Study Of Futuristic Approaches To Get Better QoR In Physical Implementation

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics & Communication Engineering

(VLSI Design)

By

Priyangi Singhvi

(15MECV17)

Electronics & Communication Engineering Department
Institute of Technology
NIRMA University
Ahmedabad-382 481
May 2017

Study Of Futuristic Approaches To Get Better QoR In Physical Implementation

Major Project Report

Submitted in partial fulfillment of the requirements for the degree of

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

By

Priyangi Singhvi
(15MECV17)

Under the guidance of

External Project Guide:	**Internal Project Guide:**
Mr. Tanuj Jindal	**Dr. N. M. Devashrayee**
Digital Design Engineer	Institute of Technology
Intel Technologies	Nirma University
Bangalore	Ahmedabad

Electronics & Communication Engineering Department
Institute of Technology
NIRMA University
Ahmedabad-382 481
May 2017

Certificate

This is to certify that the Major Project entitled **"Study Of Futuristic Approaches To Get Better QoR In Physical Implementation "** submitted by **Priyangi Singhvi (15MECV17)**, 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 her 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.

Dr. N. M. Devashrayee
Internal Guide

Dr. N. M. Devashrayee
PG Coordinator (VLSI Design)

Dr. D. K. Kothari
Head, EC Dept.

Dr. Alka Mahajan
Director, IT-NU

Date:

Place: Ahmedabad

Certificate

This is to certify that the Project entitled "**Study Of Futuristic Approaches To Get Better QoR In Physical Implementation**" submitted by **Priyangi Singhvi (15MECV17)**, towards the submission of the Project for requirements for the degree of Master of Technology in VLSI Design, NIRMA University, Ahmedabad is the record of work carried out by her under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.

Mr. Tanuj Jindal
Digital Design Engineer
Intel Technologies
Bangalore

Declaration

This is to certify that

a. 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.

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

-Priyangi Singhvi

Acknowledgement

The satisfaction that accompanies the successful completion of this task would be incomplete without the mentioning a few names, without whose constant guidance and encouragement would have made efforts go in vain. So with this gratitude, I acknowledge all those guidance and encouragement owned our efforts with success.

With profound sense of gratitude, I would like to express my heartfelt thanks to **Mr. Tanuj Jindal**, Digital Design Engineer, Intel Technologies, for providing constant encouragement, immense support and guidance whenever needed.

I thank **Dr. Alka Mahajan** (Director and Head of Department NIRMA University, Ahmedabad) for her support and suggestions.

I am also thankful to **Dr. N. M. Devashrayee** (Professor and Program Coordinator, M. Tech - EC (VLSI Design)) for providing me the best facilities and atmosphere for the thesis completion and presentation.

I thank all the faculties for their valuable support. Also I thank all the friends and well-wishers for their healthy criticism and valuable co-operation in all respect.

- Priyangi Singhvi
15MECV17

Abstract

The quality of the netlist generated during synthesis has an enormous impact on the rest of the physical design flow. For SoC's designed at advanced nodes, it is very essential to come out of synthesis with predictable timing and congestion estimates. The quality of the netlist coming out of synthesis has a big impact on the speed and predictability of the backend physical implementation. Optimizing at higher abstraction levels offer more room for QoR improvement, in addition to enabling faster runtimes. High quality physical RTL output makes the entire design flow more predictable with shorter design cycles. Along with this achieving timing closure in their first iteration is also a challenge. The design team must iterate by studying the output of the physical synthesis run, then potentially massage the input, e.g., by changing the floorplan, timing assertions, pin locations, logic structures, etc., in order to hopefully achieve a better solution for the next iteration.

In this project we are studying the approaches to get better QoR in physical implementation. The first approach is to facilitate family integration to get floorplan definition early in the design cycle, in order to avoid major critical issues in the later stage of the design cycle. The second approach is using Synopsys Physical Guidance flow. This flow is jointly supported by DC and ICC to improve runtime, correlation and routability and to reduce routing related congestion.

Contents

List of Figures

Chapter 1

Introduction

Very Large Scale Integration (VLSI) is a technology domain in which significant changes are happening every year. According to Moore's law, the density of transistors on a chip doubles every eighteen months. Even though this law appears to be unrealistic beyond a point, utilizing the power of high quality designs, modern fabrication technologies and availability of Design Automation (DA) tools have facilitated the industry to keep up with Moore's law. As the VLSI process technology node shrinks, the fabrication technology and the VLSI design process becomes complex. The design complexity is arising from the sheer fact that more functionality is included in the same chip area where as parasitic elements (resistance and capacitance) start dominating in design. Along with complexity of modern design,comes the issue of quality and efficiency.
To combat these challenges, industry resorts to automation in various design phases of a VLSI product. The complexity of modern day designs in VLSI is such that any chip/design without using design automation tools/methodologies is unthinkable. This project aims at improving the quality and efficiency in the backend design phase.

1.1 Problem Statement

In the initial stage of synthesis in ASIC design flow, due to lack of layer assignment and buffering needs, the delay estimation of Family is poor. Family Integration happens in later stage of design cycle, which is late to figure out major critical issues for timing violations.The major issues (Figure 1.1) are:

- Figuring out the buffer distance.
- Figuring out channel width.

The second challenge is to improve runtime, correlation and routablity in physical

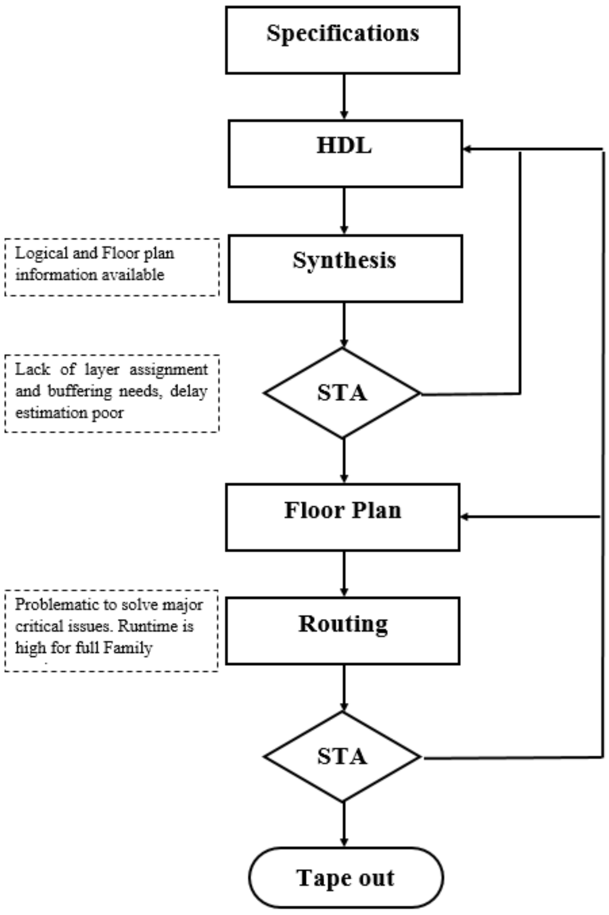

Specifications
HDL
Logical and Floor plan information available
Synthesis
Lack of layer assignment and buffering needs, delay estimation poor
STA
Floor Plan
Problematic to solve major critical issues. Runtime is high for full Family
Routing
STA
Tape out

Figure 1.1: Problem Flowchart

design, and to achieve better quality of results. For this we use Synopsys Physical Guidance Flow (SPG). This flow significantly speeds up ICC runtime because ICC uses the physical guidance information from DC as its seed placement and therefore goes through fewer placement steps.

1.2 Motivation and Overview

In the initial stage of design cycle major focus is on RTL verification. During this period the backend team focuses on estimating the pin locations, block placement,etc. Routing cannot be done at this stage as we do not have concrete data required for it. All the runs are running parallel to each other.

Now as the design advances in the later stage of design cycle, we can see that the timing models are present at the block level design, but the .lib file is not accurate. So even though the timing model is present , the tool is not able to figure out the accurate timing violations or the exact layer on which the signal should be routed. Now here we do synthesis on Design Compiler in Topographical mode, which has the capability to create a virtual layout and give post layout timing. The tool with the .lib present for the block level is able to do so for the blocks, but this is not the case at Family level.

At the Family level, the blocks are considered as black boxes. So now we do not have any information for the timing of the family. All we have is the floorplan information for it. Here the tool cannot assign layers to the signals as it does not have the accurate .lib required for it. For this we depend on the feedback from the timing team. So our intention here is to check the design for major critical issues related to timing, DRC and area, earlier in the design cycle. And try to resolve the issue during Synthesis of the design, and revert back to the RTL team if the issue cannot be resolved without changing the RTL. This will save us significant amount of runtime.

The objective of this report is to do distance based buffering, which will influence the floorplan and also it will give an estimate for the number of nets which will traverse over the block, so that the channel width can be determined. This will help us to get an estimate for the design area and avoid problems of congestion as well.

Moreover, our aim is to achieve better QoR in physical implementation, so we also try to reduce the ICC runtime by reducing the number of placement steps which the ICC tool has to undergo. This we are trying to achieve by passing the placement information during the synthesis stage. Our aim through this is to get the placement of standard cells from the synthesis stage and pass it to the place and route tool and avoiding the coarse placement step in ICC. And using the netlist generated from synthesis which contains the placement information as a seed placement for ICC, we improve the coorelation between both the stages, which further enhances the Quality of Results.

Chapter 2
Literature Review

A physical design flow consists of producing a production worthy layout from a gate-level netlist subject to a set of constraints. This report focuses on the problems imposed by shrinking technologies and it's turnaround time.

As stated by Moore, the number of transistors per square inch will double approximately every 18 months.Due to this design complexity is increased which in turn affects the turnaround time. A flow consisting of logic synthesis followed by place-and-route cannot work with deep submicron (DSM). At 0.18um and beyond, interconnect becomes a dominant factor and breaks the dichotomy between the logical and physical domains.We can no longer neglect the impact of the interconnect on timing: the gate delay depends mostly on the output capacitance it drives, of which the net capacitance becomes the largest contributor; also the delay of long nets, which depends on their capacitance and resistance, becomes larger than gate delays. Moreover, coupling capacitance (the capacitance between nets on the same layer) becomes more dominant over inter-layer capacitance (the capacitance due to overlapping of interconnect between different layers) with every new process technology. This is because the nets are getting much closer to each other and the wire aspect ratio of height to width is increasing. This means that the capacitance of a net cannot be determined without knowing both its route and that of its neighbors.

The resistance of interconnect plays a vital role in the layer assignment for signals. As the design geometries are shrinking,the area is decreasing.As a result of which the wire resistance is increasing drastically as resistance is inversely proportional to area. To accommodate the effect of high resistance, the critical signals are routed in higher layers, where the resistance as compared to the lower layers is less.

Signal Integrity plays a key role in VLSI Design. Various signals have to traverse through large distance to reach the desired flop. During this traversal, there is a high risk of the signal to get distorted due to the crosstalk, IR drop, electromigration and self-heat. This problem must be identified as early as possible since it is very costly and time consuming as it is very difficult to fix them during the final stages of implementation.

Gradually the design architecture is moving towards higher complexity level. Production designs today contain tens of millions of gates. Due to the overall complexity of the chip design, it is often divided in several logic and physical blocks, with several independent design teams working in parallel at a different pace. The number of large chips being designed flat is very less. The trend is towards more hierarchical designs. Hierarchical flow pose various problems like, how to handle timing constraints between the chip level and the block level, how to verify the feasibility of the constraints at the chip level,etc. Floor-planning is still a difficult problem, especially when it must be done considering timing and congestion.

