Mentor®
A Siemens Business

Questa® SIM Multi-Core Simulation User’s Guide

Software Version 2020.4

Table of Contents

List of Figures

List of Tables

Chapter 1
Using Questa SIM With Multiple Cores

Use the Questa SIM simulator with multi-core functionality to compile and simulate a design on multiple processors (cores). You can do this on a single computer that uses multiple CPUs or on a ring of single-CPU computers.

Configuration and Overview

Basic operation of multi-core simulation consists of configuring your operating environment, compiling and partitioning your design, and then running Questa SIM simulator commands that have arguments allowing you to customize the multi-core simulation and analyze results.

Linux

Note that multi-core simulation is supported for the Linux®[1]operating system only.

Configuration Requirement

Before running Questa SIM multi-core simulation, you need to set up your operating environment.

- You must have Python v2.3 (or later) installed on all machines.
- You must have the modeltech tree in the search path of your Linux operating system (OS).
- The full path to the modeltech tree must include the platform name, which is either linux or linux_x86_64.

  For example, use the following command to configure 32-bit Questa SIM multi-core to run on Linux:

  ```
  Set path = (<path_to_your_modeltech_tree>/linux $path)
  ```

After you run this command, your operating environment is set up to perform Questa SIM multi-core simulation as described in Running Multi-core Simulation on a Single Computer.

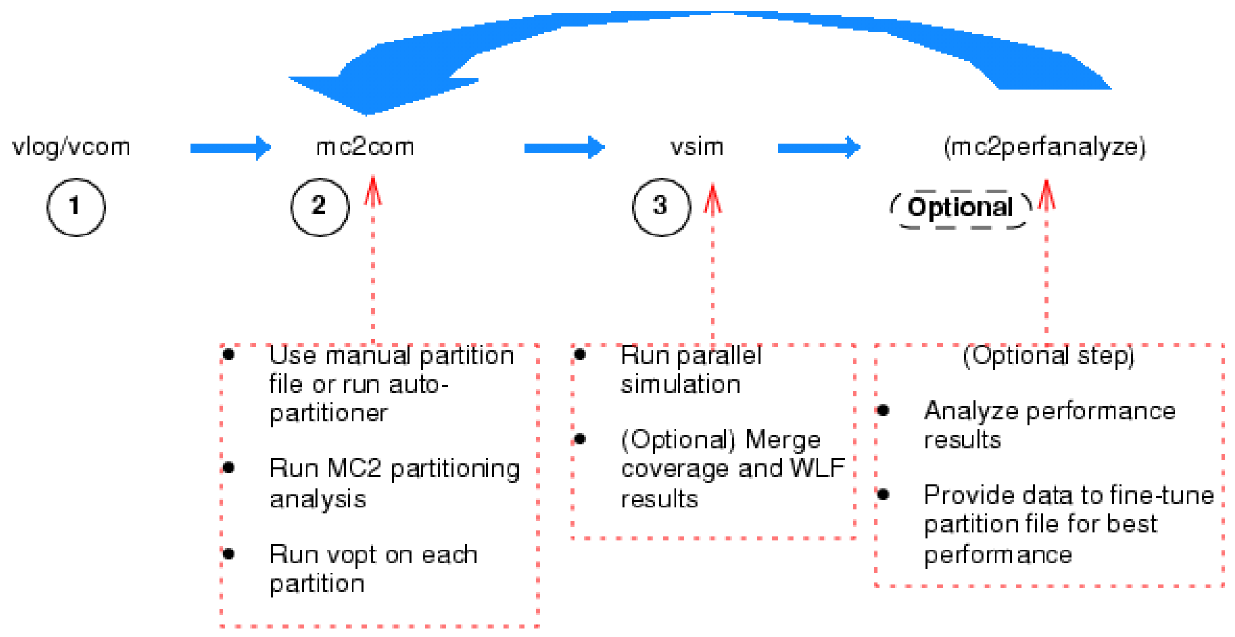
Basic Usage Flow for Multi-Core Simulation

The general procedure for running Questa SIM multi-core simulation consists of three basic steps, plus an optional step of analyzing simulation results.

Figure 1-1 shows a visual overview of this procedure.

1. Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

Figure 1-1. Commands Used for Multi-Core Simulation and Analysis


vlog/vcom
1
mc2com
2
vsim
3
(mc2perfanalyze)
Optional
• Use manual partition file or run auto-partitioner
• Run MC2 partitioning analysis
• Run vopt on each partition
• Run parallel simulation
• (Optional) Merge coverage and WLF results
(Optional step)
• Analyze performance results
• Provide data to fine-tune partition file for best performance

Running multi-core simulation consists of the following basic steps:

1. Compile your design using either the vlog or the vcom command (including arguments) as you would normally do prior to regular simulation where you use the vsim command. This is referred to as *unisim* (single-core simulation). For more information on using vlog or vcom, refer to "Compiling for Conventional Simulation."

2. Compile the design for multi-core simulation using the mc2com command, which consists of the following actions:

   - Automatic or Manual partitioning (Methods of Creating a Partition File).
   - Multi-core Partition Analysis (Analyzing a Partition).
   - Partition Optimization (Optimizing a Partition).

   You can use the mc2com command to perform any of these three actions together or individually, as needed.

3. Run multi-core simulation using the vsim command. This step will run the simulation in multi-core simulation mode. For more information on using vsim for multi-core simulation, refer to "Running Multi-core Simulation on a Single Computer."

   (Optional) After simulation has finished, perform a post-processing analysis using the mc2perfanalyze command. This analysis provides information on the performance of the multi-core simulation run.

For examples of using these steps to run Questa SIM multi-core simulation, refer to "Examples of Multi-core Simulation."

How to Perform Multi-Core Simulation

Running Questa SIM multi-core simulation consists of using three Questa SIM commands, plus an optional fourth command that you can use to analyze simulation results. Arguments for each command enable you to make modifications to the simulation procedure.

The tasks in this section describe how to use these multi-core simulation commands and arguments.

Compiling for Conventional Simulation

To begin multi-core simulation, compile your design with either the vcom or the vlog command. Invoke either command from within the Questa SIM GUI or from the command prompt of your operating system.

Note
For simplicity, this procedure assumes the use of a command shell.

Restrictions and Limitations

- Compiled libraries are major-version dependent. When moving between major versions, you have to update compiled libraries using the -refresh argument. This does not apply to minor versions (letter releases).
- All arguments to these commands are case-sensitive. For example: -WORK and -work are not equivalent.

Prerequisites

- You must have read and write access to the design that you want to simulate.
- You must have knowledge of which language (VHDL or SystemVerilog/Verilog) constitutes the top level of your design. This determines whether you use the vcom or vlog command.
- You must have access to the documentation on the arguments to the vlog or vcom command.

Tip

Refer to the entries for the vcom or vlog commands in the *Questa SIM Command ReferenceManual* for complete information on the arguments for these commands.

Procedure

1. (Optional) From the desktop of your operating system, open a command shell.
2. Run either the vcom or the vlog command, using the name of the file containing your top-level design as the <filename> argument. You can do this in either your command shell or in the Transcript window of Questa SIM.

Results

Your design is compiled into the current work library.

Examples

The following shows examples of using the compilation commands for VHDL and Verilog designs.

- For VHDL:

```
vcom toplevel.vhd
```

- For Verilog or SystemVerilog:

```
vlog toplevel.v
```

Multi-core Compilation: Partitions, Analysis, and Optimization

To compile for multi-core simulation, use the mc2com command to partition the design. You can also use arguments for this command to analyze and optimize the partition.

Tip
You can use mc2com command arguments to partition, analyze, or optimize simultaneously or individually, as needed.

The mc2com command performs the following actions while compiling the design for multi-core simulation:

- Automatic or Manual partitioning (Methods of Creating a Partition File)
- Multi-core Partition Analysis (Analyzing a Partition)
- Partition Optimization (Optimizing a Partition)

Methods of Creating a Partition File

The partition file is a text file that defines how the design is to be partitioned for the Questa SIM multi-core simulation run.

You can create a partition file in any of the following ways:

- Use the mc2com command on a design to perform multi-core simulation without specifying a partition file. This causes mc2com to perform auto-partitioning on your design. (Creating a Partition File With Auto-Partitioner)
- Use the mc2com command on a design to perform multi-core simulation flow specifying a modified auto-partition file or a manually created partition file. (Using a Partition File Without Running Auto-Partitioner)
- Run the auto-partitioner without multi-core compilation or vopt phases. Use this method when you want to experiment with different partitions. (Running Auto-Partitioner Only)
- Use the mc2com command with profiler information to generate a partition file automatically. (Using Auto-Partitioner With Profile Data)

Creating a Partition File With Auto-Partitioner

If you do not have a partition file, you can use the mc2com command to perform auto-partitioning, which creates a default partition file.

Tip

If you are using a manually created partition file or an existing auto-partition file, then you can skip this auto-partitioning step and start directly with multi-core compilation.

The auto-partitioner partitions the design at an instance boundary based on the cost/weights associated with the instances. In its default mode, the auto-partitioner provides a static estimation of the cost of individual instances. While the partitioning resulting from this estimation approach is satisfactory in designs where the static performance estimation closely corresponds to the dynamic behavior of the design, it might not be effective in generating good partitioning for other types of designs.

The Questa SIM profiler can collect information on both performance and memory utilization. The auto-partitioner can use this information from the profiler to make partitioning decisions based on performance alone, on memory alone, or on both performance and memory.

If there are not enough instances with significant weight in the design to create the requested number of load-balanced partitions, or if there are not enough instances that meet the criteria to put them on a partition boundary, auto-partitioner does not generate the requested number of partitions. In such cases, auto-partitioner automatically selects and generates the optimal number of load-balanced partitions.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- Invoke Questa SIM in GUI mode.

Procedure

1. Run the mc2com command using specified syntax and any arguments you want to apply for creating a partition. For example, the following command uses the -mc2numpart argument to automatically create three partitions:

   ```
   mc2com testbench -o top_opt -mc2numpart 3
   ```

2. Examine the progress of the partition, analysis, and optimization processes in the Transcript window.

Results

This creates a partition file, named *top_opt.part*, that has three partitions and is saved in the directory of the current project. The design is also optimized.

Examples

The following shows an example of the transcript displayed when running the mc2com -mc2numpart command (specifying the number of partitions).

```
Top level modules:
        testbench

Analyzing design...
-- Loading module testbench
-- Loading module mod1
-- Loading module mod2
-- Loading module bottom
-- Loading module mod3

Running Auto-partitioner...
-- Auto-partitioner finished creating file 'top_opt.part' with 3
partitions

Running MC2 Partitioning Analysis...
-- Cleaning up interface directory
-- Running analysis
-- Compiling interface files

Running MC2 Vopt Optimization...
-- Running Vopt for partition 'master'
-- Running Vopt for partition 'p1'
-- Running Vopt for partition 'p2'
-- Vopt for all partitions completed successfully
```

Using a Partition File Without Running Auto-Partitioner

Use the mc2com command to use a partition file of your choice, either a manually created partition file or an existing auto-partition file.

Tip
For information on partition files, refer to Partition Files.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- You have read access to an existing partition file.

Procedure

1. Run the mc2com command using the specified syntax and any arguments you want to apply for creating a partition.

For example, the following command uses the -mc2partfile argument to specify the file named *manual.part*:

mc2com testbench -o run_mp -mc2partfile manual.part

2. Examine the progress of the analysis and optimization in the Transcript window.

Results

The partition file is analyzed and used for optimizing the design.

Examples

Following is an example of the transcript displayed when running the mc2com -mc2partfile command (specifying a partition file).

```
Top level modules:
        testbench

Analyzing design...
-- Loading module testbench
-- Loading module mod1
-- Loading module mod2
-- Loading module bottom
-- Loading module mod3

Running MC2 Partitioning Analysis...
-- Cleaning up interface directory
-- Running analysis
-- Compiling interface files

Running MC2 Vopt Optimization...
-- Running Vopt for partition 'master'
-- Running Vopt for partition 'p1'
-- Running Vopt for partition 'p2'
-- Vopt for all partitions completed successfully
```

Running Auto-Partitioner Only

Optionally, you can run only the auto-partitioner phase within mc2com, without performing the multi-core simulation compilation or optimization. Use this flow when you want to experiment with different partitions.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).

Procedure

1. Run the mc2com command using specified syntax and any arguments you want to apply for creating a partition.

   mc2com testbench -o top_opt -mc2numpart 3

2. Examine the progress of the partition process in the Transcript window.

Results

The design is analyzed, and a partition file is created with three partitions. You can rerun this procedure with a different number of partitions specified, then compare the partition files.

Examples

Following is an example of the transcript displayed when running the mc2com command.

```
Top level modules:
        testbench

Analyzing design...
-- Loading module testbench
-- Loading module mod1mc2inst.wlfmc2inst.wlf
-- Loading module mod2
-- Loading module bottom
-- Loading module mod3

Running Auto-partitioner...
-- Auto-partitioner finished creating file 'top_opt.part' with 3
partitions
```

Using Auto-Partitioner With Profile Data

Auto-partitioning enables you to use profiler information to create a partition file.

Tip

For more information on theQuesta SIM profiler and profile data, refer to the Performance Profiling in the *Questa SIM User's Manual*.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- You must have valid profiler license feature in your Questa SIM license file.
- You must have write and read access to the profile database file used by the profile save and profile open commands. These commands capture profile data during a simulation session and store it for later review or passing to others for analysis.

Procedure

1. Run the Questa SIM profiler from either the command line or from the GUI. Table 1-1 provides a quick summary of both operations.

Note
The name *profile_perf.pdb* is used as a generic name for the profiler database file. You can use any legal filename that has the extension *.pdb*.

Table 1-1. Running the Profiler from the Command Line or from the GUI

If you want to...	GUI Operation
Run the profiler from the GUI.	1. Choose **Simulate** > **Start Simulation** 2. In the Start Simulation dialog box, click the **Others** tab. 3. Under the Profiler region, select Enable memory profiling (if you want memory profiling). This enables memory profiling during elaboration. 4. Click the **Design** tab, select your compiled design, and click **OK**. 5. After the design loading is complete, choose either or both of the following from the main menu: • **Tools** > **Profile** > **Performance** • **Tools** > **Profile** > **Memory** (This is already checked if you selected Enable memory profiling in Step 3, above.) 6. Choose **Simulate** > **Run** > **Run 100** (or any other Run option): 7. At the prompt ("Are you sure you want to finish"), click **No**. 8. Save the profile data by entering the following in the command window: `profile save profile_perf.pdb` 9. Choose **File** > **Quit**
Run the profiler from the command line.	Use the vsim command as you normally would, then running the profile commands with the -p argument For example, enable and save data from performance profiler: `vsim -c top -do "profile on -p; onfinish stop; run -a; profile save profile_perf.pdb; quit"`

2. Run the mc2com command with the -mc2useprofile argument to create a partition file using data resulting from either performance or memory profiling (saved in *profile_perf.pdb*).

