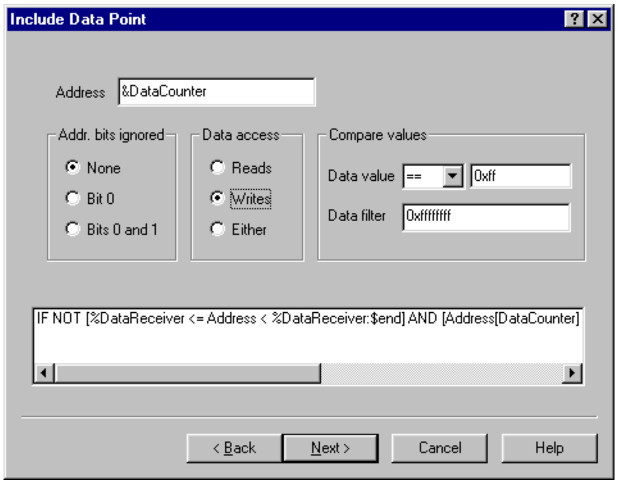
```
IF NOT [%DataReceiver <= Address < %DataReceiver:$end] AND [Address[DataCounter]
```

Figure 8-17 Address Range dialog box

9. Enter `%DataReceiver` and `%DataReceiver:$end` into the **Address range** fields. You selected NOT in the resource selection so this range is excluded from the trigger.

10. Click **Data writes**.

11. Click on **Next>**. The Data Point dialog box shown in Figure 8-18 on page 8-21 is displayed.


Include Data Point

Address: &DataCounter

Addr. bits ignored: [✓ None] Bit 0 / Bits 0 and 1

Data access: Reads / [✓ Writes] / Either

Compare values

Data value: == 0xff

Data filter: 0xffffffff

IF NOT [%DataReceiver <= Address < %DataReceiver:$end] AND [Address[DataCounter]

Figure 8-18 Include Data Point dialog box

12. In the Address field, enter `&DataCounter`, the address of the global variable.

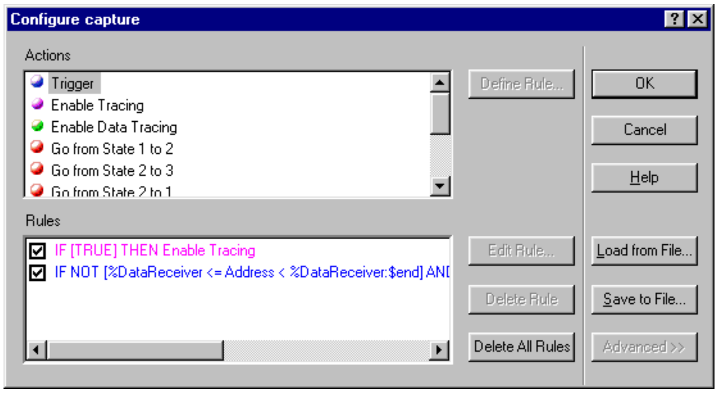
13. In the Data value field, enter 255. If you are using ETM hardware implementing ETM version 1.0 or 1.1, the drop-down list is disabled. Otherwise, it must be set to **==**.

14. In the Data access field, click on **Writes**.

15. In the Data filter field, enter `0xFF`, the mask for a single byte value.

16. Click **Next>** to display the end of rule dialog box.

17. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-19 on page 8-22 now has two rules in the Rules list.



Actions:
- Trigger
- Enable Tracing
- Enable Data Tracing
- Go from State 1 to 2
- Go from State 2 to 3
- Go from State 2 to 1

Rules:
- [✓] IF [TRUE] THEN Enable Tracing
- [✓] IF NOT [%DataReceiver <= Address < %DataReceiver:$end] ANI

Figure 8-19 Completed Configure capture dialog box

18. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values

2. Select **View Trace** from the **Trace** menu. The trace output window appears.

3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).

2. By examining the trace output you can try to determine where the fault is.

8.4.4 Solution by capturing data writes

This solution to the example is based on capturing all the instructions that write to the variable and then postprocessing the trace output. It requires you to perform the following procedures:

- *Loading the debugger*
- *Setting up the Enable Tracing trace capture rule*
- *Setting up the Enable Data Tracing trace capture rule* on page 8-26
- *Tracing the target* on page 8-28
- *Analyzing the trace data* on page 8-28.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
    - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
    - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

    Note

    After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the Enable Tracing trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-20 on page 8-24.

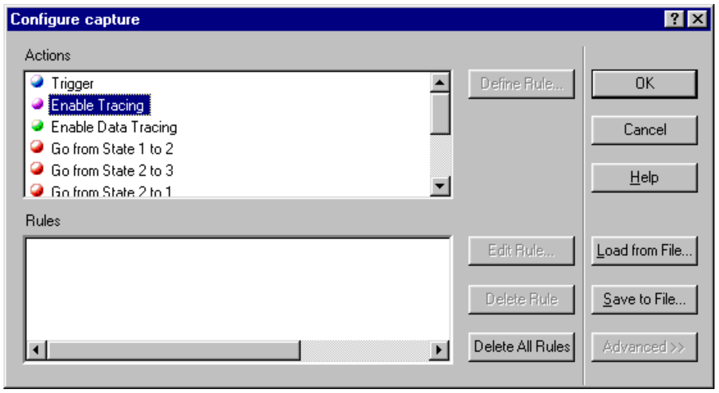


Figure 8-20 Configure capture dialog box

2. Double-click **Enable Tracing** from the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-21 is displayed.

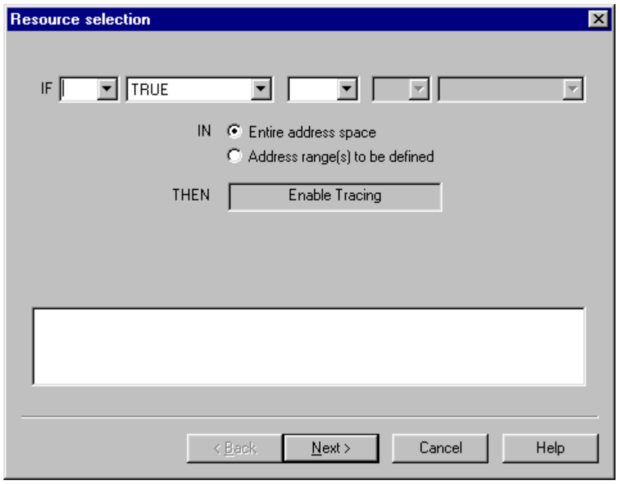


Figure 8-21 Resource selection for Enable Tracing

3. Click on **Address range(s) to be defined**.

4. Click on **Next>**. The Address Range dialog box shown in Figure 8-22 on page 8-25 is displayed.

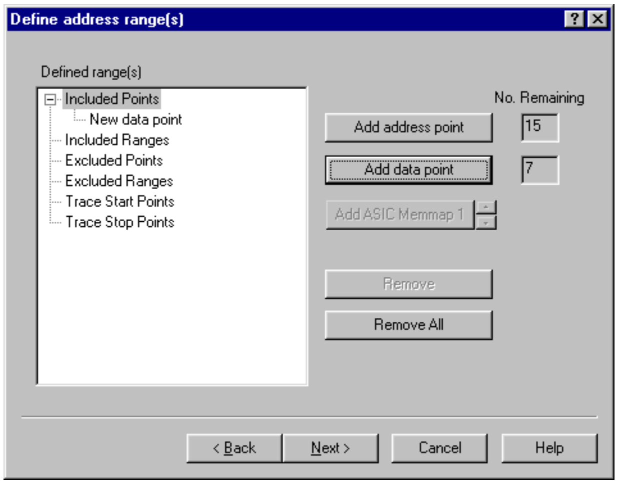
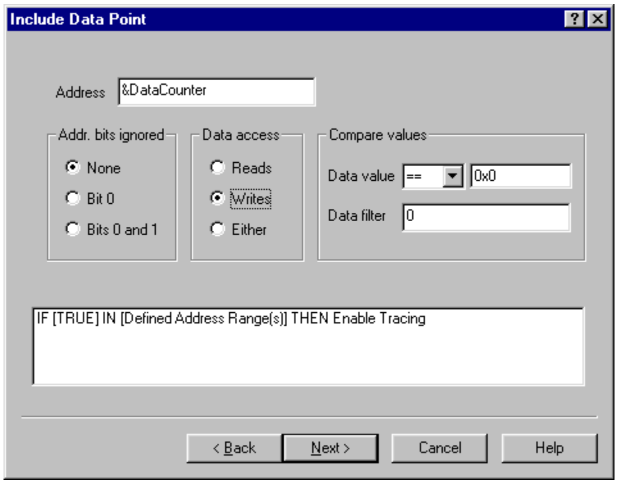


Figure 8-22 Defined address range(s) dialog box

5. With Included Points highlighted, click on **Add data point**. A list item New data point appears in the Defined range(s) tree control.

6. Click **Next>**. The Include Data Point dialog box shown in Figure 8-23 is displayed.



Address: &DataCounter

Addr. bits ignored: [✓ None] Bit 0 Bits 0 and 1

Data access: Reads [✓ Writes] Either

Data value: == 0x0

Data filter: 0

IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-23 Include Data Point dialog box

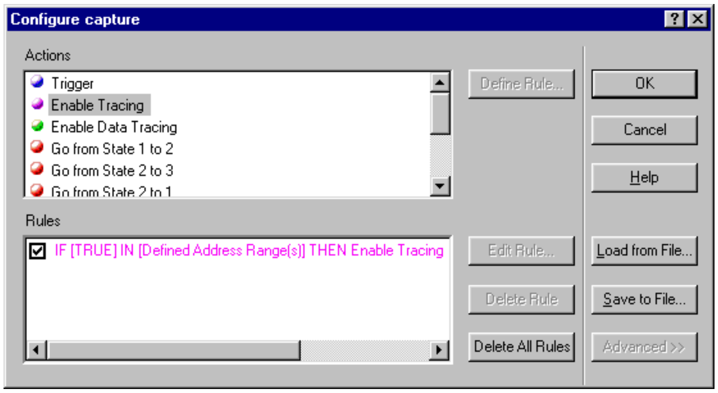
7. Type into the Address field `&DataCounter`, the name of the global variable.

8. Click on **Writes**.

9. Type into the Data filter field the number `0`. This causes the ETM to report all accesses to this address, ignoring the data value.

10. Click **Next>** to display the end of rule dialog box.

11. Click **Finish** to create this rule. The Configure capture dialog box now shows the new Enable Tracing rule in the Rules list, as shown in Figure 8-24.



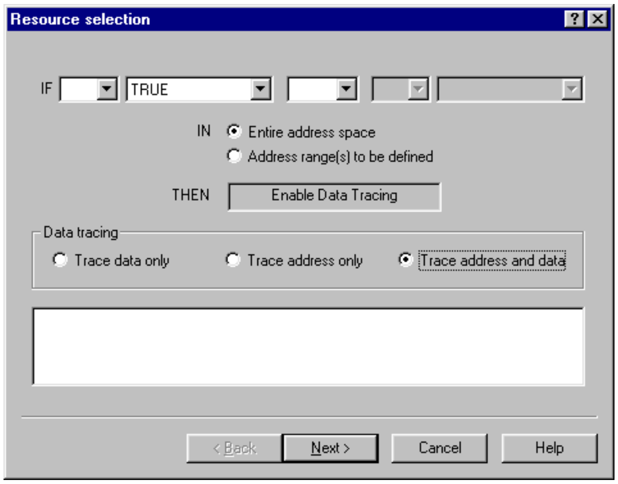
IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-24 Configure capture dialog box

Setting up the Enable Data Tracing trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Enable Data Tracing** in the Actions list to define a rule. The Enable Data Tracing Resource selection dialog box shown in Figure 8-25 on page 8-27 is displayed.



IF TRUE

IN (●) Entire address space ( ) Address range(s) to be defined

THEN Enable Data Tracing

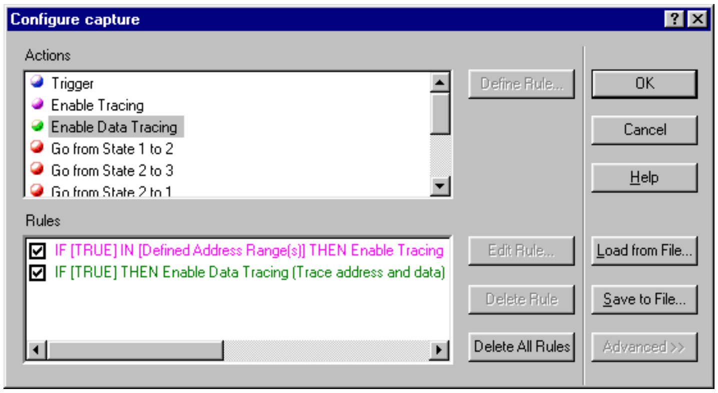
Data tracing: ( ) Trace data only ( ) Trace address only (●) Trace address and data

Figure 8-25 Resource selection for Enable Data Tracing

2. Click **Trace address and data**, so that both the data transfers and the instructions that caused them are reported to you.

3. Click **Next>** to display the end of rule dialog box.

4. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-26 now has two rules in the Rules list.



Rules:

- ☑ IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing
- ☑ IF [TRUE] THEN Enable Data Tracing (Trace address and data)

Figure 8-26 Completed Configure capture dialog box

5. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values.

2. Select **View Trace** from the **Trace** menu. The trace output window appears.

3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).

2. By examining the trace output you can try to determine where the fault is.

8.5 Debugging a system that must not be halted

This procedure describes a debugging problem including a constraint that the system being debugged cannot be stopped for any reason. For example, a device controller that includes software that ensures the hardware is maintained in a stable state.

In this example, you trace code executed during a function call using one of two techniques:

- tracing code in, or not in, a specific address range
- tracing code executed between two addresses.

8.5.1 Scenario

You have a device controller that contains an active feedback control loop using a stepper motor. There is a function, `MonitorRequests`, that continuously monitors the device requirements and generates signals that control the feedback loop.

You are experiencing difficulty in following the interactions of the `MonitorRequests` function with the rest of the control loop and because you cannot single step through it, you require a trace of its operation.

8.5.2 About the solutions

There are several possible solutions, and you must choose between them based on the trace information you require:

- The instructions that form the `MonitorRequests` function, excluding any interrupt handler code and excluding functions that `MonitorRequests` calls. This case is described in *Solution using a defined address range* on page 8-30.
- All instructions executed by the processor between the start and end of the `MonitorRequests` function, including interrupt handler code and including functions that `MonitorRequests` calls. This case is described in:
  - — *Solution using the state machine* on page 8-36, if you are using ETM version 1.1.
  - — *Solution using a start and stop point* on page 8-45, if you are using ETM version 1.2 or higher.
- All instructions executed by the processor between the start and end of the `MonitorRequests` function, excluding functions that you specify, for example, an interrupt handler. This case is described in *Solution using a trace start and stop point and an exclude address range* on page 8-50.

You can include the data values transferred to or from memory in any of the cases listed above.

8.5.3 Solution using a defined address range

This solution to the example is based on tracing the instructions that form the `MonitorRequests` function, excluding any interrupt handler code, and excluding functions that `MonitorRequests` calls. It requires you to perform the following procedures:

- *Loading the debugger*
- *Setting up the Enable Tracing trace capture rule* on page 8-31
- *Setting up the Trigger trace capture rule* on page 8-34
- *Tracing the target* on page 8-36
- *Analyzing the trace data* on page 8-36.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
   - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
   - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

   Note

   After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the Enable Tracing trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-27.

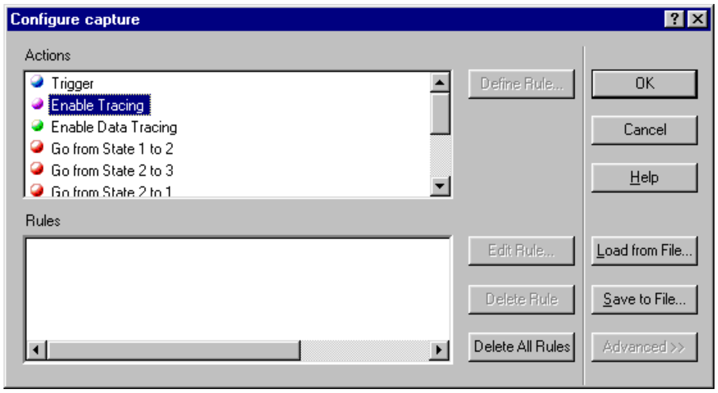


Figure 8-27 Configure capture dialog box

2. Double-click **Enable Tracing** in the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-28 is displayed.

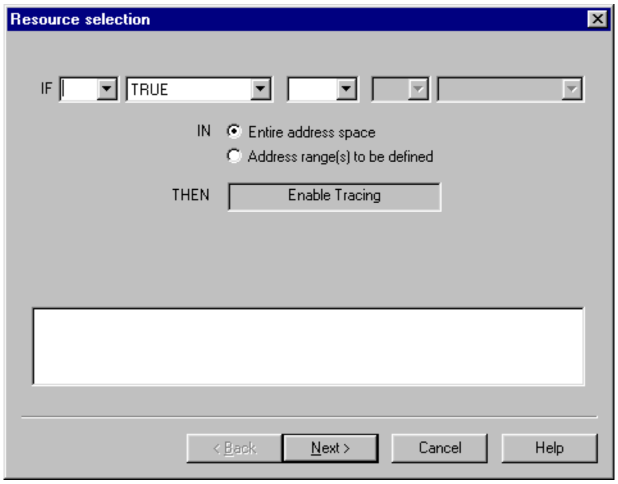


Figure 8-28 Resource selection for Enable Tracing

3. Click on **Address range(s) to be defined**.

4. Click on **Next>**. The Defined address range(s) dialog box shown in Figure 8-29 is displayed.

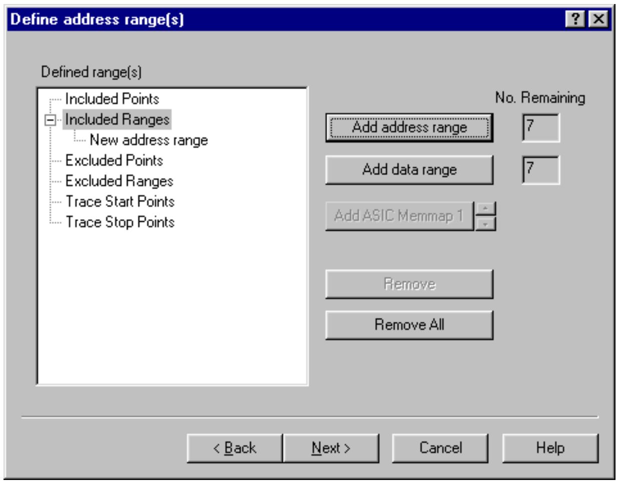


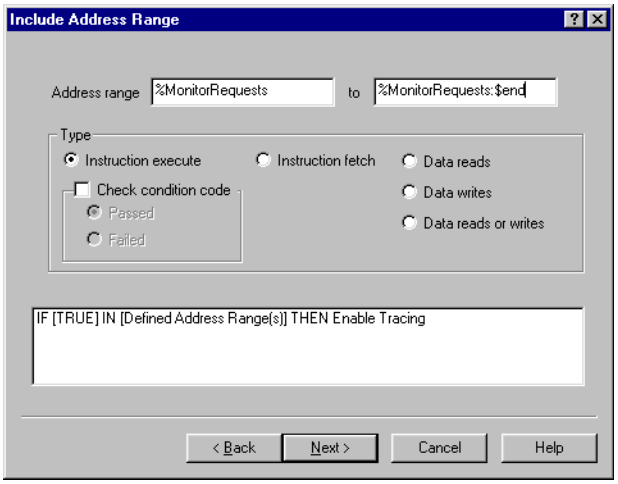
Figure 8-29 Defined Address range(s) dialog box

Note

If you are using ETM hardware implementing ETM version 1.0 or 1.1, Trace Start Points and Trace Stop Points are not listed in the Defined range(s) tree control because the hardware does not implement them.

