Hierarchical timing analysis & convergence of SoC using HyperScale methedology

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Chetana Bhadada
(14MECV04)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2016

Hierarchical timing analysis & convergence of SoC using HyperScale methodology

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Chetana Bhadada
(14MECV04)

Under the guidance of

External Project Guide:

Mr. Rahul Vishal
FCT Lead,
Intel India Technology Pvt. Ltd.,
Bangalore.

Internal Project Guide:

Dr. Usha S Mehta
Professor EC Department,
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2016

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

- Chetana Bhadada
14MECV09

Disclaimer

"The content of this paper does not represent the technology, opinions, beliefs, or positions of Intel Technologies India Pvt. Ltd. Company, its employees, vendors, customers, or associates."

Certificate

This is to certify that the Major Project entitled **"Hierarchical timing analysis convergence of SoC using HyperSclae methdelogy "** submitted by **Chetana Bhadada (14MECV04)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Internal Guide

Dr. Usha S. Mehta
(Professor, EC)

Program Co-ordinator

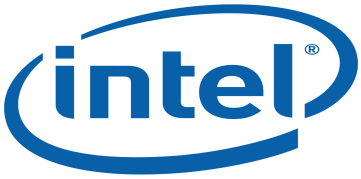
Dr. N. M. Devashrayee
(Professor,EC)

Director

Dr.P. N. Tekwani
(Director,IT-NU)

Head Of Department

Dr. P. N. Tekwani
(Head of EE Dept.)



Intel Technology India Pvt. Ltd.

Certificate

This is to certify that the Major Project entitled **"Hierarchical timing analysis convergence of SoC using HyperScale methodology "** submitted by **Chetana Bhadada (14MECV04)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Bangalore

Mr. Rahul Vishal
FCT Lead
SDG India
Intel Technology India
Bangalore.

Mr. Rahul Sonawane
Engineering Manager
SDG India
Intel Technology India
Bangalore.

Acknowledgements

First and foremost, sincere gratitude to my managers Mr. Rahul Sonawane and Mr. Shankar R Sawant. Also I am thankful to Intel Technology India Private Limited, for assigning me such project and guide me through.

I would like to express my gratitude and sincere thanks to my mentor Mr. Rahul Vishal at Intel Technology India Private Limited, Bangalore for his valuable guidance throughout this period. He has given me valuable advices and support for my project work which I am very lucky to benefit from.

I would like to express my gratitude and sincere thanks to our Director Dr. P. N. Tekwani, Head of Electrical Engineering Department for allowing me to undertake this thesis work and for his guidelines during the review process.

I would like to thank my Program Coordinator, Dr. N. M. Devashrayee, Professor, EC (VLSI Design), Institute of Technology, Nirma University, Ahmedabad for giving valuable support and motivation throughout the academic period.

I would also thank to my Project Guide, Dr. Usha S Mehta, Professor,EC Department, Institute of Technology, Nirma University, Ahmedabad for being a source of inspiration, giving valuable support and timely guidance for the project work.

I take immense pleasure to thank my team member Mr. Nischay Patel and other colleagues in the Intel, special thanks for helping me on this path and for making project at Intel more enjoyable.

I wish to thank my classmates for their delightful company which kept me in good humor throughout the journey.

- Chetana Bhadada
14MECV04

Abstract

The semiconductor industry accepts two facts: designs continue to grow in size and complexity, and time-to-market pressure is higher than ever. Static Timing Analysis (STA) is a key task during chip design that directly impacts design cycle time. With design sizes growing 2-3x every two years, new technologies like multi-core processors have helped alleviate runtime pressures. As designs continue to grow in size, hierarchical techniques are used to break down design complexity into manageable units of work. In STA, hierarchical timing models have been used to represent block timing in a compact manner to enable faster and more memory-efficient STA runs. However, these models are unable to capture the context of where the blocks reside in the full chip, thus causing differences between full-chip flat runs and hierarchical runs.

An example where a hierarchical timing model for a block does not capture the context accurately in traditional STA analysis is clock re-convergence pessimism removal (CRPR) effect. When two related clocks enter a block, it is difficult to model the CRPR properties of their top-level common source in the block-level analysis. When a full chip flat analysis is done, the entire clock network is visible, hence the CRPR effects can be accurately applied. With HyperScale technology, boundary clock CRPR relationships are captured as part of the updated context and used for accurate block-level analysis. This eliminates the need for a flat run.

HyperScale provides a 5 to 10x boost in performance and capacity, leap frogging the design gap while maintaining the accuracy of flat analysis. In this project, comparison between traditional flat run and HyperScale run have been analyzed.By keeping slack within tolerant limit reduction in runtime and peak memory has been achieved.

Contents

List of Figures

Abbreviations

SOC System On Chip

STA Static Timing Analysis

DUA Design Under Analysis

ETM Extracted Timing Model

QTM Quick Timing Model

HDL Hardware Description Language

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

ILM Interface Logic Model

SDC Synopsys Design Constraints

ASCII American Standard Code for Information Interchange

ASIC Application-Specific Integrated Circuit

CMOS Complementary Metal-Oxide Semiconductor

RTL Register-Transfer Level

SDF Standard Delay Format

SPEF Standard Parasitic Exchange Format

RSPF Reduced Standard Parasitic Format

SBPF Synopsys Binary Parasitic Format

ECO Engineering Change Order

OCV On Chip Variation

AOCV Advanced On Chip Variation

CRPR Clock Reconvergence Pessimism Removal

RAM Random Access Memory

UPF Unified Power Format

Chapter 1

Timing Models for hierarchical designs - Overview

PrimeTime have capability of hierarchical analysis of a SoC by building timing models and then using these models to show the timing behavior of complex blocks in a design. A timing model of any block gives the information about timing characteristics all the input and output without using complete netlist.

Before performing static timing analysis on a design, all the cells must have a timing model of it. For the purpose of static timing analysis, each leaf cell is treated as a simple macro cell. (eg. flip-flop, NAND and NOR) or a big complicated block (such as a memory or a processor)

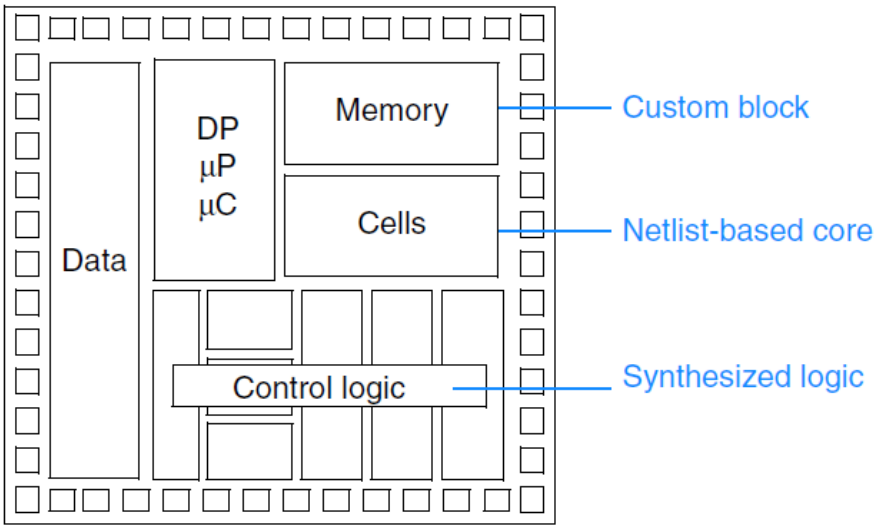


Figure 1.1: Components of a Typical Chip

Figure 1 shows a typical chip, which contains the following components:

- **Synthesized logic** - Modeled as a netlist containing gates such as NANDs, NORs, and flip-flops. The gates are modeled using the standard Synopsys modeling language.

Then the gates are compiled into a logic library database (.db) file using Library Compiler. This logic library is the same library used by Design Compiler and IC Compiler.

- **Netlist-based core** - Modeled as a netlist containing gates such as NANDs, NORs, and flip-flops. The gates are modeled using the standard Synopsys modeling language. Then the gates are compiled into a logic library database (.db) file using Library Compiler. This logic library is the same library used by Design Compiler and IC Compiler.

- **Custom block** – Defined at the transistor level and imported into the chip as a fixed unit, such as a RAM or microprocessor block. Because a gate-level netlist does not exist for this type of block, the Liberty modeling language can be used to describe the timing behavior of the block. Interface timing specification models can also be used to model custom blocks. However, the Liberty modeling language offers additional features, such as mode-dependent timing arcs, internally generated clocks, and internal pins.

1.1 Quick Timing Model (QTM)

At the very initial stage of any design, if any of its block is not completed or not have a proper netlist then we can you can use quick timing model to describe its initial timing information. Later in the cycle, we can replace the quick timing model with the netlist block to obtain more accurate timing.

While defining a quick timing model of any design or block we use specified commands to model I/O ports, the setup and hold constraints on the inputs, the clock-to-output path delays, and the input-to-output path delays. We can also specify the loads on input ports and the drive strength of output ports. Figure 2 shows a block represented as a quick timing model.