   - Performance profiler —

     ```
     mc2com -mc2useprofile=perf profile_perf.pdb -mc2numpart 8 top -o part8
     ```

   - Memory profiler —

     ```
     mc2com -mc2useprofile=mem profile_mem.pdb -mc2numpart 8 top -o part8
     ```

Results

- The vsim command with profiling enabled, followed by the profile save command, captures profile data from the simulation, and writes it to the specified profile database file, *profile_perf.pdb*.
- The mc2com -mc2useprofile=perf command reads performance profile data from the specified profile database file, *profile_perf.pdb*, and uses it to create a partition file with eight partitions.
- The mc2com -mc2useprofile=mem command reads memory profile data from the specified profile database file, *profile_perf.pdb*, and uses it to create a partition file with eight partitions.

Tcl Variables for Partition Number and Name

Use Tcl variables to identify the name or number of a partition that you create for multi-core simulation. You can then use the values that Questa SIM returns for the name and number variables in commands for a DO file (macro).

Use the following Tcl variables inside a macro (DO file) for multi-core simulation:

- get_mc2_partition_id — Returns partition number assigned to each partition. For the master partition, returns a value of 0.
- get_mc2_partition_name — Returns the partition name.

Examples

```
set myvar [get_mc2_partition_id]
set myvar1 [get_mc2_partition_name]

coverage save -onexit partition_$myvar.ucdb
// Save file as partition_0.ucdb, partition_1.ucdb, ...

coverage save -onexit partition_$myvar1.ucdb
// Save file as partition_master.ucdb, partition_p1.ucdb, ...
```

Analyzing a Partition

You can run an analysis on an existing partition without first performing auto-partitioning or optimization. Use the mc2com command and specify an existing partition file.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- You have access to an existing partition file for the design.

Procedure

Run the mc2com command, using the specified syntax and any arguments you want to apply for creating a partition.

Results

The partition file is analyzed without simulation or optimization.

Examples

- The following example specifies a partition file with the name *manual.part*:

  mc2com testbench -o top_opt -mc2partfile manual.part

- The following example shows a transcript displayed when running the mc2com -mc2novopt command.

```
Top level modules:
        testbench

Analyzing design...
-- Loading module testbench
-- Loading module mod1
-- Loading module mod2
-- Loading module bottom
-- Loading module mod3

Running MC2 Partitioning Analysis...
-- Cleaning up interface directory
-- Running analysis
-- Compiling interface files
```

Optimizing a Partition

You can optimize an existing partition without performing auto-partitioning or analysis on it. Use the mc2com command and specify an existing partition file.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- You have access to an existing partition file for the design.

Procedure

Run the mc2com command using specified syntax and any arguments you want to apply for creating a partition.

Results

The partition file is analyzed without simulation or analysis.

Examples

- The following command specifies a partition file with the name manual.part:

  mc2com testbench -o top_opt -mc2partfile manual.part

- The following shows an example of the transcript displayed when running the mc2com command.

```
Top level modules:
        testbench

Running MC2 Vopt Optimization...
-- Running Vopt for partition 'master'
-- Running Vopt for partition 'p1'
-- Running Vopt for partition 'p2'
-- Running Vopt for partition 'p3'
-- Vopt for all partitions completed successfully
```

How Optimization (vopt) Creates Parallel Processes

The vopt command, when used to optimize a partition, creates a parallel process for code generation.

You can specify the maximum number of total parallel code generation processes for all combined partitions by using the -j argument of the mc2com command, which will evenly distributes the code generation processes.

Consider an example where you have three or more code generation processes to run for each of four partitions. Use the -j argument of the mc2com command to specify a total of 12 processes, which allows up to 3 code generation processes for each vopt process of each partition.

```
mc2com testbench -o top_opt -mc2partfile manual.part -j 12

Model Technology ModelSim SE mc2com DEV Compiler 2003.05 Jul 12 2011

Top level modules:
        testbench

Skipping MC2 Partitioning Analysis...

Running MC2 Vopt Optimization...
-- Running Vopt for partition 'master'
-- Running Vopt for partition 'p1'
-- Running Vopt for partition 'p2'
-- Running Vopt for partition 'p3'
-- Vopt for all partitions completed successfully
```

Running Multi-core Simulation on a Single Computer

Once you have fully completed multi-core compilation on the design, it is ready for multi-core simulation. You run the simulation by specifying the -mc2 argument of the vsim command (along with any other necessary arguments of the vsim command that are specific to multi-core simulation).

Tip

By default, multi-core simulation is run on multiple cores of single machine. However, you can run simulation on multiple machines connected over a network using the vsim -mc2network command — refer to "Running Multi-computer Simulation."

Notes on the vsim command for multi-core simulation:

- The arguments of the vsim command that are specific to multi-core simulation are described in the vsim command reference section in Chapter 3 of this manual. Refer to the Questa SIM *Reference Manual* for complete documentation of the vsim command.
- By default, each partition creates its own transcript file with the name *<partition_name>.log*. You can override this default transcript file name by using the -mc2vsimargs argument for each partition. In addition, a transcript file containing output from all partitions combined is generated, following the normal conventions for vsim transcript files. The default name for a vsim transcript file is *transcript*. You can override the default transcript filename using vsim -l.
- If you do not specify the -do argument, then by default, the following syntax is added for all partitions:

  ```
  -do 'run -a; quit -f'
  ```

  You can override the default for all partitions by using the -do argument. You can override the default for a specific partition by using -mc2vsimargs argument.
- When you partition a design, partitions can end up with different minimum time resolutions, resulting in a time resolution mismatch error after elaboration. Time resolutions for all partitions need to match in order to run simulation correctly. You can prevent the mismatch by specifying the -t option with the overall design's time resolution at the vsim command line.

Restrictions and Limitations

- If you override the -do argument for individual partitions, make sure that all of the partitions are executing the same run commands. Otherwise, problems may occur with the simulation.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- Compile the design with the mc2com command (Multi-core Compilation: Partitions, Analysis, and Optimization) or you have access to a partition file for the design.

Procedure

Run the vsim command using the -mc2 argument and any additional arguments you want to apply for simulation.

Results

- The simulator performs multi-core simulation on the top-level design named top_opt.
- Specifying -mc2synccntl=verilog enables the synch and nba sync controls for the sync points of the partition file.

Examples

- The following vsim command uses the -mc2synccntl argument to override the default sync points of the partition by specifying the verilog alias for the Verilog partition boundary.

  ```
  vsim -mc2 top_opt -do run.do -mc2synccntl=verilog
  ```

- The following shows an example of the transcript displayed when running the vsim -mc2 command.

```
Reading /u/dvtbata/rkjain/mainline/modeltech/tcl/vsim/pref.tcl

# DEV

# Python 2.3.4
# Reading /u/dvtbata/rkjain/mainline/modeltech/tcl/vsim/pref.tcl
#
# //
# [MC2-STAT] Partition "p2", Proc Id = 1064
# vsim +nowarnTFMPC +nowarnTSCALE -do run.do -l p2.log -lib /u/
dvtbata/rkjain/mainline-ws/tests_mp/base/param/param19/work/_mc2/
p2_work -mc2partfile top_opt.part -c -wlf p2.wlf -wlfnoopt p2
# [MC2-STAT] Partition "p1", Proc Id = 1065
# vsim +nowarnTFMPC +nowarnTSCALE -do run.do -l p1.log -lib /u/
dvtbata/rkjain/mainline-ws/tests_mp/base/param/param19/work/_mc2/
p1_work -mc2partfile top_opt.part -c -wlf p1.wlf -wlfnoopt p1
# [MC2-STAT] Partition "master", Proc Id = 1063
# vsim +nowarnTFMPC +nowarnTSCALE -do run.do -l master.log -lib /u/
dvtbata/rkjain/mainline-ws/tests_mp/base/param/param19/work/_mc2/
master_work -mc2partfile top_opt.part -c -wlf master.wlf -wlfnoopt
master
# Loading ./work.testbench(fast)
# Loading ./work.mod1(fast)
# Loading ./work.mod2(fast)
# Loading sv_std.std
# Loading work.p1_master__intf_1(fast)
# Loading work.p1__intf_3(fast)
# Loading work.p1__intf_8(fast)
# do run.do
# Loading ./work.testbench(fast)
# Loading ./work.mod2(fast)
# Loading ./work.mod3(fast)
# Loading sv_std.std
# Loading work.master_p1__intf_0(fast)
# Loading work.master__intf_2(fast)
# Loading work.master_p2__intf_4(fast)
# Loading work.master__intf_7(fast)
# do run.do
# Loading ./work.testbench(fast)
# Loading ./work.mod1(fast)
# Loading ./work.mod3(fast)
# Loading sv_std.std
# Loading work.p2_master__intf_5(fast)
# Loading work.p2__intf_6(fast)
# Loading work.p2__intf_9(fast)
# ** Note: (vsim-8840) The following MC2 sync controls will be used
verilog
# do run.do
#          0 testbench.mod1_inst p2=          3
#          5 testbench p1=          2
#          5 testbench.mod1_inst p2=          3
#         10 testbench p1=          2
#         10 testbench.mod1_inst p2=          3
#          0 testbench.mod1_inst.mod3_inst.inst3 p4=          7
#          0 testbench.mod1_inst.mod3_inst.inst2 p4=          6
#          0 testbench.mod1_inst.mod3_inst p3=          5
#          5 testbench.mod1_inst.mod3_inst p3=          5
#          6 testbench.mod1_inst.mod3_inst.inst2 p4=          6
#          7 testbench.mod1_inst.mod3_inst.inst3 p4=          7
#         10 testbench.mod1_inst.mod3_inst p3=          5
#          0 testbench.mod1_inst.mod2_inst.inst1 p4=        100
#          0 testbench.mod1_inst.mod2_inst p3=         20
#          5 testbench.mod1_inst.mod2_inst.inst1 p4=        100
#          5 testbench.mod1_inst.mod2_inst p3=         20
#         10 testbench.mod1_inst.mod2_inst.inst1 p4=        100
#         10 testbench.mod1_inst.mod2_inst p3=         20
#
# [MC2-STAT] Partition     p2 (2)=> Value Triggers: 0, User Sync
Events: 0, Total Sync Events: 15, Inout Sync Events: 0, Dataless
Sync Events: 6, Idle Sync Events: 0, Max Sync Reloop: 2, Max Data
Reloop: 1, Events: 13, Processes: 33, Suspend Opt : 0
#
# [MC2-STAT] Partition     p1 (1)=> Value Triggers: 96, User Sync
Events: 0, Total Sync Events: 15, Inout Sync Events: 0, Dataless
Sync Events: 6, Idle Sync Events: 0, Max Sync Reloop: 2, Max Data
Reloop: 1, Events: 13, Processes: 23, Suspend Opt : 0
#
# [MC2-STAT] Partition master (0)=> Value Triggers: 576, User Sync
Events: 0, Total Sync Events: 15, Inout Sync Events: 0, DatalessSync
Events: 6, Idle Sync Events: 0, Max Sync Reloop: 2, Max Data Reloop:
1, Events: 18, Processes: 43, Suspend Opt : 0
#
# real   0m2.616s
# user   0m0.056s
# sys    0m0.014s
```

Running Multi-computer Simulation

By default, multi-core simulation is run on multiple cores of a single machine. However, you can run simulation on multiple machines connected over a network by using the -mc2network argument of vsim to specify a file containing a list of the hosts on which to run the simulation.

Notes on the vsim command for multi-computer simulation:

- The host file contains the list of hosts on which to run the executable, and optionally specifies how many processes can run on each host. The maximum number of processes should not exceed the number of available cores on that machine — you may want to specify fewer processes than the available cores to achieve a particular load balance between the different hosts.
- Also, you should consider the memory capacity of each host, and the expected memory requirements of each multi-core simulation partition. If you do not specify a number of processes for each host, multi-core simulation determines a maximum according to how many cores are present on each host.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).

- Compile the design with the mc2com command (Multi-core Compilation: Partitions, Analysis, and Optimization) or you have access to a partition file for the design.
- A host file containing a list of hosts on which to run the simulation, along with the number of processes that can run on each host.
- Network access and valid licensing for the hosts listed in the host file.

Procedure

Run the vsim command using the -mc2network argument and any additional arguments you want to apply for simulation.

Results

Questa SIM runs a multi-core simulation on the computers specified in the hostlist_mc.txt host file.

Examples

A vsim command with the -mc2network argument:

```
vsim -mc2 top_opt -do run.do -mc2network hostlist_mc.txt
```

Contents of the *hostlist_mc.txt* host file:

```
host1:1 # Run 1 process on host1
host2:4 # Run 4 processes on host2
host3:2 # Run 2 processes on host3
host4:1 # Run 1 process on host4
```

Related Topics

Running Multi-core Simulation on a Gate-level Design

Running Multi-core Simulation on a Gate-level Design

Multi-core simulation supports running GLS designs with SDF annotation for Verilog/ SystemVerilog designs. Delay annotations that lie within a single partition are fully supported.

Restrictions and Limitations

To implement multi-source delay annotation across partitions, specify the +multisource_int_delays argument with the vsim command. This argument has the following limitations:

- It works only for Verilog or SystemVerilog cells (this does not apply to VITAL).
- The multi-core simulator does not support partition-specific SDF files.

Prerequisites

- Compile the design with the vcom or vlog command (Compiling for Conventional Simulation).
- Compile the design with the mc2com command (Multi-core Compilation: Partitions, Analysis, and Optimization) or you have access to a partition file for the design.

Procedure

Run the vsim command for multi-core simulation with the +multisource_int_delays argument.

Results

Multi-source delay annotation is applied across the current partitions.

Examples

- The vsim command with +multisource_int_delays argument:

  ```
  vsim -mc2 top +multisource_int_delays -sdfmin /top=test.sdf  -mc2vsimargs=master
  "-do master.do" -mc2vsimargs=p1 "-do p1.do"
  ```

- The following usage is invalid because the multi-core simulator does not support partition-specific SDF files (refer to the Restrictions and Limitations above):

  ```
  vsim -mc2 top +multisource_int_delays  -mc2vsimargs=master "-sdfmin /top=test.sdf
  -do master.do"  -mc2vsimargs=p1 "-sdfmin /top=test.sdf -do p1.do"
  ```

- Note that you can still provide partition-specific SDF files without using the +multisource_int_delays argument. So the following vsim command is valid:

  ```
  vsim -mc2 top  -mc2vsimargs=master "-sdfmin /top=test.sdf -do master.do"  -
  mc2vsimargs=p1 "-sdfmin /top=test.sdf -do p1.do"
  ```

Analyzing Post-Simulation Results

After you have performed multi-core simulation on your design, use the mc2perfanalyze command to examine the results, which consist of several different types of formatted text reports from the raw data collected during the multi-core simulation (using -mc2savestat) and unisim run (using -mc2collectinfo).

Tip

For more information on the mc2perfanalyze command and its arguments, refer to the mc2perfanalyze command reference page.

The mc2perfanalyze command produces several types of reports from the raw data in the SQL database that is created by multi-core simulation. Three main factors affect multi-core simulation performance:

- Load balancing

- Concurrency
- Communication between partitions

The volume of report data can be quite large. The steps given below demonstrate how to evaluate these reports and identify performance issues.

Prerequisites

- Simulate the design using multi-core simulation with the mc2com command (Running Multi-core Simulation on a Single Computer or Running Multi-computer Simulation)

Procedure

1. Enter the following command:

   mc2perfanalyze mc2data.mdb -analyze

   The mc2perfanalyze command produces a summary report that shows any anomalies regarding the three factors.