The current flow iterates between gate-level synthesis and place-and route. After place-and-route, if the constraints are not met, the netlist is back-annotated with the actual wire load and re-synthesized. Signal integrity problems are usually handled at the detailed routing level. Place and route data are necessary to determine the timing. Trying to compensate for the lack of these data by driving synthesis with pessimistic assumptions results in over-design. Extracting net capacitance after place and route and feeding them back to synthesis in an attempt to seed synthesis with more realistic capacitance estimations is not practical. This often results in a different netlist, causing the placement and routing to change, which results in different wire loads,and hence the timing is affected. There is no guarantee that this iterative process will converge.

Most of the critical nets need to be buffered to satisfy one or more of these reasons: restore signal levels, increase the drive strength, shield a critical path from a high-load off-critical path. We can buffer the critical signals as per their layer requirement and get the delay value. This delay value can be further used to estimate the timing and avoid any major critical issues during the routing stage, as at this stage the actual timing can be estimated. So, if we are able to get information related to buffering needs of critical nets, along with their layer requirement during the synthesis stage, then we can estimate the timing to avoid the timing violations in the later stage of design cycle and eventually save a significant amount of time. Along with this if we get a better optimized netlist containing all the physical information from the synthesis stage, which will act as a seed placement to the place and route tool, will reduce the runtime and also boost the quality of results in physical implementation.

Chapter 3

Physical Design

Similar to any product development, VLSI project development also starts with developing a system level algorithm based on specifications. This algorithm is utilized for the development of Register Transfer Level (RTL) code. All the steps from system level algorithm development to generation of RTL code is considered as front end of VLSI circuit design. In a VLSI product design, usually the bottleneck is back end circuit design due to low efficiency. The various design steps from synthesis to tape out (sign off) of the design is considered as back end design. A representative physical design flow is shown in Figure 3.1. Each stage of back end design is discussed in the following sections.

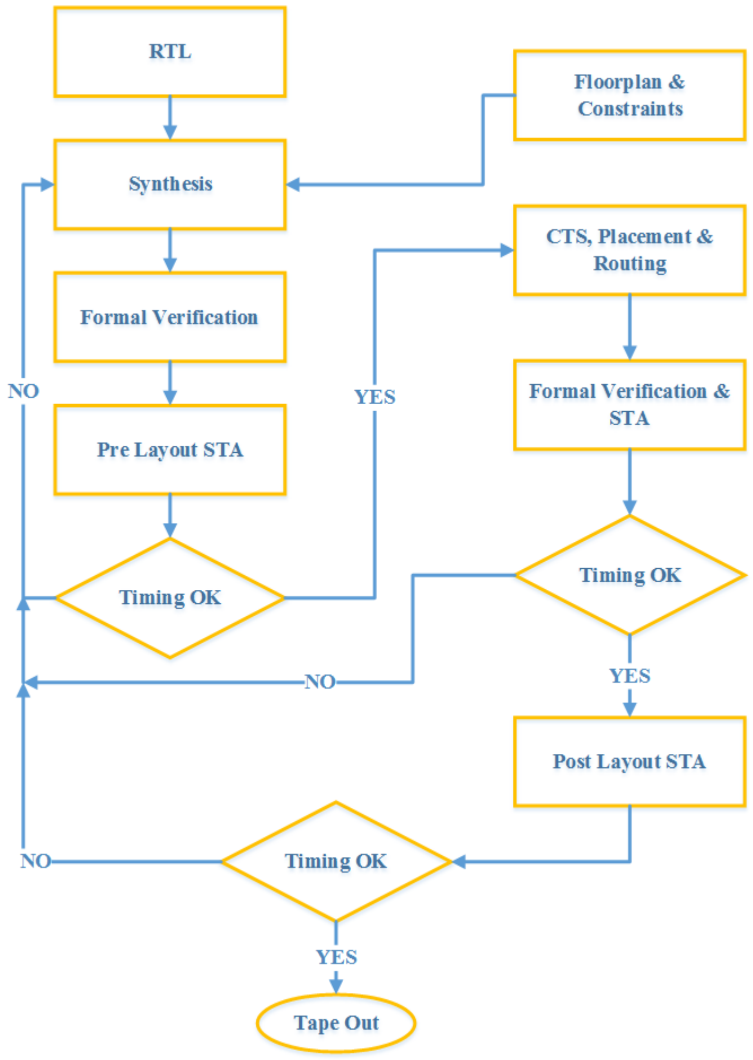


Figure 3.1: Physical Design Flow

3.1 Synthesis

Synthesis is the process of converting the RTL code to a netlist as shown in Figure 3.2. This is a two step process which involves :

- Logic Synthesis
- Technology mapping

In logic synthesis RTL code is converted to a generic netlist consisting of gates and sequential elements. The possible gates and sequential elements are obtained from

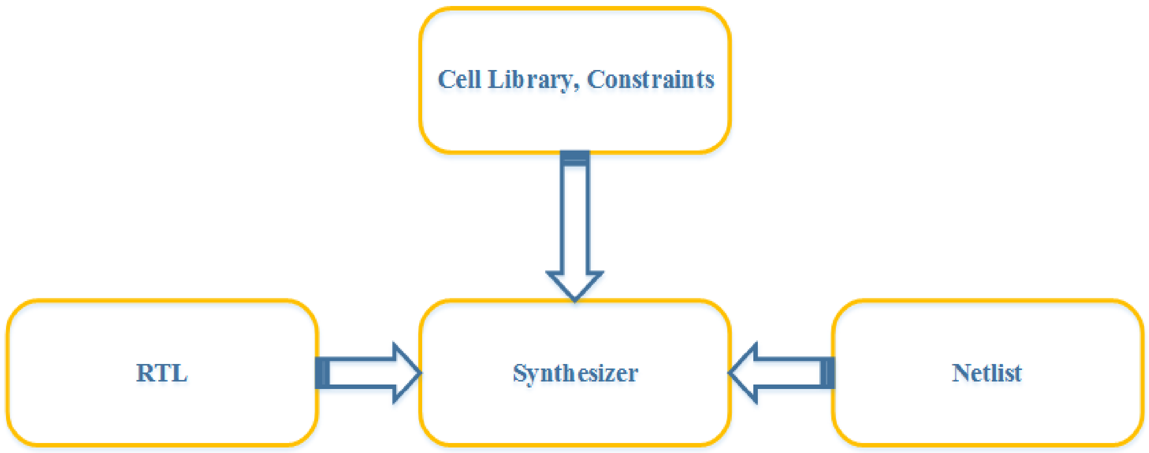


Figure 3.2: Stages in Synthesis

the target library specified for logic synthesis. The general target is to minimize the number of gates, number of levels and reduce signal activity. These three conditions reduce the total area, power and delay requirements. Apart from this the designer can also provide required constraints.

The second stage in synthesis is technology mapping. This is where the actual mapping of the generic netlist to the available cells in the library happens. Here various gates and sequential elements are clubbed together (based on the constraints specified as well as synthesis algorithm) and technology dependent netlist is generated. This technology mapping is also called as compiling in synthesis stage. For proper synthesis, it is necessary to provide the timing, power, operating conditions and area specifications.

Design for Testing (DFT) is critical for modern chips from debugging perspective. This is achieved in the synthesis stage by scan chain insertion. In an ideal scenario designers prefer to convert all the sequential cells in the design to be scannable. Since that is impossible a selected number of flip-flops are made scannable and this happens during the synthesis stage. Scan chains are inserted incrementally into the technology mapped netlist to generate the final netlist.

3.2 Formal Verification

The next step in the physical design flow is formal verification. In formal verification, the compiled netlist is compared to the RTL code to ensure that both are functionally equivalent. Not only after synthesis, but after various design changes formal verification is carried out to ensure that the logical functionality of the design hasn't deviated from the original RTL code. In short the purpose of this stage is to ensure that the design and RTL code are functionally equivalent.

In large designs it is very difficult to carry out the formal verification of the complete design. Thus for large designs, a divide and conquer approach is followed, which allows the designer to break down the design into smaller chunks and perform formal verification parallel on all the chunks. In this method, various critical nodes including the interface pins in the design are mapped to RTL code and the entire circuit is divided into large number of sub designs called cones. In the verification stage, the mathematical equivalence of the cones from the design are matched with RTL. For proper formal verification, it is necessary to have enough number of mapping points. This requirement comes from the fact that small mismatches can be missed in the mathematical model of a large block. By default all the outputs of sequential cells are mapping points. It is always advisable to add more mapping points as this will ensure that the mathematical equivalence of netist from design is matched with netlist from RTL at more number of nodes. This reduces the probability of missing mismatches. The advantage of dividing the design into FEV (Formal Equivalence Verification) cones is that this will allow the equivalence checking of a large number of cones in parallel which in turn will reduce the run time requirement as well as the memory and machine requirement.

3.3 Pre-Layout STA

Static Timing Analysis (STA) is a step which is performed at different phases of the physical design. pre-layout STA is a pure estimation based timing analysis performed on the netlist, since the placement and routing information is not available at this stage. At this stage, the parasitic information is estimated utilizing wire load models or the tool will perform a virtual (temporary) placement and routing to give the timing picture at this stage. The method chosen to obtain pre-layout STA depends on the tool. It is common practice to re-synthesize the design with a better set of constraints, if the pre layout STA failed to meet the timing requirements.

3.4 Clock Tree Synthesis, Placement and Routing

The synthesized netlist is given to Placement and Routing (PnR) tool along with the floor plan information. The PnR tool has complex algorithms to determine the placement of various elements, taking into consideration the timing as well as power aspect and area requirement which have been specified in the floor plan. Most of the times the placement provided by the tool might not be optimum, but this is always a good starting point.

Clock Tree Synthesis (CTS) is the next step in this group. The reason why clock tree synthesis is preferred over regular signal routing is to ensure that, the most critical signal in the design - clock, gets the best routing resources. This is a necessity to achieve an even distribution of clock signal to all the sequential elements and gated clock devices. The clock network is always preferred to generate the minimum latency and skew at the sequential elements or in another way the design prefers the clock signal to have the same arrival time at all the sequential elements. Even though this is an unachievable requirement, to achieve minimum variation there are various clock topologies followed by the industry. The two generic clock network types followed by industry are clock tree and clock mesh (grid).

Clock tree and clock mesh have their own set of advantages and disadvantages. Clock trees are preferred for small size ASICs. The main advantage is low routing requirement for tree networks, which in turn reduces the parasitic RC of clock network. This advantage of clock tree is counter balanced by the difficulty in balancing the path delays due to asymmetric placement of sequential elements as leafs in the design as well as tree networks' high sensitivity to variations. For tree networks, the tools usually depend on simulated annealing to achieve the optimum network. From the clock source, these buffers are driven by a global H tree. There are various clock tree networks available but H clock tree is one of the simplest and commonly used clock network structures for small sized designs. An H clock tree is shown in Figure 3.3

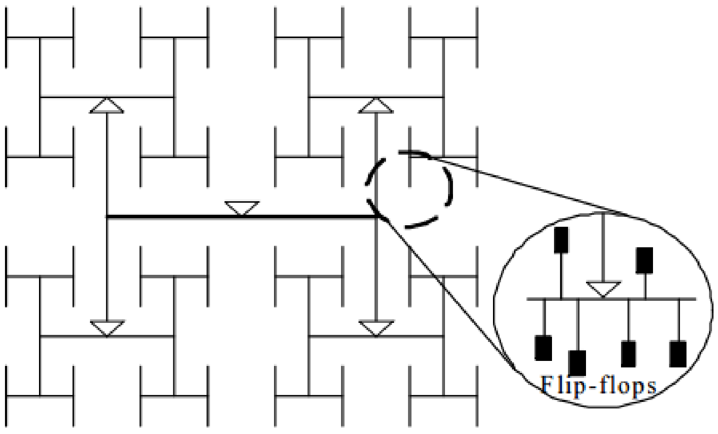


Figure 3.3: H Clock Tree

The reason for choosing mesh approach for large designs is that it is always easier to drive a finite set of buffers (global drivers) than a large number of sequential elements in the design maintaining minimum skew. The advantage of mesh topology is that the impact of skew is minimal as well as this topology offers excellent resistance to variations. But these advantages are enjoyed at the cost of larger wiring area, capacitance and power dissipation compared to tree topology. A mesh network and H tree global driver network of mesh are shown in Figure 3.4 and Figure 3.5 respectively. There are various other clock architectures like Hybrid tree with local links, Hybrid mesh with local tree etc. The choice of clock tree is always application specific and depends on the complexity of the ASIC. After CTS, regular signal routing is carried out. This is also algorithm based and the tool will try to optimize the routing based on the various design constraints provided by the designer. Similar to placement, routing rarely is optimal. After completion of PnR a formal verification is performed to ensure that the design is consistent with RTL code.

