A REPORT

ON

COMPREHENSIVE APPROACH TO ALIGN FLAT & HIERARCHICAL TIMING ANALYSIS IN SoC

BY

Name of the Student:	ID No:
MRITYUNJAY NATH	**2022HT80559**

AT

AMD India Pvt. Ltd., Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE,
PILANI (RAJASTHAN)

(November, 2024)

A REPORT

ON

COMPREHENSIVE APPROACH TO ALIGN FLAT & HIERARCHICAL TIMING ANALYSIS IN SoC

BY

MRITYUNJAY NATH ID No: 2022HT80559 Discipline: Microelectronics

Prepared in partial fulfilment of the

WILP Dissertation work course

AT

AMD India Pvt. Ltd., Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

(November, 2024)

WILP DIVISION

Organization: AMD India Pvt Ltd	**Location: Bangalore**
Duration: 5 Months	**Date of Start: April, 2024**

Date of Submission: November, 2024

Title of Project: Comprehensive Approach to Align Flat & Hierarchical Timing Analysis in SoC

Name of Student: Mrityunjay Nath

ID: 2022HT80559

Name of Supervisor: Lokesh Nema

Designation: Senior Manager (Silicon Design Engineering)

Name of Additional Examiner: Saikrishna Joginipally

Designation: Director (Silicon Design Engineering)

Name of Faculty Mentor: Sanjay Vidhyadharan

Project Areas: In this dissertation report the focus is on the timing analysis and STA closure for modern SoC and to bring upon an alignment to the two methods of Flat and Hierarchical at the SoC level timing closure.

Abstract: Modern SoC involves a great number of complexities, particularly in timing closure for high-speed clock and multiple clock domain. This crucial phase of STA is necessary for performance. The two primary methods are used: flat and hierarchical. In this report a comprehensive approach is taken to use both and align to close timing in a comprehensive manner.

[illegible] [illegible]

Signature of Student:	**Signature of Supervisor:**
Date: 2/11/2024	**Date: 2/11/2024**

Thank You, Team,

EVM

BITS Pilani WILP

Certificate

I hear by declare that the report work which is being presented in the dissertation titled "**Comprehensive Approach to Align Flat & Hierarchical Timing Analysis in SoC**" in partial fulfilment for the award of M.Tech (Microelectronics) at Birla Institute of Technology & Science, Pilani is completely an authentic record of my work carried out under the guidance & supervision of Mr. Saikrishna Joginipally, Director and Mr. Lokesh Nema, Senior Manager at AMD India Pvt. Ltd.

Acknowledgement

I take this opportunity to express my profound sense of gratitude and respect to all my friends & mentors who helped me through the duration of this thesis. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by numerous people. I acknowledge with gratitude and humility my indebtedness to **Mr. Saikrishna Joginipally,** Director, AMD India Pvt Ltd, Bangalore, under whose guidance I had the privilege to complete this thesis and my deep gratitude towards him for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to My Mentor**, Mr. Lokesh Nema,** Senior Manager, AMD India Pvt Ltd, Bangalore for his encouragement and cooperation during the completion of my thesis.

My greatest thanks to all who wished me success especially my parents and other family members and friends without whom I would not have been able to complete my thesis work.

I thank and own my deepest regards to all of them and all others who have helped me directly or indirectly.

Mrityunjay Nath

ABSTRACT

Growing design complexities of SoC which ranges to GHz and having multidomain clock crossing each other and having millions of gates count, brings time taking analysis with respect to Static Timing Analysis or STA in general. The previous methods of bottom-up timing analysis or fully flat analysis will not unmask the issues related to timing closure fully. Right from design cycle which involves Architecture and RTL coding, constraints which determines a designs STA also becomes important. Now there are many pros and cons of each method. While most accurate analysis are result of Flat approach, as it is completely transparent to the logic cell level but it requires each and every wire to be analysed and with the advent of complex design which has multiprotocol support in SoC, it requires huge runtimes that end up hours and days. To alleviate this there are approaches like hierarchical approach to close timing on blocks and then do analysis at a Sub System level and then move to SoC level with correct model to mimic timing and close the final STA for a SoC. Most companies do hierarchical and then the signoff happens with FLAT timing results. This takes huge time to close the STA and then go for tape-out. To mitigate this there are approaches that can be taken to close on design STA and Physical design can complete the PNR, clock tree synthesis etc and finally go ahead can be given for tape-out to bring the product quickly to market.

The methods described in this report is what is an alignment between the goods of both hierarchical and flat timing. The design here is complex SoC which has PCS support of Ethernet protocol with SerDes for data-path and processor for control path. The approach taken here is SDC Constraints promotion, Hyperscale analysis and IO budgeting strategies like context budgeting to justify the STA analysis for the design.

Contents

List of Figures

List of Tables

ABBREVIATIONS

CDR	Clock Data Recovery
CSR	Control & Status Register
FIFO	First In First Out
Gbps	Giga Bits per second
Ghz	Giga Hertz
MAC	Media Access Control Address
Mhz	Mega Hertz
NoC	Network on Chip
PCS	Physical Coding Sub Layer
PLL	Phase Locked Loop
PMA	Physical Medium Attachment
PNR	Place and Route
RISC	Reduced instruction set computer
RTL	Register Transfer Level
SerDes	Serializer-Deserializer
SoC	System on Chip
STA	Static Timing Analysis
VCO	Voltage controlled oscillator

CHAPTER 1. INTRODUCTION

1.1. MOTIVATION

A System on chip or otherwise known as SoC in modern semiconductor technology is a process of putting a complete system on a single die or chip. It contains large number of IPs with millions of transistors in it. System on chip or SoC, we shall further use the acronym in the complete report, provides modularity to the complex design for an application. Due to this complexity, there is an increase in reliability and reusability of the design. Cores like CPU, DSP, communication modules, mixed signal modules etc are present in a single SoC. These IPs, cores etc are verified and then integrated completely to create a bigger system that is specified to a particular application. These individual blocks then make up a sub-system which has a specific task to accomplish in a SoC. Finally, these Sub Systems are integrated and created a SoC. There are two parts to every design, a logical partition and a physical partition.

A logical partition is a wrapper around IPs to define a particular function for a sub-system. This is done by RTL designer during the design phase. Entire design is partitioned into small sub-blocks. Example, consider a processor subsystem, where the data path, control path, control and status register or csr makes up small blocks. These modules or sub-blocks are linked together in a main module that will have also parameter values for parameterized design. A final wrapper at RTL that wraps this is and has the interface definition for further connection during SoC integration is called as a logical partition or boundary. This is useful for verification to carry on the task and mitigate or resolve any issues seen in design during initial design phase.

A physical partition is when entire chip is divided into layout of smaller manageable blocks or sections. This is done for numerous reasons such as:

1. Managing the design such that different implementation team, like synthesis, PNR, STA, to work on different portions of the chip simultaneously.
2. Optimizing the performance of different sections for performance, power consumption, frequency and area.
3. This also ensure that each section has DFT blocks in it to put manufacturing constraints and reduce defects during fabrications or manufacturing.
4. Partitioning also allows more focused approach of closure for timing analysis. This is done by doing timing closures for each section independently before integrating into a whole system and then doing a final timing sign-off.
5. It also makes easier for scaling the design for evolving technology which allows for specific modifications in particular partition without affecting the timeline of entire chip.

Hereby, in this report physical partition can also be called as tiles.

Due to the complexity of modern design, a team of STA engineers work on each block using what is known as hierarchical design methodology. Each block needs constraints for clock, constraints for input and output delay, exceptions from the design. Another important consideration is the timing budgets. This budget helps STA engineer to close the particular block level timing irrespective of the one level up design STA analysis. In spite of this practical methodology, some possibilities remains where few critical timing paths are missed, more specifically the crossing paths between the hierarchies. The criticality of budgeting comes from the fact that if a reasonable delay for interconnects which are global then it ends up in many repeated iterations before the close of Full chip level timing.

Over budgeting can end up wasting the timing slack and underbudget can end up too many negative violations. If a correct model is not taken in then full chip can see many new clock paths which a block may not see at its level. All these scenarios are which we shall be taking into account for closing the current design considered here. Along with the correct model and the constraints promotion and dividing the timing analysis into multiple modes like functional, at-speed, scan etc also helps in closing the timing closure correctly and at the timely manner.

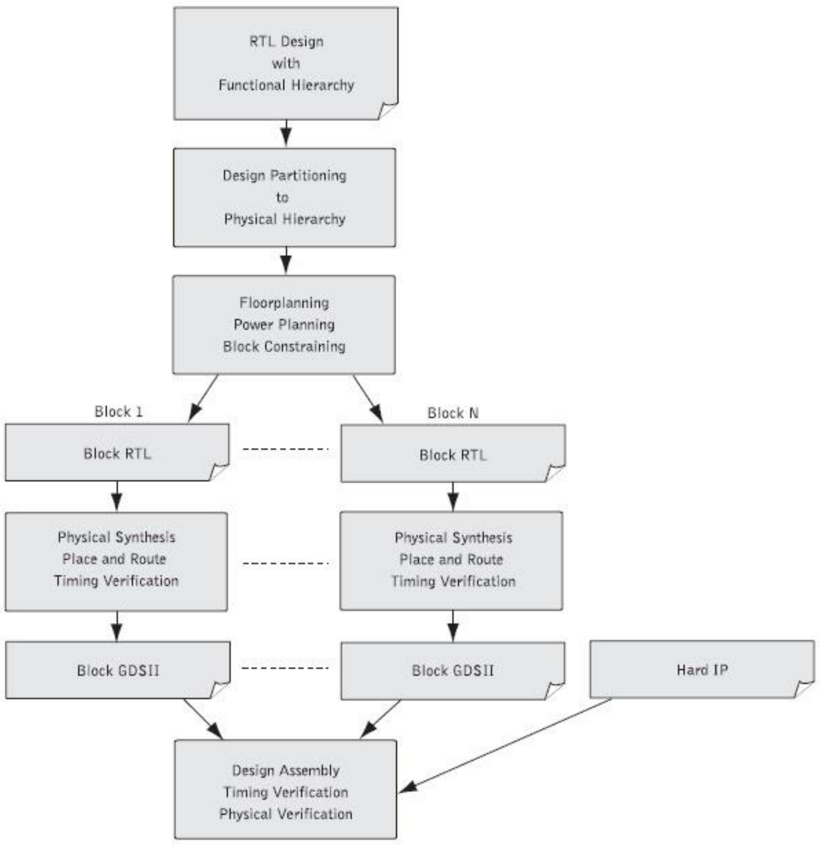


Figure 1 Generic Hierarchical Flow (Source EE times)

1.2. TIMING BUDGET & DIFFERENT TIMING MODLEs

In a large SoC there are multiple blocks inside each tile. STA engineer has to take care the timing between block to block I/O and then promote it to chip level I/O. Apart from this the STA engineer should also know when the data comes from neighbouring block when it is required to process in the current block. Each block at top level will be black boxed and only timing model will be used so the knowledge of I/O timing becomes very critical. For each block level we require the below timing information:

a) Clock source and its constraint which is used to clock the data.
b) Propagation delay at primary input specified using **set_input_delay** and output delay at primary output specified using **set_output_delay.**
c) Any timing exceptions as needed in the block.

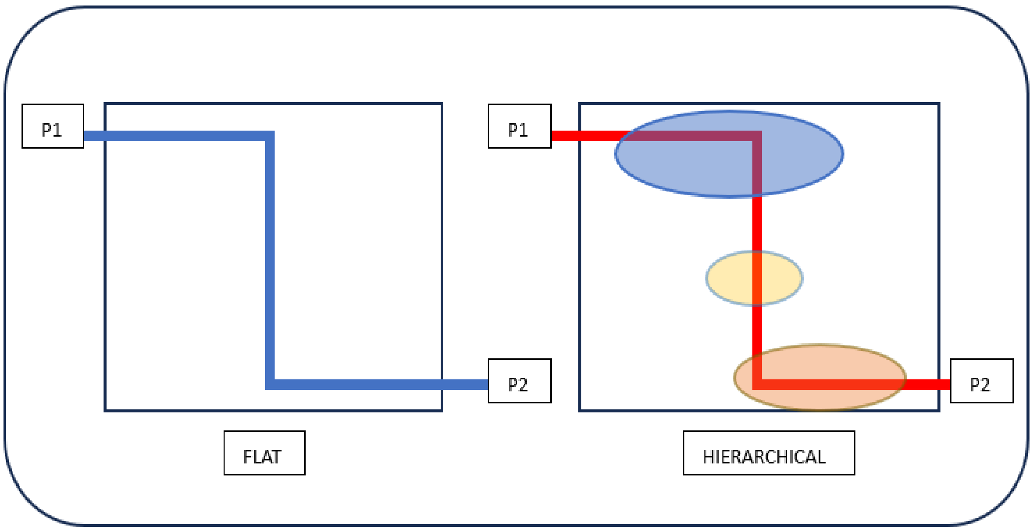


Figure 2 Flat Vs Hierarchical PNR and timing arc

Let us compare Flat design placement and corresponding hierarchical placement in ***Figure 2***. The Flat timing is very easy as there are no cells present between point **P1 to P2.** When compared to a hierarchical design we can see we have three blocks between point **P1** and **P2.** So, it's the job of STA engineer and design engineer to provide correct time by dividing the total traversal time from point **P1** to **P2** into three blocks such that it does not end up in over-budgeting or under-budgeting. It can be seen here that even though flat seems straight forward constraining but doing so in a complicated SoC can take huge amount of effort and can unmask many non-functional timings arc in the design. However, in flat the task is minimized by dividing the whole design into small blocks of tiles. It seems very trivial to use block level and then connect during integration for SoC but at SoC level the whole chip's timing budget comes into picture. Therefore, the industry practice is to do timing sign-off using flat STA.

Now despite all the measures there are some challenges in the timing budgeting: -

a) Before the design is physically placed, it is very difficult to predict any reasonable delay in global interconnects present in the chip. This interconnect delay prediction is very much necessary for timing budgeting.
b) Block level constrained should be properly mapped and promoted to chip level constraints.
c) Proper PNR is needed to budget the IO timings for SoC of chip level.
d) Delay budgeting can become more of an issue when more numbers of wires are part of global wires for which delays between pip-to-pin become stronger and depends upon the detailed routing in the implemented design.

Once the above budgeting is done for each block, they are integrated at SoC level. Once that is done the top-level timing is verified. For this the timing information of each block is needed in a format which is known as timing model. Once timing model is made available, they replace the actual block from the netlist such that one level up timing analysis can be done. These techniques minimize the STA runs faster. The industry follows one of the timing models such as Extracted Timing Model(ETM), Interface Logic Model, Quick timing model(QTM), Hyperscale analysis and context characterization.