2. Use the -info argument to obtain partition connectivity information:

   mc2perfanalyze mc2data.mdb -info

   The resulting output is formatted in a table.

3. Use the -time argument to examine load balancing:

   mc2perfanalyze mc2data.mdb -time

   The -time argument provides a comparison table showing how much time each partition spent executing different tasks, such as active simulation and communication.

4. Use the -event argument to produce a similar table that compares the partitions against different counters, including the Processes count (which is the number of processes executed by each partition):

   mc2perfanalyze mc2data.mdb -event

   Comparing the Processes counters of partitions can also reveal load balancing, but may not be as precise as comparing simulation times.

5. Use the -time argument again to examine concurrency:

   mc2perfanalyze mc2data.mdb -time

   Compare the Idle Time of each of the partitions; if the idle times are significant then it indicates a lack of concurrency between the partitions. Partitions with high No Comm. Sync Events/Idle Sync. Events in the table generated using the -event argument (Step 4) are the individually active partitions.

6. Use the -pattern argument to obtain additional information about the concurrent triggering pattern of the instances at the partition boundary:

   mc2perfanalyze mc2data.mdb -pattern

This is useful for fine-tuning partitioning for higher concurrency.

7. The communication between partitions involves data and data-less communication. Use the -info argument again to obtain more details on data communication:

   mc2perfanalyze mc2data.mdb -info

   This reveals the number of ports between the partitions. The number of ports between partitions is a very rough indication of the data communication between partitions.

8. Use the -data argument to report the amount of data that is exchanged between different pairs of partitions:

   mc2perfanalyze mc2data.mdb -data

   One of the important fields in the data report is the Count column, which shows the number of synchronizations between partitions that involved actual transfer of data. A high Count number indicates heavy communication between partitions. Data-less communication occurs at delta and time advances.

9. Use the -event argument again when there are two partitions communicating data while the other partitions are in data-less communication:

   mc2perfanalyze mc2data.mdb -event

   A varying percentage of data-less syncs may indicate an uneven distribution of load.

Examples of Multi-core Simulation

This section provides two examples of files and commands used for running multi-core simulation—a normally synchronized design and a user-synchronized design.

Each example consists of the following:

- Verilog design files
- Partition file
- Commands for compiling and simulating the design as multi-core simulation

Normally Synchronized Example

This example demonstrates how to specify, partition, and simulate a normally synchronized design.

Design Files

top.v

```
/* Design top */
module top;
   reg[0:31]   val1, val2;
   integer     i;

   wire[31:0]  res1, res2, res3, res4;
   reg[127:0] sum = 0;
   reg last = 0;

   reg   clk;
   initial begin
      #20  clk = 0;
      forever
       #50 clk = ~clk;
   end

   initial begin
      //Seed the random number generator with some constant
      #10 val1 = $random(511);
      val2 = $random();
      for(i = 0; i < 60000; i = i + 1) begin
         #200 val1 = $random();
         val2 = $random();
      end
      last = 1;
      $display ($stime,,, "%m Final Sum: %x", sum);
      #400 $finish;
   end

   child #(1) c1(clk, val1, val2, res1, last);
   child #(2) c2(clk, val1, val2, res2, last);
   child #(3) c3(clk, val1, val2, res3, last);
   child #(4) c4(clk, val1, val2, res4, last);

   always @(res1)
      sum = sum+res1;

   always @(res2)
      sum = sum+res2;

   always @(res3)
      sum = sum+res3;

   always @(res4)
      sum = sum+res4;
endmodule
```

child.v

```
module child(input wire clk
            ,input[0:31] in1
            ,input[0:31] in2
            ,output reg [0:31] out
            ,input last);

parameter op = 0;

reg [127:0] sum = 0;
reg [0:31] temp;

generate if (op == 1)
   always @(posedge clk)
      temp = in1 & in2;
else if (op == 2)
   always @(posedge clk)
      temp = in1 | in2;
else if (op == 3)
   always @(posedge clk)
      temp = in1 ^ in2;
else if (op == 4)
   always @(posedge clk)
      temp = in1 + in2;
endgenerate

always @(temp) begin
   out = temp;
   sum = sum + temp;
end

always @last begin
   $display ($stime,,, "%m Final Sum: %x", sum);
end

endmodule
```

Partition File

top.part

```
partition master {
   mod_inst = top;
}
partition p1 {
   mod_inst = top.c1;
}
partition p2 {
   mod_inst = top.c2;
}
partition p3 {
   mod_inst = top.c3;
}
partition p4 {
   mod_inst = top.c4;
}
```

Commands for Compiling and Simulating

```
vlib work

vlog top.v child.v

mc2com -mc2partfile top.part top -o fast_top

vsim -mc2 fast_top -mc2synccntl=verilog
```

User-synchronized Example

This example demonstrates how to specify, partition, and simulate a user-synchronized design.

Design Files

top.v

```
/* Design top */
module top;
   reg[0:31]   val1, val2;
   integer     i;

   wire[31:0]  res1, res2, res3, res4;
   reg[127:0] sum = 0;
   reg last = 0;

   reg   clk;
   initial begin
      #20  clk = 0;
      forever
         #50 clk = ~clk;
   end

   `include "user_sync.v"

   initial begin
      //Seed the random number generator with some constant
      #10 val1 = $random(511);
      val2 = $random();
      for(i = 0; i < 60000; i = i + 1) begin
         #200 val1 = $random();
         val2 = $random();
      end
      last = 1;
      $display ($stime,,, "%m Final Sum: %x", sum);
      #400 $finish;
   end

   child #(1) c1(clk, val1, val2, res1, last);
   child #(2) c2(clk, val1, val2, res2, last);
   child #(3) c3(clk, val1, val2, res3, last);
   child #(4) c4(clk, val1, val2, res4, last);

   always @(res1)
      sum = sum+res1;

   always @(res2)
      sum = sum+res2;

   always @(res3)
      sum = sum+res3;

   always @(res4)
      sum = sum+res4;
endmodule
```

child.v

```
module child(input wire clk
            ,input[0:31] in1
            ,input[0:31] in2
            ,output reg [0:31] out
            ,input last);

parameter op = 0;
`include "user_sync.v"

reg [127:0] sum = 0;
reg [0:31] temp;

generate if (op == 1)
   always @(posedge clk)
      temp = in1 & in2;
else if (op == 2)
   always @(posedge clk)
      temp = in1 | in2;
else if (op == 3)
   always @(posedge clk)
      temp = in1 ^ in2;
else if (op == 4)
   always @(posedge clk)
      temp = in1 + in2;
endgenerate

always @(temp) begin
   out = temp;
   sum = sum + temp;
end

always @last begin
   $display ($stime,,, "%m Final Sum: %x", sum);
end

endmodule
```

user_sync.v

```
/* Produces common user-defined sync-events for all partitions. */

event mc2_event;
integer y;
/* Generate sync event for every input value change */
initial begin
   #10  ->mc2_event;
   for( y = 0 ; y < 60000; y = y + 1 ) begin
       #200 ->mc2_event;
   end
end
/* Generate sync event for every output value change */
initial begin
   #70  ;
   forever
      begin
      #5 ->mc2_event;
       #95;
   end
end
```

Partition File

top.part

```
partition master {
   mod_inst = top;
   sync_event = top.mc2_event;
}
partition p1 {
   mod_inst = top.c1;
   sync_event = top.c1.mc2_event;
}
partition p2 {
   mod_inst = top.c2;
   sync_event = top.c2.mc2_event;
}
partition p3 {
   mod_inst = top.c3;
   sync_event = top.c3.mc2_event;
}
partition p4 {
   mod_inst = top.c4;
   sync_event = top.c4.mc2_event;
}
```

Commands for Compiling and Simulating

```
vlib work

vlog top.v child.v

mc2com -mc2partfile top.part top -o fast_top

vsim -mc2 fast_top -mc2synccntl=verilog
```

Globally Shared Memory Objects

You can use shared memory objects in multi-core simulation. By default, multi-core simulation allows implementation of memory as globally shared objects, unless you explicitly instruct the mc2com partitioner to disable this capability.

Usage Notes

In general, a globally shared object is a multi-dimensional HDL object consisting of a register and a size allocation.

The declaration of a memory object is of the form:

```
reg[0:31] MEM[0:32767] ;
```

which you must access directly by using hierarchical references. For example:

```
top.MEM
```

where the references (read or write) occur in multiple partitions.

To use globally shared objects, make sure that they meet the following criteria:

- They must be contention-free — accessing the shared object is done in a contention-free manner in the same time step.
- They must be sequentially neutral — accessing the shared object does not depend on the access sequences from other partitions in the same time step.
- They must allow exclusive access — accessing the “word address” of the form:

  ```
  MEM[ A ][ 21:40 ] = expr1 ; // partition #1
  expr2 = MEM[ A ][ 41:60 ] ; // partition #2
  ```

  at the same time step can produce unexpected results. Even though the accesses look mutually exclusive, they are not word exclusive. This restriction applies only when one or more accesses is a memory write.
- They must allow event triggering — the object cannot be used as trigger if the writer of the shared memory object resides in a different partition, an event trigger of the following form does not work:

  ```
  @top.MEM
  ```

Disabling Globally Shared Memory Objects

By default, multi-core simulation allows implementation of memory as globally shared objects. You can disable globally shared memory by using the -mc2noglobalmem argument of the mc2com command.

Procedure

Run the mc2com command for multi-core simulation with the -mc2noglobalmem argument, which disables global memory.

Results

When you specify the -mc2noglobalmem argument, the auto-partitioner creates partitions that avoid cross-partition references to memory arrays. If you specify this argument together with a partition file (that is, no auto-partitioning), the mc2com command reports an error if there are any cross-partition memory references.

Examples

The mc2com command with the -mc2noglobalmem argument:

```
mc2com top -mc2numpart 3 -mc2noglobalmem -o top_opt
```

Merging WLF and UCDB Files of a Partition

By default, each partition produces individual files for WLF (<*partition_name*>*.wlf*) and UCDB (<*partition_name*>*.ucdb*). Each of these files contains data specific to the hierarchy of its partition.

The vsim command has two arguments you can use to automatically merge individual partition files to generate design-wide data files:

- -mc2mergewlf (WLF files) — merges WLF files from all the partitions and creates a single design-wide WLF file.
- -mc2mergeucdb (UCDB files) — saves a UCDB file for each partition, merging the UCDB files from all the partitions and creating a single design-wide UCDB data file.

Manually Merging WLF and UCDB Files

As an alternative to using vsim -mc2mergewlf or vsim -mc2mergeucdb, you can use other Questa SIM commands to manually merge WLF or UCDB files for all partitions.

Procedure

Refer to Table 1-2 for the commands you can use for merging either WLF or UCDB files.

Table 1-2. Merging WFL or UCDB Files

If you want to...	Do the following:
Merge WLF files	Use the wlfman merge command with the following syntax: **wlfman merge -mark -opt -o <filename> ** **<partition_name1>.wlf ** **<partition_name2>.wlf ** **<optional_args>'**

Table 1-2. Merging WFL or UCDB Files (cont.)

If you want to...	Do the following:
Merge UCDB files.	Use the coverage save and vcover merge commands as follows: 1. Run the coverage save command for each partition at the end of simulation: **coverage save -onexit <partition_name>.ucdb** 2. After simulation completes, use the vcover merge command to merge the files: **vcover merge -combine <filename> ** **<partition_name1>.ucdb ** **<partition_name2>.ucdb ** **<optional_args>**

Debugging Standalone Partitions with VCD Flow

You can debug and run a standalone partition without communicating with (or being dependent on) other partitions. You can do this by recording Value Change Dump (VCD) values for a given partition.

VCD debugging consists of a fully automated two-step flow:

- Run full multi-core simulation and record VCD values. (VCD Recording).
- Select the partition you want to run and re-simulate it using earlier VCD recorded values. (VCD Resimulate)

Restrictions and Limitations

The VCD debugging system has the following limitations:

- Reg type ports at a boundary are not supported.
- Bit-select and part-select ports at a boundary are not supported.
- Complex SystemVerilog and VHDL types (except std_logic) are not supported.
- Objects (hrefs) crossing a partition boundary is not supported.
- Designs with user-defined events are not supported.

Prerequisites

- To be able to run a standalone partition, the simulator needs to identify which values are arriving at a boundary from other partitions and at what time. Once the simulator has this information, simulation can be run on a standalone partition without needing to run other partitions at the same time.

Procedure

1. VCD Record — While running full multi-core simulation, run the vsim command with the -mc2vcddump argument to automatically append required VCD commands in each partition to dump correct VCD values:

   vsim -mc2vcddump

   After the simulation is finished, various VCD files are generated for each boundary instance.

2. VCD Resimulate — After recording VCD values from multi-core simulation, you run single partition in standalone mode. You need to pick a partition and run vsim, using the -mc2sal argument. The -mc2sal argument automatically turns off compilation and full flow multi-core simulation and runs the simulation for only the given partition. The -mc2sal argument also automatically appends required VCD commands to the vsim command, and changes the default values for -l and -wlf options (notice the suffix _sal).

   vsim -mc2sal

Examples

The following examples show how to use vsim command options for VCD debugging.

- Run multi-core simulation in standalone mode for partition p1:

  vsim -mc2 <vsim args> -mc2vcddump

  vsim -mc2 <vsim args> -mc2sal p1

- Apply checkpoint-restore with VCD standalone flow:

  vsim -mc2 <vsim args> -mc2sal p2 -mc2vsimargs=p2 "-do ’run 100; checkpoint p2.chkpt; run -a; quit -f ’"

  vsim -mc2 <vsim args> -mc2sal p2 -mc2vsimargs=p2 "-l results/p2_sal1.log -restore p2.chkpt"

Debugging Simulation Mismatches With Unicore Flow

Use the unicore flow to do a self-test between multi-core simulation runs and unisim runs for debugging.

Comparing the two runs can indicate whether multi-core simulation and unisim simulations produce different results. For this optional mode of multi-core simulation, use the mc2com -mc2unicore command to run the entire design as a partition, in addition to the other multi-core simulation partitions of the design.

For example, if you have partitioned the design into three partitions, specifying a unicore mode would actually run four partitions. The multi-core simulation operation would run as it is (with three partitions), but the entire design would also be loaded as another partition (that is, a fourth partition). This fourth partition is referred to as the unicore partition.

Unicore Flow Usage Notes

The conventional multi-core simulation partitions communicate normally and move forward with simulation, while the unicore partition is independently running the full design (as the last partition), but it is run in time lockstep with other multi-core simulation partitions.

The unicore partition communicates with all multi-core simulation partitions to check on time lockstep and multi-core simulation partition boundary values.

- The word “unicore” is a reserved word—you cannot use it for a partition name.
- General arguments for mc2com and vsim commands that are passed to all multi-core partitions are also passed to the unicore partition. You can apply arguments specific to the unicore partition by using the -mc2voptargs=unicore argument for mc2com, or -mc2vsimargs=unicore argument for vsim.
- You can specify vsim -mc2nozerochk as a runtime option to avoid getting errors for value mismatches at time 0 (Questa SIM converts errors to warnings).

Debugging the Unicore Flow

Unicore debugging enables you to perform error checking between multi-core simulation and unicore modes during simulation, immediately identifying wherever different behavior occurs between modes.

