cādence®

Conformal Smart LEC Quick Start

Product Version 23.2
October 2023

Contents

1 Smart Instance Selection

Overview

Smart LEC provides an analytic engine that can select higher quality modules for hierarchical comparisons to minimize the overhead for module comparison.

Instance selection is based on the following design characteristics:

- Primitive gate counts
- RTL/netlist design
- Datapath types
- Boundary complexity
- Periphery datapaths

Smart Instance Selection Flow

To enable smart instance selection in a hierarchical comparison, use the `set_hier_compare_selection -smart` command option before the `analyze_hier_compare` or `write_hier_compare_dofile` command.

Note: Do not set a custom `-threshold` through `analyze_hier_compare` or `write_hier_compare_dofile` command. Smart instance selection works best with the default threshold.

For example:

```
// Read in the library and design files
read_design rtl.v -golden
read_design netlist.v -revised
...
// Apply constraints
...

// Enable smart instance selection before writing out
// the hierarchical compare dofile
set_hier_compare_selection -smart
write_hier_compare_dofile hier.do -balanced_extraction \
  -replace -usage -constraint -noexact_pin_match -function_pin_mapping


// Run the hierarchical comparison
go_hier_compare hier.do
```

A log similar to the following will be displayed after analysis:

```
// Command: set_hier_compare_selection -smart
// Command: write_hier_compare_dofile hier.do -balanced_extraction \
-replace -usage -constraint -noexact_pin_match -function_pin_mapping
...
// Skip due to automatic instance selection
// Module Pair: Foo(G) and Foo(R)
// Instance Pair: Qoo/foo(G) and Qoo/foo (R)
...
110 module pairs are not written because of mismatched ports
110 modules have mismatched pin unreachables
867 module pairs are written for hierarchical comparison
Auto-instance selection analysis:
250 instances contain many primitive gates
293 instances contain many datapath elements
24 instances contain multiplier
0 instances have a boundary with high similarity and high complexity
207 instances have a boundary with low similarity and high complexity
2 instances contain several complex datapath elements
190 instances contain many adders
10 instances contain many xor gates
// Command: go_hier_compare hier.do
```

set_hier_compare_selection Command Syntax

```
set_hier_compare_selection
     [-NOSMART | -SMART]
```

`-NOSMART`	Disables smart instance selection.
`-SMART`	Enables smart instance selection. LEC analyzes the design characteristics to select which modules to execute in order to leverage between compare complexity and runtime overhead.

2 Distributed Hierarchical Comparison

Overview

Distributed hierarchical comparison is available only with Conformal_Smart_LEC_4CPU.

Smart LEC offers distributed hierarchical comparison, where modules can be compared in parallel and over multiple computation resources. For a design with an evenly distributed module complexity, performing the hierarchical comparison this way can significantly reduce the turnaround time.

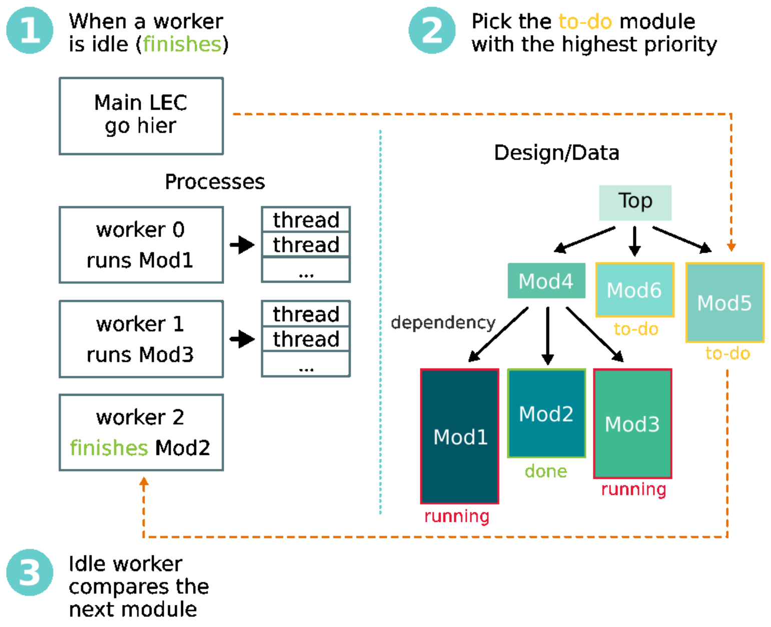
Distributed hierarchical comparison is most effective when comparing designs:

- that are large and have many submodules
- have balanced hierarchies
- whose datapath and/or complex modules are isolated to leaf modules
- where no sub-module comparison dominates run time
- where the top module comparison has few or no dynamic flattening iterations
- and there are sufficient CPU and memory resources

To achieve parallel efficiency, LEC uses *workers* to execute the module compare. A worker is an LEC process used for module comparisons. Workers can be invoked on the localhost or on remote machines.

