Model Shell for Fast Models

Version 8.1

Reference Manual

ARM

ARM DUI0457J

Model Shell for Fast Models

Reference Manual

2008-2013

Release Information

Document History

Issue	Date	Confidentiality	Change
A	31 December 2008	Non-Confidential	Release for Fast Models v.4.1
B	31 March 2009	Non-Confidential	Update for Fast Models v.4.2
C	30 April 2009	Non-Confidential	Release for Fast Models v.5.0
D	30 September 2009	Non-Confidential	Update for Fast Models v.5.1
E	28 February 2010	Non-Confidential	Update for Fast Models v.5.2
F	31 October 2010	Non-Confidential	Release for Fast Models v.6.0
G	31 May 2011	Non-Confidential	Update for Fast Models v.6.1
H	30 November 2011	Non-Confidential	Release for Fast Models v.7.0
I	31 December 2012	Non-Confidential	Release for Fast Models v.8.0
J	31 May 2013	Non-Confidential	Update for Fast Models v.8.1

Proprietary Notice

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

www.arm.com

Contents

Model Shell for Fast Models Reference Manual

List of Tables
Model Shell for Fast Models Reference Manual

Preface

This preface introduces the *Model Shell for Fast Models Reference Manual*.

It contains the following sections:

About this book

ARM Model Shell Reference Manual. This guide is technical documentation for the signaling, clock, bus, generic peripheral, and processor components included in the Fast Models software. These components provide a Programmer's View (PV) of the processor and peripheral components. It is available in HTML and PDF.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction to Model Shell

This chapter describes the main features of Model Shell, a command line tool for configuring and running a CADI-compliant model.

Chapter 2 Model Shell Commands

This chapter describes how to use Model Shell.

Glossary

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

See the *ARM Glossary* for more information.

Typographic conventions

italic	Introduces special terminology, denotes cross-references, and citations.
bold	Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.
`monospace`	Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.
`mono`space	Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.
`monospace italic`	Denotes arguments to monospace text where the argument is to be replaced by a specific value.
`monospace bold`	Denotes language keywords when used outside example code.
`<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 *mailto:errata@arm.com*. Give:

- The title.
- The number ARM DUI0457J.
- The page number(s) to which your comments refer.
- A concise explanation of your comments.

ARM also welcomes general suggestions for additions and improvements.

Other information

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

Chapter 1
Introduction to Model Shell

This chapter describes the main features of Model Shell, a command line tool for configuring and running a CADI-compliant model.

It contains the following sections:

1.1 About Model Shell

Model Shell is a command line tool for configuring and running Component Architecture Debug Interface (CADI)-compliant models.

Model Shell launches CADI-compliant models and provides:

- Semihosting stdio.
- CADI logging.
- A launch platform for debuggers, profilers, and operating environments.

Model Shell can start a CADI server 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 the simulation after it starts.

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

- When a CADI server 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 ISIM targets

Integrated SIMulator (ISIM) targets consist of Model Shell and a CADI model library.

Fast Models can create ISIM targets by statically linking Model Shell with a CADI library of a model.

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.

Related Information

The Fast Models User Guide.

Chapter 2
Model Shell Commands

This chapter describes how to use Model Shell.

It contains the following sections:

2.1 Model Shell command-line syntax

The correct arrangement for Model Shell commands, for tailoring the behavior of models.

Syntax

To start Model Shell from the command line, type `model_shell` with any options.

```
model_shell -m model [options]
```

model File name, including `.so` or `.dll` extension, for the model.

options List of command-line options.

Related References

Model Shell command-line options on page 2-14.
Configuration file syntax for specifying model parameters on page 2-17.

2.2 Model Shell command-line options

Use options to tailor Model Shell behavior from the command line.

Table 2-1 Model Shell command line options

Long	Short	Description

`--application` *filename1 filename2*	`-a`	Load application list.
		Use `-a` *instance=filename* to load an application to a system instance.
		`model_shell $MODEL -a multiprocessor.processor0=app1.axf -a multiprocessor.processor1=app2.axf`
		Use `*` to load the same application image into all the processors in a multiprocessor:
		`model_shell $MODEL -a "multiprocessor.*=app.axf"`
		Without a specified processor, Model Shell loads the image into the first instance that can run software. If this is an MPCore, Model Shell loads the image into each processor.

`--break` *address*	`-b`	Set program breakpoint at the address. Use `-b` *instance=address* to set a breakpoint for the system instance. Multiple `--breaks` set multiple breakpoints.
`--cadi-log`	`-L`	Log all CADI calls to an XML log file.
`--cadi-server`	`-S`	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.
`--cadi-trace`	`-t`	Enable diagnostic output from CADI calls and callbacks.
`--config-file` *filename*	`-f`	Use model parameters from configuration file *filename*.
`--cpulimit` *n*	-	Specify the maximum number of host seconds for the simulation to run, excluding simulation start-up and shut-down. 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.
`--cyclelimit` *n*	-	Specify the maximum number of cycles to run. If *n* is omitted, the default is unlimited.
`--data` *file@address*	`-d`	Specify raw data to load at this address. The full format is: `-d [`*INST*`=]`*file*`@[`*memspace*`:]`*address*

Table 2-1 Model Shell command line options (continued)