The unicore flow performs two kinds of error checking to ensure that unicore and multi-core simulation are in same state at the end of each time unit:

- After resolving the next-time delta for the multi-core simulation system, the simulation checks that this delta matches with the next-time delta for unicore. Any difference found indicates that the multi-core simulation and the unicore simulation are not progressing in the same state. This difference may or may not be significant, based on how and when it occurs.
- At the end of each time unit, the simulation verifies that the current values of all partition boundary ports match their counterpart unicore values. Any difference found indicates that something went wrong in multi-core simulation, and needs attention. Value differences at time 0 may be an exception.

Procedure

Invoke unicore debugging by using the -mc2unicore argument to the mc2com command. Note that you must assign this argument one of the following enumerated values: all, out, in.

Tip

For more information on the -mc2unicore argument, refer to "-mc2unicore {all | out | in}" on page 80.

Results

If multi-core simulation detects a discrepancy with a unicore run, it reports one of the following message types:

- Delta time mismatch
    - If the next time in the multi-core simulation is less than in unicore, a warning message is displayed. It is okay to have more events in the multi-core simulation flow than in the unicore run. The warning message alerts you so that you can investigate the discrepancy, if needed. For example:

      ```
      ** Warning: (vsim-8596) time 0, next time delta mismatch between
      MC2 core (dt = 1) and Uni core(dt = 10)
      ```

    - If the next time in the multi-core simulation is higher than in unicore, an error message displays. This error always indicates a problem that you need to examine and correct. For example:

      ```
      ** Error: (vsim-8596) time 10, next time delta mismatch between
      MC2 core (dt = 105) and Uni core(dt = 7)
      ```

- Boundary port mismatch

- If at any time, there is a value mismatch for the partition boundary port in unicore and multi-core simulation, an error message displays. This error also occurs for any hierarchical ref usage across the partition. For example:

  ```
  ** Error: (vsim-8647) Value mismatch between unicore (val = 0)
  and partition 'p1' (val = x) for port 'tb.U1.a_n1.z' (id = 0).
  Time: 101 ns Iteration: 0 Region: /tb
  ```

- Sometimes the multi-core simulation value has a mismatch at time 0. This can happen when partitioning results in different events ordering. It may or may not be an issue. You can use the vsim -mc2nozerochk command to convert these errors to warnings at time 0. For example:

  ```
  ** Warning: (vsim-8647) Value mismatch between unicore (val
  =xxxxxxxxxxxxxxxx) and partition 'p1' (val = 0000000000000001)
  for port 'top.sub1.so' (id = 0).
  Time: 0 ns Iteration: 3 Region: /top File: test.v
  ```

- Sometimes when an inout port is at partition boundaries, some of the elements of the port are undriven. By LRM rules, the inout port should be driven by the initial value of the port. However, synthesis tools do not create an extra driver. Also, unisim, under some optimizations, may stop creating the extra driver. To remove the simulation versus synthesis mismatch under remaining cases, use the unisim -defaultstdlogicinittoz argument. Because of partitioning, these optimizations might not take effect with the multi-core simulation run, and there might be a mismatch between unisim and multi-core simulation. Use the -defaultstdlogicinittoz option with both unisim and multi-core simulation to verify removal of the simulation mismatches.

Examples

- The following shows how to use the -mc2unicore argument with the value all:

  mc2com top -o run_mp -mc2partfile part01.part -mc2unicore all

CPU or Core Binding (Affinity)

Affinity is the process of binding a process or memory to a CPU or core on a multi-processor system.

Binding a process to a CPU eliminates the cost of process relocation between CPUs. Process affinity gives the process exclusive use of a CPU, which improves the performance of the process. Memory affinity assigns physical memory to a process from the CPU/core that the process runs on. This gives the process better memory access performance, which can improve process performance.

Using affinity can improve multi-core simulation performance. The degree of performance improvement and type of settings required can vary according to the host hardware, including CPU type and memory architecture.

CPU/Core Binding Methods

You can control the CPU/core binding for multi-core simulation by using vsim -mc2binding.

The following binding methods are available for multi-core simulation:

- If you do not specify the -mc2binding argument — By default, the -mc2binding affinity argument is enabled. Each partition is bound to a specific core using taskset or numactl, based on the type of machine. Before starting simulation, the vsim command generates a file named *.mc2.affine* in the current working directory. This generated file is specific to the machine the simulation is being run on. The *.mc2.affine* file contains the setting to bind cores to individual partitions.
- If you specify the -mc2binding affinity:user argument — You must provide the *.mc2.affine* file in the current working directory. You can manually create the *.mc2.affine* file, modify a previous run on the same machine, or use the utility mc2_util affine to generate the file. When you use this argument, vsim does not generate the *.mc2.affine* file. Instead, the simulation reads the existing *.mc2.affine* file, and core binding is done as you specify.
- If you specify the -mc2binding {core | socket | rr} argument — Partitions are bound to CPU/cores using MPICH2's built-in binding capability. Results with these binding methods can vary, based on machine configuration, and may provide better or worse performance when compared to the default affinity binding. You can use this argument if you are fine-tuning performance to see if it improves your results.

- If you specify -mc2binding none argument — CPU/core binding is disabled, and multi-core simulation is run with no core binding. This argument can affect performance. You might want to use this argument when more than one multi-core simulation is run at the same time on the same machine, and the number of cores on the machine is not equal to the sum of all partitions running for all multi-simulations on a machine.

Auto-generated Binding (Default Method)

The Questa SIM simulator can automatically generate binding information for a machine and use it during multi-core simulation. Binding information is generated in the *.mc2.affine* file.

Based on whether you have Intel or AMD CPUs, you use taskset or numactl commands to bind a CPU/core to a process during multi-core simulation. AMD CPUs are NUMA (Non-Uniform Memory Access) systems. Newer Intel CPUs are also NUMA systems, some earlier Intel architectures are UMA based.

On AMD machines, the control command is as follows:

```
numactl -c C[,C] -m M[,M] ...
```

where -c specifies the CPU(s)/core(s) the process is to be run on, and -m specifies the CPU/core the memory is allocated from.

Tip
On some NUMA systems, numactl may not be functional. Use taskset in that situation.

On an Intel machine, the control command is as follows:

```
taskset -c C[,C]
```

where -c specifies the CPU(s)/core(s) the process runs on.

Refer to man pages of numactl and taskset for more information.

The following are example *.mc2.affine* files for an AMD-based host and Intel-based host:

AMD Host

```
MC2_AUTO_AFFINE_0="numactl --physcpubind 0 --membind 0"
export MC2_AUTO_AFFINE_0

MC2_AUTO_AFFINE_1="numactl --physcpubind 1 --membind 1"
export MC2_AUTO_AFFINE_1

MC2_AUTO_AFFINE_2="numactl --physcpubind 2 --membind 2"
export MC2_AUTO_AFFINE_2

MC2_AUTO_AFFINE_3="numactl --physcpubind 3 --membind 3"
export MC2_AUTO_AFFINE_3
```

Intel Host

```
MC2_AUTO_AFFINE_0="taskset -c 7"
export MC2_AUTO_AFFINE_0

MC2_AUTO_AFFINE_1="taskset -c 6"
export MC2_AUTO_AFFINE_1

MC2_AUTO_AFFINE_2="taskset -c 4"
export MC2_AUTO_AFFINE_2

MC2_AUTO_AFFINE_3="taskset -c 5"
export MC2_AUTO_AFFINE_3
```

Binding information is generated in the *.mc2.affine* file based on available cores on the machine with sequential number in the format MC2_AUTO_AFFINE<num>. Binding with number 0 is assigned to master partition. Binding with subsequent numbers (1, 2, 3…) are assigned to slave partitions in the order partitions are defined in the partition file.

User-defined Binding

The core binding file generated by vsim (*.mc2.affine*) or MPICH2's built-in binding assign binding based on the processor's architecture. The core binding file does not take into account dynamic loading among the cores/CPUs on the system. You may be able to improve performance by applying binding based on the current load on the machine. You can apply user-defined binding, if you think it will perform better than automatically generated binding. To specify user-defined binding, you can generate a new *.mc2.affine* file using the mc2_util affine command, and then modify the existing *.mc2.affine* file.

Before you modify the existing *.mc2.affine* file, you should know the configuration of the system that you are running multi-core simulation on. The most important information is the number of cores and CPUs available on the system. You should also know how the cores are numbered and the core/CPU relation.

Linux

On most Linux installations, you can examine the file */proc/cpuinfo* to determine CPUs/ cores configuration.

For platforms containing multi-core CPUs, you should assign the master partition to the CPU where the least number of cores are being used to run the simulation. For example, assume you are running on a computer with dual Intel Quad-core CPUs, where one CPU administers all of the cores with odd numbers and the other CPU administers the cores with even numbers. If your multi-core simulation runs in five partitions, this rule suggests you should assign the master partition to 0, and one of the remaining partitions to core 2 (the same CPU as the master). Assign the remaining partitions to cores 1, 3, and 5 (a different CPU from the master).

```
MC2_AUTO_AFFINE_0="taskset -c 0"
MC2_AUTO_AFFINE_1="taskset -c 2"
MC2_AUTO_AFFINE_2="taskset -c 1"
MC2_AUTO_AFFINE_3="taskset -c 3"
MC2_AUTO_AFFINE_4="taskset -c 5"
```

By default,Questa SIM uses multi-threading for WLF logging. Therefore, for each of the partitions where you are logging signals, results might be better if you assign those partitions to two cores from the same CPU. For example, if cores are assigned to CPUs using the even/odd method described above, and if the master partition is doing the logging, you could assign 0 and 2 to the master partition, as follows:

```
MC2_AUTO_AFFINE_0="taskset -c 0,2"
```

Binding for Multi-core Simulation on Multiple Machines

Use the vsim -mc2network <hostfile> command to run multi-core simulation on multiple machines. The hostfile contains the information about which machines to run simulations on, and how many cores to use on each machine. You can specify different bindings for each of the machines in the same hostfile. You can specify that processes are bound close to each other without sharing a socket (cpu:sockets), or you can specify your own ("user") core binding.

Syntax

<hostname>[:<num_procs>] [binding=cpu:sockets|user:<c, c>]

Example

```
host1:2                        # Run 2 process on host1 without any binding
host2:4 binding=cpu:sockets    # Run 4 processes on host2 with cpu:sockets
                               #       type binding
host3:2 binding=user:0,3       # Run 2 processes on host3 with binding to
                               #       cores 0 and 3
```

Consider the load of each CPU when assigning binding. It is generally preferable to have the CPU with the master partition slightly less loaded than the others. Each CPU should have a comparable total load.

Multi-core Simulation Grid Control

Control grid-based multi-core simulation with the vsim -mc2grid argument to specify the grid type.

LSF Grid

Syntax

bsub -n <num_partition> -R "span[hosts=1]" vsim -mc2 –mc2grid lsf <vsim_options>

Example

This example runs a four-partition multi-core simulation using the LSF type grid.

```
bsub -n 4 -R "span[hosts=1]" vsim -mc2 -mc2grid lsf top_opt_4
```

Sun Grid Engine (SGE)

Syntax

qsub –pe <sge_env> <num_partition> vsim –mc2 –mc2grid sge <vsim_options>

Example—Grid Environment Configuration

For the following example, assume the SGE configuration (<sge_env>) defined below. Your grid administrator provides your configuration; this configuration is just an example.

```
pe_name                  <sge_env>
slots                    9999
user_lists               NONE
xuser_lists              NONE
start_proc_args          /bin/true
stop_proc_args           /bin/true
allocation_rule          $pe_slots
control_slaves           FALSE
job_is_first_task        TRUE
urgency_slots            min
accounting_summary       FALSE
```

This example runs a four partition multi-core simulation using the SGE type grid.

```
qsub -pe penv 4 vsim -mc2 -mc2grid sge top_opt_4
```

Recommendations and Limitations

You can improve the effectiveness of your multi-core simulation by observing recommended practices and known limitations.

Recommendations for Running Multi-core Simulation

You can optimize multi-core simulation operation by observing recommendations for system preferences.

- Multi-core simulation runs better on a single multi-processor system (SMP) than on multiple multi-processor systems. Running multi-core simulation over a multi-system can have a significant negative effect on performance. If it is absolutely necessary to run multi-core simulation over multiple systems, you should make sure that at least one of the following is true:

    - Simulation runtime footprint is too big to fit into the physical memory of a single system.
    - The sync between the partitions can be reduced to offset the cost of communication between systems. (An example is to use a user sync event to control sync.)
    - The systems are connected to the same network switch.

- Run multi-core simulation on its own dedicated system, so that it can deliver the best performance possible.
- By default, multi-core simulation applies process and/or memory affinity to each partition. Running more than one multi-core simulation on the same system at the same time can cause extreme CPU and memory contention. Therefore, you should turn off affinity for the second or later multi-core simulation runs by specifying the -mc2binding none argument to the vsim command:

  vsim -mc2binding none

Flexible Limitations

You should observe the following limitations as nearly as possible to reduce or avoid multi-core simulation failure and error conditions.

- The DUT contains few levels of hierarchy (partition designs to the higher levels of (functional) hierarchy).
    - If a design can be partitioned at a higher level of hierarchy that represents a well defined functional block of the design, it will be better for multi-core simulation partitioning and will have a better chances to deliver performance. Partitioning at a higher level will also help during the multi-core simulation debugging process, if one is needed.
    - You can also create a partition's hierarchy by partitioning the design in the same hierarchical path on the top/bottom of another partition. However, if there is communication going across all levels of these partition hierarchies, it can slow down the simulation.
- The design (DUT and test bench) has no race conditions or is not sensitive to them.
    - A race-free design always produces the *correct* results that are easier for multi-core simulation to match.
    - If the design can handle a race condition (race-insensitive), it is easier for multi-core simulation to run the design.

- Conditional force and change commands are not supported if objects in condition and in force command belong to different partitions.

Rigid Limitations

You should always observe the following limitations to avoid multi-core simulation failure and error conditions.

- The $dumpvar() command can only dump objects in the calling partition
- File I/O involving multiple partitions is not supported. You need to re-code their algorithm to avoid race conditions.
- PLI Traversal
    - Only access objects in the partition where the calls are made.
    - Static reference to objects not in the current partition may fail.
    - File I/O may need to be redesigned to avoid collision.
- Cross-partition task/function calls
    - Pure functions and tasks are supported, provided that the function does not contain or reference:
        - Events, strings, interfaces, classes, covergroups, or struct or union types or variables.
        - You can turn off pure functions and tasks by using the mc2com -mc2nopurefunc argument.
    - Impure functions are supported provided that:
        - The function does not contain or reference struct or union types or variables.
        - There are no local variables that are read before being written, and the function is called from multiple partitions.
        - You can turn off impure functions and tasks by using the mc2com -mc2noimpurefunc argument.
- Globally shared memory using hierarchical references are supported only on single multiprocessor (SMP) system.

  Write access needs to be 32-bit “bounded.” For example, if the memory is declared as reg[0:63] Mem[0:10], and partition 1 is writing to Mem[5][17] at time 10ns, then no other partition can write to any bit/part of [0:31] of Mem[5] at the same time.
- SystemVerilog complex object types on partition boundary instances' ports, or in hierarchical references which cross-partition boundaries. Complex object types include classes, structures, interfaces, and strings.

