Arm® DSM

for Cortex-R52 Processor

User Guide

arm

101167_1000_01 (ID110618)

Arm® DSM
for Cortex-R52 Processor User Guide

Release Information

The following changes have been made to this book.

Change history

Date	Issue	Confidentiality	Change
November 2017	0100-00	Non-Confidential	Initial release
September 2018	1000-00	Non-Confidential	Release 10.0
November 2018	1000-01	Non-Confidential	Release 10.0 Documentation Update

Proprietary Notice

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

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.

Product Status

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

Web Address

http://www.arm.com

Contents
Arm Design Simulation Model User Guide

Preface
About this book viii
Intended audience viii
Using this book viii
Glossary viii
Conventions viii
Additional reading ix
Feedback x
Feedback on this product x
Feedback on content x

Chapter 1 Introduction
1.1 About Design Simulation Models 1-2
1.1.1 Features of Arm DSMs 1-2
1.2 DSM package contents 1-3
1.3 Simulation with DSMs 1-4

Chapter 2 Initial DSM Configuration
2.1 Prerequisites 2-2
2.2 Extracting and Installing the DSM 2-3
2.2.1 Installing Using the Command Line 2-3
2.3 How to test the DSM 2-4
2.4 Integrating with Simulators 2-5
2.4.1 VCS 2-5
2.4.2 Cadence Incisive 2-5
2.4.3 Mentor QuestaSim (ModelSim) 2-6
2.5 Configuring TARMAC Trace 2-7
2.5.1 Enabling TARMAC Trace 2-7
2.5.2 Disabling TARMAC Trace 2-7

Chapter 3 **Model Limitations**
3.1 Supported simulators 3-2
3.2 Unsupported simulator functions 3-3
3.3 Internal scan chain modeling 3-4
3.4 Caches and registers 3-5
3.5 Waveform dumping 3-6

Preface

This preface introduces the *Arm® Design Simulation Model (DSM) User Guide*. It contains the following sections:

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

About this book

This book is for the *Arm Design Simulation Model* (DSM) for the Cortex-R52 processor.

Intended audience

This book is written for experienced hardware engineers, software engineers and System-on-Chip (SoC) designers who might have experience of Arm products.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this for a high-level description of DSMs and how they are used in simulations. This chapter describes the contents of the package and potential simulation inaccuracies when using DSMs.

Chapter 2 *Initial DSM Configuration*

Read this for a description of how to install, set up, and test a DSM.

Chapter 3 *Model Limitations*

This chapter describes some of the limitations of DSMs.

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.

The *Arm® Glossary* is available on the Arm Infocenter at http://infocenter.arm.com/help/topic/com.arm.doc.aeg0014-/index.html.

Conventions

Conventions that this book can use are described in:

- *Typographical conventions*.

Typographical conventions

The following table describes the typographical conventions:

Style	Purpose
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.

(continued)

Style	Purpose
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, UNKNOWN, and UNPREDICTABLE.

Additional reading

This section lists publications by Arm and by third parties.

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

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.
- 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.
- The number, 101167_1000_01.
- The page numbers to which your comments apply.
- 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.

Chapter 1
Introduction

This chapter introduces the *Arm Design Simulation Model* (DSM) for the Cortex-R52. It contains the following sections:

- *About Design Simulation Models on page 1-2*.
- *DSM package contents on page 1-3*.
- *Simulation with DSMs on page 1-4*.

1.1 About Design Simulation Models

DSMs are cycle accurate, simulation models that you can include in a range of target HDL simulators. Each DSM is specific to a host platform. The DSM fully matches the architecture and functionality of the RTL model.

The DSMs are derived directly from the RTL model. DSMs can function with a wide range of industry-standard Verilog simulators. Peformance depends on:

- The simulator interface efficiency.
- The complexity of the design in which it is instantiated.
- The complexity of the original design.

The DSM consists of:

- A functional core block.
- A SystemC wrapper.

