Fast Models System Creator

Version 9.1.0

User Guide

Non-Confidential

ARM DUI 0967E (ID012317)

Fast Models System Creator
User Guide

Copyright © 2017 ARM Limited. All rights reserved.

Release Information

The following changes have been made to this document.

Change History

Date	Issue	Confidentiality	Change
February 2016	A	Non-Confidential	8.3 release
May 2016	B	Non-Confidential	8.4 release
August 2016	C	Non-Confidential	8.5 release/FM 10.1
November 2016	D	Non-Confidential	9.0.0 release
February 2017	E	Non-Confidential	9.1.0 release

Confidentiality Status

Product Status

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

Web Address

http://www.arm.com

Contents

Preface

Audience

This user manual is intended for development engineers who create components for use with ARM SoC Designer.

Conventions

This book uses the following conventions:

Convention	Description	Example
`courier`	Commands, functions, variables, routines, and code examples that are set apart from ordinary text.	`sparseMem_t SparseMemCreate-New();`
italic	New or unusual words or phrases appearing for the first time.	*Transactors* provide the entry and exit points for data ...
bold	Action that the user performs.	Click **Close** to close the dialog.
<text>	Values that you fill in, or that the system automatically supplies.	<platform>/ represents the name of various platforms.
[text]	Square brackets [ ] indicate optional text.	`$CARBON_HOME/bin/modelstudio [ <filename> ]`
[text1 | text2]	The vertical bar | indicates "OR," meaning that you can supply text1 or text 2.	`$CARBON_HOME/bin/modelstudio [<name>.symtab.db | <name>.ccfg ]`

Chapter 1

Introduction

This chapter includes the following sections:

- Overview of Fast Models Creation
- Overview of Swap & Play
- Overview of Save/Restore
- Known Issues and Limitations

1.1 Overview of Fast Models Creation

The Fast Models System Creator is an add-on to SoC Designer that gives you the ability to selectively replace Cycle Accurate components in a system with ARM® Fast Models equivalents. This allows you to trade cycle accuracy for increased simulation performance in the selected components.

The components that you select are automatically integrated into a Fast Models system. Additionally, a new SoC Designer component is created to interface the Fast Models system to SoC Designer, and a new SoC Designer system is created in which the selected components are replaced by the new Fast Models component.

1.2 Overview of Swap & Play

Swap & Play allows you take advantage of the simulation speed of Fast Models to quickly run to an interesting simulation point, and to continue with cycle accuracy when needed.

The Swap & Play feature relies on a checkpoint save of the memory and registers of a Fast Model, then uses the checkpoint save data to convert the system into a SoC Designer model. The architectural state of the Fast Models components is saved to a checkpoint file, which can then be restored into the original Cycle Accurate system from which the Fast Models system was created. Components that are not included in the Fast Models can still participate in Swap & Play, provided they support traditional Save/Restore.

Swap & Play configurations are preset for Fast Models components acquired from either IP Exchange or from the SoC Designer Protocol Bundle. They should not need to be changed. However, you may need to establish configurations for components created by other means, such as:

- User-created CASI (the Cycle Accurate Simulation Interface) SoC Designer components.
- Proprietary Fast Models created with LISA (the ARM Language for Instruction Set Architectures).
- SoC Designer components created with Cycle Model Studio.

Prior to creating the configuration for such models, you should have a thorough knowledge of the registers available in both the LISA and Cycle Accurate models.

Refer to the section Using Swap & Play in this document for instructions on configuring Swap & Play. Refer also to the *Swap & Play for Custom Components Developer's Guide* (ARM DUI0960) for information about creating custom components that can participate in Swap & Play.

1.2.1 Understanding Checkpoints and Workspaces

A checkpoint is a file containing the state of all the components in a system at a point in a simulation. In SoC Designer, checkpoint files have the extension .mxc. This differs from the .mxr simulation state files, which contain the debug view (register and memory windows, breakpoints, etc.) in addition to component state.

Note: For a complete discussion of the difference between .mxs, .mxr, and .mxc files, refer to the SoC Designer User Guide (ARM DUI 0956).

When you create a checkpoint, SoC Designer automatically associates it with the current system (i.e., .mxp file). For Fast Models systems, the .mxp file associated with the checkpoint is that of the original Cycle Accurate system from which the Fast Models system was created.

Additionally, for a particular SoC Designer system, you can create multiple named workspaces to further organize checkpoints. Finally, you can enter a text description for each checkpoint, which is recorded along with other data such as a date stamp and simulation time.

1.3 Overview of Save/Restore

Models in the Cycle Accurate domain rely on the Save/Restore feature of SoC Designer components. There are two types of Save/Restore:

- **Architectural Save/Restore** — Saves memory and register state (state which is visible to CADI). This is required for a component to participate in Swap & Play. It is also possible to instrument Save/Restore of cache contents; refer to the *Developer's Guide to Swap & Play for Custom Components* (ARM DUI0960).
- **Binary Save/Restore** — Saves all internal state, including local variables of a CASI model or state from the original RTL. This is required for a component to participate in Save/Restore.

1.4 Known Issues and Limitations

The following are known issues and limitations with Fast Models System Creator functionality:

- The program memory must be included in the Fast Model. Some debug accesses made by the Fast Models CPU model will fail otherwise.
- A limited number of ports and parameters are supported at this time.
- Multiple clock domains are partially supported. If the system is supported, the CDIV components that are needed for Fast Models to cycle-accurate communication will be retained and connected to both the Fast Models component and the cycle-accurate components.
- Occasionally, semihosting input and output are not redirected to SoC Designer. This means that if your application uses semihosting, you may not be able to interact with it. This is an intermittent problem that will be fixed in a future release.
- Debug access to a Fast Models component can only be done when the model is at a debuggable point. Debug access at other times has unpredictable results.
- User signals are not used for AHB transactions within the Fast Models or crossing the boundary of the Fast Model.
- The simulation reset functionality of SoC Designer does not work with Fast Models.

CASI model may use architectural Save/Restore, but there may be some obstacles to its use:

- For large memories, the default CADI Architectural save may be inefficient (if the number of pages accessed is very large).
- The Architectural save does not capture internal state, which could be crucial if the component is capable of being interrupted in the middle of a transaction. There could be other reasons why internal state is necessary; it depends how the CASI model is coded.

Chapter 2

Installation

2.1 Requirements

Fast Models System Creator requires the following software:

- Red Hat Enterprise Linux 6.6 or Microsoft Windows 7
- SoC Designer
- ARM System Generator Fast Models Tools 10.3
- ARM System Generator Fast Models Portfolio 10.3
- For Linux environment - gcc version 4.7.2 or 4.8.3
- For Windows environment - Visual Studio 2013 Update 4
- For debugging software in the Fast Models, use the ARM DS-5 Debugger. Refer to the *SoC Designer ARM DS-5 Debugger Integration Application Note* (ARM DUI0999) for version information and instructions on integrating the DS-5 Debugger with SoC Designer.

2.2 Installation

The Fast Models System Creator is installed automatically with SoC Designer.

2.3 Supported Components

For a complete list of supported components, please refer to the *Fast Models System Creator Cycle Models Reference* (ARM DUI0985).

Chapter 3

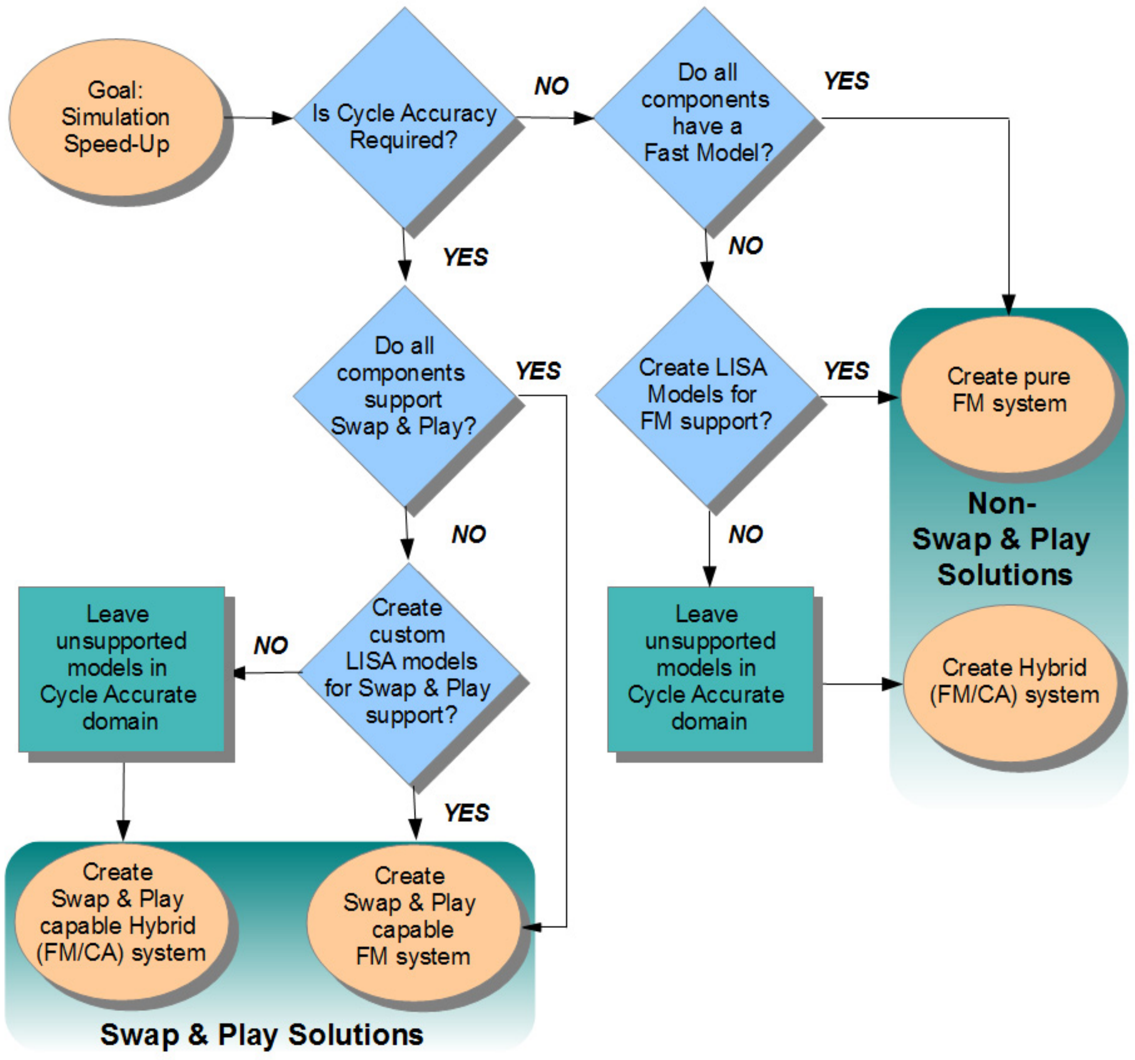
Best Practices for Planning Your Fast Models Implementation

This chapter helps you determine the best approach to implementing Fast Models and Swap & Play™, and presents guidelines and regulations that you should be aware of when initially constructing your Cycle Accurate system. The design choices you make when building your Cycle Accurate system affect the performance of your Fast Models and Swap & Play implementations.

This chapter discusses the following topics:

3.1 Overview of Choosing a Simulation Speed-Up Path

Figure 3-1 illustrates, at a high level, the decision-making process described in this chapter. The questions presented in the flowchart help you to determine the best solution for your implementation.



Goal: Simulation Speed-Up
Is Cycle Accuracy Required?
NO
Do all components have a Fast Model?
YES
YES
NO
Do all components support Swap & Play?
YES
Create LISA Models for FM support?
YES
Create pure FM system
Non-Swap & Play Solutions
NO
NO
Leave unsupported models in Cycle Accurate domain
NO
Create custom LISA models for Swap & Play support?
Leave unsupported models in Cycle Accurate domain
Create Hybrid (FM/CA) system
YES
Create Swap & Play capable Hybrid (FM/CA) system
Create Swap & Play capable FM system
Swap & Play Solutions

Figure 3-1 Simulation Speed-Up: Choosing a Solution

3.2 Simulation Speed-Up Use Cases: What's Your End Goal?

As shown in Figure 3-1 in the previous section, there are two general approaches to speeding up your simulation, depending on the kind of feedback you require. If:

- You require acceleration of the simulation to a certain critical point, then require feedback from the Cycle Accurate domain, your implementation should incorporate both Fast Models and Swap & Play. Refer to the section What to think about when constructing Swap & Play capable systems, below.
- You require simulation speedup, and have no dependency on interactive feedback from the Cycle Accurate domain, then a pure Fast Models (no Swap & Play) implementation will meet your requirements. Refer to the section What to think about when constructing Simulation Speedup Systems, below.

3.2.1 What to think about when constructing Swap & Play capable systems

The critical question when designing a system intended to implement Swap & Play is: *Can the relevant Cycle Accurate components participate in Swap & Play?* If not, determine whether an alternative component exists that is Swap & Play capable.

If a component does not support Swap & Play, you have the following options:

- Pure Fast Models system — Create a custom model, to which you map the component that is not Swap & Play compliant. Refer to the section “Category 3: Developing Custom LISA code” on page 22.
- *Hybrid* FM/CA system — Leave the component in the Cycle Accurate domain. For example, if you want to boot Linux to a certain critical point, then swap to the Cycle Accurate world to run a benchmark test, then the Hybrid solution works well.

For a list of models that support Swap & Play, refer to the *SoC Designer Fast Models System Model Reference* (ARM DUI0985).

3.2.2 What to think about when constructing Simulation Speedup Systems

In cases where you want to accelerate the performance of a system to a certain execution point, and the detail provided by cycle accuracy is not critical, implement a non-Swap & Play solution. This non-Swap & Play capable solution may be either a:

- Pure Fast Models implementation.
- Hybrid FM/CA implementation. That is, create a Fast Models that represents the system at the desired state, and incorporate interactions with Cycle Accurate models as necessary.

For example, the Hybrid approach would work well to speed up a test case in which you accelerate a regression test that takes an entire day in the Cycle Accurate domain.

In Hybrid scenarios, you must determine:

- Which components to include in the Fast Models system, and which to maintain in the Cycle Accurate domain.
- Whether the Fast Models matches the implementation of the corresponding component.

If a Fast Models counterpart exists for each component to be included in the Fast Models domain, you can begin creating a *partitioning* scheme for your system. Refer to the section "Best Practices and Requirements for Partitioning Hybrid Systems" on page 23.

If a Fast Models counterpart does not exist for each component, assess options for those components without one. For optimum speedup, it is generally best to map components to Fast Models whenever possible.

For a list of components that support Fast Models, refer to the *Fast Models System Model Reference* (ARM DUI0985).

3.3 Approaches to Mapping Models

This section discusses approaches to mapping Cycle Accurate models to their Fast Models counterparts. All Fast Models implementations fit into one of these categories.

- Category 1: Default premapped Fast Models
- Category 2: Custom mapping of Cycle Accurate component to an existing Fast Model
- Category 3: Developing Custom LISA code

If you are not sure which approach matches your system implementation, review the information in the previous section, Simulation Speed-Up Use Cases: What's Your End Goal? To summarize, you need to know:

- Whether a Fast Models exists for each component you want to convert.
- Whether the corresponding Fast Models matches the Cycle Accurate model implementation, or whether some customization is necessary to model its behavior.

For a list of models and their support for Fast Models, Swap & Play, and Save/Restore, refer to the *Fast Models System Model Reference* (ARM DUI0985).

3.3.1 Category 1: Default premapped Fast Models

ARM provides Cycle Accurate models that come pre-mapped to the appropriate ARM Fast Model. If your system consists exclusively of pre-mapped models, then all components will be mapped to the Fast Models domain. You are ready to run Fast Models System Creator. Refer to Chapter 4.

3.3.2 Category 2: Custom mapping of Cycle Accurate component to an existing Fast Model

When you have a fully-instrumented – but not premapped – Cycle Accurate model and a matching LISA file, you can map the model to the LISA file when you start the Fast Models creation process (see Chapter 4). This approach supports Swap & Play, provided your LISA implementation is coded accordingly.

Virtual Register Support

Using the Fast Models Virtual Register functionality, the Custom Mapping option also covers situations in which registers are modeled in the Cycle Accurate model, but not in the Fast Model.

Be aware of the following restriction with this approach:

- Virtual Registers provide only storage, not functionality, supporting cases such as fabrics with registers (for example, the NIC-301). The Fast Models Virtual Register functionality does not support cases in which registers in the Cycle Accurate domain are required to run cycles to process register values.

To implement the Virtual Registers solution, use the Fast Models System Creator to create customized mappings of registers in a Cycle Accurate model.

3.3.3 Category 3: Developing Custom LISA code

If you do not have a Fast Models to which to map a particular Cycle Accurate model, you have the option to create one using LISA (the ARM Language for Instruction Set Architectures). Writing your own LISA Fast Models requires hand-coding all of the ports, registers, and memories of the Cycle Accurate component.

