SYNOPSYS®
Accelerating Innovation

PrimeTime® Mode Merging

Reducing Analysis Cost for Multimode Designs

August 2013

Author
Ron Craig
Technical Marketing Manager, Synopsys

Introduction

As process technologies shrink, design teams can fit increasing amounts of logic in a single chip, combining functionality that was captured in the past by discrete devices. This increased functional complexity often accompanies more configurability, allowing portions of the logic to be selectively enabled, depending on the task being executed. Examples include a mobile chip that disables graphics processing logic when the display is not in use, or a multiprocessor configuration that can enable specific cores to complete a given processing task. Figure 1 shows the increase in modes and the corresponding increase in scenarios at smaller nodes.

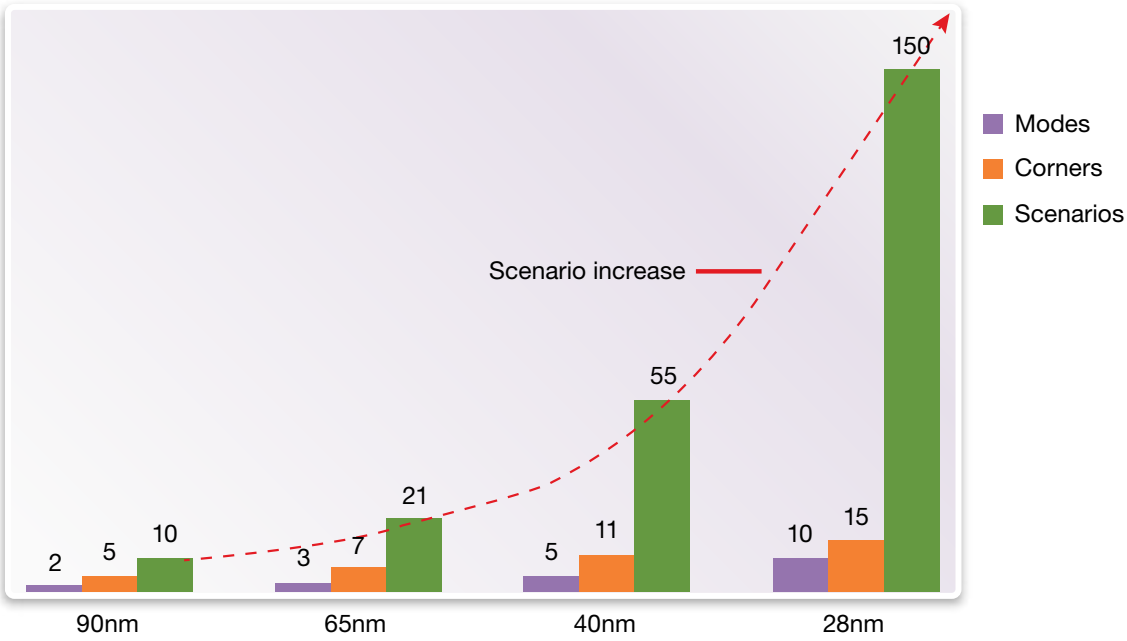


Figure 1: Mode and Scenario Increases

Faced with this increasing complexity, design teams must ensure that the chip meets its performance objectives in each different functional configuration. In addition to functional modes, it is common to have several test modes plus other modes designed to verify specific timing or I/O conditions.

For signoff purposes, these different configurations are typically represented as timing modes and are captured via timing constraints. Developing and managing these various modes individually has the following benefits:

- Parallel development – allowing the design and test teams to develop their own modes independently, rather than trying to figure out how to combine them
- Opportunity for reuse – especially for design-independent modes covering functions like test
- Fine-grained control over the signoff process – analysis can focus on specific timing-critical mode and corner combinations requiring late-stage closure effort

PrimeTime mode merging technology retains the benefit of writing individual modes, while providing an automated path to generate superset modes to minimize ECO and timing closure turnaround time. This avoids the risk and complexity of manual mode merging, while keeping the runtime and resource reduction benefits of merged modes.

Modes and Their Impact on Timing Closure

Design teams need to confirm that a given chip functions as expected under all possible configurations. Completing timing closure requires the following tasks:

- Configuring modes and setup timing analysis to take account of those modes
- Managing timing analysis runs
- Reviewing the results of multi-mode timing analysis

Management of timing closure is of particular concern; an increasing number of modes implies more runs, and hence requires either longer overall turnaround time, more hardware resources, or a combination of both. Figure 2 highlights how the number of modes in a design affects the number of ECO iterations needed to close timing. Designs with fewer than ten modes typically need fewer than ten ECO iterations, but designs with more than ten modes are much more likely to see more than ten iterations.