1.3. THESIS ORGANIZATION

This dissertation report is organized into five chapters as described and briefed below:

Chapter 1: This introduces the main motivation behind the thesis where the outlines are present for different timing parameters that are needed to be kept in check for completing STA for a modern complex SoC

Chapter 2: This chapter presents the literature review of the Design and various strategies for timing analysis considered.

Chapter 3: This chapter presents the proposed work description.

Chapter 4: This chapter presents the result from the proposed methodology described in the report.

Chapter 5: This chapter describes the conclusion of the work.

CHAPTER 2. LITERATURE REVIEW

2.1. TIMING CHALLENGES IN SoC DESIGN

The evolving complexity in semiconductor world has enabled integrations of many functions on one die after advancement in each technology node. Now a days all processing unit and integrated interfacing unit of all functions and core are enveloped into a single chip. This led to the emergence of SoC which is driven by demands from customer market. From system and technology perspective, performance of system like clock frequency has improved a lot as off-chip interconnects between different functional modules are moved into a single die. Also, the overall consumption of power reduced by this as the discrete counterparts consume more power than what is integrated into single SoC.

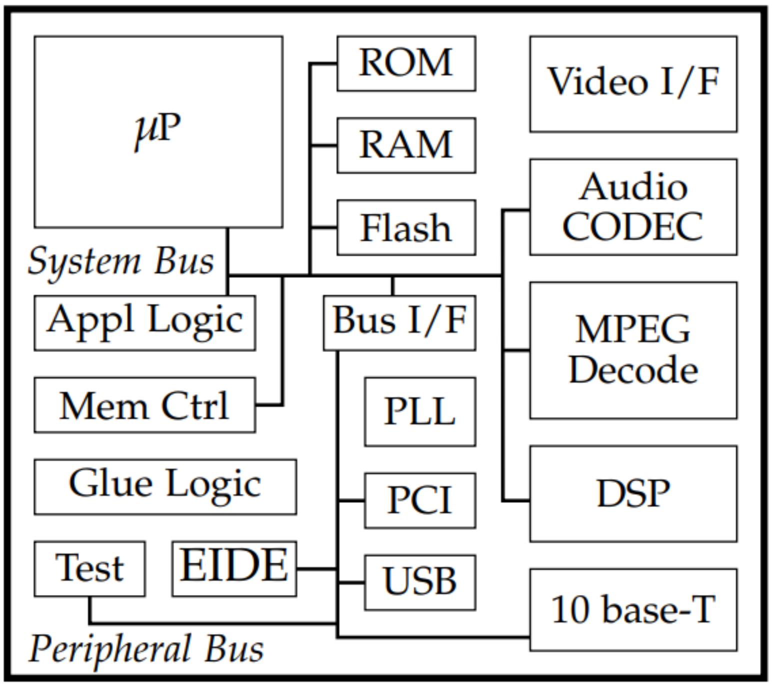


Figure 3 Example of modern System on Chip

Shown above is a traditional SoC design where more functions are integrated to a single chip.

While the above shown SoC has kept the power, efficiency and performance at a single place the increase in internal components made more complex bus communication architecture like crossbar etc. This approach brough in reusability and division of design to different IP and blocks which can be developed independently and integrated to make it work a single system. Now this brings in complexity in the design.

Time to market is what defines today's competition. Each product which makes to market in hurry, fades away as more powerful and performance centric new advance product comes in next coming days. This situation brings in challenges for designers to finish a more complicated design in very short time.

In addition, the SoC performance also depends on module basted or hierarchical style design and its verification. Timing of a chip or SoC determines whether chip will work properly in the given frequency. A timing violated block or module is very unpredictable with respect to functionality it possesses. At module or block level, the flip-flop and latches constraints are abstracted with the use of standard cell. These hard cores timings are guaranteed by proper layout and routing. Hence, the timing constraint for their interfaces should be properly verified. This led to what is known as timing model for any module or IP block used in the design. Third party IPs always provide these models as part of their package. At system level, the flow for timing verification should be able to model properly with the provided interface timing rather than using the hard cores directly, as exemplified in *section **1.2***. These methods reduce the iterations and accelerate the time to market of product with quality.

2.2. DIFFERENT TIMING ANALYSIS MODELS AND METHODS IN SoC

By modelling process variation as random variables, statistical timing analysis can account for worst-case design approaches. Since IC design relies on timing analysis results significantly, integrating the statistical timing analysis requires updates in methodology of the digital design. Additionally, hierarchical timing analysis faces new challenges when accounting for process variations.

The next challenge in hierarchical statistical timing analysis is managing correlations among different modules. Gate delays exhibit spatial correlation, meaning that the delay of one gate can be influenced by the delays of others. For example, the correlation between the delays of two gates in separate modules is determined by their on-die distance. However, this distance is not known during the timing model extraction phase, as the positions of modules are only finalized after they are integrated into the design. Consequently, establishing correlation knowledge between modules in advance is not feasible. Additionally, a module may be instantiated multiple times, such as in multi-core CPU systems, leading to correlations in delays among different instances of the same module. As a result, information about these correlations cannot be incorporated into timing models. Instead, this correlation must be addressed during the timing verification of complete design using extracted timing models. This introduces a new complexity in hierarchical statistical timing analysis, as traditional static hierarchical approaches do not account for inter-module correlations. PT or Prime time which is industry standard tool for most VLSI companies makes possible to do automatic timing model generation using two extremely popular methods which are described as below.

2.2.1 EXTRACTED TIMING MODEL

Once a block has met gate level timing constraint analysis, the internal timing details of the blocks is not needed at the SoC level timing analysis. For that engineer can write a timing model for that block. The main pointers needed from this model is

a) Complete input/output timing characteristic without complete netlist of the block. This constitutes the **timing budget** information which is needed at the SoC level.
b) Not every path inside the block needs to be detailed in the model. Critical timing paths or arcs should be modelled.
c) Original block netlist is replaced by model which contain timing arcs of the block interface.

d) These said arcs has delay information of input and output transition and load values respectively.
e) These models are used for implementation but not for sign-off as these are abstract timing model information.
f) Input to output port combinational path gives from two to four combinational delay arcs where each arc has timing sense information with maximum or minimum delay representing lower or upper bound. This information is for inverting or non-inverting paths present in the block. Like in ***Figure 5,*** represents a combinational delay arc for a pair of non-inverting path between IN and OUT1.
g) Paths to an output port from similarly clocked interface registers yield minimum and maximum sequential delay arcs, defined relative to a specific clock edge. These arcs are computed by summing the clock and data path delays for each register in the clock group, resulting in a single worst-case pair of delay arcs for the output port associated with that clock group. Like in ***Figure 5,*** since there is no inversion happening in clock tree starting from CLK to U8 CLK input, delay arc is extracted from the rising edge of CLK to OUT2.
h) Paths from an input port to the data pins of a clock group lead to the extraction of setup and hold constraint arcs, which are defined relative to a specific clock edge. In the design example, since U7 is rising-edge triggered with no inversion in the clock tree, these arcs are derived for the input relative to the rising edge of CLK.
i) Setup and hold arc equations are

Equation 1 -> Setup check = Data path(Max) – Clock path(Min) + Setup of register

Equation 2 -> Hold Check = Clock path(Max) – Data path(Min) + Hold of register

An illustration of how ETM looks like as below: -

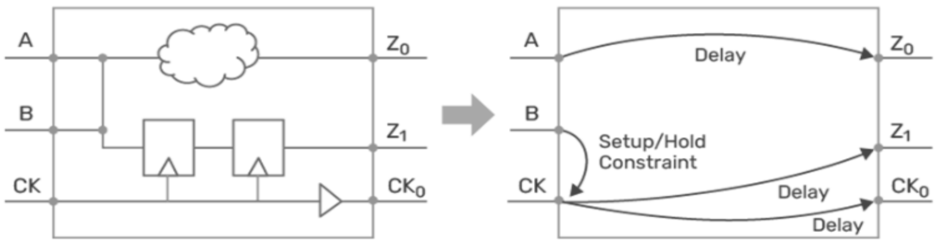


Figure 4 Original Circuit and its ETM Model

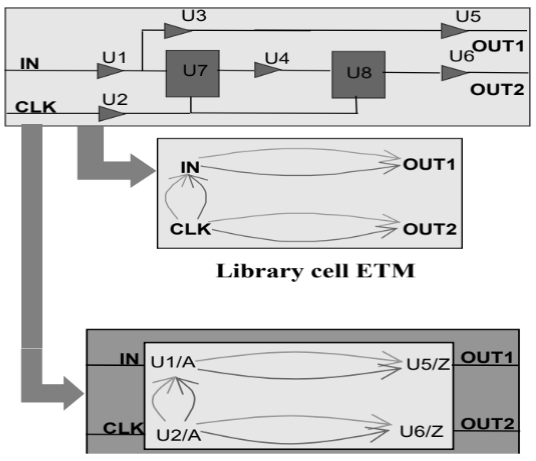


Figure 5 ETMs Arcs

This abstract interface behaviour of a design which replaces the original netlist is a library which consists of timing arcs between pins for that IP. The setup, hold & delay arcs are function of input transition and output load times. The interface timing is a context independent relationship between different pins.

Cons of ETM modelling:

The main issue with ETM is that it requires several steps. First step involves designer to verify the results matches the SDC and netlist. Second step is to generate a full chip SDC with the blocks and IPs or sub-systems replaced with their ETM models. Third step is to verify the second step SDC result matches the flat SDC results. Generation and merging. Signal integrity validation Special tools or flows are needed to achieve this. There are some limitations also, like some tools and flow does not allow multiple clock application on the same pin or ports. Designers have to then take extra steps to provide a model which is equivalent for ETM generation. Accuracy issue for generated clock and modelling hardness for exceptions in design.

For example, below is a situation where one clock is coming from pll slice in the design for a dual quad design and another clock coming from top level port and they are muxed and the output is what drives the IP block. Here IP block clock input can be driven by either of the two sources. Here is the perfect example where an equivalent modelling has to be provided if ETM has to be used.

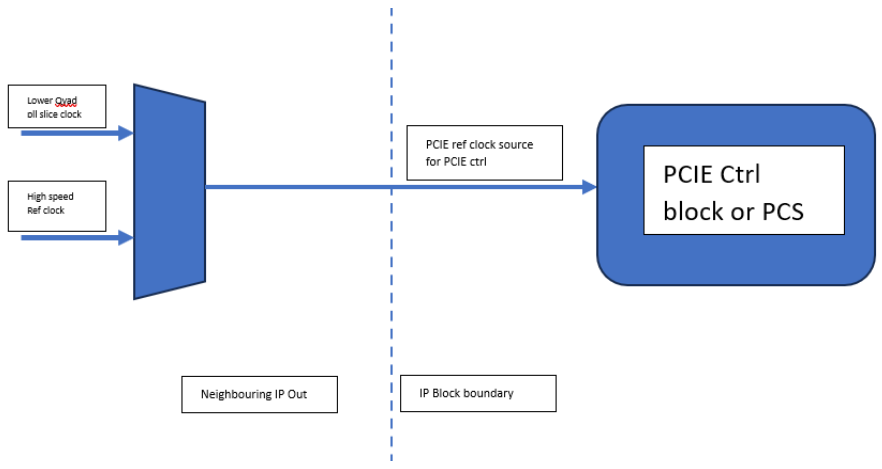


Figure 6 Multiple clock source to single port/pin of IP

The liberty format used for ETM is unable to specify here the master clock for the generated clock. In this case of multiple masters, it's the designer's task to provide external constraint to associate the generated clock with a specified master clock which becomes cumbersome.

Here the only solution is to do a flat SDC timing analysis or create a promoted constraint and then model them with two modes timing model to close the STA. Dual mode means using set case for one clock as master and then set case for another clock as master. This will take more time but will give a correct model that is required by the chip to close on timing analysis to qualify the design correctly.

Again, in some case the interconnects at the port level must be abstracted that can lead to loss in accuracy. This can impact minimal but this also once of the shortcomings of the ETM modelling.

Also, there are some situations where for ETM some path exceptions need to be re-coded or re modelled. These are situations when multiple output delays are present on a port and one of the delays has a MCP (multi-cycle path) through one of block's internal pin. Here the internal path needs to preserved properly, this creates the size of the model. Also, the design team has to manage exceptions such that same timing results are achieved as that of a flat analysis.

2.2.1.1 EXTRACTED TIMING TABLES

As explained above for the three types of extracted timing model arcs: combinational delay arc, sequential delay arc and constraint arc timing values are associated with tables that define values as function of the input transition time and output load values. This way by using an extracted table for timing rather than scalar timing values, every generated model becomes context independent and becomes applicable for numerous transition time and similar output loads. Tables are merged accordingly and appropriately for different critical paths for different transition times.

Delay's Table:

Each extracted delay arc, whether combinational or sequential, is accompanied by two-dimensional delay and transition tables. These tables define timing delays and transition times for rising and falling outputs based on input transition times and output load, with the user controlling the range and number of points for characterization.

As shown ***Figure 7*** to extract the rise delay table for the non-inverting arc from IN to OUT1, the process begins with a two-dimensional cell delay table for U5, which defines rise delays at output pin Z relative to pin A. The rise-to-rise transition table from IN to U5/A captures transition times at U5/A based on IN's transition times; this data is then used to create an intermediate table reflecting delay contributions through U5, which is combined with the delay table from IN to U5/A to produce the final two-dimensional rise delay table for the arc from IN to OUT1.

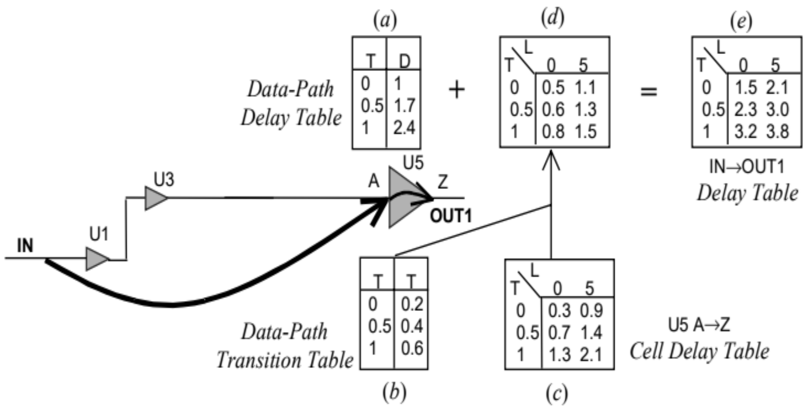


Figure 7 Extracted delay tables

Constraint’s Table:

Each extracted constraint arc is linked to a pair of constraint tables that specify setup or hold times for rising and falling transitions at an input port. These tables are determined by the input transition time and the clock transition time.

