Model Shell for Fast Models
Reference Manual

Copyright © 2008-2009 ARM Limited. All rights reserved.

Release Information

<table>
<thead>
<tr>
<th>Description</th>
<th>Issue</th>
<th>Confidentiality</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2008</td>
<td>A</td>
<td>Non-Confidential</td>
<td>New document.</td>
</tr>
<tr>
<td>March 2009</td>
<td>B</td>
<td>Non-Confidential</td>
<td>Update for Fast Models 4.2.</td>
</tr>
<tr>
<td>April 2009</td>
<td>C</td>
<td>Non-Confidential</td>
<td>Update for Fast Models 5.0.</td>
</tr>
</tbody>
</table>

Proprietary Notice

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

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

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

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

Where the term ARM is used it means “ARM or any of its subsidiaries as appropriate”.

Confidentiality Status

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

Unrestricted Access is an ARM internal classification.

Product Status

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

Web Address

http://www.arm.com
## Contents

**Model Shell for Fast Models Reference Manual**

### Preface
- About this book ........................................................................................................ vi
- Feedback ................................................................................................................ viii

### Chapter 1  Introduction
1.1 Overview ........................................................................................................ 1-2
1.2 Integrated Simulator targets ........................................................................ 1-3

### Chapter 2  Model Shell Commands
2.1 Model Shell options .................................................................................. 2-2
2.2 String syntax ............................................................................................... 2-6
2.3 Defining model parameters .......................................................................... 2-7
2.4 SMP support ............................................................................................... 2-10
2.5 Stopping Model Shell .................................................................................. 2-11
Preface

This preface introduces the *Model Shell for Fast Models Reference Manual*. It contains the following sections:

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

This book describes the operation of Model Shell, a command line tool for configuring and running CADI-compliant models.

Intended audience

This book has been written for experienced hardware and software developers to aid the development of processor architectures using the Cycle Accurate Debug Interface (CADI) models and tools.

Organization

This book is organized into the following chapters:

Chapter 1 Introduction
Read this chapter for an introduction to Model Shell.

Chapter 2 Model Shell Commands
Read this chapter for details of the command-line options for Model Shell.

Typographical conventions

The typographical conventions are:

*italic* Highlights important notes, introduces special terminology, denotes internal 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 underline* 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>

Enclose replaceable terms for assembler syntax where they appear in code or code fragments. For example:

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

Further reading

This section lists related publications by ARM® and third parties.

See http://infocenter.arm.com for access to ARM documentation.

ARM publications

This book contains information that is specific to this product. See the following documents for other relevant information:

• ARM Cycle Accurate Debug Interface Developer’s Guide (ARM DUI 0444)
• ARM Architecture Reference Manual (ARM DDI 0100)
• ARM FLEXnet License Management Guide (ARM DUI 0209)
• Fast Model Tools User Guide (ARM DUI 0370).
Feedback

ARM welcomes feedback on this product and its documentation.

Feedback on this product

If you have any comments or suggestions about this product, contact your supplier and give:
• the product name
• a concise explanation.

Feedback on this book

If you have any comments on this book, send an e-mail to errata@arm.com. Give:
• the title
• the number
• the relevant page number(s) to which your comments apply
• a concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.
Chapter 1

Introduction

This chapter describes the main features of Model Shell, a command line tool for configuring and running your CADI-compliant model. It contains the following sections:

- *Overview* on page 1-2
- *Integrated Simulator targets* on page 1-3.
1.1 Overview

Model Shell is a command line tool for configuring and running CADI-compliant models.

Model Shell connects to any CADI compliant model and provides:

• semihosting stdio
• CADI logging
• a launch platform for debuggers, profilers, and operating environments.

Model Shell can start CADIServer to enable other debuggers to connect to the model in the following ways:

• The simulation is initialized, but not run. An external debugger must control the simulation (default).

• The simulation is initialized and run immediately. An external debugger can connect to simulation after it starts.

Model Shell provides processes semihosting input and output only for standard streams:

• When CADIServer is started, semihosting output goes to the Model Shell console and to all debuggers.

• If a debugger is attached, it performs semihosting input. If not, Model Shell provides the input.
1.2 Integrated Simulator targets

Fast Models can create *Integrated Simulator* (isim) targets by statically linking Model Shell with a CADI library of a model. See the *Fast Model Tools User Guide* for more details.

All Model Shell command line options, except `--model`, can also be used with an isim target. Because the model is integrated into the target, there is no requirement to specify the model on the command line.
Chapter 2
Model Shell Commands

This chapter describes how to use Model Shell. It contains the following sections:

• Model Shell options on page 2-2
• String syntax on page 2-6
• Defining model parameters on page 2-7
• SMP support on page 2-10
• Stopping Model Shell on page 2-11.