5. Select **Included Ranges** in the Defined range(s) tree control and click on **Add address range**. A list item New address range appears in the tree control.

6. Click **Next>**. The Include Address Range dialog box shown in Figure 8-30 on page 8-33 is displayed.



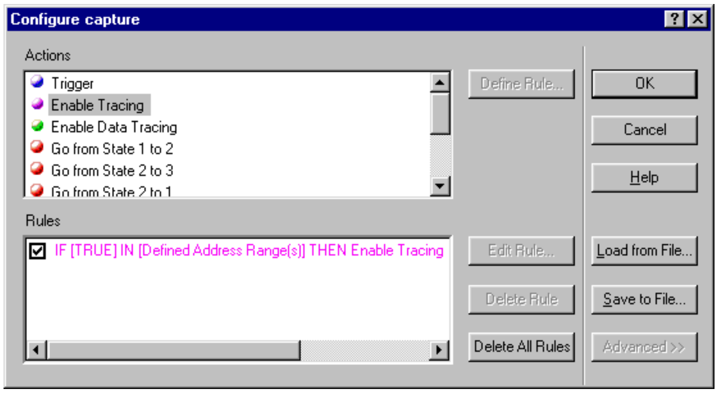
IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-30 Include Address Range dialog box

7. Type into the left Address range field `%MonitorRequests`, and into the right (`to`) field `%MonitorRequests:$end`.

8. Click **Next>** to display the end of rule dialog box.

9. Click **Finish** to create this rule. The Configure capture dialog box now shows the new Enable Tracing rule in the Rules list, as shown in Figure 8-31.



IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-31 Configure capture dialog box

Setting up the Trigger trace capture rule

In this procedure you set up the trigger rule that defines the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Trigger** in the Actions list to define a rule. The Trigger Resource selection dialog box shown in Figure 8-32 is displayed.

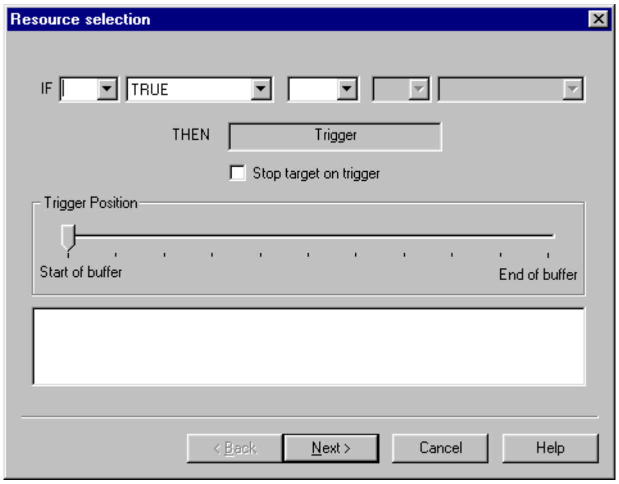


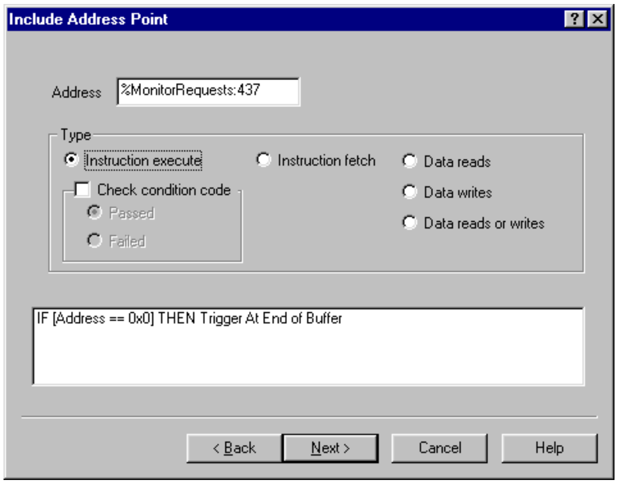
Figure 8-32 Resource selection for Trigger

2. Click on the drop-down list showing TRUE and select **Address Point**.

3. Move the Trigger Position slider to the End of buffer position.

   Setting up the trigger in this way causes the trace buffer to match on an address that you will define in the next step, and contain all the trace information up to the point the trigger matches.

4. Click on **Next>**. The Include Address Point dialog box shown in Figure 8-33 on page 8-35 is displayed.



Address: %MonitorRequests:437

Type: Instruction execute [✓]

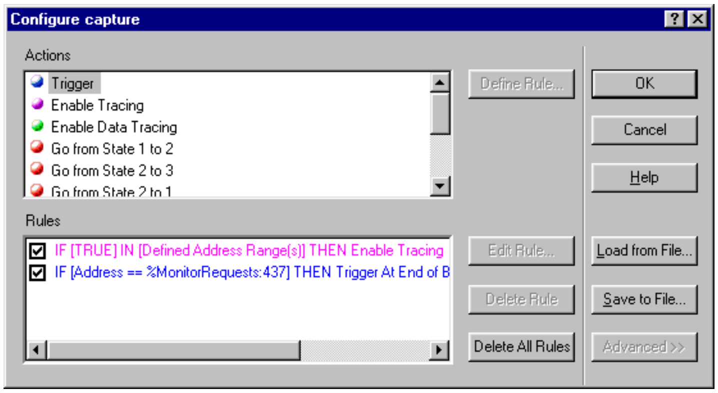
IF [Address == 0x0] THEN Trigger At End of Buffer

Figure 8-33 Include Address Point dialog box

5. In the Address field, enter `%MonitorRequests:437`, where 437 is the line number of the function return code.

6. Click **Next>** to display the end of rule dialog box.

7. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-34 now has two rules in the Rules list.



Rules:

- ✓ IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing
- ✓ IF [Address == %MonitorRequests:437] THEN Trigger At End of B

Figure 8-34 Completed Configure capture dialog box

8. Click on **OK** to activate the rules.

Tracing the target

Follow these steps:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values
2. Click **View Trace** on the **Trace** menu. The trace output window appears.
3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

Follow these steps:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).
2. By examining the trace output you can try to determine what is happening.

8.5.4 Solution using the state machine

This solution to the example is based on tracing the instructions that are executed between entry to and exit from the `MonitorRequests` function. This includes any interrupt handler code as well as functions that `MonitorRequests` calls.

Note

If you are using ETM hardware implementing ETM version 1.2, it is quicker to configure start/stop points for this task. However, you can still use the state machine for this or other solutions.

This solution requires you to perform the following procedures:

- *Loading the debugger* on page 8-37
- *Setting up the function entry state change trace capture rule* on page 8-37
- *Setting up the function exit state change trace capture rule* on page 8-40

- *Setting up the Enable Tracing trace capture rule* on page 8-42
- *Setting up the Trigger trace capture rule* on page 8-43
- *Tracing the target* on page 8-45
- *Analyzing the trace data* on page 8-45.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
   - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
   - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

   ——— **Note** ———

   After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

   ————————————

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the function entry state change trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-35 on page 8-38.

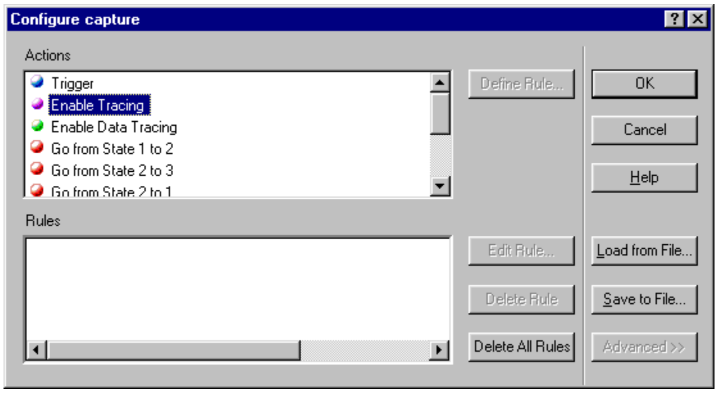


Figure 8-35 Configure capture dialog box

2. Double-click **Go from State 1 to 2** in the Actions list to define a rule. The Go from State 1 to 2 Resource selection dialog box shown in Figure 8-36 is displayed.

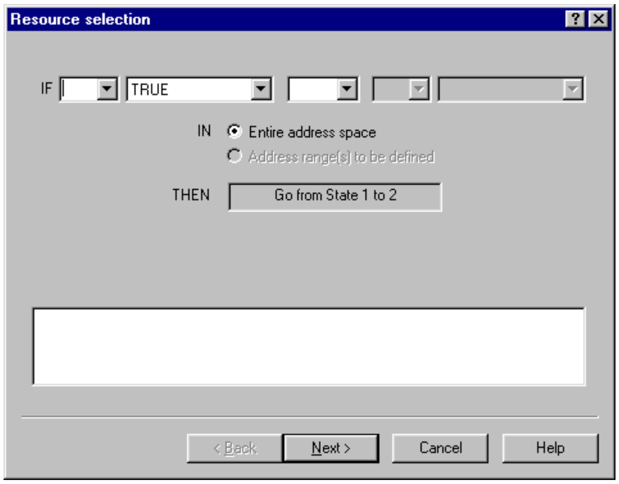
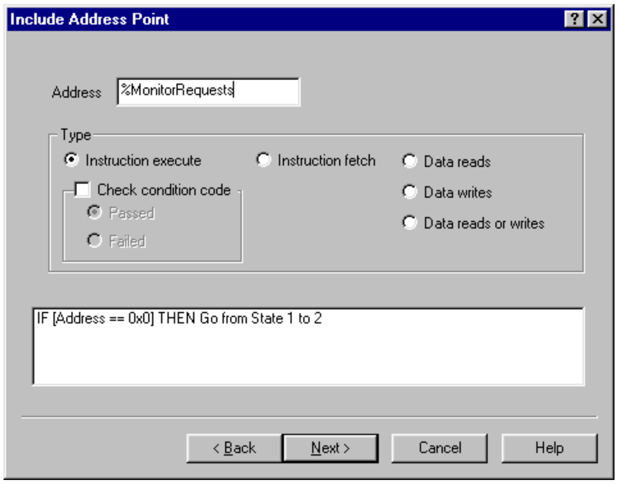


Figure 8-36 Resource selection for Go from State 1 to 2

3. Click on the drop-down list showing TRUE and select **Address Point**.

4. Click on **Next>**. The Include Address Point dialog box shown in Figure 8-37 on page 8-39 is displayed.



Address: %MonitorRequests

Type: Instruction execute

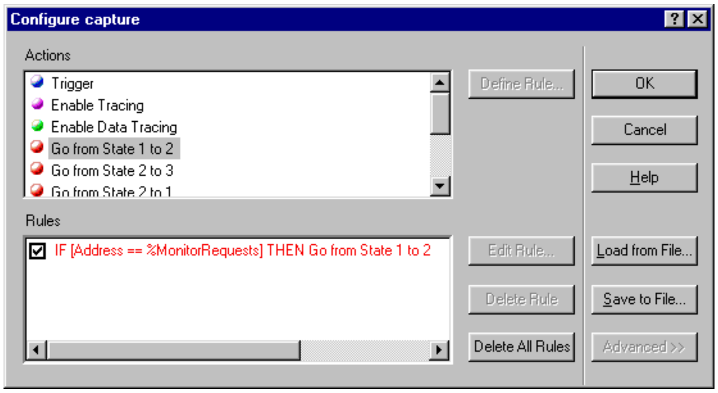
IF [Address == 0x0] THEN Go from State 1 to 2

Figure 8-37 Include Address Point dialog box

5. In the Address field, enter `%MonitorRequests`.

6. Click **Next>** to display the end of rule dialog box.

7. Click **Finish** to create this rule. The Configure capture dialog box is updated as shown in Figure 8-38.



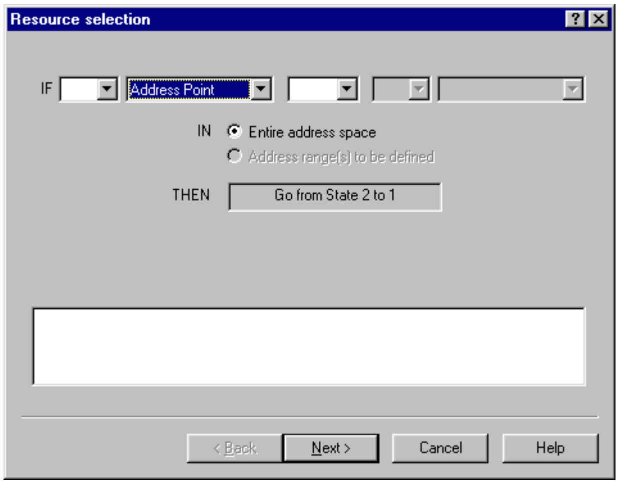
IF [Address == %MonitorRequests] THEN Go from State 1 to 2

Figure 8-38 Configure capture dialog box

Setting up the function exit state change trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Go from State 2 to 1** in the Actions list to define a rule. The Go from State 2 to 1 Resource selection dialog box shown in Figure 8-39 is displayed.



Resource selection

IF Address Point

IN (●) Entire address space

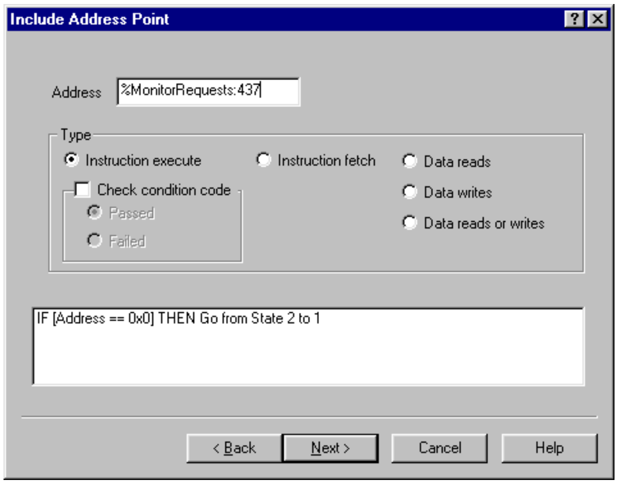
( ) Address range(s) to be defined

THEN Go from State 2 to 1

Figure 8-39 Resource selection for Enable Tracing

2. Click on the drop-down list showing TRUE and select **Address Point**.

3. Click on **Next>**. The Include Address Point dialog box shown in Figure 8-40 on page 8-41 is displayed.



Include Address Point

Address: %MonitorRequests:437

Type: [✓ Instruction execute] Instruction fetch Data reads Data writes Data reads or writes

Check condition code: [✓ Passed] Failed

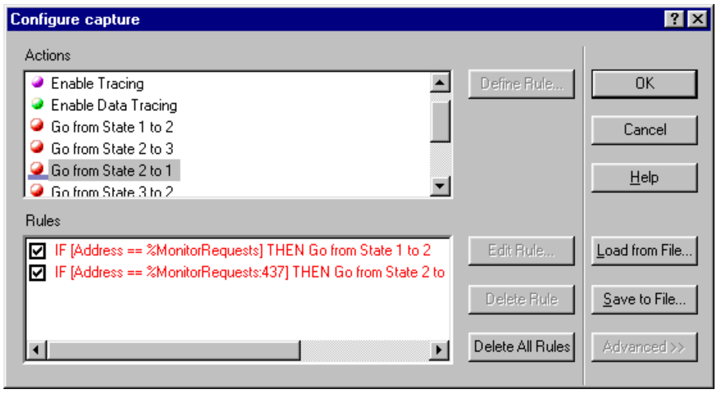
IF [Address == 0x0] THEN Go from State 2 to 1

Figure 8-40 Include Address Point dialog box

4. In the Address field, enter `%MonitorRequests:437`, where 437 is the line number of the function return code.

5. Click **Next>** to display the end of rule dialog box.

6. Click **Finish** to create this rule. The Configure capture dialog box is updated as shown in Figure 8-41.



Configure capture

Actions
- Enable Tracing
- Enable Data Tracing
- Go from State 1 to 2
- Go from State 2 to 3
- Go from State 2 to 1
- Go from State 3 to 2

Rules
- ✓ IF [Address == %MonitorRequests] THEN Go from State 1 to 2
- ✓ IF [Address == %MonitorRequests:437] THEN Go from State 2 to

Figure 8-41 Configure capture dialog box

Setting up the Enable Tracing trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Enable Tracing** in the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-42 is displayed.

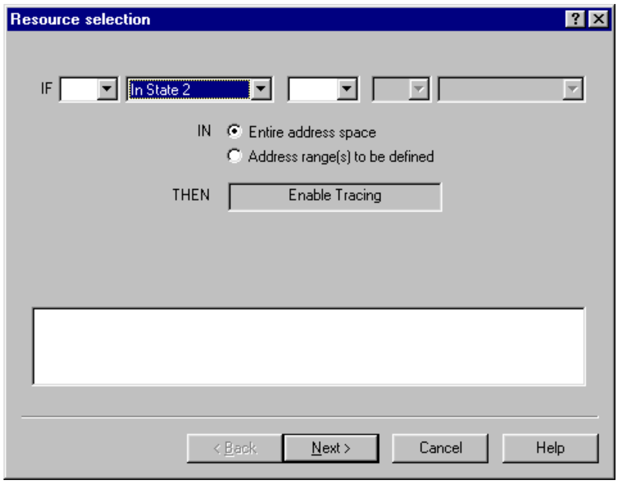
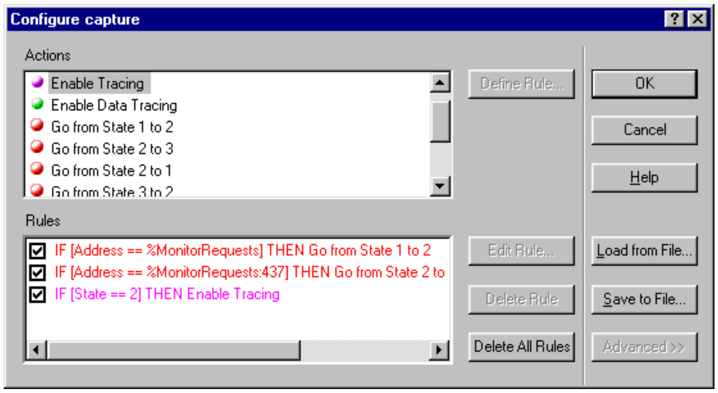


Figure 8-42 Resource selection for Enable Tracing

2. Click on the drop-down list showing TRUE and select **In State 2**.
3. Click **Next>** to display the end of rule dialog box.
4. Click **Finish** to create this rule. The Configure capture dialog box is updated as shown in Figure 8-43.



IF [Address == %MonitorRequests] THEN Go from State 1 to 2
IF [Address == %MonitorRequests:437] THEN Go from State 2 to
IF [State == 2] THEN Enable Tracing

Figure 8-43 Configure capture dialog box

Setting up the Trigger trace capture rule

In this procedure you set up the trigger rule that defines the data that is displayed in the Trace window. Follow these steps:

1. Double-click **Trigger** in the Actions list to define a rule. The Trigger Resource selection dialog box shown in Figure 8-44 is displayed.