- Static object sharing — Static objects cannot be shared across partitions. If they are shared, each partition will have an individual copy. This may still work for some test bench situations when the information can be post-processed.
- SystemC on a partition boundary is not supported.
- SDF
    - Observe caution when specifying uncompiled SDF files to vsim, as output file collisions may occur as a result of multiple partitions compiling the same SDF file to the same output location. It is recommended to compile SDF files explicitly using the sdfcom command, prior to running the mc2com or vsim commands. Uncompiled SDF files may also be specified to mc2com, but they are compiled once for each partition (to separate locations), incurring extra overhead.
    - Module path delays, which are applied on a partition boundary net or its collapsed net, are not supported.
    - Inertial (non-transport) interconnect delays may not work when the destination port is on a partition boundary. Transport interconnect delays may not work when the source or destination port is on a partition boundary. An error is reported by vsim for cases that are not supported.
    - Cross-partition multi-source interconnect delays are not supported.
- VCD Debug Flow
    - Reg type ports at partition boundary are not supported.
    - Bit-select, part-select ports at partition boundary are not supported.
    - Complex SV and VHDL types (except std_logic) at partition boundary are not supported.
    - Cross-partition hrefs cannot be driven with -vcdstim.
    - Partition file with user-defined events is not supported.
- Multi-core Simulation VHDL Flow
    - Cross-partition impure function/procedure calls access, declared in packages.
    - Any limitations in current vopt while propagating generics, port constraints, complex configurations, generate unrolling.
    - FLIs/PLIs
    - Cross-partition hier-ref/Signal Spy support is limited to signal type only.
    - Cross-partition call of hier-ref of constant and variable type.
    - Cross-partition access of package signals through hier-ref.
    - Cross-partition accessing variable of access and protected type in VHDL.

    - If hierarchical reference or Signal Spy call have unresolved paths (unrolled for-gen loop or complex expression).
    - The Following complex types at partition boundaries are not supported:
        - File type, access type, error type, incomplete type, or an array/record of these types.
        - Resolved record type at partition boundary is not supported. However, resolved record is supported for input and output ports at partition boundary.
- Signal Spy support limitations

  Cross-partition Signal Spy calls have limited support in multi-core simulation flow. The limitations are detailed in the following sections.

    - **VHDL to VHDL limitations**
        - signal_release, enable_signal_spy, disable_signal_spy and init_signal_spy with control state ON.
        - If Signal Spy calls have unresolved paths (unrolled for-gen loop or complex expression).
    - **Verilog to Verilog limitations**
        - $enable_signal_spy, $disable_signal_spy, $init_signal_driver and $init_signal_spy with control state ON
    - **Mixed language limitations**
        - Cross-partition usage of mixed Signal Spy calls has same limitations as above.
        - Port types that are not supported cross-partition boundary are also not supported in Signal Spy calls.

Feature Limitations

The following features are not yet supported in the multi-core simulation flow:

- GUI mode
- Limited CLI support — CLI that access cross-partition objects may not work correctly
- Power Aware Simulation
- Post-sim debug dataflow analysis
- Preoptimized Design Unit (PDU) flow

Chapter 2
Results and Performance Data from Multi-core Simulation

After you have run multi-core simulation, you can explore various methods of evaluating results and analyzing performance.

Multi-core Simulation Statistics from Unisim

Collect and use data from regular vsim simulation (unisim) runs for early qualification of a design for multi-core simulation.

Use the following types of information to evaluate results from multi-core simulation:

- Performance profile database obtained from a unisim run
- WLF logs

Collecting Information for Multi-core Simulation

Use performance profiling to collect multi-core information from a simulation run.

Restrictions and Limitations

- This procedure is not supported for the Pre-optimized Design Unit (PDU) flow. Some modifications are necessary to make this work if PDUs are present in the design.

Prerequisites

- You already have a good operational flow in place for optimization (vopt) and unisim simulation (vsim).

Procedure

1. Collect performance profiler data from a regular simulation run with the vsim command.

   ```
   vsim -c top_opt -do "profile option keep_unknown on; profile on -p;  onfinish stop;
   run -a; profile save profile_perf.pdb;quit"
   ```

2. (Optional) To view the profiler structural report, run the following command.

   ```
   vsim -c -do "profile open profile_perf.pdb; profile report -structural"
   ```

   For details on the arguments supported by profile report command, refer to the Questa SIM documentation. Note that generating the structural report is optional and is not a necessary step of the mc2collect flow.

3. Optimize the design using vopt -mc2collectinfo. The following examples show how to use this argument:

   ```
   vopt top -o top_portlog -mc2collectinfo=profile_perf.pdb [-mc2instlimit N]
   ```

   ```
   vopt top -o top_portlog -mc2collectinfo [-mc2instlimit N]
   ```

   The -mc2collectinfo argument first shortlists instances, then applies +acc=p to preserve the ports of these instances, then produces a vsim do file (*mc2collect.do*) with add wave commands to log these ports in the WLF file. If the profile database (obtained in Step 1) is provided to vopt, then the short list of instances is based on the profile data. In the absence of the performance profiler database, the short list is generated by walking the Vtree (Breadth First walk) and selecting the top level instances. Currently, the shortlist is limited to a maximum of the 30 most intensive instances and can be overridden using the -mc2instlimit option. The *mc2collect.do* file and the instance list are located in the *_mc2collect/* directory.

4. Run vsim again with optimized design from Step 3 using the *mc2collect.do* file

   The newly created *mc2collect.do* file contains add wave command(s) generated by the -mc2collectinfo option. Note that you should include *mc2collect.do* in front of any Questa SIM commands that are needed for the normal simulation of the design. The simulation produces the WLF file.

   ```
   vsim -c top_portlog -do _mc2collect/mc2collect.do -wlf mc2inst.wlf
   ```

Results

The -mc2collectinfo argument to the vopt command prepares a short list of instances, and internally applies +acc on the ports of the instances to preserve them from optimization. It produces a do file (*_mc2collect/mc2collect.do*) containing the add wave commands on the ports of these instances that can be provided to the vsim for WLF logging. The short list of instances is determined based on the input performance profile database provided as -mc2collectinfo=. If the profile database is not provided (-mc2collectinfo with no arguments), then top-level instances are short listed.

The number of instances short listed can be controlled with the -mc2instlimit argument. By default, ports of 30 instances are chosen for logging. If the profile data base is provided, the default number 30 is usually good enough in capturing the port activity of the computationally significant instances (as the instances with most number of hits are chosen). However, if the profile database is not specified then, based on design hierarchy, 30 might be insufficient and you may have to specify higher number through the -mc2instlimit option.

Also, note that since you are applying +acc=p on the ports of the chosen set of instances, some of the vopt optimizations will be different from the regular vopt run on these instances.

Simulation Performance Data

By default, Questa SIM always displays basic information in the Transcript window at the end of an multi-core simulation.

For example:

```
# [MC2-STAT] Partition master (0)=> Value Triggers: 90, User Sync Events:
0, Total Sync Events: 389085, Inout Sync Events: 0, Dataless Sync Events:
385122, Idle Sync Events: 76, Max Sync Reloop: 1, Max Data Reloop: 0,
Events: 770507, Processes: 389263, No More Event Opt : 0
```

where

- Value Triggers — Total number of value triggers to be sent to the other partition.
- Sync Events — Number of various types of sync events that were executed during simulation.
- Max Sync Reloop — Maximum number of times the Sync FSM re-looped in a single time step. Usually, a larger number means changes are being dripped across a partition, which indicates a potential performance issue.
- Max Data Loop — The largest sync FSM re-loop count where data exchange took place. If this number is large, it is also a sign of a potential performance problem.
- Events — Number of events executed in the partition.
- Processes — Number of process executed in the partition.

Obtaining a Detailed Performance Report

Obtain a report on multi-core simulation performance at the end of a multi-core simulation run that includes more detail than what is shown in the Transcript window.

Procedure

Use the -mc2commstat argument to the vsim command:

```
vsim -mc2 top_opt -mc2commstat
```

Results

This command reports performance information at the end of an multi-core simulation run. The report contains data on the following:

- Command Statistics
- Execution Time Distribution

- Size of Inter-Partition Data Communication

Examples

The following examples show typical contents and formatting of a performance report.

Command Statistics

The first part of the report displays a count of the various types of communications that occurred in the multi-core simulation run, as in the following example:

```
# (1) Command Statistics
#           Command           Send           Recv
# --------------------------------------------------
#                Go              3              0
#              Sync              0              3
#        Inter-Proc              0              0
#               End              3              0
#              Data        1167027        1167030
#         No-Events              0              0
#        Next-Delta        1155243        1155366
#            Active              0              0
#   Next-Time-Iter-Q            123              0
#              Idle              0              0
#           Cfg-Msg              0              3
#          Cfg-Sync              3              3
# --------------------------------------------------
```

The report from the master partition provides the most important data, since it is the most communication-loaded partition.

Execution Time Distribution

The second part of the report shows the respective times taken for simulation and communication in a given multi-core simulation run, as in the following example:

```
#  (2) Execution Time Distribution
#
#      Total Time: 8.81s
#         Sim Time:  6.68s (75.86%)
#         MC2 Overheads:  2.13s (24.14%)
#             Idle Time: 0.01s (0.12%)
#             Dataless Comm: 0.65s (7.35%)
#             Data Comm:   1.06s (11.98%)
#                 Send Prep:       0.44s (5.01%)
#                 Send Data:       0.09s (0.98%)
#                 Recv Data:       0.38s (4.36%)
#                 Recv Apply:       0.14s (1.63%)
#             Time Sync:  0.41s (4.69%)
#                 Recv Time:       0.24s (2.78%)
#                 Misc Time:         0.17s (1.91%)
```

The simulation run time distribution is categorized into the following:

- Time running simulation

- Time doing multi-core simulation sync, which consists of:
    - Time to prepare the data to be sent
    - Send time
    - Receive time
    - Time to process the received data from other partitions

Size of Inter-Partition Data Communication

The third part of the report displays a column listing of the size of communications that occurred in the multi-core simulation run, as in the following example:

```
# Partition master (Id = 0): Inter-partition Data Exchange Summary:
#                       Sent To                                    Received From
# Part   Words     Count  %Ave    Total   %Rate         Ports    Count   %Ave    Total   %Rate
# ------------------------------------------------------------------------------------------
#   1  1042779   347593   3.00   2085560   16.7 |      347593   347593   1.00   2085560   16.7
#   2  1042782   347594   3.00   2085560   16.7 |      347593   347593   1.00   2085560   16.7
#   3  1042782   347594   3.00   2085560   16.7 |      347593   347593   1.00   2085560   16.7
#   4  1042782   347594   3.00   2085560   16.7 |      347593   347593   1.00   2085560   16.7
# -------------------------------------------------------------------------------------------
```

The Part column shows the partition ID number.

There are two types of data in this part of the report:

- Sent Data (Sent To), containing the following columns:
    - Words: Total number of words sent to another partition.
    - Count: Total number of times communication has been sent with word packets (non-empty packets). All sending words from a partition are stuffed into a single packet, and then sent across to another partition.
    - %Ave: Average number of words sent per packet.
    - Total: Total number of times communication has been sent, with and without words, empty packet. If there is no data to send, the partition still needs to communicate that. This is called an empty packet.
    - %Rate: Percentage of good and meaningful data communication sent. The higher this number, the better the communication.
- Received Data (Received From), containing the following columns:
    - Ports: Total number of port’s data received by this partition. Each boundary communication port may contain various numbers of words based on their data size.
    - Count: Total number of times communication has been received with ports (non-empty packets).
    - %Ave: Average number of ports received per packet.

- Total: Total number of times communication has been received, with and without ports, empty packet.
- %Rate: Percentage of good and meaningful data communication received. The higher this number, the better the communication.

In this example, the Sent To side shows the master partition sent 1042779 words of data to partition #1. However, only 16.7% of the data exchange contained good data (83.3% are empty exchange). On average, the exchanged data size is 3 words.

The Received from side shows that master received data for 347593 boundary ports from partition #1. Only 16.7% of the sync contained good data (83.3% empty), and the average size of the exchange is 1 port. The size of the actual exchange is not reported here; however, it can be found in a similar report for partition #1.

From this report, you can evaluate the quality of the partitioning and simulation. In the example report shown above, the %Rate values are very low, which suggests that multi-core simulation may be over-syncing, or was asked to do too many types of sync. The aggregated low %Rate also suggests that the test+DUT may have serialized execution behavior. That behavior is exposed by the partitioning. Either the design is not a good fit for multi-core simulation, or you should make improvements to the partitioning.

Commands for Saving Detailed Performance Data

You can save performance data with the mc2perfanalyze command, or the -mc2savestat and the -mc2commstat arguments to the vsim command.

- mc2perfanalyze — Analyzes the database, as described in Simulation Performance Data Analysis. The results obtained from the standalone mc2perfanalyze command are more readable than the reports generated by -vsim mc2commstat.
- vsim -mc2savestat — Captures the runtime performance data of all the partitions into a single SQL database file (*mc2data.mdb*), which is more comprehensive information than that obtained by vsim -mc2commstat. Using this argument internally enables vsim -mc2commstat functionality. Refer to Types of Captured Data.

  Note

  This must be passed to all the partitions.

  This functionality can be invoked by using vsim -mc2savestat[=*optimized_design_filename*]. For example:

  vsim -mc2 -mc2savestat <*optimized_design*>

- vsim -mc2commstat — displays multi-core simulation statistics to the Transcript window with minimal formatting.

Simulation Performance Data Analysis

Multi-core simulation collects runtime data during the course of the simulation and dumps the data into a SQL database.

Currently, you can enable data capture by using the vsim -mc2savestat[=filename] command. Refer to Commands for Saving Detailed Performance Data for more information.

Types of Captured Data

For each partition, the vsim -mc2savestat command captures and saves multi-core simulation data into a single SQL database.

The vsim -mc2savestat command saves the following types of simulation data, by default to the SQL file named *mc2data.mdb* :

- Time distribution (sim.time, comm.time)
- Data traffic information
- Commands exchange summary
- Event statistics (such as, number of value change triggers, number of process triggered, number of idle sync events, and so on)
- Port trigger information
- Instance trigger information
- Instance trigger pattern

Multi-core Simulation Data Analysis Reports

The mc2perfanalyze command analyzes the runtime information previously captured by the vsim -mc2savestat command in mc2data.mdb. More vsim arguments are available for analyzing multi-core simulation data than are available for analyzing unisim data.

The following examples show the reports for different arguments of the mc2perfanalyze command. Most arguments are optional. In the absence of these arguments, Questa SIM produces applicable reports.

Tip
You can use the -sortorder, -sortby, and -limit arguments to reformat and limit the size of the tables.