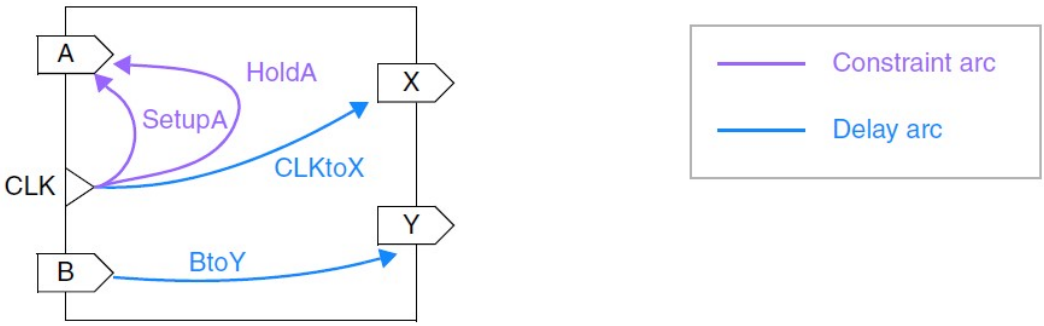


Figure 1.2: Quick Timing Model Representation of a Block

PrimeTime uses the information in the quick timing model to create a library cell with the appropriate constraint and delay arcs. We can save a quick timing model in the Synopsys .db or .lib format, then instantiate the quick timing model in a design just as you would instantiate library cells or interface timing specification models.

1.2 Interface Logic Models (ILM)

An interface logic model (ILM) is a partial netlist that contains only the interface logic of a block. An ILM contains:

- Combinational logic from each input port to the first stage of sequential elements of the block.
- Combinational logic from the last stage of sequential elements to each output port of the block.
- Clock paths to these sequential elements.
- Combinational paths from the input ports that do not encounter a sequential element and pass directly to an output port.

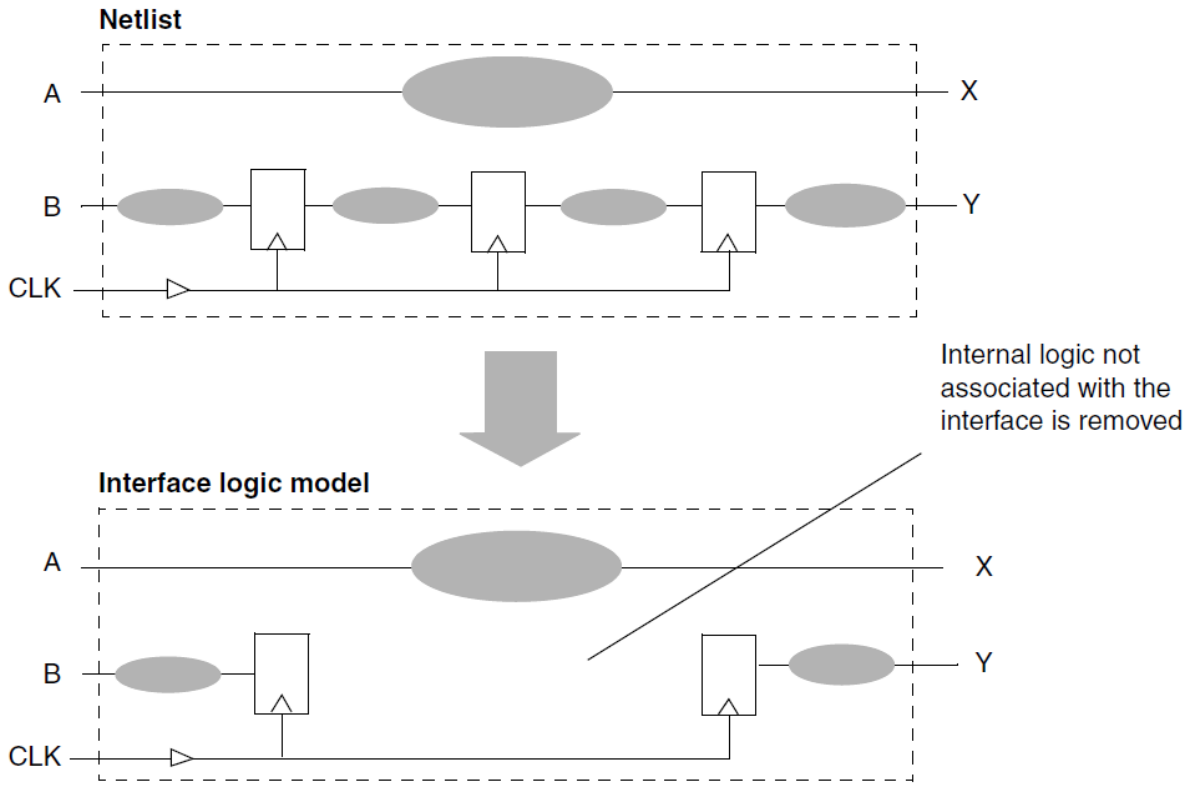


Figure 1.3: Generated Interface Logic Model

A generated ILM is context-independent, which means that the model is accurate for a range of operating environments. When the model is used in a design, the timing behavior of the model is different for different operating environments.

An ILM is a partial netlist that retains the combinational logic from each input or output port to the first or last stage of sequential elements of the block. The intent of an ILM is to produce a model that closely resembles the timing of the interface to that of the block-level netlist.

1.2.1 Benefits, Flow, and Limitations of ILMs

An ILM partial netlist can provide a smaller memory footprint, faster runtimes, and an extremely accurate timing of the interface logic of a block. You can use ILMs in Standard Delay Format (SDF) or Standard Parasitic Exchange Format (SPEF) based flows. In addition, you can use ILMs in PrimeTime SI using both crosstalk and noise analysis.

To use the model in a hierarchical static timing analysis methodology, you must validate an ILM. This validation requires that the timing and scope match the block-level netlist usage within the design. A certain amount of time is needed to validate an ILM.

1.3 Extracted timing models (ETM)

You can generate an extracted timing model (ETM) for a block with a corresponding technology-mapped gate-level netlist. Using extracted timing models provides the following benefits:

- Reduces the runtime and memory for full-chip analysis. we can run chip-level analysis with extracted models in place of the gate-level netlist for some modules, as shown in Figure 4.
- Protects the intellectual property of a netlist-based core.

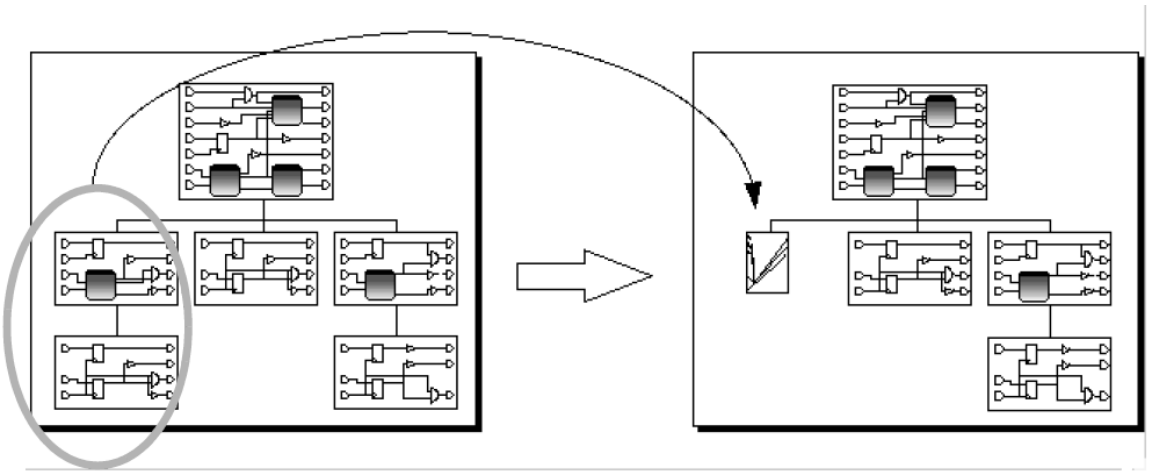


Figure 1.4: Extracted Model of a Bottom-Up Design

Extraction uses a technology-mapped netlist as input and generates a context-independent timing model in the Synopsys .db format or Liberty format. The generated model contains the same timing behavior as the original netlist. The delay values of arcs in the model are within the user-defined tolerance of the original path delays.

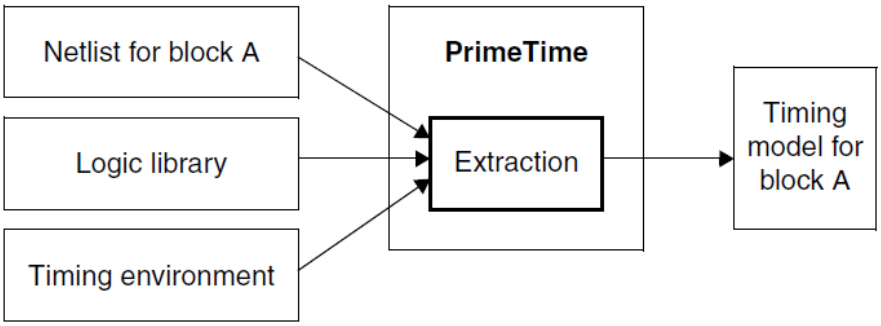


Figure 1.5: Timing Model Extraction Process

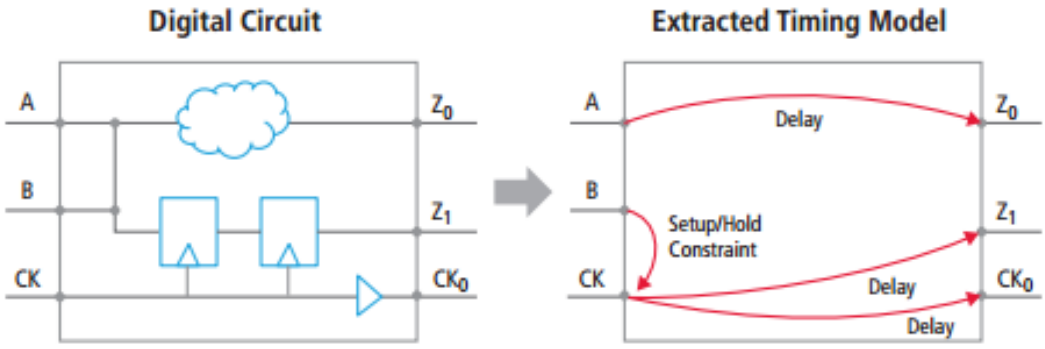


Figure 1.6: Timing Model Extraction Process

1.3.1 Limitations of Model Extraction

This section provides general guidelines and lists the general limitations of model extraction in PrimeTime:

- Complex latch structures on the design interface are not supported.
- Minimum and maximum delay constraints paths in the original design that have a minimum or a maximum delay constraint set on them are treated by extraction as follows:
  1. If you set the constraint on a timing path, PrimeTime ignores the constraint in the model.
  2. If you set the constraint on a pin of a combinational cell along a path, PrimeTime ignores the timing paths that pass through this pin. These paths do not exist in the extracted model.
- Data checks that are specified with the set_data_check command are not supported.