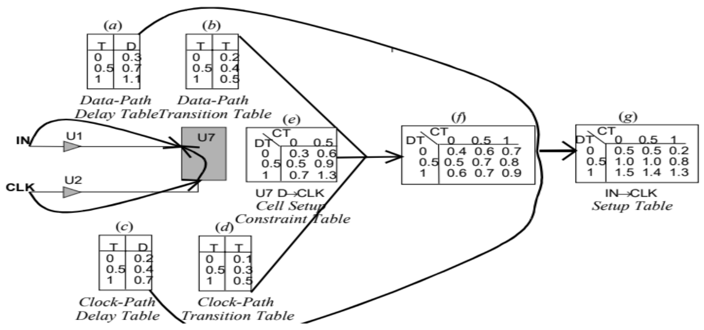


Figure 8 Constraint Extractions Table

As shown in ***Figure 8*** to compute the fall constraint table for the setup arc between IN and the rising edge of CLK, the process starts by extracting the transition tables for the maximum fall-to-fall (ff) path from IN to U7/D and the minimum rise-to-rise (rr) clock path from CLK to U7/CLK. These extracted tables are essential for defining the timing constraints for the setup arc. The extracted transition tables are used to index into the two-dimensional fall constraint table for the setup arc from D to CLK in the library cell for U7, resulting in an intermediate table that defines setup times on register U7 based on transition times from IN and CLK. This intermediate table is then combined with the delay tables for the maximum fall-to-fall path from IN to U7/D and the minimum rise-to-rise path from CLK to U7/CLK to finalize the setup constraints. This is done in accordance to two equation that we saw earlier at *Equation* 1 and *Equation 2.*

2.2.1.2 ETM GENERATION METHOD

In the PT shell the for generating ETM *extract_model* command is used to extract each timing arc. For example,

pt_shell> extract_model -library_cell -output model2 -block_scope

Two types of timing model be created using the above 1) A library cell model 2) A wrapper-and-core model.

The library cell model is the replacement for leaf cell for port-to-port information.

The wrapper-and-core model, consists of all central core leaf cell which are surrounded by a wrapper which preserves the original parasitics of boundary nets.

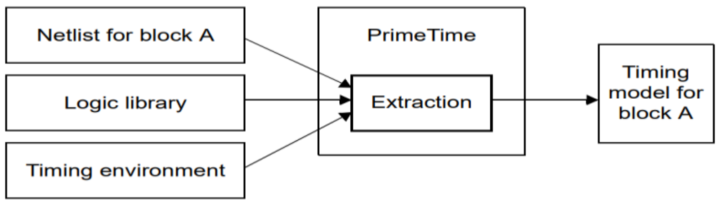


Figure 9 ETM timing model extraction process in PT shell

2.2.2 INTERFACE LOGIC MODELS

Interface logic models or generally known as ILMs represent a structural approach of model generation by replacing the original gate-level netlist with a simplified netlist that includes only the interface logic connecting input/output ports to edge-triggered interface registers. While preserving the clock tree to these registers, ILMs exclude any logic that exists solely in register-to-register paths within a block.

Identifying the interface logic of a design and writing out netlist, its related constraints and any related back annotation in an appropriate format for the interface logic. This is the basic generation requirement for ILM.

1) Each cell has timing path from input to a register (edge triggered) or from input port to an output port. Let's assume that a Latch which is transparent is encountered in the path then it is considered as combo logic and the path is traced until an edge triggered register is meet.
2) Same understanding is used for a transparent latch when the path is from a register (edge triggered) to output port.
3) Any interface registers with the associated clock driving it and any register in clock tree, together belongs to interface logic. Register driven clock gating structure is not part of interface logic but clock gating driven by external port is.

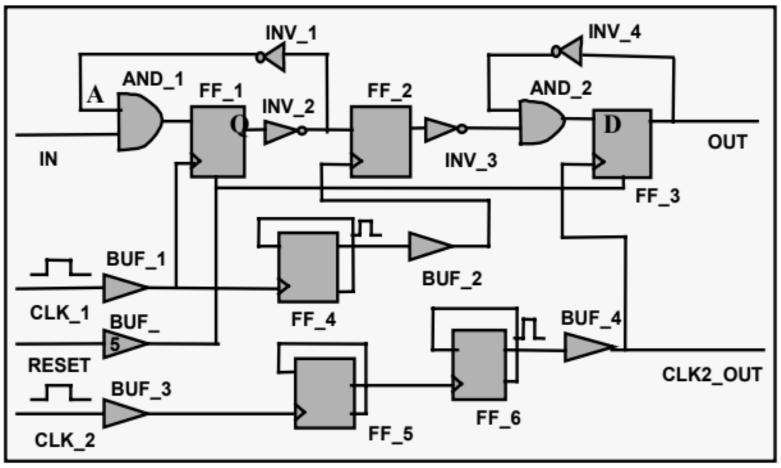


Figure 10 Gate level netlist example

Figure 10 gives us an example of how a netlist looks like for a block. The below *Figure 11* gives us its interface logic results. If carefully observed the ILM does not have any *INV_*, AND_* and FF_*.* These mentioned cells are only instantiated in the netlist for a reg-to-reg timing path or arc. However, we do find the *AND_1* in the ILM without any A pin mentioned in it due to point (3) from above. Similarly, *FF_1/Q to FF_3/D* is also left unconnected due to point (3) again as clock tree for both the flops are different i.e, CLK_1 and CLK_2 respectively.

For identifying interface logic, it is very important to ignore fanout of input ports that are connected to global chip level signals feeding internal registers. For example, in *Figure 10* the fanouts of *CLK_*, RESET* are completely ignored. However, the clock tree for a particular interface register is kept for processing.

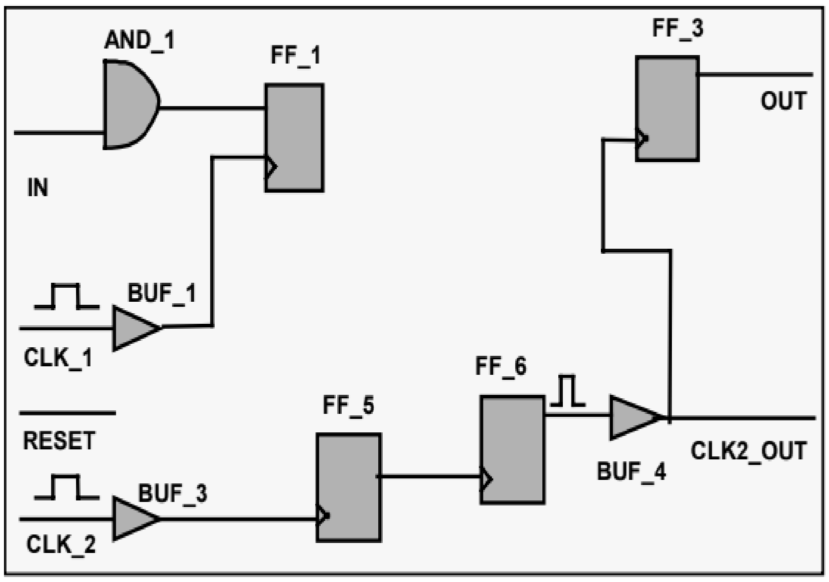


Figure 11 ILM of example gate level netlist

An ILM can provide benefits in STA as following:

a) For a range of operating environment, the model is accurate.
b) Runtime is less as well as the runtime memory requirements.
c) SPEF (standard parasitic exchange format) or SDF (standard delay format) data can be incorporated.
d) SI crosstalk and noise analysis can be done in these models.

2.2.2.1 ILM GENERATION METHOD

In the PT shell the for generating ILM *create_ilm* command is used for a block. For example,

```
pt_shell> create_ilm -block_scope -verfication_script -parasitic_options {spef \
              input_ports_nets constants_nets}
```

Now, to validate the timing of ILM model one has to generate *interface timing report* for the original netlist block using command *write_interface_timing*. Below is the example interface report as shown in *Figure 12*

```
****************************************
Command: write_interface_timing
        -significant_digits 6
Design : blockA
...
*********************************************
Section: slack
Info    : Worst-case slack for each port and path group
Design : blockA
*********************************************

Generated Clock and Source Info:

Attributes:
   L<n> - latch level where <n> is 0 for first level

                              Arc         Worst-case
From          To              Type        Slack
---------------------------------------------------
EN(r)         clock1(f)       setup        3.826993
EN(f)         clock1(f)       setup        3.805498
EN(r)         clock1(f)       hold         5.903111
EN(f)         clock1(f)       hold         5.923214
ain[0](r)     clock1(r)       setup       13.671270
ain[0](f)     clock1(r)       setup       13.650400
ain[0](r)     clock1(r)       hold        -3.937757
ain[0](f)     clock1(r)       hold        -3.929484
```

Figure 12 Interface timing report for example desing

Now the above report has to be compared with the block's netlist timing report using the command *compare_interface_timing* which gives the result as shown in *Figure 13*

pt_shell> compare_interface_timing block_netlist_timing.rep ilm_timing.rep

```
****************************************
Command: compare_interface_timing
         block_netlist_timing.rep blockA/ilm_timing.rep
         -session full
...
****************************************
                         Arc           Slack
From         To          Type    Ref        Cmp       Diff     Status
----------------------------------------------------------------------
EN(r)        clock1(f)   setup   3.83       3.83      0.00      PASS
EN(f)        clock1(f)   setup   3.81       3.81      0.00      PASS
EN(r)        clock1(f)   hold    5.90       5.90      0.00      PASS
EN(f)        clock1(f)   hold    5.92       5.92      0.00      PASS
ain[0](r)    clock1(r)   setup  13.67      13.67      0.00      PASS
ain[0](f)    clock1(r)   setup  13.65      13.65      0.00      PASS
ain[0](r)    clock1(r)   hold   -3.94      -3.94      0.00      PASS
ain[0](f)    clock1(r)   hold   -3.93      -3.93      0.00      PASS
...
         Totals     Slack    Transition Time   Capacitance    Rules
---------------------------------------------------------------------
Passed   1098       258              280             560        -
Failed      0         0                0               0        0
Total    1098       258              280             560        0
```

Figure 13 Compare ILM report to netlist timing report

If the compared generated report shows the values of *slack, transition times and capacitance match* then one has successfully validated the ILM timing. ILM limitations include mismatch of constraints, pessimism of arrival, latch-based design which require specific flows and stitching, routing for over the block.

Cons or limitations of ILM are:

- In **ILM** we can see the timing logic or arc to the first level FF from input and from the last level flop to output pins but not the internal reg-to-reg path and if optimization needed inside the partition, then it becomes very difficult. Also, over-the-block routing is another limitation as most time this is not considered during characterisation and if any such case arises, it can cause signal integrity issue such cross coupling etc as shown below for original *Figure 14* and the mentioned issue in *Figure 15* where top net Ta is causing coupling issue to earlier model.

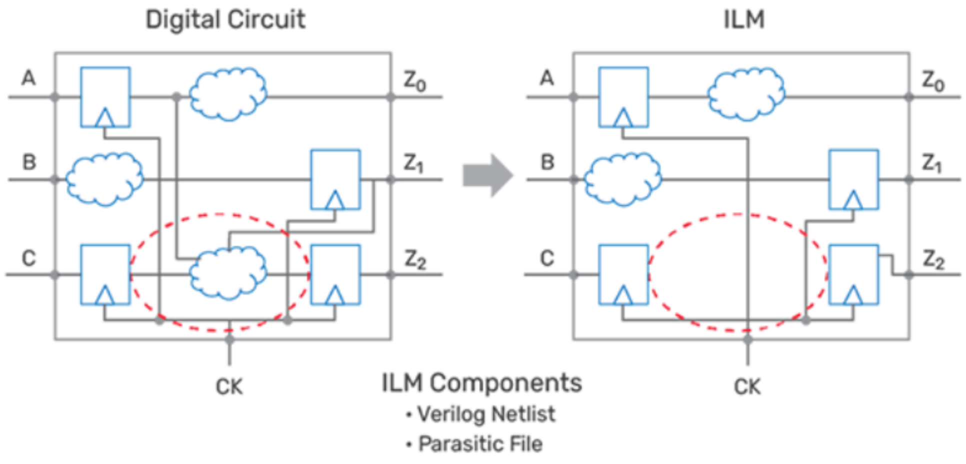


Figure 14 Example of ILM implementation of digital circuit

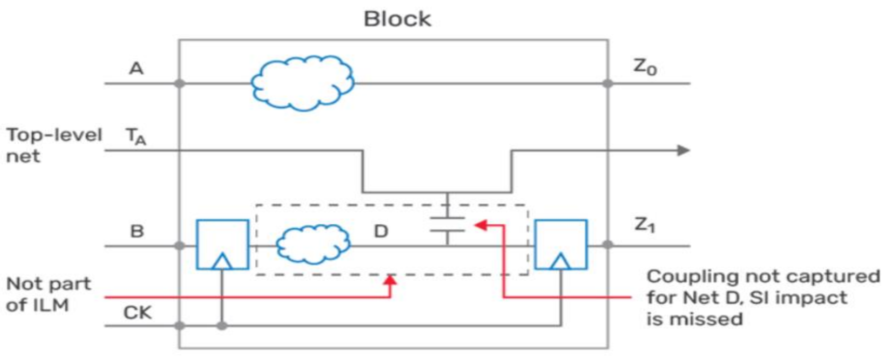


Figure 15 Over the block coupling issue in ILM

- Another issue of ILM is if correct constraint context is not preserved while moving from block to next level top, the timing analysis ends up in major setbacks during hierarchical analysis.
- Timing arcs are precalculated in **ILM** creation. Any CPPR effect from top level requires complex and special care to remove pessimism coming in arrival time window.
- Latch batched designs are common now a days and for load capacitances and signal integrity aggressors which impact ILM model impacts storage and hierarchical timing analysis runtime.

2.2.3 QUICK TIMING MODEL (QTM)

In early stage, if a block has no netlist, then STA engineer sometimes uses a model which is known as QTM or quick timing model. Once netlist is available then this model can be replaced for more correct timing.

A series of command is used to define QTM. One can define the *setup, hold* on the input ports, the delay for *clock-to-output* and *input-to-clock*, loads on input and drive strength on output.

