# Clock Driven Design Planning

*Shauki Elassaad*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2008-98
http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-98.html

August 15, 2008

**Clock Driven Design Planning**

by

Shauki Elassaad

B.S. (North Carolina State University) 1988
M.S. (North Carolina State University) 1991

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Robert K Brayton, Chair
Professor Jan M Rabaey
Professor Ilan Adler

Fall 2008

The dissertation of Shauki Elassaad is approved.

Chair Date

Date

Date

University of California, Berkeley

Fall 2008

Clock Driven Design Planning



by

Shauki Elassaad

**Abstract**

Clock Driven Design Planning

by

Shauki Elassaad

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Robert K Brayton, Chair

The phenomenal growth of the semiconductor industry has made the design of integrated circuits a daunting task. Due to the non-stop miniaturization of the design process, and the increase in the clock speed, the design of the clock network which is responsible of synchronizing the various design components has become very challenging. The ASIC design flow defers the design of the clock network until late in the design cycle when the cell placement is done and the locations of the registers are known. Up until that stage, all timing and power decisions that rely on the clock network are handled with estimates and ad-hoc methods. Worst case estimates of the clock timing makes convergence on the timing of high-end designs very difficult. In addition, power has become one of the most critical design constraints. Knowing that the clock network consumes more than 40% of the power budget, deferring the design and implementation of the clock until late in the design flow causes numerous design iterations and negatively affects the convergence and performance of the design.

In this work, we examine the design factors that affect the design, implementation, and optimization of ASIC designs. We focus on the design and implementation of

the power and clock networks as the two most important signals in the design, and study the various electrical and physical factors that affect their performance and reliability. We propose a new clock-driven design methodology that tackles the design of the clock network early in the design cycle before the implementation of the logic. The proposed methodology relies on designing a hybrid clock network that provides predictability and robustness to the design flow.

To break the reliance of the clock design on cell placement, we propose new algorithms that place the clock sinks (registers/latches) before the placement of the rest of the logic. Once the clock network is designed, laid out, and optimized, it drives the implementation of the entire design.

Nanometer designs operate under very tight design margins and constraints. Worst case decisions should be avoided whenever possible to successfully implement the design and meet its design targets. By tackling the clock network design early in the design cycle, detailed information about the timing characteristics and power consumption of the clock which is the biggest consumer of power in the design is known. This design approach facilitates relaxing the tight design budgets that are based on worst case decisions and enables a more successful physical synthesis of the design.

Professor Robert K Brayton
Dissertation Committee Chair

To my father and my mother whose unlimited love and great sacrifices made this endeavor possible.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

My advisor Professor Robert Brayton has my deepest gratitude; without his support, grace, and advice, this work would not be possible let alone completed. I have put off my Ph.D. aspirations for twelve years since I finished my masters degree in 1991 and joined the industry. In 2003, my Ph.D. journey started when I received an email from Prof. Brayton asking me "What is your next step?" shortly after leaving Cadence Berkeley Research Lab. Prof Brayton had already known of my interest in pursuing a Ph.D. degree. Although he encouraged me to contact a friend of his to pursue the Ph.D. in Europe where there are no course requirements and a faster path to completion, I am so happy that that path did not work, and I ended up in Berkeley in January of 2004. I am indebted to him for giving me the opportunity and indulging me with exciting, challenging and wonderful discussions. I will surely miss them.

I am also thankful to my friend and colleague Steve Meier who provided me with the support I needed to finish my research. I have known Steve since 1991 when he interviewed me for at job at Intel. We worked together there for a year before he left and joined Synopsys. Steve promptly answered my call when I needed his help. His support was instrumental in giving me access to software and industrial designs which allowed me to complete this work. Also, I like to thank the Synopsys IC Compiler team for welcoming and supporting me as I was finishing this work.

I am grateful to my Quals' committee members Professor Kurt Keutzer and Professor Jan Rabaey for answering the call and accepting to oversee my Quals' examination.

Special thanks go to Professor Ilan Adler who made Linear Programming such a fun topic for me. His passionate and engaging teaching style made me leave work and commute from the South Bay to Berkeley just to sit in his class. Also, I will

not forget how quickly he accepted my invitation to sit on my thesis committee even before I finished asking the question.

When I applied for the Ph.D. program, I called on Professor Robert Brodersen, Patrick Scaglia, and Luciano Lavagno to write recommendation letters on my behalf. I am very grateful to them for answering my call and supporting me in my endeavor.

I have been fortunate to be a member of Professor Brayton's research group. Although because of my work obligations, I could not spend much time on campus, however, I always looked forward to driving to Berkeley and meeting with them. The Friday research meetings were always engaging and fun. Special thanks go to Alan Mishchenko, Satrajit Chatterjee, Aaron Hurst, and Mike Case.

Also, I like to thank my office-mates Yanmei Li and Trevor Meyerowitz for their help in keeping me abreast of various issues in Cory, and for the fun times we spent together.

Last and not least, my gratitude to my friend Marwan Fawal for being there whenever I needed him, and for putting up with me especially when I was frustrated and mostly exhausted from juggling too many balls between work, family, and school. The numerous discussions over coffee related to work and research although used to often digress to parallel universes, they were always fun and sometimes thought-provoking.

# Chapter 1

# Introduction

Electronic Design Automation (EDA) has been the pillar on which the success and growth of the semiconductor industry have been based. Design challenges have increased with every new process generation. The commoditization of consumer electronics has brought tight and aggressive market challenges and constraints like time to market and time to dollars. All of these challenges can not be faced with a custom design methodology which requires a lot of resources and consumes a lot of time. Instead, an automated RTL-to-GDSII design flow that tries to satisfy the constraints outlined earlier has been devised.

However, at the time when the need and opportunity for an ASIC market were clear, the design challenges were not as numerous and complex. As it is with any industry and market, legacy forces become too strong to overcome. This has been the problem with today's RTL-to-GDSII design flow which is clinging to past successes and hobbles to tackle future problems.

In the late 1960s, Carver Mead said of then-fledging field of VLSI design "At that time...I needed to figure out (a) What kind of thing do you do with million transistor ICs? And (b) How would you ever get the design right?" Well, the second question–

How would you ever get the design right [with a million transistors]?–led Carver to his work on design methodology and tools. To quote Carver, "...even if I had a program that would wire together a bunch of logic gates...just getting a logic diagram with a million working parts is a big deal...and [a conventional approach] would preclude me from taking advantage of the things that made MOS technology so attractive, such as precharge, pull downs, and everything one could do if they know the environment in which the circuit lives."[29]

Carver's first IC design was a PLA-based finite-state machine, which is interesting in that he "invented" the circuit structure as an extension of his thinking about methodology (not knowing that TI and HP Fort Collins had invented the structure independently at about the same time), and because he generated it automatically from a program, using an "artwork language" Again, to quote Carver, "This was the first silicon compiler; it was great fun." [29]

Although the concept of 'Silicon Compiler' was born almost 40 years ago, we have not fully succeeded in engineering such a system as a result of a combination of factors, some technical and others are economic. In fact, a closer look at the latest design tool offerings from the EDA companies will reveal that the design methodologies they advocate contain many manual steps, and this shows that we are getting farther and not closer to the date where a true 'Silicon Compiler' can be engineered. A system which would abstract all the silicon engineering issues and complexities away from the system designer.

The objective is not to draw a bleak picture of the EDA industry or of the growth of the semiconductor industry. In fact, any observer of the semiconductor industry would acknowledge the phenomenal growth which it enjoys, and will see the enormous success that has been attained in both technology scaling and system integration.

On the EDA front, many accomplishments have been made in raising the ab-

straction level and enabling more automation to increase the productivity of system design. Advancements in the automated simulation of analog circuits and synthesis of digital circuits have had paramount impact on ASIC design styles. In addition, great accomplishments in the field of physical design automation and synthesis lead to great advancements in cell placement, routing, and optimization and resulted in orders-of-magnitude improvement in design productivity.

Having said this, I believe it is a shared view that the EDA industry is lagging behind the semiconductor industry. The EDA industry finds itself in this precarious position due in large to market forces as well as technology forces. The fast growth of the consumer electronics industry and the aggressive scaling of the semiconductor technology continuously create new problems and challenges for the EDA industry. In addition, the aggressive market forces that demand fast answers to complex problems force the EDA industry into an incremental mode and make it adopt a cautious approach towards adopting new design methodologies or change in the adopted design flows. Also, the time of two to three years needed to engineer and productize new products naturally make the EDA industry lag behind the semiconductor industry where a new process node is developed almost every two years. Thus, creative thinking and innovative methodologies are needed to break the barrier and enable true automation for the next multi-billion-transistor chips.

In 1991, when I started as a new college graduate at Intel working on parasitics extraction, resistance of the interconnect was almost negligible. In fact, the internally developed suite of tools did not contain any extraction technology for resistance of the interconnects. To estimate the resistance of some sensitive and critical signals, a designer had to manually count the squares on the chip plot to figure our the resistance value of the wire. Of course, things have changed dramatically since. In 1993-1994, we received a wake up call when we realized that the delay and noise models that we used were insufficient as was demonstrated by the testing results of the then-just-

fabricated Pentium chip. Today, the RTL-to-GDSII flow suffers from a combination of factors that include noise problems, timing convergence, power dissipation, and recently reliability issues due to the increasing effect of process variations both on-chip and off-chip.

Many of the design decisions that are made in today's RTL-to-GDSII methodology are based on coarse estimates or worst-case decisions. Such decisions can no longer lead to successful design due to the increased miniaturization of the process which in turn leads to tighter design margins, as well as the tight market constraints that demand a shorter turn-around design time. Today, high-end designs have on average a 1ns cycle time. Following the worst case decisions that are employed, a 10% budget is allocated for clock skew, and another 10% is allocated for clock jitter. Another 10-20% timing budget is allocated for derating the logic timing. This means 30-40% of the cycle time is lost due to these worst case decisions. Such an approach does not scale as designs get bigger and faster, and device features get smaller.

This dissertation is a modest effort at rocking the boat by proposing a new design methodology that could tackle the nanometer challenges more efficiently and proactively. In its modest contribution, it serves to raise awareness of how much legacy has factored in our adopted design approaches and techniques. This methodology does not solve all the problems that are plaguing IC design today. However, it lays the ground to tackling the two most difficult problems in the design, power and clock. Although the novel contribution presented here relates to the design of the clock, this accomplishment enables a more predictable and robust power design and thus carry to all aspects of performance and robustness of the rest of the logic. This is why we expound on both problems because we view them as very tightly coupled for a successful IC design methodology.

This dissertation is not a survey of the various EDA algorithms involved in the

RTL-to-GDSII flow, nor is it a survey of the various design methodologies and flows that are employed by designers in the field of high-end IC design. However, this work is about highlighting some of the flaws in the existing design methodologies and flows, in particular, those which are concerned with power planning and clock planning and presents a new design methodology and algorithms to tackle clock design.

Our proposed design methodology promotes the design of a predictable and robust clock network to drive the implementation of the rest of the logic in the design. Complex and high-end designs are suffering from an increase in the number of design iterations. To overcome this problem, our clock-driven design methodology strives to reduce the number of design iterations needed to converge on the design goals by eliminating the problems associated with clock and power design convergence.

To be able to construct the clock network early in the design cycle and achieve predictability and robustness, we need to design for and account for all factors that could negatively impact these design goals. Predictability in the clock design is achieved by devising a methodology and algorithms to tackle the important issues that could negatively affect the clock network early in the design cycle.

We need to deal with the following issues: 1) selection of a clock topology to meet our goals, 2) deciding on the number and size of the clock repeaters needed which are functions of the placement of the clock registers, 3) the problem of robustness of the clock network which is affected by the dependence of the clock network on nearby networks (coupling effects) and the power network (real supply voltage annotations at the inputs of the clock drivers), and 4) due to the dominance of interconnect delays, correlation problems between the global routing estimates and the detailed-routing could result in loss of predictability and failure in meeting the design constraints.

To tackle the first issue, we propose the design of a hybrid clock network early in the design cycle. This hybrid clock network is composed of a global part and a local

part. Any predictable and robust top-level clock network can serve our purpose. As a proof of concept, we present results based on a design of a hybrid clock network that is composed of a tree driving a mesh. We chose a mesh topology to improve the robustness of the design and increase the tolerance to process variations. For the second issue, we show algorithms to place the clock registers early in the design cycle and have these registers drive the implementation of the design. By placing the registers early in the design cycle, the design, implementation, and optimization of the clock network can proceed. Once the clock network is implemented, subsequent physical synthesis and routing steps will not impact it. This will ensure that all the characteristics of the clock network that were simulated and annotated will not change and will remain valid as we proceed further in the design cycle. Thus predictability is achieved.

The third issue of robustness has to do with the design of the clock network while taking into account all the noise sources that could negatively impact its reliability and robustness. Noise-induced power fluctuations are the major cause that negatively affect the robustness of the clock network. To get these fluctuations under control, we advocate a holistic approach to the design of both power and clock networks. By not viewing the design of both networks as a chicken-and-egg problem, we improve the predictability and robustness of the design. Clock shielding can mitigate other noise sources induced on the clock network as is commonly done in high-end designs today.

In regards to power planning, virtually all power planning of ICs is done manually and rely on worst case estimates to synthesize the power grid. Even the worst-case estimates sometimes fail to predict the real needs of the design and entail many design iterations to close on reliability and timing issues. For the most part, the tasks involved in designing the power network on the chip such as current-estimation, topology selection, topology optimization, and simulation involve heuristics and esti-

mates that do not guarantee 100% confidence in the generated results. In addition, the clock which is the biggest consumer of power in the design (more than 40% of dynamic power of design is consumed by the clock network) is not constructed until later in the design flow after the power network is designed and the logic cells are placed.

As for the fourth issue regarding the correlation issues between global routing and detailed routing, the problem from the clock's point of view is resolved by the fact that we design and implement the clock network before the implementation of the rest of the logic. Thus, the analysis and optimization of the clock network are based on detailed routing of the clock signal.

Hence, it is our view that a holistic design approach to the two most important signals in the design - power and clock - is needed. The existing design flow lacks such a holistic view of the problem due in part to the increasing complexity issues surrounding these networks, but also due to legacy issues that continue to plague the existing design flows and methodologies.