The wrapper uses the foreign language interface of the host simulator to instantiate the functional model. The DSM is generally derived from the RTL source of the Arm design using the Arm Cycle Model Compiler. For some Arm products, this might be augmented with extra functionality, such as TARMAC trace, added by Arm.

The DSM interfaces to the wrapper using technology developed by Arm to enable a single compiled model to function with a variety of logic simulators. For some products, the DSMs might include behavioral debug facilities, such as TARMAC trace. When you use compiled models, it enables distribution of models without compromising the intellectual property that they embody.

Note

The DSM is intended for functional simulation and verification only, not for timing or final design signoff.

1.1.1 Features of Arm DSMs

Table 1-1 shows the main features of Arm DSMs.

Table 1-1 Arm DSM features

Feature	Description
Full device functionality	Unless otherwise noted, the DSM fully matches the architecture and functionality of the RTL model.
Phase accuracy	You can expect the DSM to exhibit the same intra-cycle timing as the RTL model.
Register visibility	For register visibility, use the TARMAC trace or see the included README for a list of registers that support visibility. Refer to the *Arm Tarmac Specification* available with your DSM installation.
Cache and memory size configuration	You can configure the size of the cache, or TCM, for each particular DSM instance, where applicable. **Note** This feature does not apply if your Arm product has no such configuration, or where your DSM generation flow does not support multiple configurations.
Disassembler	To view disassembled code, use the TARMAC trace.

1.2 DSM package contents

Each DSM contains the following components:

- DSM SystemC wrapper.
- DSM library, including a dynamic .so file, static .a file, and header files.
- One or more implementation libraries as .so files.
- A testbench file to check the DSM setup.
- Documentation, including the *Arm TARMAC Specification*, this document, and a README.

1.3 Simulation with DSMs

Figure 1-1 shows the integration of DSMs (Block 1 and Block 2) into an HDL simulation. During simulation, the DSM interfaces to the simulator through the HDL to the SystemC interface, which synchronizes between the two environments.

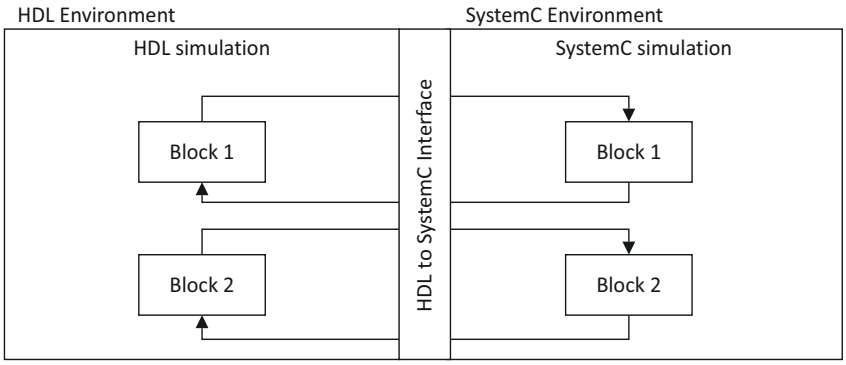

Figure 1-1 DSM integration

Note

For synthesizable cores, the DSM is a pre-implementation, pre-synthesis model. It does not contain any scan test insertion or BIST and is not a *Sign-Off Model* (SOM).

Chapter 2
Initial DSM Configuration

This chapter describes the functionality of the *Design Simulation Model for SystemC* (DSM).

It contains the following sections:

- *Prerequisites on page 2-2*
- *Extracting and Installing the DSM on page 2-3*.
- *How to test the DSM on page 2-4*.
- *Integrating with Simulators on page 2-5*.
- *Configuring TARMAC Trace on page 2-7*

2.1 Prerequisites

This section describes prerequisites to using the DSM.