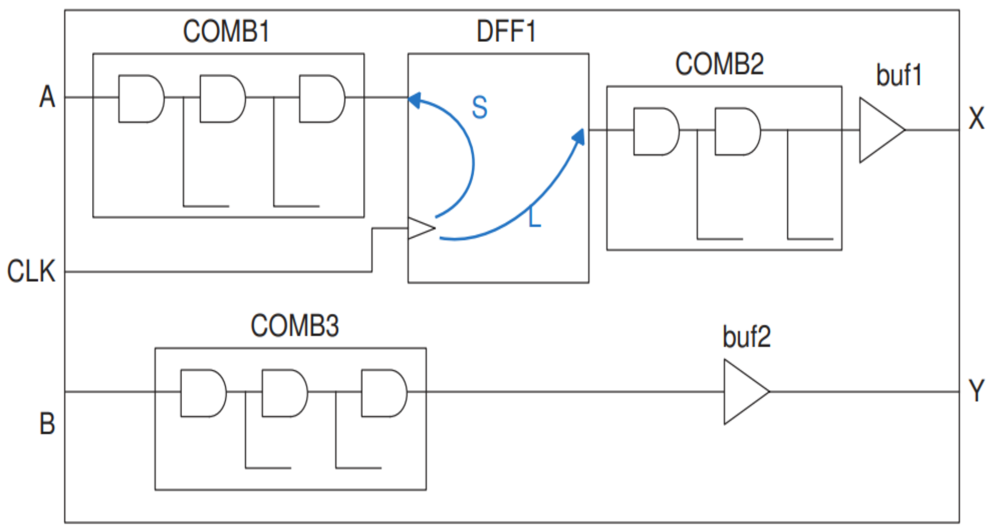


Figure 16 QTM example

PT tool calculates constraints and delay times as follows:

```
• Setup or hold time
    = number_of_levels * (delay/level) + global setup or hold time

• Clock-to-output delay time
    = number_of_levels * (delay/level) + CLK-to-output delay
      + load-dependent output delay

• Input-to-output delay time
    = number_of_levels * (delay/level) + load-dependent output delay
```

While instantiating the quick timing model, instantiate same as how a library cell is instantiated in the design, link the design or model and continue with reporting timing etc.

This below script creates a model for the above design example:

```
### Create the model and set global model parameters ###

# Specify the model name
create_qtm_model qtm_example

# Specify the logic library
set_qtm_technology -library my_lib


# Define a path type; path1 is a 2-input NAND with a fanout of 2
create_qtm_path_type path1 -lib_cell nand21 -fanout 2

# Define a load type
create_qtm_load_type load1 -lib_cell and2

# Define the drive types
create_qtm_drive_type drive1 -lib_cell buf1
create_qtm_drive_type drive2 -lib_cell buf2

# Define the global setup time, which is equivalent to the
# setup time of the DFF1 library cell
set_qtm_global_parameter -param setup \
  -lib_cell DFF1 -clock CLK -pin D

# Define the hold time
set_qtm_global_parameter -param hold -value 0.0

# Define the clock-to-output delay time, which is equivalent to
# the launch time of the DFF1 library cell
set_qtm_global_parameter -param clk_to_output \
  -lib_cell DFF1 -clock CLK -pin Q

### Specify ports, constraint arcs, delay arcs ###

# Create a clock port and set the load
create_qtm_port {CLK} -type clock
set_qtm_port_load {CLK} -value 3

# Create the input ports and set the loads
create_qtm_port {A B} -type input
set_qtm_port_load {A B} -type load1 -factor 2

# Create the output ports and set the drives
create_qtm_port {X Y} -type output
set_qtm_port_drive X -type drive1
set_qtm_port_drive Y -type drive2

# Create the setup and hold constraint arcs
create_qtm_constraint_arc -setup -edge rise -name SetupA \
  -from CLK -to A -path_type path1 -path_factor 2

create_qtm_constraint_arc -hold -edge rise -name HoldA \
  -from CLK -to A -path_type path 1 -path_factor 2

# Create the clock-to-output and input-to-output delay arcs
create_qtm_delay_arc -name CLKtoX \
  -from CLK -to X -path_type path1 -path_factor 2

create_qtm_delay_arc -name BtoY -from B \
  -to Y -path_type path1 -path_factor 3


# Save QTM to qtm_example library cell in the myqtm.db file
save_qtm_model -format db myqtm -library_cell
```

2.2.4 HYPERSCALE ANALYSIS, CONTEX CHARACTERIZATION & MODELING

Hyperscale context modelling is something which is new and followed by some advanced SoC program and also as a method which is ventured into for the proposed thesis work also. For a large chip or SoC, performing flat timing at the top level can consume too much memory resource and huge runtime. Here, hyperscale analysis with context characterization can help in completing the STA task well withing the project timeline.

Hyperscale is a method where the block and top portions are analysed separately and then accurately handles the timing interface between the top and the lower-level blocks.

Context characterization supports writing out the context of timing of a block within the top. Once that is done STA engineer can analyse the timing of block in isolation from the top during the block analysis session. It produces timing results accurately for a block operating in the context of top level.

The below *Figure 177* gives the summary of fully flat analysis and hierarchical analysis using hyperscale method.

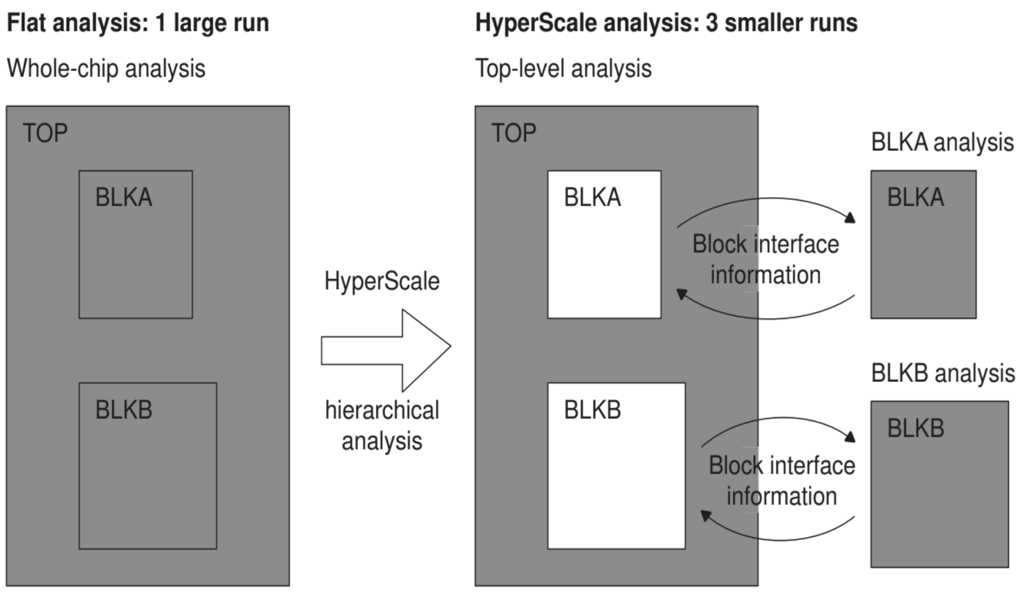


Figure 17 Flat Analysis Vs Hyperscale analysis

In the hyperscale flow, context accurate model is created for each lower blocks. This allows each block to be analysed separately. During this analysis the constraint and timing characteristics like budgeting of higher-level design is also considered. This also helps to analyse the higher-level design using efficient representation of lower block level which excludes any internal timing information. Information like input/output-delay, timing exceptions, crosstalk and CRPR are handled during this top-level analysis with lower block context. Results are with same accuracy as flat analysis with reduced turn-around time and low memory requirements.

Following are the benefits:

- Faster runtime compared to full chip flat analysis. This is achieved as we are intelligently reusing the data from block and top-level analysis which reduces the computation.
- Same delay computation is performed in hyperscale as how done in flat analysis including keeping the integrity effects across block boundaries.
- The flow automatically collects block level timing data which simplifies the top-level analysis. This streamlines the data which eliminates the designer's effort to create, validate or manage any timing model and maintains the block's timing context enabling context aware analysis as separate.
- This flow provides constraints management as one can close block level timing using the context information extracted from top level timing analysis. Latest constraints from the top-level timing overwrite the block level during top level analysis.
- At SoC level there can be multiple hierarchical levels. As shown in *Figure 18* Hyperscale helps in analysis of any number of such design across multiple levels.

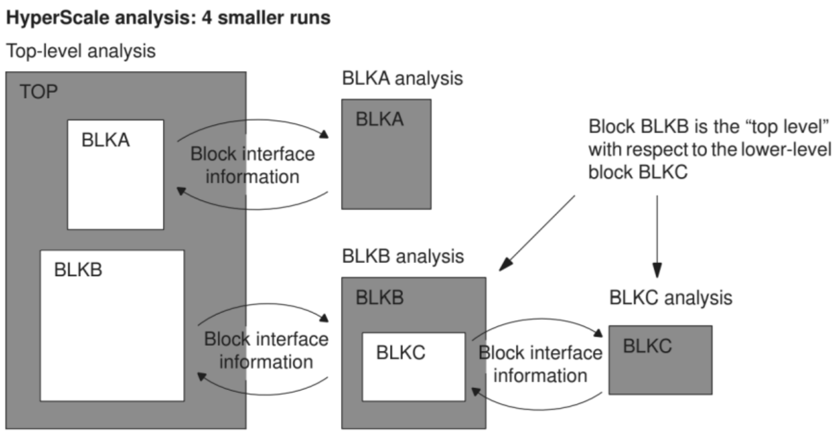

HyperScale analysis: 4 smaller runs
Top-level analysis
TOP
BLKA
BLKB
BLKA analysis
BLKA
Block interface information
Block BLKB is the "top level" with respect to the lower-level block BLKC
BLKB analysis
BLKB
BLKC
Block interface information
BLKC analysis
BLKC
Block interface information

Figure 18 Multi layer hyperscale analysis

2.2.4.1 HYPERSCALE MODELING

Hyperscale model of a block consists of following logic and represented in *Figure 19*:

- Boundary logic for Input-to-register with additional side input pins information for modelling worst paths.
- Boundary logic for register-to-Output with additional internal path stub pins information to model effects on loads in the network.
- High fanout pins which represents the removed high fanout logic timing.

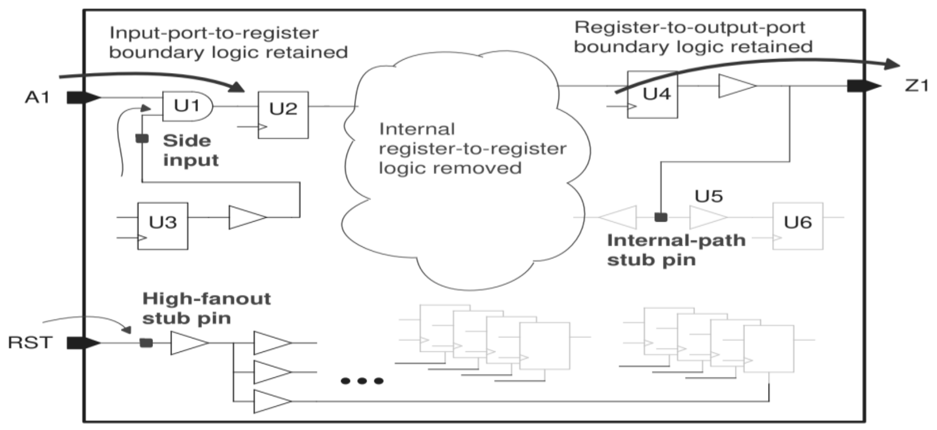

Input-port-to-register boundary logic retained
Register-to-output-port boundary logic retained
A1
U1
U2
Side input
U3
Internal register-to-register logic removed
U4
Z1
U5
U6
Internal-path stub pin
High-fanout stub pin
RST

Figure 19 Hyperscale block model

Hyperscale usage flows are:

1) Bottom-up analysis without context block models: using the budgeted timing constraints block timing is analysed. This method is used if one is not sure of the timing environment for the end use of block. This replaces the ETM model. For this approach the ETM script if available can be modified to generate hyperscale model for blocks.
2) Top-down analysis using context block models: top level and block level are separately analysed and each block level analysis reads in last top level analysis context. Top level uses hyperscale model generated by block level. This gives best performance comparable to flat analysis.
   Top-down analysis can be done with multiple instantiated modules or MIMs also. Multiple instances of the same block can be represented as a single module by merging the context of multiple to one.

   To summarize between the two how to decide which flow to be approached with, we can follow the pointers given in *Figure 20* to decide upon the best of two flows.

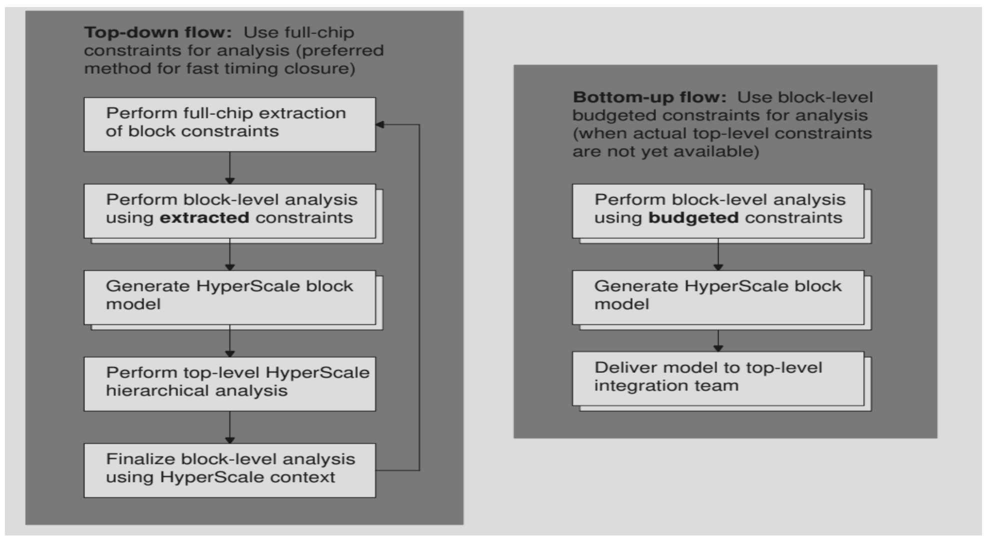

Top-down flow: Use full-chip constraints for analysis (preferred method for fast timing closure)
Perform full-chip extraction of block constraints
Perform block-level analysis using extracted constraints
Generate HyperScale block model
Perform top-level HyperScale hierarchical analysis
Finalize block-level analysis using HyperScale context
Bottom-up flow: Use block-level budgeted constraints for analysis (when actual top-level constraints are not yet available)
Perform block-level analysis using budgeted constraints
Generate HyperScale block model
Deliver model to top-level integration team

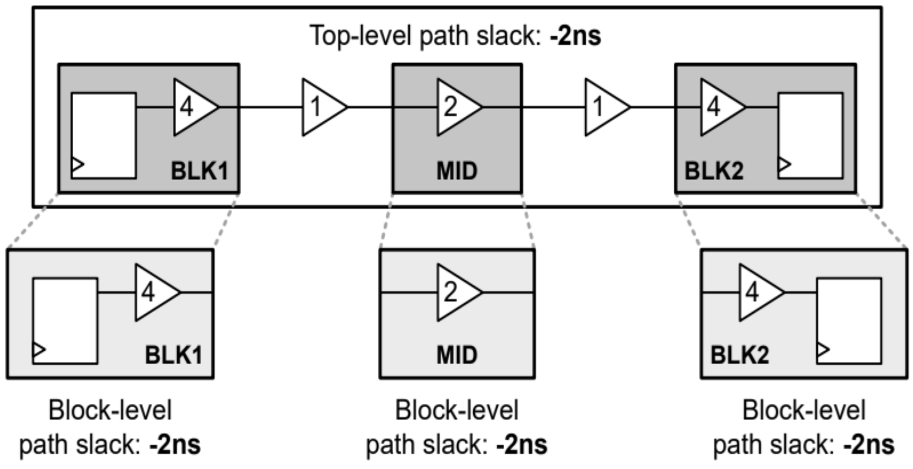
Figure 20 Hyperscale Analysis flows top down and bottom-up

Context Characterization and Context budgeting:

Another important part in the hyperscale model is context characterization and budgeting. In the characterization process the timing context of the block is captured from the environment of its parent design. This includes information related to clock, arrival times for input, delay times for output, timing exceptions, DRC information, propagated constraints, input drives, capacitive loads and wire load model etc.