## 2.1 Model Shell options

You can use the Model Shell options to tailor the behavior of Model Shell for different use cases. These options are listed in Table 2-1 on page 2-3.

To start Model Shell from the command line, type `model_shell` followed by any required options.

All of the following lines are acceptable syntax for starting Model Shell:

- `model_shell [options] model [application_list]`
- `model_shell [options] -m model [application_list]`
- `model_shell -m model [options] [application_list]`
- `model_shell -m model -a app1.axf [options]`

where:
- `options` is a list of command line options from Table 2-1 on page 2-3
- `model` is the file name, including extension, for the model.
  The `-m` option is not required, if the model name is:
  - for a system with one core, that is, a single core or an SMP core
  - at the end of the line
  - the next to last item on the line and only followed by the names of the application files.
  Model files have an extension of `.so` or `.dll`.
- `application_list` is the application, or list of applications, to load for the model.
  The `-a` option is not required if the application list is at the end of the line.
  Use the `-a instance=filename` option described in Table 2-1 on page 2-3 to load an application to a specific system instance.
  Application files have an `.axf` extension.

### Note

On Windows, it might be necessary to add the directory that contains the Model Shell executable is found to your `PATH`. This location is typically:

```
C:\Program Files\ARM\FastModelTools\version\bin
```

where `version` is the Fast Models release version such as, for example, 5.0.
Table 2-1 Model Shell command line options

<table>
<thead>
<tr>
<th>Short</th>
<th>Long option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>--cpulimit ( n )</td>
<td>Specify the maximum number of CPU seconds to run as ( n ). Fractions of a second can be specified, but the remaining time is only tested to a resolution of 100ms. If ( n ) is omitted, the default is unlimited.</td>
</tr>
<tr>
<td>-</td>
<td>--cyclelimit ( n )</td>
<td>Specify the maximum number of cycles to run as ( n ). If ( n ) is omitted, the default is unlimited.</td>
</tr>
<tr>
<td>-</td>
<td>--data file@address</td>
<td>Specify raw data to load at specified address. The full format is: --d ([\text{INST}]=)file@[\text{memspace}]:\text{address}</td>
</tr>
<tr>
<td>-</td>
<td>--dump file@address, size</td>
<td>Dump a section of memory into file. The full format is: --u ([\text{INST}]=)file@[\text{memspace}]:\text{address},size</td>
</tr>
<tr>
<td>-</td>
<td>--list-instances</td>
<td>List target instances.</td>
</tr>
<tr>
<td>-</td>
<td>--list-params</td>
<td>List target instances and their parameters. Use this to help identify the correct syntax for configuration files, as well as to find out what the target supplies.</td>
</tr>
<tr>
<td>-</td>
<td>--list-memory</td>
<td>Print memory information for the model to standard output.</td>
</tr>
<tr>
<td>-</td>
<td>--start address</td>
<td>Initialize the PC to the specified application start address. The full format is: --start ([\text{INST}]=)address</td>
</tr>
<tr>
<td>-</td>
<td>--stat</td>
<td>Print statistics at end of simulation.</td>
</tr>
<tr>
<td>-</td>
<td>--timelimit ( n )</td>
<td>Specify the maximum number of seconds to run as ( n ). If ( n ) is omitted, the default is unlimited.</td>
</tr>
</tbody>
</table>

**Note**

If \( n \) is specified as 0, Model Shell:
- initializes the system,
- loads all applications and data
- sets breakpoints and PC
- exits immediately without running the model.