Refer to the ARM LISA documentation for general LISA information.

3.4 Best Practices for Improving Simulation Performance

This section discusses:

- When your Fast Models Domain Dominates Performance
- Fast Models Simulation Control Functions

3.4.1 When your Fast Models Domain Dominates Performance

It's important to establish the correct balance between tasks handled in the Fast Models domain and those left in the Cycle Accurate domain. Ensure that the entire path to system memory - consisting of, at minimum, the CPU, memory, and interconnect components - are grouped together in the Fast Models domain. Also review the additional partitioning recommendations in the section "Best Practices and Requirements for Partitioning Hybrid Systems" on page 23.

3.4.2 Fast Models Simulation Control Functions

The most effective means of improving a hybrid system's simulation performance is fine-tuning its boundary partitions and incorporating as many components as possible into the Fast Models domain.

If you have made modifications of this kind to the system, and performance continues to be an issue, then adjusting the simulation controls may help. For example, the TLM-2.0 global quantum determines the Wait interval after the Fast Models system has been simulated. This wait interval is implemented using the TLM 2.0 `tlm_global_quantum` singleton. Refer to the SystemC TLM 2.0 documentation for more information.

3.5 Best Practices and Requirements for Partitioning Hybrid Systems

Partitioning refers to the manner in which you organize the components of your system: those intended for the Fast Models domain, and those that will remain in the Cycle Accurate domain.

In cases where no Fast Models exists for a component of your system, you must create systems that use a combination of Fast Models and Cycle Accurate components. This section contains important considerations related to partitioning these Hybrid systems.

3.5.1 Best Practices for System Partitioning

For best performance, partition your system so that highly-active blocks (for example, CPU, Memory, and Fabric) are together in a single Fast Model. In other words, components that need to communicate with one another frequently should not be separated by the Cycle Accurate/Fast Models boundary.

Refer to the following sections, Supported boundaries per AMBA AXI and ACE protocol variant and Partitioning restrictions for interdependent transactions, to determine if these important requirements and restrictions apply to your system.

3.5.2 Supported boundaries per AMBA AXI and ACE protocol variant

Table 3-1 illustrates the supported Fast Models and Cycle Accurate model combinations per protocol variant boundary:

Table 3-1: Supported FM and CA Protocol Boundary Combinations

Support Status	Protocol Variant	Master Side	Slave Side
Supported	ACE-Lite	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	AXI4	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	AXIv2 (ARM AXI3)	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	AHB	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	APB	Fast Model	Cycle Accurate

Table 3-1: Supported FM and CA Protocol Boundary Combinations

Support Status	Protocol Variant	Master Side	Slave Side
Not Supported	ACE	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	ACE-Lite+DVM	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	AXI4-Lite	Fast Model	Cycle Accurate
		Cycle Accurate	Fast Model
	APB	Cycle Accurate	Fast Model

For example, the configuration shown in Figure 3-2 is supported, because it uses ACE between the two Fast Models, and ACE-Lite at the Fast Model-to-Cycle Accurate model boundary:

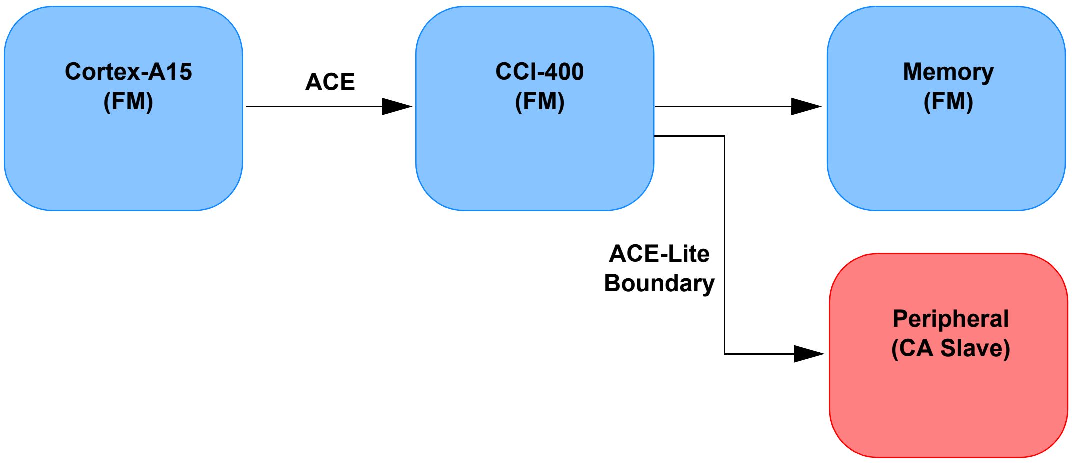


Figure 3-2 Supported Configuration

The example shown in Figure 3-3 is *not* supported, because it uses ACE at the Fast Model-to-Cycle Accurate model boundary:

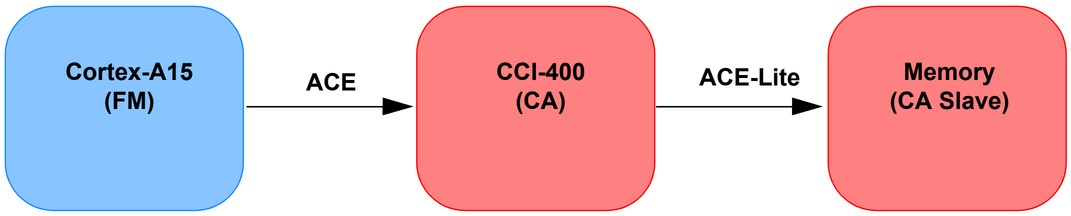


Figure 3-3 Unsupported Configuration

3.5.3 Partitioning restrictions for interdependent transactions

Hybrid systems - those consisting of components in both the Cycle Accurate and Fast Models domains - do not permit multiple domain crossings of interdependent transactions.

For example, the system shown in Figure 3-4 is invalid, because:

1. A transaction from the Cortex-A9 crosses from the Fast Models domain to the Cycle Accurate PL301.
2. The PL301 then issues a dependent transaction (i.e. a transaction that must complete in order for the first transaction to complete) to the system memory, which requires a second domain crossing.

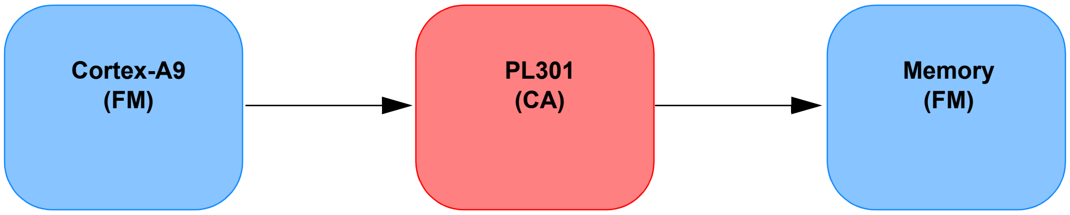


Figure 3-4 Invalid System

The system shown in Figure 3-5 shows a valid path (in green) and an invalid path (in red). If the bus master can access the peripheral directly, there are two CA/FM domain crossings.

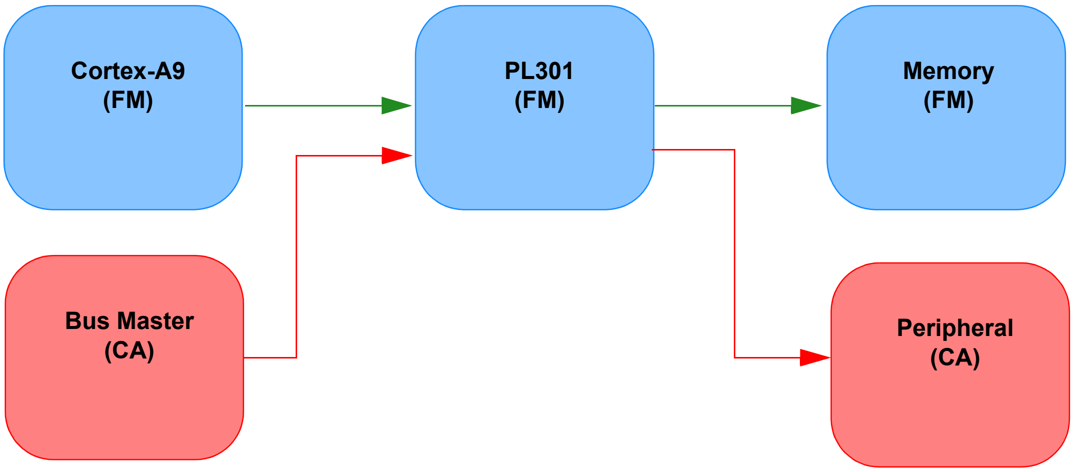


Figure 3-5 Invalid System

The system shown in Figure 3-6 is valid. Although the PL330 (DMA controller) can be the target of transactions from the Fast Models CPU as well as the initiator of transactions to the Fast Models memory, the transactions are not interdependent. The register reads and writes from the CPU are not blocked by the completion of the reads and writes to memory.

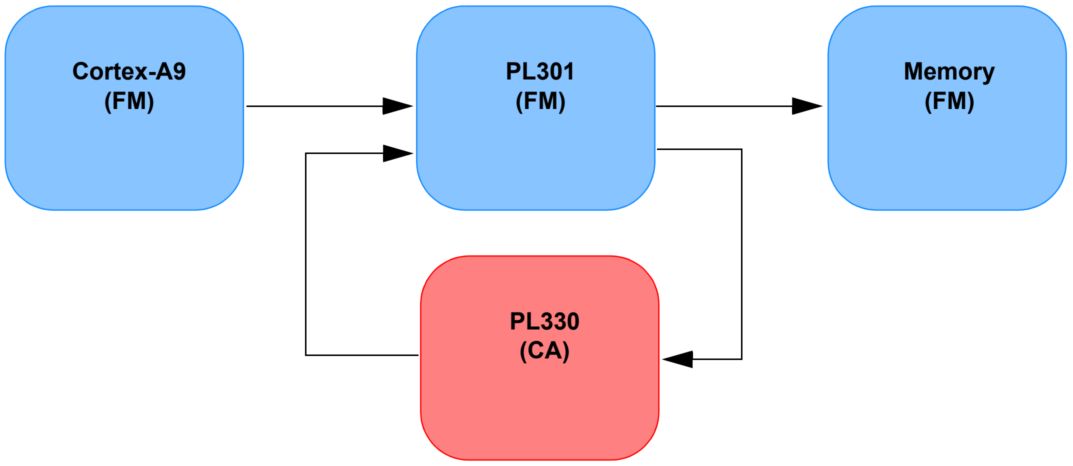


Figure 3-6 Valid system

3.6 Best Practices for Synchronization Concerns

This section discusses handling issues related to timed versus untimed domains. It includes:

- Synchronization Overview
- Coordinating Program Execution

3.6.1 Synchronization Overview

A key to achieving higher simulation throughput in the new system is the partitioning of the Fast Models and Cycle Accurate models. Synchronization concerns directly impact your partitioning scheme.

Essentially, the time domains of the two zones are not correlated. The Fast Models runs freely until it has either:

- Simulated a predetermined amount of time, or
- Requires communication with the Cycle Accurate models.

At that point, the Cycle Accurate models advance one cycle. This ensures that the two zones are synchronized from an event-ordering standpoint.

Note that the zones do not synchronize from a time standpoint. For example, a transaction from the Fast Models to one of the Cycle Accurate peripherals may occur at cycle 1,000,000 in the Fast Models domain, but at cycle 1,000 in the Cycle Accurate domain.

The ordering of events, however, is accurate. A software breakpoint in the Fast Models that occurs before the instruction that issues the boundary-crossing transaction occurs before that transaction is issued to the Cycle Accurate domain. Likewise, a software breakpoint that occurs after the instruction that issues the transaction occurs after that transaction has completed in the Cycle Accurate domain. When the Fast Models stops because of a breakpoint, SoC Designer stops as well.

The time correlation between the two domains is deterministic. For a given sequence of user input (simulation, breakpoints, debug access, etc.), the times at which simulation events occur is the same from one run to the next.

3.6.2 Coordinating Program Execution

All components in a single, self-contained Cycle Accurate or Fast Models domain are implicitly synchronized by the same clock cycle. However, Hybrid systems can not make use of this method of synchronization, and must be explicitly synchronized by other measures. Your system design (including code) must address possible synchronization issues that could occur due to programs operating in different domains.

For example, consider the configuration shown in Figure 3-7:

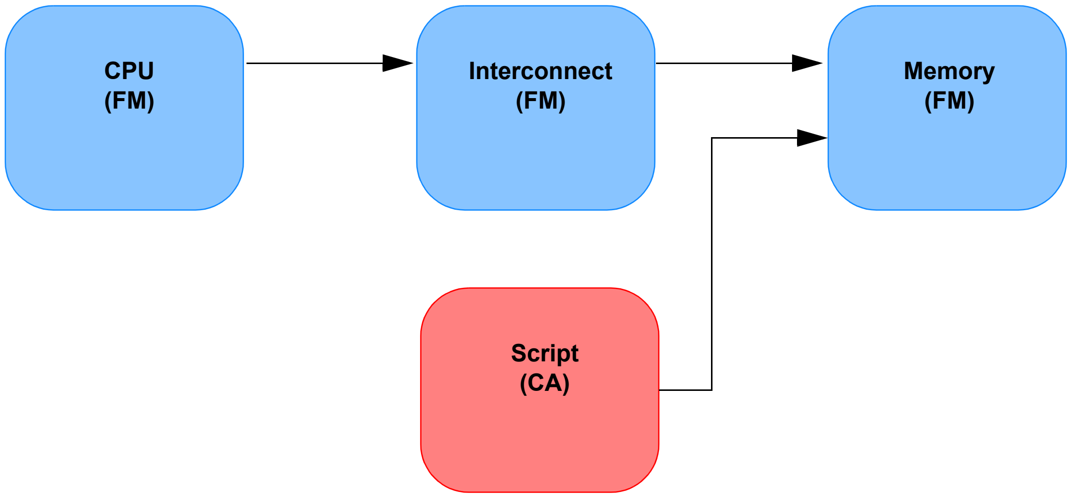


Figure 3-7 Synchronization Example

In this example, the script exists outside of the Fast Models domain, in which the CPU, interconnect, and memory are all implicitly synchronized by the same clock cycle. The script loads a set of addresses into memory, which a program running on the CPU then uses.

Without explicit coordination, this configuration allows the possibility of the program running on the CPU becoming out of sync with the script processes; it could attempt to access the addresses in memory before the script has finished loading them. Two possible approaches to preventing an out-of-sync condition are:

- Hold the CPU in Reset state until the script has loaded all of its addresses into memory.
- Store a value (for example, 0) in memory at a certain address. Configure the program running on the CPU to execute after reading a 1 at that address. As its final task, after the complete set of values has been written to memory, have the script write a 1 to that address.

3.7 Requirements for Multiple Clock Domains

In a Hybrid (Fast Models and Cycle Accurate) system, multiple clock domains are a concern when a Fast Models component communicates with a Cycle Accurate component using a clock generated by a CDIV component.

In the example below (Figure 3-8), a processor, interconnect, and memory are in the Fast Models domain, but the peripherals are in the Cycle Accurate domain. The Fast Models components use the main clock and correctly use the CDIV clock for the peripherals.

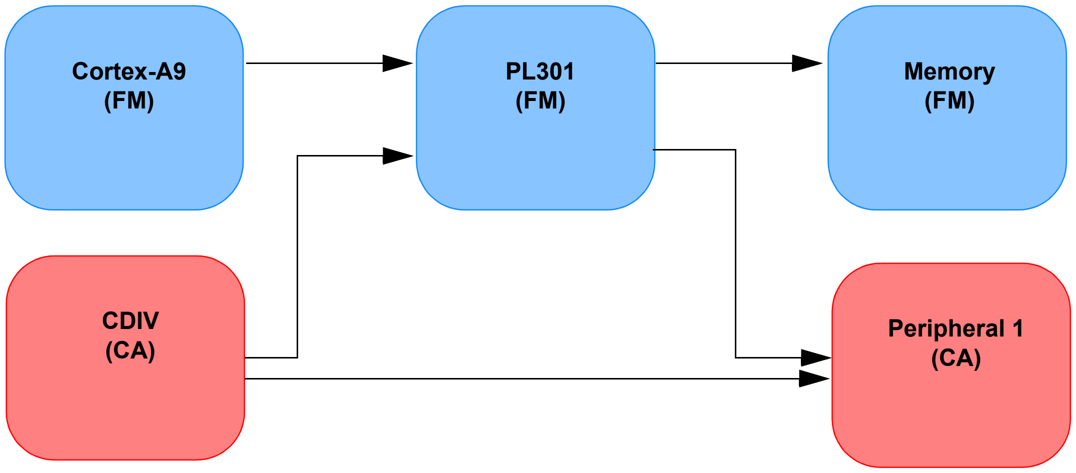


Figure 3-8 Valid Configuration for Multiple Clocking Scenario

Support for multiple clock domains is currently limited to slave and master transaction ports on a subset of the bus components, such as the PL301, NIC301, MxAXIv2, MXAHBv2 and the supporting protocol library and AMBA Misc models. If any of those ports or the component uses a clock other than the master clock and has interfaces that connect to external Cycle Accurate components, the Fast Models System Creator automatically detects the clock connections and applies them to the Fast Models system as required.