Next is context budgeting where the context of characterized block is adjusted such that slack are distributed across the blocks. This prevents blocks from over optimized or under optimized at the top or chip level.

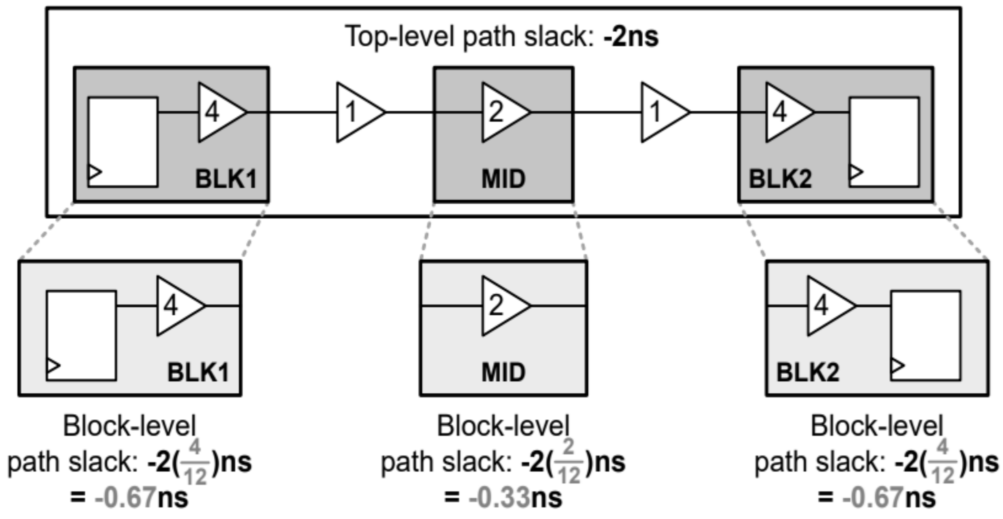
Context budgeting difference can be seen from below *Figure 21* and *Figure 222*. In below figure we can see how traditionally parallel characterized context can be seen for slack at top level.



Top-level path slack: -2ns
4
1
2
1
4
BLK1
MID
BLK2
4
2
4
BLK1
MID
BLK2
Block-level path slack: -2ns
Block-level path slack: -2ns
Block-level path slack: -2ns

Figure 21 Replication of context characterization at top level slack at blocks

Using context budgeting from PT tool we see the below differences for the same at top-level blocks as shown in below figure:



Top-level path slack: -2ns
4
1
2
1
4
BLK1
MID
BLK2
4
2
4
BLK1
MID
BLK2
Block-level path slack: $-2(\frac{4}{12})$ns = -0.67ns
Block-level path slack: $-2(\frac{2}{12})$ns = -0.33ns
Block-level path slack: $-2(\frac{4}{12})$ns = -0.67ns

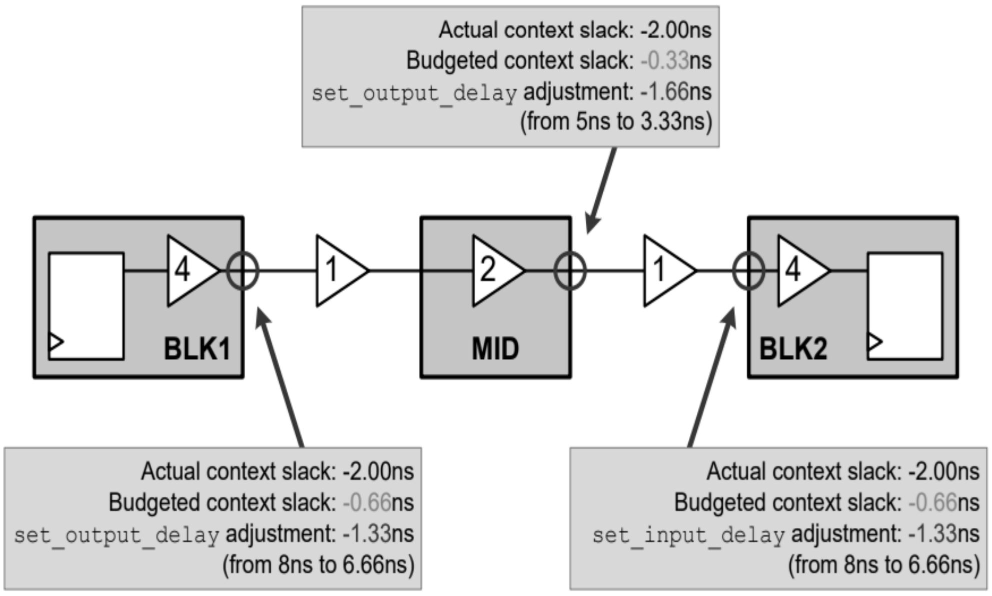
Figure 22 Top level slack allocation across blocks using context budgeting

How this is achieved?

This works by performing regular context characterization then adjusting those such that slacks are reallocated across paths.

- Register-to-output paths of block level are adjusted at block's output ports.
- Input-to-output paths of block level are adjusted at block's output ports.
- Input-to-register paths of block level are adjusted at block's input ports.

Each adjustment location the constraints that are adjusted are *set_input_delay* or *set_output_delay*. These are adjusted by the difference between the slack actual and budgeted slack. Below figure gives a rough picture how it's happening:



Actual context slack: -2.00ns
Budgeted context slack: -0.33ns
set_output_delay adjustment: -1.66ns
(from 5ns to 3.33ns)
4
1
2
1
4
BLK1
MID
BLK2
Actual context slack: -2.00ns
Budgeted context slack: -0.66ns
set_output_delay adjustment: -1.33ns
(from 8ns to 6.66ns)
Actual context slack: -2.00ns
Budgeted context slack: -0.66ns
set_input_delay adjustment: -1.33ns
(from 8ns to 6.66ns)

Figure 23 Context budgeting example

The path delay is sum of *CLK-to-Q of Launch Flop + combo path delay + capture flop CLK-to-D.*

The launch and capture register clock are not included in data path delay calculations itself but they are considered as they affect the path slack that is being budgeted.

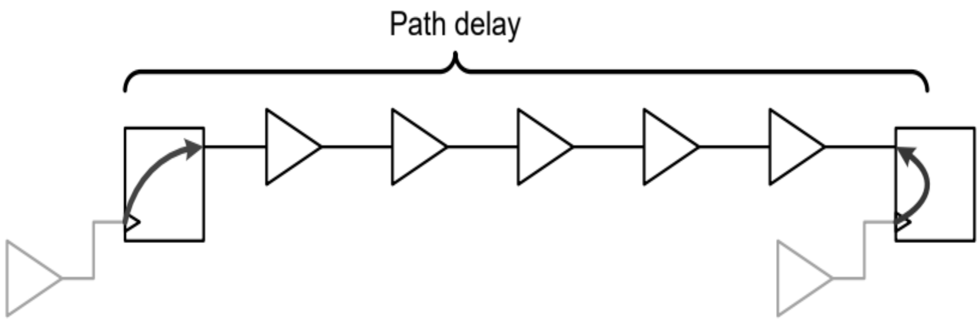


Figure 24 Path delay calculation

CHAPTER 3. PROPOSED WORK

The traditional STA evaluates and analyse timing of a chip by setting some process parameters at what is known as corners. For example, check for setup time is done at worst corner where gate delays and interconnect delays are worst or largest. With the advent and massive development in the chip design into nanometres these corner-based timing analysis methods faces new challenges. Along with the corner-based issues which is more of a device level thing, timing aspect and timing closure also made a shift from flat to hierarchical. In the proposed work along with the design explanation the approach that is going to be used for the timing closure is also discussed in this chapter.

3.1 FUNCTIONAL DESCRIPTION OF DESIGN

The design consists of two parts, Control path and Data path. The control part consists of Processor, NoC and Data path consist of Data Synchronizer block to which has FIFO, Multiplexer, De-multiplexer, Ethernet controller, and Serdes planes. Each of these below blocks partitioned into separate sub-system for a SoC. This whole SoC is done in TSMC 6nm technology. The Serdes is 5.4Gbps with system side data width of 32 bits. The whole design works on 400Mhz to 500Mhz including NoC, APB interface for CSR access.

SYSTEM DESCRIPTION:

(a) Processor Control module.
(b) System NoC for routing.
(c) Data Synchronizer Block.
(d) Ethernet Controller.
(e) Multiplexer plane.
(f) Serdes plane.

The description of each major blocks and components are as explained below:

a) **Processor Control Module:**
The controller constitutes a RISC Processor, CSR & PLL. The clock and reset network come from this sub-system and distributed across the chip. The processor is RISC with Harvard architecture, 6 stage pipeline with both data and instruction cache. The control and status register or otherwise known as CSR is used to store the register information after power on reset. PLL is analog block which receives the reference clock from board crystal and using VCO inside generates the required clock frequency for operation.

b) **NoC:**
Network on chip is for routing the different configuration from the Processor to different sub-systems. It’s a low bandwidth subsystem.

c) **Data Synchronizer block:**
This block receives the raw data from fabric side from 4 channels and then pass them to the next block which is Ethernet controller. The main purpose of this block is to channel the

correct received data and clock to a FIFO and then as per our receiving domain which is driven by the protocol specification, gets the data out of FIFO.

d) **Ethernet controller block**:
This block consists of controller for Ethernet with protocol specification 1G/10Gbps Ethernet. It has MAC, PCS submodules which also has some error correction methodology defined in it.

e) **Multiplexer plane:**
The receiving system side has 4 channel and it needs Mux-ing between them before sending to Synchronizer. Once out of controller block the data needs to be passed down to the two Serdes setup.

f) **Serdes Plane:**
The design is 4 lanes comprising of 2 SerDes. Each SerDes has 2 lanes where a lane is comprised of 2 differential signalling pairs with one pair receiving data and one pair transmitting data. It has a PCS block that implements some of the common functions such as 8B/10B, 64B/66B encoding/decoding, clock compensation and rate matching, scrambling/descrambling, word alignment, gearbox functions etc. The PMA receives and transmit differential serial data. It converts serial to parallel data for processing in the PCS. It also includes CDR for clock and data recovery.

3.2 DIAGRAM & DESIGN CONSIDERATIONS

Figure 25 below shows the representation of design on which the current work is being done.

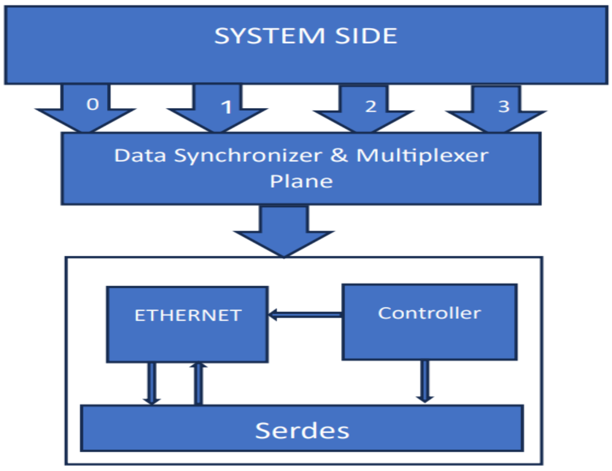


Figure 25 Reference Design Diagram

The system side can have user logic as per user defined for data processing. The data synchronizer and Multiplexer block makes up a single subsystem. The Ethernet, Processor, NoC and Serdes is what comprises and called as High speed Serdes Interface. This SS uses ethernet, MAC, PCS, PMA sub modules. Mac uses the IEEE 802.3 standard which while sending data on network will encapsulate the high-level frames into appropriate frames to transmit via transmission medium. PCS provides interface between PMA and MII and is use for data encoding/decoding, scrambling, marker insertion for alignment and at RX side block and symbol redistribution function.

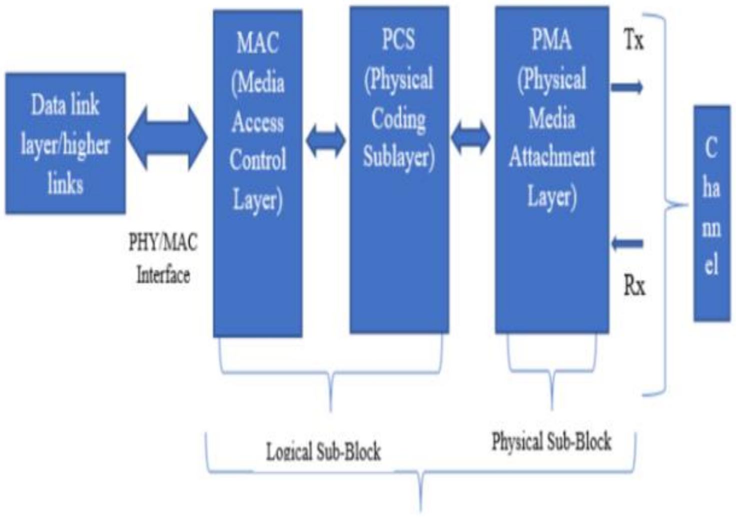

Data link layer/higher links
PHY/MAC Interface
MAC (Media Access Control Layer)
PCS (Physical Coding Sublayer)
PMA (Physical Media Attachment Layer)
Tx
Rx
C ha nn el
Logical Sub-Block
Physical Sub-Block

Figure 26 MAC, PCS and PMA sub modules

For the controller block which consists of processor, PLL and CSR, oscillator generates a signal with a particular frequency that is passed down to PLL which generates the multiple of given signal which is then passed down to the rest of the design and which works as a word clock also for the parallel data processing before being passed down to the SerDes plane.