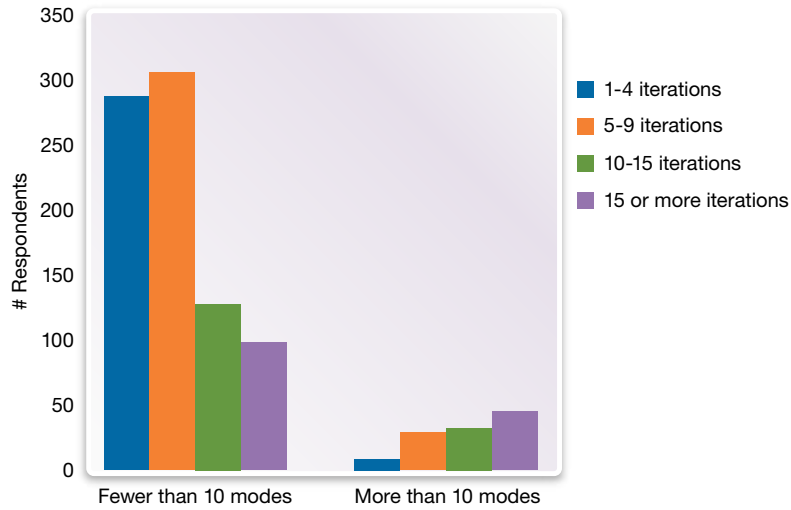


Figure 2: Effect of Number of Modes on Number of ECO Iterations Required to Close Timing

For final signoff, design teams usually perform analysis across all mode and corner combinations. This provides complete visibility into the conditions under which any timing issues occur, allowing them to trace the source of a specific violation and determine whether it can be safely waived.

Typical Approaches to Dealing With Mode Complexity

The choice between comprehensive analysis and fast turnaround

Design teams working with multimode designs are faced with conflicting demands – they need to complete comprehensive signoff while working within the constraints of their hardware resources and tape-out deadline. Each different mode potentially requires a separate timing analysis run, which requires more time and resources. When the finite limits of time and resources are reached, compromises must inevitably be made. Let's consider some typical compromises and the pros and cons of each.

Approach #1: Select the worst-case modes

When faced with an increasing number of modes, design teams often select the most important modes, the worst-case modes, for timing analysis. If the worst-case modes meet timing, other configurations would also meet timing.

Using a worst-case subset of the timing scenarios certainly reduces the number of runs, but this selective approach is not guaranteed to catch all of the worst timing violations.

Approach #2: Manual mode merging

Another approach taken by design teams is to manually merge selected modes. Manual mode merging typically requires expertise in both the design and timing constraints, and can therefore become a time-consuming joint effort between design owners and signoff timing engineers. To ensure correctness of the merged constraints, the merging process should be repeated every time there is a change to the design or original constraints – further increasing its impact on signoff schedules.

Approach #3: Add more hardware

A more popular option is to take advantage of parallel computing techniques. If your resources allow you to run n jobs in parallel, maintaining a fixed turnaround time requires n machine and license resources. If the next chip has twice as many modes as the last chip, with all other things being equal, you will need double your computing resources to complete analysis and timing closure in the same amount of time. Logistically, this may be the easiest way to address the problem of mode coverage, but it has an associated cost and does not help reduce the effort of reviewing the analysis results across all of the different modes.

Approach #4: Relax the project schedule

If the task cannot be parallelized by adding more compute resources, the increase in modes results in increased turnaround time. Remember that an increase in modes inevitably means an increase in scenarios (where a scenario is a combination of a mode with a process corner), so adding a mode does not simply add a run; it adds a number of runs equal to the number of corners. The new runs are run in serial on the available resources. In most situations, relaxing the project schedule incrementally is not a viable option.

The following sections describe the PrimeTime approach to mode merging, and how its architecture avoids the compromises outlined above.

PrimeTime Mode Merging Technology

PrimeTime mode merging technology identifies superset modes that together replicate how a given design is constrained by the original individual mode constraints. The example in Figure 3 shows nine original modes merged into three superset modes (M1, M2, and M3).

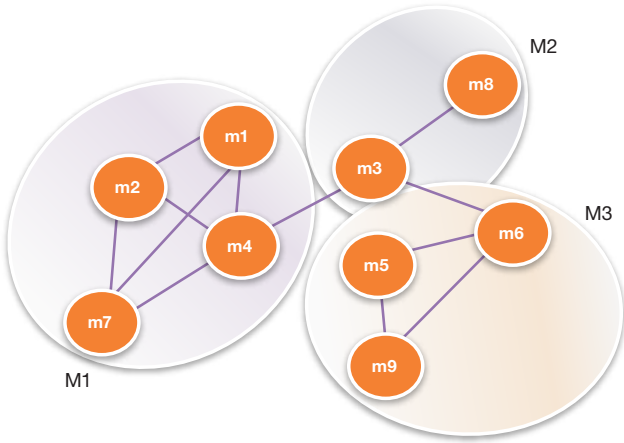


Figure 3: Mode Compatibility Graph

In Figure 3, a line joining two modes (for example, m2 and m7) indicates that they can be merged. A 'superset' is a group of modes that can all be merged with one another. Notice how modes m3 and m8 can be merged, but mode m8 cannot be merged with any other modes, so m3 and m8 are merged as part of a separate group, M2.