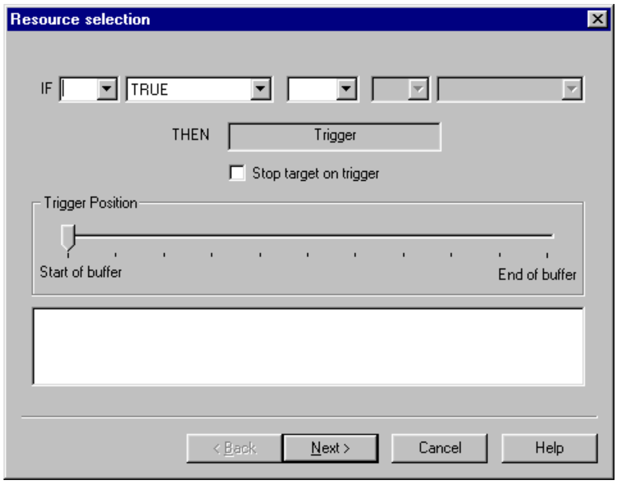


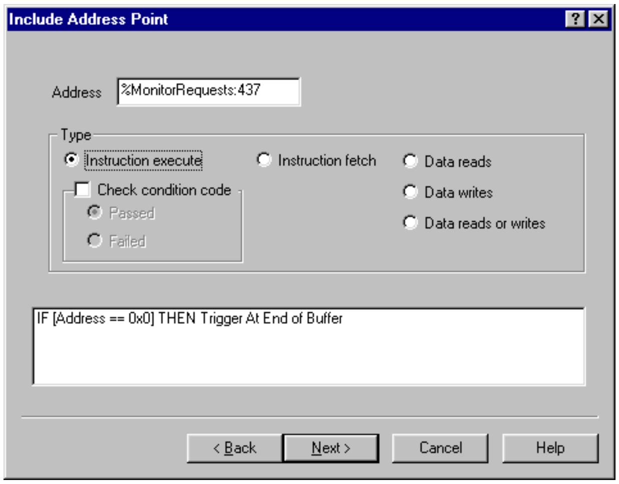
Figure 8-44 Resource selection for Trigger

2. Click on the drop-down list showing TRUE and select **Address Point**.

3. Move the Trigger Position slider to the End of buffer position.

   Setting up the trigger in this way causes the trace buffer to match on an address, which we will define next, and contain all the trace information up to the point the trigger matches.

4. Click on **Next>**. The Address Point dialog box shown in Figure 8-45 on page 8-44 is displayed.



Address: %MonitorRequests:437

Type: [✓ Instruction execute] Data reads, Data writes, Data reads or writes

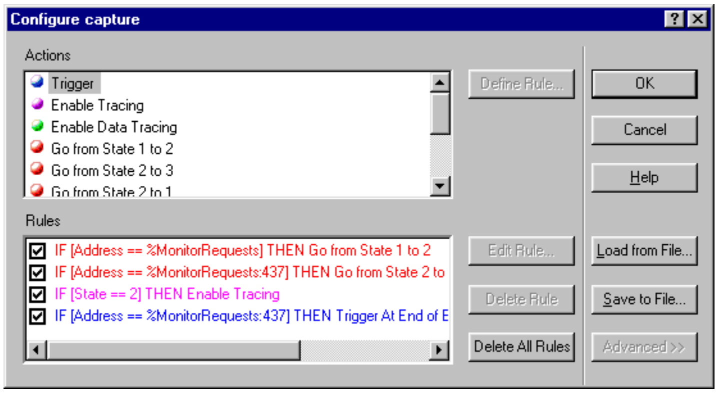
IF [Address == 0x0] THEN Trigger At End of Buffer

Figure 8-45 Include Address Point dialog box

5. In the Address field, enter `%MonitorRequests:437` , where 437 is the line number of the end of the function.

6. Click **Next>** to display the end of rule dialog box.

7. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-46 now has four rules in the Rules list.



IF [Address == %MonitorRequests] THEN Go from State 1 to 2
IF [Address == %MonitorRequests:437] THEN Go from State 2 to
IF [State == 2] THEN Enable Tracing
IF [Address == %MonitorRequests:437] THEN Trigger At End of E

Figure 8-46 Completed Capture dialog box

8. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values
2. Click **View Trace** on the **Trace** menu. The trace output window appears.
3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).
2. By examining the trace output you can try to determine what is happening.

8.5.5 Solution using a start and stop point

This solution to the example is based on tracing the instructions that are executed between entry to and exit from the `MonitorRequests` function. This includes any interrupt handler code as well as functions that `MonitorRequests` calls.

——— **Note** ———

This example cannot be performed using ETM hardware implementing ETM version 1.0 or 1.1.

———————————

This solution requires you to perform the following procedures:

- *Loading the debugger* on page 8-46
- *Setting up the function state change trace capture rules* on page 8-46
- *Tracing the target* on page 8-49
- *Analyzing the trace data* on page 8-49.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
   - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
   - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

   Note

   After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the function state change trace capture rules

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-47.

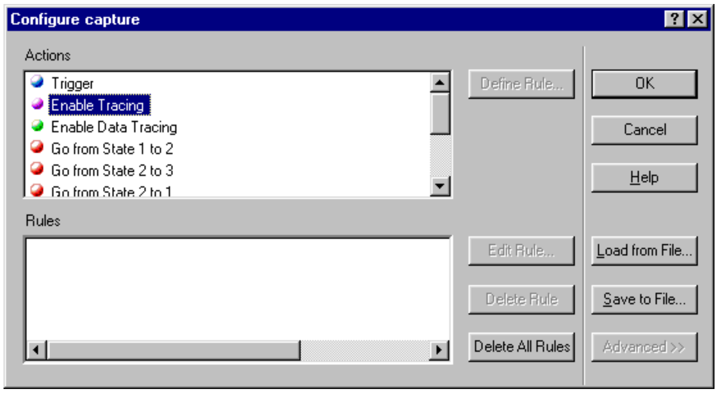
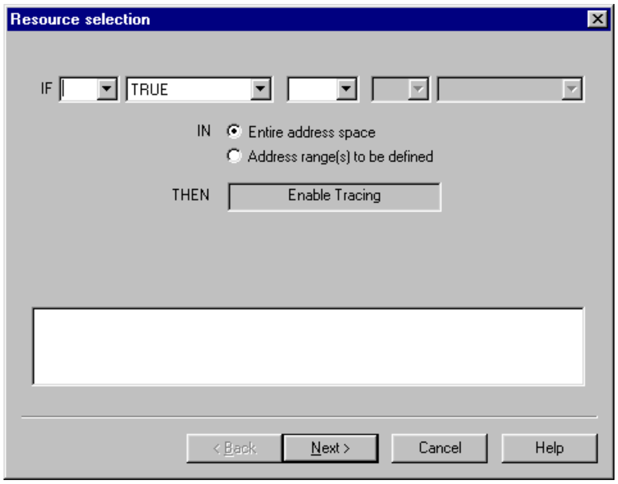


Figure 8-47 Configure capture dialog box

2. Double-click **Enable Tracing** in the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-48 is displayed.



IF | TRUE

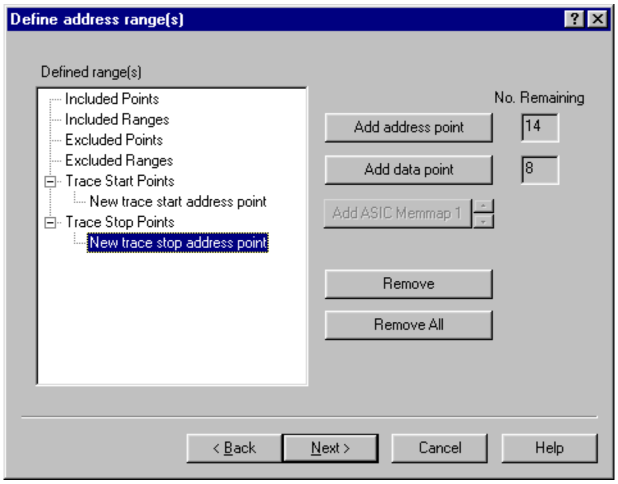
IN [✓ Entire address space] Address range(s) to be defined

THEN Enable Tracing

Figure 8-48 Resource selection for Enable Tracing

3. Click on **Address range(s) to be defined**.

4. Click on **Next>**. The Defined address range(s) dialog box shown in Figure 8-49 is displayed.



Defined range(s)

- Included Points
- Included Ranges
- Excluded Points
- Excluded Ranges
- Trace Start Points
  - New trace start address point
- Trace Stop Points
  - New trace stop address point

No. Remaining

Add address point: 14

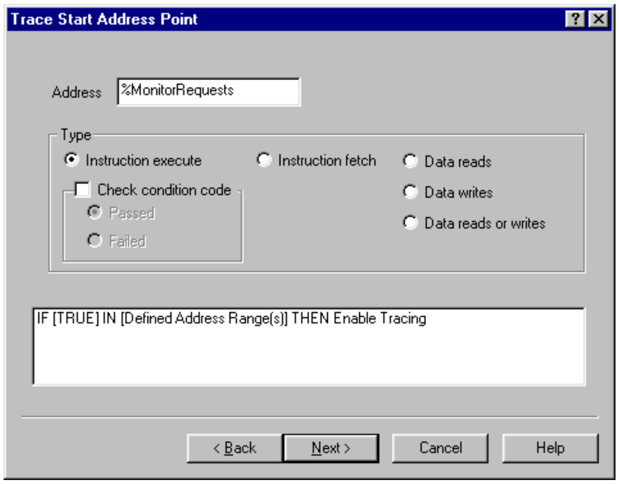
Add data point: 8

Figure 8-49 Defined address range(s) dialog box

5. Select **Trace Start Points** in the Defined range(s) tree control and click on **Add address point**. A list item New start address point appears in the tree control.

6. Select **Trace Stop Points** in the Defined range(s) tree control and click on **Add address point**. A list item New stop address point appears in the tree control.

7. Click on **Next>**. The Trace Start Address Point dialog box shown in Figure 8-50 is displayed.


Address: %MonitorRequests

Type: [✓ Instruction execute] Instruction fetch Data reads Data writes Data reads or writes

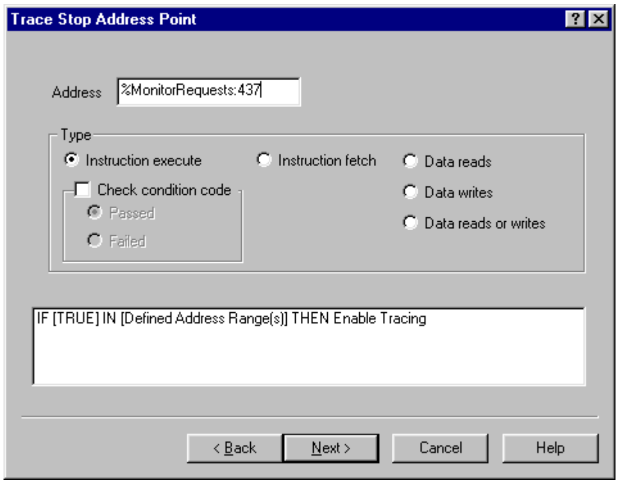
Check condition code: unchecked (Passed, Failed)

IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-50 Trace Start Address Point dialog box

8. In the Address field, enter `%MonitorRequests`.

9. Click on **Next>**. The Trace Stop Address Point dialog box shown in Figure 8-51 is displayed.


Address: %MonitorRequests:437

Type: [✓ Instruction execute] Instruction fetch Data reads Data writes Data reads or writes

Check condition code: unchecked (Passed, Failed)

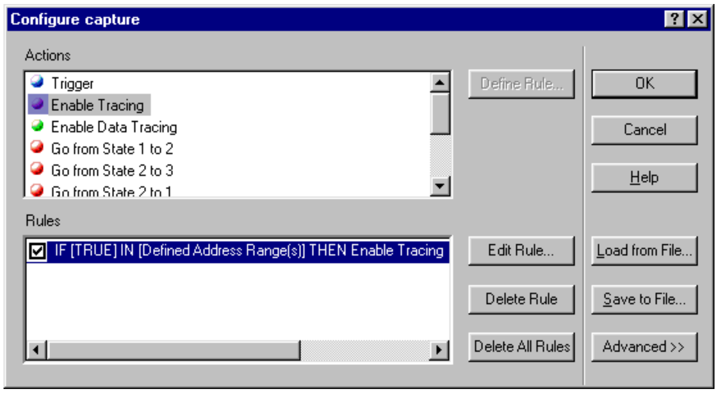
IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-51 Trace Stop Address Point dialog box

10. In the Address field, enter `%MonitorRequests:437`, where 437 is the line number of the end of the function.

11. Click **Next>** to display the end of rule dialog box.

12. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-52 now has one rule in the Rules list.



IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-52 Completed Configure capture dialog box

13. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values

2. Click **View Trace** on the **Trace** menu. The trace output window appears.

3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:

   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)

- link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
- save the trace output to a file (see *Saving the trace output to file* on page 3-23).

2. By examining the trace output you can try to determine what is happening.

8.5.6 Solution using a trace start and stop point and an exclude address range

This solution to the example is based on tracing the instructions that are executed between entry to and exit from the `MonitorRequests` function, and an exclude range to prevent another function, for example an interrupt handler, being traced.

Note

This example cannot be performed using ETM hardware implementing ETM version 1.0 or 1.1.

This solution requires you to perform the following procedures:

- *Loading the debugger*
- *Setting up the function entry state change trace capture rule* on page 8-51
- *Tracing the target* on page 8-49
- *Analyzing the trace data* on page 8-55.

Loading the debugger

In this procedure, you initialize the target and load the debugger with the debug information for the program. Follow these steps:

1. Load the program symbols into the debugger:
   - If the program is not already running, select **Load Image** from the **File** menu to load the executable for which you want to gather trace information.
   - If the program is already running, or loads or runs from ROM, select **Load symbols only…** from the **File** menu.

   Note

   After you load an image into AXD, you might have to change the `top_of_memory` variable value to correspond to the top-of-memory address of your target hardware (see the *Debugger Internals window* section in the *AXD and armsd Debuggers Guide*).

   ---

2. Set the overall configuration of the ETM hardware by selecting **Configure ETM** from the **Trace** menu. The options configured in this dialog box apply to any subsequent tracing you perform (see *Configure ETM* on page 3-7).

Setting up the function entry state change trace capture rule

In this procedure you set up the trace capture rules that define the data that is displayed in the Trace window. Follow these steps:

1. Set the tracing conditions by selecting **Configure Capture** from the **Trace** menu. The **Configure capture** dialog box is shown in Figure 8-53.

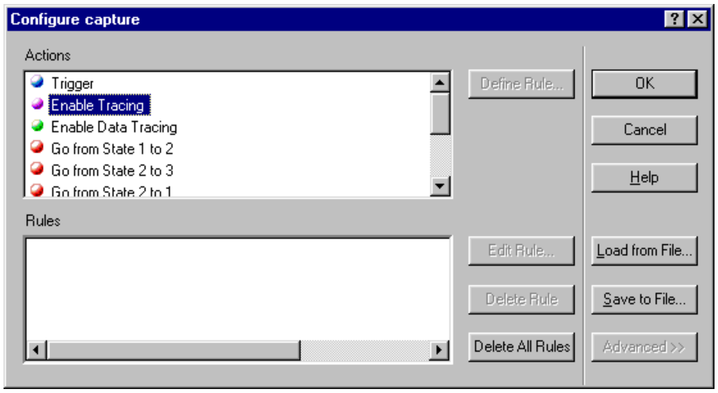


Figure 8-53 Configure capture dialog box

2. Double-click **Enable Tracing** in the Actions list to define a rule. The Enable Tracing Resource selection dialog box shown in Figure 8-54 on page 8-52 is displayed.

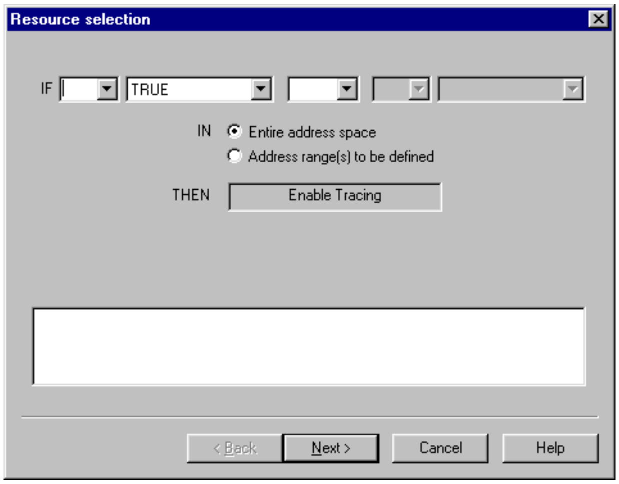


Figure 8-54 Resource selection for Enable Tracing

3. Click on **Address range(s) to be defined**.

4. Click on **Next>**. The Defined address range(s) dialog box shown in Figure 8-55 is displayed.

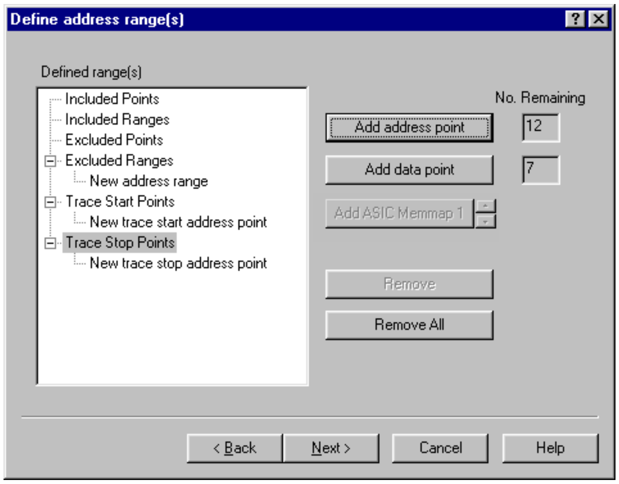


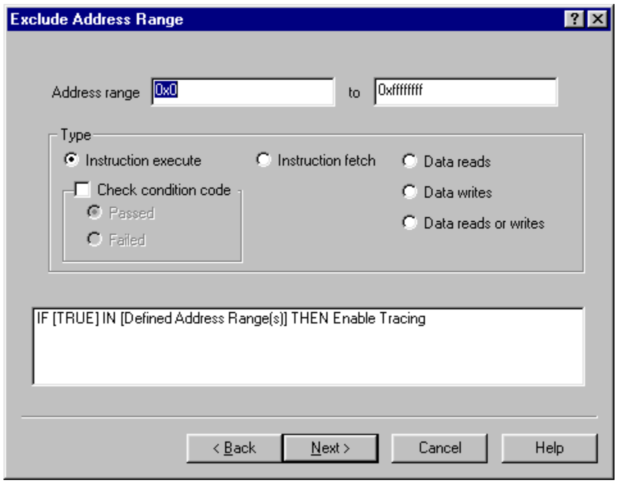
Figure 8-55 Defined address range(s) dialog box

5. Select **Excluded Range** in the tree control and click on **Add address range**. A list item New address range appears in the tree control.

6. Select **Trace Start Points** in the Defined range(s) tree control and click on **Add address point**. A list item New start address point appears in the tree control.

7. Select **Trace Stop Points** in the tree control and click on **Add address point**. A list item New stop address point appears in the tree control.

8. Click on **Next>**. The Excluded Address Range dialog box shown in Figure 8-56 is displayed.



IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

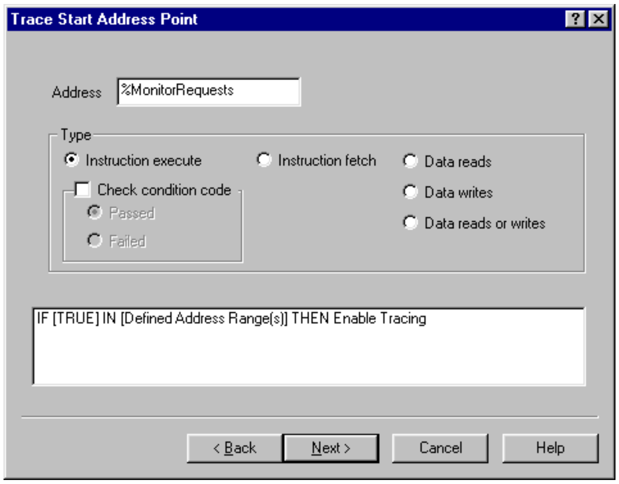
Figure 8-56 Exclude Address Range dialog box

9. Type into the left Address range field `%IntHandler`, and into the right (`to`) field `%IntHandler:437`, where 437 is the line number of the end of the function.

   Note

   If you have more than one function to exclude, you can enter a wider range of addresses into this dialog box or you can use more than one exclude address range.