In a run, workers collaborate with each other to compare the modules in parallel. When switching between the different modules, workers quickly set the target module and perform verification without reloading the design data. Workers can use all LEC multi-threading features, including compare and module datapath analysis (MDP), to achieve massive parallelization.

The following diagram shows that a worker which completed a comparison immediately picks the next available module with the highest priority to compare. In this diagram, the relative size of the module(s) represent complexity.



Distributed Hierarchical Comparison Flow

The following is the distributed hierarchical comparison flow; it is very similar to a dynamic hierarchical comparison, but steps 2 and 5 apply only to the hierarchical comparison flow:

1. Read in libraries and designs
2. Specify renaming rules and user constraints, if needed
3. Configure resource settings using `set_parallel_option` command options**New**
4. Generate a hierarchical compare dofile using the `write_hier_compare_dofile` command
5. Execute the hierarchical compare dofile using the `go_hier_compare` command**New**

For example, a dofile for a distributed hierarchical comparison flow would look like:

```
read_library lib.v -both
read_design Golden.v -verilog -golden
read_design revised.v -verilog -revised
add_pin_constraint 0 scan_en -revised
add_renaming_rule hier_rule1 "%s_%d$" "@1" -module -revised
set_parallel_option -workers localhost localhost localhost localhost
write_hier_compare_dofile hier_compare.do -constraint ... -replace
go_hier_compare hier_compare.do
```

Configuring Resource Settings

Before generating the hierarchical compare dofile, the following resource settings are required to define how LEC will start the worker processes.

These settings are done through the following options of the `set_parallel_option` command. This command is a general LEC command that sets the parameters for parallel processing, but the `-workers`, `-cluster`, and `-batch` options are only available with Smart licenses.

- `-workers <localhost | batch>:` Specifies the number of workers to use for the comparison and whether to launch them to the local machine or to a cluster of remote machines.
- `-threads <integer>[,<integer>]`: Specifies the minimum and maximum number of threads to use for multithreaded processing. If only one number is used, this specifies both the minimum and maximum.
- `-cluster <LSF | SGE>`: Specifies to run the comparison on a cluster of remote machines and specifies the cluster type. LSF is for load sharing facility clusters and OpenLava, and SGE is for Sun grid engine clusters. LSF is the default.

- `-batch_command <command>` or `-rsh_command <command>`: Specifies the command to submit jobs to the cluster.

Resourcing Workers on the Current Machine

To run the comparison using your local machine, use the `-workers` option with the keyword "localhost" and specify it for however many workers you want to assign. For example:

```
set_parallel_option -threads 4
set_parallel_option -workers localhost localhost localhost localhost
```

The above specifies that the parallel hierarchical comparison will use 4 threads and 4 workers on the current machine.

Resourcing Workers from a Cluster

To run the comparison on a cluster of remote machines, use the `-workers` option with the keyword "`batch`" and specify it for however many workers you want to assign to the cluster, specify the cluster type using the `-cluster` option, and the batch command to submit jobs using the `-batch` option. For example:

```
set_parallel_option -threads 4
set_parallel_option -workers  batch batch batch batch
set_parallel_option -cluster LSF
set_parallel_option -batch_command \
 \"/farm/bin/bsub -q super -R 'OSNAME==Linux && SFIARCH==EM64T'\"
```

The above specifies that the parallel hierarchical comparison will use 4 threads and 4 workers from an LSF cluster.

You can mix `localhost` and `batch` in worker setting. For example:

```
set_parallel_option -workers batch batch localhost localost batch localhost
```

Viewing Current Resource Settings

To report the current resource settings, use the `set_parallel_option` command without any options.

```
// Command: set_parallel_option
Current parallel processing options: ('*' indicates non-default value.)
================================================================================
Keep directory     :   NO

Multithreaded processing options:
--------------------------------------------------------------------------------
Number of threads  :   0,0
Starting port      :   50100
License list       :   SL4
Hold license       :   NO
```

```
Distributed parallization options:
------------------------------------------------------------------------------
Workers           :   localhost localhost localhost localhost
Cluster type      :   LSF
Batch command     : * bsub -q lnx64
Rsh commmand      :   rsh
....
```

Executing the Hierarchical Comparison Dofile

Once the resource settings have been configured and a hierarchical compare dofile has been generated, you can perform the distributed comparison using the `go_hier_compare` command.