If the Fast Models System Creator encounters a non-supported component in a multiple clock domain, Fast Model-to-Cycle Accurate connection, it reports errors and fails to build a system.

3.8 Providing Address Ranges for Dissimilar Interconnects

Some Fast Models differ from their Cycle Accurate counterparts in the number of interconnects. In these cases, automatic mapping/modeling on the Fast Models is not possible, and you must provide additional information - for example, address ranges - to ensure that transactions are routed properly.

For example, the Cycle Accurate version of the processor (Figure 3-9) has four AXI master ports; instruction, data, and DMA are connected to Bus 1, and peripheral is connected to Bus 2. The Fast Models version (Figure 3-10) has a single port that connects to both buses.

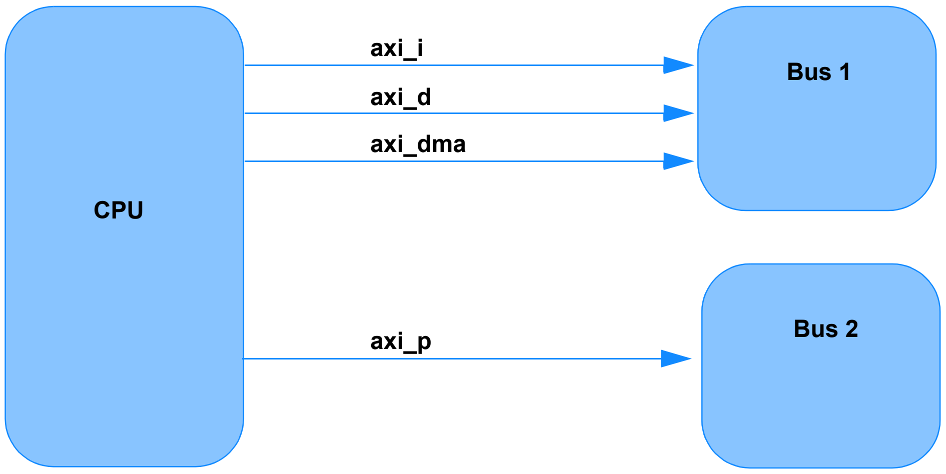


Figure 3-9 Cycle Accurate System

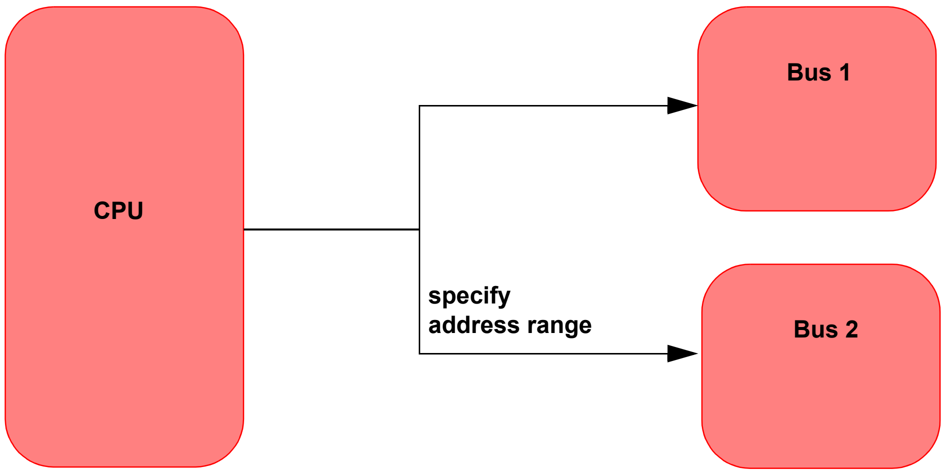


Figure 3-10 Resulting Fast Models System

To route transactions to the correct bus component, you must provide addresses. It is sufficient to provide address ranges only for ports that require them, relying on default ranges for the other ports. In the example shown in Figure 3-9 and Figure 3-10, providing an address range for the 'axi_p' port is sufficient. The three remaining ports all connect to the same component, and all addresses not in the range of 'axi_p' are routed there.

If you do not specify an address range, an error is reported because the address ranges for the two bus connections overlap.

The section "Changing the Port Range" on page 45 describes how to specify address ranges for ports.

3.9 Requirements for System Memory Map Definition

SoC Designer provides two methods for defining a system memory map:

- On slave components connected to bus components, setting Starting Address and Size parameters for memory regions (not supported for all components).
- Using the bus component to control how transactions are routed to connected slaves. There are two methods by which bus components control memory mapping:
    - Fixed, implementation-defined mapping, such as an AMBA Designer-generated bus matrix like the HPM PL301.
    - Customizable mapping controlled by the SoC Designer Memory Map Editor; use this method with the MxAHBv2 and MxAXI4, for example.

Because transaction slave ports on the Fast Models component do not support the Starting Address and Size parameters, you must do the following to ensure that memory mapping works properly:

1. Use bus-controlled memory mapping for any bus components that:
    - Remain in the Cycle Accurate system
    - Are connected to slave components to be included in the Fast Model
2. For bus components in the SoC Designer protocol bundle, set the Use MME component parameter to True (its default setting is False).

Refer to the *SoC Designer User Guide* (ARM DUI0956) for information about using the Memory Map Editor and setting component parameters.

Chapter 4

Using the Fast Models System Creator

Fast Models System Creator determines the mapping of SoC Designer components, ports, and parameters to their Fast Models counterparts. This is done based on the nearest match of the names of the component types and ports. If these mappings are not correct, you can modify them.

Some Cycle Accurate components provide a complete configuration. These configurations cannot be modified.

This chapter includes the following sections:

- Basic System Creation
- Fast Models System Creation

4.1 Basic System Creation

You must begin with a SoC Designer Canvas system of Cycle Accurate components. Figure 4-1 is a screenshot of such a system. Some components are excluded for clarity.

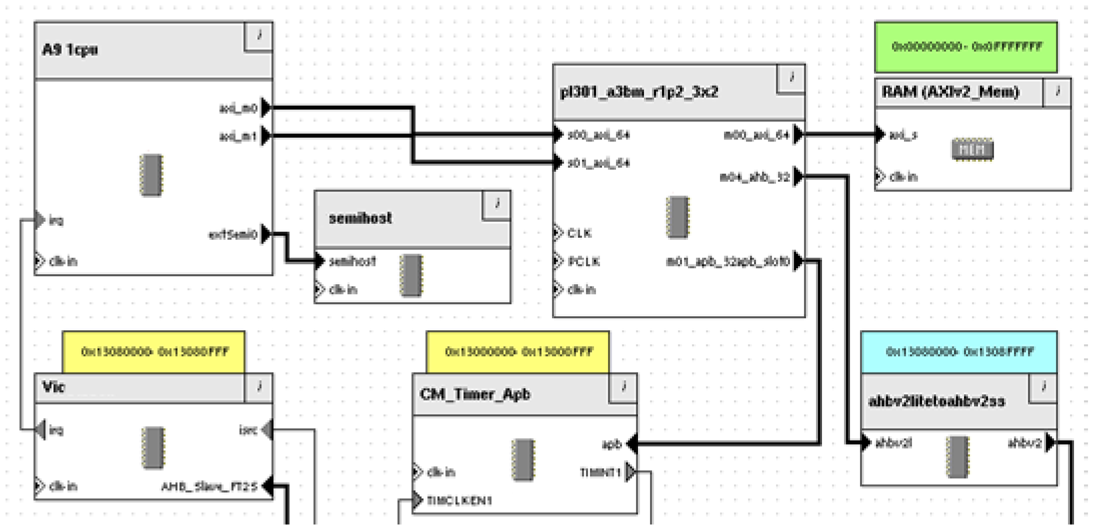


Figure 4-1 Cycle Accurate Components on the SoC Designer Canvas

4.1.1 Creating a Fast Models System

This section describes:

- Starting the Fast Models System Creator
- Specifying Top-Level Configuration Settings
- Configuring Individual Components
- Creating the Fast Models and New System

4.1.1.1 Starting the Fast Models System Creator

To run the tool:

From the SoC Designer menu, select **Tools > FastModel System…** (see Figure 4-2).

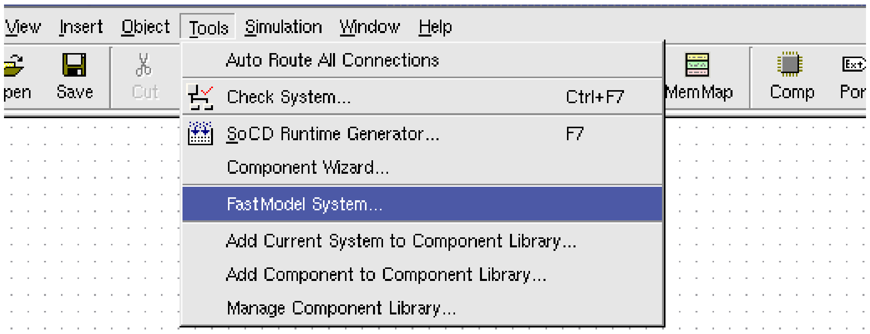
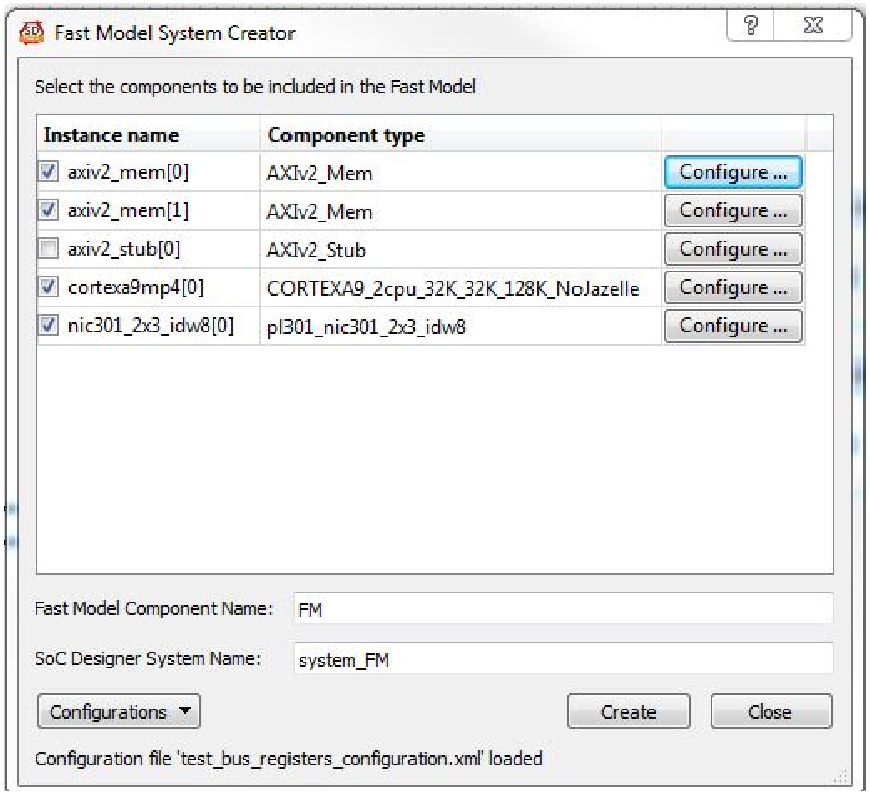

View
Insert
Object
Tools
Simulation
Window
Help
pen
Save
Cut
Auto Route All Connections
Check System...
Ctrl+F7
SoCD Runtime Generator...
F7
Component Wizard...
FastModel System...
Add Current System to Component Library...
Add Component to Component Library...
Manage Component Library...
MemMap
Comp
Por

Figure 4-2 Starting the Fast Models System Creator

This displays the Fast Models System Creator dialog (Figure 4-3):



Select the components to be included in the Fast Model

Instance name	Component type
☑ axiv2_mem[0]	AXIv2_Mem
☑ axiv2_mem[1]	AXIv2_Mem
☐ axiv2_stub[0]	AXIv2_Stub
☑ cortexa9mp4[0]	CORTEXA9_2cpu_32K_32K_128K_NoJazelle
☑ nic301_2x3_idw8[0]	pl301_nic301_2x3_idw8

Fast Model Component Name: FM

SoC Designer System Name: system_FM

Configuration file 'test_bus_registers_configuration.xml' loaded

Figure 4-3 Fast Models System Creator dialog

The Fast Models System Creator dialog allows you to partition and configure the Fast Models system. Component instances are listed, along with the type of the SoC Designer component, as well as a **Configure…** button to view/modify each component's configuration.

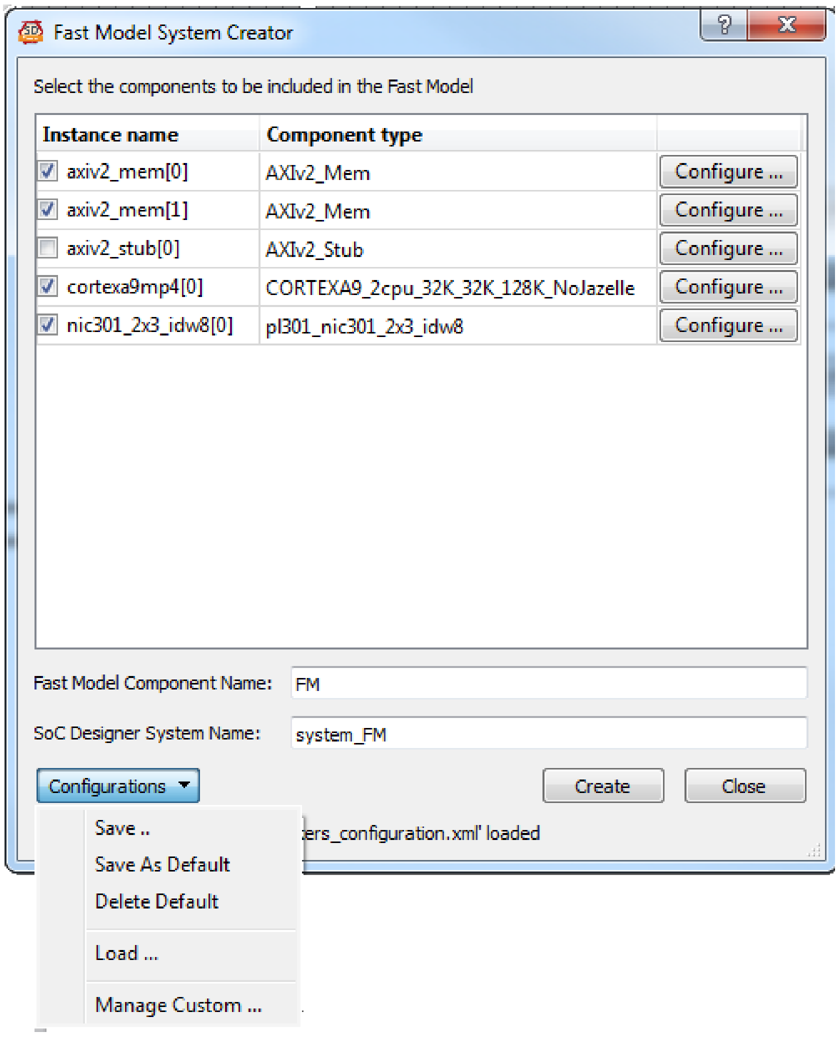
The checkbox to the left of the component name indicates whether that component is be included in the Fast Model. Initially, all components with a valid configuration are selected.

For more information on changing component configuration, refer to Section 5.

Note: For best simulation performance, include all heavily-used components (typically the processor, cache controller, bus matrix, and memory) in the Fast Model.

4.1.1.2 Specifying Top-Level Configuration Settings

Top-level configuration settings include data about the complete set of components listed in the Fast Models System Creator dialog. Use the Configurations pulldown menu (Figure 4-4) to specify, load, or save top-level configuration settings.



Fast Model System Creator

Select the components to be included in the Fast Model

Instance name	Component type	
✓ axiv2_mem[0]	AXIv2_Mem	Configure ...
✓ axiv2_mem[1]	AXIv2_Mem	Configure ...
☐ axiv2_stub[0]	AXIv2_Stub	Configure ...
✓ cortexa9mp4[0]	CORTEXA9_2cpu_32K_32K_128K_NoJazelle	Configure ...
✓ nic301_2x3_idw8[0]	pl301_nic301_2x3_idw8	Configure ...