10. Click on **Next>**. The Trace Start Address Point dialog box shown in Figure 8-57 on page 8-54 is displayed.



Address: %MonitorRequests

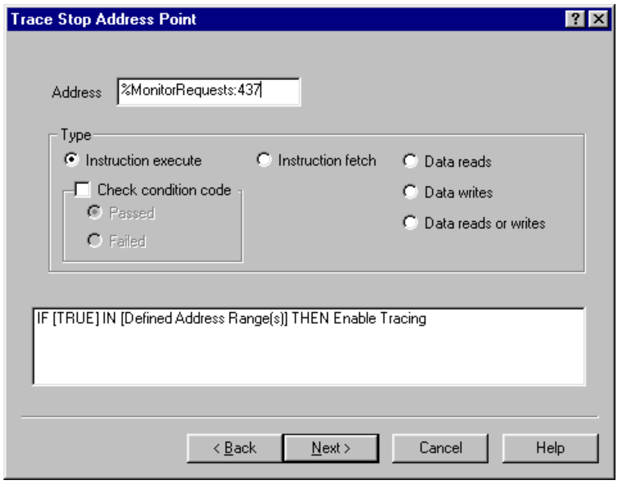
Type: Instruction execute (selected)

IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-57 Trace Start Address Point dialog box

11. In the Address field, enter `%MonitorRequests`.

12. Click on **Next>**. The Trace Stop Address Point dialog box shown in Figure 8-58 is displayed.



Address: %MonitorRequests:437

Type: Instruction execute (selected)

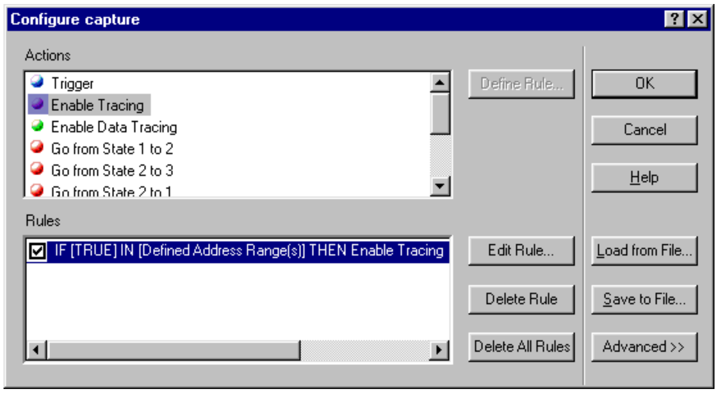
IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-58 Trace Stop Address Point dialog box

13. In the Address field, enter `%MonitorRequests:437`, where 437 is the line number of the end of the function.

14. Click **Next>** to display the end of rule dialog box.

15. Click **Finish** to complete the rule. The Configure capture dialog box shown in Figure 8-59 now has one rule in the Rules list.



IF [TRUE] IN [Defined Address Range(s)] THEN Enable Tracing

Figure 8-59 Completed Configure capture dialog box

16. Click on **OK** to activate the rules.

Tracing the target

To trace the target:

1. A dialog box is displayed asking if you want to Start tracing now?. Click on **Yes**. TDT writes the trace rules and the ETM configuration to the ETM hardware, and instructs the ETM to start collecting trace data based on these values.

2. Select **View Trace** from the **Trace** menu. The trace output window appears.

3. If your program is not already running, select **Go** from the **Execute** menu. During execution, trace information is returned to the Trace window.

Analyzing the trace data

To analyze the trace data:

1. You can now:
   - compare the trace output with its corresponding area of code in the executable (see *Viewing trace output with source code* on page 3-13)
   - link a row of trace output to its corresponding area of memory in the Memory Window (see *Viewing memory at a location derived from trace output* on page 3-20)
   - save the trace output to a file (see *Saving the trace output to file* on page 3-23).

2. By examining the trace output you can try to determine what is happening.

Appendix A
Setting up the Trace Hardware

This appendix describes how to set up the hardware for the configurations of trace hardware supported by ARM Limited. Refer to Chapter 2 *Getting Started* for more information on setting up your trace system.

When setting up the trace capture system, take care not to exceed the timing specifications of the target ETM signals or of the trace capture hardware. See the *ARM Multi-Trace Version 1.0 User Guide* for more information.

This appendix contains the following sections:

A.1 ARM MultiTrace and ARM Multi-ICE

The ARM MultiTrace analyzer is available from ARM Limited and when used with Multi-ICE it forms a complete trace solution. You connect it to the target board using the supplied ribbon cable and adaptor, and to the host workstation using a 10BaseT ethernet.

A.1.1 Setting up

To set up your hardware to use TDT, as shown in Figure A-1, you must:

1. Connect and configure the Multi-ICE interface unit as described in the *Multi-ICE Version 2.0 User Guide*.
2. Connect and configure the MultiTrace interface unit as described in the *MultiTrace User Guide*.

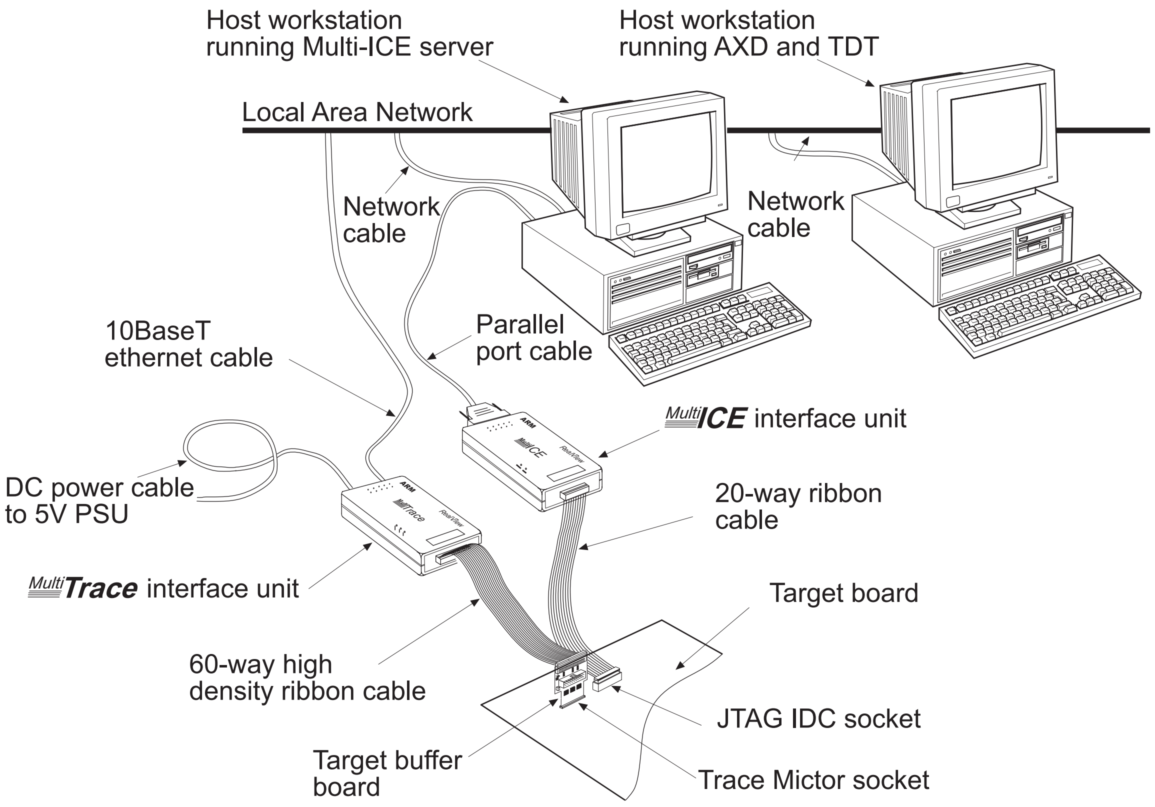


Figure A-1 Connections for Multi-ICE and Multi-Trace using a separate Multi-ICE server

——— **Note** ———

Some target boards have both a Mictor trace connector and an IDC JTAG connector. For boards with only a Mictor connector, an adaptor board must be used to connect Multi-ICE to the Mictor trace connector.

Where a board has both connectors, Multi-ICE must be connected to either the target IDC JTAG connector or the adaptor board IDC JTAG connector. Do not connect JTAG controllers to both connectors.

A.2 ARM Multi-ICE for XScale

For XScale Trace, you must have Multi-ICE version 2.1 or higher. You do not require a trace port analyzer.

A.2.1 Setting up

To set up your hardware to use TDT, as shown in Figure A-2, you must connect and configure the Multi-ICE interface unit as described in the *Multi-ICE Version 2.1 User Guide*.

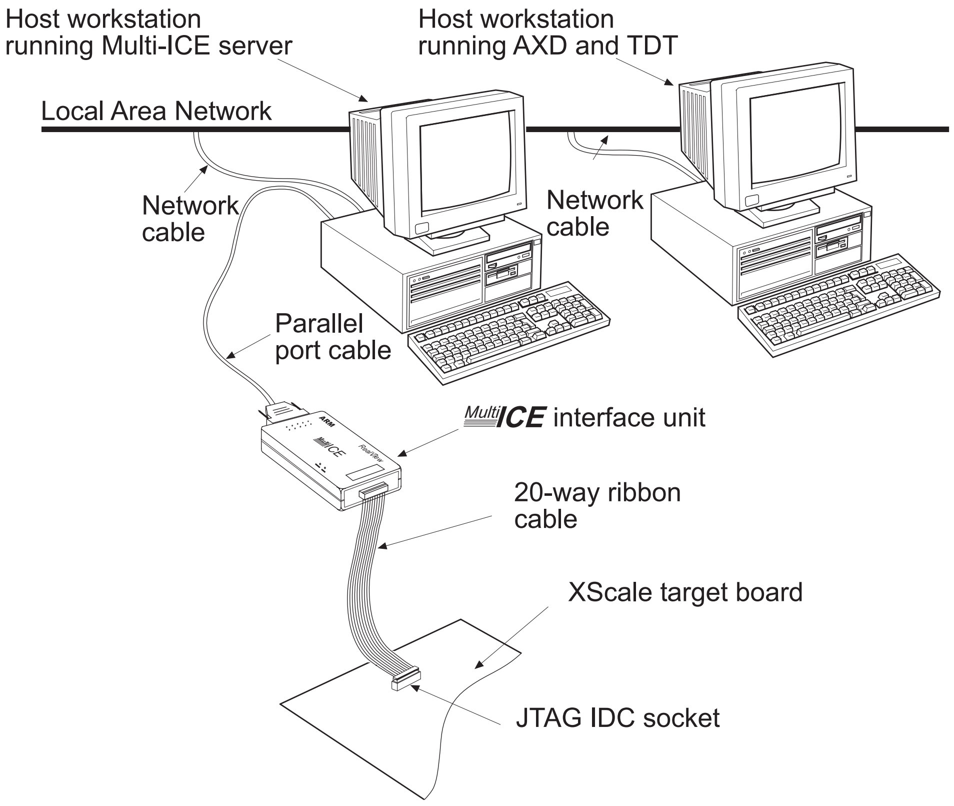


Figure A-2 Connection for Multi-ICE using a separate Multi-ICE server

A.3 Agilent 16600 or 16700 logic analyzer and Emulation module

To set up your hardware and use TDT, as shown in Figure A-3 on page A-6:

1. Set up either the Agilent 16600 or 16700 logic analyzer with the Agilent Emulation module and an Agilent logic analyzer card supporting:
    - sampling rates at least as high as the core clock frequency of the target (at least twice as high if using 4 bit data port for the ETM).
    - a minimum of 21 signal inputs
    - a minimum of 10,000 words (samples) of memory.

    Note

    ARM ADI only supports E5901B modules for logic analyzers. See the ARM ADI documentation for more details.

2. Connect up all hardware as shown in Figure A-3 on page A-6. Power-up all hardware except the ARM target board.
3. You must configure the network setup using the user interface of the logic analyzer. Typically, the network settings are part of the system administration functionality that you can access by clicking **System Admin** in the Logic Analysis System window. See the logic analyzer documentation for more details.

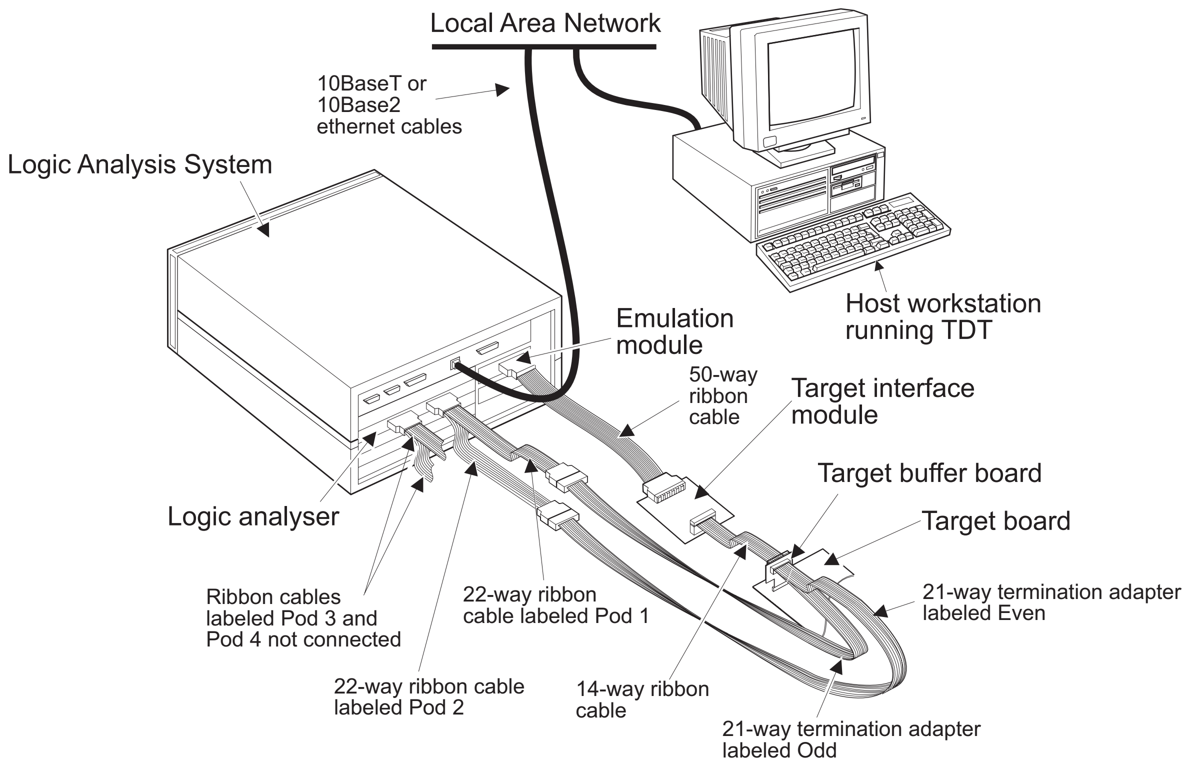

Local Area Network
10BaseT or 10Base2 ethernet cables
Logic Analysis System
Host workstation running TDT
Emulation module
50-way ribbon cable
Target interface module
Target buffer board
Logic analyser
Target board
21-way termination adapter labeled Even
Ribbon cables labeled Pod 3 and Pod 4 not connected
22-way ribbon cable labeled Pod 1
22-way ribbon cable labeled Pod 2
14-way ribbon cable
21-way termination adapter labeled Odd

Figure A-3 Connection diagram for Agilent 16600 or 16700 and the Target Interface Module

4. Check that version numbers are correct for the following:

Analysis system software

Must be `A.01.40.00` or later.

Processor support software

Must be `A.01.40.00` or later.

Emulation module firmware

Must be the following version numbers or later for the various components of the emulation module:

```
E3499B Series Emulation System
   Version:  A.07.64
   Location:  Generics
E3459B ARM7/9 JTAG Emulator
   Version:   B.02.04
E3459Q ARM7/9 Trace Port Analyzer
   Version:  Q.01.00.
```

To view the software versions, select the **Software Install** tab in the System Administration Tools window, and click **List**. If you require an upgrade for the software, contact Agilent technical support by following the instructions at the web site `www.agilent.com`.

To view the firmware version, right-click on the **Emulator** icon (in the Logic Analysis System or Workspace window) and select the **Update Firmware** option to display the current version. If you require an upgrade:

a. Download the appropriate files from the web site `http://www.agilent.com`.

b. Copy the files to the hard disk of the logic analyzer, placing them in the directory `/hplogic/firmware/run_cntrl`.

c. Click **Update Firmware** in the Update Firmware window. The upgrade occurs automatically.

Note

The Agilent web site provides more details on this process.

5. Configure the analyzer software. During this process you must record the following information so that you can set up TDT to match:

   - the number of target signals you are capturing, either wide (16-bit) or narrow (8-bit)
   - the clock definition, either single edge or double edge.

   The provided analyzer configuration files assume full rate (single edge) clocking, and no multiplexing or demultiplexing of the data. If you want to use half-rate clocking, multiplexing, or demultiplexing, you will have to modify the configurations that are loaded into the analyzer.

   You can configure the analyzer in either of the following ways:

   - Click **File Manager** in the Logic Analysis System window. Load in an appropriate generic configuration file. You can then save this back to a configuration file specific to the logic analyzer and appropriate slot.

     Note

     The following logic analyzer configuration files are available:

     — `CARMETM_9`, corresponding to an 8-bit port width (with timestamps)
     — `CARMETM_10`, corresponding to an 8-bit port width
     — `CARMETM_11`, corresponding to a 16-bit port width (with timestamps)
     — `CARMETM_12`, corresponding to a 16-bit port width.

     Configurations using an 8-bit port width are also valid for use with a 4-bit ETM trace port.

You must contact Agilent to obtain a CD-ROM software update for logic analyzers. This update always contains the latest configuration files needed for ETM tracing.

- Click **Setup Assistant** (if available) in the Logic Analysis System window. With this method, the process of loading a configuration file is split into a series of simple steps. For each step, you are prompted for information that allows the Setup Assistant to autogenerate a configuration file with your specifications. See the logic analyzer documentation for more details.

  Note

  You must select the **Setup Assistant** option in the Logic Analyzer Configuration dialog box when configuring AXD.

  ---

6. Power-up the ARM target board. Your logic analyzer hardware is now configured for use with the TDT.
7. You must now configure AXD to enable tracing.

For detailed information on setting up the Agilent 16600 or 16700 logic analyzer and Emulation module, see the Agilent documentation, which can be accessed from the website `http://www.agilent.com`. See *Other publications* on page xi for details.

A.4 Agilent 16600 or 16700 logic analyzer and Multi-ICE

The difference between this configuration and the Agilent-only configuration is that the Agilent emulation module is not required in the analyzer. It is replaced by the Multi-ICE interface unit.

To set up your hardware and use TDT:

1. Set up either the Agilent 16600 or 16700 logic analyzer with an Agilent logic analyzer card supporting:
   - sampling rates at least as high as the core clock frequency of the target (at least twice as high if using 4-bit data port for the ETM)
   - a minimum of 21 signal inputs
   - a minimum of 10,000 words (samples) of memory.
2. Connect a Multi-ICE interface unit as described in the *Multi-ICE Version 2.0 User Guide*.
3. Connect the Multi-ICE JTAG cable between the interface unit and the IDC JTAG connector on the target board, and connect the logic analyzer Mictor connector from ports Pod 1 and Pod 2 to the trace port Mictor connector.

   If you do not have separate IDC JTAG and Mictor trace connectors on the target board, you must use an adaptor board plugged into the trace connector. The board can be obtained from your logic analyzer supplier.

   Where a board has both connectors, Multi-ICE must be connected to either the target IDC JTAG connector or the adaptor board IDC JTAG connector. Do not connect JTAG controllers to both connectors.
4. Connect up the rest of the hardware as shown in Figure A-4 on page A-11.
5. Power-up all hardware except the ARM target board.
6. Start the Multi-ICE server software on the host workstation and verify that autoconfiguration of the hardware works.
7. Configure the network interface of the logic analyzer. Typically, the network settings are part of the system administration functionality that you can access by clicking **System Admin** in the Logic Analysis System window. See the logic analyzer documentation for more details.
8. Check that version numbers are correct for the following:

   Analysis system software

   Must be `A.01.40.00` or later.

Processor support software

Must be `A.01.40.00` or later.