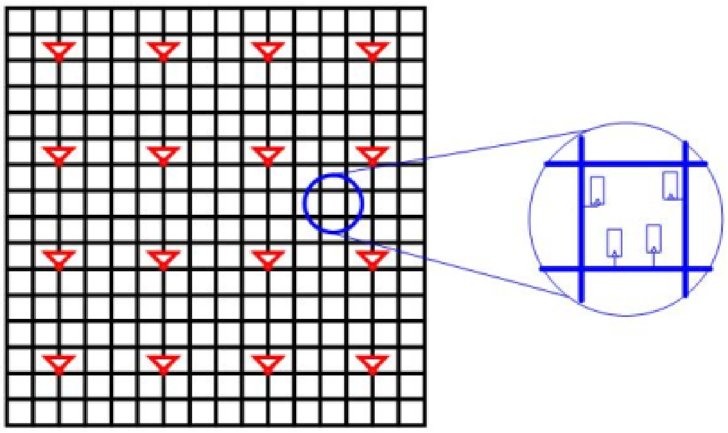


Figure 3.4: Clock Mesh

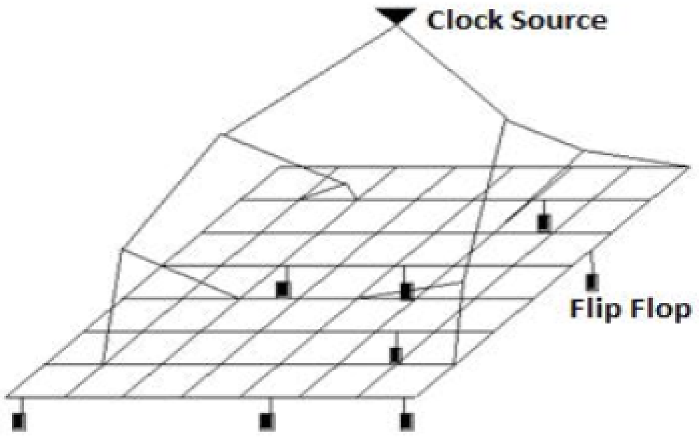


Figure 3.5: H tree global drivers of Clock Mesh

3.5 Static Timing Analysis

Static Timing Analysis (STA) is one of the critical steps in physical design. The inputs required for STA are final netlist, good quality extraction data and various timing constraints. Timing analysis ensures that the design meets the frequency requirements without failure. Basically STA is performed to identify setup (max) and hold (min) violations. Once all the violations are cleared, the design is said to be timing converged. The post layout STA will have the accurate placement and routing available with correct parasitic information available to the tool for calculating interconnect delay. This is the actual timing picture which has good correlation with silicon.

Chapter 4

A Comprehensive Study of Static Timing Analysis

Static timing analysis is an important step of Performance Verification (PV) for digital integrated circuits. STA provides an efficient method to determine whether a design can operate at the designated clock frequency without errors. The failure of a design or subset to operate at designated frequency is known as timing violations.

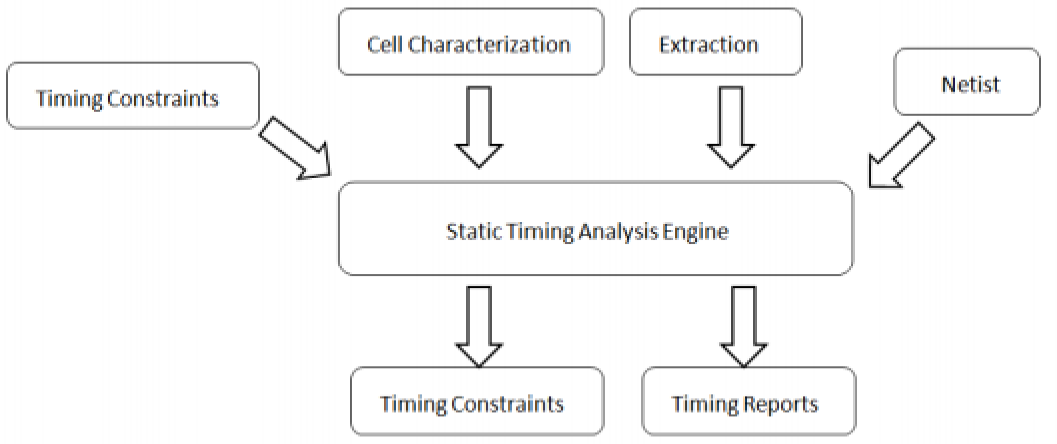


Figure 4.1: Static Timing Analysis

There are two types of timing analysis: static and dynamic. Static Timing Analysis is a non simulation based approach used to analyze the propagation of worst delays. This is in contrast to the other methodology - dynamic timing analysis; where the circuit is simulated with proper inputs. Compared to static analysis, dynamic timing analysis is time consuming and hence this approach is utilized for small portions of the circuit, especially when there is some ambiguity in the timing results. The main drawback associated with dynamic timing analysis is that the total coverage of the design depends on the selection of input test vectors. On the other hand STA is a worst case based approach where all the cells and interconnects are assigned the

worst case delay and slew as per the analysis. All other inputs along the path are assumed to be stable and such that it enables the timing path. This may lead to over-design, but will ensure that the circuit never fails from the timing perspective. STA can't handle scenarios like multi-input switching (MIS) where more than one input of a standard cell undergoes transition simultaneously. The various input and output collateral of STA are shown in Figure 4.1.
The basic timing analysis flow can be categorized into three steps:

- Reading all inputs and building the design
- Performing worst case analysis
- Generating reports

4.1 Input Collateral

The various input collaterals given to the STA tool for timing calculation are listed below.

- Design Netlist
  The tool reads the design from the netlist. The netlist provides the connectivity information of all the cells and devices in the design.
- RC Extraction
  The data generated by extraction engine is provided to the timing tool to calculate the interconnect delay as well as standard cell delays. Without proper parasitic data extraction, STA will be estimation based and inaccurate.
- Library Characterization
  All the standard cells available for the design have their characteristics recorded in a database. The purpose of standard cell characterization is to simplify the timing calculation process by providing lookup table like system for cell delay calculation.

- Timing Constraints

  These are the various user specified constraints provided to the tool for accurate timing analysis. There are default configurations available for timing analysis as part of tool, but sometimes the default ones may not be correct from design perspective. In such scenarios, the user overrides or specifications are given to the tool.

4.2 Output Collateral

The various outputs from STA are timing reports and timing constraints.

- Timing Reports

  STA of a design dumps out various reports. The main reports are the setup and hold reports which list out all the timing paths in the design and their respective slack.

- Timing Constraints

  The set of timing constraints dumped out as part of output are the timing specifications for interface pins.

4.3 Timing Concepts

Timing Path: Any path from a generating node to sampling node is a timing path. As shown in Figure 4.2, the generating and sampling node can be a sequential cell like flip flop or a latch, interface pin or a gated clock cell.
Delay: The time taken for a signal to appear at the output pin of a standard cell after applying all necessary inputs or the time required for a signal to be available at the receiver side of a net after applying the stimuli at driving side is called delay.
Slack: Slack or margin is the delta time by which the signal is beating the requirement. The calculation is different for setup and hold analysis. For setup analysis, the data is expected to be stable before the arrival of clock signal by a specified amount of time. For hold analysis, the data has to remain stable for a specified amount of

time after the application of clock signal. The Required time mentioned in eq. (4.1) and eq. (4.2) is the time at which data is expected to change at the input pin in order to avoid timing violations.

$$Slack_{Setup} = RequiredTime - ActualArrivalTime \tag{4.1}$$

$$Slack_{Hold} = ActualArrivalTime - RequiredTime \tag{4.2}$$

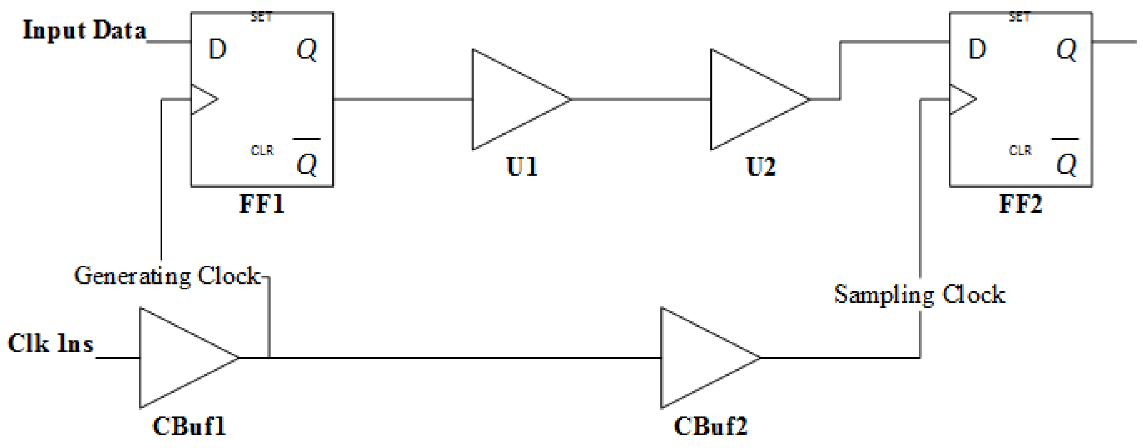


Figure 4.2: Example of Timing Path

Slew: Slew is a representation of the transition time for a signal. It is defined as the inverse of transition time tr. Slew represents how fast a signal transition occurs. Larger the slew value, faster will be the transitions. Depending on whether the signal is at input or output, slew is further classified into input slew and output slew.

$$Slew = 1/t_r \tag{4.3}$$

Timing Event: A timing event is the transition of the signal from one state to the other. A timing event can be early or late. The earliest possible transition of a signal is called early event and similarly the latest possible one is called late event. These early and late transitions (worst case scenarios for) are utilized for setup and hold analysis also known as max and min analysis respectively (where max and min represents maximum and minimum data path delay which is the criteria for these analyses).

Timing Arcs: A timing arc at a pin specifies whether a change in signal at the pin can influence the output (all logic cells can have multiple timing arcs depending on the number of input pins). For sequential cells like flip-flop and latches, there is a timing arc from clock to output also. There is also another property called unateness, which specifies how signals' transition at the input pin influences the output (the other inputs should be such that the gate is activated). The timing arc is positive unate if a rising signal at input pin results in a rising transition at the output. If the transition at output was a falling once, the arc would be categorized as a negative unate one.

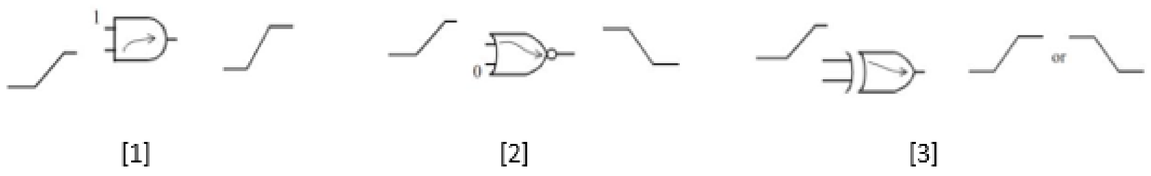


Figure 4.3: Timing Arcs

As shown in Figure 4.3.1 when one input of AND gate is at logic high, the output follows the transitions at the other input. So the timing arc is positive unate. For the NOR gate in Figure 4.3.2 when an input is at logic 0 state, the output transits opposite to that of the input transition. So the timing arc in Figure 2.9 is negative unite arc. For the XOR gate shown in Figure 4.3.3 the timing arc from selected input to output can either be positive or negative unate depending on the value of the second input. Such arcs are called non unate arcs.