The thesis starts in Chapter 1 by introducing a high-level description of the synchronous design paradigm . In Chapter 2, we discuss Design Planning and highlight the flaws in the existing design flows and tools. The treatment of the Design Planning task is done at two levels. The first is a superficial level to introduce the reader to the different tasks involved in the implementation of ICs. This helps in forming a coherent and almost complete picture of the problem. However, the issues that directly affect the clock network design are introduced and treated with more rigor. In Chapter 3, we discuss the issues related to power design & planning. In Chapter 4, the problem of clock design & planning is introduced. In-depth treatment of this subject is presented to highlight the difficulty of designing a robust and high-performance clock network. Our proposed clock-driven design planning methodology (CDP) proposes

the use of a robust hybrid clock network to facilitate the design of the power network simultaneously. Although the problem of designing a robust power network is not discussed in depth, the tight coupling between the two problems is highlighted. In Chapter 5, we present an in-depth discussion of the design and analysis of the clock mesh which is part of our proposed hybrid clock network. In Chapter 6, a novel clock-driven design methodology is discussed. This methodology tackles the clock network design as an integral part of the RTL-to-GDSII design flow as opposed to it being left as an after-thought. In Chapter 7, we discuss the design of experiments to validate the proposed methodology and show experimental results of the CDP methodology as compared to traditional clock-tree synthesis (CTS) and custom clock mesh designs (CMS). In Chapter 8, we conclude with a summary and highlight our contributions.

# Chapter 2

# Synchronous Design

## 2.1 Principles of Synchronous Design

The phenomenal growth of the semiconductor industry has made the design of integrated circuits a daunting task. Synchronous digital systems is a popular design paradigm due to the ease and clarity of data communication between the different components of the system under design. However, due to the ever-increasing miniaturization of the design process, the high degree of integration which dramatically increased the number of clock elements, and the increase in the clock speed, the design of the *clock network* which is responsible for synchronization amongst the various design components has become very challenging.

The vast majority of integrated circuits adopt a *synchronous* design methodology. Today, these systems contain billions of transistors which are broken into multiple components consisting of memory blocks, analog and mixed signal blocks, custom digital blocks, sequential elements, and combinational logic cells. In synchronous design, a special signal called *clock* governs all operations and data transfers. The system functions properly when the clock is distributed over the design and arrives at

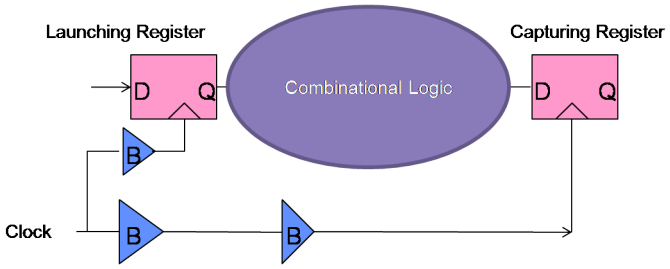



Figure 2.1. A combinational logic path between two registers

the sequential elements ideally at the same time. Although this is not possible due a variety of design and technological reasons, the objective of the design is to distribute the clock to all sequential elements with the smallest delay possible.

In this thesis, we make no distinction between a clock cell, a sequential element, a latch, or a register. A latch is a level-sensitive clock element while a register is an edge-sensitive clock element. However, for the purpose of our discussion in this thesis, they all mean the same thing.

The objective of clock design is to distribute the clock signal to all sequential elements in a reliable fashion and with the minimum variation in the arrival times of the clock signal at the inputs of the sequential elements. In addition to reducing the variations in the arrival times of the clock signal at the input of the clock elements, the goal is also to reduce clock power which has become prohibitive and consumes more than 40% of the power budget of the design. Also, constraints related to signal slew times at the different points on the clock network are present to ensure functionality and reduce short-circuit power of the clock buffers and registers.

In general, there is more than one clock in the design. Typically, all the clocks are derived from a system clock which is generated by a phase-locked-loop (PLL) circuit. The PLL synthesizes the clock based on a reference clock which is usually presented as an input to the system. The design of the PLL, the clock distribution network, and

the sequential elements play a key role in determining the performance and power consumption of the system.

To ensure proper operation, design margins are set to safe-guard the clock assertions. These safe-guards are known as setup and hold times (Figure 2.2). To improve the performance of the design, the goal of clock design is reduce the impact that the clock timing has on the data path timing. In doing so, proper care should be given to designing a high-performance and robust clock network that is tolerant of systematic and random process variations.

The cycle time of the design is a sum of a nominal part ($\omega_0$) and a variational part (Equation 2.1). For correct functionality of the design, minimal variations in the arrival of the clocks at the sequential elements must be achieved. There are two kinds of variations that affect a clock signal: spatial and temporal. Spacial variations known as *skew*, affect the arrival of the various clock edges at their sequential elements within a clock cycle ($\phi_{skew}$). Temporal variations known as *jitter* affect the arrival time of the clock edges at their sequential elements across clock cycles ($\phi_{jitter}$).

$$\omega(t) = \omega_0(t) + \phi_{skew} + \phi_{jitter} \tag{2.1}$$

Given that the speed of the system is decided by the slowest path between any pair of launching and capturing sequential elements (Figure 2.1), both variation sources ($\phi_{skew}$ and $\phi_{jitter}$) reduce the effective cycle time and thus reduce the performance of the design. A digital system is synchronous if all data transactions are executed following the arrival of clock edges. No data is captured at a receiving clock cell unless a clock has arrived first. Such a system is different from an asynchronous one where data transactions can take place at any time and are governed by a handshaking protocol to indicate launching and capturing of data.

The simplicity and clarity in data communication outlined above makes syn-

chronous design a popular approach. Since the launch and capture of data is only defined after a clock toggles and reaches the sequential element, a designer needs not worry about what happens at other times due to transient signals or glitches. However, designing a clock network that meets the timing constraints of nanometer designs has become a formidable task due to the dominance of interconnect delays in today's and future designs. We present below some definitions that are needed to understand the operation and design of the clock network:

- Data Transfer: In synchronous designs, data moves from a launching sequential element to a capturing sequential element following the arrival of the clock signal at the input of the registers (Figure 2.1).
- Clock Distribution: Clock is transmitted to the different parts of the design via a distribution network composed of cells (buffers, inverters, ...) and wires. Proper design of this network is essential to achieving correctness of the design and meeting its performance targets.
- Propagation Delay: The delay from the 50% point of the input waveform to 50% point of the output waveform.
- Rise/Fall Time: Typically defined to be the delay from the 10% point to the 90% point of the waveform. Different timing libraries can have their own thresholds (trip points).

### 2.1.1 Clock Assertions

The design cycle time is decided by a number of factors that limit its performance. Equation 2.2 shows the computation of the minimum cycle time that can be used to ensure proper functionality of the design.

$$CycleTime \geq t_{pd} + t_{skew} + t_{setup} + t_{clk-Q} \tag{2.2}$$

.

where $t_{pd}$ is the longest path delay through the combinational logic, $t_{skew}$ is the clock skew, $t_{setup}$ is the setup time of the synchronizing element, and $t_{clk-Q}$ is the delay inside the synchronizing element (Clk $\rightarrow$ Q).

For correctness of the data transfer, clock assertions are imposed on the design to ensure proper functionality. These assertions are timing constraints that relate the arrival and departure of data at the registers to the arrival of the clock signal in every cycle. All data communication in the design has to abide by these assertions. Exceptions are made to this rule by defining multi-cycle paths and other timing exceptions to enable the completion of some operation that might require more than one cycle. Clock assertions include setup time, hold time, latency, skew, and uncertainty.

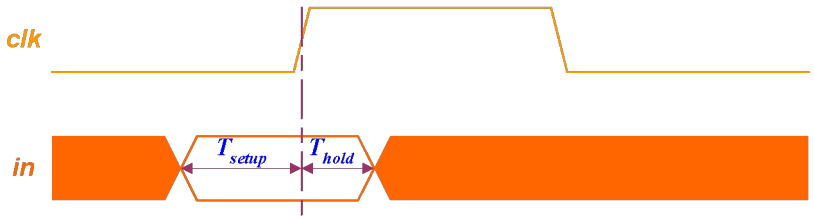



Figure 2.2. Clock Setup and Hold Time Specification

- *Setup Time* is the minimum time needed such that data is ready at input of

the latch (capturing latch) prior to the arrival of the clock signal at that latch (Figure 2.2).

- *Hold Time* is the minimum permissible time for the data to stay stable after the arrival of the clock edge to guarantee proper capturing of the data (Figure 2.2).

- *Clock Skew* is the maximum spatial variation in the arrival times of the clock edges at the inputs of the clock cells as defined in Equation 2.3. In Equation 2.3, $t_i$ and $t_j$ are the arrival times of the clock at any two sinks in the clock network (Figure 2.3).

$$t_{skew} = max_{ij} |t_i - t_j| \tag{2.3}$$

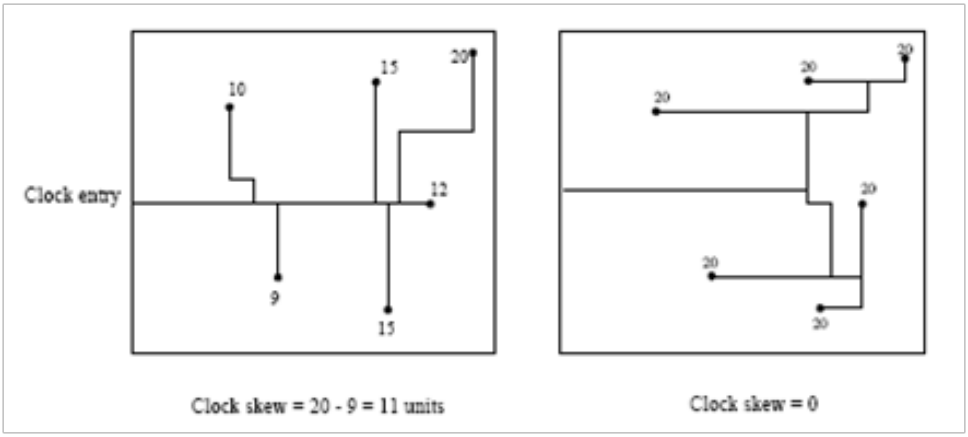



Figure 2.3. Skew in Clock Networks

- *Clock Latency* is the latest arrival time of the clock signal at the input of the clock cells.

- *Clock Slew* is typically defined as the transition time between 10% and 90% of the supply voltage.

- *Clock Uncertainty* refers to the variation in the delay of the clock signal. There are two kinds of uncertainty measures affecting the clock signal. Spatial uncertainty refers to the spatial variation in the delay of the clock signal as it arrives at two different clock elements in the same clock cycle (skew). Temporal uncertainty refers to the variation in the arrival of the clock signal edge at the input of a particular clock register (jitter). Figure 2.4 highlights the main factors that are responsible of skew and jitter in the design.

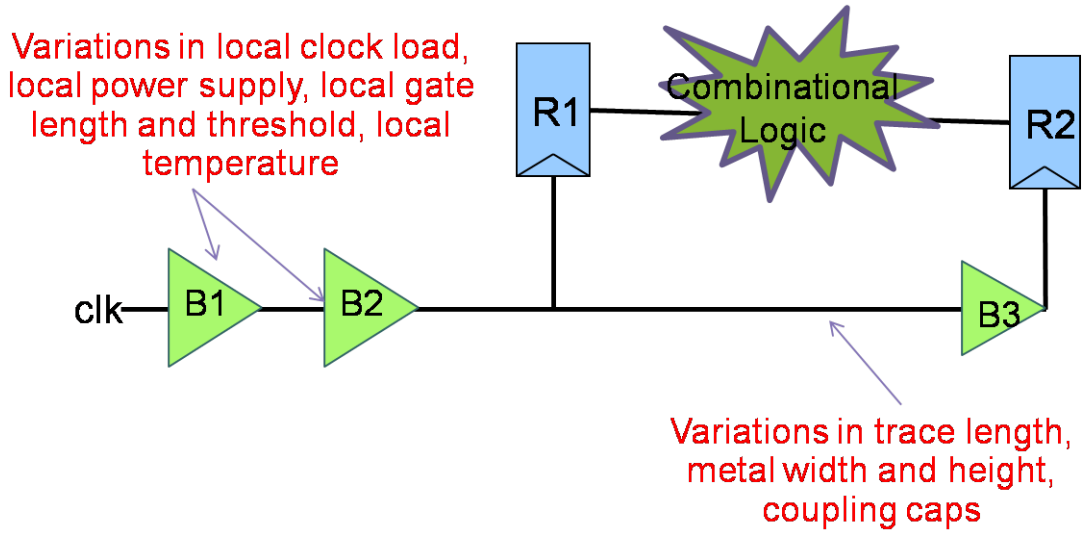



Figure 2.4. Skew and Jitter in Clock Networks. The main design factors that affect both skew and jitter are highlighted in this figure.

For proper operation of the design, the following two timing checks must be honored:

- *Slow Path Constraint* ensures that $T_{cycle} \geq T_{Clk-Q_{max}} + T_{PD_{max}} + T_{setup}$
- *Fast Path Constraint* ensures that $T_{Clk-Q_{min}} + T_{PD_{min}} \geq T_{hold}$

## 2.2 Clock Domains

Clock domain refers to a group of clocks that run at different speeds and are derived from a reference clock. These clocks still need to be balanced and synchronized against each other. Having multiple clocks in the same domain allows part of the logic in the domain to run at slower speeds than others to reduce power consumption.

There could be multiple clock domains in the design. Different blocks can be controlled by different clocks which are not in sync with each other. For proper operation of the design, synchronization or handshaking elements are added to ensure that the timings of the different blocks are honored (Figure 2.5).

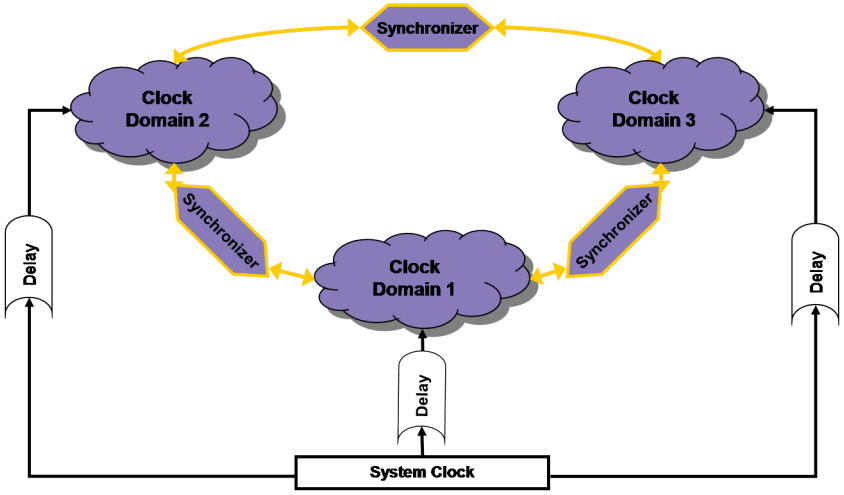



Figure 2.5. This figure shows multiple clock domains in a design. Each clock domain may contain one or more clocks that need to be synchronized with each other. Clocks across multiple clock domains do not need to be balanced against each other. Synchronizers are inserted to allow clocks across different domains to communicate with each other.