Example 2-1. Using the mc2perfanalyze Command for Multi-core Simulation Data Analysis

```
mc2perfanalyze mc2data_m256.db -analyze
#########################################################################
                              MC2 ANALYSIS REPORT
#########################################################################
# LOAD BALANCING ANALYSIS
Partitions NOT balanced.
        Based on active simulation times, following partitions are
overloaded :
                'master' (sim time (368.59s) >> min sim time (138.67s))
                'p2' (sim time (326.18s) >> min sim time (138.67s))
        Based on processes count in each partition, following partitions
are overloaded :
                'p2' (Processes (1.27B) >> min Processes (453.29M))

# CONCURRENCY ANALYSIS
Possible lack of concurrency between partitions.
        The following partition(s) were found to running alone during
simulation, while other partitions were idling : 'master'

# COMMUNICATION ANALYSIS
The following partition pair(s) communicated data a high number of times :
        'master' and 'p1' (7.78M)
The least communicated pair was :
        'master' and 'p2' (3.85M)

Significant difference in cross-partition data communication activity in
partitions. Number of sync events with data communication are:
        master (max): 10.81M, p2 (min): 7.58M.

# PARTITION ORDERING
For optimal performance, consider reordering partitions in partition file
as follows : master, p1, p2 (Suggestion is based on runtime load).
```

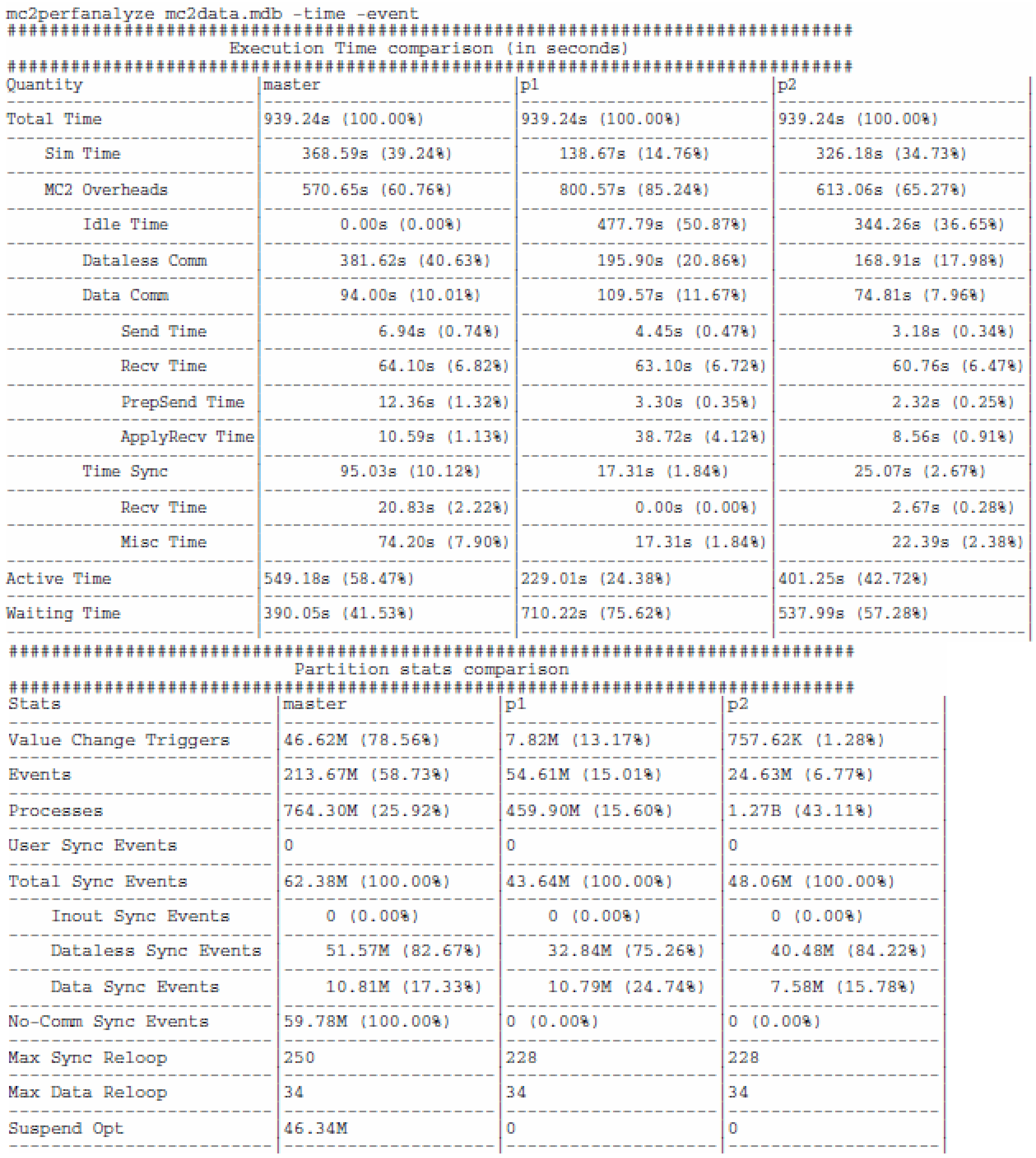
Example 2-2. Using the mc2perfanalyze Command with the -info Argument

The -info argument shows the connectivity information between partitions.

```
mc2perfanalyze mc2data.mdb -info
#########################################################
             Partition Connectivity Info
#########################################################
       Partitions|           master|              p1|              p2|
-----------------|-----------------|------------------|-----------------|
           master|               --|     15 sendports|    263 sendports|
               p1|      3 sendports|              --|               --|
               p2|    166 sendports|              --|               --|
```

Example 2-3. Using the mc2perfanalyze Command with -time -event

You can specify multiple arguments together, such as -time and -event.



```
mc2perfanalyze mc2data.mdb -time -event
##########################################################################
                   Execution Time comparison (in seconds)
##########################################################################
Quantity                 |master                   |p1                       |p2                       |
-------------------------|-------------------------|-------------------------|-------------------------|
Total Time               |939.24s (100.00%)        |939.24s (100.00%)        |939.24s (100.00%)        |
-------------------------|-------------------------|-------------------------|-------------------------|
    Sim Time             |    368.59s (39.24%)     |    138.67s (14.76%)     |    326.18s (34.73%)     |
-------------------------|-------------------------|-------------------------|-------------------------|
    MC2 Overheads        |    570.65s (60.76%)     |    800.57s (85.24%)     |    613.06s (65.27%)     |
-------------------------|-------------------------|-------------------------|-------------------------|
        Idle Time        |        0.00s (0.00%)    |        477.79s (50.87%) |        344.26s (36.65%) |
-------------------------|-------------------------|-------------------------|-------------------------|
        Dataless Comm    |        381.62s (40.63%) |        195.90s (20.86%) |        168.91s (17.98%) |
-------------------------|-------------------------|-------------------------|-------------------------|
        Data Comm        |        94.00s (10.01%)  |        109.57s (11.67%) |        74.81s (7.96%)   |
-------------------------|-------------------------|-------------------------|-------------------------|
            Send Time    |            6.94s (0.74%)|            4.45s (0.47%)|            3.18s (0.34%)|
-------------------------|-------------------------|-------------------------|-------------------------|
            Recv Time    |           64.10s (6.82%)|           63.10s (6.72%)|           60.76s (6.47%)|
-------------------------|-------------------------|-------------------------|-------------------------|
            PrepSend Time|           12.36s (1.32%)|            3.30s (0.35%)|            2.32s (0.25%)|
-------------------------|-------------------------|-------------------------|-------------------------|
            ApplyRecv Time|          10.59s (1.13%)|           38.72s (4.12%)|            8.56s (0.91%)|
-------------------------|-------------------------|-------------------------|-------------------------|
        Time Sync        |        95.03s (10.12%)  |        17.31s (1.84%)   |        25.07s (2.67%)   |
-------------------------|-------------------------|-------------------------|-------------------------|
            Recv Time    |           20.83s (2.22%)|            0.00s (0.00%)|            2.67s (0.28%)|
-------------------------|-------------------------|-------------------------|-------------------------|
            Misc Time    |           74.20s (7.90%)|           17.31s (1.84%)|           22.39s (2.38%)|
-------------------------|-------------------------|-------------------------|-------------------------|
Active Time              |549.18s (58.47%)         |229.01s (24.38%)         |401.25s (42.72%)         |
-------------------------|-------------------------|-------------------------|-------------------------|
Waiting Time             |390.05s (41.53%)         |710.22s (75.62%)         |537.99s (57.28%)         |
-------------------------|-------------------------|-------------------------|-------------------------|
##########################################################################
                         Partition stats comparison
##########################################################################
Stats                      |master                 |p1                     |p2                     |
---------------------------|-----------------------|-----------------------|-----------------------|
Value Change Triggers      |46.62M (78.56%)        |7.82M (13.17%)         |757.62K (1.28%)        |
---------------------------|-----------------------|-----------------------|-----------------------|
Events                     |213.67M (58.73%)       |54.61M (15.01%)        |24.63M (6.77%)         |
---------------------------|-----------------------|-----------------------|-----------------------|
Processes                  |764.30M (25.92%)       |459.90M (15.60%)       |1.27B (43.11%)         |
---------------------------|-----------------------|-----------------------|-----------------------|
User Sync Events           |0                      |0                      |0                      |
---------------------------|-----------------------|-----------------------|-----------------------|
Total Sync Events          |62.38M (100.00%)       |43.64M (100.00%)       |48.06M (100.00%)       |
---------------------------|-----------------------|-----------------------|-----------------------|
    Inout Sync Events      |    0 (0.00%)          |    0 (0.00%)          |    0 (0.00%)          |
---------------------------|-----------------------|-----------------------|-----------------------|
    Dataless Sync Events   |    51.57M (82.67%)    |    32.84M (75.26%)    |    40.48M (84.22%)    |
---------------------------|-----------------------|-----------------------|-----------------------|
    Data Sync Events       |    10.81M (17.33%)    |    10.79M (24.74%)    |    7.58M (15.78%)     |
---------------------------|-----------------------|-----------------------|-----------------------|
No-Comm Sync Events        |59.78M (100.00%)       |0 (0.00%)              |0 (0.00%)              |
---------------------------|-----------------------|-----------------------|-----------------------|
Max Sync Reloop            |250                    |228                    |228                    |
---------------------------|-----------------------|-----------------------|-----------------------|
Max Data Reloop            |34                     |34                     |34                     |
---------------------------|-----------------------|-----------------------|-----------------------|
Suspend Opt                |46.34M                 |0                      |0                      |
---------------------------|-----------------------|-----------------------|-----------------------|
```

Example 2-4. Using the mc2perfanalyze Command with -inst -part -topart

The -inst argument produces the trigger information of the instances in the boundary between each partition pair. Use the -part and -topart arguments to specify a partition pair.

```
mc2perfanalyze mc2data.mdb -inst -part master -topart p1

#############################################################################
Instance trigger table from 'master' to 'p1' (sorted by: 'instance_trig_count')
      (Note: Higher individual triggers implies that ports of the instance are
             triggering independently)
#############################################################################
instance_name  |nSendPorts |ports_trig_count |instance_trig_count |%individual |
               |           |                 |                    |   triggers |
---------------|---------- |---------------- |------------------- |------------|
tbleon.ram0    |         5 |            7324 |               6743 |     92.07% |
tbleon.ram1    |         5 |            7327 |               6743 |     92.03% |
tbleon.ram2    |         5 |            7322 |               6743 |     92.09% |
```

The column names indicate the following:

- instance_name — Boundary instance name between the partition pair (master and p1).
- nSendPorts — Number of send ports in the specific instance between the partition pair.
- ports_trig_count — Total number of times the ports of the instances triggered (changed value) due to the activity in the instance.
- instance_trig_count — Number of times the instance triggered. This number is less than or equal to the ports_trig_count.
- %individual triggers — Percentage of ports in an instance that are triggering independently of each other. Higher percentage indicates that ports of the instances are triggering independently of each other. This number can range from 100% to 100/ nSendports.

Example 2-5. Using the mc2perfanalyze Command with -port -part -topart

The -port argument produces the trigger information of the boundary ports in each partition pair. The partition pair can be specified by the -part and -topart arguments.

```
mc2perfanalyze mc2data.mdb -port -part master -topart p1

##################################################################
Port trigger table from 'master' to 'p1'  (sorted by: 'trig_count')
##################################################################
 port_num  |   instance_name   |          port_name   |  trig_count  |
-----------|-------------------|--------------------|--------------|
        1  |    tbleon.ram0    |                D     |        3789  |
        6  |    tbleon.ram1    |                D     |        3789  |
       11  |    tbleon.ram2    |                D     |        3789  |
        0  |    tbleon.ram3    |                A     |        2798  |
        5  |    tbleon.ram4    |                A     |        2798  |
       10  |    tbleon.ram5    |                A     |        2798  |
        2  |    tbleon.ram6    |                CE1   |         320  |
        7  |    tbleon.ram7    |                CE1   |         320  |
       12  |    tbleon.ram8    |                CE1   |         320  |
       13  |    tbleon.ram9    |                WE    |         300  |
        8  |    tbleon.ram10   |                WE    |         297  |
        3  |    tbleon.ram11   |                WE    |         295  |
        4  |    tbleon.ram12   |                OE    |         120  |
        9  |    tbleon.ram13   |                OE    |         120  |
       14  |    tbleon.ram14   |                OE    |         120  |
```

Example 2-6. Using the mc2perfanalyze Command with -pattern

The -pattern argument provides information about the concurrent activity between the instances at the boundary. A pattern is a set of active instances.

```
mc2perfanalyze mc2data.mdb -pattern -part master -topart p2 -limit 3
#####################################################################################
Instance trigger pattern between master and p2  (sorted by: 'instance_trig_count')
#####################################################################################
-------------------------------------------------------------------------------------
+tbleon.tb.p0.leon0.mcore0.clkgen0     [Instance trig. count: 38624] [Num. of Patterns: 9]
-------------------------------------------------------------------------------------
    =====> tbleon.tb.p0.leon0.iopadb5[10].iopadb51       [concurrent triggers: 24]
    =====> tbleon.tb.p0.leon0.iopadb5[12].iopadb51       [concurrent triggers: 12]
    =====> tbleon.tb.p0.leon0.iopadb5[14].iopadb51       [concurrent triggers: 12]

-------------------------------------------------------------------------------------
+tbleon.tb.p0.leon0.mcore0.reset0     [Instance trig. count: 33485] [Num. of Patterns: 6]
-------------------------------------------------------------------------------------
    =====> tbleon.tb.p0.leon0.iopadb8                    [concurrent triggers: 5]
    =====> tbleon.tb.p0.leon0.iopadb5[0].iopadb51        [concurrent triggers: 5]
    =====> tbleon.tb.p0.leon0.iopadb5[1].iopadb51        [concurrent triggers: 5]

-------------------------------------------------------------------------------------
+tbleon.tb.testmod0     [Instance trig. count: 7356] [Num. of Patterns: 29]
-------------------------------------------------------------------------------------
    =====> tbleon.tb.ram32d.rambnk[0].ramarr[1].ram0     [concurrent triggers: 7285]
    =====> tbleon.tb.ram32d.rambnk[0].ramarr[3].ram0     [concurrent triggers: 7285]
    =====> tbleon.tb.ram32d.rambnk[0].ramarr[0].ram0     [concurrent triggers: 7285]
```

The field names indicate the following:

- Instance trig. count — Total number of times the instance triggered, resulting in the activity on its ports
- Num. of Patterns — The number of different patterns this instance is part of.
- Concurrent triggers — Number of times the given pair of instances triggered concurrently.