To view the software versions, select the **Software Install** tab in the System Administration Tools window, and click **List**. If you require an upgrade for the software, contact Agilent technical support by following the instructions at the web site `www.agilent.com`.

9. Configure the analyzer software. During this process you must record the following information so that you can set up TDT to match:

   - the number of target signals you are capturing, either wide (16-bit) or narrow (8-bit)
   - the clock definition, either single edge or double edge.

   The provided analyzer configuration files assume full rate (single edge) clocking, and no multiplexing or demultiplexing of the data. If you want to use half-rate clocking, multiplexing, or demultiplexing, you will have to modify the configurations that are loaded.

   You can configure the analyzer in either of the following ways:

   - Click **File Manager** in the Logic Analysis System window. Load in an appropriate generic configuration file. You can then save this back to a configuration file specific to the logic analyzer and appropriate slot.

     Note

     The following logic analyzer configuration files are available:

     — `CARMETM_9`, corresponding to an 8-bit port width (with timestamps)

     — `CARMETM_10`, corresponding to an 8-bit port width

     — `CARMETM_11`, corresponding to a 16-bit port width (with timestamps)

     — `CARMETM_12`, corresponding to a 16-bit port width.

     Configurations using an 8-bit port width are also valid for use with a 4-bit ETM trace port.

     You must contact Agilent to obtain a CD-ROM software update for logic analyzers. This update contains the latest configuration files needed for ETM tracing.

   - Click **Setup Assistant** (if available) in the **Logic Analysis System** window. With this method, the process of loading a configuration file is split into a series of simple steps. For each step, you are prompted for information that allows the Setup Assistant to autogenerate a configuration file with your specifications. See the logic analyzer documentation for more details.

Note

If you do this, you must select the **Setup Assistant** option in the **Logic Analyzer Configuration** dialog box when configuring AXD.

10. Power up the ARM target board. Your logic analyzer hardware is now configured for use with the TDT.

11. You must now configure AXD to enable tracing.

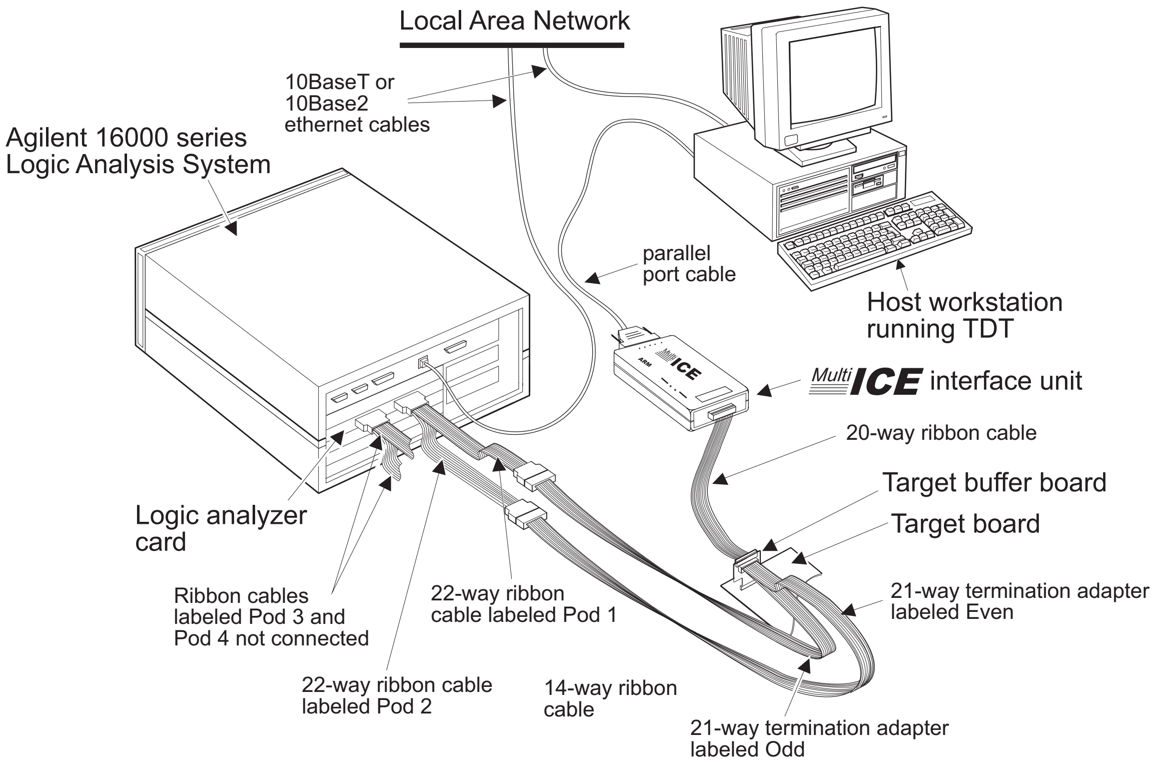

Local Area Network
10BaseT or 10Base2 ethernet cables
Agilent 16000 series Logic Analysis System
parallel port cable
Host workstation running TDT
MultiICE interface unit
20-way ribbon cable
Target buffer board
Target board
Logic analyzer card
Ribbon cables labeled Pod 3 and Pod 4 not connected
22-way ribbon cable labeled Pod 1
21-way termination adapter labeled Even
22-way ribbon cable labeled Pod 2
14-way ribbon cable
21-way termination adapter labeled Odd

Figure A-4 Agilent Trace Port Analyzer and Multi-ICE Version 2.0

For detailed information on setting up the Agilent 16600 or 16700 logic analyzer, see the Agilent documentation, which can be accessed from the website `http://www.agilent.com`. See *Other publications* on page xi for details.

A.5 Tektronix TLA 600 or TLA 700 logic analyzer and Multi-ICE

To set up your hardware to use TDT, as shown in Figure A-5 on page A-13:

1. Install the Multi-ICE software on the analyzer, using the CD-ROM unit on the rear panel of the analyzer, as described in the *Multi-ICE User Guide Version 2.0*.
2. Set up the analyzer as described in the *Tektronix TLA Logic Analyzer ARM ETM Support Package Instructions*.

   In particular, connect the Mictor socket of a Tektronix P6434 Mass Termination Probe to the target board, and the two module ends of the cable to the analyzer. Plug the A3 and A2 module end into the A3 and A2 connector on the analyzer, and the A1 and A0 module end into the A1 and A0 connector on the analyzer.

   The Tektronix cable is labeled as shown in Table A-1. There are four probe names and colors because Tektronix make four equivalent but differently labeled connectors.

Table A-1 Tektronix P6434 cable labeling

Probe	Colors	Description
A	Tan/orange	A3 and A2 Pin1 (tan) side, A1 and A0 Pin38 (orange) side
B	Blue/yellow	B3 and B2 Pin1 (tan) side, B1 and B0 Pin38 (orange) side
C	White/gray	C3 and C2 Pin1 (tan) side, C1 and C0 Pin38 (orange) side
D	Green/violet	D3 and D2 Pin1 (tan) side, D1 and D0 Pin38 (orange) side

3. Connect the analyzer port to the Mictor trace connector on the target board.
4. Verify that the Dragonfly Software TLA COM Server application is running on the Tektronix analyzer.
5. Connect the Multi-ICE interface unit to the parallel port on the rear panel of the analyzer.
6. Connect the Multi-ICE JTAG cable between the interface unit and the IDC JTAG connector on the target board.

   If you do not have separate IDC JTAG and Mictor trace connectors on the target board, you must use an adaptor board plugged into the trace connector. The board can be obtained from your logic analyzer supplier.

   Where a board has both connectors, Multi-ICE must be connected to either the target IDC JTAG connector or the adaptor board IDC JTAG connector. Do not connect JTAG controllers to both connectors.

7. Power up the ARM target board.
8. Start the Multi-ICE server software using **Start** → **Programs** → **ARM Multi-ICE v2.0** → **Multi-ICE Server**.
9. Select **Auto-configure** from the Multi-ICE server **File** menu and verify that the target board can be autoconfigured.
10. Your logic analyzer hardware is now configured for use with TDT.

    You must now configure AXD to enable tracing.

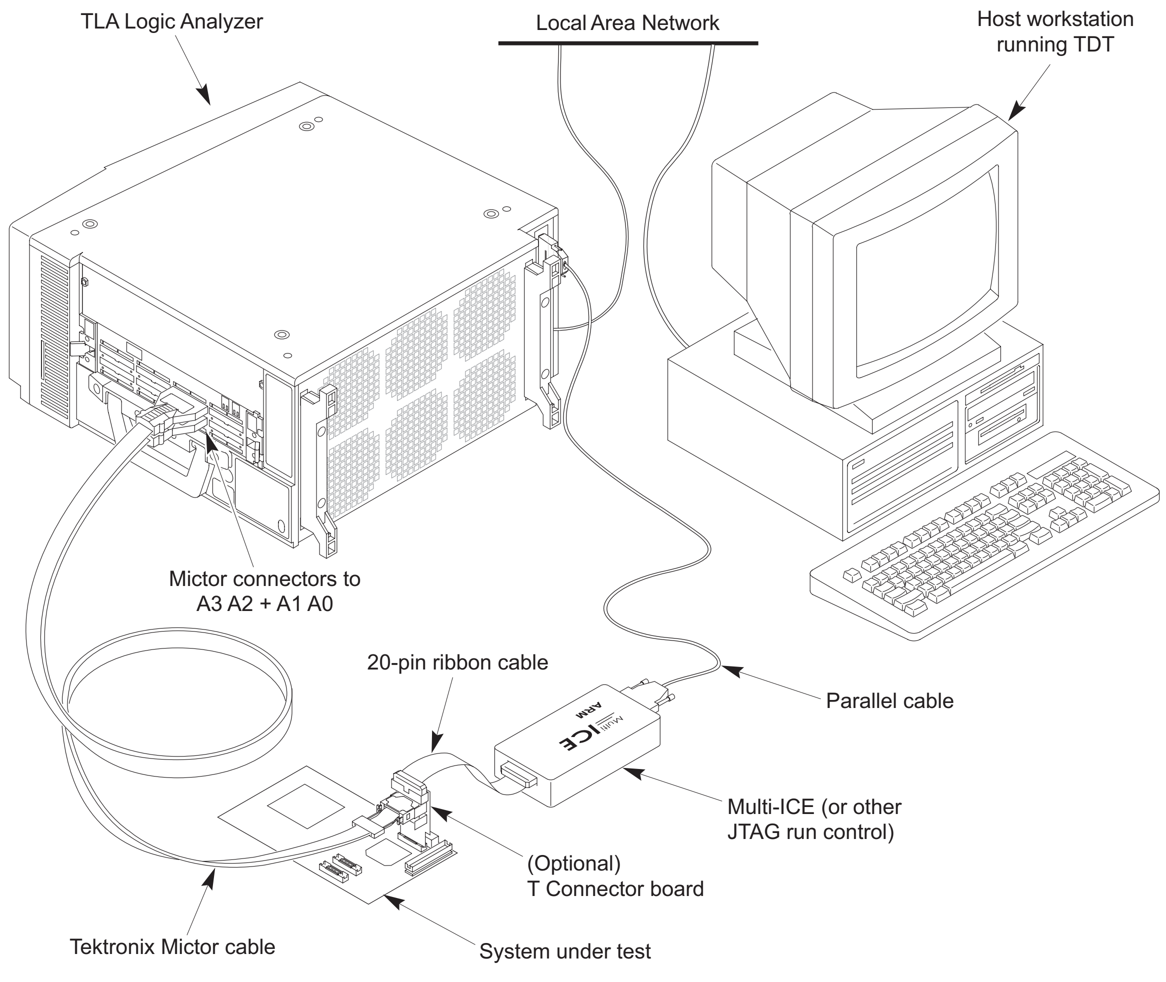

TLA Logic Analyzer
Local Area Network
Host workstation running TDT
Mictor connectors to A3 A2 + A1 A0
20-pin ribbon cable
Parallel cable
ARM Multi ICE
Multi-ICE (or other JTAG run control)
(Optional) T Connector board
Tektronix Mictor cable
System under test

Figure A-5 Tektronix TLA 700 analyzer and Multi-ICE

A.6 ARMulator ETM emulation

The ADS v1.2 ARMulator is extended with files supplied with TDT to support emulation of an ARM7TDMI core or ARM9TDMI core with an ETM. See the *ADS Debug Target Guide* for more information about configuring the ARMulator.

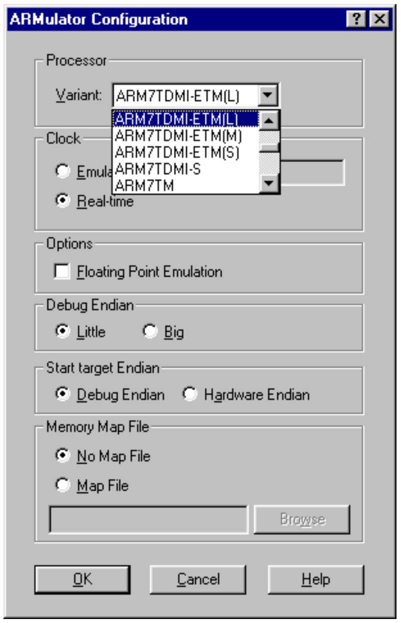
A.6.1 Setting up

After TDT 1.2 is installed, you configure the ARMulator to emulate an ETM by selecting `ARMUL` (`ARMulate.dll`) as the target in the AXD **Configure Target** dialog box as shown in Figure A-6.

Select one of the processors with a name ending in:

`ETM(L)`	A large ETM.
`ETM(M)`	A medium size ETM.
`ETM(S)`	A small ETM.

Refer to *The Embedded Trace Macrocell (ETM)* on page 1-6 for information on the differences between the sizes.


ARMulator Configuration

Processor

Variant: ARM7TDMI-ETM(L)

ARM7TDMI-ETM(L)
ARM7TDMI-ETM(M)
ARM7TDMI-ETM(S)
ARM7TDMI-S
ARM7TM

Clock

Emula [✓ Real-time]

Options

☐ Floating Point Emulation

Debug Endian

[✓ Little] Big

Start target Endian

[✓ Debug Endian] Hardware Endian

Memory Map File

[✓ No Map File] Map File

Figure A-6 Selecting an ETM enabled processor in ARMulator

Appendix B
Setting up the Trace Software

This appendix describes how to set up the software for the configurations of trace that are supported by ARM Limited. These instructions assume you have set up the hardware as described in Appendix A *Setting up the Trace Hardware*. Refer to Chapter 2 *Getting Started* for more information on setting up your trace system.

This appendix contains the following sections:

B.1 ARM MultiTrace and ARM Multi-ICE

This section describes how to set up TDT to connect to a combination of ARM MultiTrace and ARM Multi-ICE. Refer to the *Multi-ICE Installation Guide* or the *Multi-ICE User Guide* for more information about Multi-ICE, and to the *MultiTrace User Guide* for more information on setting up the MultiTrace unit.

You must:

1. Install and set up the Multi-ICE server.
2. Configure AXD.
3. Configure Multi-ICE and MultiTrace.

To install and set up the Multi-ICE server:

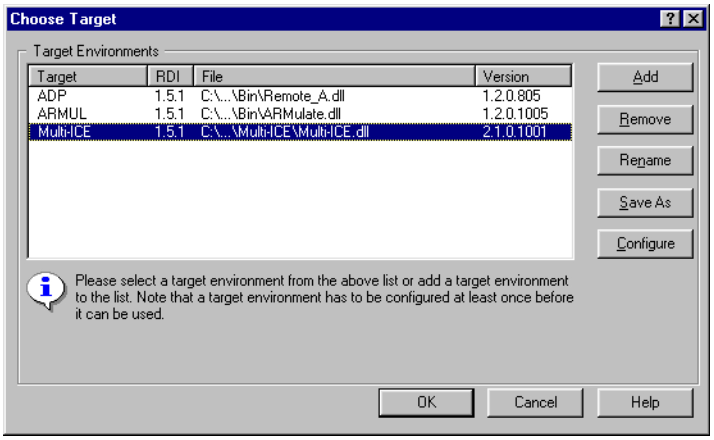
1. Install Multi-ICE Version 2.0 on both the workstation you are debugging on and the workstation that the Multi-ICE interface unit connects to.
2. Configure the Multi-ICE server on the workstation that the Multi-ICE interface unit is connected to.

To configure Multi-ICE and MultiTrace:

1. Select **Start** → **Programs** → **ARM Developer Suite v1.2** → **AXD**.
2. Select **Options** → **Configure Target** in AXD. This displays the target configuration dialog box shown in Figure B-1 on page B-3.

   Note

   The list of targets and the version numbers shown in Target Environments might vary from those shown in Figure B-1 on page B-3.



Target	RDI	File	Version
ADP	1.5.1	C:\...\Bin\Remote_A.dll	1.2.0.805
ARMUL	1.5.1	C:\...\Bin\ARMulate.dll	1.2.0.1005
Multi-ICE	1.5.1	C:\...\Multi-ICE\Multi-ICE.dll	2.1.0.1001

Please select a target environment from the above list or add a target environment to the list. Note that a target environment has to be configured at least once before it can be used.

Figure B-1 The Debugger Configuration dialog box

3. Select Multi-ICE:
   - If **Multi-ICE** is present in the Target Environments list, select it.
   - If **Multi-ICE** is not present in the Target Environments list, click **Add** and select `Multi-ICE.dll` from the Multi-ICE install directory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

——— Note ———

If you are using Multi-ICE™ as your JTAG interface unit, then for ETM tracing you must have Multi-ICE Version 2.0 or higher installed. For XScale tracing, you must have Multi-ICE Version 2.1 or higher installed. When you install TDT 1.2, Multi-ICE version 2.x will be automatically upgraded to version 2.x.1 when you next run AXD and connect to Multi-ICE.

———————————

4. Click **Configure** to show the ARM Multi-ICE 2.0.1 configuration dialog box (Figure B-2 on page B-4).

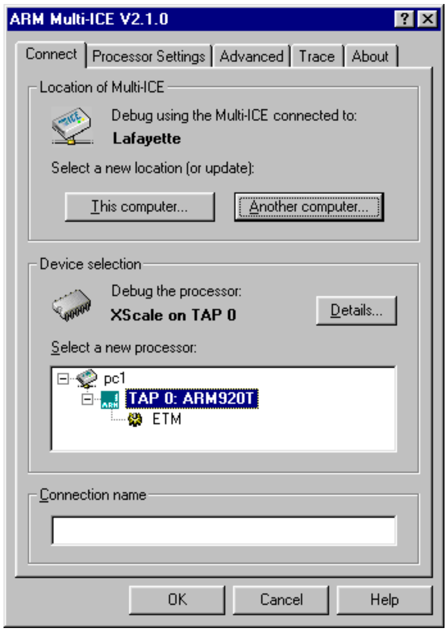


Figure B-2 Multi-ICE configuration dialog box

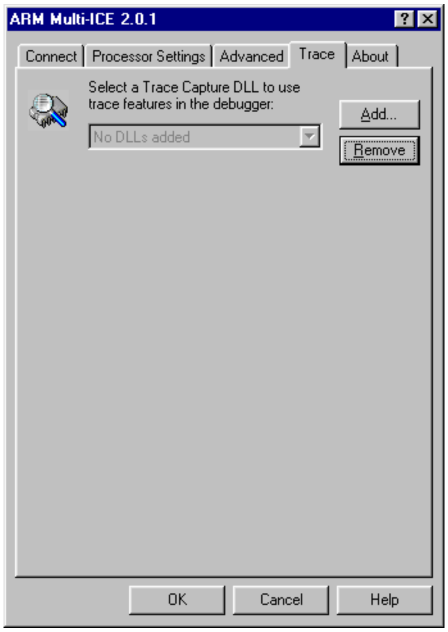
5. Depending on the location of the Multi-ICE server that you are using, click on either **This computer** or **Another computer**.

   If you select **Another computer**, in the subsequent dialog box type in the name of the remote workstation, or locate it using the tree control. Click on **OK**.

6. Select the ARM processor that you are tracing from the list shown in **Select a new processor**.

7. Optionally, enter your name in the **Connection name** text field. This shows up in the Multi-ICE server window and can help if you are sharing the server with others.

