SystemC Cycle Models

Version 11.5

Reference Platform Getting Started Guide

arm

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
101497_1105_00_en

SystemC Cycle Models

Reference Platform Getting Started Guide

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.

Release Information

Document History

Issue	Date	Confidentiality	Change
1100-00	31 May 2019	Non-Confidential	First release
1103-00	13 November 2020	Non-Confidential	Release 11.3
1103-01	01 May 2021	Non-Confidential	Documentation update
1105-00	25 June 2021	Non-Confidential	Release 11.5

Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED "AS IS". ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, third party patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

If any of the provisions contained in these terms conflict with any of the provisions of any click through or signed written agreement covering this document with Arm, then the click through or signed written agreement prevails over and supersedes the conflicting provisions of these terms. This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm's trademark usage guidelines at *https://www.arm.com/company/policies/trademarks*.

Copyright © 2019–2021 Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

(LES-PRE-20349)

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

developer.arm.com

Progressive terminology commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive terms. If you find offensive terms in this document, please contact *terms@arm.com*.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Contents

SystemC Cycle Models Reference Platform Getting Started Guide

Preface

About this book 7

Chapter 1 Introduction to Cycle Model reference platforms

1.1 Introduction to Cycle Model reference platforms 1-10
1.2 System requirements and prerequisites 1-11
1.3 Reference platform directory structure 1-12
1.4 Data collection in Cycle Model reference platforms 1-13

Chapter 2 Building and running the default reference platform

2.1 Download a reference platform from IP Exchange 2-15
2.2 Decompress the reference platform package file 2-16
2.3 Build the reference platform 2-17
2.4 Run the reference platform 2-18
2.5 Next steps 2-20

Chapter 3 Introduction to the makefiles

3.1 Makefiles included in the reference platform 3-22
3.2 Make options for the reference platform makefile and Systems/makefile 3-23

Chapter 4 Modifying reference platforms

4.1 Making build and link options available to the Makefile 4-25

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.2 Internal changes to a model in the default reference platform 4-27
4.3 Changing the core used in the reference platform 4-28
4.4 Changes to the composition of the reference platform 4-29
4.5 Creating custom systems that include Arm® models 4-31
4.6 Loading applications for simulation 4-32
4.7 Modifying the Cycle Model reference platform test bench 4-33

Chapter 5 Troubleshooting

5.1 carbon_sc_multiwrite_signal.h build error 5-38
5.2 Unrecognized command line option 5-39
5.3 Licensing errors 5-40
5.4 Bytes requested error when specifying application 5-41
5.5 Multiple target patterns error 5-42

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Preface

This guide describes downloading, installing, and running Cycle Model reference systems.

This preface introduces the *SystemC Cycle Models Reference Platform Getting Started Guide*.

It contains the following:

- *About this book* on page 7.

This guide describes downloading, installing, and running Cycle Model reference systems.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

About this book

This guide describes downloading, installing, and running SystemC-based Cycle Model reference platforms.

Using this book

This book is organized into the following chapters:

Chapter 1 Introduction to Cycle Model reference platforms
This chapter introduces Arm Cycle Model reference platforms.

Chapter 2 Building and running the default reference platform
This chapter describes downloading, compiling, and simulating the Cycle Model SystemC default reference platform.

Chapter 3 Introduction to the makefiles
This chapter summarizes the makefiles that are included in Cycle Model reference platforms and describes the available targets.

Chapter 4 Modifying reference platforms
This chapter describes modifications you can make to SystemC Cycle Model reference platforms, and how to rebuild the reference platform.

Chapter 5 Troubleshooting
This chapter provides solutions for problems that may occur when working with Cycle Model reference platforms.

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.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

`<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 *SystemC Cycle Models Reference Platform Getting Started Guide*.
- The number 101497_1105_00_en.
- 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® Developer*.
- *Arm® Documentation*.
- *Technical Support*.
- *Arm® Glossary*.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Chapter 1
Introduction to Cycle Model reference platforms

This chapter introduces Arm Cycle Model reference platforms.

It contains the following sections:

- *1.1 Introduction to Cycle Model reference platforms* on page 1-10.
- *1.2 System requirements and prerequisites* on page 1-11.
- *1.3 Reference platform directory structure* on page 1-12.
- *1.4 Data collection in Cycle Model reference platforms* on page 1-13.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

1.1 Introduction to Cycle Model reference platforms

Cycle Model reference platforms are prepackaged systems ready to simulate with a default application.

The applications and components that ship with reference platforms may differ, but all reference platforms have the same general directory structure, environment configuration, and processes for creating and executing the test bench.

After the reference platform is simulating with the default application, you can:

- Make changes to it by modifying the reference platform test bench and corresponding build system to include, instantiate, and connect new or updated models.
- Copy and migrate a SystemC Cycle Model that is part of a reference platform, and build it into your own custom platform.
- Modify an Arm reference platform by adding your own SystemC model classes to the reference platform.

For use cases and instructions, see *Chapter 4 Modifying reference platforms* on page 4-24.

Included in the package

Reference platforms include:

- Model libraries for the models included in the reference platform.
- SystemC top-level system.
- Sample applications.
- Setup scripts.
- Cycle Model SystemC Runtime, which also includes:
  - The Cycle Model Studio runtime.
  - SystemC Version 2.3.1 (64-bit Linux).
  - Google Protocol Buffer.
  - The Cycle Model Configuration tool, a command-line utility that simplifies integration of Arm SystemC Cycle Models into custom systems. For usage, see the *SystemC Cycle Model User Guide* for the CPU in your reference platform.

Related information

- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.
- *Arm® SystemC Cycle Model Runtime Installation Guide* (101146).

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

1.2 System requirements and prerequisites

This section describes space, operating system, and software requirements for running Arm SystemC reference platforms.

Disk space

The Cycle Model SystemC Runtime requires 200MB of disk space.

For Cycle Model reference platform reference platforms, more space is required. The amount of space needed varies depending on the complexity of the reference platform.

Supported operating systems

The supported Linux operating systems are:

- Red Hat Enterprise Linux 7.0 (64-bit)

Cycle Model reference platforms are not supported on Windows.

