Efficient methods to optimize PrimeTime-based full flat timing signoff runtime for large SoC designs

Akhilesh Kumar Shukla
Devyani Wad
Jyothirmayee Camasamudram

Intel

Agenda

- Possibility of Full Flat Signoff for big Designs
- Design Challenges with partition/SubFCs Roll-up at FCT : Case Studies
- Full Chip Timing Runtime Challenges
- Methods to Optimize Runtime and Peak Memory Requirement
- Results, Conclusion & Future Work

Possibility of Full Flat Signoff for big Designs

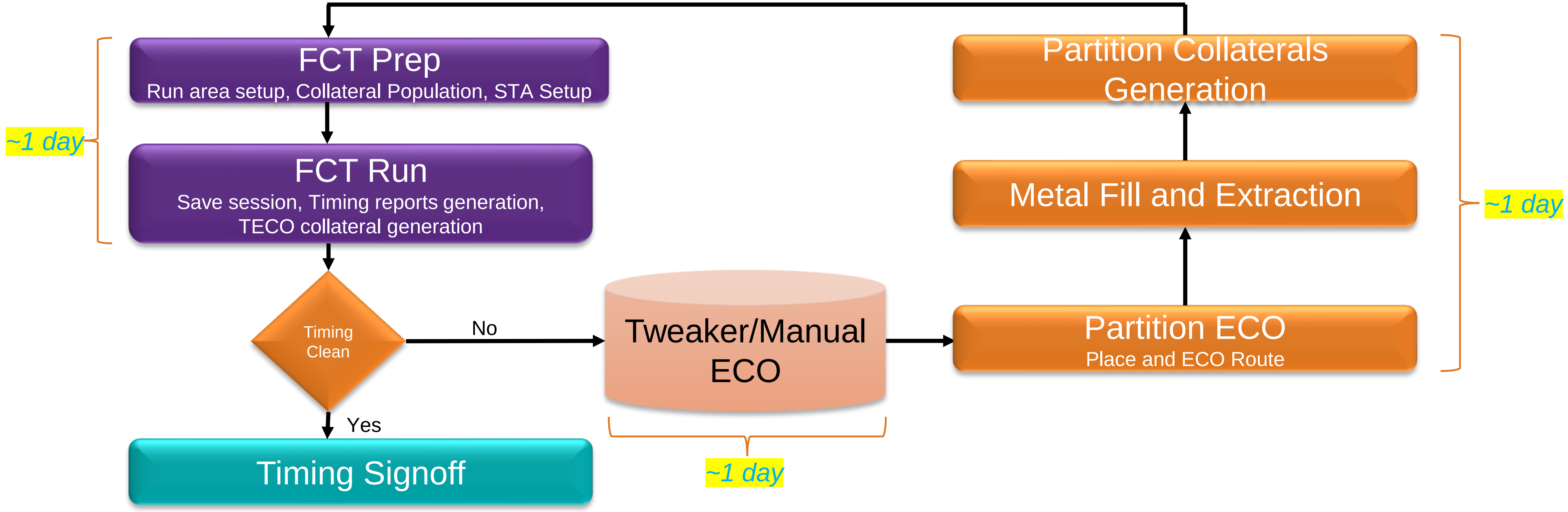
- Problem Statement
  - For Large complex SoCs, It is difficult to flatten all the hierarchies in one run, due to limited availability of bigger size compute/machines and huge runtime
    - Generally, large size SoCs follow hierarchical signoff with "**Hyperscale Scale Model**' to reduce full chip timing runtime and compute requirement
    - Need some design hand-holding/Constraint management while rolling-up partition data as "hierarchical abstraction"

- Synopsys has done lots of improvement to PrimeTime to handle big size design
  - Large SoC full flat designs can be managed with comparatively smaller compute farm with reasonably lesser runtime