Clock Skew: The variation in process and operating conditions can result in clock tree device delay changes. These variations result in ambiguity in the arrival time of clock signals and is called clock skew. The second clock variation comes from random noise and is called jitter. Jitter impacts the time period of clock. Another difference between skew and jitter is that skew is deterministic where as jitter is random. Clock skew penalty is a factor employed by STA tools to ensure that the signals remain stable for an additional time to account for skew and jitter.

False Paths: In STA, the methodology followed is worst case delay propagation through the timing paths. In such an approach it is highly possible for the algorithm to analyze paths which can never exist in a real circuit. To exclude these paths, it is necessary for the designer to specify the false paths, so that the timing engine will ignore those false paths from timing calculation and reporting. Consider a circuit scenario as shown in Figure 4.4

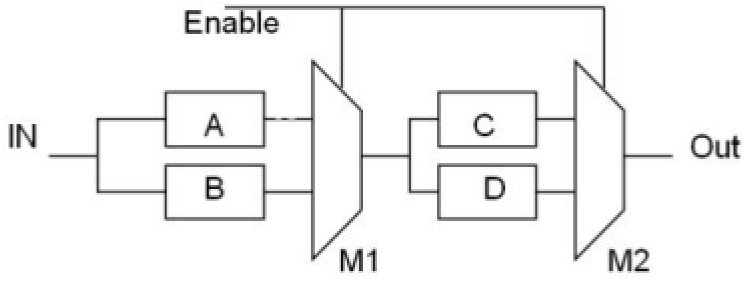


Figure 4.4: Example of False Path

From the circuit logic it is understood that data can propagate through A-M1-C-M2-Out or B-M1-D-M2-Out. Since STA doesn't perform logic analysis on the circuit, unless specified the tool will analyze the timing for data propagation through A-M1-D-M2-Out and B-M1-C-M2-Out. If the designer wants the tool to ignore these false paths, this should be specified in the timing constrains given as input to the tool. Specifying false paths is highly recommended as this will reduce the number of unnecessary timing calculations as well as timing fixes.

Operating Conditions: STA is performed on specific operating conditions. Process, Voltage and Temperature or in short PVT is alternatively called as operating conditions. Taking all the variability factors into consideration will lead to large number of combinations and is practically unfeasible. Taking large number of factors can result in a multi corner multi mode design, which is cumbersome and difficult to converge. So modern day ASICs have reduced the number of variability factors to ease the timing convergence but have added guard bands to ensure that the variability consideration is not ignored in margin analysis.

Max and Min Analysis: Max analysis is the setup analysis and min analysis is the hold analysis. Max and min essentially represent the delay of data path in the circuit. For max analysis the data generating path and data path is assumed to have maximum delay and sampling clock path has minimum delay (worst case scenario for setup analysis). In Figure 4.2, the generating path is Clk - CBuf1 - FF1 - U1 - U2. The sampling path is Clk - CBuf1 - CBuf2 - FF2. Here FF represents a Flip Flop, Clk represents clock and CBuf represents clock buffer. For min analysis the generating path and data path is assumed to have minimum delay and sampling path has maximum delay.

4.4 Delay Analysis

For timing analysis, design netlist, RC extraction, timing constraints and standard cell library characterization are required. The netlist and RC extraction data provides the design information to the timing analysis tool. The library characterization provides modeling information for calculating stage delays (cell delay). The timing constraints provided to the STA tool helps in simplifying the timing calculations and the usual constraints provided are multi cycle path overrides (by default, tools assume paths to be of length one cycle), false paths, dc signals etc. Another category of constraints that needs to be provided are environmental constraints which specify the timing requirements or arrival of signals from external world. For simplifying the physical design process, most of the modern day ASICs are partitioned into smaller blocks. While a designer is working on his block, there may be a large number of signals coming from and going to the neighboring blocks. For proper STA on the block, it is necessary to have the timing specifications available on all these interface pins. These specifications are also known as environmental constraints.

The outputs of STA are essentially the timing reports and timing constraints. The bare basic reports are setup and hold reports. Based on requirements, there can be various reports like capacitance reports of nets, resistance of nets etc dumped from the tool. The second set of outputs is timing constraints, similar to the environmental constraints which were fed as input for STA. As a part of timing analysis, a set of timing specifications on all the interface pins are produced which will be required by the relevant neighboring blocks (if these blocks have signals interacting each other). The input environmental constraints will specify the earliest and latest time a signal can be expected to be stable at the input interface pin and the earliest and latest time a signal is required to be stable at the output interface pin. The environmental constrains generated as a part of the output will also provide similar specifications as the input ones, but from the design block's perspective. This will specify the earliest and latest time a signal is required at the input pin of the block and earliest and latest time a signal will be stable at the output interface pin.

All the standard cells are characterized in such a way that the timing tool can calculate the stage delay without actually simulating it. STA tools have internal mechanism which could read the characterization and provide the stage delay based on circuit parameters. For stage delay calculation, the basic circuit parameters required are slew of the signal at the input of standard cell and load capacitance at the output pin. The characterization acts as a lookup table for the timing tool on which it will do interpolation or extrapolation to obtain delay and slew corresponding to the input signal slew and load capacitance.

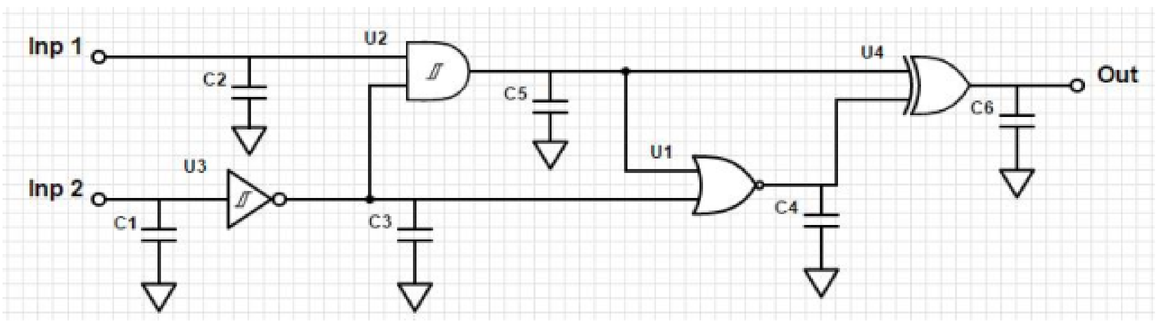


Figure 4.5: Circuit for calculating cell delays

While evaluating the cell delays (stage delays), a real circuit with R and C is viewed as a capacitor only network as shown in Figure 4.5. The output load capacitance is not the total capacitance but an effective capacitance representing the total load impedance. This effective capacitance is a function of

- Drive strength of cell
- Input impedance of the load seen from the driving cell

For a given interconnect, a cell with weak drive will have larger Ceff than strong driver. This can be easily understood with the help of Elmore delay modeling.

So the standard cells characterization can be represented as

$$CellDelay = f(inputslew, loadcap) \tag{4.4}$$

$$OutputSlew = f(inputslew, loadcap) \tag{4.5}$$

The timing tool performs analysis on individual timing paths instead of the whole design. A timing path is essentially a path from generating node to sampling node. The generating and sampling nodes can be anything from sequential cells like flops or latches or interface pins or gated clock cells. The timing engine identifies the possible timing paths from the netlist and user defined timing constraints. Once the tool identifies the timing paths, the next step is to calculate delay. For delay calculation, the modeling information like library characterization and extraction data is used. The extraction data is usually scaled and used for timing calculations. This scaling is carried out to ensure that the timing analysis captures the impacts of noise as the total capacitance value of a net can increase or decrease depending up on the cross coupled capacitance with neighboring signal nets. Further, the real R and C provided by the extraction can be highly segmented and complex for timing engine to operate upon. So before timing analysis the tool tries to simplify the extracted R and C by reducing the segmentation.

4.4.1 Interconnects Parasitics

In digital designs, a wire connecting pins of standard cells and blocks is referred to as a net. A net typically has only one driver while it can drive a number of fanout cells or blocks. After physical implementation, the net can travel on multiple metal layers of the chip. Various metal layers can have different resistance and capacitance values.

RLC for Interconnect: The interconnect resistance comes from the interconnect traces in various metal layers and vias in the design implementation. Figure 4.6 shows example nets traversing various metal layers and vias. Thus, the interconnect resistance can be considered as resistance between the output pin of a cell and the input pins of the fanout cells.

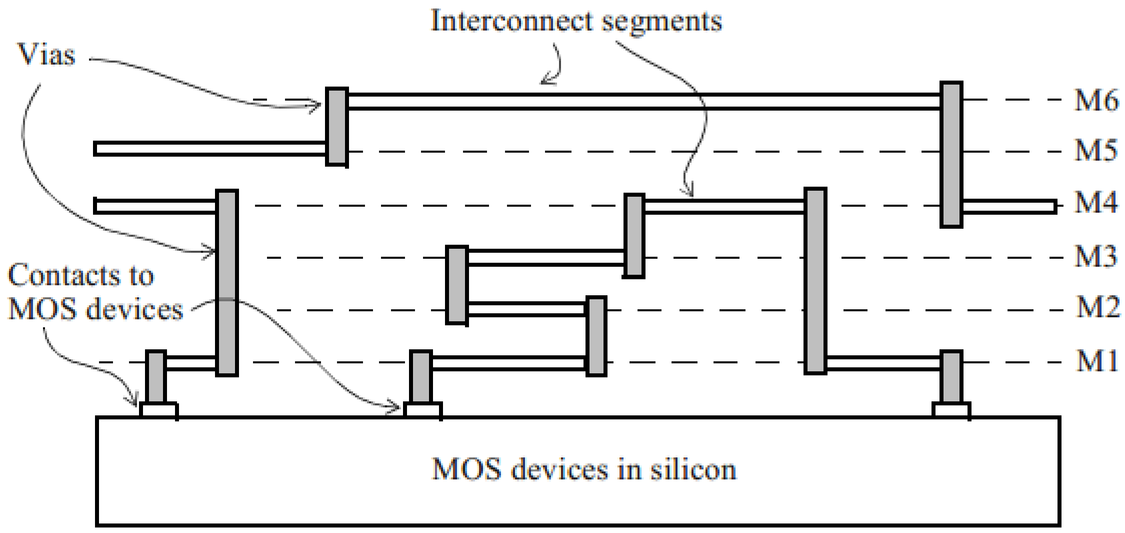
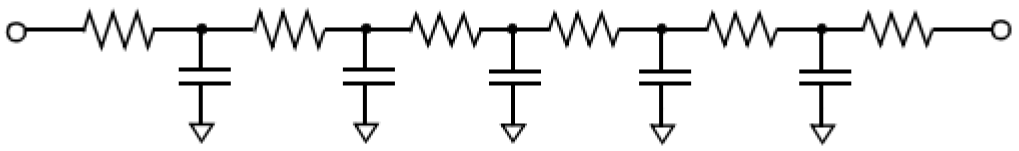


Figure 4.6: Nets on Metal Layers

The interconnect capacitance contribution is also from the metal traces and is comprised of grounded capacitance as well as capacitance between neighboring signal routes.

The inductance arises due to current loops. Typically the effect of inductance can be ignored within the chip and is only considered for package and board level analysis. In chip level designs, the current loops are narrow and short - which means that the

(a) Trace of length *L*.



(b) Distributed RC tree.