Supported GCC versions

For rebuilding Cycle Model reference platforms, GCC version 4.8.3 and GCC version 6.4.0 are supported.

Prerequisites for reference platform environments

All models in a Cycle Model reference platform reference platform must be the same release (for example, all v10.x or all v11.x). Mixing different versions within a reference platform is not supported, and results in incorrect Cycle Model behavior, incorrect Tarmac trace results, or other issues.

Licensing

You must have a valid, installed license for each runtime and Cycle Model. Visit the Arm licensing portal: *https://developer.arm.com/support/licensing/generate* and use your serial numbers to generate the licenses. Contact Arm Technical Support (support-esl@arm.com) if you need more information.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

1.3 Reference platform directory structure

This section describes the directory structure that all Cycle Model reference platforms share.

Using a Cortex-R52 reference platform as an example, the following figure describes the general reference platform directory structure and summarizes its contents. Certain reference platforms may have additional directories.

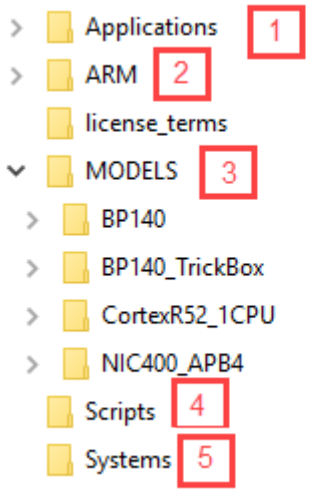

Figure 1-1 Reference platform directory structure

1. The `Applications` directory contains the source code, build files, and executable images for the sample applications that come with your reference platform.
2. The `ARM` directory contains the Cycle Models SystemC Runtime, Cycle Model Studio runtime, and third-party files for SystemC and Google Protocol Buffer.
3. The `MODELS` directory contains model libraries and SystemC wrapper source code for all the models used to build the platform.
4. The `Scripts` directory contains setup scripts (`setup.sh` and `setup.csh`) used to configure environment variables. This directory may also contain scripts that the reference platform uses for application loading or compilation.
5. The `Systems` directory contains top-level design files used to connect models. It also includes the `system_test.cpp` file (where the `sc_main` function is located), and the `makefile` used to build the reference platform.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

1.4 Data collection in Cycle Model reference platforms

Arm periodically collects anonymous information about the usage of our products to understand and analyze what components or features you are using, with the goal of improving our products and your experience with them. Product usage analytics contain information such as system information, settings, and usage of specific features of the product. They do not include any personal information.

Host information includes:

- Operating system name, version, and locale.
- Number of CPUs.
- Amount of physical memory.
- Screen resolution.
- Processor and GPU type.

Note

To disable analytics collection for all tools running in the environment, set the environment variable `ARM_DISABLE_ANALYTICS` to any value, including 0 or an empty string. This setting is not saved in persistent storage. It must be reset at subsequent invocations of the tool.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Chapter 2
Building and running the default reference platform

This chapter describes downloading, compiling, and simulating the Cycle Model SystemC default reference platform.

It contains the following sections:

- *2.1 Download a reference platform from IP Exchange* on page 2-15.
- *2.2 Decompress the reference platform package file* on page 2-16.
- *2.3 Build the reference platform* on page 2-17.
- *2.4 Run the reference platform* on page 2-18.
- *2.5 Next steps* on page 2-20.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

2.1 Download a reference platform from IP Exchange

Download a Cycle Model reference platform from IP Exchange.

Before you begin

- You must have a valid account on *https://ipx.arm.com*. Create one if necessary.

Download

To download a Cycle Model reference platform from Arm IP Exchange:

1. Visit *https://ipx.arm.com/systems* and download a SystemC Cycle Model reference platform. Reference platforms are packaged in a `.tgz` file.

Next steps

Proceed to *2.2 Decompress the reference platform package file* on page 2-16.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

2.2 Decompress the reference platform package file

After download, decompress the reference platform package file and review the `README.txt` file.

Before you begin

- Download a reference platform from Arm IP Exchange (*https://ipx.arm.com/systems*).
- Review the system and licensing requirements in *1.2 System requirements and prerequisites* on page 1-11.

Decompress the reference platform and review the Readme

1. Untar the reference platform `.tgz` file. For example:

```
$ tar xzvf R52-MP2-MC2-SysC-V10.0.0-CMS10.0.0-MK2018.09.17-SOCD9.6.0.tgz
```

   The reference platform directory structure is created. See *1.3 Reference platform directory structure* on page 1-12 for an overview of the structure of the reference platform.
2. Open and review the `README.txt` file located at the top of the reference platform directory structure:

```
$ cd R52-MP2-MC2-SysC-V10.0.0-CMS10.0.0-MK2018.09.17-SOCD9.6.0
$ ls
Applications  ARM  license_terms  Makefile  MODELS  Readme_Debug_Memory_notes.txt
README.txt  Scripts    Systems
$ less README.txt
```

   The `README.txt` file:
   - Lists environment variables that are required to be set for this reference platform.
   - Lists other requirements and dependencies specific to your reference platform.
   - For pin-based reference platforms, states the application that the reference platform runs by default. See the `Applications` directory to determine all of the applications your reference platform includes. See *2.4 Run the reference platform* on page 2-18 for instructions on running applications.
   - Describes scripts included with your reference platform.

Next steps

Proceed to *2.3 Build the reference platform* on page 2-17.

Related information

- *2.1 Download a reference platform from IP Exchange* on page 2-15
- *1.3 Reference platform directory structure* on page 1-12

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

2.3 Build the reference platform

Build the reference platform executable.

Before you begin

- Decompress the reference platform package file and review the `README` as described in *2.2 Decompress the reference platform package file* on page 2-16.

Source the setup script and set required environment variables

Note

Setup scripts for C Shell and Bash are supported. Ensure you are in the correct environment.

1. In the reference platform `Scripts` directory, source the `setup.sh` or `setup.csh` script. You can do this by including the source command in a `makefile`, or from the command line:

```
$ cd Scripts/
$ ls
create_dat_file.sh  setup.csh  setup.sh  vars.csh  vars.sh
$ source setup.sh
Arm Cycle Model SystemC setup completed.
Setup complete.
$
```

2. If you are using a version of the Cycle Model Studio runtime other than the one included in the reference platform, set CARBON_HOME. If you are using the version of the Cycle Model Studio runtime included in the reference platform, skip this step.

