Tarmac Trace for Fast Models

Version 10.0

User Guide

Tarmac Trace for Fast Models

User Guide

Release Information

Document History

Issue	Date	Confidentiality	Change
A	31 May 2014	Non-Confidential	New document for Fast Models v9.0, from DUI0532G for v8.3.
B	30 November 2014	Non-Confidential	Update for v9.1.
C	28 February 2015	Non-Confidential	Update for v9.2.
D	31 May 2015	Non-Confidential	Update for v9.3.
E	31 August 2015	Non-Confidential	Update for v9.4.
F	30 November 2015	Non-Confidential	Update for v9.5.
G	29 February 2016	Non-Confidential	Update for v9.6.
H	31 May 2016	Non-Confidential	Update for v10.0.

Contents

Tarmac Trace for Fast Models User Guide

Preface

This preface introduces the *Tarmac Trace for Fast Models User Guide*.

It contains the following:

- *About this book* on page 6.

About this book

This manual describes the use of the Fast Models Tarmac Trace plug-in from ARM, and the format of the trace files it generates.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction
This chapter introduces the document.

Chapter 2 Tarmac Trace Plug-in
This chapter describes how to set up the environment to use the Tarmac Trace plug-in, and how to start a simulation. It also describes the parameters that control the type of events to trace.

Chapter 3 Tarmac Trace File Format
This chapter describes the Tarmac Trace for Fast Models file format.

Appendix A Architecture Message Plug-in
This appendix describes the Architecture Message plug-in.

Glossary

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

See the *ARM Glossary* for more information.

Typographic conventions

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

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

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

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

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

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

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

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

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

Feedback

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:
- The product name.
- The product revision or version.
- An explanation with as much information as you can provide. Include symptoms and diagnostic procedures if appropriate.

Feedback on content

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

- The title *Tarmac Trace for Fast Models User Guide*.
- The number ARM DUI0845H.
- If applicable, the page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Note

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

Other information

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

Chapter 1
Introduction

This chapter introduces the document.

It contains the following sections:

1.1 About Tarmac Trace

Tarmac is a textual trace output.

Fast Models supports the generation of traces that consistently track the execution and related activities in the model, particularly those that affect the state of the modeled IP. Generated virtual platforms provide trace support by using plug-ins in the form of DLLs and shared objects on Windows and Linux, respectively. Using the plug-in, trace information is written to a file in textual form in the format described in this document.

ARM provides a plug-in to produce a textual trace output (Tarmac). You can use other plug-ins, using the *Model Trace Interface* (MTI), instead, or at the same time. You can connect various plug-ins to this interface in the form of a shared object loaded at simulation start-up.

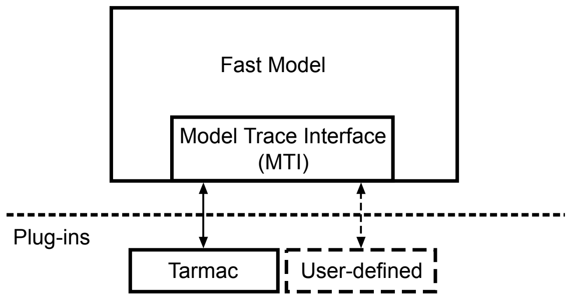


Figure 1-1 Interaction between MTI and plug-ins

This document describes:
- How to enable and disable Tarmac Trace.
- How to control Tarmac Trace.
- File formats and how to analyze the output.

Related references

Chapter 3 Tarmac Trace File Format on page 3-15.

Chapter 2
Tarmac Trace Plug-in

This chapter describes how to set up the environment to use the Tarmac Trace plug-in, and how to start a simulation. It also describes the parameters that control the type of events to trace.

It contains the following sections:

2.1 Getting started

This section provides information on specifying the location of the trace plug-ins.

This section contains the following subsections:

- *2.1.1 Pointing to the position of the Tarmac Trace plug-in* on page 2-11.

2.1.1 Pointing to the position of the Tarmac Trace plug-in

To provide the plug-in to the simulation, use a tool-specific method if possible.

When launching a model with an application that understands *Model Trace Interface* (MTI), use the tool-specific method of providing the plug-in to the simulation. Examples of such applications are Model Debugger, Model Shell, and SystemC (with *Multiple Instantiation* (MI) and command line parsed).

When specifying the plug-in to the launching tool is not possible, specify the trace plug-in by setting the environment variable `FM_TRACE_PLUGINS`.