The following illustrates the sample logfile of a dynamic hierarchical compare:

```
// Command: go_hier_compare x.do
....
Temp directory was create at /home/.../LEC.tmp.5833
// Command: APP saveenv /home/.../LEC.tmp.5833/gohier.0/env.csh
// Command: checkpoint /home/.../LEC.tmp.5833/gohier.0/ckp -rep
...
// Note: Checkpoint file /home/.../LEC.tmp.5833/gohier.0/ckp saved successfully
...
// Command: submit hier -dir .... (This is internal command)
// Command: !cfm_env gohier ....  (This is internal command)
// Finish loading 4 workers
// Job1 (G) sub and (R) sub1 starts on worker 0
// Job0 (G) sub and (R) sub0 starts on worker 1
// Job1 (G) sub and (R) sub1 finished
// Job2 (G) sub and (R) sub2 starts on worker 2
// Job0 (G) sub and (R) sub0 finished
// Job2 (G) sub and (R) sub2 finished
// All remote run finish(100%)
// Workers exit
// Comparing top top (G) and top(R)
....
```

go_hier_compare Command Syntax

The following is the `go_hier_compare` command syntax.

```
go_hier_compare
     <dofile_name>
     [-CHECK_NONEQ]
     [-DYNamic_hierarchy | -NODYNamic_hierarchy]
     [-NOANALYZE_abort | -ANALYZE_abort]
     [-NORESTart | -RESTart]
     [-RETIMED_modules [-TOP | -NOTOP]]
     [-ROOT_module <golden_module> <revised_module>]
     [-VERBOSE]
     (Setup Mode)
```

`<dofile_name>`	Specifies the name of the hierarchical dofile that was generated with the `write_hier_compare_dofile` command.
`-CHECK_NONEQ`	Identify NEQ modules before running the `analyze_datapath` command and skip datapath analysis for these modules.
`-DYNamic_hierarchy`	Auto-flattens the submodules to propagate any design errors to the top level. The flattened modules are merged to the next level in the hierarchy and compared at that level. *This is the default*. When using this option, -`noneq_stop 1` is automatically appended to the `compare` command during each module comparison.
`-NODYNamic_hierarchy`	Runs static hierarchical comparison without auto-flattening the submodules.
`-NOANALYZE_abort`	Appends "`-abort_stop 1`" to the `compare` command during each module comparison. *This is the default*. However, if you use the `set_analyze_option -auto` command, it will override this option.
`-ANALYZE_abort`	Inserts the `analyze_abort -compare` command into each uncompared and aborted module's compare script. If this option is used, the `compare` command is not appended with "`-abort_stop 1`" during each module comparison.

`-NORESTart`	Continues an interrupted session, preserving the previous compare results. *This is the default*. You can interrupt dynamic hierarchical comparison by pressing `Ctrl-c`.
`-RESTart`	Deletes the previous comparison results.
`-RETIMED_modules [-TOP | -NOTOP]`	
	Compares and blackboxes the submodules with the `pipeline_retime` attribute. The `pipeline_retime` attribute can be attached to a module using the `add_module_attribute` command. For this option to work correctly, modules with `pipeline_retime` attribute should exist in the hierarchical dofile script. `-top` runs the comparison of the top module such that submodules without the `pipeline_retime` attribute are fully flattened. This is the default for `-retimed_module`. `-notop` specifies that comparison stops after the modules with the `pipeline_retime` attribute have been compared and blackboxed. The hierarchical result is reported as 'Inconclusive' because the entire design is not compared.
`-ROOT_module <golden_module> <revised_module>`	
	Uses the specified modules as the root modules. This is similar to the `-module` option with the `write_hier_compare_dofile` command without having to the regenerate the dofile.
`-VERBOSE`	Lists all the hierarchical constraints and additional information.

3

Smart Compare

Overview

Smart Compare is done through the `analyze_compare` command. This command executes the entire comparison step in the most optimal turnaround time. Based on the design's characteristics, `analyze_compare` automatically executes the most appropriate combination of commands and options to complete the comparison.

Smart Compare Flow

Flat Comparison

To use Smart Compare in a flat comparison flow, merely use the `analyze_compare` command to execute the comparison. When using `analyze_compare`, the `analyze_datapath` commands are no longer necessary as the `analyze_compare` command will execute the best datapath options and strategies automatically.

For example, the following illustrates a sample dofile for running a flat RTL versus synthesized gate netlist comparison:

Classic LEC Compare Flow

```
//Read in the library
//and design files
read_design rtl.v -golden
read_design netlist.v -revised
...

//Flatten, remodel, and map the design
set_system_mode lec

//Enable auto setup analysis
analyze_setup -verbose

//Enable datapath analysis
analyze_datapath -module -resource \
  resouce.rpt -verbose
analyze_datapath -verbose

//Run the comparison
add_compare_point -all
compare
```

Smart Compare Flow

```
//Read in the library
//and design files
read_design rtl.v -golden
read_design netlist.v -revised
...

//Flatten, remodel, and map the design
set_system_mode lec

//Enable auto setup analysis
analyze_setup -verbose

//Run the comparison
add_compare_point -all
analyze_compare -resourcefile \
  resource.rpt -verbose
```

LEC will output the summary of the datapath quality (if it invoked datapath analysis), and the summary of compare results.

```
// Command: set_system_mode lec
...
```

```
// Command: add_compared_points -all
// 64 compared points added to compare list
// Command: analyze_compare
// Note: mult_66: quality evaluated 76% success
...
// Note: mult_78: quality evaluated 100% success
=========================================================================
Compared points      PO     DFF       Total
-------------------------------------------------------------------------
Equivalent           32     32        64
=========================================================================
```

Note that if `analyze_partition` is invoked, LEC will also output messages of every iteration in the partition run.