Fast Model Component Name: FM

SoC Designer System Name: system_FM

ers_configuration.xml' loaded

Figure 4-4 Configurations Pulldown Menu

The Configurations pulldown menu includes the following options:

- **Save** - Saves the configuration data for the entire set of components listed. Note that this includes components that will not be included in the Fast Models (deselected checkboxes) as well as those to be included in the Fast Model.
- **Save As Default** - Saves the configuration data from this dialog box as an XML file for the current design. The next time you run the Fast Models System Creator, this configuration will automatically be loaded into the Fast Models System Creator dialog.
- **Delete Default** - Deletes the top-level default configuration file so it is not loaded the next time.
- **Load** - Allows you to browse and select a top-level configuration file to load. The currently-loaded configuration file is listed in the lower left corner of the dialog.

Note: When you load a top-level configuration file, its settings override any component instance-specific settings you may have specified.

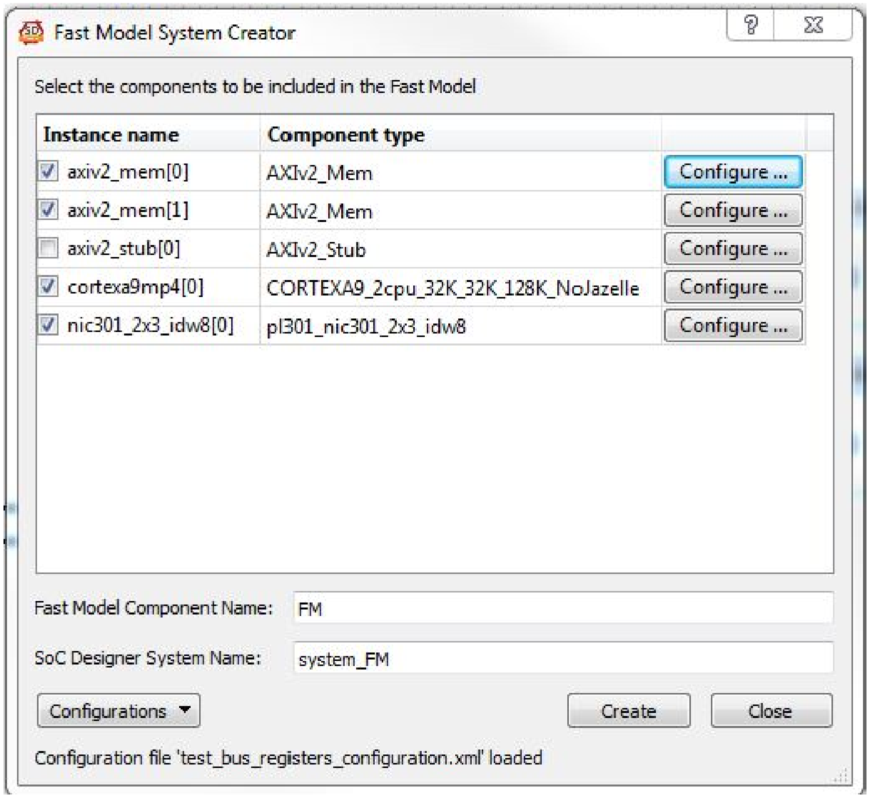
- **Manage Custom** - Allows you to include user-developed LISA components and specify custom settings. Refer to Chapter 6 for more information.

4.1.1.3 Configuring Individual Components

Clicking the **Configure…** button to the right of a component spawns a new Configuration for Component Instance dialog, allowing you to view and edit that component's properties. Refer to the section "Fast Models System Creation" on page 40 for details about this dialog's tabs and options.

4.1.1.4 Creating the Fast Models and New System

In the Fast Models System Creator dialog (Figure 4-5), follow the procedure below:



Fast Model System Creator

Select the components to be included in the Fast Model

Instance name	Component type	
☑ axiv2_mem[0]	AXIv2_Mem	Configure ...
☑ axiv2_mem[1]	AXIv2_Mem	Configure ...
☐ axiv2_stub[0]	AXIv2_Stub	Configure ...
☑ cortexa9mp4[0]	CORTEXA9_2cpu_32K_32K_128K_NoJazelle	Configure ...
☑ nic301_2x3_idw8[0]	pl301_nic301_2x3_idw8	Configure ...

Fast Model Component Name: FM

SoC Designer System Name: system_FM

Configuration file 'test_bus_registers_configuration.xml' loaded

Figure 4-5 Building the Fast Model

1. Enable the component instances that you want to include in the Fast Models by selecting their associated Instance name checkboxes. If a selected instance does not have a configuration, the text for its **Configure…** button turns red. This is a reminder that you must create a configuration for this instance.
2. Configure component-specific settings by clicking the component's associated **Configure…** button. Refer to the section "Fast Models System Creation" on page 40 for details.

Note: Create the Fast Models only after all components and ports are configured.

3. In the Fast Models Component Name field, enter a name for the new Fast Models component.
4. In the SoC Designer System Name field, enter a name for the new SoC Designer system to be created.
5. Click **Create** to build the component. This may take a few minutes.

When the model compilation is complete, the Fast Models System Creator dialog closes automatically, and a new SoC Designer system is created. The new system has the same layout as the original system, but the selected components are replaced by a single instance of the new Fast Models component (Figure 4-6).

Labels are not copied to the new system at this time.

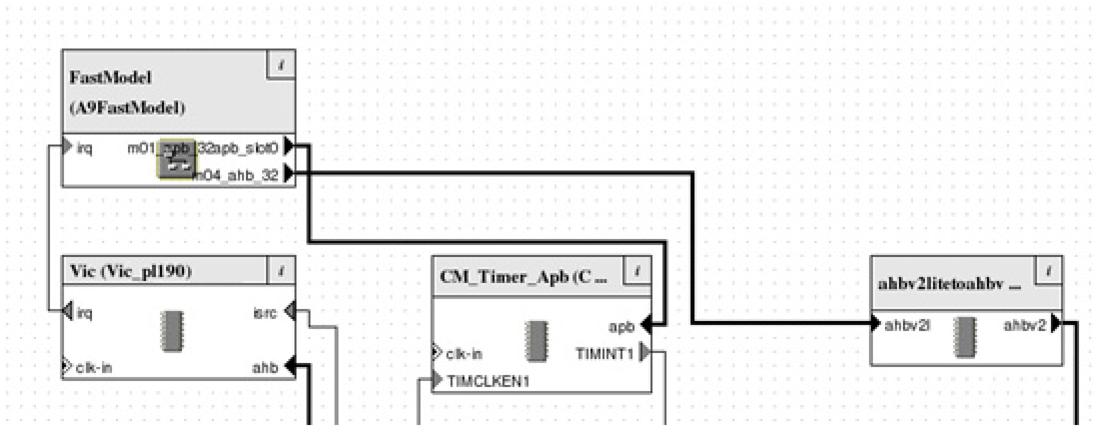


Figure 4-6 Fast Models Component on the SoC Designer Canvas

4.2 Fast Models System Creation

For each component instance listed in the Fast Models System Creator (described in "Basic System Creation" on page 34), the **Configure...** button launches a four-tab Configuration dialog. On this dialog, you can make any necessary modifications to component and port mappings, configure protocol types, port specifications, and parameter settings, and set up Swap & Play for custom-created components.

This section describes:

- Saving Configuration Changes
- Using the Basic Tab
- Using the Ports Tab
- Using the Parameters Tab
- Using Swap & Play
- Using Swap & Play with Custom Interconnects
- Working with Checkpoints

4.2.1 Saving Configuration Changes

To save configuration changes on:

- **All tabs** of the Fast Models Component Configuration dialog **for the selected instance** of a component only, enable the Restrict to this instance checkbox (Figure 4-7).
- **All tabs** of the Fast Models Component Configuration dialog at once, **for all instances** of the selected component, enable the Save Configuration checkbox (Figure 4-7).

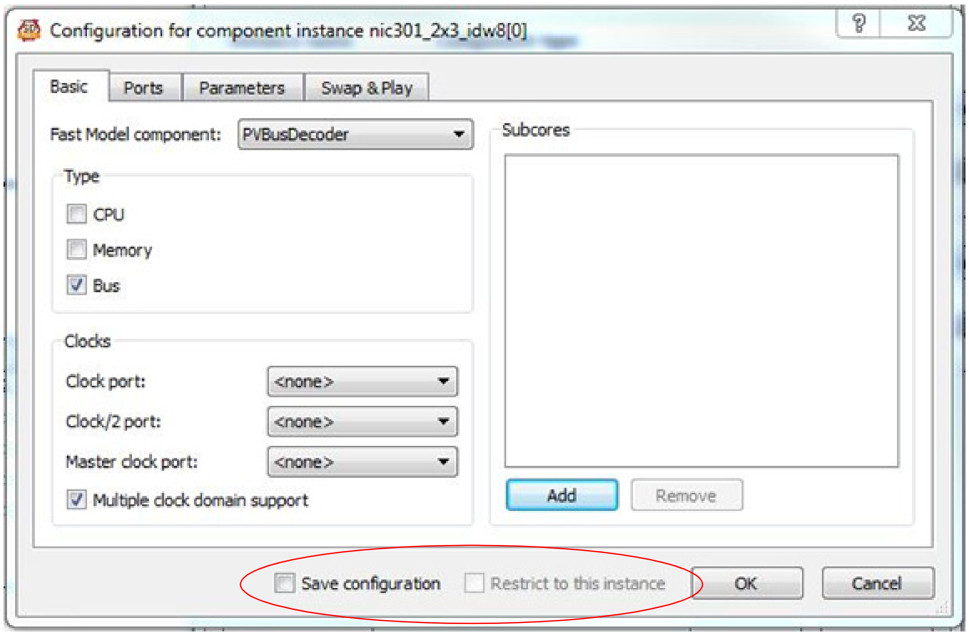

Configuration for component instance nic301_2x3_idw8[0]
Basic
Ports
Parameters
Swap & Play
Fast Model component: PVBusDecoder
Type
CPU
Memory
Bus
Clocks
Clock port: <none>
Clock/2 port: <none>
Master clock port: <none>
Multiple clock domain support
Subcores
Add
Remove
Save configuration
Restrict to this instance
OK
Cancel

Figure 4-7 Save Configuration and Restrict to Instance checkboxes

These checkboxes are always available on the Fast Models Component Configuration dialog regardless of the tab selected.

4.2.2 Using the Basic Tab

Figure 4-8 shows the Basic tab of the Fast Models Component Configuration dialog. You access this dialog by clicking the **Configure…** button next to a component on the Fast Models System Creator dialog.

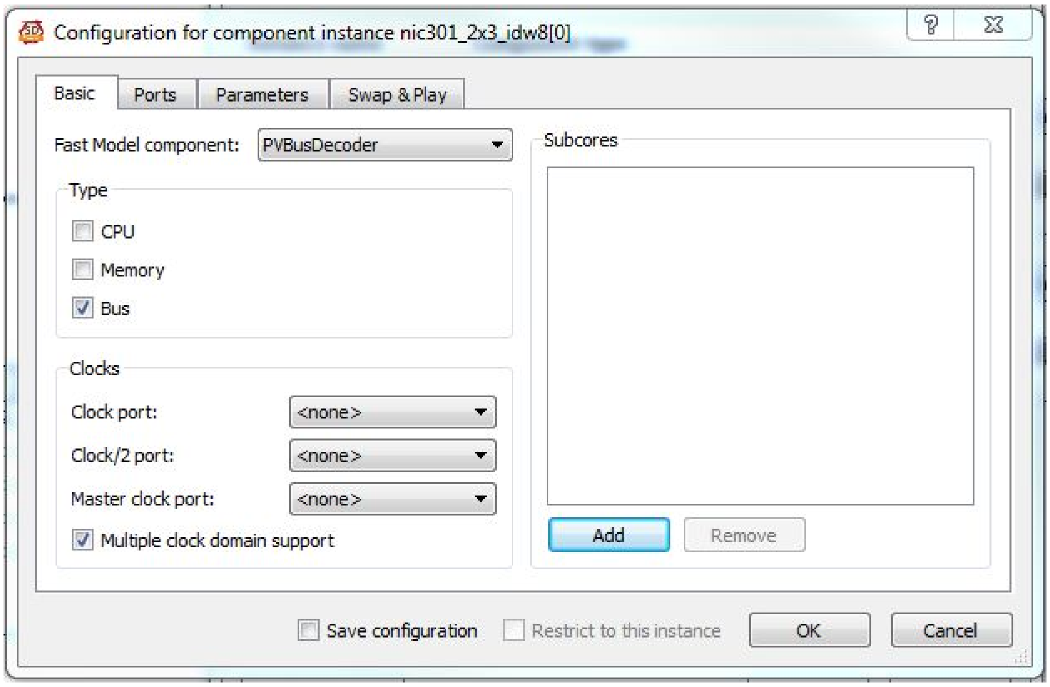

Configuration for component instance nic301_2x3_idw8[0]
Basic
Ports
Parameters
Swap & Play
Fast Model component: PVBusDecoder
Type
CPU
Memory
Bus
Clocks
Clock port: <none>
Clock/2 port: <none>
Master clock port: <none>
Multiple clock domain support
Subcores
Add
Remove
Save configuration
Restrict to this instance
OK
Cancel

Figure 4-8 Fast Models Component Configuration dialog: Basic tab

This section describes:

- Changing Component Mappings
- Setting Attributes
- Using the Clock Panel
- Using the Subcore Panel

4.2.2.1 Changing Component Mappings

You can change the Fast Models component type to which a SoC Designer component is mapped. To do so:

1. Access the Basic tab of the Fast Models Configuration dialog.
2. From the Fast Models component pulldown list, select the appropriate option.
3. Click **OK**.

Refer to the ARM Fast Models documentation for more information on specific component types.

Selecting <none> as the component type means nothing is added to the Fast Models to represent that SoC Designer component. This option is useful for components such as Semihost, which are not needed in the Fast Model.

Note: Because ports and parameters are dependent on the component type, changing the component type results in a reset of the port and parameter settings.

4.2.2.2 Setting Attributes

Certain component attributes can be set using the checkboxes in the Type area on the Basic tab of the Fast Models Configuration dialog. These attributes are:

- **CPU** - The component is a processor that can load an application file (.axf) at startup
- **Memory** - The component represents a program memory
- **Bus** - The component is a bus matrix for routing transactions

4.2.2.3 Using the Clock Panel

The Clock port, Clock/2 port, and Master clock port menus are automatically populated with the names of the pins on the Fast Model. Refer to the ARM Fast Models documentation for more information.

Multiple Clock Domain Support checkbox

If your component is configured to support multiple clock domains, then the Fast Models System Creator automatically enables the *Multiple clock domain support* checkbox.

If the component has transaction ports that will cross the Fast Model/Cycle Accurate boundary in the new system, and you receive clock boundary errors from the Fast Models generator, double-check that all transaction ports on the Ports tab have the correct clock port associated with them (refer to the section "Using the Ports Tab" on page 44), and ensure that the *Multiple clock domain support* checkbox is enabled.

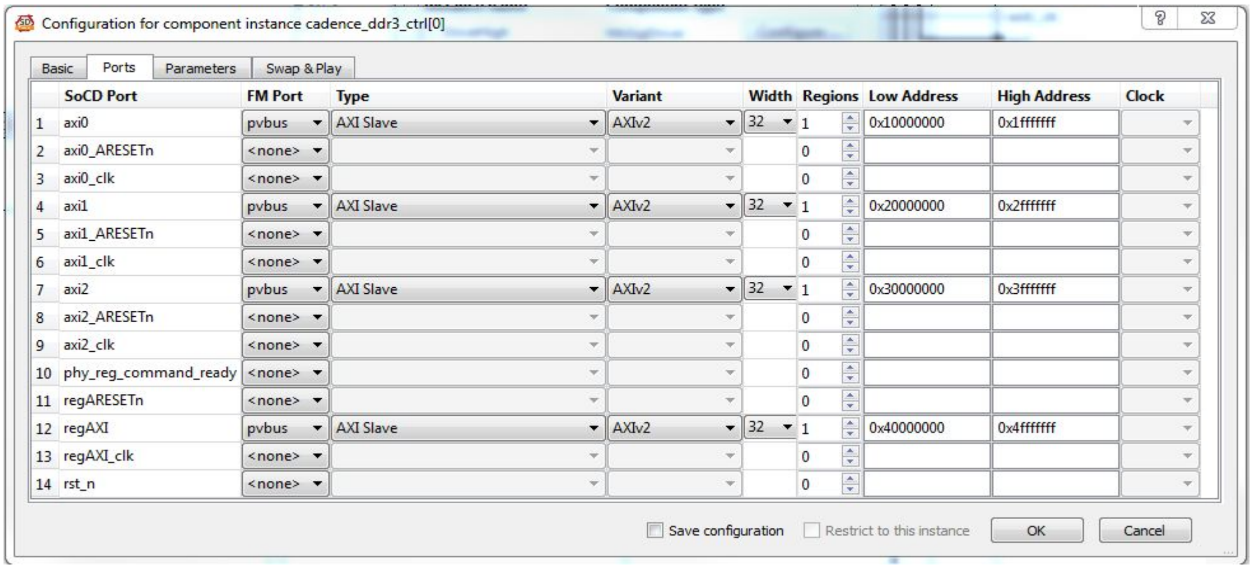
For Hybrid systems, where the clock domain crossing is entirely within either the Cycle Accurate domain or the Fast Models domain, leave the checkbox disabled.