This variable must point to the full path of the Tarmac Trace plug-in. On Linux, for `sh` users this path might be, for example:

```
export FM_TRACE_PLUGINS /home/<user>/<installation_path>/plugins/<platform>/
TarmacTrace.so
```

On Windows, the full path might resemble this example: `C:\Program Files (x86)\ARM \FastModelsPortfolio\plugins\Win64_VC20`*`XX`*`\Release`.

To use multiple plug-ins at the same time, separate them by ';'. You can also load the same plug-in multiple times. You can give a name for the plug-in instance by prefixing `instancename=` to the plug-in path or paths.

2.2 Starting the simulation

Tarmac Trace tracks a simulation with or without a debugger.

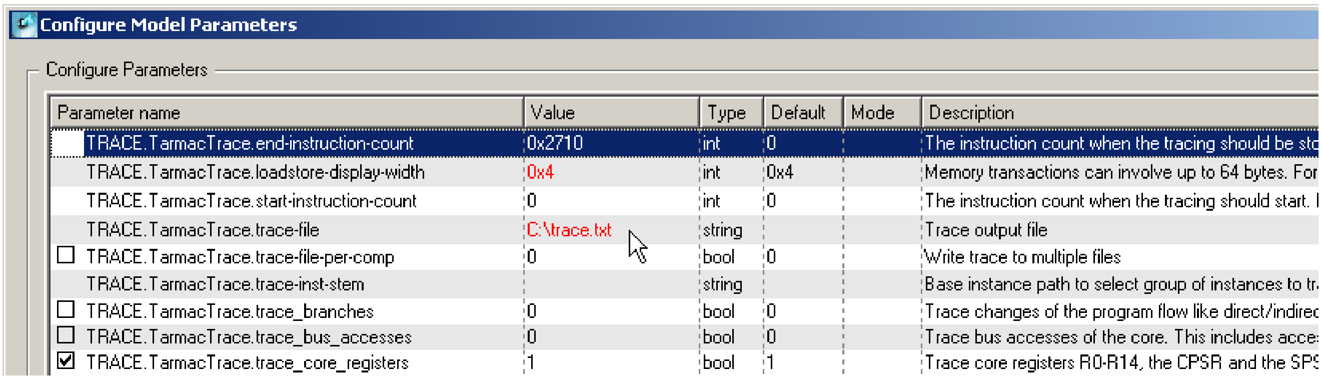
This section contains the following subsections:
- *2.2.1 Running the simulation with Model Debugger* on page 2-12.
- *2.2.2 Running the simulation without a debugger* on page 2-12.

2.2.1 Running the simulation with Model Debugger

How to run a simulation in Model Debugger.

Procedure

1. Specify the Tarmac Trace plug-in to load on simulation.
2. In Model Debugger, select **File** > **Load Model...**.
3. Set the file path to the simulation library.
4. Set the model parameters in the Configure Model Parameters dialog box.



Configure Model Parameters

Configure Parameters

Parameter name	Value	Type	Default	Mode	Description
TRACE.TarmacTrace.end-instruction-count	0x2710	int	0		The instruction count when the tracing should be sto
TRACE.TarmacTrace.loadstore-display-width	0x4	int	0x4		Memory transactions can involve up to 64 bytes. For
TRACE.TarmacTrace.start-instruction-count	0	int	0		The instruction count when the tracing should start. I
TRACE.TarmacTrace.trace-file	C:\trace.txt	string			Trace output file
☐ TRACE.TarmacTrace.trace-file-per-comp	0	bool	0		Write trace to multiple files
TRACE.TarmacTrace.trace-inst-stem		string			Base instance path to select group of instances to tr
☐ TRACE.TarmacTrace.trace_branches	0	bool	0		Trace changes of the program flow like direct/indirec
☐ TRACE.TarmacTrace.trace_bus_accesses	0	bool	0		Trace bus accesses of the core. This includes acce
☑ TRACE.TarmacTrace.trace_core_registers	1	bool	1		Trace core registers R0-R14, the CPSR and the SPS

Figure 2-1 Setting model parameters

   You might set, for example, the trace end count value and the name and location of the trace output file.

5. Load the application file.

Related tasks

2.1.1 Pointing to the position of the Tarmac Trace plug-in on page 2-11.

Related information

Model Debugger for Fast Models User Guide.

2.2.2 Running the simulation without a debugger

Start the simulation library using Model Shell. This tool permits the running of simulation targets like the ARM® Cortex®-A15 VE platform.

The corresponding executable `model_shell` is located in the `bin` directory of the installation. It provides several options to set parameters and load application files. The most convenient way to set parameters is to use a configuration file.