Chapter 2

HyperScale Flow - Introduction

The PrimeTime HyperScale methodology performs hierarchical static timing analysis (STA) with the below advantages:

- **Runtime and memory** – Full-chip analysis, HyperScale top-level analysis provides significantly faster runtime and consumes less peak memory compared to flat run. This can be achieved by smartly reusing block- and top-level analysis data and reducing the repeated computations.
- **Accuracy** – The HyperScale analysis flow gives the exact same margin computation and timing analysis as flat analysis, with the correctness of the flat PrimeTime flow for both coupled and uncoupled signal integrity analysis. HyperScale keeps full accuracy, including the effect of signal integrity on block boundaries.
- **productive top-level analysis** - The HyperScale flow keeps all records and reduces block timing data to enable more productive top-level analysis. This relieves designers from tasks of building, managing, and validating timing models. In the meantime, the timing context under which the block-level analysis is setup is also captured as the valid scope of the block timing.
- **Block scope checking** - When HyperScale fetches previously saved block data to run top-level timing analysis, the tool captures the actual top-level context and automatically checks that each block is instantiated within its scope for valid timing analysis.
- **Automatic context updates** – HyperScale automatically updates and takes the block timing context, so that no re budgeting is required to drive timing closure ECOs.
- **ECO guidance for placement and routing** - IC Compiler can use information from PrimeTime HyperScale technology to guide the timing ECO process, which assists in achieving block-level and top-level timing closure. Providing focused fixing information means that IC Compiler needs fewer iterations to close timing.

– **Management of constraints** - By the help of HyperScale flow, designers close timing at the block level with the latest contexts from the top-level timing on top of block-level constraints.

  During top-level analysis, HyperScale takes the same constraints which were used by flat analysis; this eliminates the need to generate top-level constraints. HyperScale checks and reports any inconsistencies in the constraints between the top and block. HyperScale also provides constraint extraction to automatically extract consistent block constraints from the flat top-level constraints.

2.1 Overview of the PrimeTime HyperScale flow

The HyperScale flow can be explained in the following way -

1. Block designers do the block analysis with raw constraints and save all the block session data. This stage occurs early in the design flow, before integration.
2. By the use of these block session data, the top-level designer can do top-level analysis with the top-level parasitic, netlist and golden constraints.When the top-level session data is saved, the tool updates the block boundary context data for subsequent block-level analyses. This block boundary context data includes:
   – Arrival times and clock latency for the block boundary ports.
   – Advanced timing information such as AOCV, CRPR, exceptions, and case values
3. Using the new context data from the previous top-level analysis, the block designers reanalyze the blocks. The updated block session data is available for subsequent top-level analysis.
4. Steps 2 and 3 are repeated till the design converges.

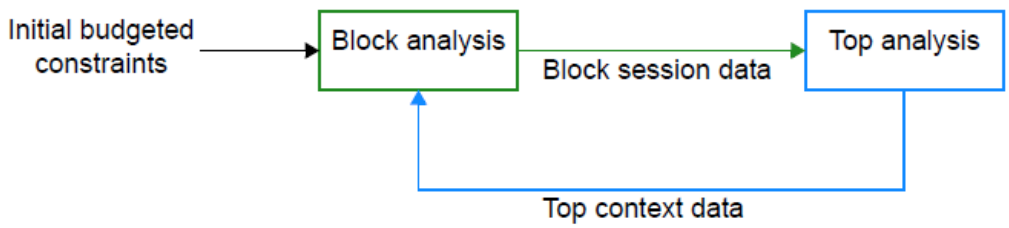


Figure 2.1: HyperScale analysis flow

The scalable HyperScale flow is suitable for existing hierarchical implementation methodologies. As block and top-level designs are updated, HyperScale session data helps

converge the design faster.

HyperScale technology takes benefit of the physical hierarchical implementation process by reusing the block-level timing for the top-level analysis. Figure 2.2 shows that after the internal register-to-register paths within a HyperScale block have been analyzed at the block level with accurate top-level context, they do not need to be reanalyzed at the top level.

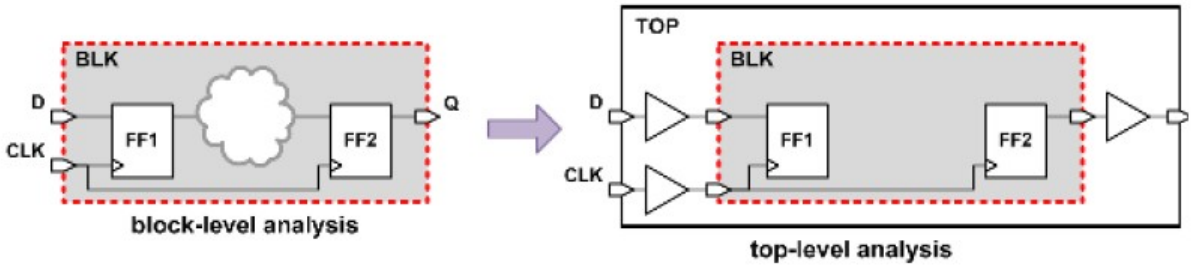


Figure 2.2: Block analysis provides block data to subsequent top analysis

If the budgeted block-level constraints bound the actual top-level context, a clean block-level analysis guarantees that the block is also clean at the top level. HyperScale omits the redundant and unnecessary work of reanalyzing the block's internal paths at the top level.

The logics of boundaries is kept for HyperScale blocks so that the block interactions with top-level design and with another blocks can be analyzed. Internal nets can be kept to fully model the boundary behavior, such as internal timing or noise aggressors to boundary nets.

In a standard flat analysis, a block is timed with budgeted constraints created by the designer. These raw constraints can be made manually, or budgeted and generated using a design planning tool (DPT). The designer has to take the responsibility of generating safe and bounding block-level constraints with as little conservatism as possible. Overly bounding constraints gives pessimistic block analysis. On the other hand, non-bounding constraints can cause missed timing violations.

HyperScale technology reduces the designer's burden by using information from the top-level analysis to generate contexts for block-level analysis. These contexts completely describe the block environment, including timing, loading, interconnect, signal integrity, and IEEE 1801 (UPF) characteristics. Figure 2.3 shows a block context derived from a top-level analysis and used in a block-level analysis.

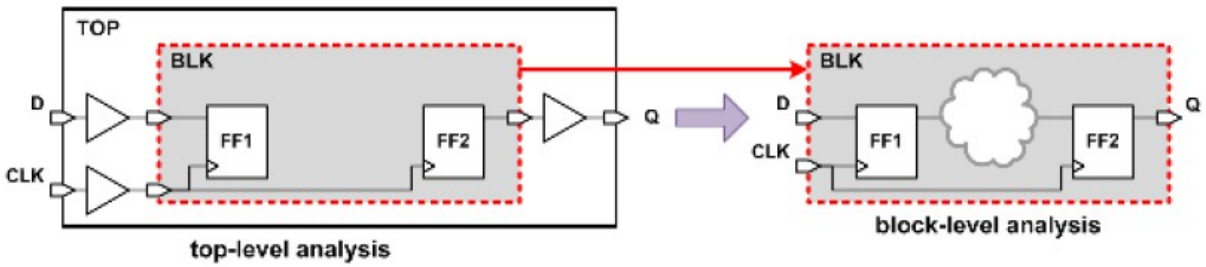


Figure 2.3: Top analysis provides context to subsequent block analysis

Unlike previous hierarchical analysis approaches, in HyperScale, block contexts are automatically generated. The disadvantages of older modeling methods are eliminated with respect to model validation and data organization. The HyperScale block-level analysis gives the same accurate block timing results by using the HyperScale contexts as seen with the flat timing analysis. HyperScale technology analyzes hierarchical designs with the same accuracy as standard full-flat analyses, without the runtime and capacity costs of full-flat flows and without the complexities of modeled flows.

In a HyperScale flow, block-level and top-level analysis runs are still performed. A Block-level run analyzes a block, and stores the resulting block analysis data on disk. The top-level run then reuses the block-level analysis data in its top-level analysis. The key to HyperScale technology is that the block-level and top-level runs now exchange pertinent design information with each other. Design and context information is shared using the following guidelines:

- After a block-level analysis run has completed, the updated information of all the block is saved and automatically becomes available to the top-level analysis.
- After a top-level analysis run has completed, the updated contexts for each block automatically become available to those block-level analyses.

  Since subsequent block-level runs automatically inherit their true top-level contexts, the designer is relieved of the burden of describing block budgets that are as minimally bounding as possible. Plus, as the top-level timing conditions around a block change (both due to changes in the top level as well as changes to neighboring blocks, these context changes are automatically provided to the block-level analysis.