SoC Runtime Target for "Full Flat" Timing Signoff

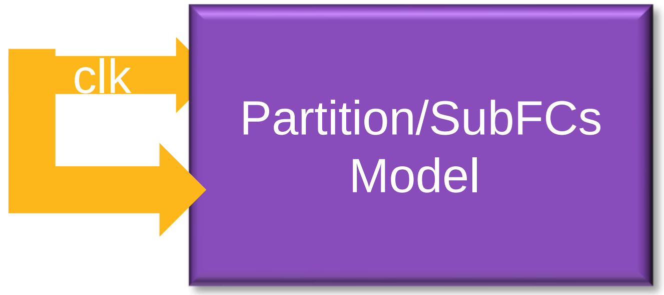
- SoC timing runtime should be ~1day and target should be to achieve minimum ~2 Timing ECO Cycles per week

- Runtime target for Full Chip Timing (FCT) including One cycle Timing ECO

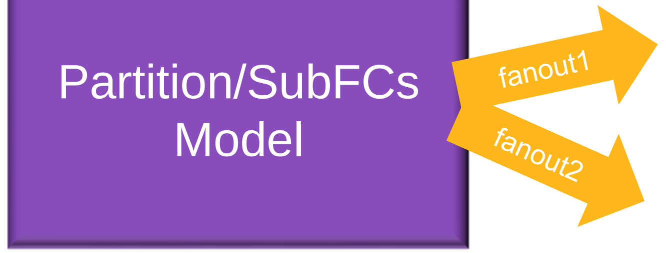


~1 day
FCT Prep
Run area setup, Collateral Population, STA Setup
FCT Run
Save session, Timing reports generation, TECO collateral generation
Timing Clean
No
Yes
Timing Signoff
Tweaker/Manual ECO
~1 day
Partition ECO
Place and ECO Route
Metal Fill and Extraction
Partition Collaterals Generation
~1 day

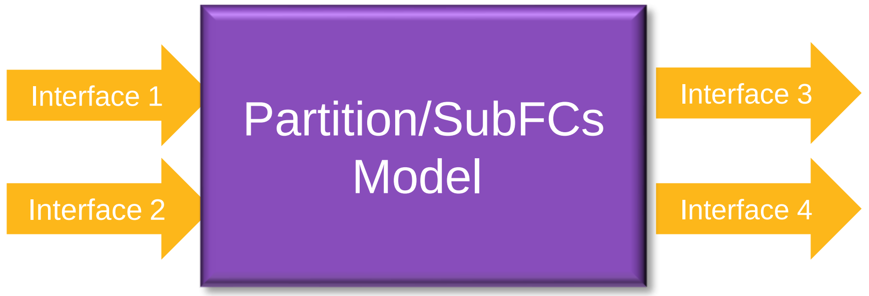
Design Challenges with partition/SubFCs Roll-up at FCT Case Study 1



- Multiple Entry points at partition level for same clock
- If partition is roll-up as hierarchical abstraction model then clock latency need to model with proper constraint in FCT