To generate this configuration file, start `model_shell` with the `--list-params` option and the simulation library. For Linux this is:

```
model_shell --list-params <path_to_simulation_library> > params.config
```

The configuration file for the parameters can have any arbitrary name and can be edited using a normal text editor to set the parameter values.

For Linux and Windows, the simulation library might be started using the parameter configuration file with the following command:

```
model_shell <path_to_simulation_library> -f params.config –a
<application_file.axf>
```

The `--help` option lists all available options for `model_shell`.

——— **Note** ———

Use the `-C`, `--parameter PARNAME=VALUE` option to set individual parameters on the `model_shell` command line. This permits priority over parameters specified in a parameter file.

———————————

2.3 Tarmac Trace parameters

Configure the Tarmac Trace plug-in with these parameters.

The plug-in prefixes parameters with the path `TRACE.instance-name`, where instance-name is `TarmacTrace`, which is overridable.

Table 2-1 Parameter descriptions

Parameter name	Type	Default	Description
`end-instruction-count`	`int`	`0x0`	Set the instruction count where tracing ends. `0x0` sets to trace until the end of the simulation.
`loadstore-display-width`	`int`	`0x4`	Memory transactions can in the case of LDM/STM involve up to 64 bytes. For easier readability you can break these up into multiple memory access records with a smaller size of bytes. `0` sets not to break up any transaction. `4` sets to break up transactions into words.
`quantum-size`	`int`	`0x100`	Set the default quantum size for tracing. The component `CORE_INFO.QUANTUM_SIZE` trace source field overrides this.
`trace_branches`	`bool`	`false`	Trace all nonsequential changes of the program flow. The information traced is sufficient to completely reconstruct program flow, and the tracing is fairly efficient.
`trace_bus_accesses`	`bool`	`false`	Trace all bus accesses. This forces all direct memory accesses to turn into full transactions, which slows down the simulation.
`trace_core_registers`	`bool`	`true`	If `true`, trace core registers (R0-R14, CPSR and SPSR). This produces a lot of data and can slow down simulation.
`trace_cp15`	`bool`	`true`	Determines whether to trace writes to CP15 registers.
`trace_events`	`bool`	`true`	Determines whether to trace exceptions and mode changes (for processors implementing modes).
`trace-file`	`string`	`""`	Name of the trace output file. If empty, the trace output goes to `stdout`. If `STDERR` the trace output goes to `stderr`.
`trace-file-per-comp`	`bool`	`false`	Create a separate trace file for each component traced, adding the component name to the trace file name. At present the only components that support trace are processors, so this option is only relevant when there are multiple processors.
`trace-inst-stem`	`string`	`""`	If set to a component path, trace only a subtree of components. In the simplest case of setting the component path of a single processor, then trace only this processor.
`start-instruction-count`	`int`	`0x0`	Set the instruction count where tracing starts. `0x0` sets to start from the beginning.
`trace_instructions`	`bool`	`true`	Determines whether to trace instructions.
`trace_loads_stores`	`bool`	`true`	Determines whether to trace load/stores. This is cheaper performance-wise than bus tracing.
`trace_vfp`	`bool`	`true`	Determines whether to trace VFP and NEON™ registers (including FPSCR and FPEXC).

Related tasks

2.2 Starting the simulation on page 2-12.

Chapter 3
Tarmac Trace File Format

This chapter describes the Tarmac Trace for Fast Models file format.

It contains the following sections:

3.1 Instruction trace

If enabled, this trace source generates one record for every instruction started.

The records (lines) of the instruction trace have this command syntax:

```
<time> <scale> <cpu> [IT|IS] (<inst_id>) <addr> <opcode> [A|T|X] <mode>_<security> :
<disasm>
```

`<time>`
: Timestamp (decimal value).

`<scale>`
: Unit for `<time>`. `clk` indicates the timestamp is not related to real time, but an increasing count.

`<cpu>`
: Processor that gave the instruction.

`[IT|IS]`

- `IT`
  Instruction passed the condition code (taken).
- `IS`
  Instruction failed the condition code (skipped).

`<inst_id>`
: Tick count of this processor. This is equivalent to the number of instructions executed, except for certain instructions like `WFI`/`WFE` (decimal value).

`<addr>`
: Fetch source address for this instruction, in hexadecimal format (virtual address).

`<opcode>`
: 16-bit/32-bit hexadecimal opcode of the instruction.

`[A|T|X]`
: Instruction set:

- `A`
  A32.
- `T`
  T32.
- `X`
  T32EE.

`<mode>`
: Processor execution mode (`svc`, `irq`, `fiq`, `usr`, `mon`, `sys`, `abt`, `und`).