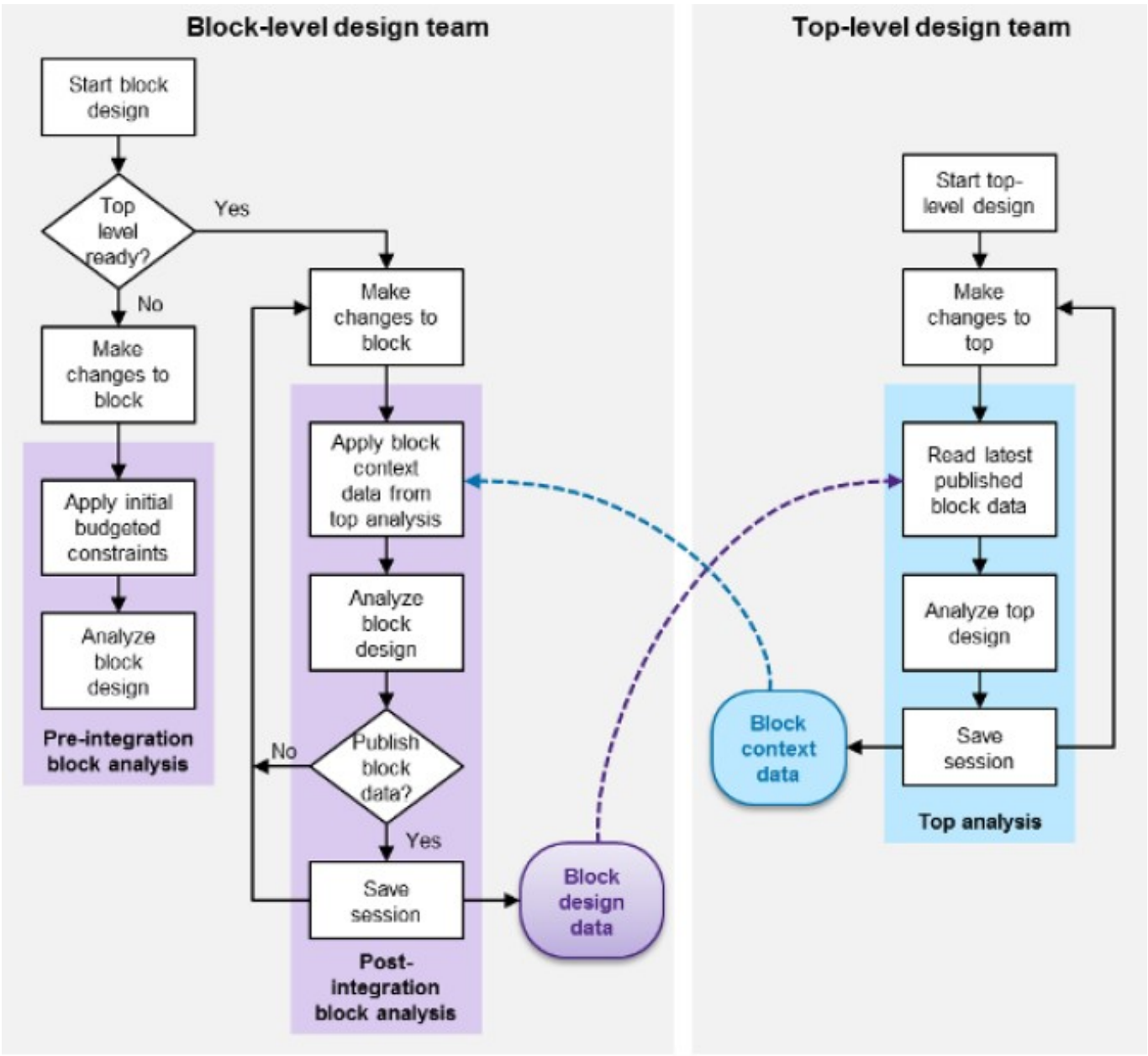


Figure 2.4: HyperScale analysis flow

In fact, the HyperScale flow has been designed to embrace the daily analysis runs that are common with large chips and tight schedules. As each block-level or top-level designer performs analysis, their analysis uses the latest available design data from other engineers. When they make their own design changes and complete their runs, their updated design data becomes available to all other engineers. The HyperScale analysis performs the data management needed to make this happen; the designer runs the analysis just as they normally would, adding a few extra HyperScale commands to their script. Figure 4 shows how the HyperScale analysis flow allows block-level and top-level design teams to work in parallel with each other, yet swap design and

context data during chip integration. For particularly large or complex chips, multi-

level HyperScale analysis can be performed where some intermediate blocks instantiate HyperScale blocks within them, but themselves are HyperScale blocks in upper-level designs. For the sake of simplicity, the remainder of this document only refers to block-level and top-level designs, with the understanding that such intermediate-level blocks fall into both of these categori.

2.2 Modifying the standard flow for HyperScale

HyperScale does not require significant changes in the static timing analysis flow and script. We can use our current PrimeTime script with the following additions:

- Enable HyperScale by setting the hyperscale_enable_analysis variable to true.
- Specify the HyperScale configuration by using the set_hyperscale_config command.

2.3 Perform block analysis

We can perform block-level HyperScale analysis during these design stages:

- Before integration, by using the initial budgeted constraints.
- After top-level analysis, by using the updated block context data.

To perform block-level HyperScale analysis, follow these steps:

1. Enable HyperScale by setting the hyperscale_enable_analysis variable to true.
2. Specify the HyperScale configuration by using the set_hyperscale_config command. If the tool does not find block context data from the top HyperScale analysis (which is expected during a pre-integration block analysis), the tool issues the following error message: Error: failed to read required data from '/disk1/top/TOP_HS_DATA/' for current design 'blkA', it must be created and populated before use.(HS-013).
3. Read and link the design data as we would in the standard flat flow.
4. Load the block constraints:
   - For the initial (pre-integration) block-level analysis, load the initial budgeted constraints.

- For context-adjusted block-level analysis, load the updated block boundary context.

5. Perform block-level timing analysis by using the update_timing command.
6. Report timing by using the report_timing command.
7. Save the block session data by using the save_session command.

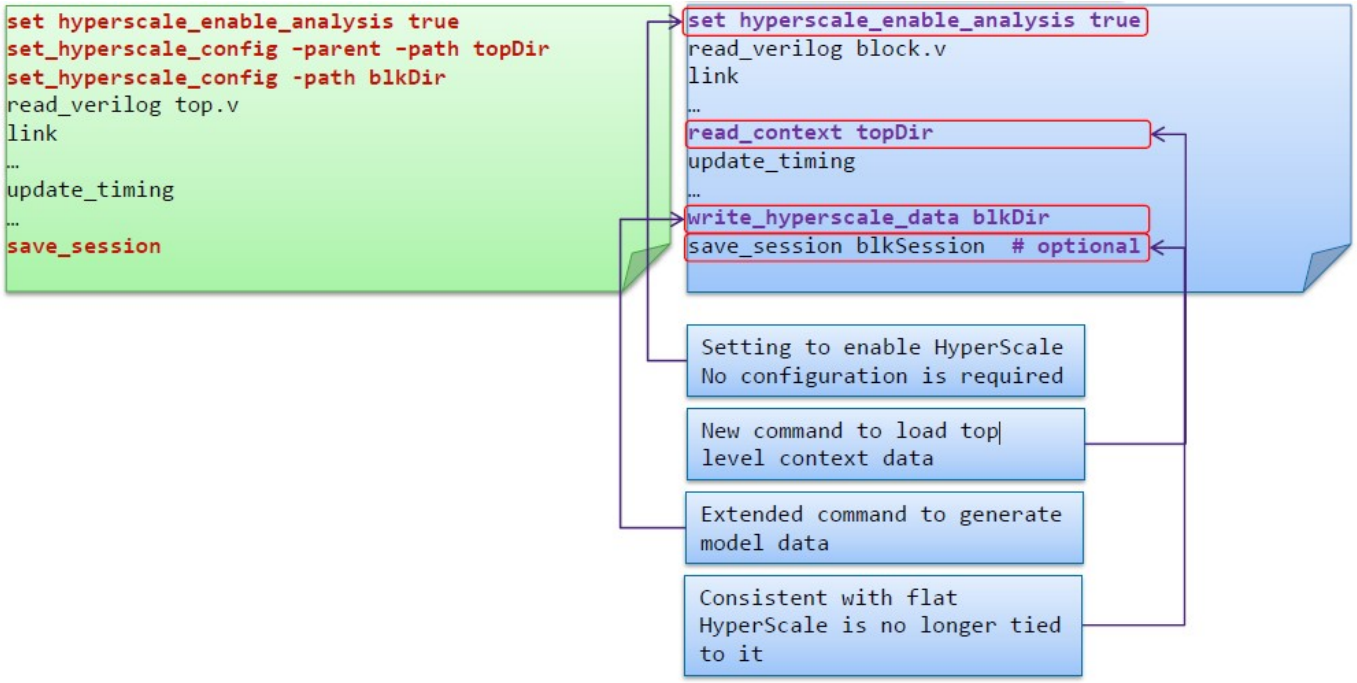


Figure 2.5: Block level Flow

2.4 Block analysis with adjusted context

After the top-level HyperScale analysis is done, true block context data is available to the block-level designers. The HyperScale block-level analysis proceeds as it did in the pre-integration block-level runs, except for one key difference. When the timing update begins, the HyperScale analysis checks to see if a block-level context from a previous top-level run is available in the top-level session data. If so, this updated block-level context for each port is overlaid on top of the designer's budgeted constraints. If block-level constraints were missing or unnecessary (such as the missing case analysis constant violation), they are applied or removed. If constraints were provided but with incorrect values (such as the nonbounding input delay violation), the values are corrected. This explains the mechanism through which fixable boundary violations are actually fixed. The following user-specified boundary constraints are automatically adjusted:

- Data input delays and output delays
- Clock latencies (static and dynamic)
- Input transitions
- Output loadings
- Case analysis values

This context adjustment process does not modify our budgeted block-level constraint script files. They remain unmodified on disk, and the context adjustment process happens in memory during the beginning stages of the update_timing command.