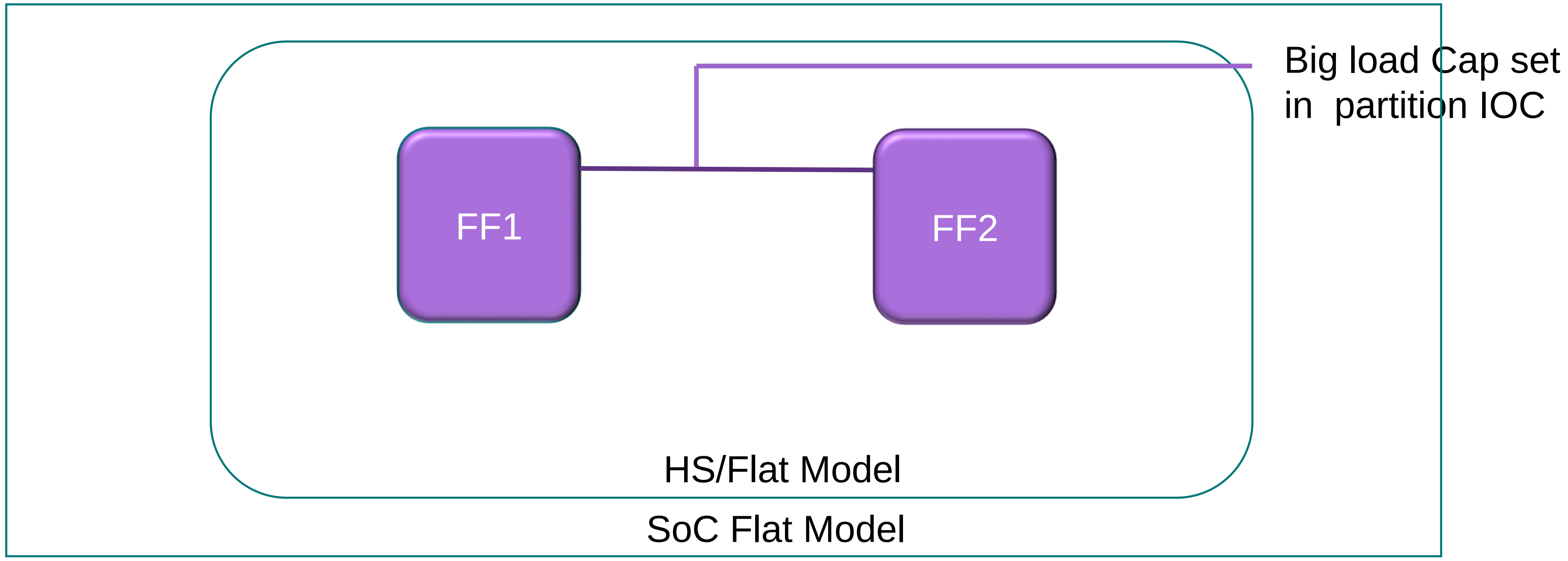


- Multiple fanout from the same port
- By default all fanouts not retained
- Need to use a special switch to retain all the fanouts



- Any variation/crosstalk delta delay in clock network outside the “hierarchical abstraction” boundary that need to be modeled with appropriate constraint to avoid any timing gaps/miscorrelation

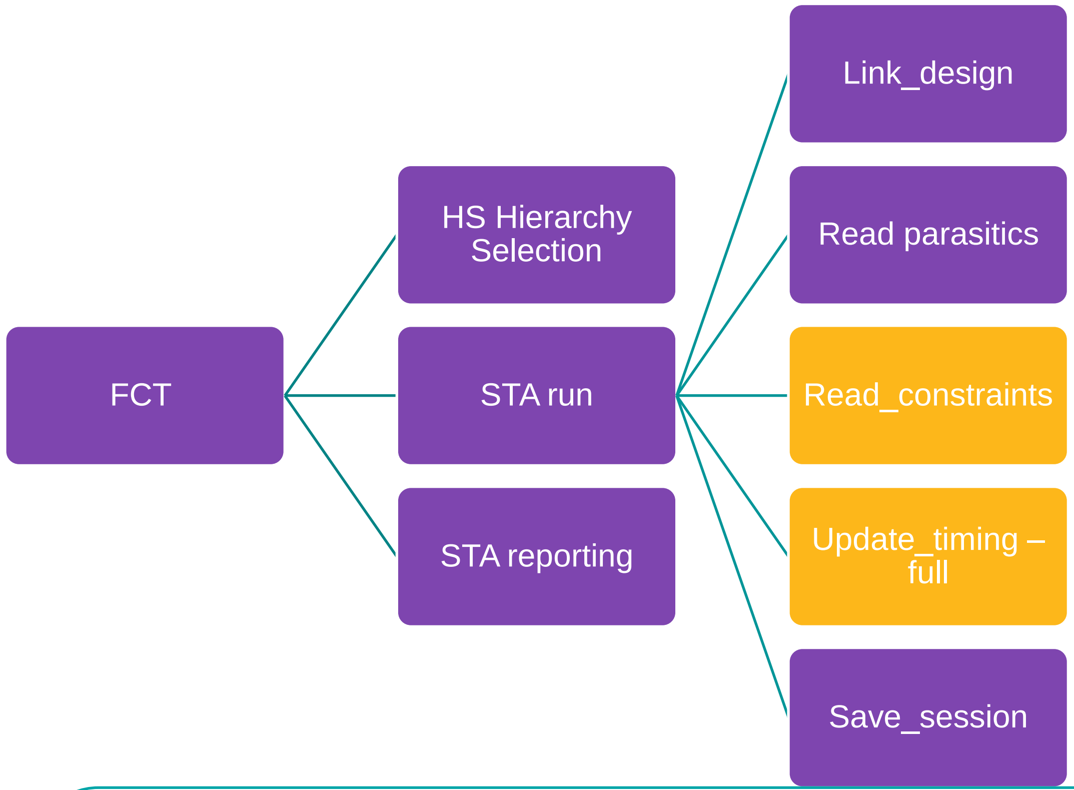
Design Challenges with partition/SubFCs roll-up at FCT Case Study 2