4.2.2.4 Using the Subcore Panel

If a Cycle Accurate component has subcomponents, then the Subcores panel lists the subcores in the Fast Models that correspond to the Cycle Accurate subcomponents. Contact ARM technical support before making changes to the Subcores panel.

4.2.3 Using the Ports Tab

Figure 4-9 shows the Ports tab of the Fast Models Component Configuration dialog.



	SoCD Port	FM Port	Type	Variant	Width	Regions	Low Address	High Address	Clock
1	axi0	pvbus	AXI Slave	AXIv2	32	1	0x10000000	0x1fffffff	
2	axi0_ARESETn	<none>				0			
3	axi0_clk	<none>				0			
4	axi1	pvbus	AXI Slave	AXIv2	32	1	0x20000000	0x2fffffff	
5	axi1_ARESETn	<none>				0			
6	axi1_clk	<none>				0			
7	axi2	pvbus	AXI Slave	AXIv2	32	1	0x30000000	0x3fffffff	
8	axi2_ARESETn	<none>				0			
9	axi2_clk	<none>				0			
10	phy_reg_command_ready	<none>				0			
11	regARESETn	<none>				0			
12	regAXI	pvbus	AXI Slave	AXIv2	32	1	0x40000000	0x4fffffff	
13	regAXI_clk	<none>				0			
14	rst_n	<none>				0			

Figure 4-9 Fast Models Component Configuration dialog: Ports tab

This section describes the tasks performed on the Configuration tab:

- Changing the Fast Models Port Mapping
- Changing the Protocol Type
- Changing the Port Width
- Changing the Port Range

4.2.3.1 Changing the Fast Models Port Mapping

To change the port on the Fast Models component to which a SoC Designer port is mapped:

On the Ports tab of the Fast Models Component Configuration dialog (Figure 20), select the appropriate port from the FM Port pulldown list.

Selecting <none> for the port means that the port will not be mapped to the Fast Models component.

Refer to the ARM Fast Models documentation for more information on the available ports for a specific component.

4.2.3.2 Changing the Protocol Type

To change the protocol type for a port:

1. On the Ports tab (Figure 4-9), select the appropriate protocol type from the Type pulldown list. The list of protocols is restricted to the set (signal or transaction) that is valid for the Fast Models port type.

   For AHB ports, *pre-interconnect* refers to the interface that includes the master-side signals (e.g. the arbitration signals HBUSREQ and HGRANT). *Post-interconnect* refers to the interface that includes the slave-side signals (e.g. the select signal HSEL).
2. If applicable, select a protocol variant from the Variant pulldown list.

4.2.3.3 Changing the Port Width

To change the port width:

On the Ports tab (see Figure 4-9), select the width you want from the Width pulldown list. The list of available widths is restricted to those that are valid for the port.

4.2.3.4 Changing the Port Range

Some Fast Models port types are ranged (or addressable) and require starting and ending addresses. Refer to the ARM Fast Models documentation for more information. If the Low Address and High Address fields are grayed out, the port is not ranged.

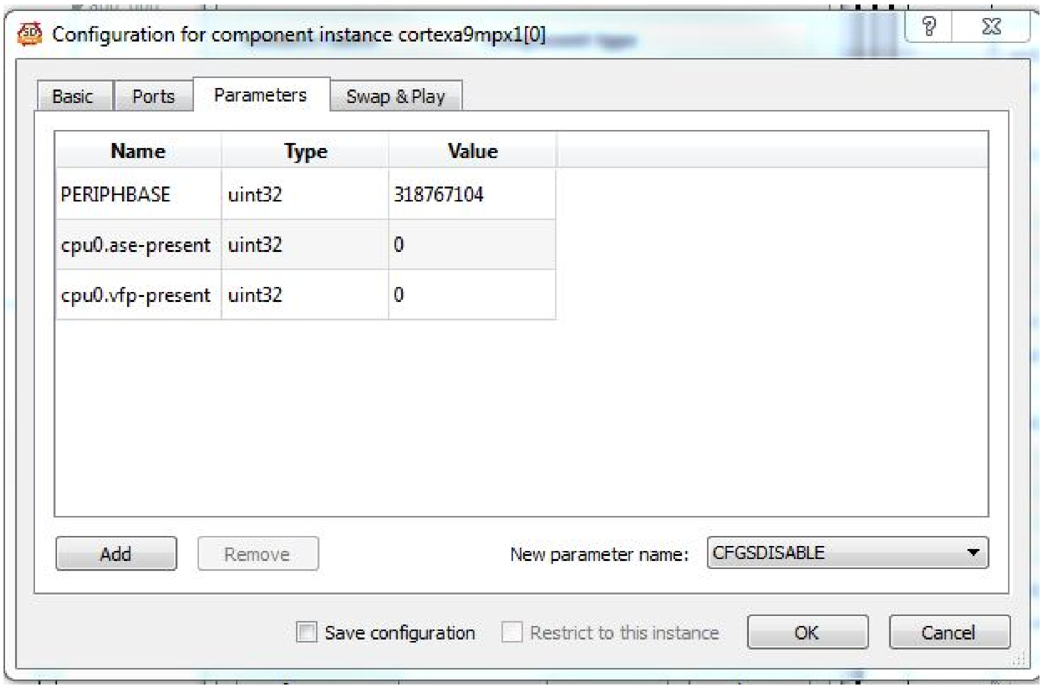
To edit the Low and High addresses of a ranged port:

1. On the Ports tab (see Figure 4-9), click in the Low Address field and enter the new value.
2. Click in the High Address field and enter the new value. Values must be entered in hexadecimal format starting with "0x".

You can change the number of address ranges associated with a port using the Regions spinbox control. True ranged ports require at least one address range. Address ranges may also be specified for non-ranged ports for managing systems with a complex interconnect scheme. Refer to "Providing Address Ranges for Dissimilar Interconnects" on page 30 for more information.

4.2.4 Using the Parameters Tab

Figure 4-10 shows the Parameters tab of the Fast Models Component Configuration dialog.


Configuration for component instance cortexa9mpx1[0]
Basic
Ports
Parameters
Swap & Play
Add
Remove
Save configuration
Restrict to this instance
OK
Cancel

Name	Type	Value
PERIPHBASE	uint32	318767104
cpu0.ase-present	uint32	0
cpu0.vfp-present	uint32	0

New parameter name: CFGSDISABLE

Figure 4-10 Fast Models Component Configuration dialog: Parameters tab

4.2.4.1 Editing a Parameter's Name and Value

To edit a parameter's value:

On the Parameters tab (Figure 4-10), click in the Type or Value field and enter the desired value.

Note: You can enter values in either decimal or hexadecimal format; hexadecimal values are then displayed as decimal values.

Refer to the ARM Fast Models documentation for more information on the available parameters for a specific component.

4.2.4.2 Adding and Removing Parameters

To add a parameter:

On the Parameters tab (Figure 4-10), select the desired parameter from the New parameter name: pulldown list and click **Add**.

You can select multiple parameters to remove at once. To remove a parameter:

Select the parameter(s) in the table and click **Remove**.

4.2.5 Using Swap & Play

This section discusses:

- Swap & Play Procedure and Usage Notes
- Swap & Play Tab
- Adding a Register Remap
- Loading Virtual Registers

For more about the Swap & Play feature, see "Overview of Swap & Play" on page 12.

4.2.5.1 Swap & Play Procedure and Usage Notes

To use Swap & Play:

1. Use SoC Designer to simulate a system containing a Fast Models component.
2. Stop the simulation at the point where you want to create a checkpoint. Note that the processor in the system must be at a debuggable point (see the following section).

Usage Note: Running to Debuggable Point

If the simulation was not stopped because of a software breakpoint or single-stepping, then before continuing you must use the "Run to Debuggable Point" feature of SoC Designer. There are two ways to do this:

- Click Run to Debuggable Point in the Disassembler window.
- Use the context-sensitive component menu:
  1. Right-click on a Fast Models component.
  2. From the menu, select **Run Child to Debuggable Point**.
  3. Select the system's CPU.

Usage Note: Database Reset due to Predated Time Stamp

This note applies if you are using System Analyzer, and you are specifying cycle ranges in which to collect data.

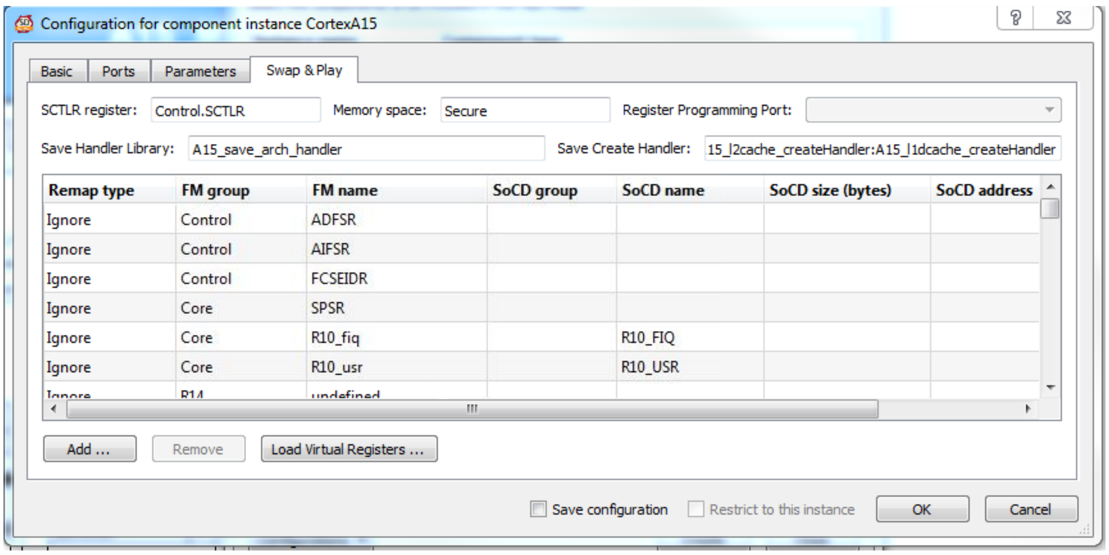
If you restore a checkpoint that has a simulation time stamp (cycle count) earlier than the current time stamp, any data previously recorded between the current time stamp and the checkpoint's time stamp is dropped. Data is collected after the checkpoint starting at the checkpoint's timestamp.

For example, you might start a simulation and begin monitoring at 10,000 cycles, then remove the monitor at 20,000 cycles. If you then load a checkpoint that starts at 15,000 cycles, your simulation has moved backward in time; under these circumstances System Analyzer would reset the database — not just for the monitored connection, but for the entire database.

4.2.5.2 Swap & Play Tab

The Swap & Play tab of the Fast Models Component Configuration dialog allows you to map Fast Models registers to registers of the SoC Designer model; there may not always be a 1-to-1 mapping. Where a one-to-one mapping exists between the Fast Models and SoC Designer group names and register names, you do not need to specify entries on this tab; their values are saved and restored automatically.

Figure 4-11 shows the Swap & Play tab on the Fast Models Component Configuration dialog:


Configuration for component instance CortexA15
Basic
Ports
Parameters
Swap & Play
Add ...
Remove
Load Virtual Registers ...
Save configuration
Restrict to this instance
OK
Cancel

SCTLR register: Control.SCTLR
Memory space: Secure
Register Programming Port:
Save Handler Library: A15_save_arch_handler
Save Create Handler: 15_l2cache_createHandler:A15_l1dcache_createHandler

Remap type	FM group	FM name	SoCD group	SoCD name	SoCD size (bytes)	SoCD address
Ignore	Control	ADFSR				
Ignore	Control	AIFSR				
Ignore	Control	FCSEIDR				
Ignore	Core	SPSR				
Ignore	Core	R10_fiq		R10_FIQ		
Ignore	Core	R10_usr		R10_USR		
Ignore	R14	undefined				

Figure 4-11 Fast Models Component Configuration dialog: Swap & Play tab

This dialog box features:

- **SCTLR Register field** — The name of the System Control Register of an ARM processor. It is specific to ARM CPUs and takes effect only if the CPU checkbox in the Basic tab of the dialog is enabled.

  The Fast Models checkpoint must distinguish between physical and virtual address space, and uses the state of the System Control Register to make this distinction.

- **Save Handler Library** and **Save CreateHandler fields** — These fields are used only with user-developed models that implement architectural cache Save/Restore. Refer to "Cache Save/Restore" on page 75 for more information.

- **Memory Space field** — The name of the system memory space to be saved when a CPU is checkpoint saved. This takes effect only when the CPU check box in the Basic tab of the dialog is enabled. The name of the memory space must be meaningful to CADI (the ARM Cycle Accurate Debug Interface).

- **Register Programming Port field** — This field is for memory controllers. Specify the port to be used for programming the registers.

- **Remap Type column** — Determines how, or whether, to map a Fast Models register to a SoC Designer register. Refer to Section 4.2.5.3, Adding a Register Remap.

- **FM group column** — Currently-mapped Fast Models group for this register.

- **FM name column —** Currently-mapped Fast Models name for this register.
- **SoCD group column —**Currently-mapped SoC Designer group for this register.
- **SoCD name column —** Currently-mapped SoC Designer name for this register.
- **SoCD size (bytes) column —** Size of this register in bytes.

Note: If the SoC Designer model has a different register size, specify it in the SoCD Size field.

- **SoCD offset column —** Address as a hexadecimal value.
- **Add… Button —** Launches the Add Register Remap dialog, which is used to map Fast Models registers to SoC Designer registers for the checkpoint save. Refer to Section 4.2.5.3, Adding a Register Remap.
- **Remove Button —** Removes the selected mapping(s).
- **Load Virtual Registers —**Allows you to load an XML file populated with a device's register state. Refer to Section 4.2.5.4, Loading Virtual Registers.

4.2.5.3 Adding a Register Remap

Register Remap determines how, or whether, to map a Fast Models register to a SoC Designer register. Because CADI, which implements the checkpoints, handles registers with a group name in its API, the group name must be considered in the mapping.

Note: If the SoC Designer model has a different register size, specify it in the SoCD Size field.

Note: Be aware of potential risks when using Register Remap with Swap & Play of Custom Models. This method is sufficient for creating checkpoints for Swap & Play in cases where writing a register into the SoC Designer model has no side effects. However, if the register restored has side effects, the CADI implementation of register writes must handle all side-effects and ensure there are no remaining side effects.

Remap Types

The Remap types are:

- **Ignore** - Indicates that you want a register to be ignored (i.e., not saved) in the checkpoint save. You must provide the register group (FM Group field) and register name (FM Name field).
- **Remap** - Use when registers from the Fast Models have the same group name but different register names in the SoC Designer model. You must provide the Fast Models group name (FM Group field) and register name (FM Name field), and the SoC Designer model Register name (SoCD Name field).
- **Group Remap** - Use when both the Fast Models group names and Fast Models register names differ from the SoC Designer model group names and register names. You must provide the Fast Models group name (FM Group field) and register name (FM Name field), and the SoC Designer model group name (SoCD Group field) and register name (SoCD Name field).
- **Full Group Remap** - Use when register groups in the Fast Models have the same register names in the SoC Designer model, but the group names are different. You must provide

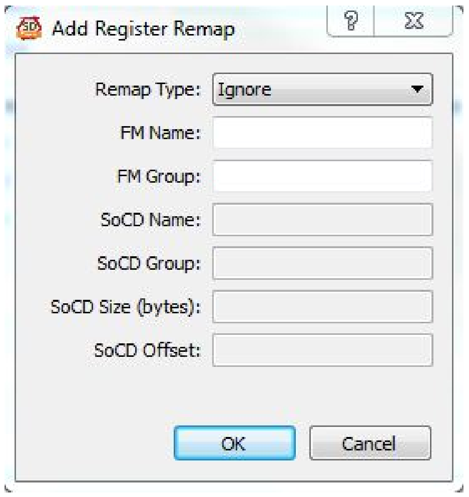
only the group names (FM Group and SoCD Group fields); all registers are assumed to be the same size between the Fast Models and SoC Designer Model.

- **Virtual Remap** - Use for specifying interconnect device register. This allows the Simulator to restore the device in the correct state during Swap & Play (see Section 5.4.2, Loading Virtual Registers).

Procedure to add a Register Remap

To add a register remap:

1. On the Swap & Play tab of the Fast Models Component Configuration dialog (Figure 4-11), click **Add Register Remap**. The Register Remap dialog opens (Figure 4-12).