- DSM built and downloaded in .tgz format from Arm IP Exchange (*www.armipexchange.com*).
- GCC Compiler. If you are using Synopsys VCS, refer to the Synopsys documentation for the required gcc versions.
- Refer to the README file in your installation directory for the specific tool versions required, model-specific limitations or restrictions, and supported registers.

2.2 Extracting and Installing the DSM

Before proceeding, ensure you have built and downloaded the <*dsm package*>.tgz for the desired IP from Arm IP Exchange (*www.armipexchange.com*).

2.2.1 Installing Using the Command Line

To install using the command line, follow the instructions below:

1. cd to the directory where you wish to install the DSM.
2. Extract the files from the <*dsm package*>.tgz as follows:

```
% tar -xvzf <dsm package>.tgz
gcc/docs
gcc/dsm
gcc/testbench
…
```

The gcc directory now contains the DSM release.

2.3 How to test the DSM

Note

You must set the DSM_PATH environment variable to point to the directory that contains the DSM implementation libraries. If you do not set the environment variable, the ${DSM_MODEL_NAME}_TESTBENCH.sh script assume the DSM_PATH is one directory above the scripts location in the dsm directory. If the path is not set correctly, the simulator compiler exits with an error looking for the [${DSM_MODEL_NAME}] files or library files.

For all simulators, the model search path must include the path to the DSM libraries. Reference the *_DSM_TESTBENCH.sh for a detailed example. In the following sections, variables are defined as follows:

$DSM_PATH — Path the DSM release dsm directory (for example, $DSM_RELEASE_DIR/dsm).

${DSM_MODEL_NAME} — Component name specified on Arm IP Exchange.

* — Arm IP name.

Execute the ${DSM_MODEL_NAME}_DSM_TESTBENCH.sh executable within the expanded package to check that operation is valid. The DSM is contained within a testbench that is supplied with the package. The script requires that you select a system-installed simulator. You can specify the simulator to use by setting one of the following parameters:

vcs — Selects the VCS simulator.

ius — Selects the Cadence Incisive simulator.

mti — Selects the QuestaSim simulator.

Note

The DSM is not intended for use with Accellera (OSCI) simulations.

A test that executes correctly prints the following message at the end of the simulation:

```
DSM: *** TEST PASSED ***
```

2.4 Integrating with Simulators

To use the DSM Model, add the following environment variables to your environment:

bash commands:

- export PATH=$DSM_PATH/univentUtil/bin:$PATH
- export LD_LIBRARY_PATH=$DSM_PATH/univentUtil/lib:$DSM_PATH:$LD_LIBRARY_PATH
- export DSM_PATH=*your_release_dir*/dsm
- export DSM_MODEL_NAME=*component_name_from_portal*

C shell commands:

- setenv PATH $DSM_PATH/univentUtil/bin:$PATH
- setenv LD_LIBRARY_PATH $DSM_PATH/univentUtil/lib:$DSM_PATH:$LD_LIBRARY_PATH
- setenv DSM_PATH *your_release_dir*/dsm
- setenv DSM_MODEL_NAME *component_name_from_portal*

Note

After these environment variables are defined, you can copy and paste the integration commands directly from this document into your command line. Use the commands in the sections *VCS*, *Cadence Incisive*, and *Mentor QuestaSim (ModelSim) on page 2-6*.

2.4.1 VCS

To integrate the DSM with VCS:

1. Create the Verilog wrapper:

   ```
   syscan -full64 -sysc=230 -cflags "-DCM_SYSC_IO_UNIVENT_TARMAC -DCM_SYSC_REMOVE_SCOPE
   -std=c++11"  $DSM_PATH/${DSM_MODEL_NAME}.cpp:${DSM_MODEL_NAME}
   ```

2. Add the following to the VCS command line:

   ```
   -full64 -sysc=230 -sysc=adjust_timeres -LDFLAGS "-L $DSM_PATH " $DSM_PATH/univentUtil/lib/*_tarmac_dpi.so
   $DSM_PATH/libcarbon5.so $DSM_PATH/lib${DSM_MODEL_NAME}.icm.so
   ${DSM_MODEL_PATH}/libicm_runtime.so $DSM_PATH/univent_tarmac.cpp
   ```