Figure 4.7: Interconnect Trace

current return path is through a power or ground signal routed in close proximity. In most cases, the on-chip inductance is not considered for the timing analysis.
The resistance and capacitance (RC) for a section of interconnect trace is ideally represented by a distributed RC tree as shown in Figure 4.7. In this figure, the total resistance and capacitance of the RC tree - Rt and Ct respectively - correspond to Rp * L and Cp * L where Rp, Cp are per unit length values of interconnect resistance and capacitance for the trace and L is the trace length. The Rp, Cp values are typically obtained from the extracted parasitics for various configurations and is provided by the ASIC foundry.

Representation of Extracted Parasitics

Parasitics extracted from a layout can be described in three formats:

- Detailed Standard Parasitic Format (DSPF)
- Reduced Standard Parasitic Format (RSPF)
- Standard Parasitic Extraction Format (SPEF)

Here in Synopsis tools we use SPEF format.

Standard Parasitic Exchange Format

The SPEF is a compact format which allows the representation of the detailed parasitics. An example of a net with two fanouts is shown below.

```
*D_NET NET_27 0.77181
*CONN
*I *8:Q O *L 0 *D CELL1
*I *10:I I*L 12.3
*CAP
1 *9:0 0.00372945
2 *9:1 0.0206066
3 *9:2 0.035503
4 *9:3 0.0186259
5 *9:4 0.0117878
6 *9:5 0.0189788
7 *9:6 0.0194256
8 *9:7 0.0122347
9 *9:8 0.00972101
10 *9:9 0.298681
11 *9:10 0.305738
12 *9:11 0.0167775
*RES
1 *9:0 *9:1 0.0327394
2 *9:1 *9:2 0.116926
3 *9:2 *9:3 0.119265
4 *9:4 *9:5 0.0122066
5 *9:5 *9:6 0.0122066
6 *9:6 *9:7 0.0122066
7 *9:8 *9:9 0.142205
8 *9:9 *9:10 3.85904
```

```
9 *9:10 *9:11 0.142205
10 *9:12 *9:2 1.33151
11 *9:13 *9:6 1.33151
12 *9:1 *9:9 1.33151
13 *9:5 *9:10 1.33151
14 *9:12 *8:Q 0
15 *9:13 *10:I 0
*END
```

The units of the parasitics R and C are specified at the beginning of the SPEF file. Due to its compactness and completeness of representation, SPEF is the format of choice for representing the parasitics in a design.

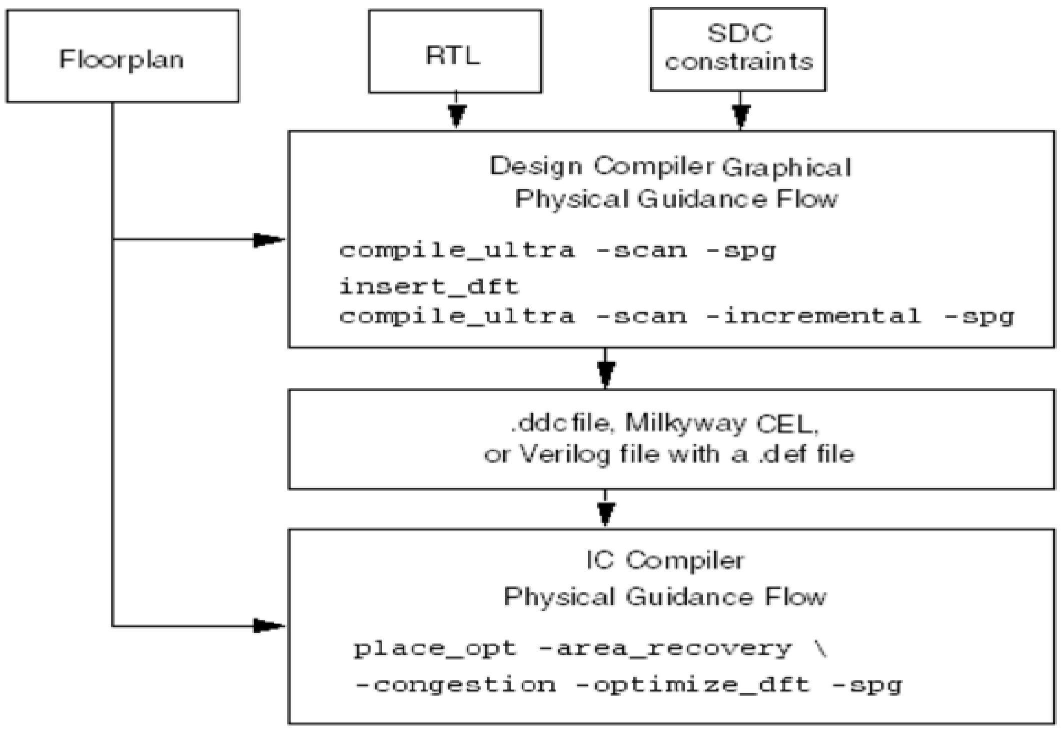
Chapter 5
Synopsys Physical Guidance

Design Compiler and IC Compiler together support the physical guidance flow to improve routability, runtime, correlation, along with reducing routing-related congestion. Physical Guidance speeds up IC Compiler runtime because the output from Design Compiler Graphical acts as a seed placement for IC Compiler and hence goes through fewer placement steps.
There must be consistency of design and tool setup between Design Compiler and IC Compiler for ensuring good Qor and correlation between both the tools. We have to use consistent library settings along with the same sets of logic and physical libraries, and consistent design goal specifications such as design rule settings and timing constraints for logical and physical implementation through both tools.
You enable the physical guidance flow by using the -spg option with the compile_ultra command in Design Compiler Graphical and by using the -spg option with the place_opt command in IC Compiler. The -spg option works seamlessly with the existing compile_ultra options in Design Compiler and with the place_opt options in IC Compiler.Physical guidance performs the following functions:

- In Design Compiler Graphical, the -spg option provides the following:
  - Further refined placement that is consistent with the place_opt command in IC Compiler
  - Enhanced post-placement delay optimization to provide a better optimized starting netlist for physical implementation
  - Optimizes the design for congestion to improve routability
- In IC Compiler, the -spg option enables the place_opt command to use the Design Compiler Graphical placement as its seed placement. The place_opt

command is enhanced to use topographical mode placement as the starting point in physical guidance mode.

You can save the placement information derived by Design Compiler Graphical in binary format, in a .ddc file or a Milkyway CEL view, or you can save it in a .def file.


Floorplan
RTL
SDC constraints
Design Compiler Graphical Physical Guidance Flow
.ddc file, Milkyway CEL, or Verilog file with a .def file
IC Compiler Physical Guidance Flow

```
compile_ultra -scan -spg
insert_dft
compile_ultra -scan -incremental -spg
```

```
place_opt -area_recovery \
-congestion -optimize_dft -spg
```

Figure 5.1: Physical Guidance Flow

The physical guidance flow in IC Compiler supports all the compile options used by Design Compiler in topographical mode, such as -scan, -clock_gate, and -incremental. In addition, the flow supports the -congestion option, which reduces routing-related congestion. Note that the -congestion option is not needed in Design Compiler because Design Compiler reduces routing-related congestion by default with the -spg option.The physical guidance flow in IC Compiler also supports the multicorner-multimode based flow, the multivoltage-UPF flow, the power flow, the DFT flow, and the hierarchical flow.

Use the following guidelines to maintain good QoR and correlation between Design

Compiler and IC Compiler when you use the physical guidance flow:

- Use consistent design and tool setup between Design Compiler topographical mode and IC Compiler. That is, you should use the same sets of logic and physical libraries to drive both tools and use consistent library settings, including the dont_use list. Also, use consistent design goal specifications, such as timing constraints and design rule settings to drive logical and physical implementation. You should load the same sets of the following files in both tools:

  - Logic library (.db)
  - Milkyway reference library
  - Technology file (.tf)
  - Mapping file (maps the technology file layer names to TLUPlus layer names)
  - Min and max TLUPlus files for net parasitics

  Make sure that each standard cell and macro in the logic libraries has a corresponding abstract physical view in the Milkyway reference library. Otherwise, the cells are treated as dont_use cells for mapping and optimization.In addition, you should use consistent library-specific attribute settings with IC Compiler. For instance, your library cell dont_use list, dont_touch settings, and design rule settings (maximum allowable fanout load and transition time) should be aligned with the settings used in IC Compiler.

- Apply a complete set of physical constraints by using a floorplan DEF file or Tcl commands before the first compile_ultra command step in Design Compiler. Also, reapply the same sets of physical constraints in IC Compiler to preserve the placement consistency for physical guidance.The same DEF file must be used by both tools. The DEF file contains floorplan information that must be consistent between tools. The tools issue various errors and warnings if your constraints, floorplan information, and libraries are not consistent.

- Since the physical guidance information is restored when you run the IC Compiler place_opt command, it is recommended that you run the place_opt -spg command as the first placement-related command in IC Compiler so that Design Compiler placement is restored properly. Using other placement-related commands (such as create_placement, remove_buffer_tree, and so on) might degrade correlation and can lead to the loss of some physical guidance information.

5.1 Reducing Routing Congestion

Wire-routing congestion in a design occurs when the number of wires traversing a particular region is greater than the capacity of the region. If the ratio of usage-to-capacity is greater than 1, the region is considered to have congestion problems. Congestion can be caused by standard cells or the floorplan. Typically, standard cell congestion is caused by the netlist topology, for example, large multiplexer trees, large sums of products (ROMs), test decompression logic, or test compression logic. Floorplan congestion can be caused by macro placement or port location.

If Design Compiler passes a congested design to IC Compiler, it might be difficult and time consuming for IC Compiler to reduce the congestion. Design Compiler can reduce congestion more easily than IC Compiler because Design Compiler can optimize RTL structures. Traditionally, to minimize congestion related to the netlist topology, you would use scripting solutions to restrict technology mapping tools, such as Design Compiler and IC Compiler, from choosing library cells that you perceive as poor for congestion. But these solutions do not significantly improve congestion; instead, they often create poor timing or area results.

Design Compiler Graphical predicts wire-routing congestion hot spots during RTL synthesis and uses its interactive capability to visualize the designâĂŹs congestion. When the tool determines that the design has congestion problems, it minimizes congestion through congestion optimization, resulting in a better starting point for layout.

Design Compiler uses the Synopsys virtual routing technology to predict wire-routing congestion. In this technology, congestion is estimated by dividing the design into a virtual grid of global routing cells followed by a global route to count the number of wires crossing each grid edge. The capacity and size of each global routing cell is calculated by using the physical library information. A cell is considered congested if the number of wires passing through it is greater than the number of available tracks. Design Compiler predicts the congestion number by subtracting the maximum wires allowed for a particular edge direction from the total number of wires crossing per edge. Any number over zero is a congestion violation. This calculation is represented by the following equation:

$$Congestion = (Number\_of\_wires) - (Maximum\_wires\_allowed) \tag{5.1}$$

The tool considers floorplan-related physical constraints when it estimates routing congestion. In particular, the wiring keepout physical constraint ensures that the tool is aware of areas that must be avoided during routing. This physical constraint helps to achieve consistent congestion correlation with layout.