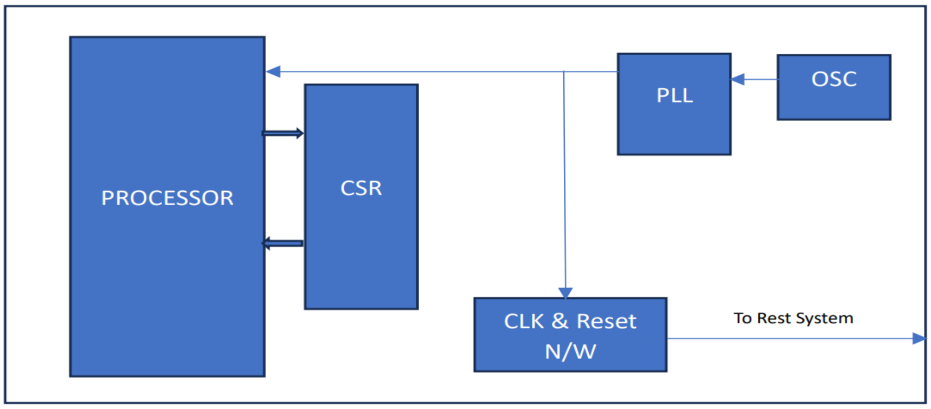


Figure 27 Controller Module and Clock Reset network

3.3 CONSTRAINT CREATION SCHEME

- The clock coming out of controller module drives the NoC which drives the rest of the system.
- The system clock works on the highest frequency of 500Mhz on which the CSR for SerDes and other modules and set.
- OSC clock is the Keep alive clock driven to SerDes for initial processing.
- The SerDes works at 10Gbps and system side bus is 40 bits wide. A gearbox in SerDes plane manages the data to SerDes.

- Serdes Plane has its own PLL that receives a reference clock and generates the clock for serializer and de-serializer block.
- Input and output delays are specified with 70-30 ratio respectively which will be corrected as per final netlist level runs.

The design's high level clocking scheme is as below diagram. Osc_clk is what goes to the processor plane which using a local pll generates the clock on which the whole SoC works. The system side works on higher rate clock than word clock which is same as SerDes side word clock. Hence, we need synchronizer module to take care data transfer from fabric side to SoC transmitter side. The reference clock manages the PLL for Serdes to generate high speed clock for the transmitting the data.

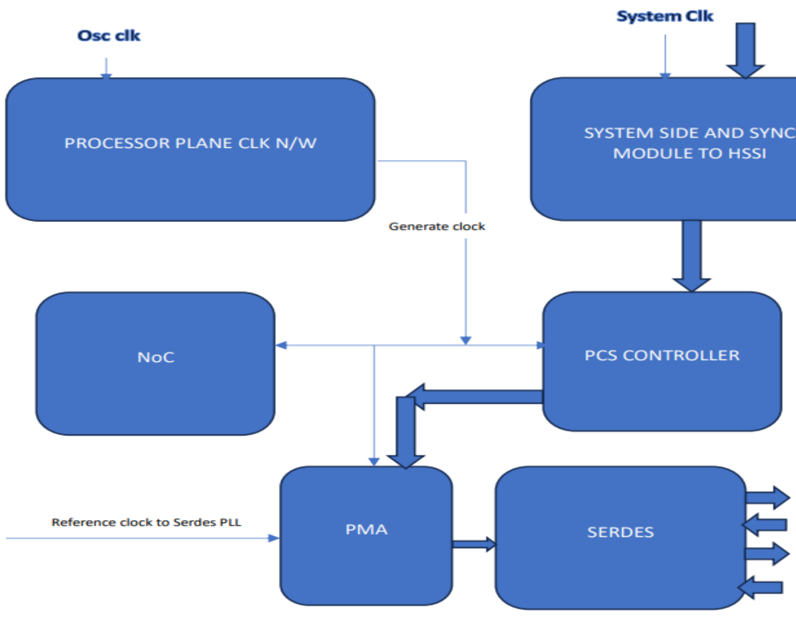


Figure 28 High level Clocking Scheme

Constraint Writing strategy for current considered design:

- Front End designer will give a basic functional clocking constraints or SDC for each subsystem or at block level such as "**creat_clock", "create_generated_clock"** & clock groupings like **"set_clock_groups"**.
- Along with the clocking constraints other SDC constraints like, exceptions such as multicycle values, false path constraints will also be written as per designer's design intent such as **"set_multicycle_path"** & **"set_false_path"**.
- Interface constraints such as **"set_input_delay" & "set_output_delay".** The values for these should be as per the technology which we are working on.

Once block level and SS level constraints are created, the next step which is followed is **Constraints Promotion** using Synopsys Timing Constraint Manager which is built on Fishtail tool. This step involves some manual and mostly automation efforts to manage the constraints to next level from IP to SoC.

The holistic approach which is followed for this SoC design can be represented using a diagram as below:

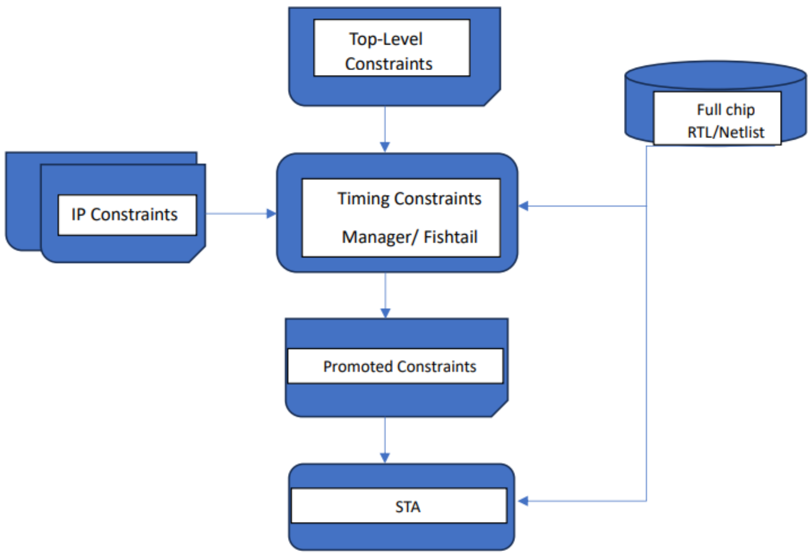


Figure 29 Constraint Promotion Flow

Hyperscale method which is the prescribed methodology discussed in this project also helps in promoting or writing top level constraint by mapping and promotion. This method is called hierarchical constraint extraction method. It shall be discussed further in the next section where the proposed approach for STA at SoC level is discussed in detail.

For example, we have a port *CLKIN* in named boundary *block.* When used constraint promotion and extraction feature is used then the scenario can be as below:

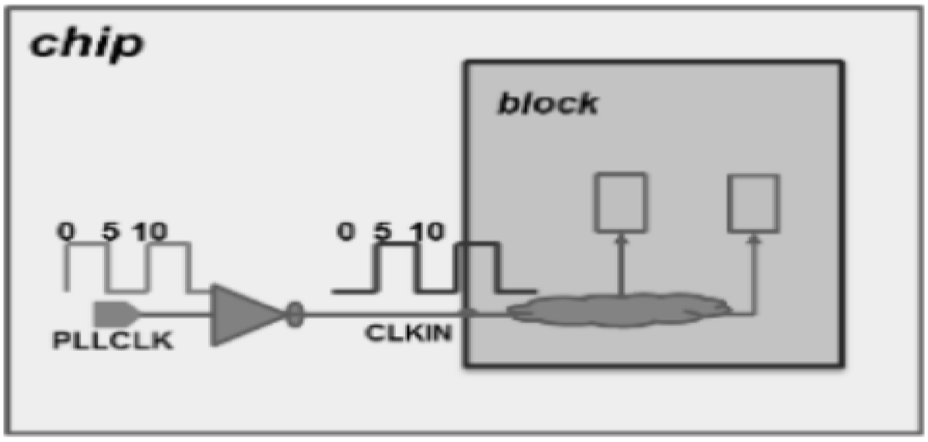


Figure 30 Clock mapping and constraint promotion

Using the constraint promotion feature we get the below results:

Original constraints:

```
create_clock -name PLLCLK -period 10 -waveform {0 5} [get_pins PLL/OUT]
set_input_delay -clock PLLCLK ...
set_clock_latency -source -rise -clock PLLCLK ...
set_false_path -rise_from PLLCLK ...
...
```

Constraints generated by constraint extraction:

```
create_clock -name PLLCLK -period 10 -waveform {5 10} [get_ports CLKIN]
set_input_delay -clock PLLCLK ...
set_clock_latency -source -fall -clock PLLCLK ...
set_false_path -fall_from PLLCLK ...
...
```

Apart from this, the margins are also provided in top level run. This can be applied to all blocks or some particular cell and its pins, like below

```
set_context_margin 0.2   ;# apply to all blocks
set_context_margin 0.4 [get_cells {UBLK*}]
set_context_margin 0.5 [get_pins {UBLK2/RSTN}]
```

This margin can be absolute or a percentage of existing delay from original SDC or it can also be a percentage from clock period. These also has details of whether minimum or maximum or with respect to clock edge like rise or fall, to data path or clock path as latency etc.

These steps are called **automatically fixable block boundary violations.** The below *Figure 31* shows an example where this feature can be applied.

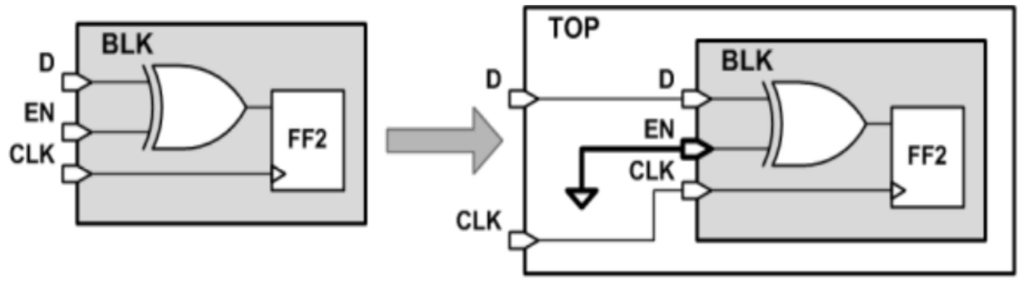


Figure 31 Example of fixable boundary violation for block

The *report_constraint -boundary_check* gives the below report:

```
HyperScale constraints report

  Constraint                                      Num_Violations
  ------------------------------------------------------------
* data_arrival                                                 1
* global_timing_derate                                         1
* boundary_logic_value                                         1
```

Figure 32 Boundary check violation report snip

These are caused by the differences in case analysis and derating differences. It has also caused delay to the arrival time of D pin of blk. The *report_constraints* gives the below summary report as shown:

```
Constraint: global_timing_derate

  Instance      Derate type    Window type    Block     Top      Slack
  ---------------------------------------------------------------------
  BLK           data_cell      max_rise        1.00     1.10     -0.10
  BLK           data_cell      max_fall        1.00     1.10     -0.10
  BLK           clock_cell     max_rise        1.00     1.10     -0.10
  BLK           clock_cell     max_fall        1.00     1.10     -0.10

Constraint: boundary_logic_value

  Instance      Pin                             Block                Top
  ---------------------------------------------------------------------
  BLK           BLK/EN                            0                    1

 Constraint: data_arrival

  Instance    Pin           Window        CLK     Block     Top      Slack
              (Port name)   type
  ---------------------------------------------------------------------
  BLK         BLK/U1/A(D)   max_rise    CLK(r)    0.00      0.25     -0.25
  BLK         BLK/U1/A(D)   max_fall    CLK(r)    0.00      0.20     -0.20
```

Figure 33 Report showing automatic constraints violations

Now these are corrected in further runs as these are automatically correctable ones. These also does not require the constraints update to the original constraints but it is desirable to update the block level constraints such the future runs benefits from the improved constraints of the block.

Certain constraints need **manually fix** for violations such as which occurred in clock mapping, clock attribute, or clock characterization mismatch.

For example, one of the block constraints defined a clock with period 10 which is incorrect when the actual should have been 12 from the top level.

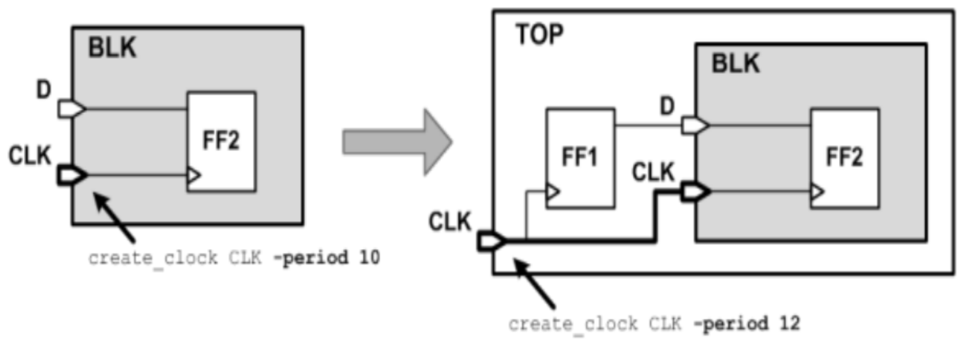


Figure 34 Example manual boundary violation

Again, when report_constraint is run we get block boundary violations

```
HyperScale constraints report

  Constraint                                           Num_Violations
  -----------------------------------------------------------------
  clock_mapping                                                     1
```

These violations must be fixed before moving to the next run process. These can be fixed by modifying the constraints to improve the context for top level. Also, these can be fixed by correcting block level constraints.

3.4 STA USING HYPERSCALE TOP-DOWN APPROACH WITH CONTEXT BUDGETING

In this proposed approach top level and block level is separately analysed and the top-level analysis gives out or writes out the lower-level block's context also. Now each block level analysis will read in the context from the previous last top level and writes out a block model which is hyperscale timing interface details. Every subsequent top-level analysis now reads in this generated hyperscale model of block. Top-down flow is considered here as it has been seen that timing closure is less complex and faster to achieve completion. It also guarantees that the block level constraints are to be consistent with that of top level from the starting of design phase.

The below *Figure 355* shows the analysis with the code snip used to enable such run in pt shell.