These merged superset modes satisfy the following criteria:

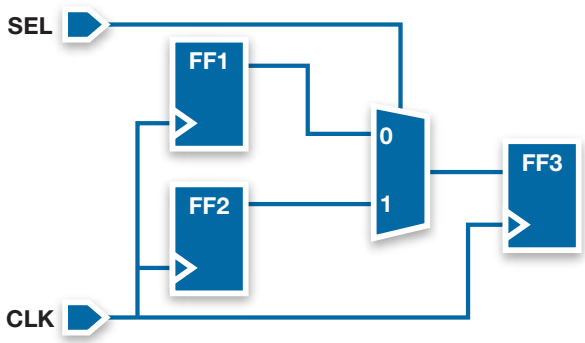
- Any timing violation identified in an individual mode is also identified when using merged mode constraints.
- If a timing path occurs in any of the individual modes, it must be present in the merged mode.
- If a timing path occurs in the merged mode, it must be present in at least one of the individual modes.

Mode merging reduces timing analysis turnaround time (TAT) without missing any timing violations.

Examples of Mode Merging

The following examples show individual modes and the outcome of the merging process. In Example 1, the modes are suitable for merging, and the merging result is provided. Example 2 shows two modes that are not suitable for merging.

Example 1: Two modes that can be automatically merged




```
# mode1.sdc
create_clock –period 5 [get_ports CLK]
set_case_analysis 0 [get_ports SEL]

# mode2.sdc
create_clock –period 5 [get_ports CLK]
set_case_analysis 1 [get_ports SEL]

# merged mode SDC
create_clock –period 5 [get_ports CLK]
set_disable_timing [get_ports SEL]
```

In this example, the merged mode does not retain the `set_case_analysis` settings in the individual modes. The merged mode constraints add a `set_disable_timing` constraint on the `SEL` input port, since it is not active in any of the original modes.

Example 2: Two modes that cannot be automatically merged


```
# mode3.sdc
create_clock –name clk_mode1 [get_ports C1] …
create_generated_clock –name gclk_mode1 \
                       [get_pins buf1/Z]

# mode4.sdc
create_clock –name clk_mode2 [get_ports C1] …
```

In this example, mode3 and mode4 cannot be merged since the gclk_mode1 generated clock in mode3 would block the propagation of the clk_mode2 clock in mode4.

Compatibility With an Existing PrimeTime Setup

Mode merging uses the PrimeTime distributed multi-scenario analysis (DMSA) infrastructure, and can easily be added to an existing PrimeTime setup for a multi-scenario design. This minimizes the effort needed to set up mode merging, and ensures that the scenario and constraint setup used for PrimeTime analysis is identical to that being used for mode merging.

The following script example shows the mode merging flow:

```
# define modes and corners
set modes {MODE1 MODE2 MODE3}
set corners {best worst}
<add DMSA-specific setup>

# create scenarios
<mode/corner combinations to create analysis scenarios>

# merge modes
<run mode merging>
```

Fast Assessment of How Well Mode Constraints Can Be Merged

In increasingly complex multi-scenario designs, it can be difficult to maintain constraint consistency between modes. Constraint additions or modifications made manually during timing analysis in one mode might not always be reflected in other modes. Since the timing behavior of such modes, if merged, could be significantly different from the original mode timing, therefore such conflicting modes are not merged.

PrimeTime mode merging technology offers an up-front assessment of how well a given set of modes can be merged – before embarking on the full mode merging process and will highlight constraint conflicts that prevent merging and avoiding timing discrepancies between merged and unmerged constraints. The assessment report also provides an example of the constraint that would need to be added to resolve the merging conflict. Revisiting Example 2, the assessment report would provide the following guidance:

```
An internal clock gclk_mode1 defined at mode3.sdc line 2 would stop
propagation of clock clk_mode2 defined at mode4.sdc line 1. Consider adding a
similar clock in mode mode4 to allow for mode merging as follows (please note
that you might have to change other constraints that refer to clock names as
well):
create_generated_clock –divide_by 1 –name clk_mode2_gen –add \
                       –master_clock clk_mode2 –source [get_ports C1]
                       [get_pins buf1/Z]
```

Using Merged Constraints

Reducing the Resource and Runtime to Manage Multimode Designs

PrimeTime mode merging technology reduces the effort required to complete timing analysis with increasing numbers of modes and scenarios. Table 1 shows a reduction in modes of approximately three times on average. This results in three times faster TAT with the same hardware resources, or the same TAT with three times fewer hardware resources.