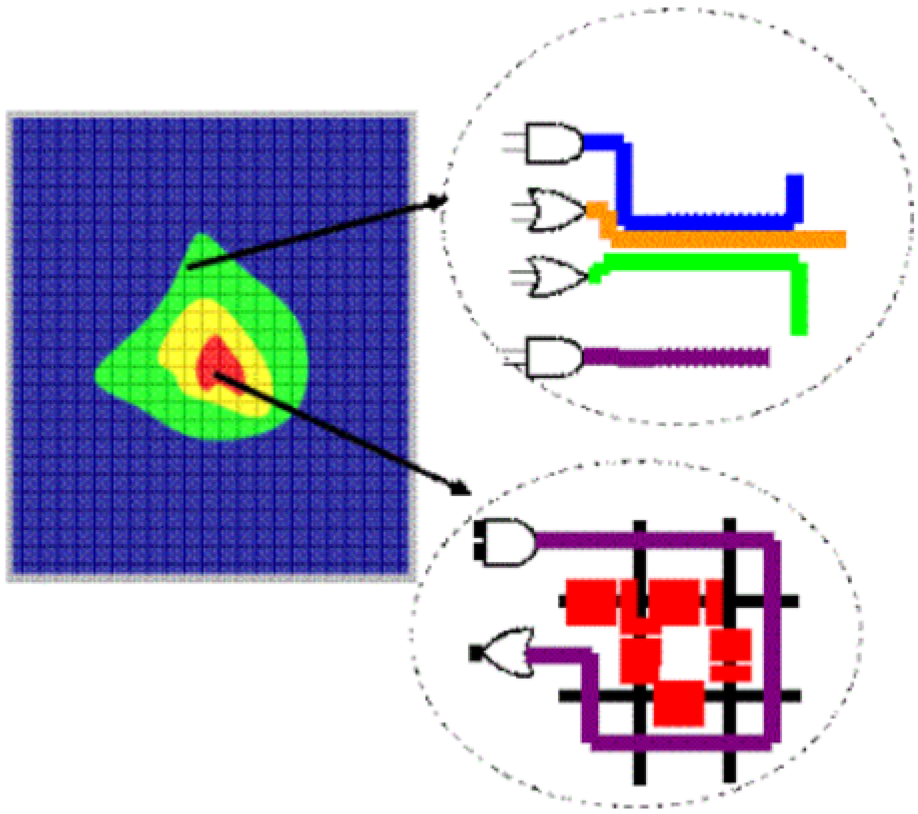


Figure 5.2: Predicting Congestion Using Virtual Grid Route

5.2 Controlling Placement Density

You can control how densely cells can be packed during placement by setting the placer_max_cell_density_threshold variable. This variable enables a mode of coarse placement in which cells are not distributed evenly across the surface of the chip, but are allowed to clump together. The value you specify sets the threshold of how tightly the cells are allowed to clump. The default is -1, which disables this feature, and cells are distributed evenly across the surface of the chip.

A reasonable value is one that is above the overall utilization of the design but below 1.0. For example, if your overall utilization is 50%, or 0.5, a reasonable value for this variable is a value between 0.5 and 1.0. The higher the value, the more tightly the cells clump together.

For example, if the global utilization is 50 percent, choose a value between 0.5 and 1:

dc_shell-topo> set_app_var placer_max_cell_density_threshold 0.7

If you have not set this variable and the tool detects a design with low utilization, the tool automatically uses a value of 0.5 internally during optimization. This check avoids long route QoR issues for low-utilization designs. The check occurs during the compile_ultra -spg (full and incremental) and place_opt -spg commands. The variable is reset to its default of -1 when the command completes.

The placer_max_cell_density_threshold value is not transferred to the IC Compiler tool. You must set it to be consistent with Design Compiler Graphical.

As with Design Compiler Graphical, the IC Compiler user-specified placer_max_cell-_density_threshold value takes precedence even if the design has low utilization and the user-specified value is set lower than 0.5; however, the tool issues a warning.

5.3 Using Layer Optimization to Increase the Accuracy of Net Delay Estimation

When Design Compiler computes the resistance and capacitance of a net, it uses an average based on the resistance and capacitance of all available layers from the logic library. That average works well when all layers have the same or close enough unit resistance and capacitance value. However, with submicron technologies, the layer characteristics can vary greatly. Such variation can cause correlation or timing issues for nets that are known to be routed with only some specified layers. In this case, averaging is not an appropriate solution.

Layer optimization in Design Compiler Graphical allows you to define specific layer assignment constraints and apply them to nets. This increases the accuracy of net delay estimation during optimization. Layer optimization is supported only in Design Compiler Graphical, and it is performed automatically when you use the -spg option with the compile_ultra command. To disable automatic layer optimization, use the -no_auto_layer_optimization option when you run the compile_ultra -spg or the compile_ultra -spg-incremental command.

You can perform layer optimization with the following methods:

- A net pattern
- User constraints
- Automatic recognition

In the net pattern methodology, you use the create_net_search_pattern command to define a pattern to identify nets, and you use the set_net_search_pattern_delay_estimation_options command to define which layers to use for a specific pattern.
In the user constraints methodology, you use the set_net_routing_layer_constraints command to specify which nets will have specific layer assignment constraints applied to them.
In the automatic recognition methodology, you allow Design Compiler Graphical to automatically identify nets and assign them to specific layers to reduce the resistance and capacitance on those nets, instead of using the averaging technique.
Regardless of the methodology you use, layer optimization can have an effect on runtime.

Chapter 6

Implementation

As prerequisite for project, the proper understanding of the backend flow,i.e, from RTL to GDS2 is required. The logical synthesis and place and route steps play a key role in design flow. Using a block level design we learnt synthesis on Design Compiler-topographical tool, provided by Synopsys and the place and route through IC Compiler tool, also provided by Synopsys.
Now in the following sections the actions taken towards the completion of the project are portrayed.

6.1 Figuring Out Buffering Requirement For Critical Signals During Synthesis

6.1.1 Gate Level Netlist

To get a gate level netlist for an IC design from it's RTL is synthesis. We use Synopsys Design Compiler tool for synthesis. The inputs required for synthesis are RTL list, library files and constraints file. There are three type of libraries required by the DC tool. The first one is Technology Libraries, which contains basic logic gates and flip-flop.It can be given as Target library and Link library. It is used for technology mapping. The second is DesignWare libraries which contains complex cells like adders, comparators, etc. It is also called Synthetic library. The last is Symbol library.
Constraints are the designer's specification and environmental restrictions under which synthesis is to be performed. For optimum results, designers have to methodically constraint their design by describing the design environment, target objectives and design rules. Define the environment by specifying operating conditions, wireload models and system interface characteristics. Operating conditions include PVT conditions. System interface characteristics include input drives, input and output loads

and fanout loads. Design Constraints consist of

- Timing
- Power
- Area

Timing constraints include clock frequency, input and output delay, max and min transitions. Area is set to zero to maximize efficiency. In power constraint we define maximum leakage power. The highest priority is given to timing during optimization. For giving the clock information to the tool , we have gathered the complete data which is used to define clocks and their relationship into one central location, .xls file. Now using a script this file is converted to a TCL file which contains all the procedures related to generation of the clocks. This is beneficial as to debug or understand a script is rather difficult than to go through a excel spreadsheet. Any new data can be easily uploaded and easy review process.

The outputs for synthesis are

- Gate level netlist
- *.sdc file
- *.def file

Reports generated:

- Timing Report
- Area Report
- Power Report

- Quality of Report-Overall quality of synthesis

DC synthesis is important as the timing if satisfied here, then only this generated netlist is given as input to place and route tool.

6.1.2 Fetching the Coordinates

After generating the gate-level netlist for the family design, we need the pin coordinates of each block wrt to family.
To achieve this, we have to write a TCl script. In the script first we have to read the DEF or LEF file for each block and fetching it's pin coordinates. Then we need to get the location of these pins wrt family. So, we read the floorplan file for the family and then get the exact coordinates for the pins wrt family floorplan.
A Library Exchange Format (LEF) file contains library information for a class of designs. It defines the elements of an IC process technology and associated library of cell models. Library data includes layer, via, placement site type, and macro cell definitions.LEF inputs are in ASCII form.
General Rules for creating a LEF file:

- Indentifiers like net names and cell names are limited to 2,048 characters.
- Distance is specified in microns.
- Distance precision is controlled by the UNITS statement.
- LEF statements end with a semicolon (;). Space must be left between the last character in the statement and the semicolon.

You can define all of your library information in a single LEF file; however this creates a large file that can be complex and hard to manage. Instead, you can divide the information into two files, a âĂIJtechnologyâĂİ LEF file and a âĂIJcell libraryâĂİ LEF file. A technology LEF file contains all of the LEF technology information for a design, such as placement and routing design rules, and process information for layers.

A technology LEF file can include any of the following LEF statements:

[VERSION statement]

[BUSBITCHARS statement]

[DIVIDERCHAR statement]

[UNITS statement]

[MANUFACTURINGGRID statement]

[USEMINSPACING statement]

[CLEARANCEMEASURE statement ;]

[PROPERTYDEFINITIONS statement]

[LAYER (Nonrouting) statement

| LAYER (Routing) statement] ...

[MAXVIASTACK statement]

[VIA statement] ...

[VIARULE statement] ...

[VIARULE GENERATE statement] ...

[NONDEFAULTRULE statement] ...

[SITE statement] ...

[BEGINEXT statement] ...

[END LIBRARY]

A cell library LEF file contains the macro and standard cell information for a design.

A library LEF file can include any of the following statements:

[VERSION statement]

[BUSBITCHARS statement]

[DIVIDERCHAR statement]

[VIA statement] ...

[SITE statement]

[MACRO statement

[PIN statement] ...

[OBS statement ...]] ...
[BEGINEXT statement] ...
[END LIBRARY]

When reading in LEF files, always read in the technology LEF file first. Statements in LEF file can be defined in any order, but the data must be defined before it is used.For example, the UNITS statement must be defined before any statements that use values that are dependent on UNITS values, LAYER statements must be defined before statements that use the layer names.

The PIN statement which contains the information of pin name , its assigned layer and its coordinates, is included in the LEF specification for each macro. The PIN statement defines the pins for the macro. All pins including the VSS and VDD must be specified.

The pinname specifies the name for the library pin.

The statement of our concern within the PIN section is PORT. This defines a collection of geometries that are electrically equivalent points (strongly connected). A pin can have multiple ports. Each PORT of the same PIN is considered weakly connected to the other PORTs, and should already be connected inside the MACRO (often through a resistive path). Strongly connected shapes (that is, multiple shapes of one PORT) indicate that a signal router is allowed to connect to one shape of the PORT, and continue routing from another shape of the same PORT. Weakly connected shapes should not be used by routers. The syntax for describing pin port statements is defined as follows:

PORT
[CLASS NONE | CORE | BUMP ;]
{layerGeometries} ...
END ...

layerGeometries defines port geometries for the pin.

A port can be one of the following:

- BUMP - Specifies the port is a bump connection point. A bump port should only be connected by routing to a bump (normally a MACRO CLASS COVER BUMP cell).

- CORE - Specifies the port is a core ring connection point. A core port is used only on power and ground I/O drivers used around the periphery. The core port indicates which power or ground port to connect to a core ring for the chip (inside the I/O pads).
- NONE - Specifies the port is a default port that is connected by normal "default" routing. NONE is the default value if no PORT CLASS statement is specified.

A Design Exchange Format (DEF) file contains the design-specific information of a circuit and is a representation of the design at any point during the layout process. It defines the elements of an IC design relevant to physical layout, including the netlist and design constraints. The DEF inputs are in ASCII form.
DEF conveys logical design data to, and physical design data from, place-and-route tools. Logical design data can include internal connectivity (represented by a netlist), grouping information, and physical constraints. Physical data includes placement locations and orientations, routing geometry data, and logical design changes for back-annotation. Place-and-route tools also can read physical design data, for example, to perform ECO changes. For standard-cell-based/ASIC flow tools, floorplanning is part of the design flow. You typically use the various floorplanning commands to interactively create a floorplan. This data then becomes part of the physical data output for the design using the ROWS, TRACKS, GCELLGRID, and DIEAREA statements. You also can manually enter this data into DEF to create the floorplan. It is legal for a DEF file to contain only floorplanning information, such as ROWS. In many cases, the DEF netlist information is in a separate format, such as Verilog, or in a separate DEF file. It is also common to have a DEF file that only contains a COMPONENTS section to pass placement information. General Rules for creating a DEF file:

- Indentifiers like net names and cell names are limited to 2,048 characters.