2.4.2 Cadence Incisive

To integrate the DSM with Cadence Incisive, add the following to the irun command line:

```
-64bit $DSM_PATH/univentUtil/lib/*_tarmac_dpi.so -sysc -scautoshell verilog -Wcxx,-std=c++11
-DCM_SYSC_IO_UNIVENT_TARMAC -DCM_SYSC_REMOVE_SCOPE -I $DSM_PATH -L$DSM_PATH
$DSM_PATH/${DSM_MODEL_NAME}.cpp $DSM_PATH/univent_tarmac.cpp -L$DSM_PATH -lcarbon5
$DSM_PATH/lib${DSM_MODEL_NAME}.icm.so ${DSM_MODEL_PATH}/libicm_runtime.so
```

2.4.3 Mentor QuestaSim (ModelSim)

To integrate the DSM with Mentor QuestaSim:

Before running vsim, compile and link the libraries as follows:

```
sccom -64 -g -suppress 6102 -std=c++11 -suppress 6165 -DCM_SYSC_IO_UNIVENT_TARMAC
-DCM_SYSC_REMOVE_SCOPE -I$DSM_PATH $DSM_PATH/${DSM_MODEL_NAME}.cpp
$DSM_PATH/univent_tarmac.cpp
```

```
sccom -64 -link -suppress 6102 -L $DSM_PATH -l ${DSM_MODEL_NAME} -l${DSM_MODEL_NAME}.icm -licm_runtime
-lcarbon5 $DSM_PATH/univentUtil/lib/*_tarmac_dpi.so
```

2.5 Configuring TARMAC Trace

This section describes enabling and disabling TARMAC trace.

2.5.1 Enabling TARMAC Trace

To enable the TARMAC trace, refer to the TARMAC specifications in the docs directory of the DSM release, or review the ${DSM_MODEL_NAME}_TESTBENCH.sh file for the *TARMAC* environment variables.

You must also compile with the define CM_SYS_IO_UNIVENT (for example, -DCM_SYS_IO_UNIVENT) to enable.

2.5.2 Disabling TARMAC Trace

To disable the TARMAC trace, remove the CM_SYS_IO_UNIVENT define from the compile. Or, set the environment variable *_TARMAC_ENABLE to never.

Chapter 3
Model Limitations

Although DSMs match the architecture and functionality of the appropriate core designs, they are subject to the limitations that the following sections describe:

- *Supported simulators on page 3-2*
- *Unsupported simulator functions on page 3-3*.
- *Internal scan chain modeling on page 3-4*.
- *Caches and registers on page 3-5*.
- *Waveform dumping on page 3-6*

3.1 Supported simulators

The DSM model is supported on Linux 64-bit, and the simulator version must be able tos support C++11. It has been tested and is supported on the following simulators:

- Synopsys VCS
- Cadence Incisive
- Cadence Xcelium
- Mentor QuestaSim

3.2 Unsupported simulator functions

The following simulator functions are not supported:

Save and Restore, also known as checkpointing

Saving the simulation at a determined point of time, also known as a snapshot, and restoring the simulation to that point of time is not supported.

Power-aware simulation

Power-aware simulation is not supported.

3.3 Internal scan chain modeling

DSMs are derived from the RTL description of the core that they model. The final netlist for the core might contain internal scan chains that were added during synthesis. It is not possible to use DSMs to model these scan chains because they do not exist in the device RTL. However, the *Sign-Off Model* (SOM) of a device models the scan chains.

3.4 Caches and registers

The DSM Model provides visibility into certain registers. Refer to the README file included with your DSM Model for a list of supported registers.

3.5 Waveform dumping

Refer to the README file included with your DSM Model for instructions and limitations of Waveform dumping for your particular model.