Build the reference platform executable

1. `cd` to the reference platform `Systems` directory.
2. Build the reference platform executable (`system_test`) by entering `make`:

```
$ cd Systems
$ make
-std=c++11 -I/home/reference_systems/v11/R52-MP2-MC2-SysC-Vmainline-CMSmk.snapshot-
MKmainline-SOCDmainline/ARM/CycleModels/Runtime/cm_sysc/mainline -I/home/sandbox/
reference_systems/v11/R52-MP2-MC2-SysC-Vmainline-CMSmk.snapshot-MKmainline-
SOCDmainline/ARM/CycleModels/Runtime/cm_sysc/mainline/include
.
.
.
$
```

Next steps

Proceed to *2.4 Run the reference platform* on page 2-18.

Related information

- *Chapter 5 Troubleshooting* on page 5-37

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

2.4 Run the reference platform

Simulate with an application included with the reference platform.

Before you begin

- Build the reference platform executable as described in *2.3 Build the reference platform* on page 2-17.
- Ensure that the environment variable `ARMLMD_LICENSE_FILE` is set to your license server; for example, `export ARMLMD_LICENSE_FILE=`*`port@host`*.

Set runtime environment variables and run the simulation

1. Source the setup script to set environment variables that are required at runtime:

```
$ source ../Scripts/setup.sh
Arm Cycle Model SystemC setup completed.
Setup complete.
$
```

2. From the `Systems` directory, simulate the reference platform application using the `system_test` test bench:

```
$ ./system_test -a ../Applications/hello_world/armcc/elf/test.elf
```

- TLM-based reference platforms require you to specify the application to run using the `-a` or `--application` argument.
- For pin-based reference platforms, do not use `-a` or `--application`. Run `./system_test` to run the default application. The reference platform `README.txt` file lists the default application for pin-level reference platforms; see the `Applications` directory for additional applications provided with your reference platform.

See *4.6 Loading applications for simulation* on page 4-32 for more information about application loading.

Result

The test bench starts the simulation, and loads and runs the application specified by `system_test.cpp`. Following is sample output from a TLM-based Cortex-R52 reference platform:

```
$ ./system_test -a ../Applications/hello_world/armcc/elf/test.elf
Starting Simulation
[kite_tarmac] skipping configuration file: kite_tarmac.cfg
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[0].u_cpu.u_noram.u_follower)
detected capture module (enabled)
[M] Constructing kite follower
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[0].u_cpu.u_noram.u_follower)
created filter 'DECODE' (kite_tarmac_decode)
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[0].u_cpu.u_noram.u_follower)
sending captured stream to 'DECODE'
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[1].u_cpu.u_noram.u_follower)
detected capture module (enabled)
[M] Constructing kite follower
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[1].u_cpu.u_noram.u_follower)
created filter 'DECODE' (kite_tarmac_decode)
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[1].u_cpu.u_noram.u_follower)
sending captured stream to 'DECODE'
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[0].u_cpu.u_noram.u_follower)
detected capture module (enabled)
[M] Constructing kite follower
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[0].u_cpu.u_noram.u_follower)
created filter 'DECODE' (kite_tarmac_decode)
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[0].u_cpu.u_noram.u_follower)
sending captured stream to 'DECODE'
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[1].u_cpu.u_noram.u_follower)
detected capture module (enabled)
[M] Constructing kite follower
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[1].u_cpu.u_noram.u_follower)
created filter 'DECODE' (kite_tarmac_decode)
[kite_tarmac] (KITE.g_dcls_cpu_container.g_near_cpu_group[1].u_cpu.u_noram.u_follower)
sending captured stream to 'DECODE'
Starting Sim
UART0: Hello World!<0a>
UART1: Hello World!<0a>
```


Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

```
UART2: Hello World!<0a>
UART3: Hello World!<0a>
UART0: My name is Kite<0a>
UART1: My name is Kite<0a>
UART2: My name is Kite<0a>
UART3: My name is Kite<0a>
UART0: I wish you a great day<0a>
UART1: I wish you a great day<0a>
UART2: I wish you a great day<0a>
UART3: I wish you a great day<0a>
UART0: ** TEST PASSED OK **<0a>
UART1: ** TEST PASSED OK **<0a>
UART2: ** TEST PASSED OK **<0a>
UART3: ** TEST PASSED OK **<0a>
UART0: <04>
UART1: <04>
.                                                                              .
.
.
$
```

The simulation results show that the Hello World application ran successfully the four Cortex-R52 cores (output is sent through the UART model attached to each CPU). Simulation results also include performance monitoring data for each CPU (not shown in the example).

Next steps

When your reference platform simulates properly, see *2.5 Next steps* on page 2-20.

Related information

- *Chapter 5 Troubleshooting* on page 5-37
- *4.6 Loading applications for simulation* on page 4-32

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

2.5 Next steps

When your default reference platform is simulating properly, learn about and change its operation.

Review the processor model guide

The *SystemC Cycle Model User Guide* for the processor model in your reference platform describes the functionality of the model compared to the hardware.

The model guide includes instructions for viewing model parameters, enabling waveform dumping, resetting the model, and connecting to a debugger.

View the list of processor model parameters

The set of available parameters varies, depending on the type and functionality of the processor model. To list model parameters:

- Enter `./system_test --list-params`.
- View the *`model`*`_params.cfg` file located in the directory *`reference_platform`*`/MODELS/`*`CPU`*`/gcc`*`version`*`/SystemC`.

Modify the reference platform

You can change the reference platform to include different or additional models, make changes to the default application, or run the reference platform with a different application. See *Chapter 4 Modifying reference platforms* on page 4-24.

Related information

- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.
- *Arm® Development Studio Getting Started Guide* (101469).
- *Arm® Development Studio User Guide* (101470).

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Chapter 3
Introduction to the makefiles

This chapter summarizes the makefiles that are included in Cycle Model reference platforms and describes the available targets.