- DEF statements end with a semicolon (;). You must leave a space before the semicolon.
- Each section can be specified only once. Sections end with END SECTION.
- One must define all objects before referencing them except for the + ORIGINAL argument in the NETS section.
- No regular expressions or wildcard characters are recognized except for (* pinName) in the SPECIALNETS section.

The information which we want to extract is present in the PINS section. It defines external pins. Each pin definition assigns a pin name for the external pin and associates the pin name with a corresponding internal net name. The pin name and the net name can be the same. When the design is a chip rather than a block, the PINS statement describes logical pins, without placement or physical information. Within this the PORT section conatins all the LAYER, POLYGON, and VIA statements are all part of one PORT connection, until another PORT statement occurs. If this statement is missing, all of the LAYER, POLYGON, and VIA statements are part of a single implicit PORT for the PIN. This commonly occurs for power and ground pins. All of the shapes of one port (rectangles, polygons, and vias) should already be connected with just the port shapes; therefore, the router only needs to connect to one of the shapes for the port. Separate ports should each be connected by routing inside the block (and each DEF PORT should map to a single LEF PORT in the equivalent LEF abstract for this block). The syntax for describing PORT statements is defined as follows:
[[+ PORT]
[+ LAYER layerName
[SPACING minSpacing
| DESIGNRULEWIDTH effectiveWidth]
pt pt

| + POLYGON layerName

[SPACING minSpacing

| DESIGNRULEWIDTH effectiveWidth]

pt pt pt

| + VIA viaName pt

] ...

]

LAYER layerName pt pt : Specifies the routing layer used for the pin, and the pin geometry on that layer. If you specify a placement status for a pin, you must include a LAYER statement.

POLYGON layerName pt pt pt : Specifies the layer and a sequence of at least three points to generate a polygon for this pin. The polygon edges must be parallel to the x axis, the y axis, or at a 45-degree angle.

The floorplan file for the family consists of the instances fr each of the block, along with the block's coordinates. Now we read the floorplan file and fetch the coordinate of each instance of the blocks. Using this coordinate as the reference coordinate wrt to family, we manipulate the coordinates extracted from the DEF/LEF files for the block.

6.2 Generating Optimized Netlist Using SPG

6.2.1 Enabling the Physical Guidance Flow

To enable physical guidance in Design Compiler, use the compile_ultra command with the -spg option. You must specify the die area at a minimum to ensure floorplan consistency between Design Compiler and IC Compiler. It is recommended that you also specify the macro and block abstraction locations, the site arrays, and the port locations for all primary I/Os.

Design Compiler performs explicit physical guidance usability checks the first time you run the compile_ultra -spg command in topographical mode to ensure that you have provided the required floorplan information, and error messages and warning messages are issued for any missing floorplan information. For example, Design Compiler issues a warning message if a macro location is missing during a physical guidance flow; however, the compile_ultra -spg command assigns the macro location automatically. If a port, macro, or block abstraction location is missing, you can proceed with synthesis. However, you must provide the location information in IC Compiler. Otherwise, correlation could be affected. Alternatively, you can use Design Compiler Graphical floorplan exploration to place the ports, macros, or block abstractions. After you specify the physical constraints, update the floorplan in floorplan exploration. The compile_ultra -spg command can move ports incrementally to enhance timing if the port locations have not been defined.

The IC Compiler restore_spg_placement command restores the port locations created by the Design Compiler compile_ultra -spg command for ports that were not constrained by the create_terminal command or defined in the DEF file. The restore_spg_placement command also restores the standard cell placement created by the compile_ultra -spg command. For more information, see the IC Compiler documentation.

In the physical guidance flow, the tool uses both RTL congestion optimization and congestion-aware cell placement to consider cell density and provide a more accurate net model for optimization in topographical mode. This helps to ensure that timing is consistent between the endpoint in Design Compiler and the startpoint in IC Compiler. In addition, placement and placement-based optimization in physical guidance enhances the delay optimization effort in Design Compiler so it is consistent with IC Compiler behavior.

At the end of the compile_ultra -spg command run, Design Compiler creates signatures of the physical constraints that were used. The signatures are saved in binary format, in either the .ddc file or the Milkyway CEL, to be later matched in IC Compiler against the physical constraints used for the place_opt -spg command run. This allows you to ensure that the physical constraints used in Design Compiler match the physical constraints used in IC Compiler.

At this point in the flow, the mapped top-level design is marked with a read-only dct_spg_flow_done attribute to signify that the design was implemented with physical guidance. You can query the attribute to help determine if your design .ddc file or Milkyway CEL was generated using physical guidance.

The derived placement for standard cells and the design floorplan information used to drive synthesis are retained on the in-memory implemented design. You can save the physical guidance information in a .ddc file by using the write_file command or to a Milkyway CEL view by using the write_milkyway command. In the ASCII flow, you can save the physical guidance information in a .def file by using floorplan exploration.

6.2.2 Enhancing Placement in Synthesis

Once the design floorplan information is read in the tool for synthesis flow, we aim to enhance the placement of standard cells in synthesis. Generally the DC tool places the standard cells in the centre of the design. But our aim is to place the standard cells in synthesis and then pass this information to place and route tool through a more optimized netlist. Once we place these standard cells in the synthesis stage, then we can estimate the timing and also it will help prevent the creation of hot-spots and routing congestion.

Now once the tool places all the macros after reading the floorplan information through a tcl file or def file, we create major bounds for all the major macros present in the design. For this we need to know the core area and along with it we must have the location for tha macros. These major bounds are created such that all the standard cells interacting with a particular macro is placed in the bound created above that macro.

For creation of macros we need to write a TCL script. In this first we get collection of ports and for those ports we create a collection of all the cells. Then we create the bounds using create_bounds command. Similarly we create the TX-RX bounds.

These bounds are necessary as the major timing delay comes from the TX-RX data paths. So if during synthesis we place the standard cells in the proper TX-RX region, reducing the data path to be traversed by the signal, we reduce the delay and hence getting a better timing closure.

Now, once the bounds are created, we must take care that the standard cells placed within the bounds do not cluster too close to each other. If such condition arises then there is a high possibility of routing congestion and the placement density will also be affected. In order to avoid these problems, we write out a TCL script for staggered placement blockage. Staggered placement blockage term points to the idea that partial blockages will be created throughout the bounds region, barring the tool from placing the standard cells within the blockages.Now due to this, the standard cells will be at a proper distance avoiding high placement density issues. In this script we have the choice of specifying the percentage of blockage we need for creating the partial blockages. The create_placement_blockages command along with the coordinates and the type of blockages will be used for creation of staggered placement blockages.

6.2.3 Exporting the Design

Saving in Binary Format: You can save standard cell placement in .ddc format by using the write_file -format ddc command, or save it in Milkyway CEL format by using the write_milkyway command.

In regular topographical mode, the standard cell placement can only be used by Design Compiler for further optimization. IC Compiler does not use it. However, in the physical guidance flow, the standard cell placement information is propagated to IC Compiler through the .ddc file or the Milkyway CEL.

The physical constraints and floorplan information used for synthesis are not passed from Design Compiler to IC Compiler. However, physical constraint signatures are created from the physical constraints at the end of the compile_ultra -spg and the compile_ultra -incremental-spg command steps in the physical guidance flow. These signatures are saved in the .ddc file or Milkyway CEL view. IC Compiler reads the signatures when the physical guidance information is restored and matches them to the physical guidance information in IC Compiler to ensure consistency between the floorplan used in synthesis and the floorplan used for physical implementation.

All binary netlists that are generated in the physical guidance synthesis flow, whether a .ddc file or a Milkyway CEL, also contain a read-only dct_spg_flow_done attribute to signify that the design was implemented with the physical guidance flow. You can query the attribute to determine if your design .ddc file or Milkyway CEL was generated using the physical guidance flow.

Saving in ASCII Format: When you save the design in ASCII format, either in a Verilog or a VHDL netlist, the file that is created does not contain standard cell placement information or physical constraint signatures.

You can save the standard cell placement information in a DEF file by using the write_def command in Design Compiler or in floorplan exploration. You can then read the DEF file, along with the ASCII netlist, back into Design Compiler to be optimized with the compile_ultra -incremental -spg command, or you can read it into IC Compiler before the place_opt -spg step.

6.2.4 Using Physical Guidance in IC Compiler

To enable physical guidance in IC Compiler, use the -spg option with the place_opt command. When you run the place_opt -spg command, the stored physical guidance information from Design Compiler is used as the initial seed placement.

In Design Compiler, we have passed the floorplan information, along with the bounds as well as the staggered blockages information due to which the standard cells placement is affected. All this inofrmation is passed to the ICC and then we compare the results.

Chapter 7
Results

7.1 Reporting Physical Guidance Information

The physical guidance information is stored with the design in binary format. It is not available as a standalone file. To view the physical guidance information, you must run IC Compiler and execute the place_opt -spg command.
For correlation result comparison between non-physical guidance and physical guidance flows, you should compare post-synthesis results against respective post-placement results for each flow. However, you should measure and compare post-placement results instead of post-synthesis results when looking at overall QoR comparison between non-physical guidance and physical guidance flows.

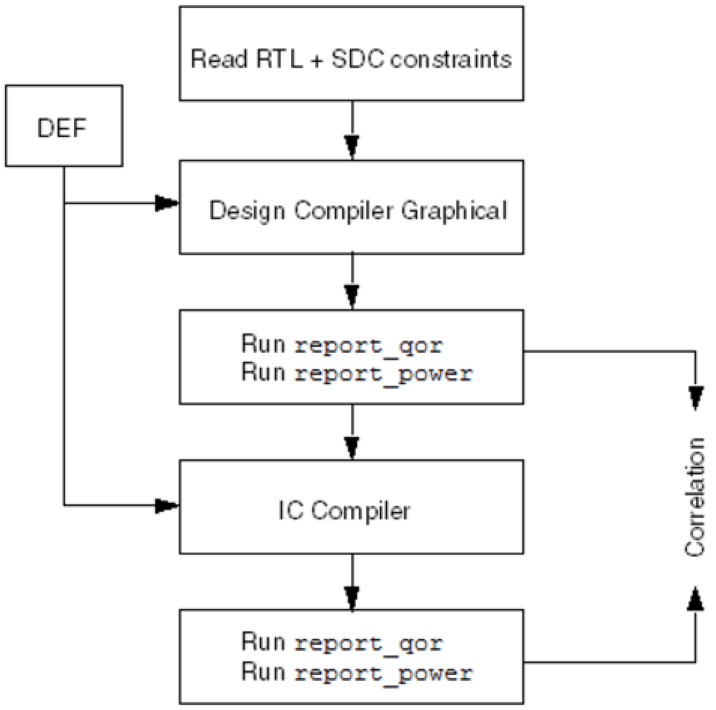
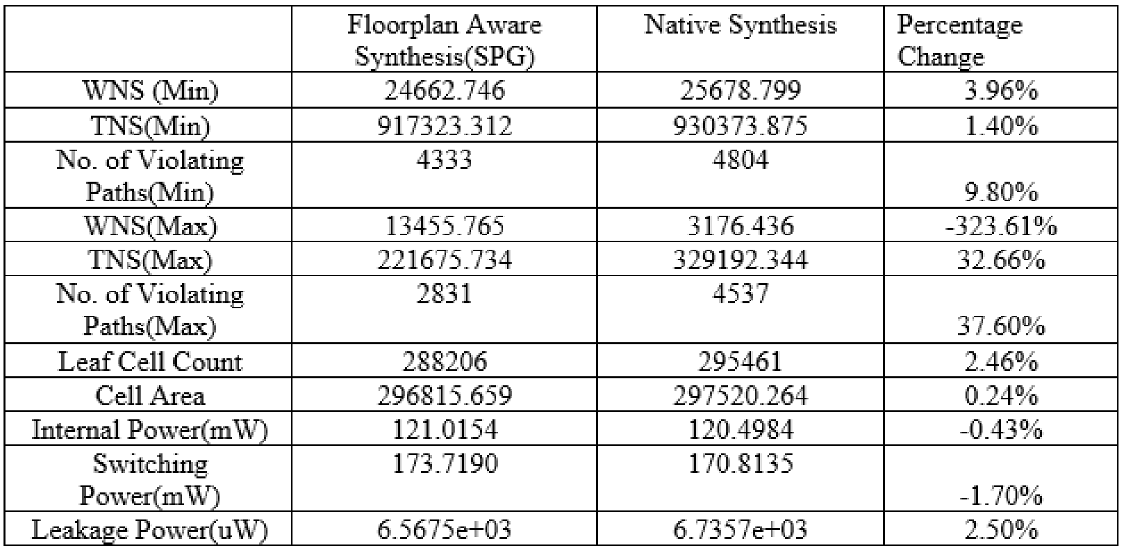