Unisim Data Analysis Reports

Use the mc2perfanalyze command to render useful analysis based on the WLF file generated from vopt -mc2collectinfo in the unisim flow.

The mc2perfanalyze command analyzes the given WLF file and extracts important information into an SQL database (named *mc2collect.mdb*), which can be provided back to mc2perfanalyze with different options to get port and instance activity information.

The following command produces the SQL database with an optional filename that you can specify. The default database name is mc2collect.mdb.

```
mc2perfanalyze mc2inst.wlf -wlf2mdb[=<filename>]
```

If the WLF file is very large, this command can take some time to finish, but it is a one-time analysis on WLF. For all subsequent unisim information, the SQL database is used. The *mc2collect.mdb* database file contains information on port activity and instance triggering (using information from the -inst and -port arguments) that can help in partitioning of the design.

The -wlf2mdb argument produces two files *mc2collect.mdb* and *mc2_vsig.do*. The *mc2_vsig.do* file is a vsim DO file that customer support can use to further analyze the *mc2inst.wlf* file for concurrency. The *mc2collect.mdb* is a SQL database containing port and instance trigger information that you can analyze further by mc2perfanalyze and use to generate reports.

The following commands produce reports that show the ports that are highly active and the instances that trigger a high number of times.

```
mc2perfanalyze mc2collect.mdb -port
mc2perfanalyze mc2collect.mdb -inst
```

The following examples provide more details on analyzing unisim data, showing the reports resulting from using different arguments of the mc2perfanalyze command.

Example 2-7. Using the mc2perfanalyze Command with -info in Unisim

The -info argument provides general info on the captured data.

```
mc2perfanalyze mc2collect.mdb -info
#########################################################################
                             Unisim Details
#########################################################################
                    Quantity|          Data|
----------------------------|--------------|
           Num. of Instances|            56|
----------------------------|--------------|
               Num. of Ports|           233|
----------------------------|--------------|
        Num. of Sigs or Vars|            72|
----------------------------|--------------|
                  Start Time|             0|
----------------------------|--------------|
                 Start Delta|             0|
----------------------------|--------------|
                    End Time|        289860|
----------------------------|--------------|
                   End Delta|             0|
----------------------------|--------------|
               Time Advances|         29825|
----------------------------|--------------|
              Delta Advances|        203202|
----------------------------|--------------|
```

Example 2-8. Using the mc2perfanalyze Command with -inst in Unisim

The -inst argument provides an instance trigger report. The -limit argument limits the report size. By default, the table is sorted in descending order of instance trigger count.

```
mc2perfanalyze mc2collect.mdb -inst -limit 5
########################################################################
Instance trigger table for Unisim data (sorted by: 'instance_trig_count')
     (Note: Higher individual triggers implies that ports of the instance are
           triggering independently)
########################################################################
instance_name      |nPorts |ports_trig_count |instance_trig_count |%individual |
                   |       |                 |                    |   triggers |
------------------ |------ |---------------- |------------------- |----------- |
/tbleon/proc0      |    27 |          235696 |             139606 |     59.23% |
/tbleon/m0         |    32 |          172630 |             125014 |     72.42% |
/tbleon/iu0        |    19 |          194870 |             103860 |     53.30% |
/tbleon/a0         |    12 |          118214 |              91895 |     77.74% |
/tbleon/c0         |    15 |          169688 |              81111 |     47.80% |
```

Example 2-9. Using the mc2perfanalyze Command with -port -limit in Unisim

The -port argument provides a port trigger report. The -limit argument limits the report size. By default, the table is sorted in descending order of port trigger count.

```
mc2perfanalyze mc2collect.mdb -port -limit 5

##############################################################
Port trigger table for Unisim data (sorted by: 'trig_count')
##############################################################
port_num |        instance_name |port_name |trig_count |
-------- |-------------------- |--------- |---------- |
      77 |/tbleon/iu0           |iuo       |     40343 |
      94 |/tbleon/p0            |iuol      |     40343 |
     207 |/tbleon/a0            |iuo       |     40343 |
     244 |/tbleon/p0            |iuo       |     40343 |
     266 |/tbleon/m0            |iuo       |     40343 |
```

Viewing a Virtual Signals File

You can use the vsim command to view the virtual signal file produced by the -wlf2mdb argument of mc2perfanalyze. There is one virtual signal for each instance, and an event on the virtual signal indicates an activity on the ports of that instance. It is possible to infer concurrent activity of instances by viewing these virtual signals as events.

Prerequisites

- You have run the mc2perfanalyze -wlf2mdb command.
- You have read access to the virtual signal file.

Procedure

1. Open the WLF file using the vsim command in interactive mode.

   vsim -view mc2inst.wlf

2. In the vsim command window, run the do command on the file *mc2_vsig.do*.

   do _mc2_collect/mc2_vsig.do

   This will open the wave window and shows all the virtual signals.

3. Select all the virtual signals.
4. Right-click the selected signals, which displays a popup menu for Format.
5. From the Format menu, choose Event.

Results

The format of the virtual signals changes from logic to event.

Chapter 3 Reference Information for Partition Files and Commands

This chapter provides reference information for partition files and the commands used for multi-core simulation. This information includes a description of how to modify the default set of sync points based on the post-simulation analysis of the design.

Partition Files

A partition file is a text file that identifies which portions of the design are assigned to a specific core for simulation.

Partition Definition

You can define multiple partitions according to design hierarchy and synchronization events.

Syntax

```
partition <partition_name> {
    <list_of_partition_members>;
}
```

where <partition_member> is a semicolon-separated list of mod_inst definition and/or sync_event definitions. You must name one partition master. This master partition oversees communication and control during the simulation.

Module Instance

Each partition requires at least one definition of a module instance that represents the top-level module of that partition.

When you specify a module instance for a partition, that instance and all its children instances become part of the partition. If any child instance is specified as a module instance for another partition, then this child instance and its children instances are not considered as part of the partition that contains its parent instance as module instance. Instead, they become part of the partition that specifies children instances as module instances.

Each partition can contain multiple module instance definitions.

Syntax

```
mod_inst = ( <module_name>, <instantiation_name> ) ;
```

or

```
mod_inst = <instatiation_name> ;
```

where *instantiation_name* is the full hierarchical pathname of the instance. Instance name identifiers (including escaped, extended, uppercase, lowercase, generates, array instance, and so forth) in the hierarchical pathname must follow the same language syntax as used in the design.

Sync Event Control

You use this construct to identify a user-defined sync event you introduce into the partition. The event is then monitored by Questa SIM multi-core simulation. Data synchronization is performed when this event is detected.

In addition to this specification in the partition file, you also need to modify the netlist in order for event synchronization to take effect—see User-defined Synchronization Events.

Syntax

```
sync_event = <event>
```

where *event* is full path name of the user sync event you added.

Common Module Definition

If your design has many hierarchical references to global objects, such as Vcc or ground signals, which are defined and driven inside a single module, you can improve performance by defining this as a “common module” in the partition file. This causes the mc2com command to replicate the common module’s logic in all the partitions and avoids large numbers of cross-partition hierarchical references to objects contained within that module, which improves simulation performance.

If your design has many such hierarchical references, but objects are not defined in a single module, you can modify the design to create a common module, and define and drive all such global objects from this common module.

The following syntax shows how to specify common modules in a partition file:

Syntax

```
common_module {
      <list_of_partition_members> ;
}
```

where *list_of_partition_members* is defined in “Partition Definition” on page 73.

Examples

Example 1 — Verilog module

```
module dsgn_globals;
    supply0 gnd;
   supply1 vcc;
endmodule
```

Example—Partition File

```
partition master {
  mod_inst = top;
}

partition p1 {
  mod_inst = top.sub1;
}

common_module {
  mod_inst = dsgn_globals;
}
```

Example—mc2com Command

```
mc2com top dsgn_globals -o run_mp -mc2partfile top.part
```

User-defined Synchronization Events

For each user sync event you define for a partition, you normally add the following type of construct to your Verilog netlist.

In the following example, the event is named my_sync_event, and the top of the partition is named top_of_partition:

```
module top_of_partition;
  event my_sync_event
  always @some_trigger
    -> my_sync_event
endmodule
```

You can also use an existing event from your design.

You should note the following points when using user-defined synchronization:

- The number of sync events used should be same for all partitions. If the number of sync events is not the same for all partitions, an error results.
- You can either create common events, using the common module for all partitions, or create equivalent event logic in each partition's hierarchy.
- If a user sync event is triggered by clock, make sure that each partition is running its clock without needing to synchronize. This can be done by duplicating clock logic in each partition or by defining clock logic in a common module and including it in all partitions. Using a common module makes it is easier for all partitions to share the same events.
- Make sure that events trigger in all partitions around the same time. Synchronization is controlled by this user-defined event. If an event triggers in some partitions and not in other partitions, the simulation may hang or produce incorrect results.
- You should still provide regular sync controls, such as Verilog. User-defined sync events only direct the simulator when to start synchronizations. How fine-grained synchronization should be in each time cycle is still determined by regular sync controls.

Command Reference

Questa SIM provides two commands, and several arguments to the vsim command, that are specific to multi-core simulation.

mc2com

The mc2com command initiates multi-core simulation in Questa SIM.

Syntax

```
mc2com [options]
    <top-level_design> -o <partition_name>
    [-designdir <directory_name>] [-j <num>] [-mc2allowgenhref] [-mc2analysisreport <file>]
    [-mc2autopartreport[=verbose] <file>] [-mc2genfsdb] [-mc2ignoreexterns]
    [-mc2makeinoutnets] [-mc2multidimunrollimit <num>] [-mc2network <hostfile>]
    [-mc2noglobalmem[=<mem_pathname>]] [-mc2noiface] [-mc2noimpurefunc]
    [-mc2nomultidim] [-mc2nopurefunc] [-mc2novarsel] [-mc2numpart <num>]
    [-mc2partfile <filename>] [-mc2unicore {all | out | in}]
    [-mc2useprofile[=<mem | perf>] <file>] [-mc2vhdlpart] [-mc2vj <num>]
    [-mc2vlogargs <vlog_arguments>] [-mc2vlogpart]
    [-mc2voptargs[="<prttn_name>"] "<vopt_args>"] [-stats] [-vv]
```

Note

Currently, SDF compiled with mc2com does not work, unless it is limited to the netlist within a single partition. The incr SDF flow works even if it crosses partitions.

Arguments

- top-level_design

  Name of the top-level design on which to perform multi-core simulation.

- -o <partition_name>

  Specifies the name of the partition file to create based on your input settings.

- -designdir <directory_name>

  Specifies the location where the mc2com command creates the various *design.bin* files based on the different partitions. When you point Visualizer to this directory, it will combine and present them as a single result.

  Use this argument only if you are using the Visualizer Debug Environment with Questa SIM simulation.

- -j <num>

  Specifies the maximum number of total parallel code generation processes for all partitions combined used by the vopt command. This argument evenly distributes the code generation processes across the vopt jobs for all partitions.

- -mc2allowgenhref

  Allows hierarchical references to bit- and part-selects with index expressions composed of genvars.

- -mc2analysisreport <file>

  Generates a partition analysis report and saves to the file specified.

- -mc2autopartreport[=verbose] <file>

  Generates auto-partitioner report on the design nodes, algorithm parameters, and run-times. It reports the instances that could not be mapped to partitions and the reasons for not mapping them. The verbose mode (=verbose) reports more information about unsupported hierarchical references, signal-spy actions, and global variables that rendered the instances unmappable. On large designs, verbose report could be overwhelming. If you are using verbose report mode, it is recommended that you look at the “Unmappable nodes summary” first, and if you need more information about particular unmappable node, lookup the verbose report to get more details on unsupported hierarchical references, signal-spy actions or global variables.

- -mc2genfsdb

  Enable auto-dumping of fast-signal database (fsdb) files from all the partitions.

  Note

  Fast-Signal Database (fsdb) is a type of Verilog system task or command that is supported as part of the Verdi Automated Debug System from Synopsys, Inc. For more information on how to generate an fsdb file, refer to TechNote MG500305, which you can find in Support Center.

  By default, this is switched off—only one partition that has fsdb APIs generates fsdb. Currently, this argument supports only the $fsdbDumpfile and $fsdbDumpvar APIs.

  Tip

  You can use standard utilities such as fsdbmerge on the fsdb for all generated partitions. However, fsdbmerge can be slow for large fsdb files, in which case, you should use the Verdi virtual file flow. For more information on using Verdi Automated Debug System refer to the documentation for Verdi3, available on the worldwide web at www.synopsys.com.

- -mc2ignoreexterns

  Ignore failed externs during auto-partition analysis.

- -mc2makeinoutnets

  Makes port direction for all Verilog net connections at multi-core simulation partition boundary 'inout'. This argument may be necessary for correct results, when a Verilog net connected to an “output” port on a partition boundary is forced within the driving partition, and also forced or released by another partition. This argument can also be used to ensure nets connected to partition output ports, which have contributions (drivers) in other partitions, always show the correct value when logged in a WLF file.

- -mc2multidimunrolllimit <num>

  Specify a limit to the unrolling of multi-dimension ports. By default this limit is set to 256.

- -mc2network <hostfile>

  Run partition vopt jobs on multiple computers. Also useful when one machine does not have sufficient memory or CPU resources to run all vopt processes at once, or to speed up the processing if the host machine can not efficiently run all vopt processes at once. The usage for this switch (hostfile format, and so on.) is the same as for the -mc2network switch for vsim. Refer to Running Multi-computer Simulation for details on <hostfile> syntax.

- -mc2noglobalmem[=<mem_pathname>]

  Disables shared memory usage to implement all cross-partition RTL memories. If full pathname of <mem_pathname> is specified, then only this memory is disabled as shared memory. This argument can be specified multiple times.

- -mc2noiface

  Disallow cross partition usage of interface/modport.

- -mc2noimpurefunc

  Disallow cross-partition impure function usage in auto-partitioner.

- -mc2nomultidim

  Disallow multi-dimension nets as ports at partition boundary.

- -mc2nopurefunc

  Disallow cross-partition pure function usage in auto-partitioner.

- -mc2novarsel

  Hierarchical references to bit- and part-selects with variable index expressions are supported by default. Use this argument to disallow variable index expressions.

- -mc2numpart <num>

  Specify number of partitions needed from auto-partitioner.

- -mc2partfile <filename>

  Specifies a partition file to be used for multi-core simulation.

- -mc2unicore {all | out | in}

  Run unicore flow (See “Debugging Simulation Mismatches With Unicore Flow” on page 44), where:

  all — Generates unicore ports for all boundary ports of current partition.

  out — Generates unicore ports for only those boundary ports that are going out of current partition (that is, 'output/inout' module ports in top/higher boundary and 'input' module ports in bottom/lower boundary of current partition).

  in — Generates unicore ports for only those boundary ports that are coming in the current partition. (that is, 'input' module ports in top/higher boundary and 'output/inout' module ports in bottom/lower boundary of current partition).

For example, the following command generates all required information to run the entire design as a separate partition:

```
mc2com top -orun_mp -mc2partfile part01.part -mc2unicore all
```

- -mc2useprofile[=perf] <file>

  Specify profile database file to feed back to auto-partitioner.

  perf — specifies performance database.