`<security>`
: Processor security state (`s` or `ns`).

`<disasm>`
: Disassembly of the instruction.

3.2 Program flow trace

If enabled, every executed branch instruction triggers this trace source. This is a more efficient way to reconstruct the program flow than by tracing every instruction.

Command syntax:

```
<time> <scale> [FD|FI|FR] (<inst_id>) <addr> <targ_addr> [A|T|X]
```

`<time>`
: Timestamp (decimal value).

`<scale>`
: Unit for `<time>`. This gives consistency with device-specific Tarmac Trace formats.

`[FD|FI|FR]`
: Program flow change by:

  `FD`
  : A direct branch.

  `FI`
  : An indirect branch.

  `FR`
  : A return from exception.

`<inst_id>`
: Tick count of this processor. This is equivalent to the number of instructions executed, except for certain instructions like `WFI`/`WFE` (decimal value).

`<addr>`
: Fetch source address for this instruction, in hexadecimal format (virtual address).

`<targ_addr>`
: Address (virtual) at which the execution continues.

`[A|T|X]`
: Instruction set after the branch:

  `A`
  : A32.

  `T`
  : T32.

  `X`
  : T32EE.

3.3 Register trace

If enabled, this source traces all writes to the processor registers.

This trace source includes writes to core registers R0 to R14, CPSR and SPSR, VFP registers such as S0 to S31, D0 to D31, FPSCR, FPEXC, and writes to CP14 and CP15 registers. Banked registers are traced separately using the mode as a suffix to the register name, for example r13 (current register R13) and r13_mon (banked register R13).

Command syntax:

```
<time> <scale> R <register> <value>
```

`<time>`
: Timestamp (decimal value).

`<scale>`
: Unit for `<time>`. This gives consistency with device-specific Tarmac Trace formats.

`<register>`
: Register name in lowercase letters. Banked core registers can have a mode appended with a single underscore. Banked CP14/CP15 registers have `_s` or `_ns` appended to indicate access of either the secure or non-secure banked register.

`<value>`
: Hexadecimal value written to the register (64 bits maximum).

3.4 Event trace

If enabled, this source traces exceptions and interrupts occurring.

Command syntax:

```
<time> <scale> E <value> <number> <desc>
```

`<time>`
: Timestamp (decimal value).

`<scale>`
: Unit for `<time>`. This gives consistency with device-specific Tarmac Trace formats.

`<value>`
: Hexadecimal representation of a value associated with the event.

`<number>`
: Event number.

`<desc>`
: Event name.

Table 3-1 Supported values for value, number and desc

Number	Event description	Value
00000001	CoreEvent_Reset	-
00000002	CoreEvent_UndefinedInstr	-
00000003	CoreEvent_SWI	SWI number
00000004	CoreEvent_PrefetchAbort	-
00000005	CoreEvent_DataAbort	-
00000007	CoreEvent_IRQ	-
00000008	CoreEvent_FIQ	-
0000000E	CoreEvent_ImpDataAbort	-
00000019	CoreEvent_ModeChange	New mode

3.5 Processor memory access trace

If enabled, this source traces processor data accesses.

Command syntax:

`<time> <scale> M<rw><sz><attrib> <addr> <data>`

`<time>`
: Timestamp (decimal value).

`<scale>`
: Unit for `<time>`. This gives consistency with device-specific Tarmac Trace formats.

`<rw>`

- `R`
  Read access.
- `W`
  Write access.

`<sz>`
: Size of the data transfer in bytes (1, 2, 4, 8).

`<attrib>`
: Optional access attribute:

- `X`
  Exclusive access.
- `T`
  Translated (unprivileged) access.
- `L`
  Locked access (SWP, SWPB instructions).

`<addr>`
: Virtual address used to access memory in hexadecimal format.

`<data>`
: Hexadecimal value of data transferred. The data padding is according to the size of the transfer.

3.6 Memory bus trace

If enabled, this source traces transactions initiated through the memory bus master port of the processor. These accesses use physical addresses.

Command syntax:

```
<time> <scale> B<rw><sz><fd><lk><p><s> l<wrcbs> O<wrcbs> <master_id> <addr> <data>
```

`<time>`
: Timestamp (decimal value).

`<scale>`
: Unit for `<time>`. This gives consistency with device-specific Tarmac Trace formats.

`<rw>`

- `R`
  Read access.
- `W`
  Write access.

`<sz>`
: Size of the data transfer in bytes.

`<fd>`

- `I`
  Opcode fetch.
- `D`
  Data load/store or an MMU access.