Figure 7.1: Correlation Measurement Between Design Compiler and IC Compiler



	Floorplan Aware Synthesis(SPG)	Native Synthesis	Percentage Change
WNS (Min)	24662.746	25678.799	3.96%
TNS(Min)	917323.312	930373.875	1.40%
No. of Violating Paths(Min)	4333	4804	9.80%
WNS(Max)	13455.765	3176.436	-323.61%
TNS(Max)	221675.734	329192.344	32.66%
No. of Violating Paths(Max)	2831	4537	37.60%
Leaf Cell Count	288206	295461	2.46%
Cell Area	296815.659	297520.264	0.24%
Internal Power(mW)	121.0154	120.4984	-0.43%
Switching Power(mW)	173.7190	170.8135	-1.70%
Leakage Power(uW)	6.5675e+03	6.7357e+03	2.50%

Figure 7.2: Comparison of Synthesis

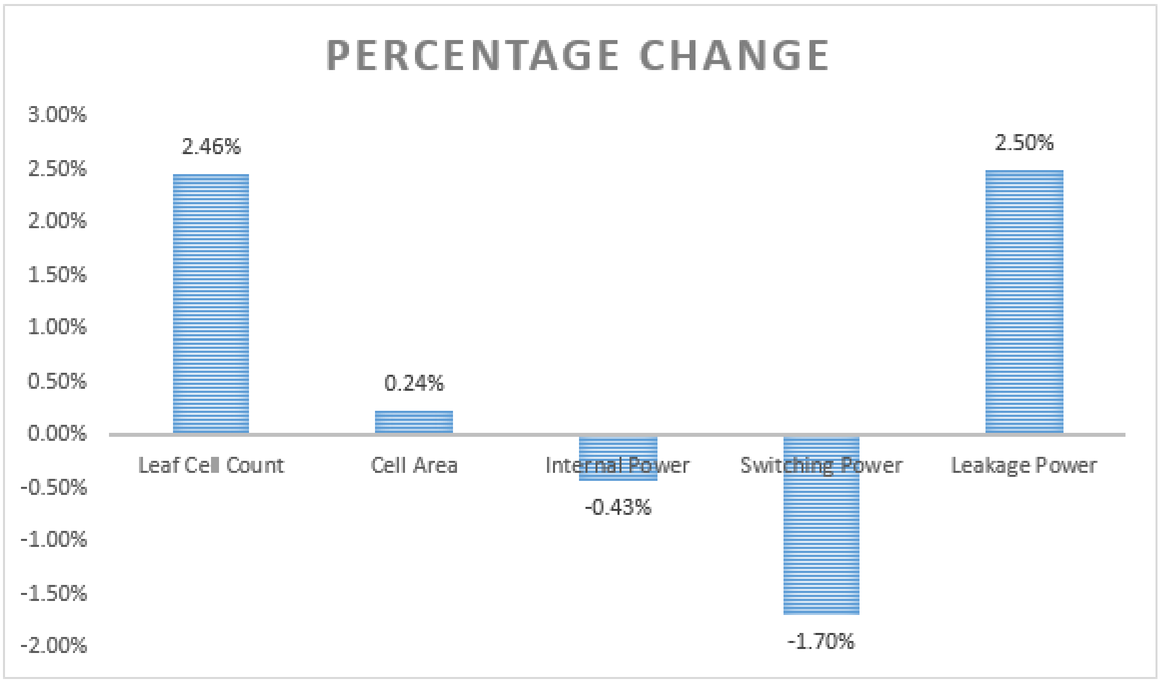
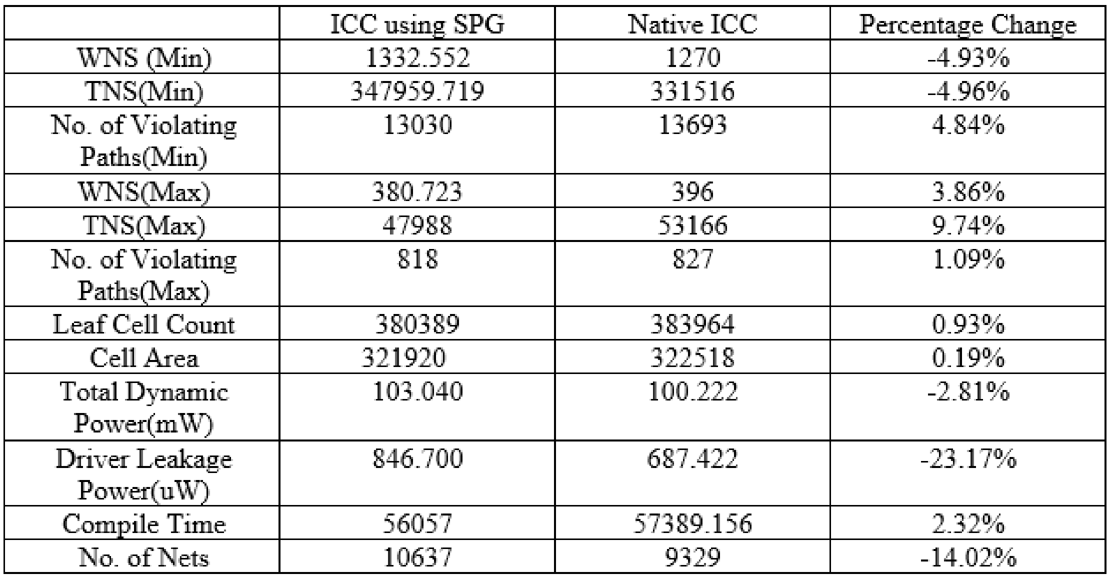


Figure 7.3: Percentage Change in Synthesis



	ICC using SPG	Native ICC	Percentage Change
WNS (Min)	1332.552	1270	-4.93%
TNS(Min)	347959.719	331516	-4.96%
No. of Violating Paths(Min)	13030	13693	4.84%
WNS(Max)	380.723	396	3.86%
TNS(Max)	47988	53166	9.74%
No. of Violating Paths(Max)	818	827	1.09%
Leaf Cell Count	380389	383964	0.93%
Cell Area	321920	322518	0.19%
Total Dynamic Power(mW)	103.040	100.222	-2.81%
Driver Leakage Power(uW)	846.700	687.422	-23.17%
Compile Time	56057	57389.156	2.32%
No. of Nets	10637	9329	-14.02%

Figure 7.4: Comparison of PnR Runs

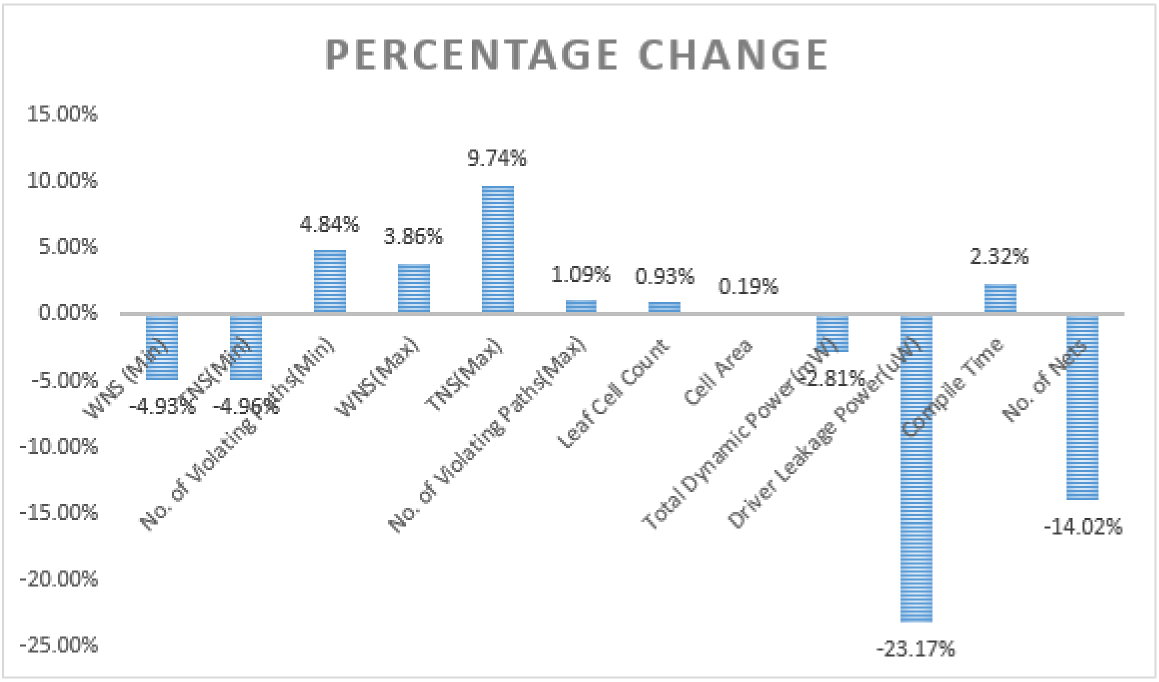


Figure 7.5: Percentage Change in PnR

Chapter 8

Conclusion

From the results we can conclude that the floorplan aware synthesis degrades the WNS but reduces the number of violating paths along with the cell area by .24% . Now this newly optimized netlist when taken to PnR, we see that WNS is improved along with major improvement in TNS. We can also see that there is improvement in the violating paths for both min and max, which will help in timing closure of the block. As the violating paths are getting reduced so it will be easier for fixing the setup violations with the PT-ECO Flow on this.

The main problem that we see is the power dissipation increment with the SPG flow. The reason behind this is due to the usage of Low VT Cells in the design during the optimization.

The main objective was to reduce the runtime. The usual spg flow improves the runtime by 1.5%, but due to the modifications we can see that we have achieved a improvement of 2.3%.

References

[1] Pinaki Chakrabarti, Vikram Bhatt, Dwight Hill, Aiqun Cao, Clock mesh framework, in Proc. Thirteenth International Symposium on Quality Electronic Design (ISQED), Santa Clara, USA, pp 424-431.

[2] O. Coudert, "Timing and design closure in physical design flows," Proc. - Int. Symp. Qual. Electron. Des. ISQED, vol. 2002-January, pp. 511-516, 2002.

[3] C. J. Alpert et al., "Techniques for fast physical synthesis," Proc. IEEE, vol. 95, no. 3, pp. 573-599, 2007.

[4] Rakesh Chandra, J Bhaskar, Static timing analysis for nanometer designs, A practical approach.

[5] Synopsys Design Compiler user guide June 2015.

[6] Himanshu Bhatnagar, Advanced ASIC chip synthesis using Synopsys Design Compiler, Physical Compiler, and PrimeTime.

[7] Synopsys whitepaper PrimeTime Advanced OCV Technology.

[8] Ginovanni De Micheli, Synthesis and Optimizatioin of Digital circuits.

[9] Jason Cong, Flow Map : An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup Table Based FPGA Designs.

[10] LEF/DEF Language Reference, Cadence.