Use this option to convert applications to raw binary. For example:
```
model_shell --timelimit 0 -m mymodel.dll -a app.axf
-u app.raw@0x8000,0x10000
```
<table>
<thead>
<tr>
<th>Short</th>
<th>Long option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>--application filename</td>
<td>Load application filename. Use <code>-a instance=filename</code> to load a specific system instance. See SMP support on page 2-10 and String syntax on page 2-6. The -a option is not required if the application filename is at the end of the command line.</td>
</tr>
<tr>
<td>-b</td>
<td>--break address</td>
<td>Set program breakpoint at given address. Use <code>-b instance=address</code> to set a breakpoint for a specific system instance.</td>
</tr>
<tr>
<td>-C</td>
<td>--parameter parameter=value</td>
<td>Set a parameter to the specified value. See Defining model parameters on page 2-7. Use <code>-C instance.parameter=value</code> to set a parameter for a specific system instance.</td>
</tr>
<tr>
<td>-d</td>
<td>--disassemble</td>
<td>For Core Generator models only, disassemble program on stdout and exit.</td>
</tr>
<tr>
<td>-f</td>
<td>--config-file filename</td>
<td>Use model parameters from configuration file filename. See Configuration file on page 2-8. See also String syntax on page 2-6.</td>
</tr>
<tr>
<td>-h</td>
<td>--help</td>
<td>Print out the list of Model Shell command line options and exits.</td>
</tr>
<tr>
<td>-K</td>
<td>--keep-console</td>
<td>Keep console window open after completion.</td>
</tr>
<tr>
<td>-L</td>
<td>--cadi-log</td>
<td>Logs all CADI calls to an XML log file.</td>
</tr>
<tr>
<td>-m</td>
<td>--model filename</td>
<td>Load model filename. See String syntax on page 2-6. The -m option is not required if the model name is at the end of the command line and not followed by any text except for the optional application file name.</td>
</tr>
<tr>
<td>-o</td>
<td>--output filename</td>
<td>Redirect output from the --list-instances and --list_params commands to file. See String syntax on page 2-6. The contents of the file are formatted correctly for use as input by the --config-file option.</td>
</tr>
<tr>
<td>-P</td>
<td>--prefix</td>
<td>Prefix semihosting output with the name of the target instance.</td>
</tr>
</tbody>
</table>
### Table 2-1 Model Shell command line options (continued)

<table>
<thead>
<tr>
<th>Short</th>
<th>Long option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-R</td>
<td>--run</td>
<td>Run simulation immediately after loading. The simulation is run immediately even if CADI server has been started.</td>
</tr>
<tr>
<td>-q</td>
<td>--quiet</td>
<td>Suppress Model Shell output.</td>
</tr>
<tr>
<td>-S</td>
<td>--cadi-server</td>
<td>Start a CADI server. This enables attaching a debugger to debug targets in the simulation. To shut down the server, return to the command window that you used to start the model and press Ctrl + C to stop the CADI server. The Model Shell process must be in the foreground before you can shut it down.</td>
</tr>
<tr>
<td>-t</td>
<td>--cadi-trace</td>
<td>Enable diagnostic output from CADI calls and callbacks.</td>
</tr>
<tr>
<td>-V</td>
<td>--verbose</td>
<td>Enable verbose messages of message class ModelShell. This command has the same effect as: model_shell --enable-verbose ModelShell.</td>
</tr>
<tr>
<td>-v</td>
<td>--version</td>
<td>Print the Model Shell version number and exit.</td>
</tr>
</tbody>
</table>
2.2 String syntax

Filenames and similar strings that are entered on the command line must be within double quotes if there is white space in the string. For example:

model_shell -a "my application file.axf" ...

There is, however, no requirement to use quotes if the parameter is a single word with no spaces. Both the following forms are valid:

model_shell --list-instances output.txt ...

model_shell --list-instances "output.txt" ...
2.3 Defining model parameters

You can define model parameters on the command line that you use to start Model Shell.

The definition must be in the following format:

```
--parameter [instance.]parameter_name=value
```

where:

- `instance` is the name of the component instance. If there is only one instance, the instance specifier can be omitted.
  
  `instance` can be a hierarchical path with each level separated by a period (`.`) character.
  
- `parameter_name` is the parameter to be modified.
  
  For hierarchical systems, you must specify the parameter using its complete name, even if there is only a single instance of it.

- `value` is the value to assign to the parameter.
  
  If `value` is a string, additional formatting rules might apply. See String syntax on page 2-6.
  
  Boolean values can be set using either `true/false` or `1/0`.

The `--parameter` option can be replaced by the short form `-C`. 
2.3.1 Configuration file

You can configure a model that you start from the command line with Model Shell by including a reference to an optional plain text configuration file by using:

```
model_shell --config-file my_configuration_file.txt ...
```

Each line of the configuration file must have the same `instance.parameter=value` syntax as used for command-line assignments.

You can also include comment lines and blank lines in your configuration file by using the `#` character before the comment or blank text.

To generate a configuration file, use the `--list-instances` and `--list-params` options on the command line. The command line can also include parameter assignments. For example, the command line:

```
model_shell --list-params --list-instances -C top-mm=0x3 -C top-mm=0x3 -o file.config -m model.so
```