`<lk>`

- `L`
  Locked access.
- `X`
  Exclusive access.
- `_`, underscore
  Normal access.

`<p>`

- `P`
  Privileged access.
- `_`, underscore
  Normal access.

`<s>`

- `S`
  Secure access.
- `N`
  Non-secure access.

`I<wrcbs>`
: Inner cache attributes. See `O<wrcbs>`.

`O<wrcbs>`
: Outer cache attributes:

  <w>

  - `W`
    Allocate on write.
  - `_`, underscore
    No allocate on write.

<r>

`R`
Allocate on read.

`_`, underscore
No allocate on read.

<c>

`C`
Cacheable access.

`_`, underscore
Non-cacheable access.

<b>

`B`
Bufferable access.

`_`, underscore
Non-bufferable access.

<s>

`S`
Shareability access.

`_`, underscore
Non-shareability access.

`<master_id>`
Master ID of the transaction.

`<addr>`
Physical address used to access memory in hexadecimal format.

`<data>`
Hexadecimal value of data transferred. The data padding is according to the size of the transfer. Byte ordering is from lowest to highest byte. This means that for accesses in little endian mode, the data occurs mirrored compared to the register/memory access records.

3.7 Example of the Fast Models Tarmac Trace file format

The Tarmac Trace plug-in produced this trace file, which shows instruction, register, event, and processor memory access traces.

```
10 clk IT (10) 00001088 e89d00ff A mon_ns : LDMIA sp,{r0-r7}
10 clk MR8 00103fbc 0000000000000060
10 clk MR8 00103fc4 0010400000000000
10 clk MR8 00103fcc 0000000000004000
10 clk MR8 00103fd4 0000000000000000
10 clk R r0 00000060
10 clk R r1 00000000
10 clk R r2 00000000
10 clk R r3 00104000
10 clk R r4 00004000
10 clk R r5 00000000
10 clk R r6 00000000
10 clk R r7 00000000
11 clk IT (11) 0000108c e28dd03c A mon_ns : ADD sp,sp,#0x3c
11 clk R r13_mon 00103ff8
12 clk IT (12) 00001090 f8bd0a00 A mon_ns : RFEIA sp!
12 clk MR8 00103ff8 0000001300000000
12 clk R r13_mon 00104000
12 clk R cpsr 00000013
12 clk E 00001090 00000019 CoreEvent_ModeChange
25 clk IS (25) 000010c0 13a00000 A svc_ns : MOVNE  r0,#0
26 clk IT (26) 000010c4 eee80a10 A svc_ns : FMXR FPEXC,r0
26 clk R fpexc 01c00000
27 clk IT (27) 000010c8 ed236a06 A svc_ns : FSTMDBS r3!,{s12-s17}
27 clk MW8 00104000 4455667700112233
27 clk MW8 00104008 ccddeeff8899aabb
27 clk MW8 00104010 89abcdef01234567
27 clk R r3 00104000
33 clk IT (33) 00001200 ed334b08 A abt_s : FLDMDBD  r3!,{d4-d7}
33 clk MR8 00105000 2222333300001111
33 clk MR8 00105008 6666777744445555
33 clk MR8 00105010 aaaabbbb88889999
33 clk MR8 00105018 eeeeffffccccdddd
33 clk R d4 2222333300001111
33 clk R d5 6666777744445555
33 clk R d6 aaaabbbb88889999
33 clk R d7 eeeeffffccccdddd
34 clk IT (34) 00001204 f3ba01c2 A abt_s : VZIP.32 q0,q1
34 clk R d0 487201bf46b94bfb
34 clk R d1 37cf1ce11c667e81
34 clk R d2 37200f47ff6abddf
34 clk R d3 2313de569e2cfb54
47 clk IT (47) 00001240 5a0a T abt_s : LDRH r2, [r0,r1]
47 clk MR2 00105000 1111
47 clk R r2 00001111
```

Note

This file does not show program flow or memory bus traces.

Appendix A
Architecture Message Plug-in

This appendix describes the Architecture Message plug-in.

It contains the following sections:

A.1 VE - Architecture Message plug-in - parameters

This section describes the parameters.

Note

To use these parameters, load the ArchMsgTrace plug-in into a model.

```
TRACE.ArchMsg.parameter=value
```

Table A-1 Architecture Message plug-in error and warning message parameters

Parameter	Type	Allowed values	Default value	Description
`suppress_repeated`	`bool`	`true`, `false`	`true`	Suppress repeated messages from similar call sites.
`suppress_sources`	`string`	-	-	Space-separated list of components or events to not print.
`trace-file`	`string`	-	-	ArchMsg output file.