Remap Type: Ignore

Figure 4-12 Add Register Remap dialog

2. Enter the required data on this dialog box. Non-required fields are automatically grayed out.
3. Click **OK**.

4.2.5.4 Loading Virtual Registers

The Load Virtual Registers option allows you to load a device's register data. The device's register content is included in the `.ccfg` file generated by Cycle Model Studio. Refer to the *Cycle Model Studio User Manual* for more information.

When you load a device's register state into the Swap & Play tab of the Fast Models Component Configuration dialog, the Fast Models Creator can restore this data in the Cycle Accurate domain for Swap & Play and Save/Restore.

To load virtual register data:

1. On the Swap & Play tab of the Fast Models Component Configuration dialog (see Figure 4-11), click **Load Virtual Registers**. A Browse dialog launches.
2. Select the XML file that contains the register state.
3. Click **OK**.

The Swap & Play tab is populated with the data from the register XML file.

4.2.6 Using Swap & Play with Custom Interconnects

If you have a custom interconnect device that has registers whose state is needed during the Cycle Accurate simulation, then you need to create Virtual Registers to represent these registers in the Fast Models domain. There are two ways to do this:

- For devices created in Cycle Model Studio, you can export the registers to a file using the Export command in the Cycle Model Studio Registers tab (see the *Cycle Model Studio User Manual* Section 5.2.4.2, Registers Tab).

  After you export the register file, import it into the Fast Models dialog using the Load Virtual Registers command (see "Loading Virtual Registers" on page 51). You may want to save the configuration so you don't have to load this file each time you generate the Fast Model.
- You can create the Virtual Registers in the Fast Models dialog's Swap & Play tab by using the **Add…** command. Each Virtual Register needs:
  - a SoC Designer Name,
  - a SoC Designer group name,
  - the address of that register, and
  - the size in bytes.

Don't forget to save the configuration, otherwise you must create these registers every time you regenerate the Fast Model.

There are two known limitations with this approach:

- When the Virtual Registers are programmed during the Fast Models simulation, any side effects are lost when the checkpoint is loaded during the Cycle Accurate simulation. This means that if programming a register causes other state changes in the model, those other changes are lost when the checkpoint is loaded.
- If more than one slave port can be used to program the registers in the device, they must both use the same base address. Some devices can have different base addresses depending on the port being used to program the registers. These base addresses must be the same.

4.2.7 Working with Checkpoints

SoC Designer provides a Checkpoint Manager for Swap & Play to:

- Create and organize your checkpoints (see Section 4.2.7.1)
- Launch simulations from checkpoints (see Section 4.2.7.2)

4.2.7.1 Creating Checkpoints

There are two ways to create checkpoints. Both are discussed in this section:

- Manually using the Checkpoint dialog
- Using the Checkpoint Manager

Creating Checkpoints Manually

To create a checkpoint, from within SoC Designer Simulator:

1. Click **Save** or select **File > Save** or **File > Save As** from the menu. In the Save as… dialog (Figure 4-13), select "Swap & Play Checkpoint (*.mxc)".

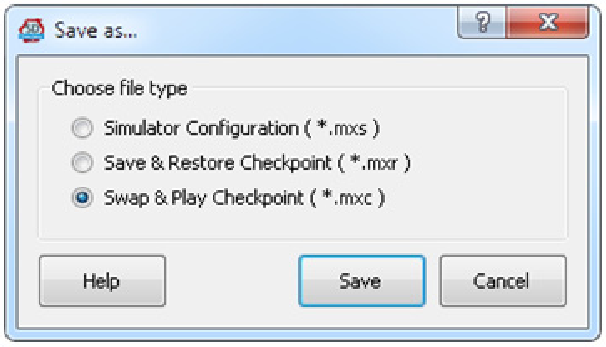

Save as...
Choose file type
Simulator Configuration (*.mxs)
Save & Restore Checkpoint (*.mxr)
[✓ Swap & Play Checkpoint (*.mxc)]

Figure 4-13 Save As dialog for Checkpoints

Note: *When you save a Swap & Play checkpoint (.mxc file), a second file is automatically created, which contains all memory data for that checkpoint. This file has the extension .mem and is stored in the same location as the .mxc file. If you copy or move the .mxc file, ensure that you also copy or move its associated .mem file. When you load a checkpoint, its associated .mem file is loaded automatically provided the two files are in the same location.*

2. Click **Save**. The Save Checkpoint dialog appears (Figure 4-14), in which you can configure the checkpoint.

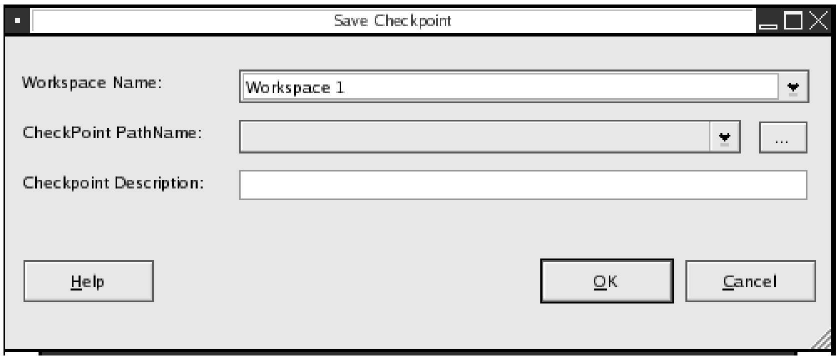


Figure 4-14 Save Checkpoint dialog

3. In the Workspace Name: field, enter a workspace name.
4. Click the '**...**' button to select a Checkpoint PathName. The SoC Designer file browser displays (Figure 38). Browse to a different directory if desired. When you enter a name in the File name field, the .mxc extension is added automatically. Alternatively, you may select an existing checkpoint file to overwrite it.

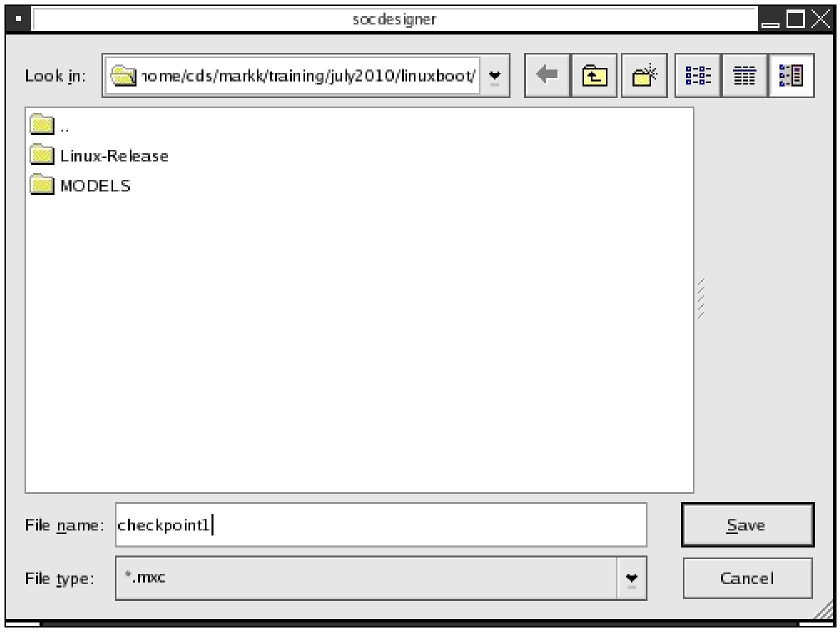
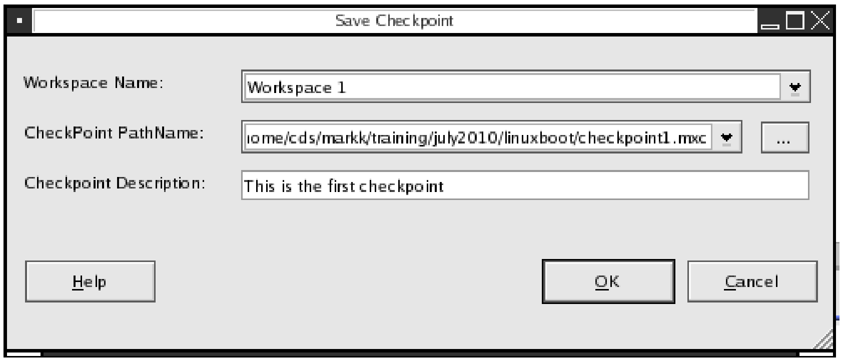


Figure 4-15 Naming a Checkpoint

5. Click **Save**. You are returned to the Save Checkpoint dialog.

6. In the Checkpoint Description field, enter a description for the checkpoint.


Save Checkpoint

Workspace Name: Workspace 1

CheckPoint PathName: ıome/cds/markk/training/july2010/linuxboot/checkpoint1.mxc

Checkpoint Description: This is the first checkpoint

Figure 4-16 Save Checkpoint dialog with Description

7. Click **OK** to complete the checkpoint creation.

Note that you can create multiple checkpoints from a single Fast Models simulation. Simply continue to the next simulation point and repeat the above steps.

Using the Checkpoint Manager to create checkpoints

After the first checkpoint has been created for a system, you can create additional checkpoints through the Checkpoint Manager:

1. Invoke the Checkpoint Manager by clicking its button in the SoC Designer toolbar, labeled **ChkPtMgr** (Figure 4-17).

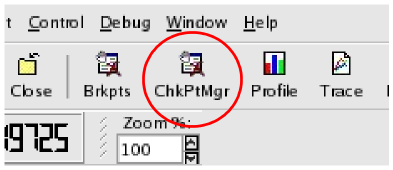
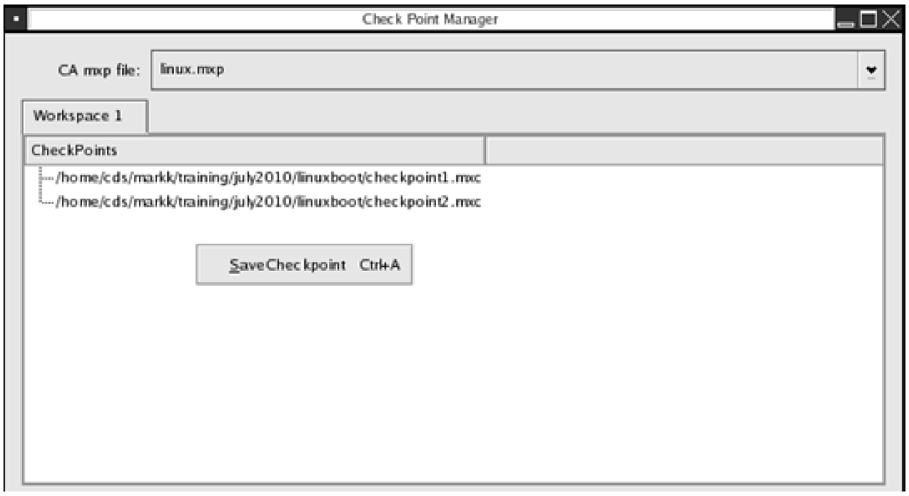


Figure 4-17 Checkpoint Manager button

2. In the Checkpoint Manager dialog, right-click in a blank area of the checkpoint list.

3. From the context menu, select **Save Checkpoint** (Figure 4-18).



/home/cds/markk/training/july2010/linuxboot/checkpoint1.mxc
/home/cds/markk/training/july2010/linuxboot/checkpoint2.mxc

Figure 4-18 Save Checkpoint context menu

4.2.7.2 Launching Simulations from Checkpoints

There are two ways to launch simulations from Checkpoints. Both are described in this section:

- Manually from the SoC Designer Simulator File menu
- Via the Checkpoint Manager

Note: If you are using System Analyzer, refer to the section "Usage Note: Database Reset due to Predated Time Stamp" on page 47 *for information about database behavior during checkpoint restoration.*

Manually Launching Simulations

To manually launch a simulation from a checkpoint:

1. Open the system in SoC Designer Simulator.
2. Before starting the simulation, select **File > Restore checkpoint view** (Figure 4-19)

.

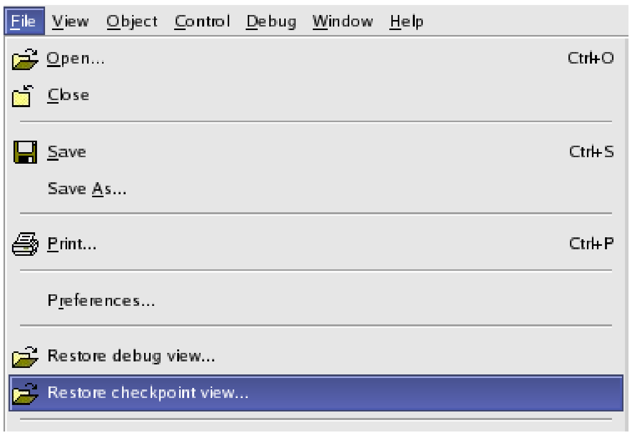


Figure 4-19 Restore Checkpoint View menu selection

3. In the Load component state dialog (Figure 4-20), browse to the checkpoint file you would like to load:

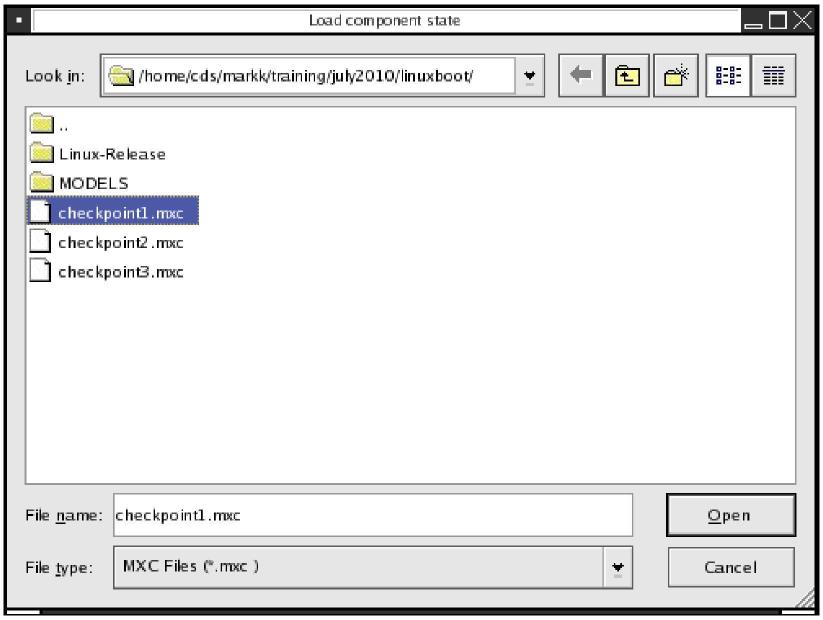

Load component state
Look in: /home/cds/markk/training/july2010/linuxboot/
..
Linux-Release
MODELS
checkpoint1.mxc
checkpoint2.mxc
checkpoint3.mxc
File name: checkpoint1.mxc
File type: MXC Files (*.mxc)
Open
Cancel

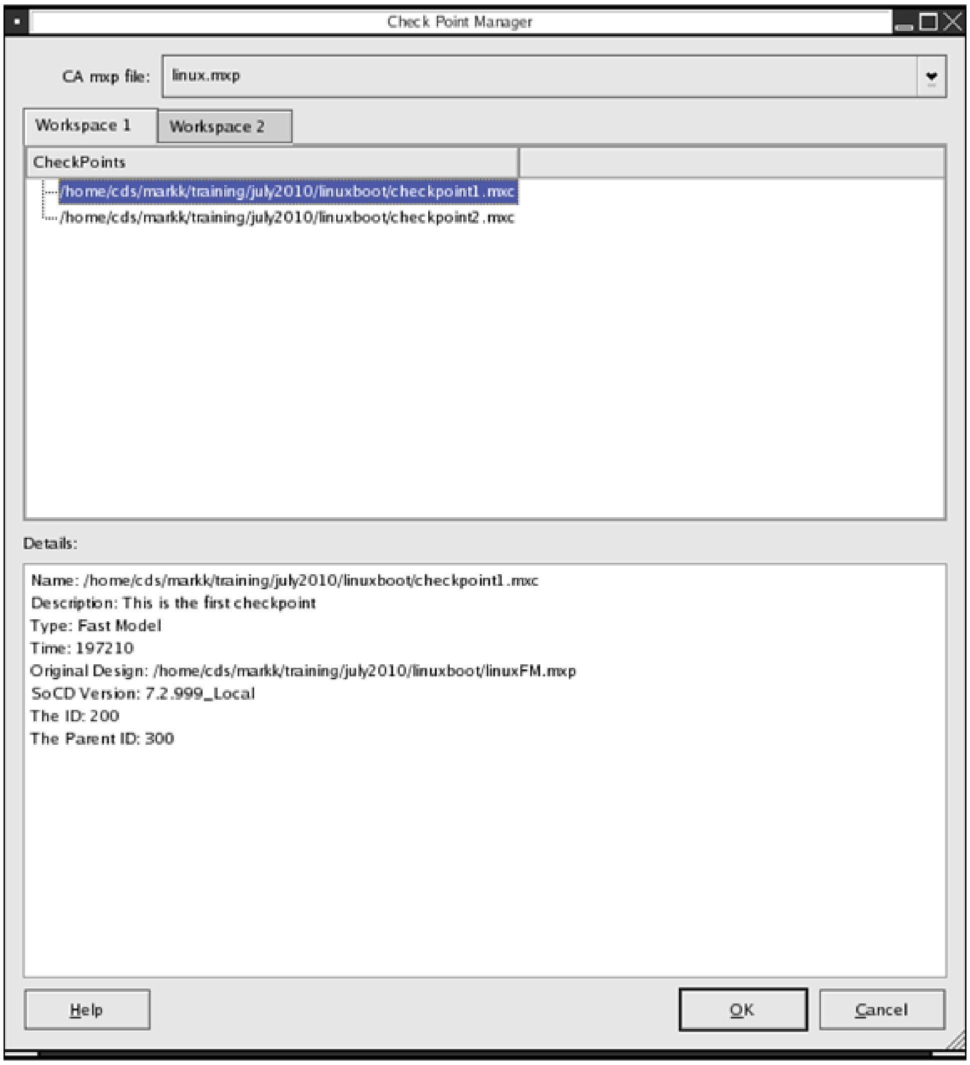
Figure 4-20 Load Component State dialog

4. Click **Open**.

Using the Checkpoint Manager to Launch Simulations

To use the Checkpoint Manager to launch a simulation:

1. Open the Checkpoint Manager and browse to the desired checkpoint.
2. Select the SoC Designer system from the **CA .mxp File:** pulldown menu (Figure 4-21).