It contains the following sections:

- *3.1 Makefiles included in the reference platform* on page 3-22.
- *3.2 Make options for the reference platform makefile and Systems/makefile* on page 3-23.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

3.1 Makefiles included in the reference platform

This section explains what you need to know about makefiles included in the Cycle Model reference platform.

The reference platform directory includes the following `makefiles`:

/Systems makefile

The `Makefile` you will use most frequently when working with reference platforms is the `Systems/Makefile`. This is the command script for your build infrastructure. When you make a change to the reference platform, such as adding a model to your design or changing one model for another, rebuild the reference platform using this `Makefile`.

See *3.2 Make options for the reference platform makefile and Systems/makefile* on page 3-23 for reference platform build, run, and simulation options.

reference platform makefile

You can use the `Makefile` at the top level of the reference platform to execute the `Systems/Makefile`. This top-level `Makefile` sets certain environment variables.

/MODELS makefiles

Each model in a reference platform system includes its own makefiles, located in `MODELS/`*`CPU_name`*`/`*`gcc_version`*`/SystemC/makefile`.

Note

Do not modify the makefiles in the `MODELS` directories. If you are adding or replacing a model in the reference platform, see *Chapter 4 Modifying reference platforms* on page 4-24.

/Applications makefile

The `Applications/Makefile` defines the application that is launched by default when you run the reference platform simulation. Modify this `Makefile` if you want to simulate using a different application. Do not modify the initialization code contained in the `Applications/Makefile`.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

3.2 Make options for the reference platform makefile and Systems/makefile

This section describes the available build, run, and simulation targets; targets that are available only in the top-level Cycle Model reference platform `Makefile` are noted.

Use the `Systems/Makefile` to rebuild the reference system after making modifications. This `makefile` includes the following available targets:

`make all`
Builds the system.

`make app-setup [ APP=path ]`
Sets up the application that the system uses during simulation. `path` is the path to the compiled application (`.elf`) file. This option is available only in the top-level reference platform `makefile`, not the `Systems/makefile`. See *4.6 Loading applications for simulation on page 4-32* for information about application loading.

`make clean`
Removes all the binaries and object files.

`make help`
Prints all available targets. This option is available only in the top-level reference platform `makefile`, not the `Systems/makefile`.

`make req`
Prints out all the components and dependencies for the reference platform.

`make run [ APP=path ] [ RUNLOG=file_name ]`
Runs the simulation. `path` is the path to the compiled application (`.elf`) file and `file_name` is the name of the file for log output. If `APP` is not specified, it runs the default application or the one that was last set up using `app-setup`.

Note

Run this command only with TLM-based reference platforms, not with pin-based reference platforms. This command uses the `-a` flag as a means of passing in the application, and pin-based reference platforms do not accept the `-a` flag. See *4.6 Loading applications for simulation on page 4-32* for more information.

`make system`
Uses the reference platform test bench (`system_test.cpp`) and the library files to generate the `system_test` binary.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Chapter 4
Modifying reference platforms

This chapter describes modifications you can make to SystemC Cycle Model reference platforms, and how to rebuild the reference platform.

It contains the following sections:

- *4.1 Making build and link options available to the Makefile* on page 4-25.
- *4.2 Internal changes to a model in the default reference platform* on page 4-27.
- *4.3 Changing the core used in the reference platform* on page 4-28.
- *4.4 Changes to the composition of the reference platform* on page 4-29.
- *4.5 Creating custom systems that include Arm® models* on page 4-31.
- *4.6 Loading applications for simulation* on page 4-32.
- *4.7 Modifying the Cycle Model reference platform test bench* on page 4-33.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.1 Making build and link options available to the Makefile

When making changes to a Cycle Model reference platform, ensure that the build options for all models that are part of the reference platform are available to the `Makefile`. Your reference platform will not build successfully without the required build options for all components.

The required build and link options may be available to the Cycle Model Configuration Tool (`cm_config`) through the model's associated XML file, or you may need to explicitly specify them.

The availability of build and link options depends on the type of model being added to the reference platform:

- Model built with Cycle Model Studio - No associated, `cm_config`-readable XML file. Provide the build options explicitly to the `Makefile` as described in *4.1.1 Specifying build options for Cycle Models without an XML data file* on page 4-25.
- Custom model that you create – No associated, `cm_config`-readable XML file. Provide the build options explicitly to the `Makefile` as described in *4.1.1 Specifying build options for Cycle Models without an XML data file* on page 4-25.
- Model built on *https://ipx.arm.com* – These models are built with the required, `cm_config`-readable XML file. `cm_config` automatically pulls the required build and link information from these files when you run `make`.

Related information

- *3.2 Make options for the reference platform makefile and Systems/makefile* on page 3-23
- The SystemC Cycle Model guide for the CPU model in your reference platform. This guide contains additional information about the Cycle Models Configuration Tool, and about further customizing your build (such as specifying your own version of SystemC). Model Guides are found at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.

This section contains the following subsection:

- *4.1.1 Specifying build options for Cycle Models without an XML data file* on page 4-25.

4.1.1 Specifying build options for Cycle Models without an XML data file

If you add a model to your Cycle Model reference platform, but the model does not have an associated XML data file that is readable by the Cycle Model Configuration Tool (`cm_config`) tool, you can specify the build and link options for the new model using the `Makefile`.

The `Makefile` includes four variables that hold the build options for models you want to integrate into your reference platform:

- `CXXFLAGS` - Compiler flags, include paths, and compiler defines for the system. These are used when building the object files and linking them into the executable.
- `LDFLAGS` - Linker flags, libraries, and library paths for the system. These are passed through to the linker when linking object files into an executable.
- `RPATHS` - Runtime paths to associate with the executable. Prefix the paths with the string `-Wl, -rpath`. Alternatively, you can place the paths in your `LD_LIBRARY_PATH` environment variable.
- `SRCS` - Source files to be built into object files and linked into the system executable.

You can specify the build options held by these variables in the following ways:

- On the command line when calling `make`.
- By adding build options the variables in the `Makefile`.
- By modifying the `Makefile` directly using valid `make` syntax.