## 2.3 Sources of Clock Variations

There are multiple sources of variations that impact the clock timing. The clock generation circuitry (PLL) can be affected by a number of variation sources such

as power supply fluctuations, temperature, and device mismatches . In addition, any variation that impacts a logic cell or wire in the design could impact the clock distribution network as well. Due to the criticality and sensitivity of the clock signal, special care must be made to ensure a robust design of the clock network. Process variations can occur in the die (on-chip variations), die to die, wafer to wafer, or lot to lot. On chip process variations that affect the delay of the clock buffers as well as interconnect delays need to be controlled and designed for. Variations in gate-length, oxide thickness ($T_{ox}$), inter-layer dielectric (ILD), and doping concentrations are examples of variations that could severely affect the performance of the clock network.

Furthermore, process variations can be divided into two categories: systematic and random variations. The systematic variations are deterministic in nature, and proper design and control can help mitigate or eliminate their impact. Random variations are harder to control, and proper guard bands must be adopted to ensure correct functionality of the design.

Also, design environment changes can affect the clock and design performance. Changes in temperature, supply voltage, or capacitive coupling between signals are dynamic in nature and must be accounted for during the design. If these factors are not accounted or guarded against during the design of the clock network, performance of the design will be affected at best, and the design could fail to function correctly at worst [28], [4], [1].

To honor the design constraints imposed on the clock signal as presented above, proper design and planning of the clock network is needed. The clock signal is the biggest consumer of power in the design. In addition, the high-speed switching of the clock injects a lot of noise into the power network. Any fluctuations in the power supply will impact the delays of all circuitry in the design, and hence proper design

of the power network is needed as well. In Chapter 3 will present some of the issues and techniques that are related to the design and optimization of the power network and how they impact the performance and reliability of the clock network.

# Chapter 3

# ASIC Design Planning

## 3.1 Introduction

In this chapter, we discuss the design and planning of integrated circuits. Although we focus on issues related to ASIC design styles, most of our discussion is applicable to microprocessors and other custom design methodologies.

In order to assess the impact of layout related issues on power and clock designs, we need to understand the different design steps that shape the layout and optimize its performance. For example, design partitioning and block placement play prominent roles in deciding the difficulty of balancing the top-level clock network. Hence, this discussion is very relevant to our proposed clock-driven design methodology so that we understand the impact of our design steps on the performance and reliability of the rest of the design.

Design planning was not a necessity in previous generations of process technologies due to two reasons: (1) the size of the design in terms of the number of gates was reasonable, and (2) the performance targets were modest. In current and future process technologies, two forces made design planning a necessity: the exponential

growth of the number of transistors that can be packed on a die, and the aggressive and tight design constraints and market forces.

There are two kinds of digital design styles: custom and structured. Each design style has its own design methodologies and goals. Custom designs are typically used for high-end microprocessors, high-end graphics, and communication designs. Such designs tend to be at the leading edge of process technology and performance objectives. In these kinds of designs, designers employ many manual steps in the design flow to be able to manage the complexity and meet the design goals. In addition, these designs tend to usually have a sizable budget and abundant resources as compared to ASIC designs.

The other design style is that of the Application Specific Integrated Circuits (ASIC) market. These designs are typically (but not necessarily) based on the previous process-generation technology, and their design goals are not as aggressive. Further, this design style relies on automation for almost all design steps. In addition, the size of these chips tend to be less aggressive than the custom market. However, the market pressures on these designs tend to be higher due to the market segments that they address. Due to the commodotization of the consumer electronics industry, the time-to-market and time-to-money forces are huge. In addition, such designs usually do not have the kind of engineering resources and budget that the large custom designs have. With the phenomenal growth of the mobile application industry and the advent of nanometer technologies, the constraints on ASIC designs are changing, and the market pressure is tightening. In particular, power constraints on ASICs designed for the mobile industry are much more severe than those for the custom high-end designs.

In this work, our goal is to improve the ASIC design methodology on two fronts:

(1) provide early predictability in the design cycle, and (2) shorten the time-to-market of these designs. In doing so, a lot has been learned from the custom design paradigm.

This chapter discusses the RTL-to-GDSII flow from a high level point of view. The objective is to introduce the different steps that make up the flow so that the general problems of clock and power designs are put in perspective. This discussion is not meant to be exhaustive, but the main steps in the flow are highlighted and more details are given to the steps that influence the design of the power and clock distribution networks. A lot of emphasis is given to the section on Design Planning. This is done to highlight the importance of planning in order to cope with the advent of the billion-transistor chips.

There are a number of design steps that are done as part of the 'Chip Finishing' phase, and they are not discussed in this chapter since they are not relevant to the discussion at hand. Also, issues related to design-for-manufacturability (DFM) and design-for-testability (DFT) are not addressed here.

## 3.2 ASIC RTL-to-GDSII Flow

Once a design has been through the architecture, micro-architecture, and the RTL stages, the design is passed on to the logic synthesis stage. In this stage, high-level description of the design (RTL) is synthesized and mapped to a target design library which captures the manufacturing technology. Traditionally, very little is done to capture the impact of the back-end on the synthesis process. However, this topic has been the subject of extensive research and development, and more success is seen in this regard. Figure 3.1 shows the various design steps that constitute the RTL-to-GDSII flow. After logic synthesis is done, the design goes through the physical design stage. In physical design, a set of transformations is carried out that transforms

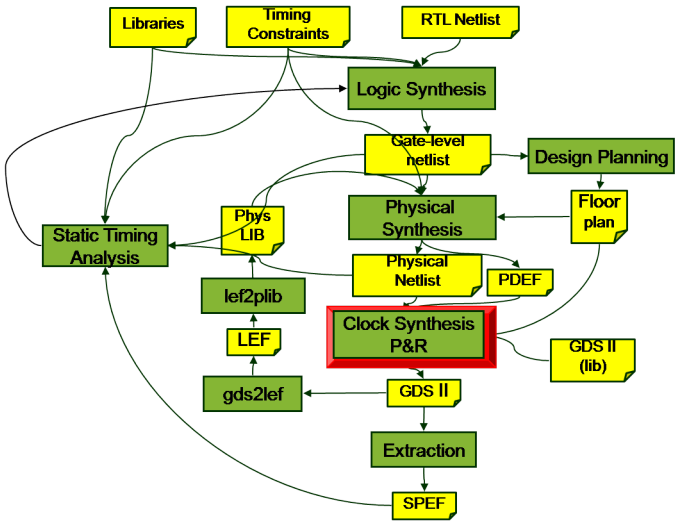



Figure 3.1. This figure shows the different design steps that are carried out in the RTL-to-GDSII design flow.

the logic netlist into a set of polygons. For efficient and high-quality results, the physical design is divided into a number of tasks to facilitate the transformations and manage the complexity of the design. The physical design stages are Design Planning, Block Placement, Optimization, Routing, Timing Sign-off, Physical Verification, and Chip-Finishing.

Our objective is to present a new design methodology which covers some of the shortcomings of the existing methodologies and to improve the QoR of the designs as they move through the different stages of the flow. It is not our objective to present the various algorithms used at the various stages except if needed to highlight a deficiency or an improvement in the produced results. Thus, we will not delve in details on the various placement, routing, extraction, timing analysis, and other areas. Most of these problems are known to be NP-complete, and thus, the literature is full of heuristics and algorithms to attack them.

To highlight the work on Clock-driven Design Planning, we will delve into some details on the placement and clock synthesis problems. As for other discussions, it

will be more from an applied point of view to present the design problem and the various metrics that are tackled by the different stages of the physical synthesis flow.

## 3.3 Design Planning

Design planning was not a concern when designs were relatively small (less than one million placeable components). The implementation of those designs relied on a flat design methodology where the whole design was viewed as one entity. However, as the level of integration increased and multi-million cell designs started to appear, these designs exceeded the capacity of a flat design flow, and convergence issues became more severe. At that point, design planning became a mandatory step for a successful and efficient implementation of the designs.

Design planning here means the necessary design steps that are needed to manage the implementation and verification of the various components of the design. To manage complexity, a design planning system is responsible of partitioning the design into a number of components/blocks such that each can be designed and optimized independently. The top-level design constraints are partitioned and mapped onto the blocks to ensure that the over all design meets its design targets. Once each block is designed, design planning is responsible of the the necessary steps to integrate these blocks and ensure that the design goals are met.

Design planning (Figure 3.2) consists of Macro Planning, Partitioning & Global Placement, Power Planning, Top-level Routing (pre-routes), Constraints Management, and Top-level Clock Planning.

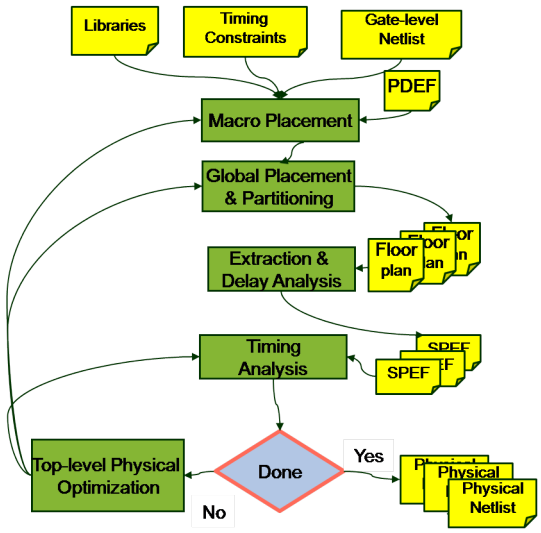



Figure 3.2. This figure shows the different design steps that are carried out and the data produced at every step in the ASIC Design Planning flow.

### 3.3.1 Macro Planning

Almost every design contains some IP blocks. IP blocks could come in three different forms: soft, firm, and hard. Soft IPs are typically RTL designs with their verification counterparts. These IPs are usually synthesized with the rest of the logic in the design and are handled just like any other HDL module. Firm IPs are those that consist of a synthesized netlist (logic netlist), and are usually placed so that the module can be characterized with respect to timing and power. Such a module is represented as a soft macro in the physical design world, and could be placed manually or automatically in the floorplanning stage. Hard IPs are usually the most common form of IPs. These IPs form the memory blocks, analog, RF, and other custom circuitry. Most often, the designer is responsible of the placement of these IPs because of their dependence on their outside connectivity (off-chip buses), or because of the sensitivity of their circuits as in the analog/RF case. Such blocks are usually very sensitive and require special attention when placing them and routing

over or near by them. For the most part, traditional placement engines do not do a good job of placing the top-level macros automatically. However, better automated placement can be attained by relying on some hints provided by the user that can specify the side of the die where the macros should reside, or some form of clustering which serves to simplify the placement job and improve the QoR.

### 3.3.2 Partitioning

The number of devices on a single die is increasing rapidly due to the continuing shrinking of the process technology. Today, billion-transistor systems have become a reality. Such complexity necessitates a divide-and-conquer approach to manage the design process. Partitioning plays a key role in attaining the design goals in an acceptable turn-around time. However, due to the tight design constraints present, partitioning becomes a formidable task.

Partitioning the top-level constraints amongst the different blocks is a formidable task by itself since it entails performing budgeting of the top-level constraints amongst the various blocks. Since the partitions have not been implemented yet, it is hard to estimate the performance and area of these partitions. This makes deciding on accurate timing budgets early in the design planning stage a difficult task.

To avoid all the issues mentioned above, designers tend to opt for a flat implementation of the design whenever possible. However, due to the large size of ICs today, the implementation algorithms are not scaling at a comparable rate to the aggressive levels of design integration. This fact forces designers to engage in design planning and carry out the partitioning step to be able to manage and design the partitions concurrently and independently. Typically, partitioning and global placement go hand-in-hand. To produce good quality partitions, the top-level connectivity (global routing) of the partitions has to be taken into account. Most of the global

placement algorithms have some form of partitioning and global routing embeded in them to accomplish this task.

### 3.3.3 Power Planning

Power integrity is an important factor to any successful design. Power plays a key role in achieving the speed target set for the design. In addition, it plays a key role in the reliability and proper functionality of the design. Chapter 4 presents a detailed discussion on this topic.

### 3.3.4 Multi-voltage Domain Planning

To manage the power consumption of the design, the logic is separated into multiple voltage domains. This classification aids in trading performance for power. By utilizing multiple voltage domains, [32] reports savings of 16% of active power and 50% of standby power in one telecommunication chip. Voltage domains can be implemented as coarse or fine-grained islands. The coarse voltage islands are implemented as separate clusters on the chip with its own power grid (Figure 3.3), while the fine-grained implementation mixes between the high voltage and low voltage logic by employing inter-digitated power grids for the different supply levels.

### 3.3.5 Constraints Management

Constraints management is necessary to map the constraints between the RTL netlist and the physical netlist. The constraints reflect the logic hierarchy produced by logic synthesis. Though it may make sense to synthesize a module that has few thousand gates independently, it makes no sense to physically implement such a module. After logic synthesis, the netlist often gets re-partitioned. The physical

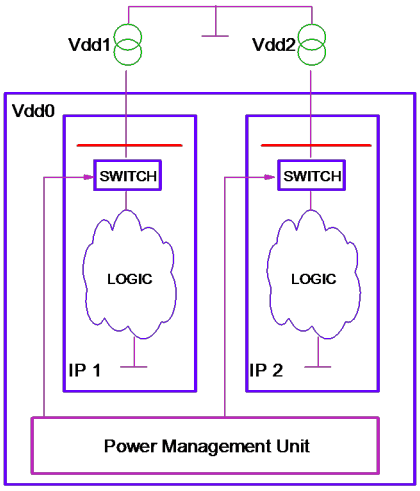



Figure 3.3. This figure shows the physical implementation of coarse voltage islands in a design [32]. In this case, the voltage islands are physically separated, and level shifters are introduced to pass data across. When different blocks in the design can be optimized to run at different speeds to save power, the supply voltages are scaled accordingly.

partitions could be very different than those of the logic hierarchy. Thus, mapping of the RTL or synthesis constraints is needed to drive the physical implementation of the design. The constraints mapping is done based on the floorplan of the design (global placement and global routing). This means that the constraints mapping process may not reflect the final implementation of the design. In fact, this could be one source of potential convergence problems in the design implementation.