might generate the `file.config` configuration file listed in Example 2-1:

```
Example 2-1 Generated configuration file

# Instances:
# Instance id: instance name (SW: y/n, component, type, version) : description
# instance.parameter=value       #(type, mode) default = 'def value' : description : [min..max]
#-------------------------------------------------------------------------------------
# Instance  0:       (SW: no , NoCore, , 1.0) : Regression test system without PVLIB usage.
#  top-p=0x2          # (int   , init-time) default = '0x2'    : test display name
#  top-str="empty"   # (string, init-time) default = 'empty'   : test string param
#  top-mm=0x3         # (int   , init-time) default = '0x6'    : test min(2) max(6) param : [0x2..0x6]
# Instance  1: a1    (SW: no , A, , 1.0) :
#  a1.p1=0x2          # (int   , init-time) default = '0x2'    : A parameter p1
#  a1.p2=0            # (bool  , run-time ) default = '0'      : A parameter p2
# Instance  2: a1.b  (SW: no , B, , 1.0) :
#  a1.b.p1=0x2        # (int   , init-time) default = '0x2'    : B parameter p1
#  a1.b.p2="test"    # (string, run-time ) default = ''      : B parameter p2
# Instance  3: a2    (SW: no , A, , 1.0) :
#  a2.p1=0x2          # (int   , init-time) default = '0x2'    : A parameter p1
#  a2.p2=0            # (bool  , run-time ) default = '0'      : A parameter p2
# Instance  4: a2.b  (SW: no , B, , 1.0) :
#  a2.b.p1=0x2        # (int   , init-time) default = '0x2'    : B parameter p1
#  a2.b.p2="test"    # (string, run-time ) default = ''      : B parameter p2
p2#-------------------------------------------------------------------------------------
```
For Example 2-1 on page 2-8, the instance information is provided as comment lines to enable the file to be used as input file to Model Shell. The values of the `top-mm` and `a2.b.p2` parameters are changed from their default values because new values for the parameters were specified on the command line.

Example 2-2 shows another example of specifying run-time parameters:

```
# Disable semihosting using true/false syntax
coretile.core.semihosting-enable=false
#
# Enable VFP at reset using 1/0 syntax
coretile.core.vfp-enable_at_reset=1
#
# Set the baud rate for UART 0
baseboard.uart_0.baud_rate=0x4800
```
2.4 SMP support

There are two usage patterns:

**Simple** This is only usable for systems that have one SMP core.
The same application is loaded in all CPUs. The application is listed in command line remainder:

```
model_shell smp_model.so app.axf
```

**Standard** This is usable for all cases and uses the `-a` option to list the applications for each CPU.

In addition to loading individual applications for each CPU, the `-a` option also enables loading the same application in all CPUs:

**Use -a to load the same application for all CPUs**

Use `*` as suffix for the core instead of its index. Use one of the following commands:

```
model_shell -m smp_model.so -a core.cpu*=app.axf
model_shell -m smp_model.so -a "core.*"=app.axf
```

Note On Unix, quotes might be required to escape `*` character.

**Use -a to load a different application to each CPU**

Use the full instance names for each CPU to specify an application for that CPU:

```
model_shell -m smp_model.so -a core.cpu0=A.axf -a core.cpu1=B.axf
```
2.5 Stopping Model Shell

Stopping simulation execution and terminating model shell can be done by user action or conditionally by specifying command line arguments.

2.5.1 Manual shutdown of Model Shell

To manually stop Model Shell execution, either:

- press Ctrl+C. The program starts shutting down the simulator and exits after shutdown is complete. Press Ctrl+C again to terminate Model Shell immediately.

  —— Note ———
  Some models can assign their own Control C handlers that override Model Shell behavior.

- press Ctrl+Break (Windows only) to terminate Model Shell immediately.

- stop simulation by, for example, closing an LCD window.

2.5.2 Automatic shutdown of Model Shell

To cause an automatic shutdown of Model Shell, specify stop conditions in the command line. Conditions can typically be combined so that the first fulfilled condition breaks the execution.

—– Note ———
Not all stop conditions are available in CADI server mode.

The following conditions are supported for automatic shutdown of Model Shell:

- the simulation stops.

  —— Note ———
  This if the default condition if no CADI server was started.
  If a CADI server was started with the --cadi-server option, this condition is ignored.

- the simulation hits a breakpoint. Use the --break option to specify the breakpoint.

  —— Note ———
  If a CADI server was started with the --cadi-server option, this condition is ignored.
This option cannot be combined with the --cyclelimit option.

----------

- the number of cycles is exceeded. Use the --cyclelimit option to specify the cycle limit.

Note

This option cannot be combined with the --cadi-server option.

Using this option might slow down execution speed.

Breakpoints are ignored.

----------

- the time limit is exceeded. Use the --timelimit option to specify the running time limit in wall-clock seconds.

- the cpu limit is exceeded. Use the --timelimit option to specify the running time limit in CPU seconds.

Note

A time granularity of 0.1 second is used to test against the specified time to avoid performance loss.

Both user and kernel time are included in the measurement of elapsed CPU time.