See the following examples.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Example: Specifying build options on the command line when calling make

The following example shows how to add a new source and header file for a custom SystemC component by specifying them on the command line:

```
make run CXXFLAGS=-Ipath/to/my_header_dir/ SRCS=path/to/my_source.cpp
```

Example: Adding build options to the variables

The following examples shows how to add new source and header files for a custom SystemC component by overriding the variables:

```
override CXXFLAGS+=$(shell $(CM_CONFIG_CMD) --compile)
override CXXFLAGS+=-Ipath/to/my_header_dir/
…
override SRCS+=$(shell $(CM_CONFIG_CMD) --sources --model-type $(CONNECTION))
override SRCS+=path/to/my_source.cpp
```

Example: Modifying the Makefile directly

The following example shows how to add new source and header files for a custom SystemC component by adding them to the targets:

```
System_test: $(OBJS) path/to/my_source.o
      $(CXX) $(CXXFLAGS) -Ipath/to/my_header_dir/  $^   \
      $(RPATHS) $(LDFLAGS) -o $@

%.o: %.cpp
      $(CXX) $(CXXFLAGS) -Ipath/to/my_header_dir/  -o $@ -c $^
```

Related information

- *3.2 Make options for the reference platform makefile and Systems/makefile* on page 3-23

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.2 Internal changes to a model in the default reference platform

This use case describes making a change to a model that exists in the default Cycle Model reference platform.

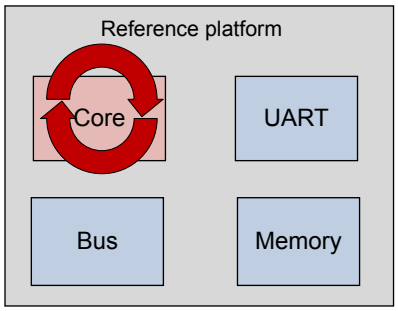

Figure 4-1 Change to existing model

You can make changes in the source files for models included in the default reference platform. For example, you might drive an additional input by commenting out a port binding in the reference platform `MODELS/`*`model`*`/gcc_`*`version`*`/SystemC/`*`model`*`ResetImp.h` file. In this case, ensure that you also drive the value of the port.

1. Make your modifications in `MODELS/`*`model`*`/gcc_`*`version`*`/SystemC/`*`model`*`ResetImp.h`.
2. Rebuild the reference platform using the `Systems/makefile`.

   See the *SystemC Cycle Model User Guide* for your IP for information about binding or tying ports.

Related information

- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.3 Changing the core used in the reference platform

This use case describes replacing the core in a Cycle Model reference platform.

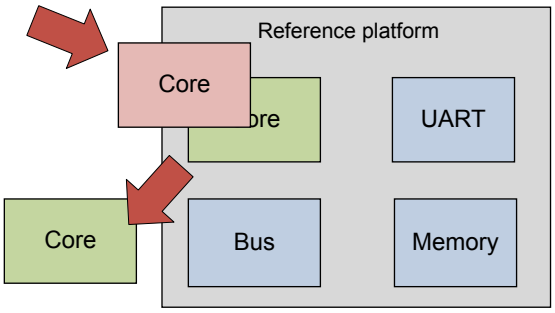

Figure 4-2 Changing the core

You can replace the core included in the reference platform by default with:

- a core of the same IP type, but with a different build configuration.
- a core of a different IP type.

Note

If the new core is built on Arm IP Exchange (*https://ipx.arm.com*), the Cycle Model Configuration Tool (`cm_config`) finds its associated XML data file when you run `make`. If the new core was not built on Arm IP Exchange, see *4.1 Making build and link options available to the Makefile* on page 4-25.

To integrate the new core into your reference platform:

1. Build the new or revised core.
2. Replace the existing core in the reference platform `MODELS` directory.
3. If the new core is of a different IP type than the original core, update the `system_test.cpp` test bench to reference the header files, ports, and bindings of the new core.
4. In the `Systems/makefile`, update the `COMP_NAMES` variable and the model directory to reflect the new model. To get the exact component name, run the Cycle Models Configuration tool with the `--list` argument and look at the `Component Type: model:` section. For example, in the following output, `CortexR52` is the string to use in the `COMP_NAMES` variable:

```
$ cm_config --list
Component Type: model:
CortexR52                 mainline        /home/reference_platforms/R52-MP2-MC2-SysC/
MODELS/CortexR52_2CPU/gcc640/SystemC/.data/CortexR52.xml
CortexR52                 mainline        /home/reference_platforms/R52-MP2-MC2-SysC/
MODELS/CortexR52_2CPU/gcc483/SystemC/.data/CortexR52.xml
cm_sysc_models            mainline        /home/reference_platforms/R52-MP2-MC2-SysC/ARM/
CycleModels/Runtime/cm_sysc/mainline/etc/.data/cm_sysc_models.xml
Component Type: runtime:
.
.
.
```

   For more information about the Cycle Models Configuration Tool, see the *SystemC Cycle Model User Guide* for the CPU model included in your reference platform.
5. Rebuild the reference platform using the `Systems/makefile`.

Related information

- *3.2 Make options for the reference platform makefile and Systems/makefile* on page 3-23
- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.4 Changes to the composition of the reference platform

This use case describes making changes to the models that make up a Cycle Model reference platform.

Prerequisites

When you run `make` after adding a new component to the reference platform, the Cycle Model Configuration Tool (`cm_config`) extracts the required build and link options for all reference platform components that were built on Arm IP Exchange. These models have an associated XML data file that is readable by `cm_config` .

However, if you are adding components to the reference platform that were not built on Arm IP Exchange (such as those built using Cycle Model Studio or custom models), you must provide build and link data to the `Makefile`. See *4.1 Making build and link options available to the Makefile* on page 4-25.

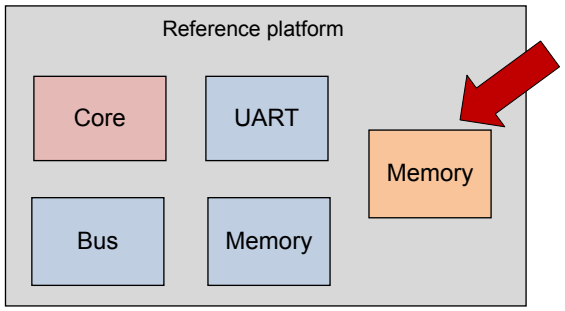

Figure 4-3 Adding a flash memory model to an existing reference platform

This section describes adding models to, or removing models from the reference platform platform; for example, adding a flash memory to the existing system.