Some boundary violations are irresolvable; these violations are the most serious differences in constraints, such as missing or incorrect clock definitions. To be absolutely safe when it comes to the design intent, the HyperScale flow requires user action rather than making such significant changes to the block-level constraints. The context adjustment process is not limited to boundary violations. Whenever possible, the HyperScale flow uses the actual block context information in place of the budgeted constraints for all ports of the block. The goal is to provide a block-level analysis that provides the runtime and capacity advantages of a block-level analysis, yet accurately reproduces the block timing as it exists within the top-level design.

2.5 Top analysis

When we perform HyperScale top-level analysis, the analysis works a little bit differently than a standard analysis. As previously described, the netlists for any HyperScale blocks are automatically obtained from their respective HyperScale session data directories to satisfy the link requirements for those blocks. In addition, the detailed parasitics for the HyperScale blocks are automatically applied. Because the block-level design teams are actively running their own HyperScale block-level analyses, the top-level run automatically uses the latest block session data available from those teams. To perform top-level HyperScale analysis, follow these steps:

1. Enable HyperScale by setting the hyperscale_enable_analysis variable to true.
2. Specify the HyperScale configuration by using the set_hyperscale_config command.
3. Read and link the design data as we would in the standard flat flow.
4. (Optional) Allow design mismatches by setting the link_allow_design_mismatch variable to true.

5. Check the design configuration by using the report_hyperscale command.
6. Perform top-level timing analysis by using the update_timing command.
7. Report timing can be obtained by using the commands report_timing, report_global_timing, and report_analysis_coverage.
8. Check the clock mapping by using the report_clock -map command.
9. Verify that the block-level constraints bound the top-level context by using the report_constraint -boundary_check command. If there are violations that cannot be automatically fixed by the tool, we need to manually adjust the constraints.
10. Publish block contexts for subsequent block-level analyses by using the save_session command.

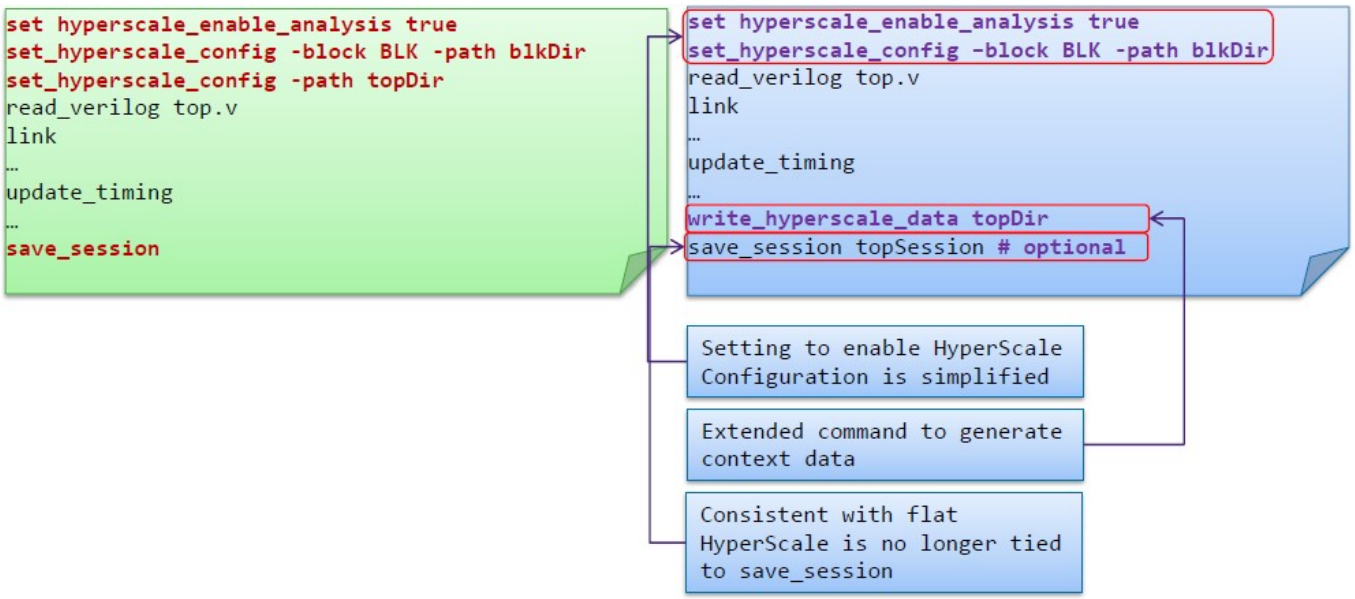


Figure 2.6: Block level Flow

2.5.1 Example of top-level HyperScale configuration

Figure 13 shows a design example with two levels of hierarchy. The top-level block is TOP, and the subblocks are blkA, blkB, and blkC. To configure HyperScale for the design in Figure 2.4, use these commands:

- Block design setup

1. set_hyperscale_config -block blkA -path blkA/HS_DATA -name fast_func

2. set_hyperscale_config -block blkB -path blkB/HS_DATA -name fast_func
3. set_hyperscale_config -block blkC -path blkC/HS_DATA -name fast_func

- Top design setup

set_hyperscale_config -path TOP_HS_DATA -name fast_func

2.5.2 Report the HyperScale configuration

After we configure the design with the set_hyperscale_config command, confirm the design configuration with the report_hyperscale command. We can use this command in block- or top-level analysis any time after link_design.

By default, the report_hyperscale command shows the file paths in which the block design is called and used during linking and the target path to where top-level session and context data is written. It also shows the date and time that the HyperScale data is created, the scenario name of HyperScale sub blocks, and the current design.

2.6 Read and link the design data

After specifying the HyperScale configuration commands, we read and link the design.When HyperScale subblocks are instantiated in the design being analyzed, we do not read the netlists for these blocks; the HyperScale analysis automatically satisfies the link requirements for these blocks using the HyperScale block session data configured with the set_hyperscale_config command.

After the design has been read and linked, apply any detailed parasitics with the read_parasitics command. Hierarchical extraction is required for a HyperScale flow, with separate detailed parasitics files generated for the top-level and for each of the physically-unique HyperScale blocks. If an analysis contains HyperScale subblocks, we do not read the detailed parasitics for those blocks; the detailed parasitics are already a part of the HyperScale block session data, and they are automatically loaded after the design is linked. Instead, read in the top-level parasitics file that contains the detailed parasitic information up to the HyperScale block boundaries. PrimeTime stitches the top-level parasitics and HyperScale block parasitics together.

2.7 Apply the constraints

Constraints are clock definitions, timing exceptions, clock gating settings, and any othercommands that are applied to design objects. Applying constraints in a HyperScale block-level

analysis is identical to standard block-level analysis; we apply the full set of block constraints as we normally would.

In the HyperScale top-level analysis, we apply the same constraints that are used in a normal top-level flat chip run. HyperScale handles the flat constraints and clock definitions covering both top and subblocks with necessary consistency checking. Additionally, any boundary-related exceptions are handled with the appropriate context update and applied to the blocks.

2.8 Extract the HyperScale block-level constraints

To start using HyperScale, we need consistent block-level constraints. We can extract block level clocks, case values, and exceptions with the HyperScale constraint extractor. The boundary delay values are estimated placeholder values, which are updated later with accurate HyperScale context.

For extraction of initial constraints from a top-level flat PrimeTime run, set the hyperscale_enable_block_constraint_extractor variable to true. This run -

- Reuses the same HyperScale configuration used in a top-level HyperScale analysis.
- Skips reading parasitics and SDF files if present in the script
- Extracts block-level constraints and PrimeTime variables for each block specified in the HyperScale configuration.

2.9 HyperScale technology and block boundary adjustment

Context adjustment might seem like automation of a laborious task that could be done manually—updating the block constraints with missing commands and correcting constraint values. However, HyperScale technology provides a level of accuracy in block boundary modeling that cannot be achieved through standard Tcl or Synopsys Design Constraints (SDC). One category of advanced boundary modeling is CRPR modeling.

In the top-level analysis, a CRPR credit of 0.2 is applied to the timing path from FF1 to FF2 due to the min/max delay variation in clock buffer U1. In the block-level analysis, while the clock and data arrival times can be described using the set_input_delay and set_clock_latency commands, there is no way to describe the CRPR relationship between FF2 and the launching flip-flop FF1, which exists outside the block.

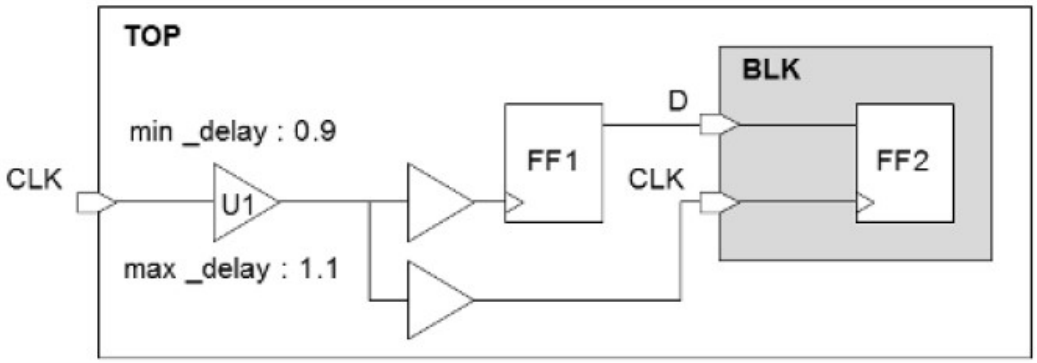


Figure 2.7: CRPR advanced boundary modeling

In the top-level analysis, a CRPR credit of 0.2 is applied to the timing path from FF1 to FF2 due to the min/max delay variation in clock buffer U1. In the block-level analysis, while the clock and data arrival times can be described using the set_input_delay and set_clock_latency commands, there is no way to describe the CRPR relationship between FF2 and the launching flip-flop FF1, which exists outside the block. However, the HyperScale context adjustment fully reproduces the effects of these external CRPR relationships at the block boundary. This CRPR context adjustment includes the static and dynamic components of CRP that can be introduced by on-chip variation, delta delay effects, and static and dynamic IR drop effects.