8. Click the **Trace** tab to make it visible (Figure B-3 on page B-5).



Select a Trace Capture DLL to use trace features in the debugger:

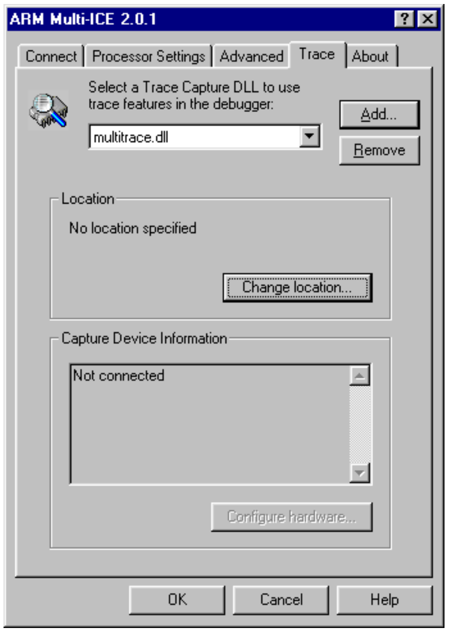
No DLLs added

Figure B-3 Multi-ICE configuration dialog box showing the Trace tab

The tab contains the controls required to configure the trace control software. The **Select a Trace Capture DLL...** drop-down list contains the names of the currently available trace capture drivers. These drivers read the trace information from the ETM and translate it into the format required by the debugger.

9. Select the MultiTrace driver from the **Select a Trace Capture DLL...** drop-down list. This specifies the driver file that is used to control the trace capture device:
   - If `multitrace.dll` is present in the drop-down list, select it.
   - If `multitrace.dll` is not present in the drop-down list, click **Add...** and select `multitrace.dll` from the MultiTrace installation directory. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

10. With `multitrace.dll` selected, the MultiTrace configuration controls are added to the Multi-ICE Trace tab (Figure B-4 on page B-6). You must configure MultiTrace as described in the MultiTrace User Guide.



Select a Trace Capture DLL to use trace features in the debugger:

multitrace.dll

Location

No location specified

Capture Device Information

Not connected

Figure B-4 MultiTrace configuration

11. Click **OK** to exit the Multi-ICE dialog box.

12. Click **OK** to exit the AXD Target Configuration dialog box. AXD restarts with the new configuration. AXD is now configured for tracing.

——— **Note** ———

You must exit AXD or use **File** → **Save Session** to permanently save this configuration.

———————

B.2 ARM Multi-ICE for XScale

This section describes how to set up the TDT to connect to ARM Multi-ICE. Refer to the *Multi-ICE Installation Guide* or the *Multi-ICE User Guide* for more information about Multi-ICE.

You must:

1. Install and set up the Multi-ICE server
2. Configure AXD.

To install and set up the Multi-ICE server:

1. Install Multi-ICE Version 2.1 on both the workstation you are debugging on and the workstation that the Multi-ICE interface unit connects to.
2. Configure the Multi-ICE server on the workstation that the Multi-ICE interface unit is connected to.

To configure Multi-ICE:

1. Select **Start** → **Programs** → **ARM Developer Suite v1.2** → **AXD**.
2. Select **Options** → **Configure Target** in AXD. This displays the target configuration dialog box shown in Figure B-1 on page B-3.

   ——— **Note** ———

   The list of targets and the version numbers shown in Target Environments might vary from those shown in Figure B-1 on page B-3.

   ————————

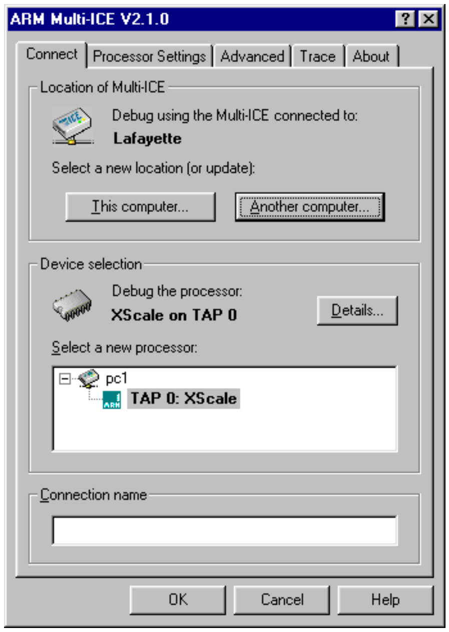
3. Select Multi-ICE:
   - If **Multi-ICE** is present in the Target Environments list, select it.
   - If **Multi-ICE** is not present in the Target Environments list, click **Add** and select `Multi-ICE.dll` from the Multi-ICE install directory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

   ——— **Note** ———

   If this is the first time you have used Multi-ICE Version 2.1 in AXD after you installed TDT 1.2, a message box is displayed informing you that Multi-ICE has been upgraded to version 2.1.1. TDT 1.2 requires enhanced Multi-ICE DLL software to operate. The original DLL is still in the Multi-ICE installation directory but has been renamed.

   ————————

4. Click **Configure** to show the ARM Multi-ICE 2.1 configuration dialog box (Figure B-2 on page B-4).



Location of Multi-ICE
Debug using the Multi-ICE connected to:
Lafayette
Select a new location (or update):

Device selection
Debug the processor:
XScale on TAP 0
Select a new processor:
pc1
TAP 0: XScale

Connection name

Figure B-5 Multi-ICE configuration dialog box (XScale)

5. Depending on the location of the Multi-ICE server that you are using, click on either **This computer** or **Another computer**.

   If you select **Another computer**, in the subsequent dialog box type in the name of the remote workstation, or locate it using the tree control. Click on **OK**.

6. Select the ARM processor that you are tracing from the list shown in **Select a new processor**.

7. Optionally, enter your name in the **Connection name** text field. This shows up in the Multi-ICE server window and can help if you are sharing the server with others.

8. Click **OK** to exit the Multi-ICE dialog box.

9. Click **OK** to exit the AXD Target Configuration dialog box. AXD restarts with the new configuration. AXD is now configured for tracing.

——— **Note** ———

You must exit AXD or use **File** → **Save Session** to permanently save this configuration.

B.3 Agilent 16600 or 16700 Logic Analyzer and Emulation Probe

Note

ARM ADI is not supplied with ADS 1.2 and must be purchased separately.

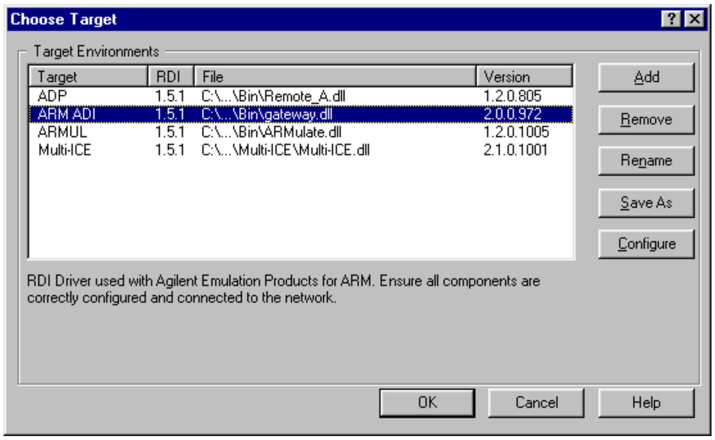
This section describes how to set up TDT to connect an Agilent Logic Analyzer and an Emulation Probe. You must have already set up the logic analyzer software as part of the hardware setup.

To configure the Agilent 16600 or 16700 analyzer:

1. Select **Start** → **Programs** → **ARM Developer Suite v1.2** → **AXD**.
2. Select **Options** → **Configure Target**, as shown in Figure B-6.

   Note

   The list of targets and the version numbers shown in Target Environments might vary from those shown in the figure.



Target	RDI	File	Version
ADP	1.5.1	C:\...\Bin\Remote_A.dll	1.2.0.805
ARM ADI	1.5.1	C:\...\Bin\gateway.dll	2.0.0.972
ARMUL	1.5.1	C:\...\Bin\ARMulate.dll	1.2.0.1005
Multi-ICE	1.5.1	C:\...\Multi-ICE\Multi-ICE.dll	2.1.0.1001

RDI Driver used with Agilent Emulation Products for ARM. Ensure all components are correctly configured and connected to the network.

Figure B-6 Debugger Configuration dialog box

3. Select Gateway:
   - If **ARM ADI** is present in the Target Environments list, select it.
   - If **ARM ADI** is not present in the Target Environments list, click **Add** and select `gateway.dll` from the ADS …`\bin` subdirectory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

4. Click on **Configure** to display the Gateway Configuration dialog box (Figure B-7).

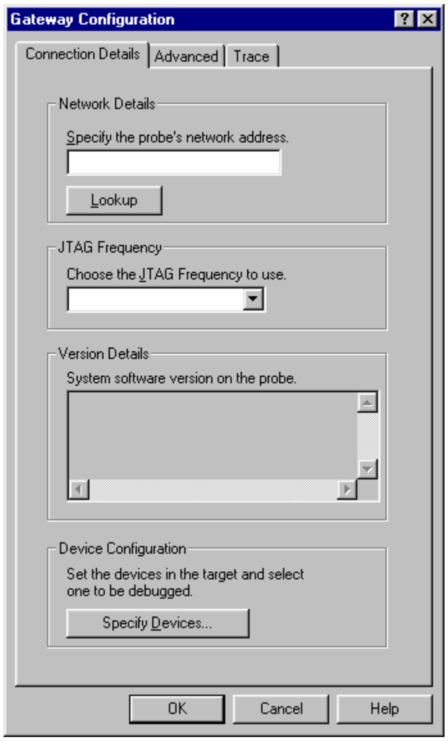
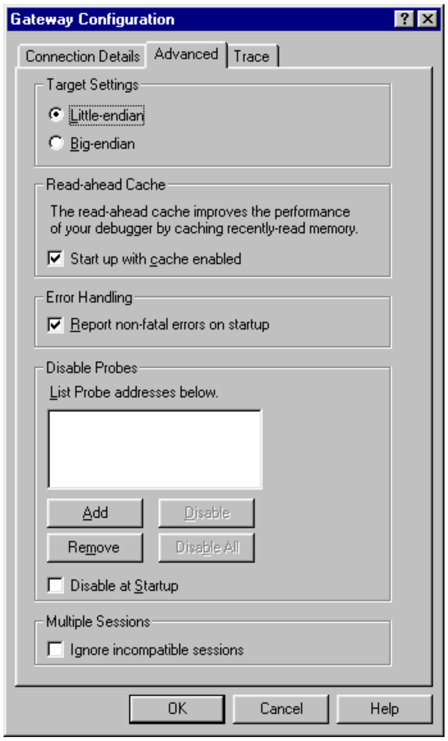

Gateway Configuration
Connection Details
Advanced
Trace
Network Details
Specify the probe's network address.
Lookup
JTAG Frequency
Choose the JTAG Frequency to use.
Version Details
System software version on the probe.
Device Configuration
Set the devices in the target and select one to be debugged.
Specify Devices...
OK
Cancel
Help

Figure B-7 Gateway Configuration dialog box, Connection Details tab

5. Ensure that the **Connection Details** tab is displayed.
6. Fill in the **Network Details** for the Emulation Probe, and click **Lookup**.
7. Click on the **Advanced** tab (Figure B-8 on page B-12).


Gateway Configuration

Connection Details | Advanced | Trace

Target Settings

(●) Little-endian
( ) Big-endian

Read-ahead Cache

The read-ahead cache improves the performance of your debugger by caching recently-read memory.

[✓] Start up with cache enabled

Error Handling

[✓] Report non-fatal errors on startup

Disable Probes

List Probe addresses below.

Add | Disable | Remove | Disable All

[ ] Disable at Startup

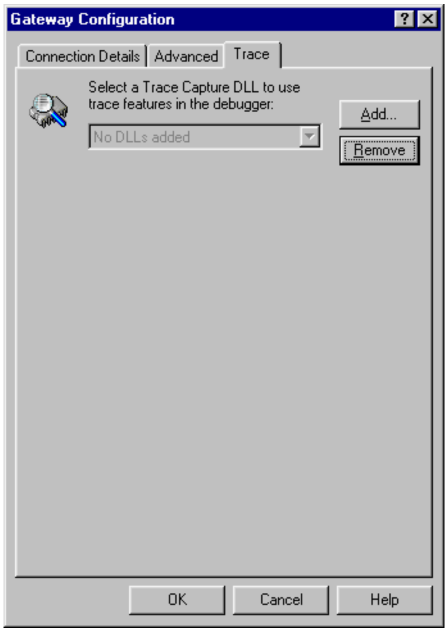
Multiple Sessions

[ ] Ignore incompatible sessions

Figure B-8 Gateway Configuration dialog box, Advanced tab

8. If your target is running in big endian mode, click on **Big-endian**.

9. Click on the **Trace** tab (Figure B-9 on page B-13).



Select a Trace Capture DLL to use trace features in the debugger:

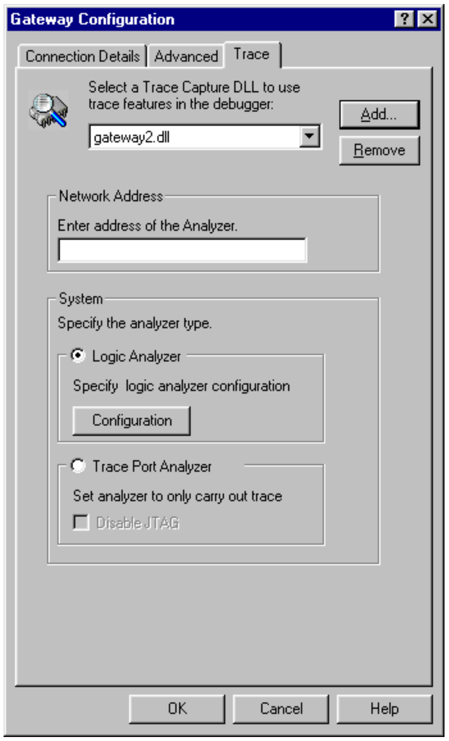
No DLLs added

Figure B-9 Gateway Configuration dialog box, Trace tab

10. Specify the Gateway2 driver in the **Select a Trace Capture DLL...** drop-down list. This specifies the driver file that is used to control the trace capture device:

    - If `Gateway2.dll` is present in the drop-down list, select it.
    - If `Gateway2.dll` is not present in the drop-down list, click **Add...** and select `Gateway2.dll` from the ADS …`\bin` subdirectory. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

    The configured Gateway **Trace** tab is shown in Figure B-10 on page B-14.



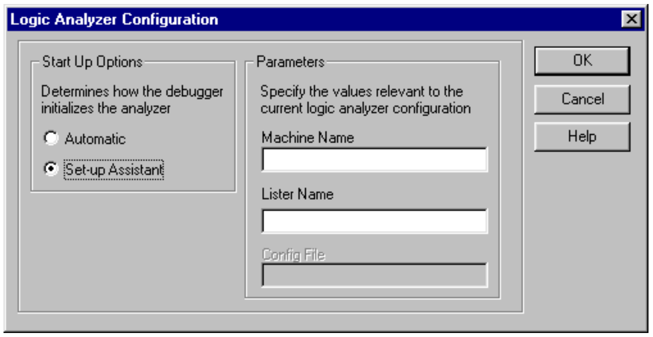
Select a Trace Capture DLL to use trace features in the debugger: gateway2.dll

Network Address
Enter address of the Analyzer.

System
Specify the analyzer type.
[✓ Logic Analyzer] Trace Port Analyzer
Logic Analyzer: Specify logic analyzer configuration — Configuration
Trace Port Analyzer: Set analyzer to only carry out trace — Disable JTAG

Figure B-10 Configuring Gateway2

11. In the System section of the dialog box, click on **Logic Analyzer**.
12. Enter the **Network Address** of the Agilent Logic Analyzer.
13. Click **Configuration** to display the Logic Analyzer Configuration dialog box (Figure B-11).



Start Up Options
Determines how the debugger initializes the analyzer
Automatic [✓ Set-up Assistant]

Parameters
Specify the values relevant to the current logic analyzer configuration
Machine Name
Lister Name
Config File

Figure B-11 Logic Analyzer Configuration dialog box

14. Select the appropriate startup option to indicate the level of initialization carried out by the debugger:
    - Select the **Automatic** option if you want the debugger to ensure, at start-up, that the logic analyzer is fully initialized to carry out tracing. In this case, you must specify the appropriate Machine Name, Lister Name, and Config File in the dialog box.

      You must specify the full directory path to the configuration file.
    - Select the **Set-up Assistant** option if you do not want the configuration file to be loaded by the debugger. The other parameters (Machine Name and Lister Name) are initialized with the given values. This mode is appropriate only when you are loading a configuration file from the logic analyzer user interface, which can be done using either of the following:
      - the Setup Assistant
      - the File Manager tool.

    If you are using the default logic analyzer configuration files provided by Agilent, you must:
    - set the machine name as `ARM ETM Analyzer`
    - set the lister name as `ETM Data`.

    Note

    The default logic analyzer configuration files cannot be loaded directly by the analyzer. Instead, using the file manager tool on the analyzer user interface, you must load each file, save the configuration back to an appropriate filename, and specify this new name as the configuration to load.

15. Click **OK** successively to exit each of the dialog boxes.
16. Select **Configure ETM** from the TDT **Trace** menu. Use the information you recorded when setting up the analyzer to set up the ETM as follows:
    a. Set the **Trace data width** to `16 bit` for a `wide` analyzer connection, otherwise set it to `8 bit` or `4 bit` (depending on the target hardware).
    b. Enable **Half-rate clocking** for double edge clocking, otherwise disable it.
    c. Click **OK** to accept the changes.

    AXD is now configured for tracing.

For instructions on performing a trace capture, see *Tracing procedure* on page 3-4. AXD restarts with the new configuration when you exit the configuration dialog box. You must exit AXD to permanently save these new configuration settings.

For detailed information on setting up the Agilent 16600 or 16700 Logic Analyzer and Emulation Probe, see the Agilent documentation, which can be accessed from the website `http://www.agilent.com`. See *Other publications* on page xi for details.

For detailed information on setting up and using ARM ADI, see the *ARM ADI User Guide*.

B.4 Agilent 16600 or 16700 Logic Analyzer and ARM Multi-ICE

——— **Note** ———

ARM ADI is not supplied with ADS 1.2 and must be purchased separately.

———————————

This section describes how to set up TDT to connect to a combination of Agilent Logic Analyzer and ARM Multi-ICE. You must have already set up the logic Analyzer software as part of the hardware setup.

Refer to the *ARM ADI User Guide* for details on setting up ARM ADI.

You must:

1. Install and set up the Multi-ICE server.
2. Configure AXD.
3. Configure Multi-ICE and ARM ADI.

To install and set up the Multi-ICE server:

1. Install Multi-ICE Version 2.0 on both the workstation you are debugging on and the workstation that the Multi-ICE interface unit connects to.

2. Configure the Multi-ICE server on the workstation that the Multi-ICE interface unit is connected to.

To configure the Agilent 16600 or 16700 Analyzer with Multi-ICE:

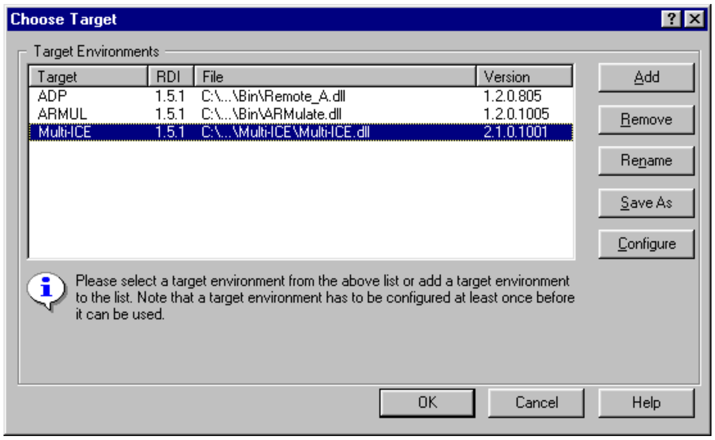
1. Select **Start** → **Programs** → **ARM Developer Suite v1.2** → **AXD**.