1. If you are adding a model to the reference platform, add its directory to the `reference_platform/MODELS` directory. Ensure that the new component has an associated `.XML` file.
2. If you have removed a model from the reference platform, remove its corresponding directory from `reference_platform/MODELS`.
3. Update `Systems/system_test.cpp`. This file defines the models included in the reference platform and their port bindings.
4. If the name of the model directory has changed, further modifications to the `Systems/Makefile` are required. Update the `COMP_NAMES` variable to include models and directories you are adding, and exclude models and directories you are replacing. To get the exact component name, run the Cycle Models Configuration tool with the `--list` argument and look at the `Component Type: model:` section. For example, in the following output, `CortexR52` is the string to use in the `COMP_NAMES` variable:

```
$ cm_config --list
Component Type: model:
CortexR52                  mainline        /home/reference_platform/R52-MP2-MC2-SysC/
MODELS/CortexR52_2CPU/gcc640/SystemC/.data/CortexR52.xml
CortexR52                  mainline        /home/reference_platform/R52-MP2-MC2-SysC/
MODELS/CortexR52_2CPU/gcc483/SystemC/.data/CortexR52.xml
cm_sysc_models             mainline        /home/reference_platform/R52-MP2-MC2-SysC/ARM/
CycleModels/Runtime/cm_sysc/mainline/etc/.data/cm_sysc_models.xml
.
.
.
```

5. Rebuild the reference platform using the `Systems/makefile`.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Related information

- *3.2 Make options for the reference platform makefile and Systems/makefile* on page 3-23
- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.5 Creating custom systems that include Arm® models

This use case describes adding an Arm model to a custom system.

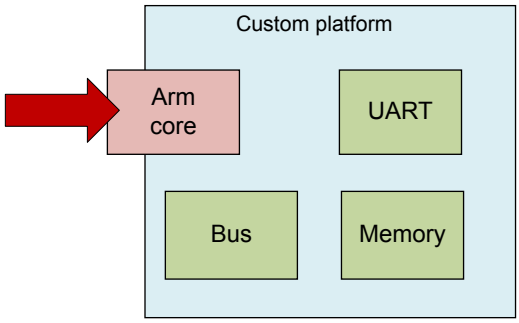

Figure 4-4 Adding an Arm core to a custom system

You can build your own, custom SystemC platform that includes a Cycle Model downloaded from Arm IP Exchange. Use the Cycle Models Configuration Tool (`cm_config`), included in the SystemC Cycle Models Runtime, to extract its build options. See the *SystemC Cycle Model User Guide* for your IP for more information about the Cycle Models Configuration tool.

Contact Arm Support for more information.

Related information

- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.6 Loading applications for simulation

This section describes what you need to know about loading applications for Cycle Model reference platform simulations.

Applications included with your reference platform are found in the reference platform `Applications` directory. Pin-level reference platforms and TLM-based reference platforms handle application loading differently.

Pin-level reference platform application loading

reference platforms implemented through pin-level connections use the memory-loading capabilities of the memory component included in the reference platform. The application to be loaded is determined by the contents of the `.hex` files in the reference platform `Systems` directory. These hex files are created using the `create_dat_file.sh` script located in the `Scripts` directory. The application is loaded into the reference platform upon initialization of the memory.

Alternate applications must be in `.elf` file format. To change the application used by the reference platform:

1. Run the `create_dat_file.sh` script on the new `.elf` file to create the required hex files.
2. Run `system_test`.

See the `README.txt` file for more information about the application run by default.

Note

Do not use the `-a` or `--application` command line arguments to specify applications for reference platforms with pin-level models. These arguments have no effect on pin-level reference platforms, and result in multiple warnings.

TLM-based reference platform application loading

The information in this section applies to TLM-based reference platforms that support debugging. These reference platforms use the standard SystemC framework to determine which application to load. This means that you must specify which application to load.

Specify the application on the command line using the `-a` or `--application` argument. For example:

```
$ ./system_test -a test.elf -S
```


Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

4.7 Modifying the Cycle Model reference platform test bench

Each reference platform has its own test bench called `system_test.cpp`, which is located in the `Systems` directory.

`system_test.cpp`:

- Instantiates the models
- Defines the connections between models
- Initializes model parameters
- Provides simulation controls (start, stop, run, specific number of cycles, etc.)

To make changes to the reference platform system, alter the reference platform test bench. After altering the test bench, recompile the system. Models are recompiled automatically as part of the system-level recompile.

This section contains the following subsections:

- *4.7.1 Modifying the test bench for pin-level models* on page 4-33.
- *4.7.2 Modifying the test bench for TLM models* on page 4-35.

4.7.1 Modifying the test bench for pin-level models

This section describes the areas of the Cycle Model reference platform test bench you might want to change.

Note

See *4.6 Loading applications for simulation* on page 4-32 for information about how pin-level reference platforms handle application loading.

Required includes

The test bench contains the SystemC wrapper files for any models in the system, the corresponding reset module file for each model (for pin-level models only), and `includes` required by the Fast Models runtime. Ensure you add these files for any new models added to your system. Here is an example of the `includes` section of a reference platform test bench:

```
// Include the systemc wrapper files for the models
#include "libCortexR8.systemc.h"            // CortexR8 CPU
#include "libNIC400.systemc.h"              // NIC400 Interconnect
#include "libBP140.systemc.h"               // BP140 Memory
#include "libBP140_TrickBox.systemc.h"      // BP140_TrickBox UART

// Include the reset modules for the above models (pin-level models only)
#include "CortexR8ResetModule.h"
#include "NIC400ResetModule.h"
#include "BP140ResetModule.h"
#include "BP140_TrickBoxResetModule.h"
#include "../perf_common.h"
#include <iostream>
#include <time.h>

// These includes are need by the SCX FastModel Runtime
#include <scx/scx.h>
#include <scx/scx_signal_sizer.h>
#include <scx_simcontroller.cpp>
#include <scx_scheduler_mapping.cpp>
#include <scx_report_handler.cpp>
```


Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Initialization of the simulation environment and model instantiation

The `sc_main()` section of the test bench must first initialize the SCX simulation environment. It must state clocking specifications, then instantiate all IP and reset models (for pin-level models only) for any models included in the system. For example:

```
int sc_main(int argc, char *argv[])
{
  // Debug initialization
  scx::scx_initialize("R8-SysC-Debug");

  // If you want to see messages about 'port not bound' change SC_DO_NOTHING to SC_DISPLAY.
  // If you want it to abort on a 'port not bound' error comment out the line below.
  sc_report_handler::set_actions(SC_ID_COMPLETE_BINDING_,SC_ERROR,SC_DO_NOTHING);

  // Clock Object
  sc_clock clk("clk", 1, SC_NS, 0.5);

// Instantiate IP
  CortexR8 core("cortexR8");
  NIC400 nic400("NIC400");
  BP140 bp140("BP140");
  BP140_TrickBox bp140_trickbox("BP140_TrickBox");
  ARM::Models::Cycle::ModelCortexR8::CortexR8ResetModule core_reset("core_reset");
  ARM::Models::Cycle::ModelNIC400::NIC400ResetModule nic400_reset("nic400_reset");
  ARM::Models::Cycle::ModelBP140::BP140ResetModule bp140_reset("bp140_reset");
  ARM::Models::Cycle::ModelBP140_TrickBox::BP140_TrickBoxResetModule
bp140_trickbox_reset("bp140_trickbox_reset");
```

Signal bindings

The test bench specifies all signal bindings, including those for reset modules.

- Declare bindings using an `sc_signal` call in the test bench file. The signal must be the same type and width as the two ports being connected. If the ports are the same type but different widths, use `scx_signal_sizer` instead of `sc_signal`. For example:

  ```
  scx::scx_signal_sizer<sc_uint<13>, sc_uint<16> >ARIDsignal1;
  core.ARIDM0.bind(ARIDsignal);
  nic400.arid_s_axi_64.bind(ARIDsignal);
  ```

- Signals must be bound to both ports. For example:

  ```
  sc_signal(bool) signal1;
  Inst1.port1.bind(signal1);
  sc_signal(bool) signal2;
  Inst2.port1.bind(signal2);
  ```

Clock bindings

All models must be bound to the system clock; for example:

```
// Bind all the models to the system (cpu) clock
core.CLKIN.bind(cpu_clk);
mem.ACLK.bind(cpu_clk);
bus.mainclk.bind(cpu_clk);
uart.ACLK.bind(cpu_clk);
core_reset.clk.bind(cpu_clk);
bus_reset.clk.bind(cpu_clk);
bp140_reset.clk.bind(cpu_clk);
bp140_trickbox_reset.clk.bind(cpu_clk);
```

Functions to generate performance data

The following example shows the use of SCX API functions to specify PMU and Tarmac output. See the *SystemC Cycle Model User Guide* for the CPU in your reference platform for more information:

```
scx::scx_set_parameter("cortexR8_core.PMU_ENABLED",true);
scx::scx_set_parameter("cortexR8_core.TARMAC_ENABLED",true);
```


Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Parsing of command line arguments

The test bench calls the SCX `scx_parse_and_configure()` function to parse any command line arguments used by the SCX runtime:

```
scx::scx_parse_and_configure(argc, argv);
```

Simulation call

To simulate the system, the test bench calls `sc_start()`:

```
sc_start();
```

Specify all includes, initializations, bindings, functions, and command line options before `sc_start()`.

Related information

- *SystemC Cycle Model User Guide* for the CPU in your reference platform. You can find Cycle Model guides at *https://developer.arm.com/tools-and-software/simulation-models/cycle-models/docs*.

4.7.2 Modifying the test bench for TLM models

The `sc_main()` function in the test bench has the same basic flow for both pin-level and TLM models. One difference is that the TLM models are connected using TLM sockets rather than pins. This section describes the TLM-specific instructions.

Note

See the file `libcomponent.tlm.h` in your installation directory for socket names.

Note

See *4.6 Loading applications for simulation* on page 4-32 for information about how TLM-based Cycle Model reference platforms handle application loading.

Required includes

The SystemC pin-level models are wrapped with TLM functionality. The test bench includes the SystemC wrapper files for any models included in the system, and `includes` required by the Fast Models runtime. Ensure you add these files for any new models added to your system. Here is an example of the `includes` section of a reference platform test bench:

```
// Include the systemc wrapper files for the models

#include "models/SimpleMem.h"
#include "models/SimpleFlash.h"
#include "models/SimpleFlashImp.h"
#include "models/RAMBlock.h"
#include "models/SimpleBus.h"
#include "models/BasicUART.h"
#include "models/ElfLoader.h"
#include "libCORTEXR8.tlm.h"                 // CPU
#include <tlm_utils/simple_initiator_socket.h>

#include <iostream>

// These includes are need by the SCX FastModel Runtime
#include <scx/scx.h>
#include <scx_simcontroller.cpp>
#include <scx_scheduler_mapping.cpp>
#include <scx_report_handler.cpp>
```


Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Initialization of the simulation environment and model instantiation

The `sc_main()` section of the test bench must first initialize the SCX simulation environment. It states clocking specifications, then instantiates any models included in the system. Ensure you instantiate any new models added to your system. For example:

```
int sc_main(int argc, char *argv[])
{
// Debug initialization
  scx::scx_initialize("R8-SysC");

  // If you want to see messages about 'port not bound' change SC_DO_NOTHING to SC_DISPLAY.
  // If you want it to abort on a 'port not bound' error comment out the line below.
  sc_report_handler::set_actions(SC_ID_COMPLETE_BINDING_,SC_ERROR,SC_DO_NOTHING);

  // Clock Object
  sc_clock cpu_clk("clk", 1, SC_NS, 0.5);

  ARM::Models::Cycle::ModelCortexR8::CortexR8Imp core("CortexR8");

  // Main Memory
  ARM::Models::RAMBlock ram_block;
  ARM::Models::SimpleMemConfig simple_mem_params;
  simple_mem_params.delay = 1;
  simple_mem_params.ram_block = &ram_block;
  simple_mem_params.busWidthBits = 64;

  ARM::Models::SimpleMem simple_mem("RAM", simple_mem_params);
  ARM::Models::BasicUART uart("UART", std::cout, "UART: ");

  // BUS & its Mappings
  ARM::Models::SimpleBus bus("Interconnect", 1, 1, 64);
  bus.addMap(0, 0, 0xFFFFFFFF); // Main Memory
```