Since the constraints mapping may produce too tight (infeasible) constraints for some partitions while other partitions enjoy relaxed constraints, the implementation could be driven by the wrong constraints. This could lead to long turn-around times in implementing some blocks, and difficulty in converging on the global floorplan when the blocks are stitched together and sign-off analysis is carried out.

### 3.3.6 Top-level Global Placement and Routing

Global or coarse placement of the logic cells and macros is needed for accurate estimates of routing resources and timing information. Performance-driven planning requires estimates of the cell and interconnect delays. The delay estimates of the interconnects can only be trusted if the route estimates reflect the detailed routing results at the end of the implementation stage. The objective of global placement and global routing is to build a reasonable picture of the die's floorplan as reflected by the planning of the macros and the logic partitions. At this stage, cell overlap is allowed since we are only interested in the big picture of where cells are in the design. Global placement is necessary to create partitions that facilitate the implementation of the design in an efficient turn-around time, and to meet the timing constraints. Global routing is responsible of providing an accurate picture of the congestion of the partitions and of top-level floorplan. In addition, global routing is necessary for good estimates of the interconnect delays. Extraction and delay calculation engines rely on the global routes to synthesize reasonably accurate electrical networks for the wires and compute their delays.

One of the critical design steps for a successful implementation of large and complex designs is *global signal planning*. Although the global signals are a small percentage of the total signals in the design (<10%), they are usually the most troublesome to close on their timing. Global signals are inter-block nets that exhibit usually long wire length. This means they have long delays, and special optimization and planning are needed to manage the timing of these signals. Top-level repeaters are inserted in the design to reduce signal delays. For high frequency designs, it has been reported that at 70nm process node, designs could need in the order of 700K repeaters to optimize timing and reach closure. The design and planning of these repeaters become a serious challenge. In some designs, uniform planning of repeaters is done such that

dedicated areas in the chip are reserved for them (repeater banks). Alternatively, repeater planning can be done as part of the global signal routing and optimization. In either case, careful planning of top-level repeaters is needed to minimize signal timing without significantly affecting the area and congestion of the design. Figure 3.4 shows the scaling of the signal delay with respect to the scaling of the process technology. It is clear that signal planning problems will become more severe and require careful handling to achieve design convergence.

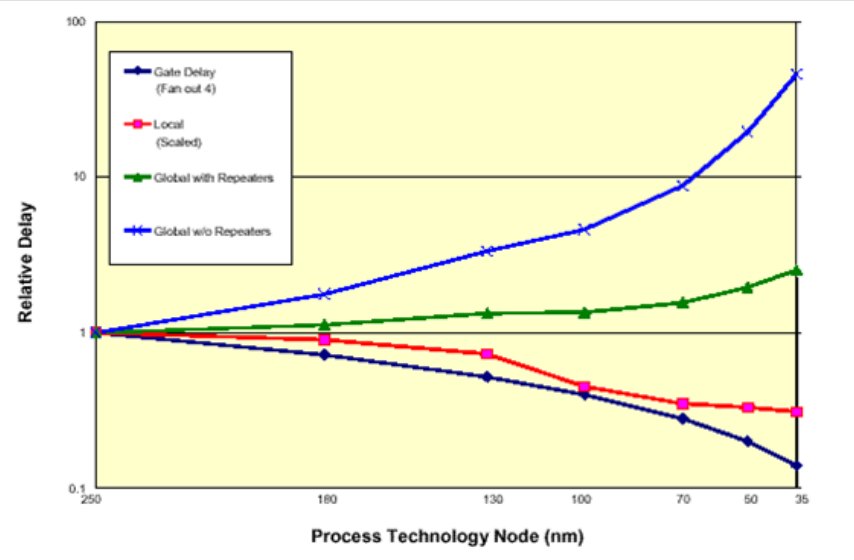



Figure 3.4. This is the ITRS 2003 Road map on Interconnects. This figures shows how signal delay scales with the process technology. It is clear from this figure that insertion of repeaters is mandatory to optimize the global signals and manage their delays.

At the design planning stage, when the design is partitioned and globally placed, not all block timing problems can be resolved. Typically, the design planning keeps iterating on macro placement, partitioning and and global placement and routing until the block's timing is within 20% of the timing constraints. At that point, design planning is done and the design moves into the physical synthesis stage where each block is placed, optimized, and routed independently. Physical synthesis is not

complete until all timing constraints are honored. Failure to do so means that the planning of the design is not done properly, and a costly design iteration is needed.

### 3.3.7 Clock Planning

In a hierarchical implementation of the design, each partition will have its own clock root and clock network. One way of creating the clock root for each partition/block is to synthesize the entire clock network in a flat fashion, and then use the synthesized network to punch clock ports in the design macros. In addition, the produced clock network will provide information on the latency and skew for each macro. This information can be used to drive the placement and optimization of the different partitions concurrently.

Since the flat implementation of the clock relied on a preliminary placement of the logic cells in the blocks, it is obvious that such a design flow does not provide guarantees of convergence. None the less, for hierarchical design implementation, typically this is the adopted flow. It is this lack of convergence guarantees and the complexity of the flow that make designers lean towards a flat implementation whenever such an approach is feasible.

The flow mentioned above not only does not guarantee a convergent clock network design, but also it does not guarantee timing convergence for the whole design. When implementing each partition, the timing analysis is done with ideal clocks whose latency, skew, and jitter/uncertainty values are estimated based on the global clock planning. Since neither the placement nor the routing on which the clock plan relied on reflect the final detailed implementation of the design, decisions made during clock design may be inaccurate and could result in lack of convergence.

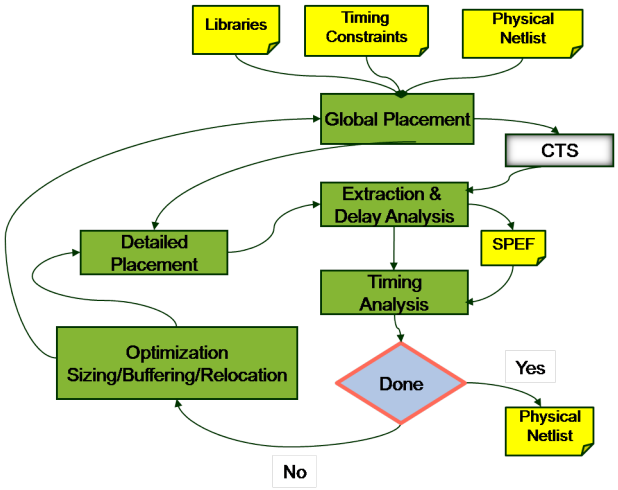



Figure 3.5. This figure shows the design steps that are carried out in the Physical Synthesis flow. Most of the design steps are carried out before detailed routing of the design. In nanometer designs, the need to do further optimization in the post detailed routing stage has become necessary.

## 3.4 Physical Synthesis

This phase is concerned with placement, routing, and optimization of a block in the design. The block could be the entire design, or it could be one partition in the design. Figure 3.5 shows a typical physical synthesis flow. After the design planning stage, a set of macros is generated each with its own set of design constraints. As mentioned above, these constraints are either the result of the top-level constraints that were partitioned across the different partitions, or they are a result of the design planning stage to guarantee correctness when the design is put together after physical implementation is done.

### 3.4.1 Placement

This step is tasked with placing the cells (placeable components) legally such that there is no cell overlap and congestion is minimized. The objective of the placement is to reduce area, wire-length, and improve timing. This is a sizable task given the

complexity of today's designs and the aggressive performance targets that are desired. It has been shown that Cell Placement is an NP-hard problem, and thus heuristics are employed to efficiently place the cells while optimizing the objectives mentioned earlier.

Two formulation of placements are prevalent, analytical and constructive. Analytical placement mathematically formulates the problem and solves it using numerical optimization algorithms. Early on, simulated annealing was the algorithm of choice to carry out the placement step (TimberWolfe). Since then, the most ubiquitous formulation is based on quadratic formulation [27]. In the constructive placement methods, an iterative clustering and spreading algorithms and heuristics are employed to achieve the placement objectives.

Placement is divided into two parts, global placement and detailed placement.

### Global Placement

In global placement, the objective is to distribute the cells over the die-area in such a fashion that the global design objectives (timing, wire-length) are attained. It is permissible to have overlaps amongst the cells. At this stage, the objective is to be able to compile some estimates of the die-area, wire-length, and timing violation. If the produced estimates are not satisfactory, better partitioning, design guides, and re-planning the IO signals or the hard macros are carried out to improve the results of the global placement.

### Detailed Placement

In detail placement, the cells which are clustered together or are on top of each other as a result of the previous step are spread and re-ordered. The objective is to produce a legal placement (no overlaps), minimize congestion, and improve wire-

length. Again different design constraints can be imposed on the detail placer so that the legalization step does not wreak havoc in the timing of the design.

### 3.4.2 Optimization

There is a number of knobs that can be exploited in the physical optimization stage. Unfortunately, due to the size and complexity of the task at hand, most of the physical optimization algorithms are iterative in nature. The basic and most important operations at this stage are:

- *Gate Sizing*: To improve timing of a path, sizing up some of the gates on the path will reduce the cell delay (increase power) as well as the over-all path delay. Equation 3.1 shows the dependence of the gate delay on the voltage swing, the capacitance load at the output, and on the drain current. To reduce the power consumption of the design, gates on logic paths that have a positive slack can be sized down to reduce their power consumption without violating the timing constraint.

Figure 3.6. Propagation delay based on the alpha-power model [38]

$$t_{pHL}, t_{pLH} = \left(\frac{1}{2} - \frac{1 - v_T}{1 + \alpha}\right) t_\tau + \frac{C_L * V_{DD}}{2 I_{DO}} \tag{3.1}$$

where $\alpha$ is the velocity saturation index, $v_T = \frac{V_{th}}{V_{DD}}$, $C_L$ is the output capacitance of the device, $I_{DO}$ is the drain current of the device at ($V_{GS} = V_{DS} = V_{DD}$).

- *Gate Relocation*: When interconnect delays dominate, changing the physical topology of the net/path is needed to achieve the desired improvement. Relocating a gate could reduce the wire length and hence reduce the wire delay as well as the load seen by the driver on the net. This will reduce the net delay, improve timing, and could reduce congestion.

- *Repeater Insertion*: The delays of wires grow quadratically with the their length (Equation 3.2). To improve the timing of a signal, repeaters are inserted to reduce the delay and make it grow linearly with the number of wire segments. Repeaters can be either inverters or repeaters that are inserted at regular intervals (Figure 3.7).

$$\tau = \frac{rcL^2}{2} \tag{3.2}$$

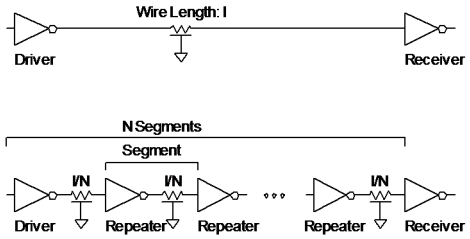



Figure 3.7. Repeaters inserted on long wires to reduce signal delays.

- *Layer Assignment*: By assigning critical signals to top-level metals which tend to be wider and thicker, signal delays are improved.

- *Shield Insertion*: Reducing the capacitive coupling of critical nets is needed to meet their timing constraints. For sensitive signals, shields are inserted to reduce signal delays at the expense of added area (Figure 3.8).

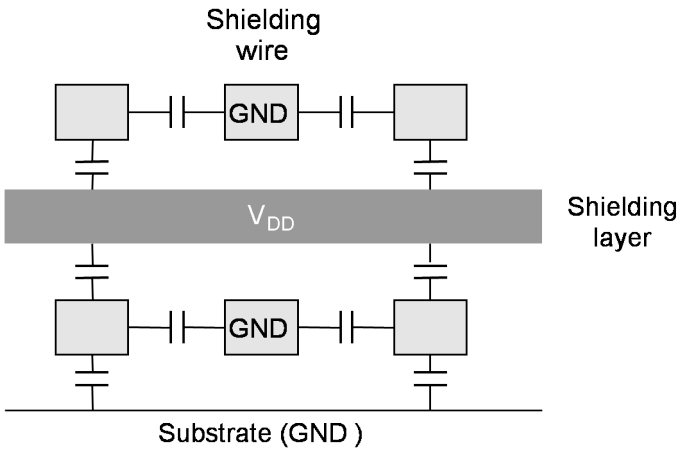



Figure 3.8. Shield insertion to reducing capacitive coupling between neighboring signals [33].

- *Latch Clustering* reduces the load capacitance seen by the local clock drivers. Since most of the clock power is in the last stage of the clock network, reducing the load capacitance plays a significant role in reducing the clock dynamic power of 25% or more as reported by [32].

- *Power Gating* optimizes the power consumption of the design by inserting switches (sleep transistors) to shut off the power from inactive circuits. Since leakage power is a major concern in 90nm and below, shutting off the power from inactive circuits significantly reduces the leakage power (>20X) with virtually no performance penalty and up to 5% area penalty [32].

- *Multiple Supply Voltages* optimize the power consumption by running different functional blocks in the design at different supply voltages.

- *Multiple Threshold Voltages* optimize the leakage power in the design or functional block while still honoring the performance constraints.

- *Low Power Mapping* gives the physical synthesis engine another optimization

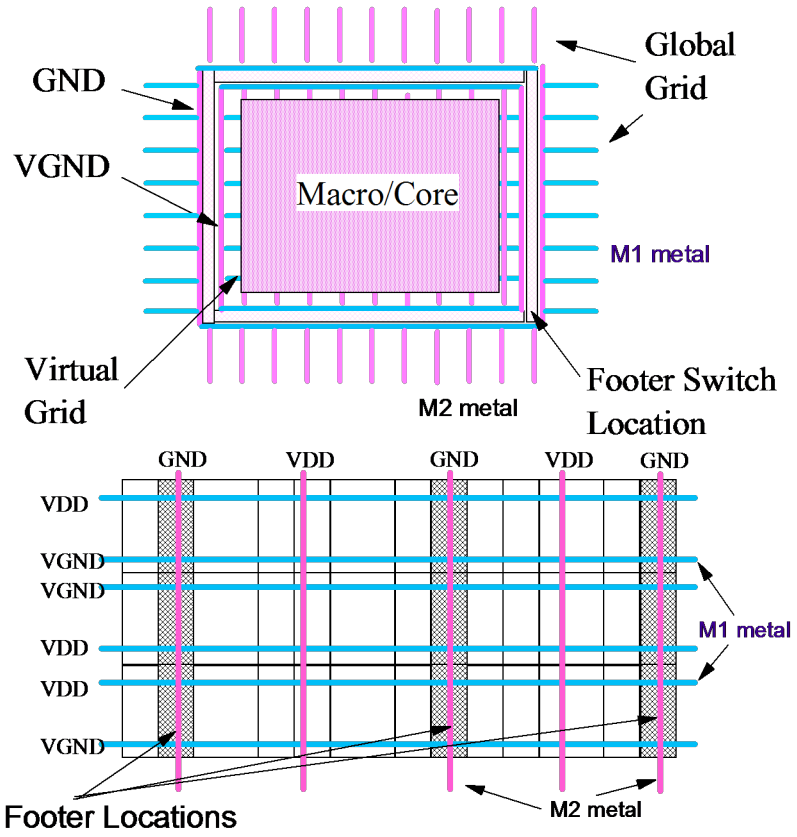



Figure 3.9. This figure shows footer-switch insertion for power gating. To manage power in the design, the power can be shut off from blocks that are not active. To shut off power from a block, a power-switch can be inserted between the global power grid and the local power grid of the block.

knob if the library contains power-efficient circuits. Often, such mapping and optimization is only done in custom design.