CA mxp file: linux.mxp

/home/cds/markk/training/july2010/linuxboot/checkpoint1.mxc
/home/cds/markk/training/july2010/linuxboot/checkpoint2.mxc

Name: /home/cds/markk/training/july2010/linuxboot/checkpoint1.mxc
Description: This is the first checkpoint
Type: Fast Model
Time: 197210
Original Design: /home/cds/markk/training/july2010/linuxboot/linuxFM.mxp
SoCD Version: 7.2.999_Local
The ID: 200
The Parent ID: 300

Figure 4-21 Check Point Manager dialog

The tabs on the top left select among the workspaces for the system. The checkpoints within a workspace are displayed in the upper half of the dialog. When selected, a checkpoint's properties are displayed in the lower half (Details pane) of the dialog.

3. Right-click a checkpoint and select **Restore Checkpoint** (Figure 4-22). This spawns a new SoC Designer process and loads the checkpoint.

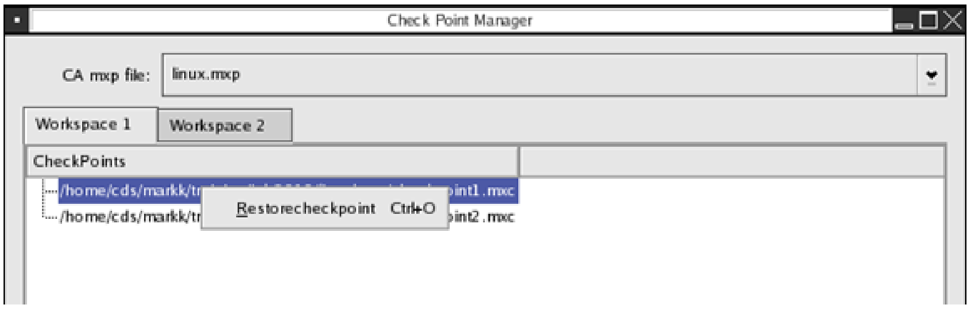


Figure 4-22 Restore Checkpoint context menu

4.2.7.3 Application File Handling During Checkpoint Restore

When loading a simulation (either manually or through the Checkpoint Manager) to restore a checkpoint, you are presented with the dialog used for loading application files into processor instances. No applications should be loaded through this method, since the application is part of the checkpoint state.

Chapter 5

Using Debuggers with Fast Models Simulations

For instructions on starting the SoC Designer Simulator using the UI or command line, refer to the *SoC Designer User Guide* (ARM DUI0956). Refer to the section Requirements on page 15 for supported debugger versions.

This chapter describes:

- Launching ModelDebugger for Fast Models CPUs
- Simulating with DS-5
- Simulating without a Debugger

5.1 Launching ModelDebugger for Fast Models CPUs

SoC Designer supports bringing up or attaching a Fast Models ModelDebugger for an ongoing simulation from within the SoC Designer GUI.

To do so:

1. Right-click on the CPU block (or "FastModel" block) to display the context menu. See Figure 5-1.

2. Select **Launch ModelDebugger Debugger** and select the desired CPU instance.

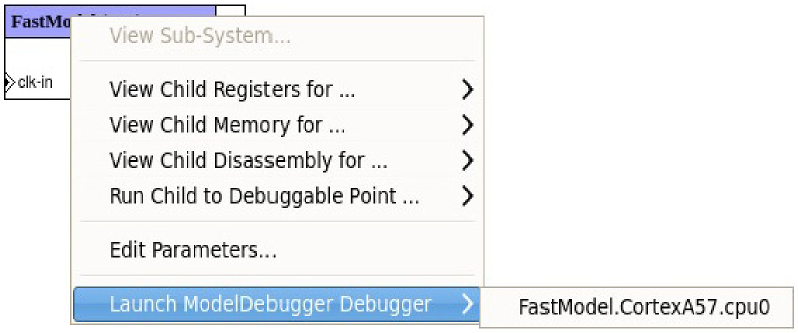

FastMo
clk-in
View Sub-System...
View Child Registers for ...
View Child Memory for ...
View Child Disassembly for ...
Run Child to Debuggable Point ...
Edit Parameters...
Launch ModelDebugger Debugger
FastModel.CortexA57.cpu0

Figure 5-1 Selecting

Known limitations

For a multi-core system, although options are visible for various CPUs in the "Launch Model-Debugger.." menu, selecting any of the CPUs brings up ModelDebugger connected to *all* the Fast Models CPUs in the system.

5.2 Simulating with DS-5

Instructions for integrating the ARM DS-5 Debugger with SoC Designer are available in the *SoC Designer DS-5 Debugger Integration Application Note*.

5.3 Simulating without a Debugger

When simulating without a Debugger, CADI access to the registers, memory, and disassembly of the Fast Models can be done through the normal SoC Designer interfaces. Each CADI-enabled component in the Fast Models appears as a child of the Fast Models SoC Designer component.

Chapter 6

Managing User-Developed Component Files and Configurations

The Fast Models System Creator allows you to include LISA files, created by you or a third party, in your Fast Models system.

To access the Custom Fast Models Manager dialog that handles this functionality:

> From the Configurations pulldown menu on the Fast Models System Creator dialog (Figure 4-4 on page 37), select "Manage Custom." The Custom Fast Models Manager dialog displays (Figure 6-1).

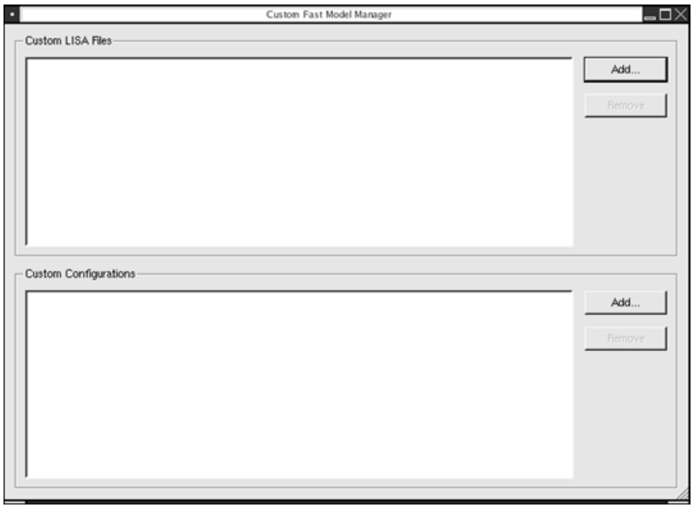


Figure 6-1 Custom Fast Models Manager dialog

This chapter includes the following sections:

6.1 Importing User-Developed LISA Components

To import a custom LISA file to your Fast Models System:

1. Click the **Add** button in the "Custom LISA Files" section. This displays the Import Custom LISA File dialog (Figure 6-2).

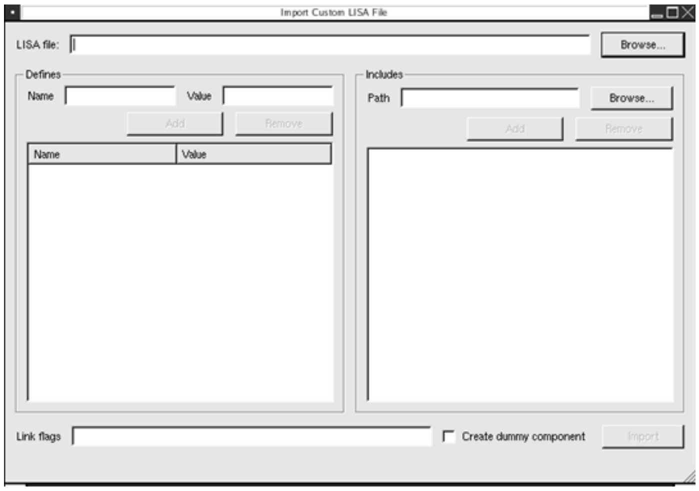
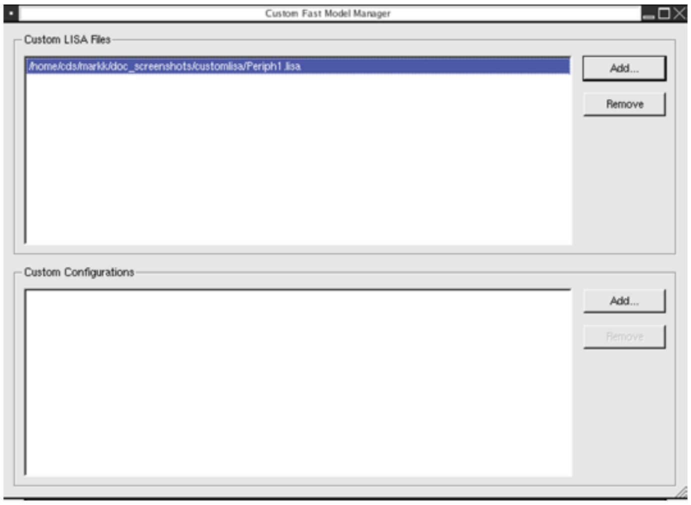


Figure 6-2 Import Custom LISA File dialog

2. Click **Browse** to browse to your LISA file. You can also enter the path manually.
3. Click **Import** to finish the component import. The file is parsed and added to the list of saved configurations (Figure 6-3). The component is also added to the set of available

Fast Models components. It can now be selected as the replacement for a Cycle Accurate component as described in "Changing Component Mappings" on page 42.



/home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

Figure 6-3 Custom Fast Models Manager dialog showing path

6.1.1 Specifying Additional Import Options

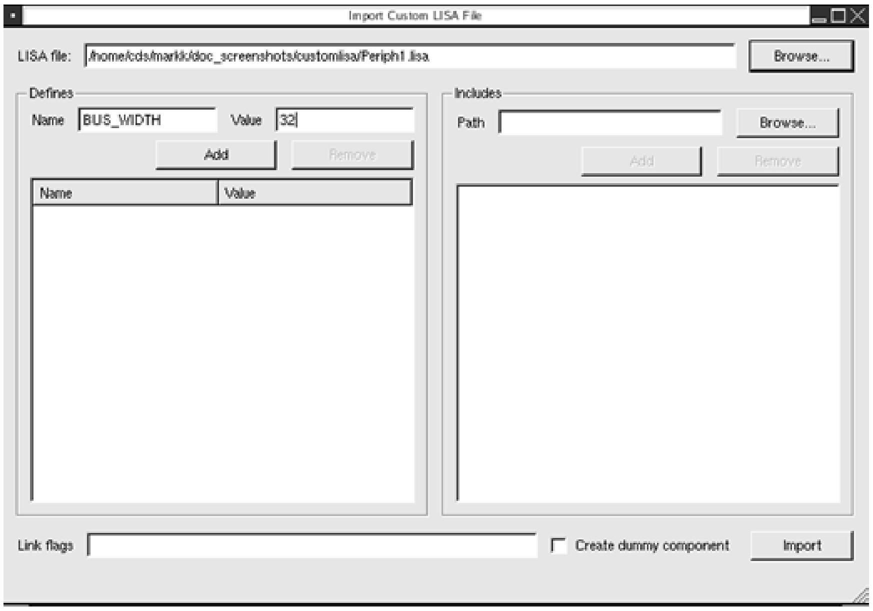
You can add defines, include paths, and link flags to an imported LISA component. These are used not only for the parsing of the LISA component during the import, but also when building the System Generator system using the component. Specify these additional options before clicking the **Import** button.

This section discusses:

- Adding Defines
- Adding Include Paths
- Adding Link Flags

6.1.1.1 Adding Defines

To add a define, type its name and value into the corresponding boxes in the Defines panel (Figure 6-4). Alternatively, you can enter the path manually:



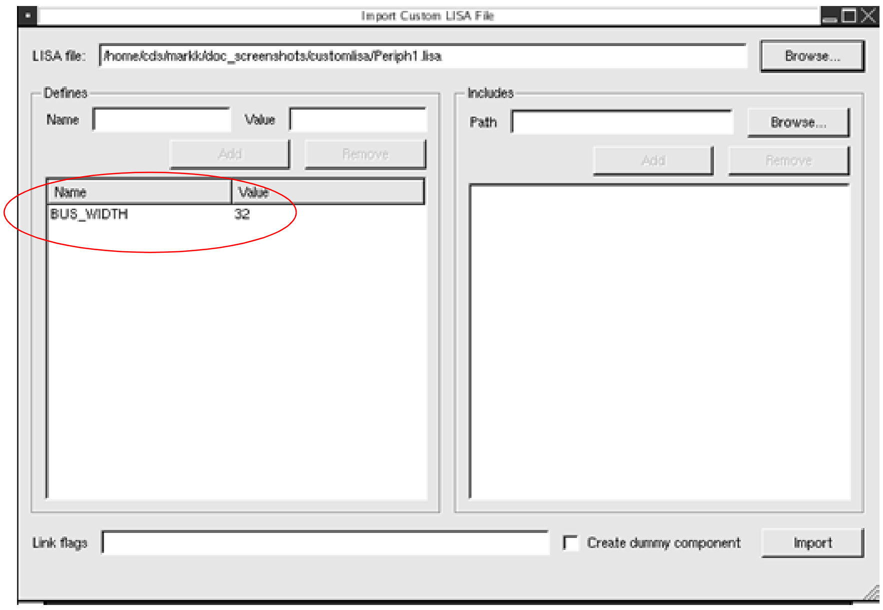
LISA file: /home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

Name: BUS_WIDTH Value: 32

Name	Value

Figure 6-4 Adding a Define

Click **Add** to add this define. It is displayed in the Defines panel (Figure 6-5):



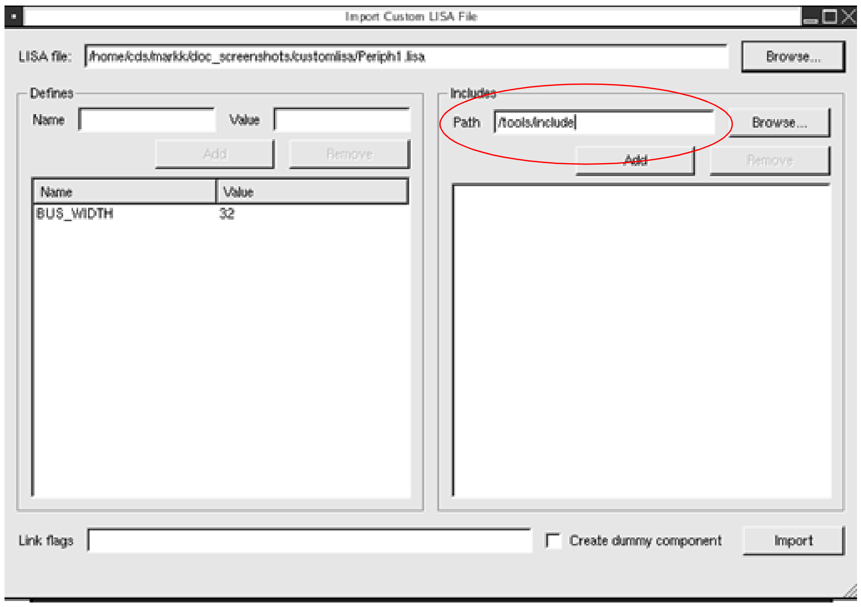
LISA file: /home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

Name	Value
BUS_WIDTH	32

Figure 6-5 Define added

6.1.1.2 Adding Include Paths

To add an include path, click **Browse** in the Includes section and select the path. Alternatively, you can enter the path manually (Figure 6-6).