Port bindings

The test bench specifies all TLM port bindings. Signal bindings required for the system are also specified in this area. Specify any additional TLM port bindings and signal bindings in this area:

```
// Core Main Memory Port Bindings
        core.iSkt_AXI3_Master_PORT0->bind(bus.tSkt);
        core.directIskt_AXI3_Master_PORT0.bind(simple_mem.directTskt);

// Core Low Latency Peripheral Port Bindings
        core.iSkt_AXI3_Master_PERI->bind(uart.tSkt);
        core.directIskt_AXI3_Master_PERI.bind(uart.directTskt);

// Bus iSkt[0] connected to Main Memory
        bus.iSkt[0]->bind(simple_mem.tSkt);

// Clock
        core.clk(cpu_clk);
        simple_mem.clock.bind(cpu_clk);
        uart.clock.bind(cpu_clk);
        bus.clock.bind(cpu_clk);
```

For information about signal bindings, clock bindings, performance data generation, command line arguments, and simulation calls, use the instructions in *4.7.1 Modifying the test bench for pin-level models* on page 4-33.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

Chapter 5
Troubleshooting

This chapter provides solutions for problems that may occur when working with Cycle Model reference platforms.

It contains the following sections:

- *5.1 carbon_sc_multiwrite_signal.h build error* on page 5-38.
- *5.2 Unrecognized command line option* on page 5-39.
- *5.3 Licensing errors* on page 5-40.
- *5.4 Bytes requested error when specifying application* on page 5-41.
- *5.5 Multiple target patterns error* on page 5-42.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

5.1 carbon_sc_multiwrite_signal.h build error

Incorrectly set `CARBON_HOME` environment variable results in fatal error.

Cause

When using a version of the Cycle Model Studio runtime other than the one included in the Cycle Model reference platform, the environment variable `CARBON_HOME` defines the runtime location. When unset (this is the default for the reference platform), the Cycle Model Studio runtime included in the reference platform is used.

If `CARBON_HOME` is set, it must be set to a Cycle Model Studio runtime installation, or to a full version of Cycle Model Studio, that is Version 11 or later. Otherwise, an error similar to the following may occur when building the reference platform:

```
../MODELS/CortexR52_2CPU/gcc483/SystemC/libCortexR52.systemc.h:19:48: fatal error: carbon/
carbon_sc_multiwrite_signal.h: No such file or directory
#include "carbon/carbon_sc_multiwrite_signal.h"

compilation terminated.
make: *** [system_test.o] Error 1
```

Solution

Either:

- `source` the setup script of a Version 11.0 (or later) Cycle Model Studio runtime or Cycle Model Studio.
- Unset `CARBON_HOME` to use the version of the Cycle Model Studio runtime included in the reference platform.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

5.2 Unrecognized command line option

Running an unsupported GCC version results in the build error `unrecognized command line option "-std=c++11"`.

Cause

Using an unsupported GCC version may result in a build error similar to the following:

```
cc1plus: error: unrecognized command line option "-std=c++11"
make: *** [system_test.o] Error 1
```

Solution

Check your GCC version against the supported versions listed in *1.2 System requirements and prerequisites* on page 1-11. In the following example, GCC 4.4.7 is unsupported and GCC 4.8.3 (a supported version) is sourced:

```
$ gcc --version
gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-11)
Copyright (C) 2010 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

$ source /o/Linux64/etc/setup.sh
$ gcc --version
gcc (GCC) 4.8.3
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```


Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

5.3 Licensing errors

A valid Arm Cycle Model runtime license is required to run reference platform simulations.

Cause

If a valid license is not available, an error similar to the following may occur when running the test bench simulation:

```
Checkout of CM_ARM_Runtime : No feature match. - checkout failed (nowait): Cannot find
license file.
The license files (or license server system network addresses) attempted are
listed below.  Use LM_LICENSE_FILE to use a different license file,
or contact your software provider for a license file.
Feature:       CM_ARM_Runtime
Filename:      /license
License path:  /license:
```

Solution

- Ensure that the environment variable `ARMLMD_LICENSE_FILE` is set to your license server; for example, `export ARMLMD_LICENSE_FILE=`*`port@host`*.
- Contact Arm Techical Support (support-esl@arm.com).

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

5.4 Bytes requested error when specifying application

`Number of bytes requested` error message occurs when starting a simulation and specifying an application.

Cause

When launching a simulation with a specified application, use the `-a` or `--application` flag only with TLM-based Cycle Model reference platforms. Using these flags with pin-level reference platforms results in an error similar to the following:

```
$ Number of bytes requested for write (1) does not match numbers of bytes reportedly
written(0)
```

Solution

Specify the application according to the instructions in *4.6 Loading applications for simulation* on page 4-32.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL

5.5 Multiple target patterns error

When adding a new model to a system, `make` fails with the error `Makefile:108: *** multiple target patterns. Stop.`

Cause

The build and link options used in the makefile's target are incorrect due to an unhandled `cm_config` error, or due to improper syntax in build and link options you added.

Solution

When you add a new model to an existing Cycle Model reference platform, ensure that the build and link options for all reference platform components are available to the Makefile. If the required build and link options are not found when you run `make`, this error is output.

Models built on Arm IP Exchange (*https://ipx.arm.com*) have an associated XML data file, which is readable by the Cycle Model Configuration Tool (`cm_config`) that ships with the Cycle Model Runtime. The XML data file provides the required build and link options to the `Makefile` using `cm_config`.

However, custom models, and models created by Cycle Model Studio, do not have the associated XML data file. In these cases, you must provide the build options to the `Makefile`. This error may occur if these options have incorrect `make` syntax. See *4.1 Making build and link options available to the Makefile* on page 4-25.

Copyright © 2019–2021 Arm Limited or its affiliates. All rights reserved.
Non-Confidential - REL