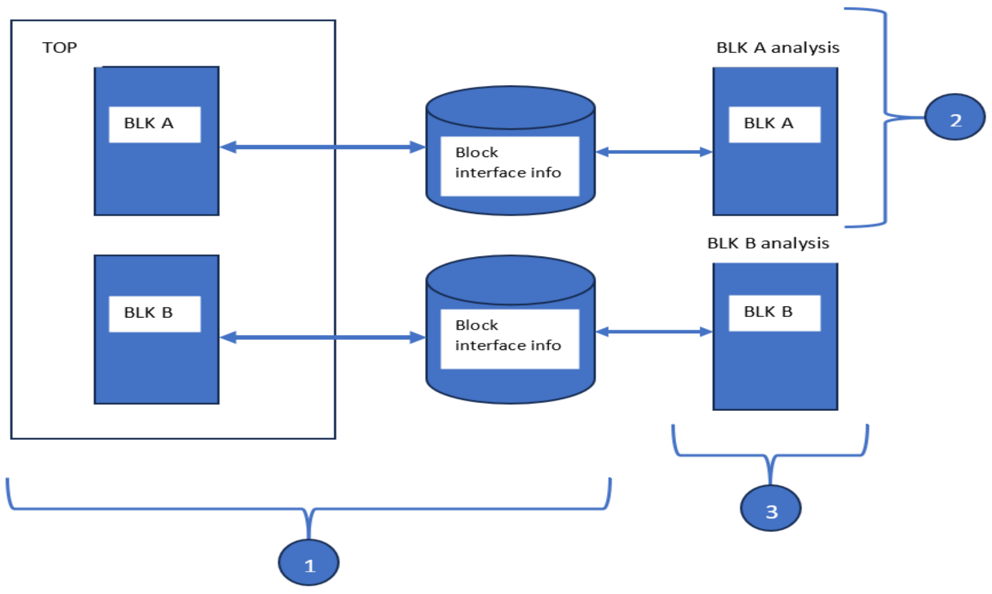


Figure 35 Hyperscale run analysis for Top and below blocks

For label "1"

set hier_enable_analysis true

set_hier_config -block BLKA -path $model_A

set_hier_config -block BLKB -path $model_B

read_verilog TOP.v

link_design TOP

update_timing

write_hier_data $topConextDir

For label "2" and "3"

set hier_enable_analysis true

*read_verilog BLK**A/B**.v*

*link_design BLK**A/B***

read_context $topContextDir

update_timing

*write_hier_data $model_**A/B***

For Multiple instances merging, multiple instances of a particular block can be put into a group and one single model of the block a particular group can be created.

Scripts for the same is as below, here instances a1, a2 in one group g1 and instances a3, a4 in group g2:

set_app_var hier_enable_analysis true

set_hier_config -block A -path $blkModel

set_hier_config -block A -path $blkModel_g1 -instances {a1 a2} -name g1

set_hier_config -block A -path $blkModel_g2 -instances {a3 a4} -name g2

3.4.1 HYPERSCALE FLOW: TOP-DOWN

Top-down method consists of four basic steps. These steps are to be run in separate PT session or Prime Time session which is the chosen tool for this analysis.

Step1: As a separate PT session, extraction of block constraints at full-chip level where minimal update is present and no parasitic are extracted. The commands to same are:

```
source full_chip_analysis_script.tcl         # Read, link, update
characterize_context -block blkA             # Generate context
update_timing
write_context -format ptsh -output $cnsDir   # Write constraints
```

Step2: Using the step1 extracted constraint's initial block level timing is procured. The commands for same are:

```
set_app_var hier_enable_analysis true
source $cnsDir/CONSTRAINT/blkA/hier_default/variables.pt
read_verilog BlkA.v

link_design
read_parasitics BlkA.spef
source $cnsDir/CONSTRAINT/blkA/hier_default/constraints.pt
update_timing -full          # Also run update_noise if applicable

write_hier_data $blkDir1     # Write initial block-level timing data
```

Step3: For generating accurate context data of the block, top-level hierarchical hyperscale timing analysis using block's timing results. The steps for the same are:

```
set_app_var hier_enable_analysis true
set_hier_config -block blkA -path $blkDir1   # Set block configuration
source full_chip_analysis_script.tcl  # Read, link, parasitics, update
write_hier_data $topDir               # Write block context
```

Step4: Now block level timing analysis is done using accurate block context. The steps for the same are:

```
set_app_var hier_enable_analysis true
source $cnsDir/CONSTRAINT/blkA/hier_default/variables.pt
read_verilog blkA.v
link_design
read_parasitics blkA.spef
source $cnsDir/CONSTRAINT/blkA/hier_default/constraints.pt
read_context $topDir          # Read block context
update_timing -full           # Also run update_noise if applicable
write_hier_data $blkDir2      # Write block-level timing data
```

Steps mentioned in **Step3** and **Step4** can be repeated if needed for timing closure convergence.

For reporting a timing path this model structure can be done same as how done as in flat timing analysis from the SoC top.

In the example design when tried to report timing from port **data to BLKA/ff11** then the report from the top level looked like below

pt_shell> ***report_timing -from data [1] -to ff11***

```
 Startpoint: data[1] (input port clocked by clock)
 Endpoint: ff11 (rising edge-triggered flip-flop clocked by clock)
 Path Group: clock
 Path Type: max

 Point                                         Incr       Path
 -------------------------------------------------------------

clock clock (rise edge)                 0.00       0.00
clock network delay (propagated)        0.00       0.00
input external delay                    0.34 @     0.34 r
data[1] (in)                            0.00 @     0.34 r
and1/A (AND2)                           0.00 @     0.34 r
and1/Z (AND2)                           0.06 &     0.41 r
ff11/D (FD3)                            0.00 &     0.41 r
data arrival time                                  0.41

clock clock (rise edge)                 5.00       5.00
clock source latency                    0.25 @     5.25
clock (in)                              0.00 @     5.25 r
cbuf1/A (BUF)                           0.00 @     5.25 r
cbuf1/Z (BUF)                           0.13 &     5.38 r
ff11/CP (FD3)                           0.02 &     5.40 r
clock reconvergence pessimism           0.00       5.40
clock uncertainty                      -0.05       5.35
library setup time                     -0.15       5.20
data required time                                 5.20
-----------------------------------------------------------------
data required time                                 5.20
data arrival time                                 -0.41
-----------------------------------------------------------------
slack (MET)                                        4.79
```

Another path from our design is between the flip flop named "L" and the primary output "Po"

```
pt_shell> report_timing -path_type full_clock -delay_type max \
 -input_pins -max_paths 1 -sort_by slack
  ...
  Startpoint: ffL (rising edge-triggered flip-flop clocked by clk)
  Endpoint: Po (output port clocked by clk)
  Path Group: clk
  Path Type: max

  Point                                     Incr       Path
  ---------------------------------------------------------------
  clock clk (rise edge)                     0.00       0.00
  clock source latency                     10.57 @    10.57
  clk (in)                                  0.00 @    10.57 r
  u1/A (B1I)                                0.00 @    10.57 r
  u1/Z (B1I)                                0.57      11.15 r
  u2/A (B1I)                                0.00      11.15 r
  u2/Z (B1I)                                0.58      11.73 r
  ffL/CP (FD2)                              0.00      11.73 r
  ffL/Q (FD2) <-                            1.77      13.50 r
  u4/A (B1I)                                0.00      13.50 r
  u4/Z (B1I)                                0.56 @    14.06 r
  Po (out)                                  0.00 @    14.06 r
  data arrival time                                   14.06

  clock clk (rise edge)                    10.00      10.00
  clock network delay (propagated)          0.00      10.00


  clock reconvergence pessimism             4.28      14.28
  MIM pessimism                             0.28      14.56
  output external delay                     5.16 @    19.72
  data required time                                  19.72
  ---------------------------------------------------------------
  data required time                                  19.72
  data arrival time                                  -14.06
  ---------------------------------------------------------------
  slack (MET)                                          5.67
```

True block context is available the top-level analysis in hyperscale is done. Now, during the timing updated it is checked if a block's context from previous top-level runs is available. If yes then that block context is overlaid on top of designer's constraints that are budgeted. Apart from this the missing or unwanted constraints such as case analysis etc are applied or removed. The values are corrected and it explains that in each run the boundary violations are actually fixed and updated.

The boundary constraints that are automatically updated and adjusted in each run are:

- Input & Output delays
- Clock latency (static & dynamic)
- Input transition values
- Output loading values
- Values of set case analysis

The adjusted values are not physically modified to block constraint but remains in memory and applied during the stage of *update_timing*.

3.4.2 CONTEXT BUDGETING

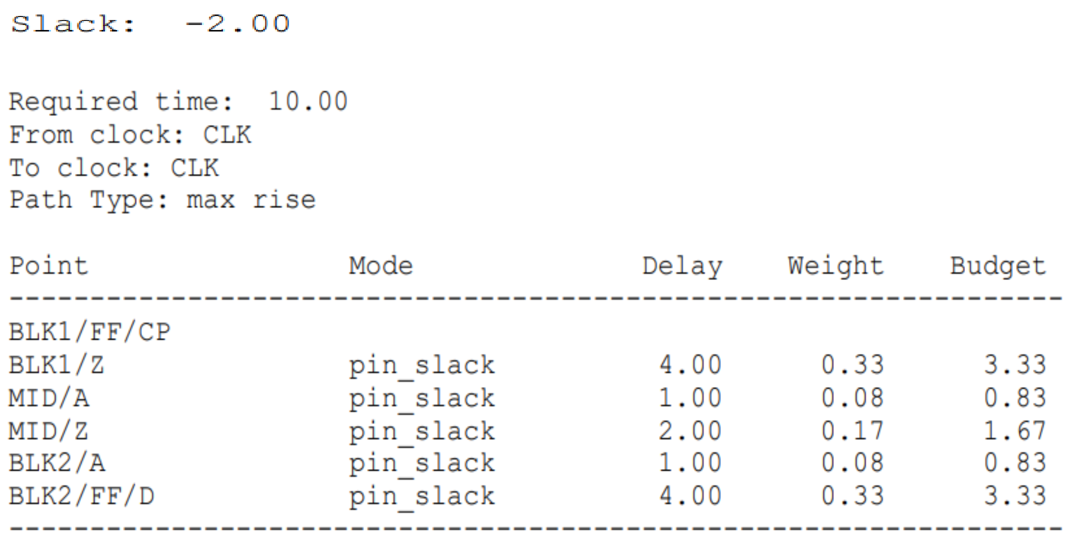
As earlier explained in *2.2.4.1 HYPERSCALE MODELING* and shown in *Figure 21 Figure 22* in the current design context budgeting is also performed. The script used for the purpose is as below:

```
read_verilog top.vg
link_design top top

create_clock  -period 10 -waveform {0 5} CLK
set_propagated_clock CLK
set_input_delay -clock CLK 2 [get_ports in]
set_output_delay -clock CLK 2 [get_ports out]

# specify which blocks to characterize/budget
characterize_context -block block1 -instances BLK1
characterize_context -block block2 -instances BLK2

# update the design timing
update_timing

# perform budgeting and report some results
set_timing_budget -mode pin_slack
update_budget
report_budget -through BLK1/Z
report_budget -through BLK2/A

write_context -nosplit -format ptsh \
  -output BLKA_context [get_cells BLKA]
```

Command *update_budget* adjust the block context and this must be run after *update_timing* for the design is run.

Post the *update_budget* command execution, use *report_budget* to report the budgeted paths. Example, one of the blocks register MID gives the below reported path along the hierarchical pins delineating the segments.

pt_shell> report_budget -through MID/Z



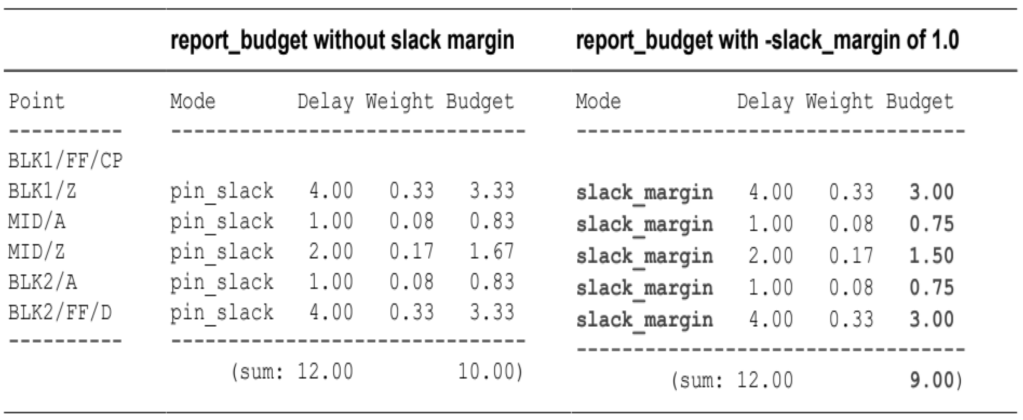
```
Slack:   -2.00

Required time:  10.00
From clock: CLK
To clock: CLK
Path Type: max rise

Point                 Mode              Delay    Weight    Budget
-----------------------------------------------------------------
BLK1/FF/CP
BLK1/Z                pin_slack          4.00      0.33      3.33
MID/A                 pin_slack          1.00      0.08      0.83
MID/Z                 pin_slack          2.00      0.17      1.67
BLK2/A                pin_slack          1.00      0.08      0.83
BLK2/FF/D             pin_slack          4.00      0.33      3.33
-----------------------------------------------------------------
```

According to the reported budget, the slack margin can be allocated proportionally for the entire path, not to segment paths. Command to do so is

pt_shell>   set_timing_budget -positive_slack_margin 0.50 -slack_margin 1.00

Here, *slack_margin* affects the violating path where it subtracts the margin value from the negative slack. The *positive_slack_margin* affects by subtracting the specified value from original value of positive slack.