  The default implies performance profile database.

- -mc2vhdlpart

  Allow only VHDL instances at a partition boundary in auto-partitioner.

- -mc2vj <num>

  Specify maximum number of partition vopt jobs to run in parallel. This argument controls the number of partition vopt processes run in parallel, but does not control the code generation processes that get spawned by vopt (controlled by the -j option of this command). The purpose of -mc2vj is to handle a host that does not have sufficient memory resources to run all the partition vopt processes at once. You can specify "-mc2vj 0" or "-mc2vj 1" to allow only one partition vopt process to be run at a time.

- -mc2vlogargs <vlog_arguments>

  Specify vlog command arguments for MC2 interface file compilation.

- -mc2vlogpart

  Allow only Verilog/SystemVerilog instances at a partition boundary in auto-partitioner.

- -mc2voptargs[="<prttn_name>"] "<vopt_args>"

  Specify vopt command arguments for all or individual partitions.

  =<prttn_name> — Optionally specifies the partition name. If you do not specify a partition name, vopt arguments are passed to all partitions.

  <vopt_args> — Specifies vopt command arguments to be applied to all of the mc2com phases: auto-partition, partition analysis, and partition optimization.

- -stats [=[+|-]<feature>[,[+|-]<mode>]

  (Optional) Controls display of compiler statistics sent to a logfile, stdout, or the transcript. Specifying -stats without options sets the default features (cmd, msg, and time).

  Multiple features and modes for each instance of -stats are specified as a comma separated list. You can specify -stats multiple times on the command line, but only the last instance will take effect.

  [+|-] — Controls activation of the feature or mode. You can also enable a feature or mode by specifying without the plus (+) character. Setting this switch will add or subtract features and modes from the default settings "cmd,msg,time".

  all — Display all statistics features (cmd, msg, perf, time). Mutually exclusive with none option. When specified in a string with other options, +|-all is applied first.

cmd — (default) Echo the command line.

msg — (default) Display error and warning summary at the end of command execution.

none — Disable all statistics features. Mutually exclusive with all option. When specified in a string with other options, +|-none is applied first.

perf — Display time and memory performance statistics.

time — (default) Display Start, End, and Elapsed times.

Modes

kb — Print memory statistics in Kb units with no auto-scaling.

list — Display performance statistics in a Tcl list format when available.

verbose — Display verbose performance statistics information when available.

Note

mc2com -quiet disables all default or user-specified -stats features.

- -vv

  Echo subprocess invocations to STDOUT.

Examples

- Enable the display of Start, End, and Elapsed time, as well as a message count summary. Echoing of the command line is disabled.

  ```
  mc2com -stats=time,-cmd,msg
  ```

- The first -stats option is ignored. The none option disables all *modelsim.ini* settings and then enables the perf option.

  ```
  mc2com -stats=time,cmd,msg -stats=none,perf
  ```

vsim

Questa SIM provides arguments for the vsim command that are specific to multi-core simulation.

Tip

For the complete reference description of all arguments to the vsim command for unisim, refer to vsim in the Questa SIM *Command Reference Manual*.

Syntax

vsim -mc2 [-mc2network <hostfile>]
[-mc2binding <type>] [-mc2grid {lsf | sge}] [-mc2sal <partition_name>]
[-mc2mergeucdb <filename>] [-mc2mergewlf <filename>] [-mc2vcddump]
[-mc2commstat] [-mc2savestat[=filename]] [-mc2sanitycheck[=warn]]
[-mc2nozerochk] [-mc2noidlesyncopt] [-mc2nosuspendopt] [-mc2noactivequeueopt]
[-mc2activequeueopt] [-mc2synccntl=<sync_cntl>[,<sync_cntl>...]]
[-mc2vsimargs="<partition>" "<args>"]
[-qwavedb=+wavedir=<*dirname*>]
[-vv]

Arguments

- -mc2

  Run simulation in multi-core simulation mode. This argument also implies the vsim argument.

- -mc2network <hostfile>

  Run simulation in multi-computer mode. See Running Multi-computer Simulation for details on <hostfile> syntax.

- -mc2binding <type>

  Run simulation with specified core binding, where <type> is one of: none, affinity, affinity:user, core, socket, rr. Default binding is affinity.

- -mc2grid {lsf | sge}

  Run simulation with the specified grid type. Supported grid type options are Platform LSF (LSF), or Sun Grid Engine (SGE).

- -mc2sal <partition_name>

  Runs simulation on one standalone partition, <partition_name>, in VCD mode.

- -mc2mergeucdb <filename>

  Merge UCDB files from all partitions to the specified filename.

- -mc2mergewlf <filename>

  Merge WLF files from all partitions to the specified filename.

- -mc2vcddump

  Generate VCD info and enable later standalone mode simulation run on this partition.

- -mc2commstat

  Generate multi-core simulation statistics.

- -mc2savestat[=filename]

  Save multi-core simulation statistics to the specified filename.

- -mc2sanitycheck[=warn]

  Perform sanity design checks for allowable boundary conditions. Optional '=warn' argument converts severity of error message to warning.

- -mc2nozerochk

  Converts severity of time 0 unicore value mismatch errors to warnings.

- -mc2noidlesyncopt

  Disables optimization, which suppresses synchronizations when all partitions but one are idle.

- -mc2nosuspendopt

  Use this argument to disable an optimization on partitions without any activity. If time synchronization between any partitions is incorrect or resulting in incorrect output, then this argument can be used. This argument used together with -mc2noidlesyncopt option forces the partitions to synchronize at each simulation time step.

- -mc2noactivequeueopt

  Disables optimization on active queue communications. When only VHDL sync control is specified and the design either shows incorrect results or is a mixed HDL design you can use this argument to see if results improve.

- -mc2activequeueopt

  Enable optimization for active queue communication.

- -mc2synccntl=<sync_cntl>[,<sync_cntl>...]

  Specify predefined MC2 sync control options for simulation. Sync control options are specified in the form: name=value. For example:

  ```
  vsim -mc2 run_mp -mc2synccntl=vhdl,looplimit=10,nba
  ```

  The options for sync_control are named events that are case-insensitive; no space is allowed between multiple options. For more comprehensive information on synchronization controls, values, sync control aliases, and sync point setting strategy, see Synchronization Control.

Table 3-1. Sync Control Named Events — Quick Reference

Named Event	Description
active	Sync will be performed at the end of active queue.
events	Perform communication of VHDL signal values when events occur on signals.
hipri	Sync at the end of the high-priority queue.
immed_ca	Sync after all continuous assignments have been performed, but before their fanouts have taken effect.
inactive	Sync at the end of inactive queue.
looplimit=***n***	If value = ***n***, the synchronization will be relooped a maximum of n times.
nba	Sync at the end of the Verilog non-blocking assignment event queue.
observed	Sync at the end of Observed queue.
preponed	Sync at the end of Preponed queue.
postponed	Sync at the end of Postponed queue.
reactive	Sync at the end of Re-active queue.
reloop	Reloop the synchronization when needed.
renba	Sync at the end of Re-NBA queue.
synch	Sync at the point after all PLI read/write events are completed.
systemc	Sync at the end of the SystemC queue.

Table 3-2. SyncControl Alias Names — Quick Reference

Alias	Description
verilog	Enables the synch and nba sync controls.
verilog_conservative	Enables the active, synch, and nba sync controls.
verilog_aggressive	Enables only the nba sync control.
vhdl	Enables the active and hipri sync controls.
vhdl_aggressive	Enables the active, hipri, and events sync controls.
sv_conservative	Enables the reactive, renba, postponed, preponed, and observed sync controls.

- -mc2vsimargs="<partition>" "<args>"

  Specify additional vsim options, <args>, for a particular partition. By default, vsim command line options are applied to all partitions. This argument can be used to apply specific vsim options to a particular partition. You can group multiple space-separated vsim arguments within the quotation marks.:

  ```
  -mc2vsimargs=p1 "-l users.log -wlf users.wlf"
  ```

- -qwavedb=+wavedir=<*dirname*>

  (optional) Creates the waveform database file, *qwave.db*, in the current working directory. The waveform database file is required to run Mentor Visualizer Debug Environment.

  +wavedir —Creates a dataset which is read into Visualizer.

  <*dirname*> —A user-specified name for the dataset.

- -vv

  Print verbose multi-core simulation command line information.

mc2perfanalyze

The mc2perfanalyze command provides a variety of ways to obtain a report on performance of multicore simulation.

Syntax

mc2perfanalyze
<filename> [-help | --help | -h] [-info] [-time] [-cmd] [-data]
[-event] [-port] [-inst] [-pattern] [-analyze] [-list] [-part <from_partname>]
[-topart <to_partname>] [-sortby <column_name>] [-sortorder <asc|desc>]
[-limit <n>] [-wlf2mdb]

Arguments

- -help | --help | -h

  Prints usage information.

- -info

  Prints the connectivity info between partitions.

- -time

  Prints the Exec. Time (comparison table) for all partitions.

- -cmd

  Prints the Command exchange summary (comparison table) for all partitions.

- -data

  Prints the Data traffic summary (comparison table) for all partitions.

- -event

  Prints the event (comparison table) for all partitions.

- -port

  Prints the port trigger information for every pair of communicating partitions.

- -inst

  Prints the boundary instance trigger information for every pair of communicating partitions.

- -pattern

  Prints the boundary instance concurrency information for every pair of communicating partitions.

- -analyze

  Prints a summary analysis report for the collected data.

- -list

  Prints the output in list format instead of comparison tables for options -time, -cmd, -event.

- -part <from_partname>

  Prints information specific to this partition.
- -topart <to_partname>

  Prints information specific to the partition pair specified by -topart and -part.
- -sortby <column_name>

  Sorts the tables as per the specified column name, default is table specific.
- -sortorder <asc|desc>

  Specifies the sort order for the sort by column, default is descending order.
- -limit <n>

  Limit on the number of lines to be printed, applicable to -port, -inst, -pattern.
- -wlf2mdb

  Produces a *unisimdata.db* file containing the port and instance trigger data, and a vsig.do file containing virtual signal expressions useful for viewing the WLF in the waveform window.

Synchronization Control

The mc2com command creates a default set of sync points based on its analysis of the design. To override these default sync points, you can use the vsim -mc2synccntl command.

The controls (whether default or user-specified) that override the default sync points provided by vsim -mc2synccntl are shown in a message during vsim execution. For example:

```
# ** Note: (vsim-8840) The following
MC2 sync controls will be used : verilog.
```

Syntax

vsim -mc2synccntl=<sync_cntl>[,<sync_cntl>...]

where <sync_cntl> is comma-separated list of the named events listed below. For example:

vsim -mc2 -mc2synccntl=active,vhdl

The values for named events you can specify for synchronization points are described below.

- active — Perform sync at the end of active queue. This value is very costly and should be used only for a design that is highly sensitive to the number of propagations within a time step. Most designs do not require this control to be enabled in order to simulate correctly.
- events — Perform communication of VHDL signal values when events occur on signals. Communication of VHDL signal values will usually occur when signals at port

boundaries have activity. A common occurrence of activity that occurs without a value change is when a resolved signal has multiple drivers, a value of one of the drivers changes, but the resolved value remains unchanged. Use of the events sync control will cause communication of signal values only when events occur on signals, rather than just activity. Unless a port on a partition boundary is connected to a signal that has the 'quiet, 'transaction, or 'active attributes, using the events sync control may reduce communication without affecting simulation accuracy.

- hipri— Sync at the end of the high-priority queue. This is first the possible synchronization point for VHDL events.
- immed_ca — Sync after all continuous assignments have been performed, but before their fanouts have taken effect.
- inactive — Sync at the end of inactive queue.
- looplimit= — Specify a maximum number of times to reloop synchronization. If value = ***n***, the synchronization will be relooped a maximum of ***n*** times. By default, the reloop limit is set to match the simulator's delta iteration limit, which prevents infinite looping during simulation.
- nba — Sync at the end of the Verilog non-blocking assignment event queue.
- observed — Sync at the end of Observed queue.
- preponed — Sync at the end of Preponed queue.
- postponed — Sync at the end of Postponed queue.
- reactive — Sync at the end of Re-active queue.
- reloop — Reloop the synchronization when needed. When enabled, if data is exchanged between any partitions at any of the above sync points and new events are generated in a higher-priority queue, synchronization will "reloop" to the first synchronization point. By default, the reloop sync control is enabled.
- renba — Sync at the end of Re-NBA queue.
- synch — Sync at the point after all PLI read/write events are completed.
- systemc — Sync at the end of SystemC queue.

Sync Control Aliases

The following aliases for named events specify several Sync Controls and are provided for your convenience.

- verilog — This alias should be used with Verilog partition boundary. I It enables the synch and nba sync controls.
- verilog_conservative — This alias should be used with Verilog partition boundary and when design partitions need tighter synchronization, for example, situations like 0-delay

loop across partitions. This alias is *more expensive* than the Verilog sync control as it does more fine-grained synchronizations. This alias enables the active, synch, and nba sync controls.

- verilog_aggressive — This alias should be used with Verilog partition boundary and when the design is highly synchronous and does not need fine-grained synchronizations. This control should be used with care, and its usage should be re-evaluated if you re-partition the design or if there are major changes in the design itself. This alias will produce faster results than other Verilog sync controls. This alias enables only the nba sync control.
- vhdl — This alias performs synchronization for VHDL design units at partition boundaries. It enables the active and hipri sync controls.
- vhdl_aggressive — This alias performs synchronization of VHDL design units at partition boundaries and only when the design does not have activity sensitive attribute usage (for example, 'quiet, 'transaction, or 'active). This alias enables the active, hipri, and events sync controls.
- sv_conservative — This alias is required when cross-partition communication is sensitive to SystemVerilog scheduling regions. It will turn on all SystemVerilog fine-grained controls. You can also choose to use few individual controls, as per your design construct usage. This alias enables all the SystemVerilog-specific sync points: reactive, renba, postponed, preponed, and observed.

You need to use minimum set of sync controls required for correct simulation. These set of controls are governed by partition's boundary HDL language and also by design HDL language. Excessive usage can badly hurt the performance, while inadequate usage can result in incorrect behavior. So, it is important to use right set of sync controls.

- For Verilog boundary, use the verilog sync control alias.
- For VHDL boundary, use the vhdl sync control alias. There are other variants of these sync control aliases as described in the Sync Control Aliases section.
- For mixed-HDL design, you may need both of the verilog and vhdl sync control aliases. Most of the time, the verilog control alias is sufficient, unless you are partitioning at VHDL boundaries. Sometimes, the vhdl control alias is required for mixed-HDL designs, such as when design requirement is to produce delta-accurate results (This translates to more fine-grained synchronization and hence use of the vhdl control alias).

Exact sync controls that a design would require is mainly governed by how partition boundaries are driven by HDL design constructs and how they fan-out to other HDL constructs. A good way to find out required sync controls for a design is to start using sufficient controls required for partition boundary's HDL. If you are not able to produce correct results, add more sync controls according to the design's HDL.

Once you get the desired simulation behavior with a set of sync controls, and if you are interested in further speeding up simulation, try to reduce fine-grained sync controls (or use aggressive controls). Take care when using aggressive controls—they may not produce the same behavior over time, such as when your design is changing heavily or if partitioning is changing.