Long	Short	Description
`--dump` *file*`@`*address*`,`*size*	`-u`	Dump a section of memory into file, on model shut-down. Multiple `--dumps` are possible. The full format is: `-u [`*INST*`=]`*file*`@[`*memspace*`:]`*address*`,`*size*
`--help`	`-h`	List the Model Shell command line options, and then exit.
`--keep-console`	`-K`	Keep console window open after completion, on Microsoft Windows.
`--list-instances`	-	List target instances.
`--list-memory`	-	List memory information for the model to standard output.
`--list-params`	-	List target instances and their parameters. Use this to help identify the correct syntax for configuration files, and to find out what the target supplies.
`--list-regs`	-	List model register information to standard output.
`--output` *filename*	`-o`	Redirect output from the `--list-instances`, `--list-memory`, `--list-params`, and `--list-regs` commands to a file. The contents of the file are formatted correctly for use as input by the `--config-file` option.
`--parameter` [*instance.*]*parameter=value*	`-C`	Set a parameter to this value. For hierarchical systems, specify the complete name of the parameter.
`--plugin` *filename*	-	Specify plugins. These plugins or those in environment variable `FM_PLUGINS` are loaded.
`--prefix`	`-P`	Prefix semihosting output with the name of the target instance.
`--quiet`	`-q`	Suppress Model Shell output.
`--run`	`-R`	Run simulation on load, with a CADI server: `-S --run`. The default is to start the simulation in a stopped state.
`--start` *address*	-	Initialize the PC to this application start address, overriding the `.axf` start. The full format is: `--start [`*INST*`=]`*address*
`--stat`	-	List statistics at end of simulation.

Table 2-1 Model Shell command line options (continued)

Long	Short	Description
`--timelimit` *n*	`-`	Specify the maximum number of seconds to run as *n*. If *n* is omitted, the default is unlimited. **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`
`--verbose`	`-V`	Enable verbose messages.
`--version`	`-v`	List the Model Shell version number, then exit.

Related References

Model Shell command-line syntax on page 2-13.
Model Shell shutdown on page 2-19.

2.3 Configuration file syntax for specifying model parameters

Text files can configure models for Model Shell from the command line, thus setting many parameters at once.

Syntax

```
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.

Include comment lines and blank lines in configuration files with a `#` 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.

Examples

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

might generate:

```
# 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=""                 # (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
#-------------------------------------------------------------------------------
```

This is another way 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
```

Related References

Model Shell command-line syntax on page 2-13.

2.4 SMP support

Model Shell provides Symmetric MultiProcessing support. It can be simple or standard.

Simple This is only suitable for model systems that have one SMP multiprocessor. The same application is loaded in all processors.

```
model_shell -m smp_model.so -a app.axf
```

Standard This is suitable for all cases and uses the `-a` option to list the applications for each processor.

Use the full instance name of each processor.

```
model_shell -m smp_model.so -a multiprocessor.processor0=app1.axf -a
multiprocessor.processor1=app2.axf
```

In addition to loading individual applications for each processor, the `-a` option also enables loading the same application in all processors.

Replace the index of the processor with `*`.

```
model_shell -m smp_model.so -a multiprocessor.processor*=app.axf
```

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

——— **Note** ———

On Unix, the `*` character requires escape quotes.

———————————

2.5 Model Shell shutdown

This section describes the actions that stop Model Shell manually, and the options that stop it automatically.
It contains the following sections:

2.5.1 Manual Model Shell shutdown

User actions that stop Model Shell.

Action	Result
Press `Ctrl+C`.[a]	The program starts shutting down the simulator and exits after shutdown is complete. On a second press, Model Shell terminates immediately.
Press `Ctrl+Break`.[b]	Model Shell terminates immediately.
Close an LCD window.	The simulation stops.

2.5.2 Automatic Model Shell shutdown

Command-line options that stop Model Shell.

Option	End point
None[c]	Simulation end.
`--break`[c]	Breakpoint.
`--cyclelimit`[cd]	Cycles > cycle limit.
`--timelimit`	Time > running time limit.
`--cpulimit`[e]	Time > process time limit.

Note

The first fulfilled condition stops Model Shell.

[a] Some models can assign their own `Ctrl+C` handlers that override Model Shell behavior.
[b] Windows only.
[c] `--cadi-server` overrides this.
[d]
- Might reduce execution speed.
- Ignores breakpoints.

[e]
- Tested to a granularity of 0.1s to avoid performance loss.
- Elapsed processor time includes user time and kernel time.

2.6 License checking messages from Model Shell and ISIM systems

The license checking messages appear in the `stderr` and `stdout` outputs, and are useful for the detection and diagnosis of licensing issues.

The `model_shell` and `isim_system` executables return a status value when they exit:

`0`

no error (for example, clean simulator shutdown).

`1`

error (for example, license checking or file not found).

For exit status 1, parse the `stderr` output. A message might, for example, appear in the GUI, with other `WARNING`, `ERROR` or `Fatal Error` messages. See the lines that follow for more information from the license checking module. When a license is about to expire, Model Shell prints a warning message to `stdout`, but the simulation still starts correctly.

Examples

- ```
  ERROR: License check failed!
  Either the license file or the license server could not be found.
  Please set the environment variable 'ARMLMD_LICENSE_FILE'
  to your license file location or refer to the ARM_FLEXnet_Guide
  for instructions on where to obtain a license file, where to install
  the license file, and how to setup a license server.
  Error Code : -1
  ```
- ```
  ERROR: License check failed!
  No licenses 'FM_Simulator' available.
  No such feature exists.
  License files searched:
  h:\tmp\win\warningtest_ANY_04-feb-2011.dat
  Error Code : -5
  ```
- ```
  WARNING:
  Licenses 'SG_ARM1176_CT' expire in 0 days.
  Please contact ARM support to renew your license or to receive a new license.
  License files searched:
  h:\tmp\win\warningtest_ANY_04-feb-2011.dat
  ```
  ```