- Min Timing was failing when “partition" is flattened at SoC
- There was big load cap setting in “partition" IO constraint, delaying the "clk->q" of flop FF1
- No buffer was added at physical boundary at partition, hence internal flop delay calculation was dependent on external load cap set in IOC
- In flat run, real load cap seen and FF1 "clk-q" delay was faster compared to partition run and showing-up min failure

Methods to Optimize Runtime and Peak Memory Requirement

Full Chip Timing Runtime Challenges



S.No.	Flow Step	Runtime
1	link design	34 mins
2	read parasitics	38 mins
3	**read_constraints**	**16hr:50mins**
4	**update_timing - full**	**12hr:12mins**
5	save session	21m

Solutions implemented for the runtime issues

- read_constraints – Tool enhancements done by Synopsys + Constraints Optimization techniques
- update_timing –full – Tool enhancements done by Synopsys for full flat FCT runs
- These enhancements helped improve overall runtime and memory requirements of the FCT runs for both HS and full flat

Constraints Optimization : Collection Reordering

Issue

- Same collection used multiple times during read_constraints.
- Example : same set of through pins for multiple set_multicycle_paths/set_false_paths

Solution

- Generate the collection and save into a variable
- Use $variable with the different constraints

Original script	**set_multicycle_path -setup… - through …. [all_fanout -from [get_pins */*$pattern* --nocase -filter “direction==in”] …. -to [get_clocks clk*]** **set_multicycle_path -hold… - through …. [all_fanout -from [get_pins */*$pattern* --nocase -filter “direction==in”] …. -to [get_clocks clk*]**
Modified script	set pins [get_pins ….[all_fanout -from [get_pins $pattern -nocase -filter “direction==in”]…. set_multicycle_path -setup….-through $pins -start -to [get_clocks clk*] set_multicycle_path -hold ……-through $pins -start -to [get_clocks clk*]