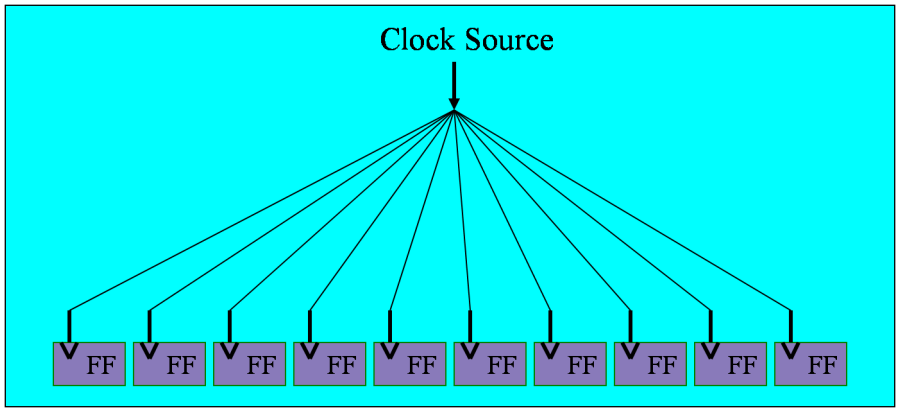
Other dynamic techniques such as *Voltage and Frequency Scaling* and micro-architectural techniques are beyond the discussion on design planning. Most of the available EDA tools deal with the above enumerated factors, though there is no technical barrier to addressing other structural factors such as *parallelism* and dynamic factors when optimizing for timing, power, or reliability.

### 3.4.3 Clock Tree Synthesis

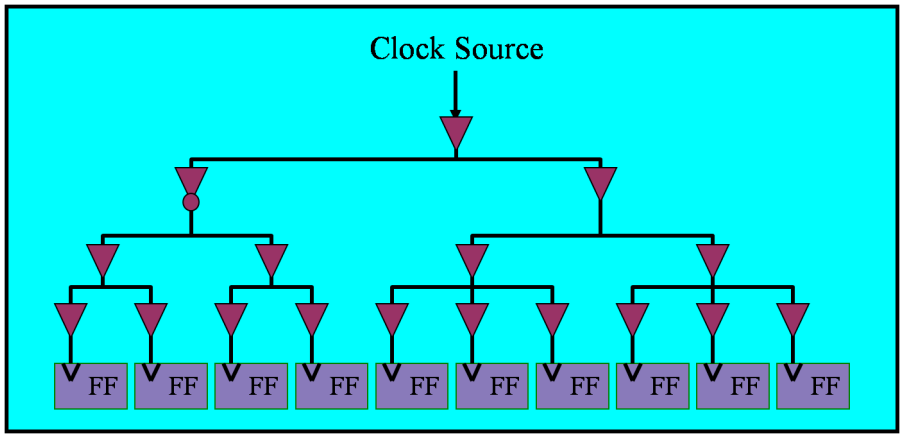
The clock synthesis problem aims at designing a clock network that delivers the clock signals to all the registers with minimal skew. In addition, there are other considerations in the design of the clock network which deal with power consumption and noise integrity (Figure 3.10).

Given the importance of the clock signal to the over all operation of the design, special care should be given to the design of the clock network. In microprocessors and high-end designs, dedicated circuit designers employ custom approaches to design the clock network at various stages in the design flow. The estimates and modeling of the clock plan are continuously visited and verified with simulation to ensure that at each stage in the design, enough knowledge and details of the clock are provided to the design and optimization teams.

ASIC designs handle the problem differently. The ASIC design flow tackles the problem of building a low-skew clock network late in the design cycle as will be detailed in later chapters. All of the design and optimization decisions made up until the cell placement is done are based on estimates of the clock network (virtual or ideal clocks).




(a) Un-synthesized clock tree.




(b) Synthesized clock tree.

Figure 3.10. Figure (a) shows an unsynthesized clock network. The delays between the clock root and the clock sinks (latches) in this network are not balanced. Figure (b) shows a a synthesized clock tree where buffers were inserted to balanced the delays between the clock root and the clock sinks (latches).

### 3.4.4 Routing

### 3.4.5 Global Routing

Global routing decides on the global topology of the nets. This task deals with constructing a grid that covers the design, and routing the nets across these grid cells. The details of how the wires are laid-out inside the cells is deferred to a later stage (Detailed Routing). The grid cells that make up the design area are divided into horizontal/vertical tracks for each layer. The assignment of the nets to tracks inside these cells constitutes the global routes of the nets.

In general, global routes provide a fairly accurate picture of the congestion of the design and the wire-length of the nets. Due to the dominance of interconnect delay in 90nm and below technologies, global routing is an embeded step in the placement stage so that the delay estimates of the wires that the placer relies on are reasonable. In addition, given that placement requires an accurate picture of the congestion of the design to decide on spreading and moving the cells around, global routing is needed to provide a reasonable congestion estimate as well.

### 3.4.6 Detailed Routing

Detailed routing finalizes the routing of the nets by laying down the polygons based on the global routing stage. Detouring and layer-assignment changes are part of this task to be able to clear obstructions and congested areas. In general, most physical optimization decisions are done based on the global routes. However, at 65nm and below, leading edge designs suffer from correlation issues between the delay of global routes and those of detailed routes. This is mostly due to under estimating the noise from near by aggressors especially when the design margins are so tight.

This is a significant change in the implementation flow. Most of the existing implementation optimization algorithms rely on global routing to make optimization decisions. Detailed routing is a time consuming step, and to include it as part of the inner-loop of the physical synthesis means adding more stress to an already over-stressed design flow.

## 3.5 Timing Convergence & Signal Integrity Problems

Due to the complex nature of the RTL-to-GDSII flow, a divide-and-conquer approach is necessary to tackle the complexity of chip design and implementation. Implementation steps are inter-dependent; thus estimation models have to be utilized to account for the impact of the down stream design steps.

Timing convergence first became an issue due to the separation between the logic synthesis world and that of physical design. In the mid to late 1990s, it was realized that close interaction if not tight integration is needed. A great deal of progress was made and a new generation of physical synthesis tools emerged. With the advent of multi-million-gate designs, even in the realm of physical design, there is a gap in modeling and analysis between pre-route and post-route design steps. The models used based on pre-routes at the placement and placement-based optimization do not accurately predict the performance and design characteristics of the post-route layout. This gap in modeling leads to another set of timing convergence issues.

## 3.6 Chip Integration

Once all the blocks have been implemented, they are brought back to the design planning stage where the blocks are stitched together. Timing sign-off analysis is performed, and if all the design constraints are honored, the design is successfully completed. Otherwise, the design will suffer from costly design iterations. Multiple design iterations are notorious in the ASIC industry due to the challenging task of closing on timing and signal integrity in large system-on-chips (SOCs). These iterations are typically a result of poor design planning or poor implementation inside the blocks. With the tight market window and tight budget that most ASIC designs operate under, any additional design iteration could spell failure for the whole design.

## 3.7 On-chip Process Variations

In addition to the notorious timing convergence problems, issues related to process variations have become more severe. The modeling and analysis of process variations are becoming part of most design steps that have been outlined above. Due to the tight noise and design margins, worst-case decisions are not feasible to guard against process variations. This further complicates the issues of timing and signal integrity of the design [4],[28].

# Chapter 4

# Power Design & Planning

## 4.1 Introduction

Power integrity is an important factor to any successful design. Power integrity refers to the notion of providing each circuit in the design the required supply voltage to enable proper switching. Given the lossy nature of chips and the various noise-inducing factors that make this task almost impossible, the goal of power design is to provide reliable power levels within acceptable design margins to the various switching devices in the chip.

Given that the design of a reliable and robust clock network necessitates a design of a robust and reliable power network, in this chapter we discuss the various factors that play a role in the design of the power delivery system. Reliable power directly affects the performance and reliability of the design. The delays of the switching devices are directly proportional to the power levels they receive. In addition, the design of the power distribution network is a function of the number of switching devices, their switching speeds, their sizes, there locations, and their interconnections. Failure to design a robust power network and provide the required power levels to the

different parts of the chip will cause the design to violate its performance constraints, and potentially, it might lead to failure in functionality. With the down-scaling of the process technology, the noise margins have shrunk to 10's of mVolts. Any perturbation in the power delivery network could cause a design failure.

In nanometer technologies, designs switching at high frequencies require a comprehensive design approach of the power network that takes into account the chip and package. Noise sources in the package such as inductive noise, signal reflections due to impedance mismatches, and signal coupling are no longer negligible; they could travel to the chip core and affect the power levels seen by the clock buffers (power supply droop and ground bounce). This will limit the performance of the clock network and may negatively affect its reliability. Thus if the design of the power network does not account for the package's effects, it can not be guaranteed to provide the necessary power levels and could cause the clock network design to fail. This is why we believe an in depth study of the power design in the package and chip is needed.

We start this chapter by introducing the power delivery system and its various components on the board, package, and in the chip. We elaborate on the the design factors impacting the chip's core and IO power networks. We then discuss the constraints imposed on the design of a power network, in particular we focus on $IRDrop$, $L\frac{dI}{dt}$ and $Electromigration$. We then discuss the various components that make up the power delivery network: switching current sources, decoupling capacitors, and the parasitics ($R$, $L$, $C$) of the power distribution interconnects and pins. It is important to mention that the extraction and simulation techniques developed for the analysis of power networks can also be used for the extraction and analysis of the clock networks.

The design of a power delivery system extends from the system's power supply, traveling through the traces on the board, up to the package pins (balls), up to the

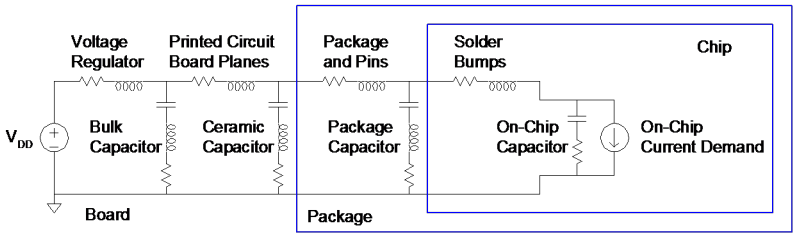



Figure 4.1. This figure shows a System Power Delivery Model. The model captures the different parts that make up the power delivery system: the power supply, the voltage regulator, the board traces and decoupling capacitors (decaps), the package power planes and decaps, and the on-chip power grid, the current sources, and the on-chip decaps.

chip power IO pins (power pads), and down to the supply pins of the cells/devices. In this journey, the power suffers from inductive noise ($L_{board}\frac{dI}{dt}$ and $L_{pkg}\frac{dI}{dt}$) on the board and in the package, and from resistive ($IR\,Drop$) and possibly inductive noise ($L_{wire}\frac{dI}{dt}$) in the chip. In addition, high-speed clocks and signals play a major role in inducing noise on the power distribution system and affect its reliability.

## 4.2 System Power Delivery

The power delivery of the ASIC consists of the board's power supply, VRM (voltage regulator module), decoupling capacitors (low and mid-frequency components), and traces to the package balls/pins (Figure 4.1).

The objective of the power delivery system is to provide a low-impedance distribution system to the design at all relevant switching frequencies. The migration to finer process technologies (45nm and below) has reduced the supply voltage below 1V. In addition, the number of integrated devices has grown rapidly which in turn placed a huge stress on the power delivery system. These factors pose serious DC ($IR$ Drop) and AC (inductive noise and other dynamic fluctuations) challenges to the design of the power delivery system.

## 4.3 Package Power Delivery

The power delivery system in the package plays an important role in the reliability of the chip's power network. If power design in the package is not handled properly, noise sources in the package can travel to the core and affect the switching devices through the shared power planes of the IO and core.

The power travels through the board traces from the power supply to the package balls (pins). In the package, power travels from the balls through a set of planes (package layers) to the on-chip power pins/pads (Figure 4.2). The impedance of the package power distribution network is dominated by the inductance of the power planes in package. From an AC analysis point of view of high-end designs, power ($V_{dd}$) and ground ($V_{ss}$) are equivalent. The height of the signal trace above the reference power plane in the package decides the size of the return path and affects the resultant inductance. To reduce the impedance of the power distribution network, careful design and allocation of the package layers to power and ground networks is needed. By sandwiching the signals between the power planes, the return current path is reduced and the inductance of the current loop is minimized. Since there are multiple $V_{dd}$ and $V_{ss}$ planes in the package, an effective technique to reduce the inductance of a current loop is to staple the $V_{dd}$ planes together as well as staple the $V_{ss}$ planes together. This will provide the traveling currents with a shorter path as they travel from one power plane to the other.

[14] provides a detailed discussion on chip-package power distribution system design and analysis.

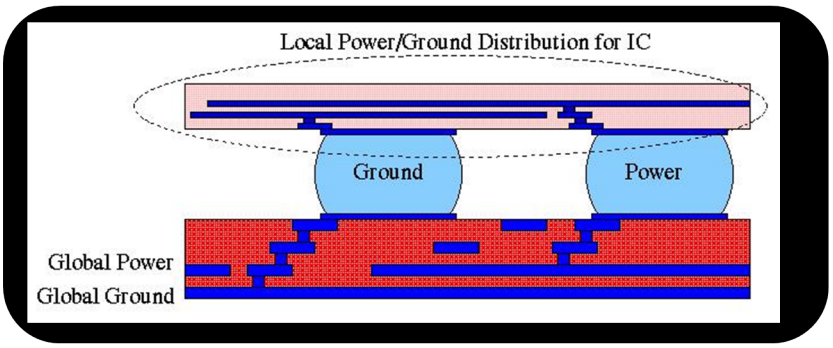



Figure 4.2. This figure shows the power distribution network in the package. This network is composed of the package balls (pins), the power planes in the package, the package bumps that feed the chip, and the on-chip power grids.