Below is the comparison of budget without slack margin and with slack margin.



	report_budget without slack margin				**report_budget with -slack_margin of 1.0**			
Point	Mode	Delay	Weight	Budget	Mode	Delay	Weight	Budget
BLK1/FF/CP								
BLK1/Z	pin_slack	4.00	0.33	3.33	**slack_margin**	4.00	0.33	**3.00**
MID/A	pin_slack	1.00	0.08	0.83	**slack_margin**	1.00	0.08	**0.75**
MID/Z	pin_slack	2.00	0.17	1.67	**slack_margin**	2.00	0.17	**1.50**
BLK2/A	pin_slack	1.00	0.08	0.83	**slack_margin**	1.00	0.08	**0.75**
BLK2/FF/D	pin_slack	4.00	0.33	3.33	**slack_margin**	4.00	0.33	**3.00**
		(sum: 12.00		10.00)		(sum: 12.00		**9.00**)

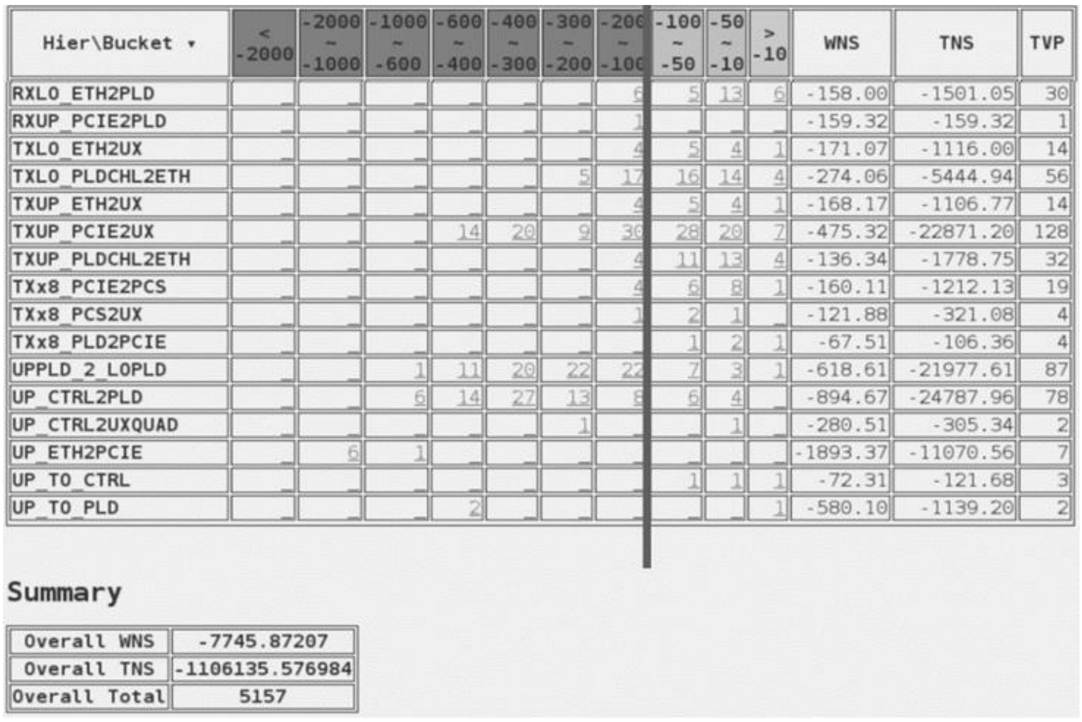
Figure 36 Report showing with and without budgeting context

CHAPTER 4. RESULT AND EVALUATION

Proposed method of STA for a complex SoC, Hyperscale with top-down approach with constraint promotion approach have been tested and verified for the example design. As a front-end design engineer the main motive is to align the constraints when moving from block to immediate top level and finally at the Soc level. The tool used to meet the requirement are Synopsys PT or primetime tool and for constraint promotion both using the PT and fishtail which is again from Synopsys. Hyperscale timing model and the original circuits sizes were compared to provide the efficiency result of the proposed method.

The traditional approach of design and verifying independently which used to take more time and memory. With the proposed method these two category pillars were reduced.

The below *Figure 377* shows the setup constraint check for FUNC mode using the hyperscale hierarchical method.



Hier\Bucket ▾	< -2000	-2000 ~ -1000	-1000 ~ -600	-600 ~ -400	-400 ~ -300	-300 ~ -200	-200 ~ -100	-100 ~ -50	-50 ~ -10	> -10	WNS	TNS	TVP
RXLO_ETH2PLD							6	5	13	6	-158.00	-1501.05	30
RXUP_PCIE2PLD							1				-159.32	-159.32	1
TXLO_ETH2UX							4	5	4	1	-171.07	-1116.00	14
TXLO_PLDCHL2ETH						5	17	16	14	4	-274.06	-5444.94	56
TXUP_ETH2UX							4	5	4	1	-168.17	-1106.77	14
TXUP_PCIE2UX				14	20	9	30	28	20	7	-475.32	-22871.20	128
TXUP_PLDCHL2ETH							4	11	13	4	-136.34	-1778.75	32
TXx8_PCIE2PCS							4	6	8	1	-160.11	-1212.13	19
TXx8_PCS2UX							1	2	1		-121.88	-321.08	4
TXx8_PLD2PCIE								1	2	1	-67.51	-106.36	4
UPPLD_2_LOPLD			1	11	20	22	22	7	3	1	-618.61	-21977.61	87
UP_CTRL2PLD			6	14	27	13	8	6	4		-894.67	-24787.96	78
UP_CTRL2UXQUAD						1			1		-280.51	-305.34	2
UP_ETH2PCIE		6	1								-1893.37	-11070.56	7
UP_TO_CTRL								1	1	1	-72.31	-121.68	3
UP_TO_PLD				2						1	-580.10	-1139.20	2

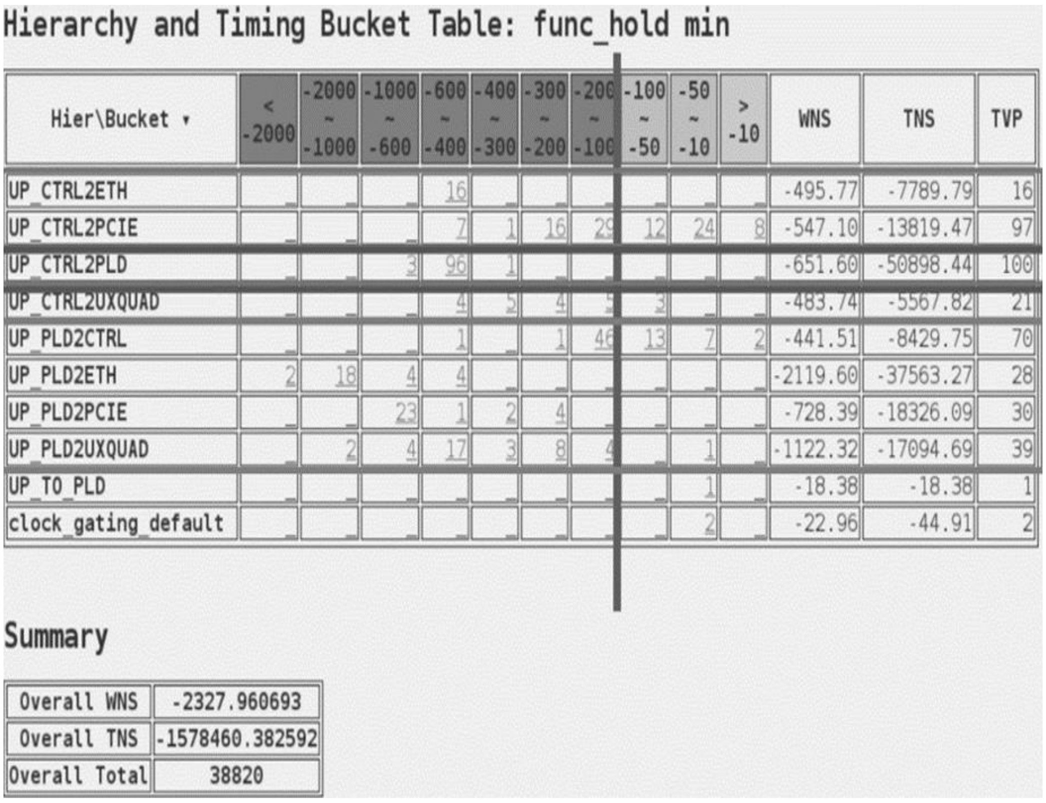
Summary

Overall WNS	-7745.87207
Overall TNS	-1106135.576984
Overall Total	5157

Figure 37 Summary of Setup check for Func mode

This value does have some WNS value which enables the Physical design engineer to adjust appropriate buffer manually on the netlist to meet the final timing for the SoC.

On similar lines for the *Figure 388* shows the hold constraint check for FUNC mode on different arc using the proposed method.


Hierarchy and Timing Bucket Table: func_hold min

Hier\Bucket ▾	< -2000	-2000 ~ -1000	-1000 ~ -600	-600 ~ -400	-400 ~ -300	-300 ~ -200	-200 ~ -100	-100 ~ -50	-50 ~ -10	> -10	WNS	TNS	TVP
UP_CTRL2ETH	_	_	_	16	_	_	_	_	_	_	-495.77	-7789.79	16
UP_CTRL2PCIE	_	_	_	7	1	16	29	12	24	8	-547.10	-13819.47	97
UP_CTRL2PLD	_	_	3	96	1	_	_	_	_	_	-651.60	-50898.44	100
UP_CTRL2UXQUAD	_	_	_	4	5	4	5	3	_	_	-483.74	-5567.82	21
UP_PLD2CTRL	_	_	_	1	_	1	46	13	7	2	-441.51	-8429.75	70
UP_PLD2ETH	2	18	4	4	_	_	_	_	_	_	-2119.60	-37563.27	28
UP_PLD2PCIE	_	_	23	1	2	4	_	_	_	_	-728.39	-18326.09	30
UP_PLD2UXQUAD	_	2	4	17	3	8	4	_	1	_	-1122.32	-17094.69	39
UP_TO_PLD	_	_	_	_	_	_	_	_	1	_	-18.38	-18.38	1
clock_gating_default	_	_	_	_	_	_	_	_	2	_	-22.96	-44.91	2

Summary

Overall WNS	-2327.960693
Overall TNS	-1578460.382592
Overall Total	38820

Figure 38 Summary of Hold check for Func mode

From the above two figures the leftmost side shows of figure shows all respective components through which clock propagates. All these components undergo STA checks. All timing violations among these components are reported. Using data worst negative slack is calculated after which designers have to fix timing issues between these paths.

Flat analysis was needed to be run for timing signoff where the runtime comparison is done. The below table shows the results. This is based on no of cells = 1700 and instances of 50M gate count.

Table 1 Flat vs Hyperscale runtime comparison

STA method	Accuracy	Memory required	Runtime (hrs)
FLAT	100%	Greater than 512G	~98
Hyperscale with context	98%	Less than 256G	~22

We do see that using hyperscale has significantly reduced the runtime but to achieve this we do needed few cycles of run to refine the context data to generate a perfect model for timing analysis. Also, the runtime memory requirement is also reduced than the conventional Flat analysis. Using hyperscale with context can be a significant methodology for timing sign-off in place of Flat analysis to save cost, runtime and resources.

CHAPTER 5. CONCLUSION AND FUTURE WORK

Semiconductor devices are continuously reducing in size scaling. Apart from these scaling, there are process variations also which also causes complexity in analysis for a design. Also, the designs are pushed to different corners where usage is getting explored more. Reducing the runtime and accuracy increase in evaluation, static timing analysis is introduced so as to validate that the design can work in the high-speed frequency. More and more blocks are coming as third-party vendor IP block in modern SoC. These aggregations demand a new approach and renovation in the hyperscale context flow for timing closure. The timing models contain interface constraints and delay information. When process variations come into picture all the gate delays inside a module or model become random variables. The proposed method does not consider process variation. Another challenge is correlation among modules during the model extraction and generation, which is usually unknown. This information should be incorporated during timing verification process of the complete design.

Focus of the current thesis is to align the flat and hierarchical timing approach but again the two challenges can be failing point of the proposal. Therefore, the future can be to work on how to accommodate these in the proposed method.

There are still some timing arcs which one has to manually do like in case of context budgeting, complicated clocking structure while moving from top to a block level. For example, here only one type of protocol controller is present but in modern days Hyperscale FPGAs where SoC and programable logic also reside and can have both PCIE and Ethernet protocol working with the Serdes. In this case for each lane there can be a situation where either of the two-protocol working but the word clock can be coming from either controller. To model these requires a very practical approach with manual intervention for model generation. To automate these can be futuristic work for the hyperscale method.

Although runtime and memory requirement has reduced much with the use of hyperscale and context model generation but still the runtime is days to weeks for a more than a million-gate count SoCs. Further reduction and correct modelling of bottom blocks is something further require future work.

There are some context budgeting commands for slack which can only be done for input-to-output on block feedthrough segments. Commands like *update_budget* can be only be run once.

The proposed method does establish a complete and a very effective hierarchical timing analysis flow. It maintains efficiency as well as accuracy which enables modern SoC to close timing analysis with quality and on time but its complete uses for too much complex SoC design is questionable as there are some multi-mode multi corner cases which this proposed model can fail and needs more research before it can replace flat timing analysis fully for modern SoC and so as to reduce cost and time.

REFERENCES

[1] Static Timin Analysis for nanometre design, by J Bhasker and Rakesh Chadha.

[2] Hierarchical Timing Analysis: Pros, Cons, and a New Approach, Cadence white paper.

[3] Hierarchical Methodology approach to SoC Design – A comprehensive look, Vivek Bharadwaj, SSRN.

[4] Synopsys, "PrimeTime: Synopsys Static Timing Solution", Technical White Paper, July 1999

[5] S.V. Venkatesh et al., "Timing Abstraction of Intellectual Property Blocks", CICC'97, May 1997

[6] H. Yalcin et al., "An Advanced Timing Characterization Method Using Mode Dependency", DAC'01, June 2001

[7] Robert B. Hitchcock, Gordon L. Smith, and David D. Cheng. Timing analysis of computer hardware. IBM Journal Research Development, 26(1):100–105, 1982.

[8] Automated Timing Model Generation by Ajay J. Daga, Loa Mize, Subramanyam Sripada, Chris Wolff, Qiuyang Wu

[9] Synopsys, "Library Compiler Reference Manual", Version 2001.08, August 2001

[10] Synopsys, "PrimeTime Modelling User Guide", Version 2001.08, August 2001

Checklist of Items for the Final Dissertation / Project / Project Work Report

1.	**Is the final report neatly formatted with all the elements required for a technical Report?**	**Yes**
2.	Is the Cover page in proper format as given in Annexure A?	**Yes**
3.	Is the Title page (Inner cover page) in proper format?	**Yes**
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	**Yes** **Yes**
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	**Yes** **Yes**
6.	Is the title of your report appropriate?	**Yes**
7.	Have you included the List of abbreviations / Acronyms?	**Yes**
8.	Does the Report contain a summary of the literature survey?	Yes
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	**Yes** **Yes** **Yes** **Yes** **Yes** **Yes**
10.	Is the conclusion of the Report based on discussion of the work?	**Yes**
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Are all the references cited in the body of the report	**Yes** **Yes** **Yes**
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a PowerPoint Presentation, or a user manual. Source code of software need not be included in the report.	**Yes**

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

[illegible]

Place: Bangalore **Signature of the Student**

Date: 10/11/2024 **Name: Mrityunjay Nath**

ID No.: 2022HT80559