Hierarchical Comparison

To use Smart Compare in a hierarchical comparison, use `write_hier_compare_dofile -compare_string` or `analyze_hier_compare_dofile -compare_string` to replace the default `compare` command with the `analyze_compare` command. The following is a sample dofile for an RTL-to-synthesized gate netlist hierarchical comparison using `analyze_compare`.

Genus Synthesized Netlist

```
// Reads in the library, design, and resource files
read_design rtl.v -golden
read_design netlist.v -revised
read_implementation_information fv/top -golden fv_map -revised map_lecv
...
// Enables auto setup analysis
set_analyze_option -auto

// Creates the hierarchical dofile script that will compare the designs
write_hier_compare_dofile hier.do -compare_string "analyze_compare -verbose"
go_hier_compare hier.do
```

or

```
// Reads in the library, design, and resource files
read_design rtl.v -golden
read_design netlist.v -revised
read_implementation_information fv/top -golden fv_map -revised map_lecv
...
// Enables auto setup analysis
set_analyze_option -auto
set_flatten_model -enable_analyze_hier_compare
set_system_mode lec

// Creates the hierarchical dofile script that compares the designs
analyze_hier_compare -dofile hier.do -compare_string "analyze_compare -verbose"
go_hier_compare hier.do
```

DC Synthesized Netlist

```
// Reads in the library design files
read_design rtl.v -golden
read_design netlist.v -revised
...

// Enables auto setup analysis
set_analyze_option -auto

// Creates the hierarchical dofile script that will compare the designs
write_hier_compare_dofile hier.do -compare_string \
  "analyze_compare -verbose -resourcefile resourcefile.name"
go_hier_compare hier.do
```

or

```
// Reads in the library design files
read_design rtl.v -golden
read_design netlist.v -revised
...

// Enables auto setup analysis
set_analyze_option -auto
set_flatten_model -enable_analyze_hier_compare
set_system_mode lec

// Creates the hierarchical dofile script that compares the designs
analyze_hier_compare -dofile hier.do -compare_string \
    "analyze_compare -verbose -resourcefile resourcefile.name"
go_hier_compare hier.do
```

analyze_compare Command Syntax

```
analyze_compare
     [-RESOURCEFILE <filename>]
     [-THREADS <integer>[,<integer>]]
     [-NONEQ_Stop <integer>]
     [-VERbose]
```
(LEC Mode)

`-RESOURCEFILE <filename>`	Specifies the resource filename to analyze the datapath modules. This is used for DC netlists.
`[-THREADS <integer>[,<integer>]]`	Specifies the minimum and maximum number of threads. If `-threads` is not specified, LEC honors the setting from '`set_parallel_option -threads`'.
`-NONEQ_STOP <integer>`	Stops after finding the specified number of nonequivalent points.
`-Verbose`	Provides additional information.

4

Smart Dofile Template

```
read_library ...
read_design ...
....

#*****************************************************************
# Specify the parallel hier compare workers here
#*****************************************************************
# set_parallel_option -cluster [LSF | SGE]
# set_parallel_option -workers [localhost | batch |  ...]
# set_parallel_option -batch_command "..."
# set_parallel_option -rsh_command "..."


#*****************************************************************
#* Enable smart instance selection
#*****************************************************************

set_hier_compare_selection -smart

#*****************************************************************
#* Generate the hierarchical dofile script for
#* hierarchical comparison
#*****************************************************************
write_hier_compare_dofile hier.do -replace -usage \
  -constraint -noexact_pin_match -verbose \
  -prepend_string "report_design_data; usage;" \
  -compare_string "analyze_compare -verbose"   \
  -balanced_extraction -input_output_pin_equivalence \
  -function_pin_mapping

#*****************************************************************
# Executes the hier.do script with parallel hier compare
#*****************************************************************

go_hier_compare hier.do  -verbose
# run_hier_compare hier.do  -verbose

#****************************************************************
#* Generates the reports for all compared hierarchical modules
#****************************************************************

report_hier_compare_result -all -usage
report_hier_compare_result -abort -usage
report_hier_compare_result -noneq -usage
report_verification -hier
```