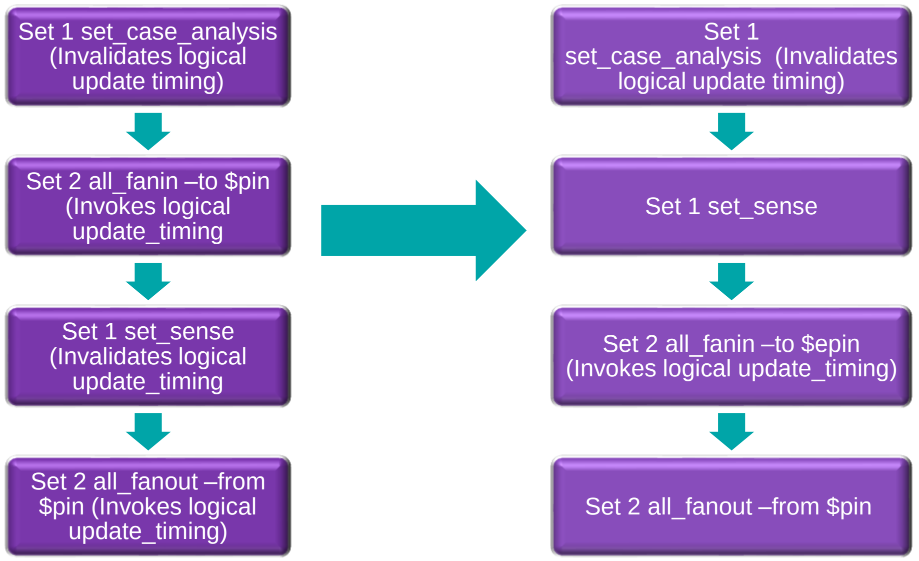
Constraints Optimization : Constraints Reordering

Issue

- 2 sets of constraints in Primetime
  - Set1 : Invalidate logical timing
  - Set2 : Update logical timing
- If constraints sourced alternately, logical timing invalidated and validated multiple times

Solution

- Reorder the constraints such that the logical update timing validation/invalidation is minimized



Set 1 set_case_analysis (Invalidates logical update timing)
Set 2 all_fanin –to $pin (Invokes logical update_timing
Set 1 set_sense (Invalidates logical update_timing
Set 2 all_fanout –from $pin (Invokes logical update_timing)
Set 1 set_case_analysis (Invalidates logical update timing)
Set 1 set_sense
Set 2 all_fanin –to $epin (Invokes logical update_timing)
Set 2 all_fanout –from $pin

Original sequence of commands	**~5-6 logical updates**
Modified sequence of commands	2 logical updates. Reducing ~3-4 logical updates giving runtime advantage

Constraints Optimization – Command Restructuring

Issue

- Few Primetime commands invalidate timer update and few invoke timer update
- These were part of foreach_in_collection loop, resulting in multiple timer updates

Solution

- One of the commands was annotating the delays on objects
- Redirected all these commands into a file and sourced the file outside of the foreach_in_collection loop

Original script	**foreach_in_collection clkpins [get_pins [*/clk] {** **set arrival [get_attribute [get_timing_paths..-through $temp_pins]arrival] → trigger timer update** **set_annotated_delay -max_delay -incr -to $object $val** **}** **Here set_annotated_delay is part of foreach loop which invalidates timer update**

Modified script	foreach_in_collection clkpins [get_pins [*/clk] {
	set arrival [get_attribute [get_timing_paths… -through $temp_pins] arrival] → Timer update once
	redirect -append -file annotation.tcl [\
	set_annotated_delay -max_delay -incr -to $object $val
	]
	}
	source -e -v annotation.tcl
	Set_annotated_delay commands written out into a file and sourced separately

Constraints Optimization – Collection Redefining

Issue

- Commands like get_pins –hier * and get_cells –hier * called at multiple places inside multiple user defined procs
- Every time these collections were invoked, they consume some runtime.

Solution

- Redefine these collections as global variables to be used anywhere in the flow

Original script

```
set var1 [get_pins –hier * -quiet]
set var2 [get_cells –hier * -quiet]
proc proc1 {
…
set var3 [get_pins –hier * -quiet]
}
proc proc2 {
..
set var4 [get_cells –hier * -quiet]
}
```

Modified script

```
set CONFIG_GET_CELLS_ALL [get_cells -quiet -hier *]
  set CONFIG_GET_PINS_ALL [get_pins -quiet -hierarchical *]
  global CONFIG_GET_CELLS_ALL
  global CONFIG_GET_PINS_ALL
set var1 $CONFIG_GET_PINS_ALL
set var2 $CONFIG_GET_CELLS_ALL
proc proc1 {
…
set var3 $CONFIG_GET_PINS_ALL
}
proc proc2 {
..
set var4 $CONFIG_GET_CELLS_ALL
}
```