## 4.4 On-chip Power Delivery

Power is distributed to the different devices in the chip using a grid-like structure (Figure 4.3). Depending on the packaging technology used for the design, the power could be either provided from the periphery of the chip, or distributed over the entire area of the die. In wire-bond packages, the power is fed to the design from the peripheral IO ring. In flip-chip packages, the power is fed to the circuit by an array of C4 (Controlled Collapse Chip Connection) bumps which are distributed over the entire area of the die. Flip-chip packaging technology allows the IO circuits to be placed over the entire die (area IOs). However, the placement of area-IOs in the design complicates the task of physical design, and most design methodologies choose to limit themselves to periphery IOs to simplify the task even if they are targeting flip-chip packages.

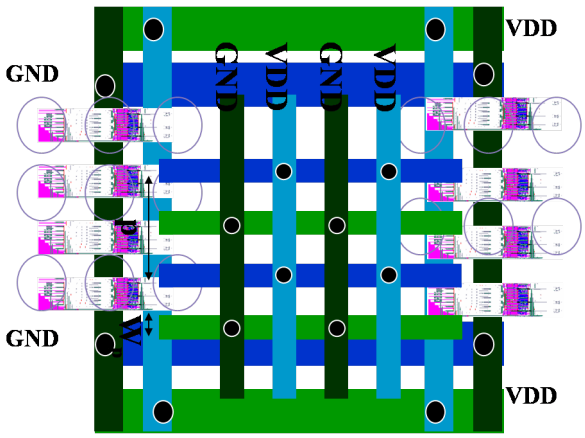



Figure 4.3. This figure shows the on-chip power grids: power and ground. In this figure, the power IO pads are placed on the periphery as opposed to area power IOs.

One major advantage of the flip-chip packaging technology is the abundance of core power supply, and hence the *IR Drop* is not much of a problem. Due to the cost of flip-chip packaging, high-end and high IO-count designs opt for this packaging technology, while the rest of designs choose wire-bond packages. In addition, the inductance of the package bump is less than 1/10 that of the wire-bond, and this is another reason why high speed IO circuits favor flip-chip packages over wire-bond packages.

### 4.4.1 IO Power

Due to the fast switching speeds of high-end IO circuits (SERDES,...), careful design of the power network for these circuits is required. The IO power network is separated from the core power network. This is done not only because the IO circuits might have different supply voltages, but also to protect the IO and the core from the high-frequency effects caused by the switching of these IO cells. The IO circuits are typically large buffers that draw large currents when switching on and off. Due to the high-inductive nature of the package and board traces, the inductive noise ($L\frac{dI}{dt}$)

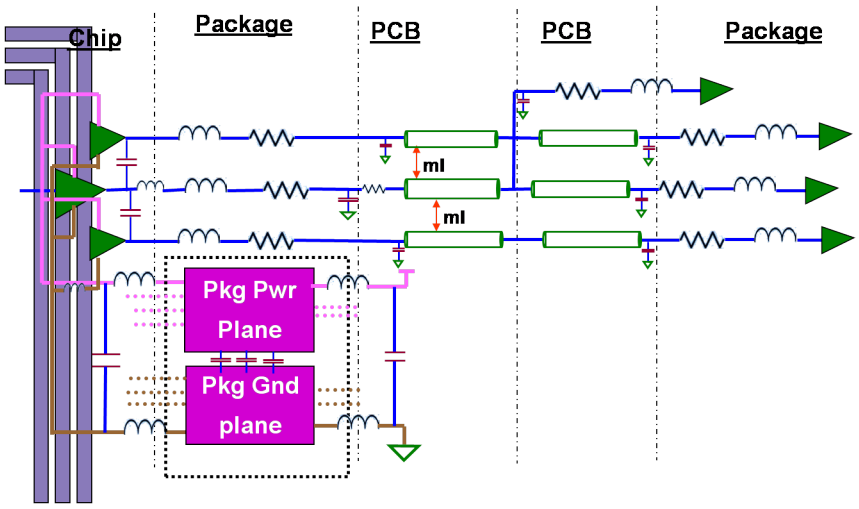



Figure 4.4. This figure shows the chip-package-board power distribution network. To carry out the analysis of simultaneous switching noise at the IOs, RLC modeling of the traces in the chip, package, and board is needed. The power planes in the package need to be accounted for as well to accurately model the inductance in the package.

is typically very large and could cause logic failures to the core if special care is not given to the design and layout of the power networks (Figure 4.4).

Due to the high inductance of the package traces, the inductive noise could wreak havoc in power supply of the switching IO drivers as well. One of the major noise problems in the design and planning of the IO circuits is *simultaneous switching noise* (SSN). This noise factor causes the power droop and ground bounce phenomena. These phenomena cause the collapse of the power supply network as seen by the switching devices at their supply pins ($V_{dd}$ and $V_{ss}$).

### 4.4.2 Core Power

The core power network is separated from the IO power network as discussed above. This results in separate power and ground pads on the chip to supply current to the IOs and core. The number of power pads needed is a function of the current needs of the logic, the size of the die, and the layout of the chip. If the pads are

bounded to the periphery of the die, then the number of pads needed is a function of the resistivity of the power grid and the estimate of the current needs. This ensures that the *IR Drop* constraint is honored and the needed current is supplied to the design. However, if the supply pads are distributed over the die (C4 bumps), then *IRDrop* is not an issue, and the main factor becomes computing a reasonable estimate of the current needs of the devices.

The inductance of the power grid is gaining importance in nanometer and high-end designs. Careful design of the power and ground networks is needed to make sure the current loops are as small as possible to reduce the effective inductance that is seen by the switching devices.

## 4.5 Power Network Design & Constraints

In this section, we discuss a number of design constraints and factors that affect the design of a reliable power distribution network. We focus on *IR Drop*, *Inductive Noise*, *Electromigration*, *Current Estimation*, and *Decoupling Capacitance Insertion*.

### 4.5.1 IR Drop

The power network has resistance associated with its wires. This resistance causes a voltage drop as power is transferred from the power pads to the target devices (Equation 4.1). To reduce the IR drop in the power grid, sufficient number of power IOs, decoupling capacitors, and sufficiently wide grid wires (low resistance) are needed.

$$\text{IR Drop} = max(V_{dd} - v_i) \quad \cdots v_i \text{ is the potential at any node on the power grid} \tag{4.1}$$

Due to the complexity of the analysis of power grids as well as the lack of information about the current consumption early in the design cycle, the design of power grid tends to be conservative. To do that, the number of the power pads and the number and width of power straps are decided based on an $IR\,Drop$ budget. Typically, the $IR\,Drop$ budget is 10% of the supply voltage $V_{dd}$. As $V_{dd}$ scales down further, it will be harder to rely on worst case decisions when designing the power grid, and more accurate estimates of the switching currents along with the size of the grid will be needed.

$$Delay_{gate} = \frac{C_L * V_{swing}}{I_{avg}} \tag{4.2}$$

Equation 4.2 shows a first-order model of the cell delay dependence on the voltage swing, the drain current of the device, the capacitive load at its output. Thus, any $IR\,Drop$ in the supplied voltage, which results in reduced voltage swing, is reflected in worse delays seen by the active devices.

### 4.5.2 Inductive Noise

As switching frequencies increase, the inductive component $L$ of the wire impedance ($Z = R + jwL$) can not be ignored any more. As the number of simultaneously switching devices increases, the $L\frac{dI}{dt}$ noise associated with these switching devices increases. This inductive noise imposes a lot of stress on the power grid. Figure 4.5 shows how inductance in the power and ground nets change the supply levels seen by the device. To overcome this problem, proper design of the power network in terms of the number of power planes , the number of power pads, the number of decoupling capacitors, and the metal layers allocated for the power network straps should be done. When deciding on the aforementioned design factors, it would be very pessimistic to assume a total noise of ($IR + L\frac{dI}{dt}$) since the inductive noise and

the $IR$ drop do not occur simultaneously. As mentioned earlier, with the shrinking of the noise margins and the supply voltage, worst-case decisions should be avoided whenever possible.

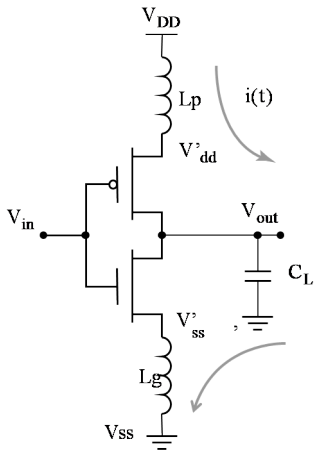



Figure 4.5. Inductive noise $L\frac{dI}{dt}$ affects the potentials seen at the $V_{dd}$ and $V_{ss}$ pins. Due to power droop and ground bounce, the supply pin of the device sees a voltage level $V'_{dd} < V_{dd}$, and the ground pin of the device sees $V_{ss} > V'_{ss}$. This fluctuation in the power negatively affects the performance of the device and increases its delay.

### 4.5.3 Electromigration

Electromigration refers to the process of mass transport in metal interconnects due to increase in the current flow. Equation 4.3 developed by Black in the late 60s [3] shows the mean-time-to-failure model used to estimate the life-time of the chip's interconnect.

$$MTTF = \frac{A}{J^n}.exp\left(\frac{E_a}{k.T}\right) \tag{4.3}$$

where $A$ is a constant based on the cross-sectional area of the interconnect, $J$ is the current density, $E_a$ is the activation energy of the Electromigration process, $k$ is the Boltzmann constant, $T$ is the temperature, and $n$ a scaling factor (usually set

to 2 according to [3]). It is clear that $J$ followed by $T$ are the main components to design for for a robust grid structure.

For maximum reliability of the power grid, a maximum current constraint based on Black's Equation 4.3 is computed for each routing layer (Equation 4.4).

$$I_{ij} <= w_{ij} * \sigma \tag{4.4}$$

where $I_{ij}$ is the current flowing in the wire, $w_{ij}$ is the wire width, and $\sigma$ is a EM constant for each layer with a specific thickness.

### 4.5.4 Current Estimation

One of the most difficult tasks of power grid design is estimating the current needs of the switching devices. Since the design of the power grid precedes the rest of the back-end implementation (Figure 3.1), the exact switching frequency, location, size, and connectivity of the switching devices are not known. The only data available at this point is the gate-level netlist as produced by the logic synthesis engine. It is not possible to extract accurate characteristics of the gates in this netlist since placement and sizing of these gates are done later in the physical synthesis stage. This is another important reason why tackling the clock network early in the design cycle eliminates a major source of uncertainty given that the clock consumes more than 40% of the power in the design.

To design and optimize the power network, the waveforms of the switching currents are needed. Different waveform models are used, most are based on simple models like a triangular or trapezoidal waveforms [10]. Typically, designers resort to historical data based on previous designs and a set of benchmarks to compute the power consumption and derive the average as well as the peak currents needed. In

addition, if circuit models of the blocks are available, simulation of these models is carried out to compute the internal average and peak currents of the devices inside the blocks. The computation of the external average and peak currents is done based on estimates of the load seen by the blocks as well as other heuristics [10].

Since not all circuits switch every clock cycle, a model for the switching activity is needed. Simulation vectors as well as random and constraint-random simulation are used to estimate the switching frequency for each node. The switching circuit models are augmented by the switching activity factors to estimate the hot spots in the power network. Accurate estimation of the current needs in the design is a subject of on-going research and more in-depth analysis is beyond the scope of this thesis.

### 4.5.5 Decoupling Capacitance Insertion

The power delivery network is generally inadequate to solely rely upon it for a robust power delivery due to the various dynamic phenomena present. Decoupling capacitors (decaps) provide a temporary charge reservoirs that supply power to the switching devices should the supply rails fail to do so or lag behind when needed. The collapse of the power grid happens for a number of reasons including those caused by the resistance of the wires ($IRDrop$), and those caused by the simultaneous switching of devices (clock buffers, latches, or other core or IO drivers) that cause a large voltage drop due to the $L\frac{dI}{dt}$ noise. Decoupling capacitors also help in shortening the current loop and reducing EMI by providing an AC path from power to ground (Figure 4.6).

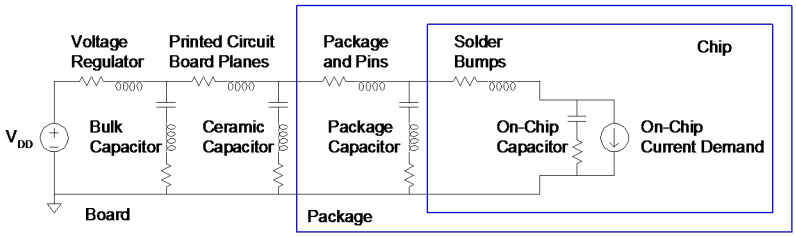



Figure 4.6. System Power Delivery Model

In high-speed designs, circuits switch at very high frequencies. To be able to supply current for the switching circuits, nearby charge supplies are needed for the various frequencies in the switching currents (Figure 4.7). In the absence of those, power has to travel from the power supply through the large inductance elements present in the board and package to get to the current-hungry devices. This increases the $L\frac{dI}{dt}$ noise and exacerbates the collapse of the power network. Each decoupling capacitor is best effective at its resonant frequency (Equation 4.5).

$$\omega_{cap} = 1/\sqrt{LC} \tag{4.5}$$

A hierarchy of decoupling capacitors is needed to support the frequency spectrum of the switching devices (Figure 4.8) and improve the performance of the decoupling capacitors. High-frequency components of the switching currents are supported by an on-chip network of decoupling capacitors, the mid-frequency components are handled by the package-decoupling capacitors, and the low-frequency components are handled by the board decoupling capacitors [22], [2].

For the on-chip decoupling capacitors, it is not enough to rely on the implicit decoupling capacitance that is available due to the non-switching logic, or the n-well capacitance (reversed-bias pn junction capacitor between the n-well and the p-substrate), or the thin-oxide capacitors present between the n-well and the polysilicon gate. However, there should not be over-design of decoupling capacitance based on

worst-case estimates since it has been reported that more than 10% of the die area is needed for additional insertion of decoupling capacitors. An integral approach to the design and planning of the decoupling capacitance is needed. Typical approaches rely on an iterative algorithm to estimate the decap values. These algorithms estimate the area needed in the floorplan and reserve space at the desired locations based on the hot-spots provided by the simulation of the power network. This iterative approach continues until the $\Delta V_{dd}$ is under control [10].