Chapter 3

Block boundary checks in HyperScale

HyperScale uses saved block session data for top-level analysis. It is important to have consistent scopes for the top- and block-level analyses. This ensures analysis accuracy for signoff. The scope includes timing constraints such as clocks, operating condition, timing and signal integrity analysis setup, and exceptions. Checking the analysis consistency and scope consistency is an important part of the flow in top-level analysis. This ensures the block-level data that is used in current top-level analysis is valid and top-level analysis is accurate, and that the updated context is relevant for subsequent block-level analysis.

For each HyperScale block, the budgeted block-level constraints must bound the actual top-level context. To ensure this bounding, perform a block boundary check by using the report_constraint command. This block boundary check reports any violations where a block-level constraint did not bound the actual top-level context behavior. In addition to numerical bounding checks, existence checks are also performed. For example, if a clock arrives at the input pin of a HyperScale block, but that clock was not defined (or was defined improperly) in the block-level constraints, the mismatch is reported. If we use the report_constraint command with no arguments, block boundary checks are performed by default, along with timing and design rule checks. To perform only the boundary checks without the other default timing and design rule checks, use the -boundary_check option. By default, only a single boundary check violation of each type is reported. To report all boundary check violations, use the -all_violators option. To perform more detailed reporting of each boundary violation, use the -verbose option.

If we use the report_constraint command with no arguments, block boundary checks are performed by default, along with timing and design rule checks. To perform only the boundary checks without the other default timing and design rule checks, use the -boundary_check option. By default, only a single boundary check violation of each type is reported. To report all boundary check violations, use the -all_violators option. To perform more detailed reporting of each boundary violation, use the -verbose option. There are two types of block

boundary violations:

- **Automatically fixable block boundary violations** - These violations indicate a boundary violation. However, HyperScale automatically adjusts the block-level boundary context during the next block-level run to resolve the violation. HyperScale automatically fixes these violations with a context adjustment.
- **Non-automatically fixable block boundary violations** - These violations indicate serious problems with the budgeted block-level constraints. HyperScale cannot automatically fix these violations; we must manually adjust the constraints.

3.1 Automatically fixable block boundary violations

HyperScale automatically fixes certain block boundary violations with a context adjustment. Examples of automatically fixable violations include arrival time, clock latency, case analysis, and timing exceptions.

The following example shows some fixable block boundary violations. In Figure 3.1, no case analysis constant was specified on the EN port in the block-level analysis. However, a constant does actually arrive at block pin BLK/EN in the top-level analysis. In addition, a late timing derating is specified at the top level but not at the block level.

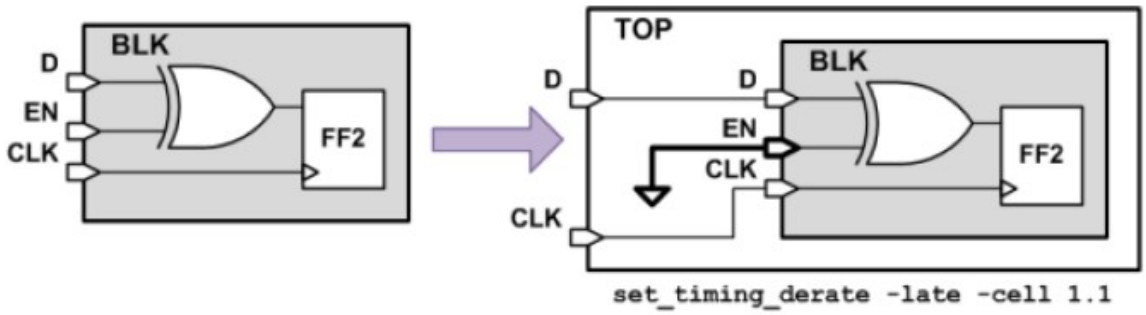


Figure 3.1: A fixable block boundary violation

3.2 Non-automatically fixable block boundary violations

We need to manually fix non-automatically fixable violations, which typically include clock-related violations, such as clock mapping or clock attribute violations.

An example of a non-automatically fixable violation is a mismatch in clock characteristics

at the block boundary. Figure 3.2 shows the clock period was incorrectly defined as 10 in the block-level analysis, and the actual clock period arriving from the top level is 12.

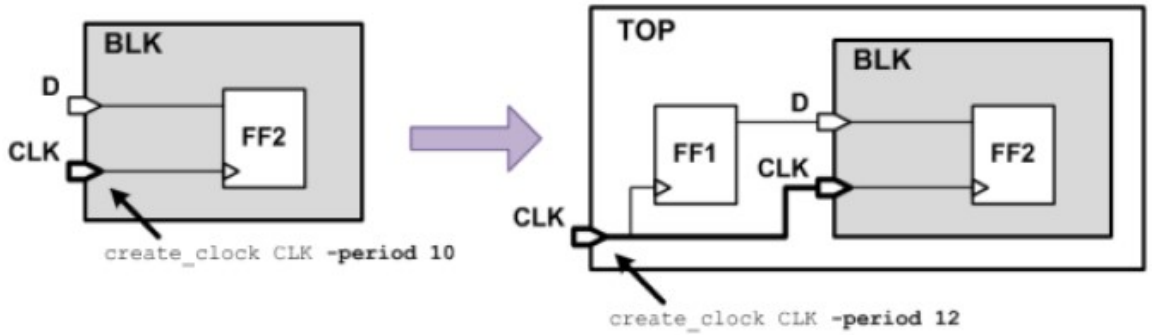


Figure 3.2: A non-automatically fixable block boundary violation

3.3 Clock mapping check

In hierarchical analysis, we need consistent clock definitions between block- and top-level analysis. When block session data is linked in top-level analysis, HyperScale establish the clock relationships between the clocks that are defined and used in block-level analysis and the clocks that are defined in top-level analysis. It is fully automatic process that occurs during timing update.

Complete clock mapping is required for HyperScale to ensure good accuracy in timing analysis. Tool maps the clocks for each HyperScale instances in current design. Clock maps between various types of clocks. The tool maps the clocks between physical, virtual, and generated clocks. It also maps between inverted and noninverted clocks across the block boundary.

Chapter 4

HyperScale constraint extraction

We can extract consistent block-level clocks, case values, and exceptions from full-chip flat constraints by using the HyperScale constraint extractor. The tool reports any issues in the full-chip flat constraints, so we can resolve the issues before using the extracted constraints in block-level HyperScale runs.

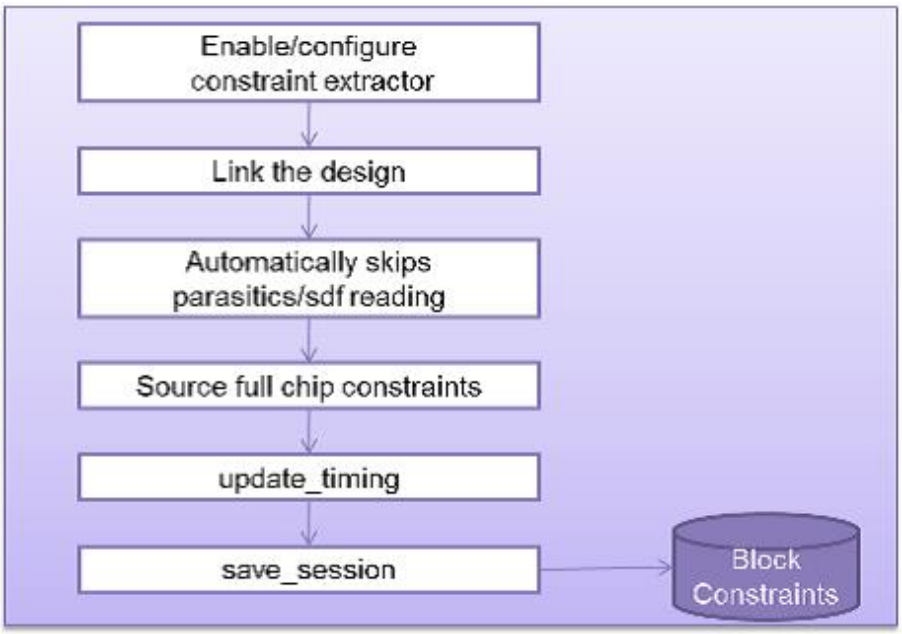


Figure 4.1: Flow for extracting and using constraints

4.1 Constraint extractor input and output files

The constraint extractor reads a flat design with full-chip constraints. The design can contain abstracted instances, such as an extracted timing model (ETM), quick timing model (QTM),HyperScale, or black box.The constraint extractor generates these output files:

- Block-level constraints as Synopsys Design Constraints (SDC) or write_script format:

  - Clocks
  - Exceptions, excluding crossing-boundary exceptions
  - Case, disable timing
  - Deratings
  - I/O delays, represented as placeholder values that are overwritten by accurate top context during HyperScale block runs

- Nondefault settings of environment and PrimeTime application variables

4.2 Assumed block-level constraints from the constraint extractor

The constraint extractor assumes the following settings:

- set_app]_var timing_input_port_default_clock false The PrimeTime default is false; you cannot set this variable to true in the HyperScale flow
- The group_path command is not written in block-level constraints. If needed, manually add the command.
- set auto_wire_load_selection false; set_wire_load_mode top These are automatically set by HyperScale top; set these variables in the top-level constraint file.