Primetime Tool Enhancements

- Constraint optimization did help a lot to improve the overall constraint read runtime, but we needed additional tool optimizations to achiever our targets

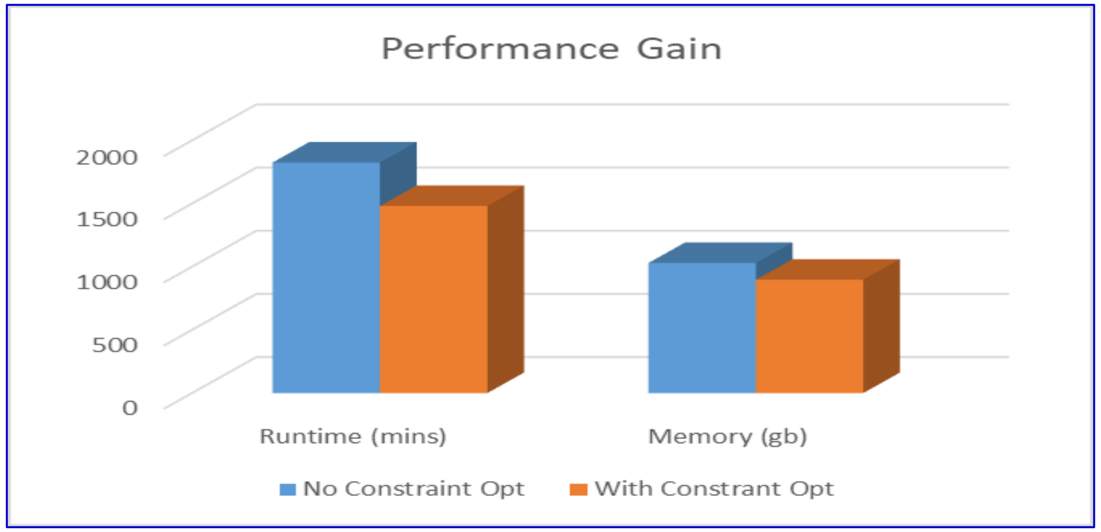
- Starting **T-2022.03-SP5*** version (Jan 2023 release) of PrimeTime, SNPS R&D optimized PrimeTime internal exception processing capabilities

- Tool enhancements not only helped executing full flat runs but also in memory reduction from Terabytes to Gigabytes.

Results and Conclusion

Runtime and Memory Impact with Constraints Optimization

- Runtime and Memory impact on Design A after implementing all the constraint optimization techniques discussed
- There is no change in PT version used
- Results are on the HS based model of Design A