LISA file: /home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

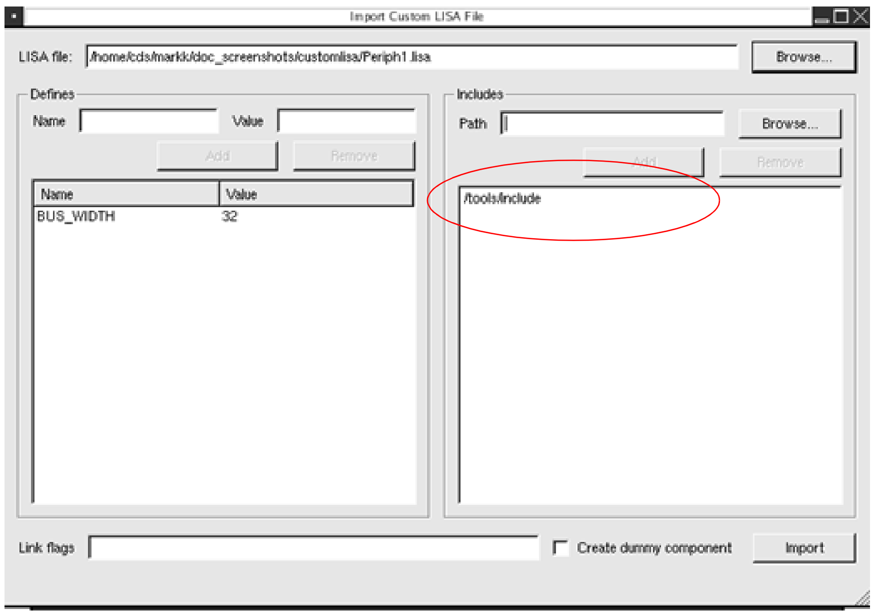
Path: /tools/include

Name	Value
BUS_WIDTH	32

Link flags:

Figure 6-6 Adding an Include

Click **Add** to add this include path. It is displayed in the Includes pane (Figure 6-7):



LISA file: /home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

Name	Value
BUS_WIDTH	32

/tools/include

Link flags:

Figure 6-7 Include added

To remove a define or include path, select its entry in the Includes panel and click **Remove**.

6.1.1.3 Adding Link Flags

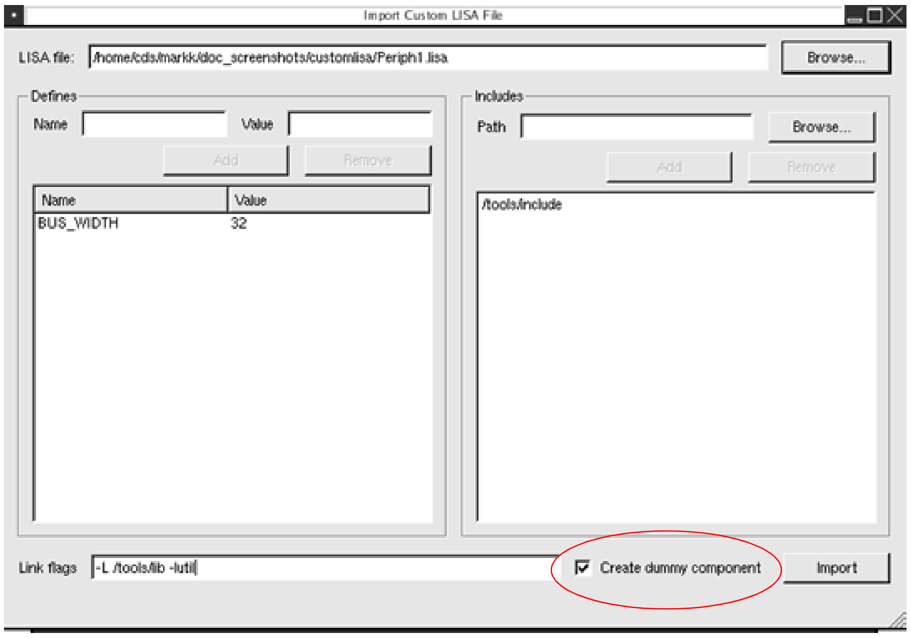
To add link flags, enter them in the "Link flags" field.

Note: Link flags are not actually used when linking the System Generator system at this time.

6.2 Creating a Dummy Cycle Accurate Component

In some cases, a LISA component may exist, but a Cycle Accurate component may not. In order to create a Fast Models system using the custom LISA component, a Cycle Accurate component is required to describe system connectivity. The LISA import tool can automatically create this component.

To enable this feature, select the "Create dummy component" checkbox before clicking Import (Figure 6-8).


Import Custom LISA File

LISA file: /home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

Defines

Name	Value
BUS_WIDTH	32

Includes

/tools/include

Link flags: -L /tools/lib -lutil

[✓] Create dummy component

Figure 6-8 Create dummy component checkbox

After the dummy component has been created, close the Fast Models System Creator dialog and return to the SoC Designer Canvas. Add the newly-created component to your system and make the appropriate connections. Save the system and restart the Fast Models System Creator dialog.

Note: The dummy component is to be used for system connectivity only. It cannot be used for simulation.

6.3 Restrictions on Custom LISA Components

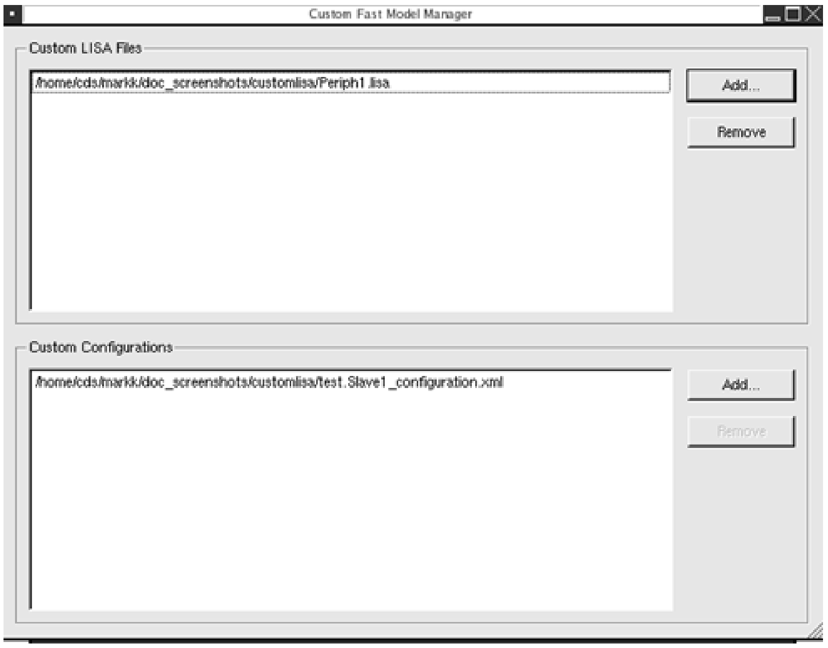
Generally, C++ code can be inserted inline in LISA code. For custom components to be imported into the Fast Models System Creator, however, there are some restrictions. Specifically, C++ code in the "resources" section of the file is limited to variable declarations. Place more complex code, such as class or function definitions, in the includes section, either directly or through a #include file.

A side effect of this is that the code is globally scoped instead of class-local, so the specific code and its uses may need to change. For example, a function defined in the "resources" section is a member function of the generated component class, while a function in the "includes" section is global.

6.4 Adding a Custom Component Mapping

You can save custom component mappings (involving default or custom LISA components) as described in “Saving Custom Configurations” on page 68. While mappings created in that manner are automatically saved to your preferences, you can add mappings created by others to the preferences manually.

To do this, click **Add** in the "Custom Configurations" panel and browse to the saved XML file for the custom mapping. This is typically in the same directory as the .MXP file used to create the mapping. After selecting the file, it is displayed in the table.



/home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

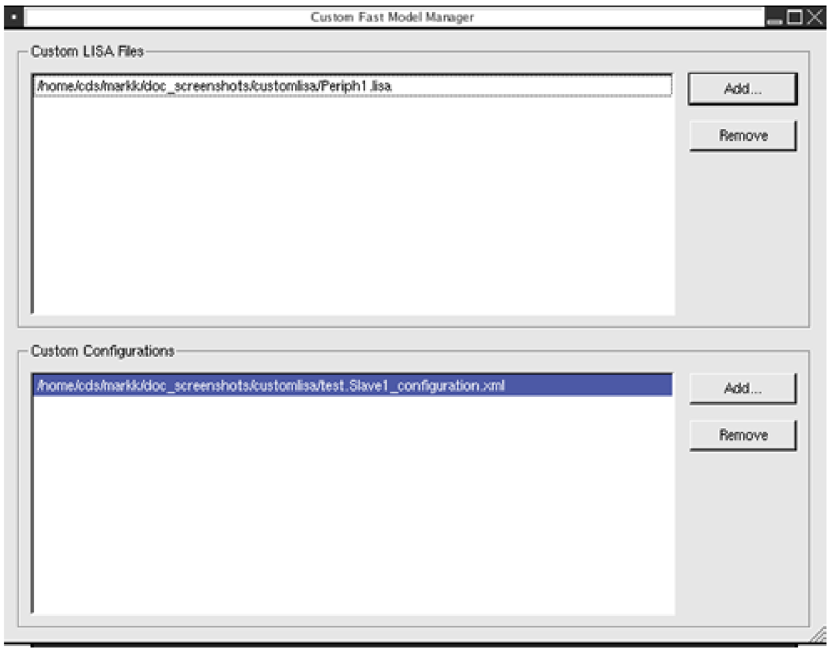
/home/cds/markk/doc_screenshots/customlisa/test.Slave1_configuration.xml

Figure 6-9 Custom component mapping added to Custom Fast Models Manager dialog

Note: Mappings are not applied immediately to matching components in the current Fast Models System Creator instance. You must close and re-invoke the dialog for the mapping to be applied.

6.5 Removing Custom Components and Configurations

To remove a saved custom component or component configuration from your preferences, select it from the list and click **Remove**.



/home/cds/markk/doc_screenshots/customlisa/Periph1.lisa

/home/cds/markk/doc_screenshots/customlisa/test.Slave1_configuration.xml

Figure 6-10 Custom configuration selected for removal

The removed configuration is still available and active for the current instance of the Fast Models System Creator tool. After you close the dialog, the removal is complete.

6.6 Saving Custom Configurations

After you create a custom component configuration, you can save it for future use by selecting the "Save configuration" checkbox (see Figure 4-7 on page 41) before closing the dialog. This adds the configuration to the SoC Designer preferences so that all instances of the Cycle Accurate component are automatically mapped to a Fast Models component.

In some cases, the custom configuration applies only to this particular instance of the Cycle Accurate component, not to all instances. To restrict the saved configuration to this instance, select the "Restrict to this instance" checkbox (see Figure 4-7 on page 41).

Appendix A

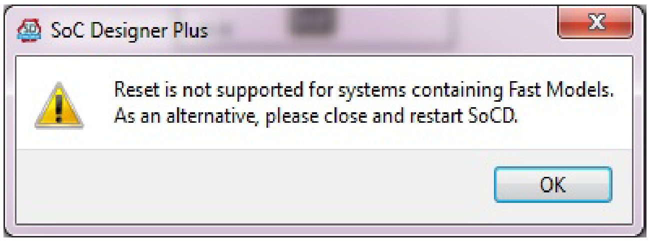
SoC Designer Behavior Changes Due to Multiple Instantiation Support

SoC Designer supports ARM SystemC Export with Multiple Instantiation (MI). This appendix describes the changes in SoC Designer behavior that you can expect to see based on the MI support.

For more information about Multiple Instantiation, refer to the *ARM Fast Models User Guide* (ARM DUI0835).

A.1 Hard/Soft Reset Not Supported

For systems containing any Fast Models component, Hard/Soft Reset is not supported. The alert shown in Figure A-1 will appear if a reset is attempted. This limitation applies to the SoC Designer GUI as well as MxScript-based invocation.



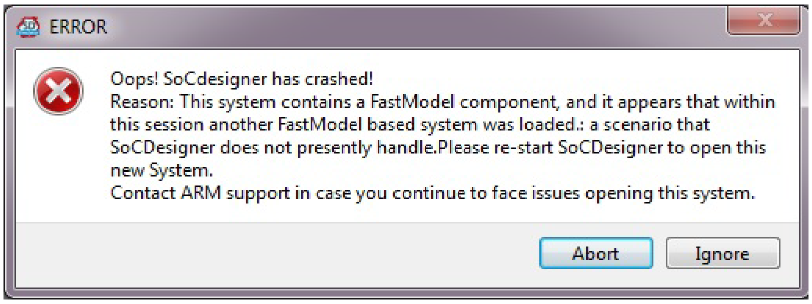
Reset is not supported for systems containing Fast Models.
As an alternative, please close and restart SoCD.

Figure A-1 Reset Error Message

The workaround is to close and restart SoC Designer.

A.2 Re-opening Fast Models-based Systems

When you bring up SoC Designer Simulator for a system containing any Fast Models component, you can not open the same or any other system containing Fast Models components within the same invocation. Attempting to do so causes a SoC Designer Simulator crash (see Figure A-2). This limitation does not affect SoC Designer Canvas.


ERROR

Oops! SoCdesigner has crashed!
Reason: This system contains a FastModel component, and it appears that within this session another FastModel based system was loaded.: a scenario that SoCDesigner does not presently handle.Please re-start SoCDesigner to open this new System.
Contact ARM support in case you continue to face issues opening this system.

Figure A-2 Simulator Error when loading a second Fast Model-based system

The workaround is to close and restart SoC Designer Simulator.

A.3 Changes in Timing of Simulations

The timing of certain actions in Fast Models-based systems may differ between this version of SoC Designer and previous versions. For example, you may notice differences in timestamps in Linux boot or the number of SoC Designer steps it takes to finish a Fast Models-based simulation.

A.4 Instruction Step Behavior Change in Multi-core Systems

In a multi-CPU system, when you do a single-step on one CPU, the other CPUs advance to their quantum end, and then do not advance until you issue a `run`. This happens because the CPU that is executing the instruction step pauses the SystemC time, while the other CPUs proceed to end of their immediate quantum and wait for SystemC time to resume.

A.5 Instruction Step Behavior Change on Cycle Count

Before SoC Designer Version 8.4, issuing an instruction-level single step from the Disassembly window incremented the cycle count (top right window in SoC Designer) by 1 or 2.

With SoC Designer Version 8.4 and later, the cycle count does not increase with the instruction step.

A.6 MxScript Change in Behavior of bpInfoLastHit()

In a multi-core system, it is now possible in SoC Designer for multiple CPUs to hit their breakpoints in a given run. The MxScript command `bpInfoLastHit` has been enhanced to report all the breakpoints hit for the last run, one after the other, in no specific order.

If existing MxScripts are written to wait for multiple breakpoints and take action based on breakpoint IDs, you may need to change those scripts to repeatedly call this API until it returns "-1".

Refer to the *MxScript v3.3 for Cycle Models Reference Manual* (ARM DUI1011) for more information.

A.7 Instruction Count Reporting

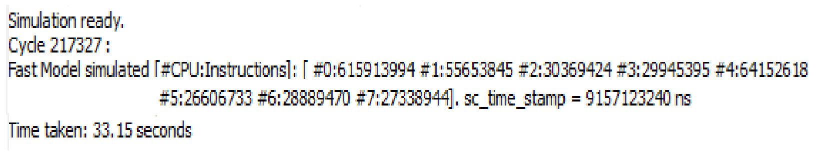
Every time SoC Designer Simulator halts by hitting a breakpoint or finishing an instruction step or SoC Designer cycle step, you will now see number of instructions executed on various CPUs during the last execution.

For example:

- Single Core — The log indicates Fast Model CPU Index 0 (there's only one CPU) - and the number of instructions executed:

```
Fast Model simulated [#CPU:Instructions]: [ #0:302565595].
sc_time_stamp = 6740791220 ns
```

- Multicore/Multi-cluster system (4x2) — The log indicates Fast Model CPU Index 0…7 and the number of instructions executed on each CPU.



```
Simulation ready.
Cycle 217327 :
Fast Model simulated [#CPU:Instructions]: [ #0:615913994 #1:55653845 #2:30369424 #3:29945395 #4:64152618
                    #5:26606733 #6:28889470 #7:27338944]. sc_time_stamp = 9157123240 ns
Time taken: 33.15 seconds
```

```
Indexing:
#0 signifies CPU0 of Cluster 0
#1 signifies CPU1 of Cluster 0
..
#3 signifies CPU3 of Cluster 0
#4 signifies CPU0 of Cluster 1
..
#7 signifies CPU3 of Cluster 1
```

A.8 Launching ModelDebugger for Fast Models CPUs

SoC Designer supports bringing up or attaching a Fast Models ModelDebugger for an ongoing simulation from within the SoC Designer GUI. Refer to "Launching ModelDebugger for Fast Models CPUs" on page 5-59 for instructions.