4.3 Unsupported constraints

The constraint extractor does not support these constraints:

- Constraints that are not supported by the write_script command, such as: set_clock_sense -stop_propagation
- Advanced on-chip variation (AOCV) commands, such as:
  - set_timing_derate -aocvm_guardband
  - read_aocvm
  - remove_aocvm
  - reset_aocvm_table_group
  - set_aocvm_coefficient
  - set_aocvm_table_group
- Signal integrity analysis

- Noise analysis
- UPF

If such commands are encountered, the constraint extractor issues the HCEXT-011 warning, and you need to manually add those constraints to the block-level constraint files.

4.4 Errors and warnings that cause block-level constraint extraction to fail

The following errors and warnings report design issues that prevent the extraction of correct block-level constraints; you need to resolve these issues in the flat chip-level constraints:

- HCEXT-004 error: Generated clock definition inside HyperScale block has a problem.
- UITE-461 error: Generated clock is not satisfactory.
- PTE-004 error: The generated clock pin is in a loop or is in the fan out of two clock sources.
- PTE-024 error: A loop of generated clocks is formed in that there is a circular dependency of generated clock to master clock.
- PTE-025 error: The master of the generated clock is not connected to any clock source.
- PTE-023 warning: The generated clock has not been expanded.
- PTE-075 error: Generated clock has no path to its master clock.

4.5 Generated clock extraction issues

All generated clock extraction issues are caused by sequential source networks that span block boundaries. To resolve these issues, We need unambiguous generated clock definitions. We need to modify the constraints for the following conditions:

- **Condition 1** - Generated clocks across HyperScale block boundaries need to be defined appropriately in flat constraints to extract block-level constraints correctly.

  The cross-boundary generated clock needs a combinational-only path from the master clock sources to the block boundary. When this condition happens, the tool issues an HCEXT-001 error message and does not write constraints.

- **Condition 2** - Source network of generated clock leaves and reenters two or more times. When this condition happens, the tool issues an HCEXT-002 error message and does not write constraints.

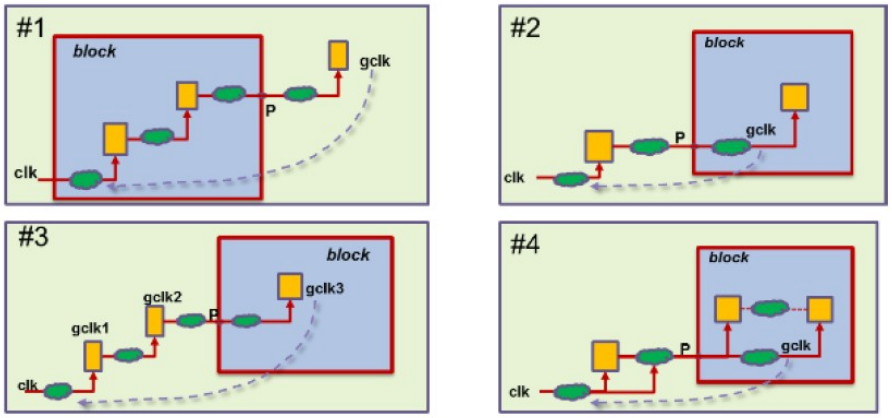


Figure 4.2: Conditions that cause generated clock errors

Chapter 5

Work done

In this project, we have started with a hierarchical design.For this design we we have performed flat as well as HyperScale run and measured peak memory and runtime.

5.1 4-level hierarchical design

The design on which I am working has the following hierarchy diagram. This was the very first design on which we started.

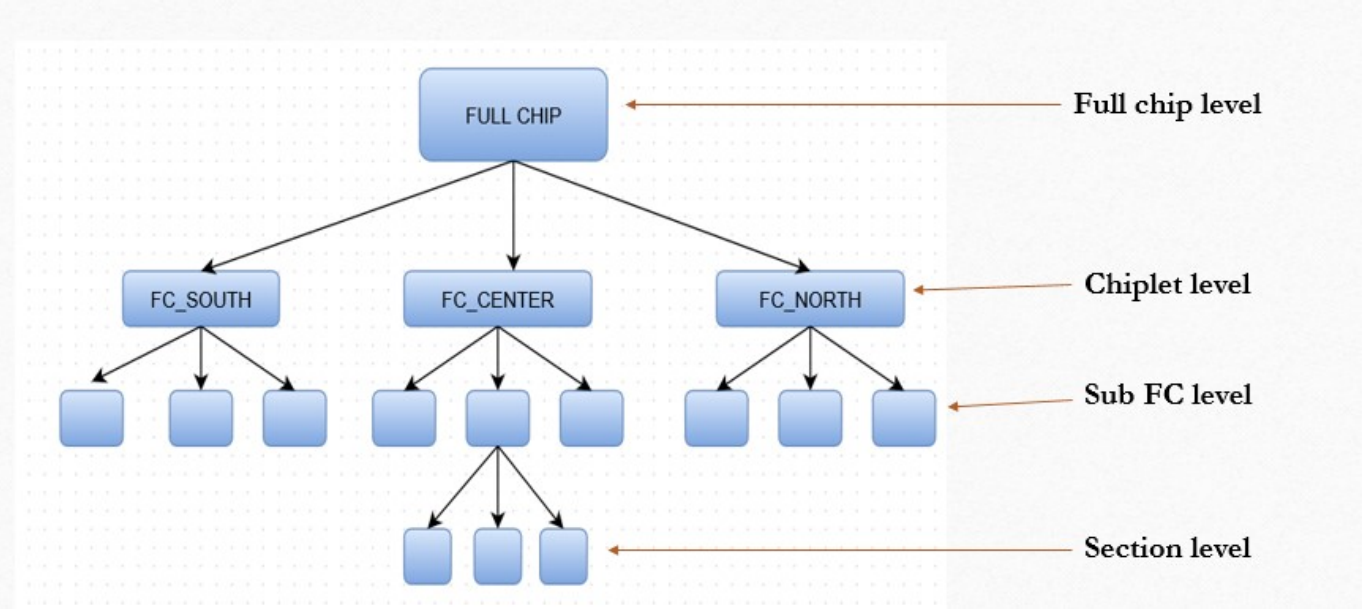


Figure 5.1: Hierarchical design-1

The design which was given has total 4 levels as follow-

1. **Top level** – This is the super parent of the design. At this level all 3 chiplet level designs are called.

2. **Chiplet level** – At this level sub FC's are instantiated. Many of the sub FC's are instantiated multiple times. Those blocks are known as multiple instantiated modules (MIM).

3. **Sub FC level** – All the sections are called at this level.

4. **Section level** – This is very level of design.

Due to complex design it was difficult to manage all the levels so we removed one level of hierarchy (Chiplet level) from the design and in next model all the sub FC will be called at Full chip level.

5.2 3-level hierarchical design

As we have removed chiplet level from the design and now all the sub fc's will be instantiated at top level directly. According to this new model we have modified the script of all the blocks according to child parent information of the new model.and verilog netlist has also been changed.

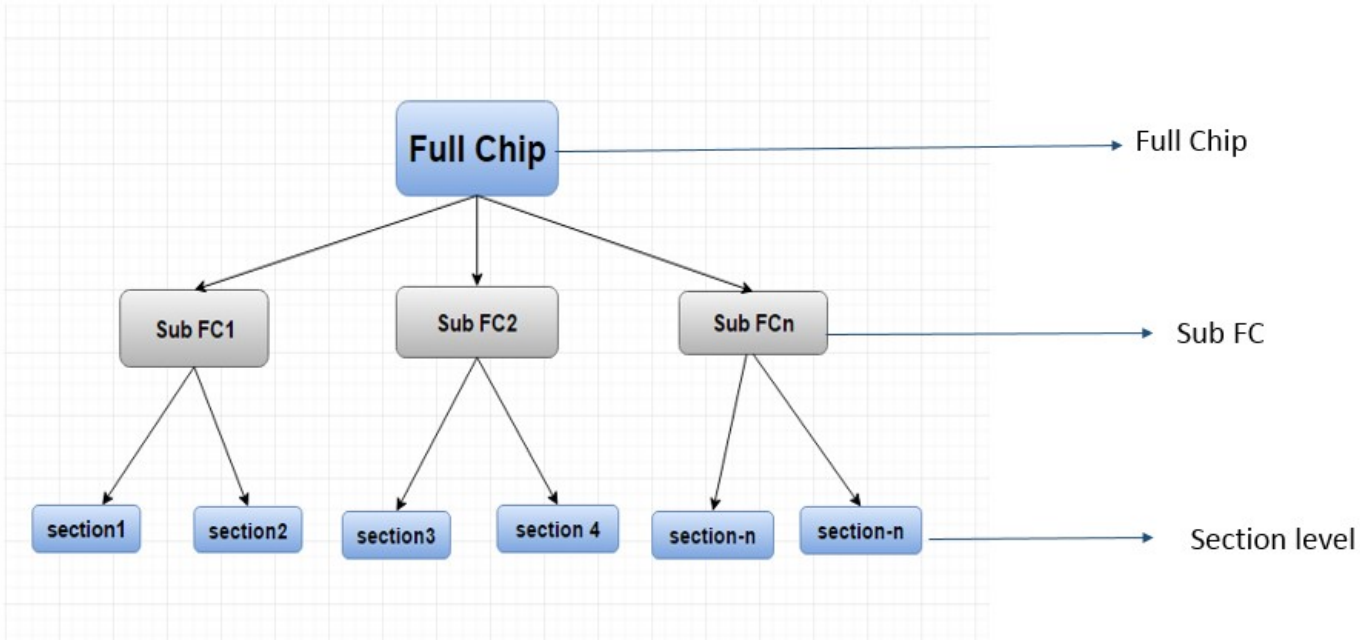


Figure 5.2: Hierarchical design-2

For this model also we faced issue with some of the Sub FC blocks and it was crashing continuously during update timing stage. So we made those particular blocks to run for single core during update timing stage. But that was also not working.