Design size (M instances)	Number of original modes	Number of merged modes
11	13	3
3.9	9	2
1.5	9	4
1.5	11	7
0.75	8	2
0.36	13	4
23	3	1

Source: Synopsys Customer Test Cases

Table 1: Mode Merging Results

Even though the merged constraints are marginally more complex than the original constraints, the reduction in PrimeTime ECO or pure timing analysis turnaround time closely matches the reduction of modes. In Table 2, a three-times reduction in modes shows a 2.53-times reduction in PrimeTime ECO TAT. The quality of results (QoR) of both original and merged mode ECO runs is virtually identical, demonstrating that there is no price to pay for reduced ECO TAT.

	Number of original modes	Number of merged modes
Total scenarios	18	6
Available hosts	6	6
Total STA + ECO TAT	346 minutes	137 minutes
Fix rate (setup/hold)	99.1/94.6%	99.1/92.3%
WNS (setup/hold)	-5.23/-0.18ns	-5.23/-0.18ns
TNS (setup/hold)	-1201.52/-62.84ns	-1169.12/-69.73ns
Number of violating paths (setup/hold)	1255/3041	1239/3203
Area	2953853	2953628
Saved session size	28.89GB	9.19GB

Table 2: PrimeTime ECO Results With Merged Constraints

It is normal for design teams to complete signoff with all scenarios, but timing analysis runs prior to final signoff are normally completed with either worst-case scenarios or a set of scenarios that are the outcome of manual merging.

Using PrimeTime merged mode constraints during ECO iterations avoids the risk of these manual approaches in two key ways:

- The design is not overconstrained as it would be if using worst-case modes and scenarios. As a result, timing targets during ECO iterations are more achievable.
- Manual mode merging can often result in unconstrained endpoints, when subsets of clocks are selected. By definition, PrimeTime mode merging is designed to avoid this, so timing ‘escapes’ do not happen.

Figure 4 outlines how the mode assessment and merging process fit into a typical design flow. You can easily create a PrimeTime mode merging script from an existing PrimeTime DMSA setup, by adding specific mode and corner information for each scenario. The output mode constraints can then be used –within an existing DMSA setup – to drive the PrimeTime ECO step.

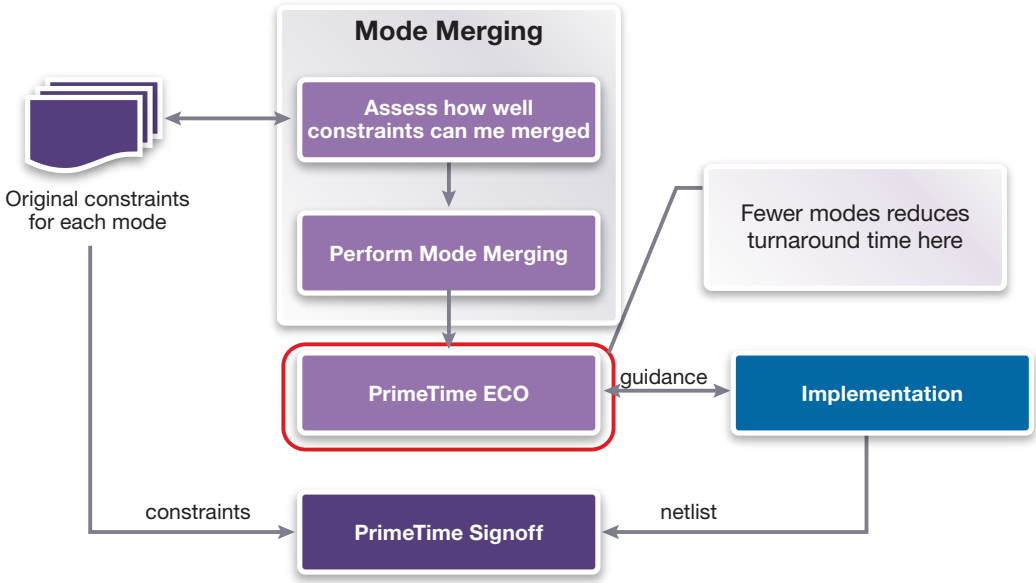


Figure 4: Mode Merging in the Design Flow

Conclusion

Scenario growth at smaller geometries poses significant risk to project schedules and budgets. At the early stages of the timing closure and signoff flow where frequent iterations are common, solutions to reduce turnaround time or hardware cost often come at the cost of accuracy. Until now, design teams could not reliably reduce scenarios while avoiding inaccuracies, which can ultimately make timing closure more difficult. PrimeTime mode merging provides actionable guidance that allows design teams to improve their mode constraints to maximize mode reduction, and ultimately addresses the goal of scenario reduction without compromising timing accuracy.