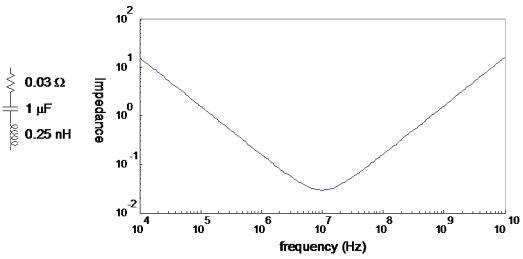



Figure 4.7. This figure shows the frequency response of a decoupling capacitor. The capacitor is most effective at its resonant frequency $f_{resonant}$. The trough in this figure denotes the resonant frequency of this capacitor.

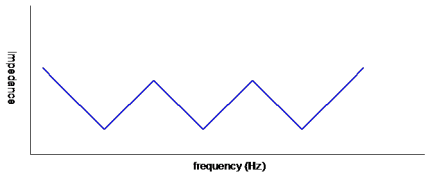



Figure 4.8. This figure shows the frequency response of a hierarchy of decoupling capacitors. A hierarchy of decaps is needed to respond to the different switching frequencies, and to reduce the inherent parasitics of the decaps.

On-chip Decoupling capacitors are also needed for designs running at low speeds as well. As the supply voltage scales down, the noise margins become tighter as well. Designing a robust power grid where the voltage does not go below 10% of $V_{dd}$ becomes very hard. Thus, designers rely on an array of decoupling capacitances with different resonant frequencies to supply the needed currents [22].

In package design, space for decoupling capacitors is reserved in case sign-off analysis shows that they are needed. For high-speed IO circuits, the simultaneous switching noise becomes a burning issue. Decaps are designed in to help the design of the power planes. Decaps by themselves are not sufficient to design a low-impedance power network; careful attention should be given to the design of the power planes to reduce their inductance. This is accomplished by reducing plane perforations, adding stapling vias, increasing the number of power planes, and assigning power and ground planes close to each other to shorten the current loop.

## 4.6 Power Network Design and Optimization

Designing a power distribution network which meets the design constraints outlined above is a formidable task. Most automation algorithms make some simplifying assumptions in modeling the power network, and this is why most of these algorithms are either academic in nature, or they are useful for constructing an initial network. In reality, most of the chips today rely on experience (previous tape-out designs) or simulation-based methods to close on a power network. Algorithm 1 shows a typical flow to design an initial on-chip power distribution network.

**Algorithm 1** Typical flow for designing an initial power distribution network

**Input:** Current estimates of the switching devices and blocks

**Output:** Initial estimates of the number of power pads and mesh density

1: Construct an initial power mesh and attach estimated current sources
2: Simulate model to identify hot spots
3: Estimate number of power pads and locations on the die
4: Revisit the floorplan to place and optimize the power pad locations
5: Construct a new power network based on the location of the current sources and the power pads
6: Simulate the new power network. Annotate the nodes with the computed potentials
7: Refine the design of the power mesh to reflect the current needs of the switching devices.
8: Estimate the decaps needs of the design
9: Integrate decaps as part of design planning.

Due to the criticality of the power network, most designers tend to over-design it. This is done to ensure a robust and reliable power delivery. However, with the increasing number of switching devices (high integration), and the high frequencies of designs, more automation is needed to cope with the complexity and accuracy of the power network design. Several algorithms for the optimization of the power network have been devised, some are based on linear programming [46], [54], [45], [16], others are based on convex optimization [5], and others employ non-linear formulation techniques [43] and [55].

### 4.6.1 Power Network Extraction

To fully capture the behavior of the power network, its model should account for the resistance and inductance of the wires and the capacitance of the wires and the devices (decaps and other implicit device capacitances). The switching devices are typically modeled as current sources as was discussed above.

The task of extracting the resistance and capacitance of the wires and devices is more or less under control. The difficulty is in extracting the inductance since it relies on knowing apriori the return path of the current. Given the complex nature of the power traces in the chip and package, it is almost impossible to know the return path without detailed simulation of the power network. The complexity of the power network renders spice-like simulation prohibitive. Thus, assumptions have to be made on the current return path. Partial Element Equivalent Circuit (PEEC) models are the most commonly used models to estimate the return path and extract the inductance in the chip and package [37], [13], [36]. This will allow constructing an RLCK (K accounts for mutual inductance in the network) network for analysis and optimization.

In nanometer technologies and high-frequency designs, inductance of the global wires can no longer be ignored, especially for the power and clock networks, and should be accounted for when extracting the parasitics of the respective networks.

### 4.6.2 Power Network Analysis

We have outlined above the different factors that constitute the power distribution model. Early in the design planning stage, the current consumption is the hardest factor to predict. First, the current needs of the design are dynamic in nature (design dependent), and thus very hard to predict. Second, early in the design cycle, the

placement and size of the current sources are not available, thus designers rely on estimates to study their locations and sizes.

The power network is composed of two parts, linear and non-linear. The linear part models the parasitics of the wires and the decoupling capacitors and is modeled as an RLCK network. The non-linear part is composed of all the switching transistors which are modeled as non-linear time-varying current sources. For first-order analysis, the current-sources can use linear or piece-wise linear models to ease the simulation task. However, proper care should be given when modeling the network so that worst case decisions and simplistic assumptions are avoided.

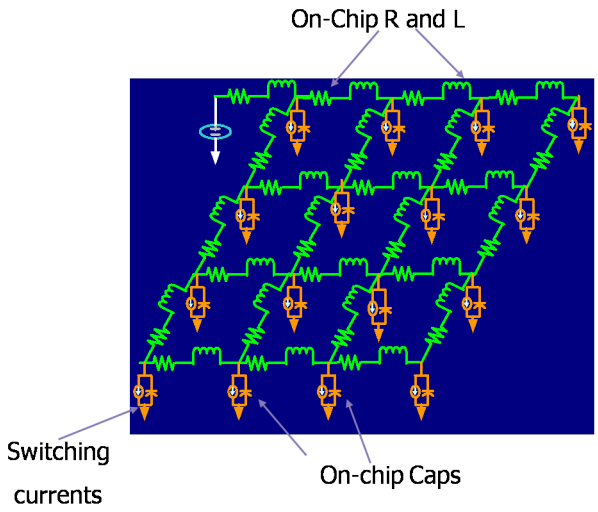



Figure 4.9. This figure shows an RLC electrical network of IC power grid. The on-chip power grid wires are modeled with $RLC$ segments. The switching devices are modeled with current sources in parallel with the on-chip decaps.

Due to the complexity of the RLCK network of the power grid which contains millions of $RLCK$ elements and devices (Figure 4.9), special simulators are designed for their analysis. The most successful simulators today are based on model order reduction (MOR) techniques [42], [53], [57], [31], [6]. Typically, MOR handles the linear part of the power network (RLC(K) network)), and an outer loop handles the

non-linear devices via fast-spice engines. [44] provides a good survey of the analysis methods used in the IC power and ground network simulation.

After this discussion of power network design and issues related to its reliability, Chapters 5 & 6 tackle issues related to the design of a robust and predictable clock distribution network.

# Chapter 5

# Clock Design & Planning

## 5.1 Introduction

The design of the clock network has become a challenging task due to the growing complexity of the designs, the down-scaling of the process technology, and the increasing frequency of the devices. In nanometer designs, tight design constraints related to skew, power, and latency are imposed on the clock network. In addition, the growing severity of process variations is making the task more complex and harder to converge.

This task is further complicated by the fact that reasonably accurate cell and interconnect delay estimates are needed for the design of the clock network. However, given that the standard cells (logic and sequential) are not placed until late in the design cycle, the ASIC design flow defers the clock network design until after cell placement. However, as was discussed in Chapter 3, accurate power planning depends on knowing the placement and sizes of the cells in the design, especially the clock buffers and sequential elements. This is because clock buffers are usually large buffers and consume a significant fraction of the clock power (Figure 5.1), and because the

clock signal is very sensitive to any supply perturbation, and thus careful power design that takes into account the clock network is needed. In [41], it is reported that a 10% noise in the power network may cause a 29% inverter delay variation in 45nm technology.

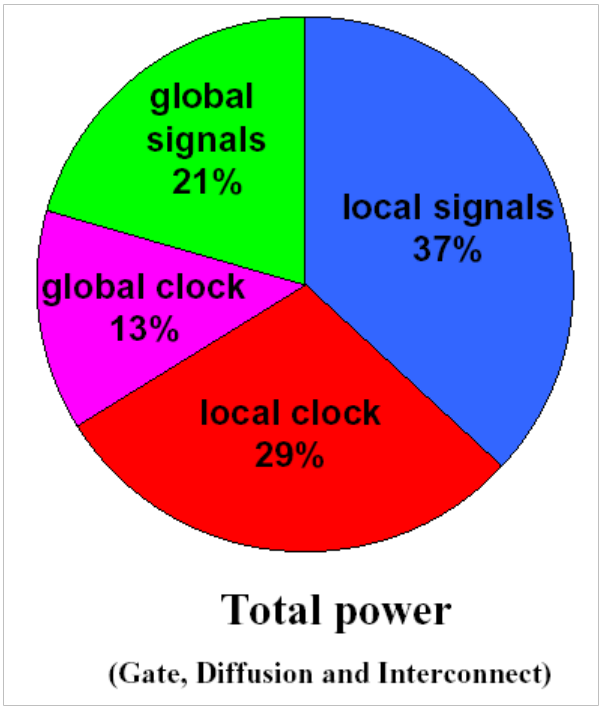



Figure 5.1. Intel's Pentium M Power Breakup

This deferment of the design of the clock network creates a big hole in today's ASIC design flow. This is because real timing annotations of the clock network are needed for the placement and optimization of the logic paths. Since the clock network is not designed until after placement-based optimization, significant design efforts are exerted to fix timing violations based on ideal (not real) clock annotations. This is problematic since any timing analysis requires knowing the clock assertions at the input of the sequential elements. By relying on timing analysis based on estimates of the clock assertions to compute delays and validate the timing assertions in the design, no guarantee of convergence is possible except if timing analysis relied on

conservative and worst-case assumptions about the clock. However, in nanometer designs, worst-case assumptions are hitting a wall and render designs infeasible due to the tight design constraints and shrinking noise margins.

This chapter is not designed to be a survey of clock synthesis algorithms. There is a large body of research that deals with issues related to physical synthesis of the clock, and they deserve a book of their own. However, this chapter is designed to give a detailed picture of the design of the clock network, its place in the design flow, the different factors that affect its performance, the different topologies that have been adopted by different designs, and the different optimization techniques that designers employ to improve and exploit the attained clock network design.

In large system-on-chip (SOC) designs, there are multiple clock domains running on the same die. A clock domain is a group of clocks that are derived from the same base clock. By having multiple clocks in a single domain running at different speeds, not all logic has to run at maximum performance and expend a lot of power. Clocks in a single domain have to be balanced and synchronized with respect to each other. This complicates the task of clock design. All further discussion on clock design with respect to a single clock applies to single and multiple clock domain designs.

In this chapter, we start by discussing the modeling of the clock network in the synthesis and physical stages of the design.

## 5.2 Clock Network Modeling for Logic Synthesis

### 5.2.1 Virtual Clocks

Prior to the interconnect-dominated era, a clock tree was synthesized and the buffers were inserted based on some load-driven delay estimates. Since the inter-

connect resistance was so low, those estimates did not differ much from the actual delay values after implementation. In the interconnect-dominated era, such an approach is no longer viable. To overcome this problem, most designers estimate the clock timing annotations (latency, uncertainty, skew) and annotate them on an ideal clock network. The hope is that these estimates will more conservative than the implementation results, and the design converges.

### 5.2.2 Trial Clock Network Synthesis

A second approach is to do placement and clock tree synthesis under the hood while doing logic synthesis in order to get reasonable clock annotations. Once logic synthesis is done, the clock network is removed before handing the netlist off to the physical synthesis stage. Although there is the issue of correlation between the final clock network that is synthesized after the P&R stage and that built during logic synthesis based on global placement information, this approach is an improvement over the ideal clock assumption since it captures the global placement as well as the global congestion in the design when synthesizing the clock network. Since physical synthesis could make big changes to the design in order to close timing or improve some design metric be it power, routability, or noise-related issue, the clock estimates generated during logic synthesis are likely to be off compared to the final numbers.

## 5.3 Clock Design at Implementation Stage

In the implementation stage, the designer is faced with the same issue as in the logic synthesis stage. As we mentioned earlier, clock network synthesis requires cell placement, in particular latch placement to be able to extract realistic parasitics and do delay calculation. However, early in the design planning stage, cell placement is

not done, and the need to make some assumptions about the clock network is still present. In a similar fashion to logic synthesis, physical synthesis has to make some assumption about the clock annotations or synthesize a clock network under the hood as it tries to optimize the physical netlist to meet the design constraints. It is worth mentioning that in both the logic synthesis and the physical synthesis, the algorithms are iterative in nature. This leads to multiple clock network synthesis processes if the chosen route is to synthesize the clock network under the hood. Since at the end of both stages and prior to the final clock network synthesis the clock network is discarded off, a lot of time and resources are wasted. This incremental and iterative refinement of the design is a by-product of the incremental and iterative design flow. In some cases, it is due to the lack of the proper automation algorithm due to the complexity of the problem at hand (NP-complete problems), and in other cases, the iterative and incremental nature of the flow is due to legacy reasons.

### 5.3.1 Clock Network Synthesis in a Flat Design Flow

The clock network synthesis approach is directly affected by the RTL-to-GDSII design flow. The clock network synthesis done in a flat fashion if the RTL-to-GDSII flow is flat. However, if the design flow is hierarchical, the clock network synthesis can be done either hierarchically or in a similar fashion to the flat approach.

### 5.3.2 Clock Network Synthesis in a Hierarchical Design Flow

In a hierarchical implementation of the design, each partition will have its own clock driver and its clock tree or network. One way of creating the clock port for each partition is to synthesize the clock network flat, and then use the information of the produced network to add clock ports to the partitions. In addition, the produced clock network will provide information on the latency and skew in each partition.

This information can be used to drive the placement and optimization of the different partitions concurrently.

It is obvious that such a design flow does not provide guarantees of convergence. None-the-less, for hierarchical design implementation, this is a typical flow that is adopted by designers. It is due to this lack of convergence guarantees as well as the complexity of the flow, designers lean towards a flat implementation whenever such a route is feasible.

The flow mentioned above not only does not guarantee convergent clock network, but also does not guarantee timing convergence of the design. When implementing each partition, the timing analysis is done with ideal clocks whose latency, skew, and jitter/uncertainty values are estimated based on the global clock planning. Since neither the placement nor the routing on which the clock plan relied are the final placed and routed netlist, the estimates could be off as compared with the final place, optimized, and routed designs.