2. Select **Options** → **Configure Target** in AXD. This displays the target configuration dialog box shown in Figure B-12 on page B-18.

   ——— **Note** ———

   The list of targets and the version numbers shown in Target Environments might vary from those shown in Figure B-12 on page B-18.

   ———————————



Target	RDI	File	Version
ADP	1.5.1	C:\...\Bin\Remote_A.dll	1.2.0.805
ARMUL	1.5.1	C:\...\Bin\ARMulate.dll	1.2.0.1005
Multi-ICE	1.5.1	C:\...\Multi-ICE\Multi-ICE.dll	2.1.0.1001

Please select a target environment from the above list or add a target environment to the list. Note that a target environment has to be configured at least once before it can be used.

Figure B-12 Debugger Configuration dialog box

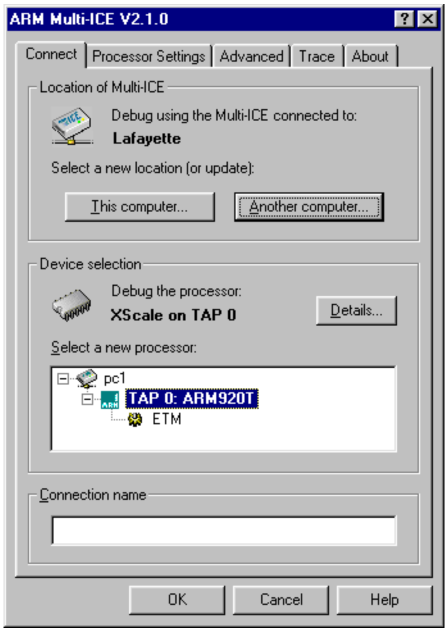
3. Select Multi-ICE:
   - If **Multi-ICE** is present in the Target Environments list, select it.
   - If **Multi-ICE** is not present in the Target Environments list, click **Add** and select `Multi-ICE.dll` from the Multi-ICE install directory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

——— **Note** ———

If this is the first time you have used Multi-ICE Version 2.0 in AXD after you installed TDT 1.2, a message box is displayed informing you that Multi-ICE has been upgraded to version 2.0.1. TDT 1.2 requires enhanced Multi-ICE DLL software to operate. The original DLL is still in the Multi-ICE installation directory but has been renamed.

———————

4. Click on **Configure** to display the ARM Multi-ICE 2.0.1 configuration dialog box (Figure B-13 on page B-19).



Location of Multi-ICE

Debug using the Multi-ICE connected to:
Lafayette

Select a new location (or update):

Device selection

Debug the processor:
XScale on TAP 0

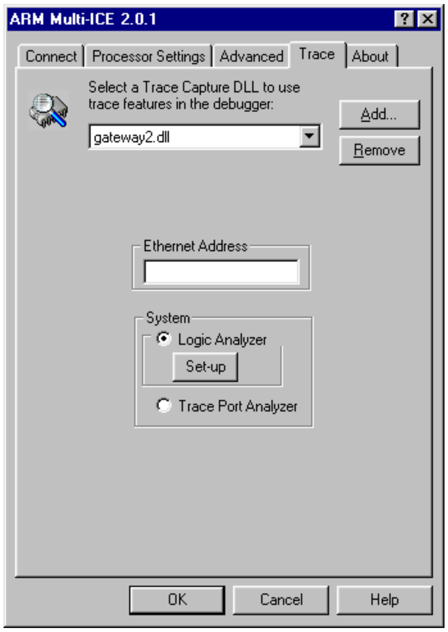
Select a new processor:

pc1
TAP 0: ARM920T
ETM

Connection name

Figure B-13 Multi-ICE configuration dialog box

5. Ensure the **Trace** tab is displayed, as shown in Figure B-14 on page B-20.


ARM Multi-ICE 2.0.1

Select a Trace Capture DLL to use trace features in the debugger:

gateway2.dll

Ethernet Address

System

Logic Analyzer [✓]

Trace Port Analyzer

Figure B-14 Multi-ICE configuration dialog box showing the Trace tab

6. Specify the ARM ADI driver in the **Select a Trace Capture DLL...** drop-down list. This specifies the driver file that is used to control the trace capture device and to carry out operations such as start and stop tracing:
   - If `Gateway2.dll` is present in the drop-down list, select it.
   - If `Gateway2.dll` is not present in the drop-down list, click **Add** and select `Gateway2.dll` from the ADS `…\bin` subdirectory. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

7. Click **Set-up** to display the Logic Analyzer Configuration dialog box. You must also select the appropriate startup option to indicate the level of initialization carried out by the debugger:
   - Select the **Automatic** option if you want the debugger to ensure, at start-up, that the logic analyzer is fully initialized to carry out tracing. In this case, you must specify the appropriate machine name, lister name, and configuration file name in the dialog box.

     You must specify the full directory path to the configuration file.

- Select the **Set-up Assistant** option if you do not want the configuration file to be loaded by the debugger. The other parameters (lister name and machine name) are initialized with the given values. This mode is appropriate only when you are loading a configuration file from the logic analyzer user interface, which can be done using either of the following:
  - — the Setup Assistant
  - — the File Manager tool.

If you are using the default logic analyzer configuration files provided by Agilent, you must:

- set the machine name as `ARM ETM Analyzer`
- set the lister name as `ETM Data`.

——— **Note** ———

The default logic analyzer configuration files cannot be loaded directly by the analyzer. Instead, using the file manager tool on the analyzer user interface, you must load each file, save the configuration back to an appropriate filename, and specify this new name as the configuration to load.

8. Click **OK** successively to exit each of the dialog boxes. AXD is now configured for tracing. AXD restarts with the new configuration when you exit the Debugger Configuration dialog box. You must exit AXD to permanently save these new configuration settings.

For detailed information on setting up the Agilent 16600 or 16700 Logic Analyzer, see the Agilent documentation, which can be accessed from the website `http://www.agilent.com` . See *Other publications* on page xi for details.

B.5 Agilent Trace Port Analyzer and Agilent Emulation Probe

Note

ARM ADI is not supplied with ADS 1.2 and must be purchased separately.

This section describes how to set up TDT to connect to a combination of the Agilent Trace Port Analyzer and Emulation Probe. You must have already set up the logic analyzer software as part of the hardware setup.

You must:

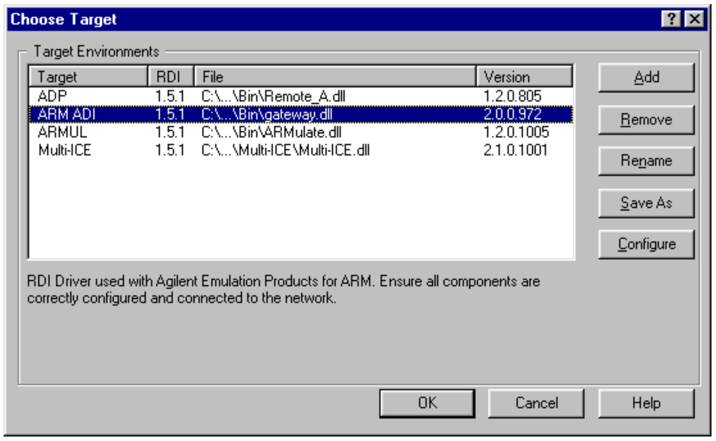
1. Configure AXD.
2. Configure ARM ADI.

To configure the Agilent Analyzer and probe:

1. Select **Start** → **Programs** → **ARM Developer Suite v1.2** → **AXD**.
2. Select **Options** → **Configure Target**, as shown in Figure B-15.

   Note

   The list of targets and the version numbers shown in **Target Environments** might vary from those shown in the figure.



Target	RDI	File	Version
ADP	1.5.1	C:\...\Bin\Remote_A.dll	1.2.0.805
ARM ADI	1.5.1	C:\...\Bin\gateway.dll	2.0.0.972
ARMUL	1.5.1	C:\...\Bin\ARMulate.dll	1.2.0.1005
Multi-ICE	1.5.1	C:\...\Multi-ICE\Multi-ICE.dll	2.1.0.1001

RDI Driver used with Agilent Emulation Products for ARM. Ensure all components are correctly configured and connected to the network.

Figure B-15 Debugger Configuration dialog box

3. Select Gateway:
   - If **ARM ADI** is present in the Target Environments list, select it.

- If **ARM ADI** is not present in the Target Environments list, click **Add** and select `gateway.dll` from the ADS …`\bin` subdirectory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

4. Click on **Configure** to display the Gateway Configuration dialog box (Figure B-16).

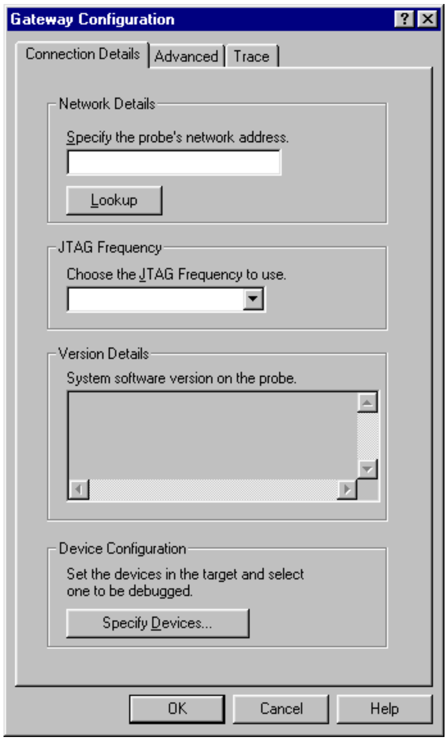
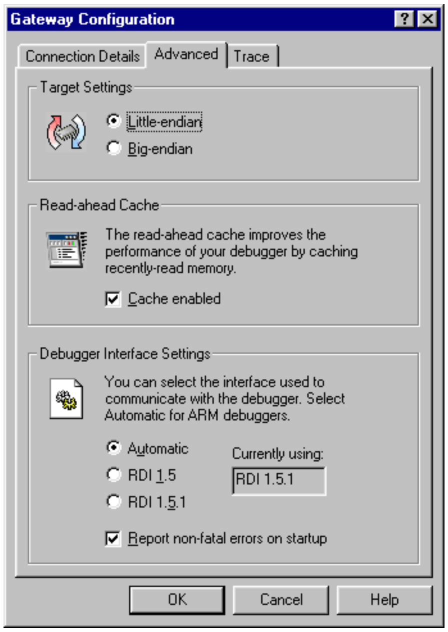

Gateway Configuration
Connection Details
Advanced
Trace
Network Details
Specify the probe's network address.
Lookup
JTAG Frequency
Choose the JTAG Frequency to use.
Version Details
System software version on the probe.
Device Configuration
Set the devices in the target and select one to be debugged.
Specify Devices...
OK
Cancel
Help

Figure B-16 Gateway Configuration dialog box

5. Fill in the network address for the Emulation Probe, and click **Lookup**.

6. If your target is running in big-endian mode:
   a. Click **Advanced** to display the advanced configuration tab, as shown in Figure B-17 on page B-24.
   b. Click **Big-endian**.


Gateway Configuration

Connection Details | Advanced | Trace

Target Settings

(●) Little-endian
( ) Big-endian

Read-ahead Cache

The read-ahead cache improves the performance of your debugger by caching recently-read memory.

[✓] Cache enabled

Debugger Interface Settings

You can select the interface used to communicate with the debugger. Select Automatic for ARM debuggers.

(●) Automatic
( ) RDI 1.5
( ) RDI 1.5.1

Currently using: RDI 1.5.1

[✓] Report non-fatal errors on startup

Figure B-17 Gateway Configuration dialog, Advanced tab

7. Click **Trace** to display the trace configuration tab, as shown in Figure B-18 on page B-25.

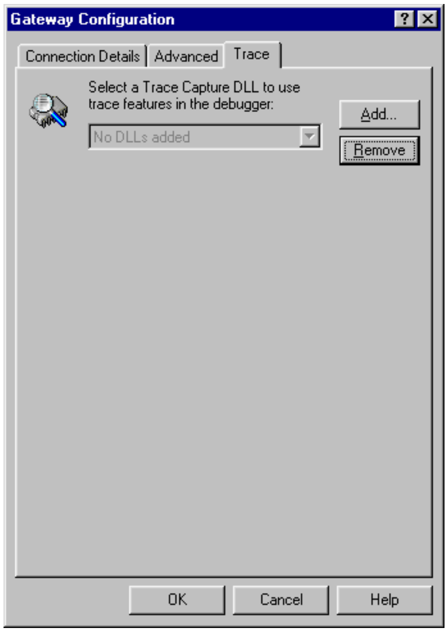
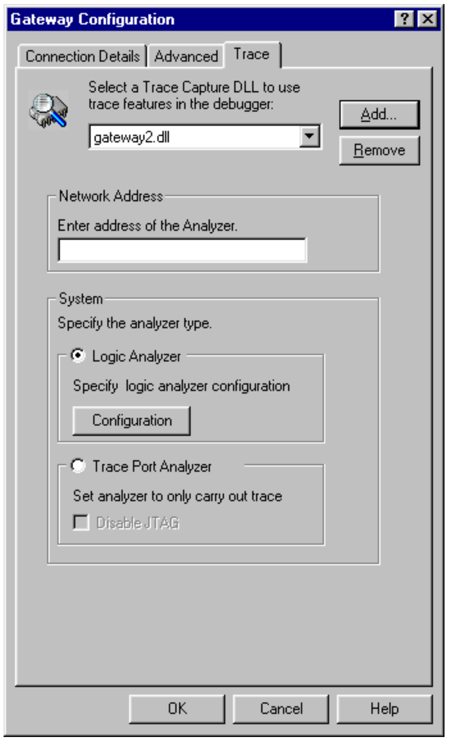


Figure B-18 Gateway Configuration dialog, trace tab

8. Specify the Gateway2 driver in the **Select a Trace Capture DLL...** drop-down list. This specifies the driver file that is used to control the trace capture device and to carry out operations such as start and stop tracing:

   - If `Gateway2.dll` is present in the drop-down list, select it.
   - If `Gateway2.dll` is not present in the drop-down list, click **Add...** and select `Gateway2.dll` from the ADS …`\bin` subdirectory. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

   The configured dialog box is shown in Figure B-19 on page B-26.


Gateway Configuration

Connection Details | Advanced | Trace

Select a Trace Capture DLL to use trace features in the debugger:

gateway2.dll

Add... | Remove

Network Address

Enter address of the Analyzer.

System

Specify the analyzer type.

(●) Logic Analyzer

Specify logic analyzer configuration

Configuration

( ) Trace Port Analyzer

Set analyzer to only carry out trace

☐ Disable JTAG

OK | Cancel | Help

Figure B-19 Configuring Gateway2

9. Click **Trace Port Analyzer**.
10. Enter the network address of the Agilent Trace Port Analyzer in the **Network Address** field.
11. Click **OK** successively to exit each of the dialog boxes. AXD is now configured for tracing. For instructions on performing a trace capture, see *Tracing procedure* on page 3-4. AXD restarts with the new configuration when you exit the Debugger Configuration dialog box. You must exit AXD to permanently save these new configuration settings.

Note

For details on configuring for an Agilent Emulation Probe, see the *Setting Up the Trace Port Analyzer* chapter of the *Trace Port Analysis for ARM ETM User's Guide*.

For detailed information on setting up the Agilent Trace Port Analyzer and Agilent Emulation Probe, see the Agilent documentation, which can be accessed from the website `http://www.agilent.com` . See *Other publications* on page xi for details.

For detailed information on setting up and using ARM ADI, see the *ARM ADI User Guide*.

B.6 Tektronix TLA 600 or TLA700 and ARM Multi-ICE

This section describes how to set up TDT to connect to a combination of ARM Multi-ICE and the Dragonfly Software Tektronix trace capture agent. Refer to the *Multi-ICE Installation Guide* or to the *Multi-ICE User Guide* for more information about Multi-ICE, and to the *Tektronix TLA Logic Analyzer ARM ETM Support Package Instructions* for more information on setting up the Tektronix interface software.

You must:

1. Install and set up the Multi-ICE server.
2. Configure AXD.
3. Configure Multi-ICE and the Dragonfly Software trace capture agent.

To install and set up the Multi-ICE server:

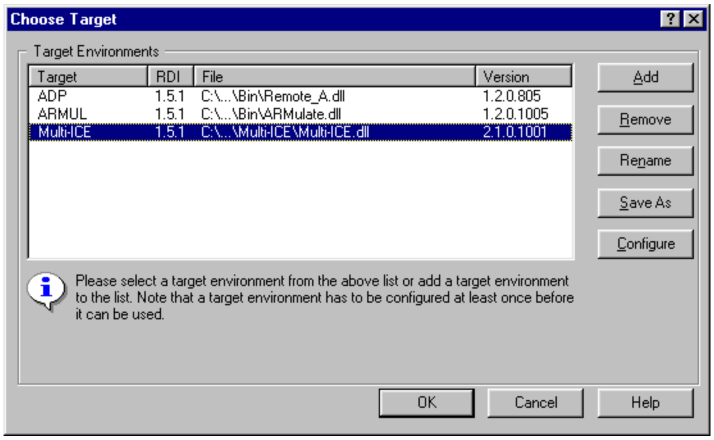
1. Install Multi-ICE Version 2.0 on both the workstation you are debugging on and the workstation that the Multi-ICE interface unit connects to.
2. Configure the Multi-ICE server on the workstation that the Multi-ICE interface unit is connected to.

To configure Multi-ICE and Dragonfly Software trace capture agent:

1. Select **Start** → **Programs** → **ARM Developer Suite v1.2** → **AXD**.
2. Select **Options** → **Configure Target** in AXD. This displays the target configuration dialog box shown in Figure B-20 on page B-29.

   Note

   The list of targets and the version numbers shown in Target Environments may vary from those shown in Figure B-20 on page B-29.



Target	RDI	File	Version
ADP	1.5.1	C:\...\Bin\Remote_A.dll	1.2.0.805
ARMUL	1.5.1	C:\...\Bin\ARMulate.dll	1.2.0.1005
Multi-ICE	1.5.1	C:\...\Multi-ICE\Multi-ICE.dll	2.1.0.1001

Please select a target environment from the above list or add a target environment to the list. Note that a target environment has to be configured at least once before it can be used.

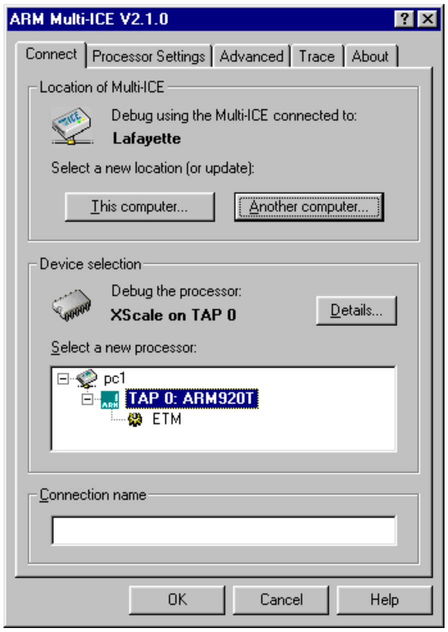
Figure B-20 Debugger Configuration dialog box

3. Select Multi-ICE:
   - If **Multi-ICE** is present in the Target Environments list, select it.
   - If **Multi-ICE** is not present in the Target Environments list, click **Add** and select `Multi-ICE.dll` from the Multi-ICE install directory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

Note

If this is the first time you have used Multi-ICE Version 2.0 in AXD after you installed TDT 1.2, a message box is displayed informing you that Multi-ICE has been upgraded to version 2.0.1. TDT 1.2 requires enhanced Multi-ICE DLL software to operate. The original DLL is still in the Multi-ICE installation directory but has been renamed.

4. With Multi-ICE selected, click **Configure** to display the ARM Multi-ICE 2.0.1 configuration dialog box (Figure B-21 on page B-30).