5.3 2-level hierarchical design

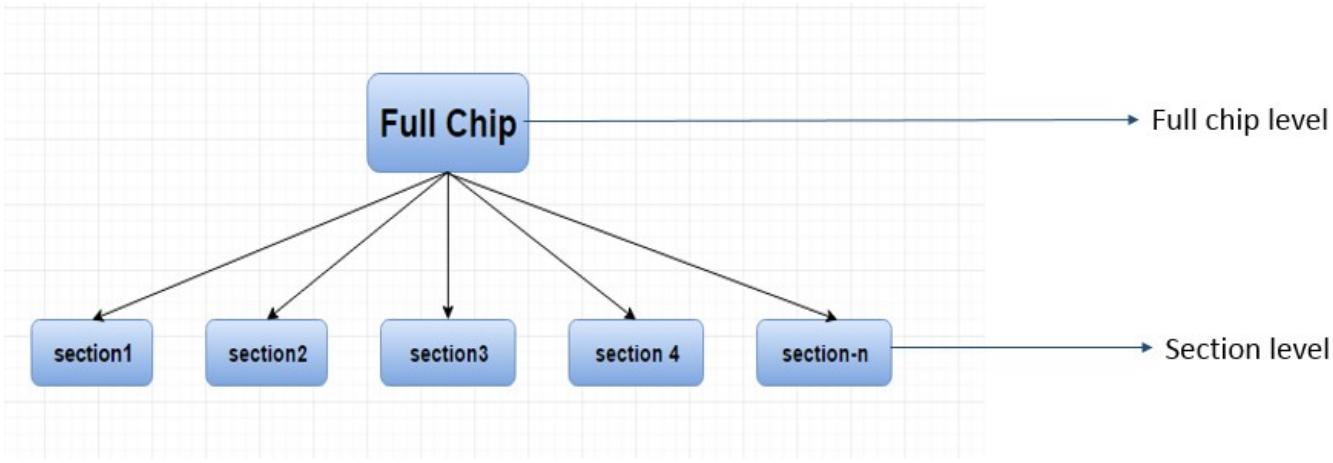


Figure 5.3: Hierarchical design-3

We tried another new model. In new model all the sections are directly instantiated top level. We tried to remove chiplet level and sub FC level from main design, but we could not modified the Verilog netlist because of lots of changes.

So we dropped the idea of using this model 3 and started with new model. A new model with one of the chiplet as flat was also tested.

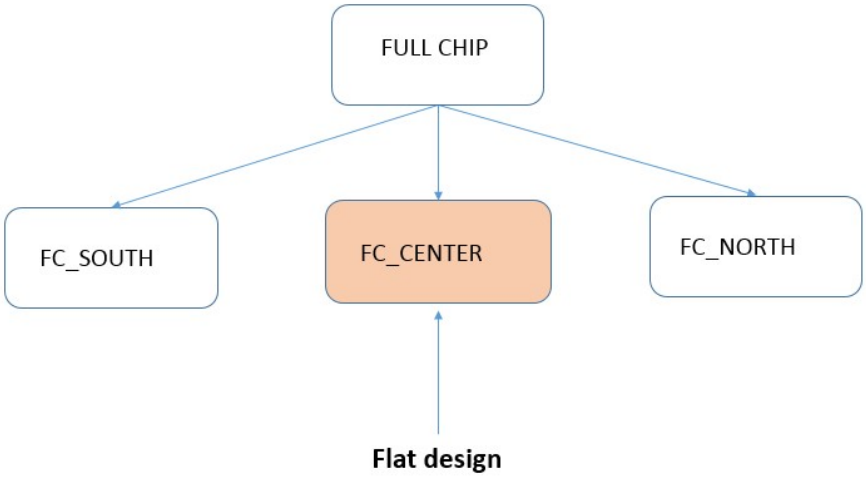


Figure 5.4: Hierarchical design-4

After this a new model was build. In that model we have dropped those modules which were multiply instantiated and modified verilog netlist and scripts of all the bolcks according to that. This was the first success model.

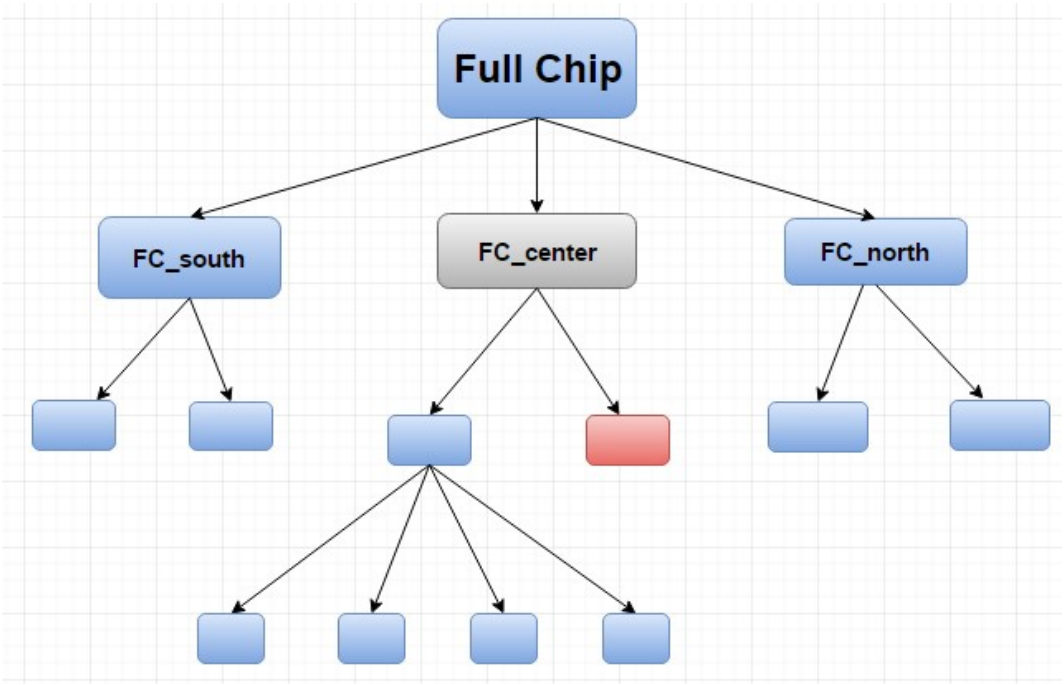


Figure 5.5: Hierarchical design-5

Chapter 6

Results and conclusion

In practical terms, a hierarchical approach reduces the number of design iterations between the block and full-chip level necessary to meet signoff. It reduces the number of times that a time-consuming flat netlist analysis is necessary. By making the process of meeting timing requirements more deterministic, the approach enables better forecasting of when a design reaches signoff. Measures of the number of violations per iteration also help design management understand if the signoff process is progressing in the right direction.

6.1 Memory comparison of flat v/s HyperScale

- As we were expecting, there was advantage in terms of memory and runtime.
- Peak Memory and Runtime both got decreased by approximately 42%

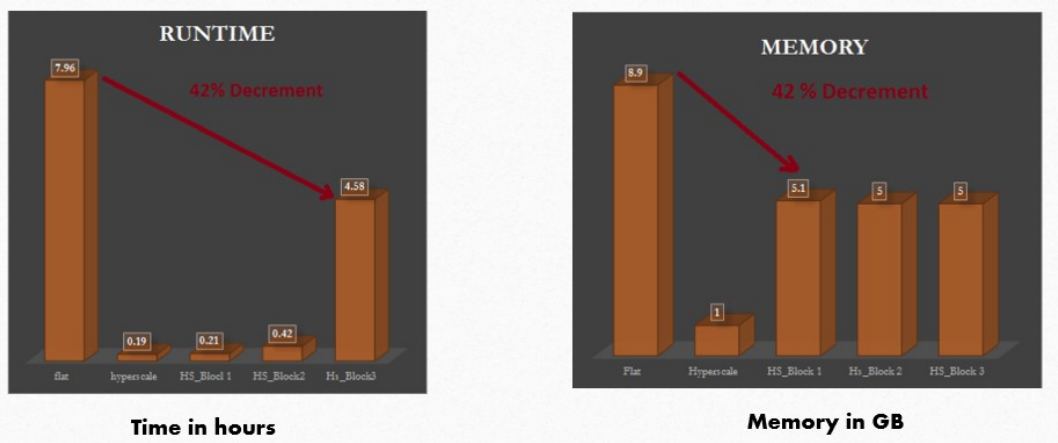


Figure 6.1: peak memory and runtime comparison

6.2 Margin comparison

The margin variation between flat and HyperScale was more than 5ps, which was not tolerable so we went for second iteration through constraints extraction method

6.3 Conclusion

- Compared to flat, full chip analysis HyperScale provide faster runtime with same accuracy.
- Constraints management is better in HyperScale.
- HyperScale uses less peak memory.
- This methodology is not compatible for high MI-ness.

References

[1] Sunil Walia.(2011).Reducing turnaround time with hierarchical timing analysis (online). Available FTP : `http://www.eetimes.com/document.asp?doc_id=1279120`

[2] Robert Hoogenstryd. (December 6, 2012). 20nm timing analysis – a practical and scalable approach (online). Available FTP :`http://www.techdesignforums.com/practice/technique/20nm-timing-analysis/`

[3] Bing Li, Ning Chen, Yang Xu, Ulf Schlichtmann "On Timing Model Extraction and Hierarchical Statistical Timing Analysis" *IEEE trans. on computer-aided design of integrated circuits and systems, vol. 32, NO. 3, march 2013*

[4] `http://www.vlsi-expert.com/p/static-timing-analysis.html`