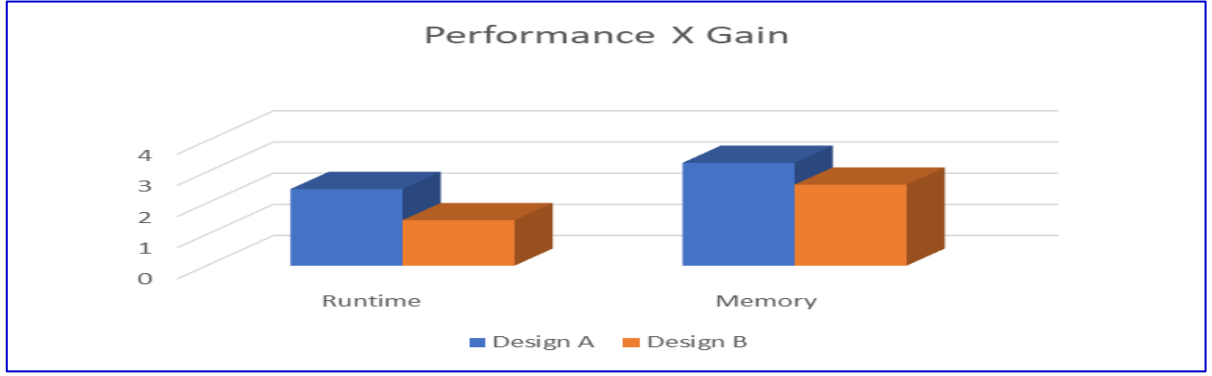


Runtime	Design A (HS model R* version)		
	No constraints opto	With constraints opto	Runtime Reduction
read_constraints	~17hrs	~14.2hrs	16%
update_timing	~12hrs	~8.5hrs	29%
total_runtime	~30.5hrs	~24.5hrs	19.67%

Memory	Design A (HS model R* version)		
	No constraints opto	With constraints opto	Runtime Reduction
read_constraints	859G	846G	1.5%
update_timing	1.03T	886G	13.98%
total_memory	1.03T	897G	12.91%

Runtime and Memory Gain with Tool Enhancements for HS Based Designs

- Runtime and Memory impact on Designs A & B with the T* versions
- Results are on the HS based model of Designs A & B



Design A	R* Version	T* Version	Runtime Gain
Read_constraints	~16h	~6h	62%
Update_timing	~12h	~5h	58%
Total Runtime	~30h	~12h	60%

Design A	R* Version	T* Version	Memory Gain
Read_constraints	869G	131G	85%
Update_timing	1.03T	312G	70%
Total memory	1.03T	312G	70%

Design B	R* Version	T* Version	Runtime Gain
Read_constraints	~12h	~9h	25%
Update_timing	~10h	~5.5h	45%
Total Runtime	~24h	~16h	33%

Design B	R* Version	T* Version	Memory Gain
Read_constraints	362G	154G	57%
Update_timing	1.14T	432G	62%
Total memory	1.14T	433G	62%

Runtime and Memory Gain with Tool Enhancements for Full Flat Designs

- Runtime and Memory impact on Designs A & B with the T* versions
- Results are on the full flat models of Designs A & B
- Design A full flat runs always crashed in R* versions. So there is no reference data

Design A	R* Version	T* Version	Performance
Read_constraints	Tool Crash	~6h	NA
Update_timing	Tool Crash	~5h	NA
Total Runtime	Tool Crash	~12h	NA

Design A	R* Version	T* Version	Memory Gain
Read_constraints	3TB+	237G	NA
Update_timing	3TB+	611G	NA
Total memory	3TB+	611G	NA

Design B	R* Version	T* Version	Runtime Gain
Read_constraints	~7h	~10h	-42%
Update_timing	~7h	~7h	0&
Total Runtime	~16h	~19h	-18%

Design B	R* Version	T* Version	Memory Gain
Read_constraints	505G	260G	48%
Update_timing	1.77T	602G	66%
Total memory	1.77T	602G	66%

Conclusion & Future Work

- Primetime is Capable of handling large designs with optimized runtime/peak memory without compromising on overall quality
- It is recommended to flatten the design as much as possible for better silicon correlation
- During initial phase of designs, better to focus the timing convergence with HS model for quick turn-around time
- Need to continue partnering with SNPS for further performance gains in the next versions

Future PT enhancements

- Primetime U* release is enhanced further to reduce the number logic updates which help in runtime improvement.
- Work in progress to check the QoR impact on multiple designs.

Design A (T*version)	Runtime	Memory	Logical Updates
Read_constraints	~13h	225G	24
Update_timing	~10h	567G	25
Total	~27h	612G	25

Design A (U*version)	Runtime (hours)	Memory (GB)	Logical Updates
Read_constraints	~8.5h	215G	11
Update_timing	~9.5h	559G	12
Total	~26h	592G	12

snug

THANK YOU

Our Technology, **Your Innovation**™