Debug using the Multi-ICE connected to:
Lafayette

Select a new location (or update):

Debug the processor:
XScale on TAP 0

Select a new processor:

- pc1
  - TAP 0: ARM920T
    - ETM

Figure B-21 Multi-ICE configuration dialog box

5. Depending on the location of the Multi-ICE server that you are using, click on either **This computer** or **Another computer**.

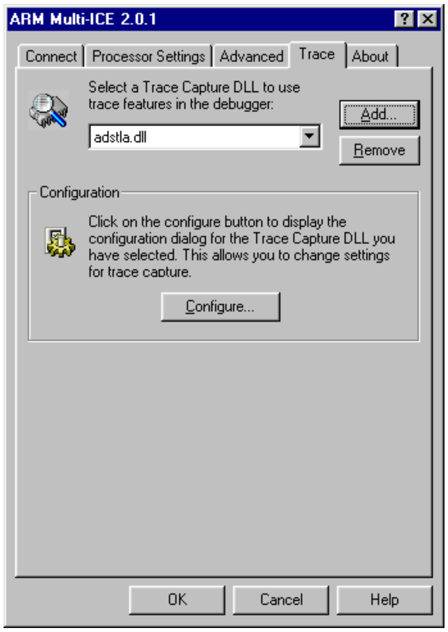
   If you select **Another computer**, in the subsequent dialog box type in the name of the remote workstation, or locate it using the tree control. Click on **OK**.

6. Select the ARM processor that you are tracing from the list shown in **Select a new processor**.

7. Optionally, enter your name in the **Connection name** text field. This shows up in the Multi-ICE server window and can help if you are sharing the server with others.

8. Click on the **Trace** tab to make it visible (Figure B-22 on page B-31).

   The tab contains the controls required to configure the trace control software. The drop-down list labeled **Select a Trace Capture DLL...** contains the names of the currently available trace capture drivers. These drivers read the trace information from the ETM and translate it into the format required by the debugger.



Select a Trace Capture DLL to use trace features in the debugger:

adstla.dll

Configuration

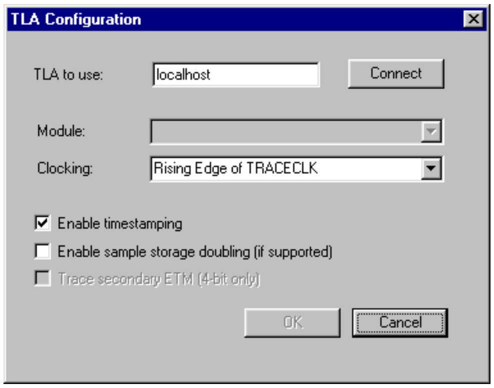
Click on the configure button to display the configuration dialog for the Trace Capture DLL you have selected. This allows you to change settings for trace capture.

Figure B-22 Multi-ICE configuration dialog box showing the Trace tab

9. Select the Dragonfly driver `adstla.dll` from the **Select a Trace Capture DLL...** drop-down list:
   - If `adstla.dll` is present in the drop-down list, select it.
   - If `adstla.dll` is not present in the drop-down list, click **Add...** and select `adstla.dll` from the Dragonfly Software TLA installation directory. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)

   This specifies the driver file that is used to control the trace capture device and to carry out operations such as start and stop tracing, tracing:

10. Click on **Configure...** to display the Dragonfly trace capture configuration dialog box (Figure B-23 on page B-32).


TLA Configuration

TLA to use: localhost

Module:

Clocking: Rising Edge of TRACECLK

☑ Enable timestamping

☐ Enable sample storage doubling (if supported)

☐ Trace secondary ETM (4-bit only)

Figure B-23 Dragonfly TLA Configuration dialog box

11. Configure the Dragonfly Software trace capture agent as described in *Tektronix TLA Logic Analyzer ARM ETM Support Package Instructions* .

12. Click **OK** successively to exit each of the dialog boxes. AXD is now configured for tracing.

AXD restarts with the new configuration when you exit the Target Configuration dialog box.

Note

You must exit AXD or use **File** → **Save Session** to permanently save this configuration.

Appendix C
Using RealMonitor with TDT

This appendix contains the following section:

C.1 Configuring RealMonitor to use TDT

You can use RealMonitor to enable TDT to manipulate target memory without stopping the processor. RealMonitor is a component of ADS, and requires either Multi-ICE Version 2.0 or ARM ADI.

——— **Note** ———

You cannot use RealMonitor with XScale.

———————————

RealMonitor consists of two components:

- the host software, RMHost, that interfaces with Multi-ICE and the host debugger
- the target software, RMTarget, that reads and executes instructions from the host over the DCC.

Because RealMonitor uses the target processor DCC channel, RealMonitor cannot be used with target processors that do not have a DCC channel, or that use the DCC channel for other purposes. You must also integrate the target software component of RealMonitor into your target system before using RMHost.

You can use RealMonitor with any trace capture buffer that supports the use of Multi-ICE as the trace controller.

To configure TDT to use RealMonitor with Multi-ICE:

1. Install and set up the Multi-ICE server.
2. Install and configure RealMonitor.
3. Configure Multi-ICE.
4. Configure TDT using the Multi-ICE Trace configuration tab

To install and set up the Multi-ICE server, you must:

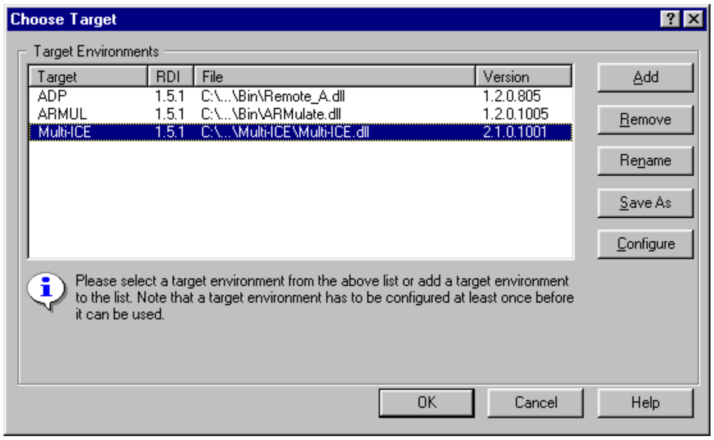
1. Install Multi-ICE Version 2.0 on both the workstation you are debugging on and the workstation that the Multi-ICE interface unit connects to.
2. Configure the Multi-ICE server on the workstation that the Multi-ICE interface unit is connected to.

To Install and configure RealMonitor:

1. Install RealMonitor on the workstation you are debugging on. Refer to the RealMonitor installation instructions for more information.
2. Select **Options** → **Configure Target** in AXD. This displays the target configuration dialog box shown in Figure C-1 on page C-3.

Note

The list of targets and the version numbers shown in Target Environments might vary from those shown in the figure.



Target	RDI	File	Version
ADP	1.5.1	C:\...\Bin\Remote_A.dll	1.2.0.805
ARMUL	1.5.1	C:\...\Bin\ARMulate.dll	1.2.0.1005
Multi-ICE	1.5.1	C:\...\Multi-ICE\Multi-ICE.dll	2.1.0.1001

Please select a target environment from the above list or add a target environment to the list. Note that a target environment has to be configured at least once before it can be used.

Figure C-1 Debugger Configuration dialog box

3. Select RealMonitor:
   - If **RMHost** is present in the Target Environments list, select it.
   - If **RMHost** is not present in the Target Environments list, click **Add** and select `RMHost.dll` from the Multi-ICE install directory. Click **Open** to add it to the Target Environments list. (You might have to use Windows Explorer to ensure that files with the extension `.dll` are not hidden from view.)
4. Click **Configure** to display the RealMonitor Configuration dialog box (Figure C-4 on page C-5).

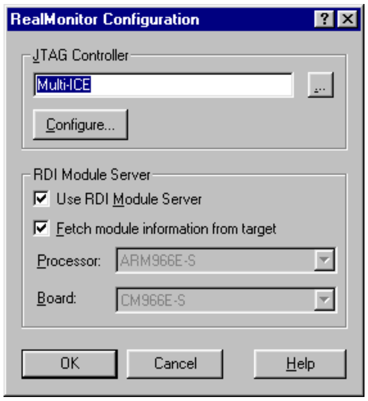


Figure C-2 The RealMonitor Configuration dialog box

5. With Multi-ICE shown in **JTAG Controller**, click **Configure** to configure Multi-ICE. This displays the configuration dialog box shown in Figure C-3.

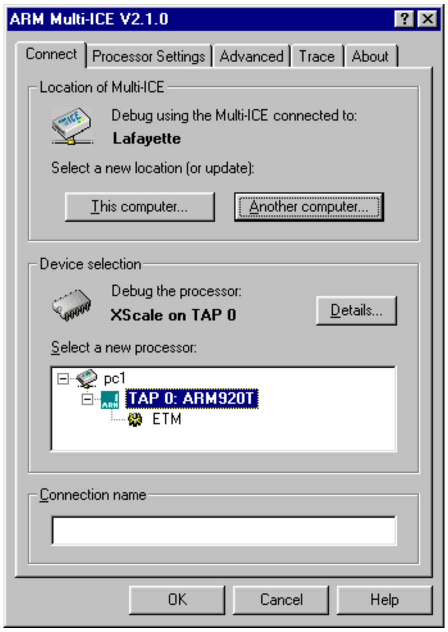
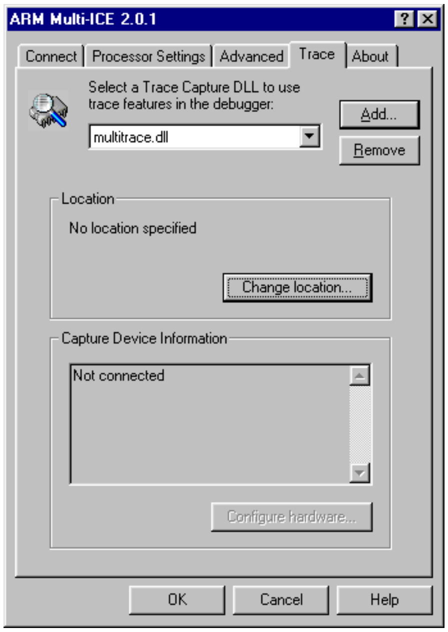


Figure C-3 The Multi-ICE configuration dialog box

6. Configure Multi-ICE as required. This is described in the *Multi-ICE Version 2.0 User Guide*.

7. In the Multi-ICE configuration dialog box, click the **Trace** tab (Figure C-4).



Select a Trace Capture DLL to use trace features in the debugger:

multitrace.dll

Location

No location specified

Capture Device Information

Not connected

Figure C-4 Multi-ICE configuration dialog box showing the Trace tab

8. Configure the trace capture software as described in the appropriate section of Appendix B *Setting up the Trace Software*.

9. Click **OK** in the Multi-ICE configuration dialog box to confirm the Multi-ICE configuration. Configure the rest of RMHost as described in the *ARM RMHost User Guide*.

10. Click **OK** in the RealMonitor configuration dialog box to complete configuration of the target.

11. Click **OK** in the AXD target configuration dialog box to connect to the target system.

C.2 Using RealMonitor with TDT

There are some interactions between TDT and RealMonitor that are described in the following sections:

- *Tracing RMTarget code*
- *Performance and resources* on page C-7.

C.2.1 Tracing RMTarget code

The RMTarget component of RealMonitor runs as an exception handler on the target processor, and the RMTarget initialization code is run as part of application initialization. Unless you take steps to prevent it, TDT traces the RMTarget code together with the application. Because TDT uses RMTarget to read target memory, RMTarget instructions appear in the trace buffer asynchronously to application program events.

You can prevent RMTarget code from entering the trace buffer by:

- using an Address Exclude Range to prevent Enable Tracing from being TRUE during execution of the RMTarget exception handler
- using a Start and Stop Tracing resource pair to prevent Enable Tracing from being TRUE during execution of the RMTarget exception handler.

For more information on setting up these resources, refer to Chapter 5 *Configuring Trace Capture*.

C.2.2 Modified breakpoint behavior

RealMonitor implements breakpoints without stopping the processor. It does this by entering a loop when a breakpoint is detected and only exiting the loop when the user requests that execution restart. Using a purely JTAG run control system, for example Multi-ICE, a breakpoint halts the processor, and at that point TDT stops tracing. Because RMTarget does not stop the processor in the same way, TDT does not stop tracing. You must use the **Stop tracing** menu option to stop tracing, or you must define a trigger event to make the ETM write a block of trace data around the trigger point.

If you set a breakpoint using RealMonitor and then exit the debugger, leaving the target running the application, you must re-apply any breakpoints you require after you reattach the debugger to the target.

If you attach to a running target with RealMonitor and **Start tracing** it, unless the loaded trace capture rule set includes a trigger rule, no trace data is returned to TDT until you:

- stop the target application, using **Stop** on the AXD **Execute** menu, or click the equivalent toolbar button
- stop tracing the target, by selecting **Stop tracing** on the **Trace** menu.

You can add or change a trigger rule without stopping the target application.

C.2.3 Performance and resources

RealMonitor affects target performance in a number of ways:

- it requires a small quantity of instruction and data memory on the target
- it uses some processor cycles to service host debug requests
- with certain trace capture rule sets, it might use either:
  - ETM port bandwidth and trace buffer space to trace its instructions
  - ETM resources to prevent the ETM from tracing its instructions
- it requires a processor that has a DCC, and requires exclusive access to it
- it requires that it is linked into the exception handlers of the target application.

RealMonitor does provide the important ability to read and write target memory without stopping the processor. This ability is essential in some situations and useful in many.

Appendix D
ETM Versions

This appendix describes the relationship between the version number of the ETM implementation and the protocol version implemented by the ETM. It contains the following section:

D.1 ETM versions

The ETM is identified in the following ways:

- the ETM hardware implementation version
- the processor family number, for example `9` for ARM9TDMI family processors
- the protocol version number
- the ETM architecture version.

The hardware implementation is referred to in the following way:

`ETMx Rev y`

where *x* is the processor family number and *y* is the hardware revision number.

The protocol version number identifies the hardware and the software interface that the hardware provides. It is reported by TDT in the View ETM State dialog box. Software can use the protocol version number to determine the facilities that the ETM might implement, and also to determine how to find out which facilities are actually implemented.

Note

It is not sufficient to describe an ETM implementation as an ETM Rev 0. You must specify the processor family number with the ETM implementation, for example ETM7 Rev 0, because the ETMs vary in functionality between processor families, as shown in Table D-1.

Reference to protocol versions is deprecated in ETMs supporting architectures ETMv2 and above. Protocol versions are now referred to by the corresponding architecture as shown in Table D-1.

Table D-1 ETM version number relationships

Implementation	ETM Architecture version	Implementation version	Protocol number (deprecated)
ETM7 Rev 0	ETMv1.1	0	1
ETM7 Rev 1	ETMv1.2	0	2
ETM7 Rev 1a	ETMv1.2	1	4
ETM9 Rev 0	ETMv1.0	0	0
ETM9 Rev0a	ETMv1.1	0	1
ETM9 Rev 1	ETMv1.2	0	2

Table D-1 ETM version number relationships (continued)

Implementation	ETM Architecture version	Implementation version	Protocol number (deprecated)
ETM9 Rev 2	ETMv1.3	0	3
ETM9 Rev 2a	ETMv1.3	1	5
ETM10 Rev 0	ETMv2.0	0	-

The implementation in use can be determined from the combination of the processor family, the ETM architecture, and the ETM implementation version.

Glossary

This glossary contains definitions for most of the terms contained within this User Guide. For terms that are not in this chapter, please refer to the *Glossary* section of the *ARM Debuggers Guide*.

ARM Developer Suite	A suite of applications, together with supporting documentation and examples, that enables you to write and debug applications for the ARM family of RISC processors.
ARM eXtended Debugger	The *ARM eXtended Debugger* (AXD) is the latest debugger software from ARM that enables you to make use of a debug agent in order to examine and control the execution of software running on a debug target. AXD is supplied in both Windows and UNIX versions.
ARM target board	Within this manual only, a circuit board containing an ARM7-family or ARM9-family processor linked to an ETM with the memory and peripherals required to run the application that you are tracing.
ARMulator	ARMulator is an instruction set simulator. It is a collection of modules that simulate the instruction sets and architecture of various ARM processors.
ASIC	*Application-Specific Integrated Circuit*. An integrated circuit designed to perform a particular function by defining the interconnection of a set of circuit-building components. These components are drawn from a library provided by the circuit manufacturer. An ARM ASIC contains the cores that allow you to utilize the Real-Time Trace solution.

CPU	Central Processor Unit.
ETM	*Embedded Trace Macrocell*. A block of logic, embedded in the ASIC, that is connected to the address, data, and status signals of the processor. It broadcasts branch addresses, and data and status information in a compressed protocol through the ASIC trace port. It contains the resources used to trigger and filter the trace output.
GPIO	*General Purpose Input/Output*. Refers to the pins on an ASIC that are used for I/O. Some of these GPIO pins can be multiplexed to extend the trace port width.
Java	A highly portable software language, enabling cross-platform execution. Execution of Java programs is accelerated by the ARM Jazelle architecture extensions.
Jazelle	ARM architecture extensions that accelerate the execution of Java.
Nonstop debugging	Using a debugger to inspect a process without halting it, and without interfering with the operation of the process in other than user requested ways. Although memory can be changed, processor registers cannot be inspected or modified, because they are constantly changing. *See also* Start-stop debugging.
Multi-ICE	Multi-processor EmbeddedICE interface. ARM registered trademark.
Processor Core	The part of a microprocessor that reads instructions from memory and executes them, including the instruction fetch unit, arithmetic and logic unit and the register bank. It excludes optional coprocessors, caches, and the memory management unit.
RDI	*See* Remote Debug Interface
RealTrace	A grouping of hardware and TDT whose collective goal is to enable you to trace instructions and data accesses in real time. The ASIC must contain an ARM CPU macrocell and an ETM. The ASIC must have two ports: • a JTAG port that connects to a JTAG unit receiving commands from AXD • a Trace port that connects to a TPA that stores its information.
Remote Debug Interface	Is an open ARM standard procedural interface between a debugger and the debug agent. The widest possible adoption of this standard is encouraged. RDI gives the debugger a uniform way to communicate with: • a debug agent running on the host (for example, ARMulator) • a debug monitor running on ARM-based hardware accessed through a communication line (for example, Angel) • a debug agent controlling an ARM processor through hardware debug support (for example, Multi-ICE).

Resource	Hardware on the ETM that implements continuous rule expression evaluation that in turn enables or disables actions such as Enable Tracing.
Start-stop debugging	Using a debugger to debug a process by halting the process, examining or changing the state of memory or registers and then restarting the process. Typified by single stepping. *See also* Nonstop debugging.
TAP	*See* Test Access Port.
TAP Controller	The unit which controls access to the TAP port on a device. *See also* TAP, IEEE1149.1.
TDT	*Trace Debug Tools*. A software product add-on to AXD that extends the debugging capability with the addition of real-time program and data tracing. TDT provides a user-friendly interface from which you can obtain an historical, non-intrusive trace of instruction flow and data accesses to help you identify a bug more efficiently than by using traditional debugging methods.
Test Access Port	A set of circuits on a device which allow access to some or all of that device for test purposes. The circuit functionality is defined in IEEE1149.1.
TIM	*Target Interface Module*. A printed circuit board supplied by Agilent that customizes the emulation probe or module, enabling it to work with an ARM processor.
TPA	*Trace Port Analyzer*. An external device that stores the information from the trace port. This information is compressed so that the analyzer does not need to capture data at the same bandwidth as that of an analyzer monitoring the core buses directly.
Trace capture hardware	A device that is connected to the trace port on an ETM and converts the trace port signals into a form that the host computer and debugger can access. For example, it may convert it into packets on an ethernet network.
XScale	The Intel® XScale™ Microarchitecture is an implementation of the ARM v5TE architecture designed by Intel Corporation. For details of Intel XScale Microarchitecture, visit http://www.intel.com.

Index

The items in this index are listed in alphabetical order, with symbols appearing at the end. The references given are to page numbers.