## 5.4 Low Power Clock Synthesis

At 65nm, static power is equal to active power [32]. In addition, the dynamic clock power is accounting for up to 45% of the over all dynamic power of the chip [15]. This number is forecasted to grow as technology scales down and the number of devices increases. The task of reducing the clock power should be handled at different stages of the design and using different design techniques. Later in this chapter, we will address the different clock topologies and their impact on the power consumption.

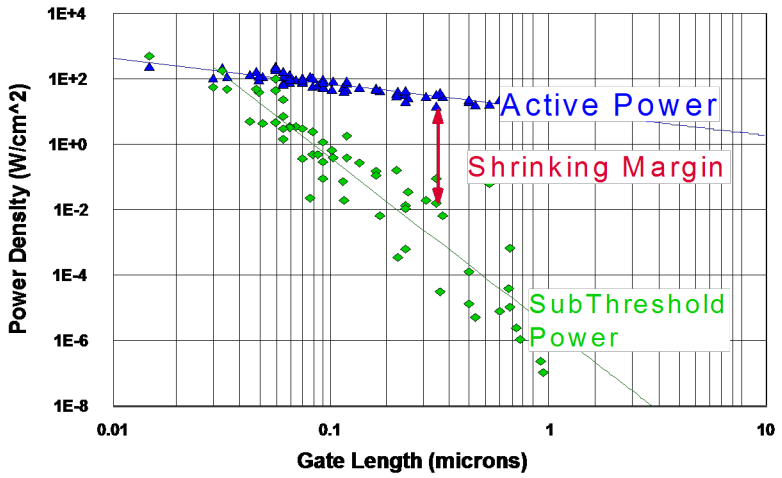



Figure 5.2. Clock distribution accounts for half of active power [32]

Equations 5.1 and 5.2 show the different factors that make up the dynamic clock power. $C_T$ refers to the input capacitance of the buffers, the interconnect capacitance of the clock network, and $C_{register}$ refers to the input capacitance of the registers (Equation 5.3). On top of this, there is the internal power of the clock buffers and the registers (Equation 5.4).

$$Power_{clock} = Power_{internal} + Power_{switching} \tag{5.1}$$

$$Power_{switching} = \frac{1}{2}\alpha\ C_T V_{dd}^2 f \tag{5.2}$$

$$C_T = C_{buf} + C_{wire} + C_{register} \tag{5.3}$$

$$Power_{internal} = V_{dd} I_{sc} \tag{5.4}$$

Clock power reduction is handled at different stages in the RTL-to-GDSII flow. In Chapter 3, we discussed *latch clustering* as one of the physical optimization knobs

that can reduce the power consumption and reduce skew. Here, we will discuss two techniques for reducing the dynamic power of the clock: *Clock gating* and *Relative Placement*.

- *Clock Gating* is the most important technique to reduce the dynamic power consumption. Clock gating exploits the knowledge of idle cycle time or future inactivity of some blocks to shut down the clock and save dynamic power. Clock gating can be done at two different stages: at the RTL stage where the designer uses his knowledge of the design to add the gating elements to shut down the inactive parts, or at the synthesis stage where automated tools are used to insert the gating elements. During the physical synthesis of the clock network, the clock gates can be merged or split (cloned or de-cloned) to satisfy the physical constraints (congestion, placement obstructions) or performance constraints (maximum capacitance or transition time at output of clock gates) (Figure 5.3). In some instances, the RTL designer inserts clock gates that are meant to be exposed to the outside world and can be used for debugging and testing. In this case, these clock gates should not be removed.

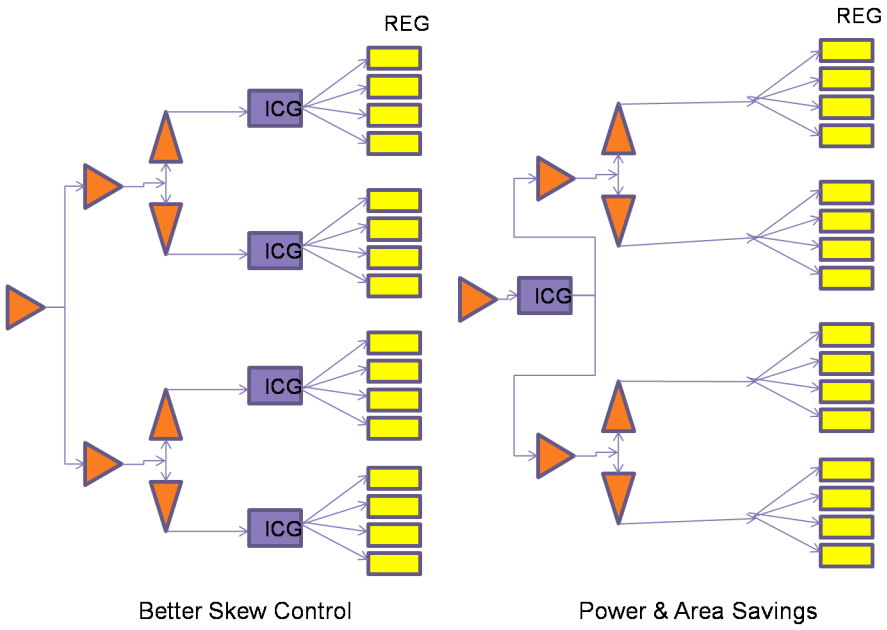



Figure 5.3. Cloning and de-cloning of clock gates for skew and power management. The decision to clone or de-clone a clock gate is dependent on the clock gate meeting its constraints (maximum capacitive load, slew).

- *Structured Clock Gate Placement* (Relative Placement) is a technique that is used in regular logic like datapath circuits. This technique reduces the load capacitance seen by the clock gates in the last stage of the clock network. The resulting effect is similar to that of latch clustering. However, in structured clock gate placement, the registers and their driving clock gates are constrained to be placed in pre-defined templates as shown in Figure 5.4.

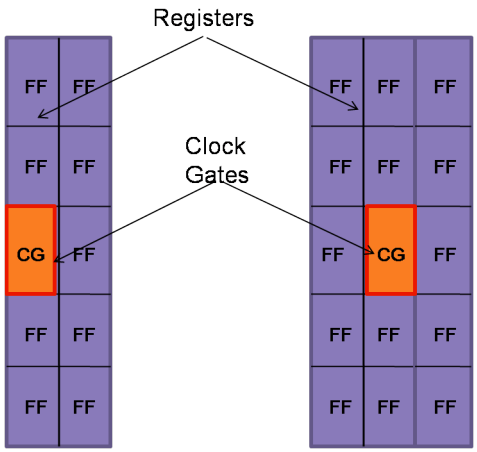



Figure 5.4. Structured placement of clock gates and the latches in their fanout cone. By constraining the placement of the clock gates and the latches in datapath blocks, the local clock networks exhibit low skew and power.

- *Insertion of Inverters vs Buffers in a Clock Tree* In balancing the clock network, either buffers or inverters can be employed provided that the signal polarity of the clock signal is maintained. There are many advantages to using inverters over buffers. The main advantage is the power savings achieved due to the smaller foot print of the inverter as compared to the buffer. In addition, due to smaller foot print, savings in area can also be achieved.

  A very promising technique mixes buffers and inverters in the same clock tree to reduce the peak current of the clock network [40],[30],[12]. When the clock switches, all the buffers at a stage in the clock tree switch in the same direction (high-to-low or low-to-high) in a well designed clock network. This in turn puts a huge burden on the power network to supply all the needed current simultaneously. This large current demand could lead to collapse in the power network. By mixing buffers and inverters at the same level in the clock tree, the signal polarity of the clock can be manipulated so that the buffers and inverters switch in opposite directions. The clock cells that switch from high to

low sink current into the ground network, while those that switch from low to high consume current from the power network (Figure 5.5).

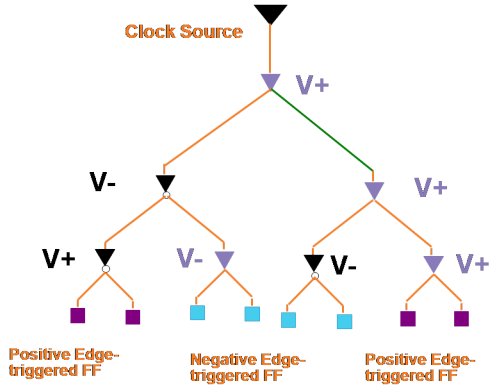



Figure 5.5. This figures shows manipulation of the polarity assignment of the clock cells in the clock network [40]. This technique leads to lower peak current consumption of the clock network.

This manipulation of the signal polarity reduces the total peak current of the clock network and reduces the clock-induced noise in the power network. [40] reports reduction in the average peak current of 38-44% and the reduction of power noise variation is 44-50%. In [12], the polarity assignment is done only in the last stage of the clock tree without any degradation in the peak current savings while improving the resulting skew of the clock network.

## 5.5 Clock Distribution Architectures

The clock distribution network is responsible to provide a reliable and stable environment for the clock signal to reach the clock cells. To do so, the distribution network should provide immunity from systematic and random variations which could distort the clock signal as it travels to its destinations.

The clock distribution network is typically composed of two parts: global and local. The global clock network delivers a reliable and low skew clock to different

parts (sections or blocks) of the chip. In high-end designs, the global clock network is mostly custom designed using know-how and relying on extensive Spice-simulation efforts. The global clock network could be either a grid, a synthesized tree, an H-tree, or a hybrid network which uses a combination of the these topologies. H-tree driving a mesh or a set of spines is a favorite top-level clock network topology due to the simplicity of the H-tree although its implementation in nanometer designs has become very challenging.

The integrity of both the global and the local parts of the distribution network are needed to provide a reliable and robust clock network.

### 5.5.1 Tree

It is the most common topology choice for ASIC designs. Although trees provide the least control over skew, they exhibit low power consumption, and low area overhead. Trees are popular because of the ease in constructing them (Figure 5.6). Low to middle frequency designs employ trees while high-end designs employ custom-made topologies.

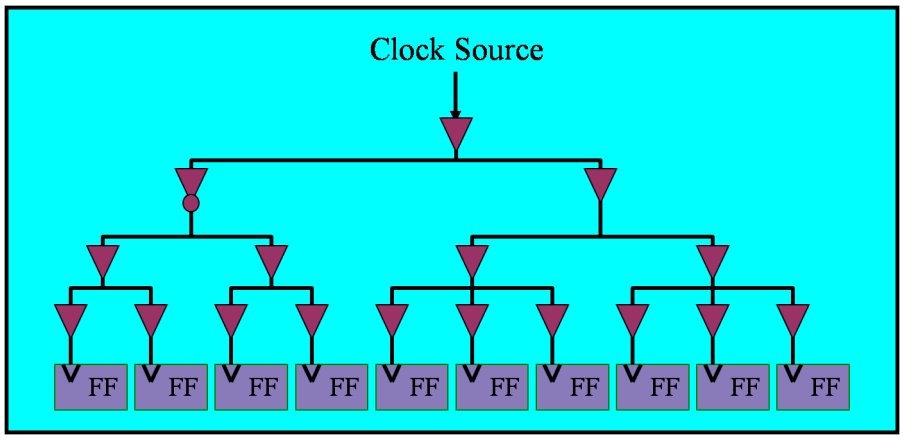



Figure 5.6. A tree generated by a clock tree synthesis engine. Repeaters are inserted at multiple levels to minimize skew.

### 5.5.2 H-tree

H-tree is a tree topology which relies on matching delays to all clock sinks in the network (Figure 5.7). This is accomplished by placing nodes at equidistant positions from their roots and by matching delays to all nodes at the same level. In real designs where hard macros and congestion may make such an ideal network unrealizable, designers settle for the best H-tree they can construct and try to optimize for whatever skew present at the leaves of the H-tree.

One major draw back to this topology is the skew and variations of two leaves that are fed by different parts of the tree (A & B) in Figure 5.7.

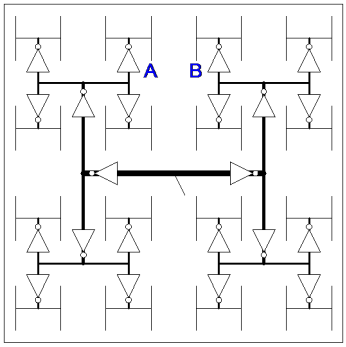



Figure 5.7. H-tree balances skew by designing equidistant paths from root to all sinks

### 5.5.3 Grid or mesh

A grid is composed of a custom-designed buffer configuration and a grid of wires. In high-speed designs, the grid is designed, sized and laid-out after several iterations of circuit simulation based on estimates of the die size, the interconnect delay, and the total load capacitance.

The network of buffers in this topology tries to equalize delays by overcoming the grid parasitics. Several generations of DEC's Alpha microprocessors used this topology in their designs as shown in (Figures 5.8 and 5.9).

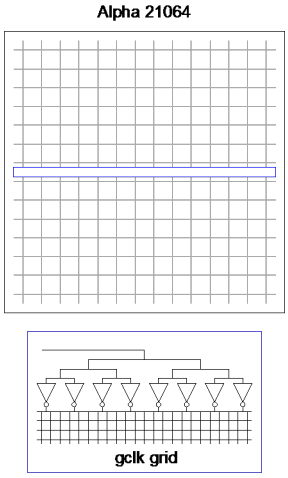



Figure 5.8. Alpha 21064 Clock Grid

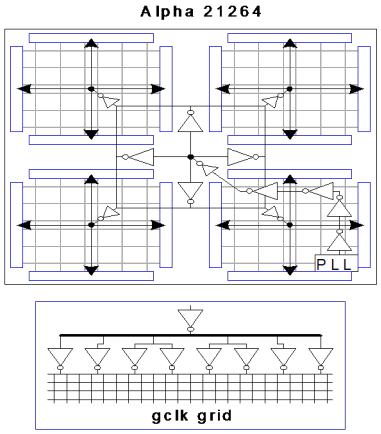



Figure 5.9. Alpha 21264 Clock Grid

Clock mesh or grid is the favored clock distribution topology in microprocessors and other high-end designs. The redundant nature of the mesh makes it more immune to on-chip process variations. However, this comes at a cost in power and area. The internal and switching power of the clock buffers are big due to the large sizes of the buffers needed to drive the large capacitive load presented by the mesh wires. In addition, the mesh wires consume significant area. To reduce the delay of the mesh

wires, top-level metal layers in the design are used for the design of the clock mesh. A detailed description of the design and analysis factors that affect mesh designs are presented in the next chapter.

### 5.5.4 Hybrid Networks

To get the best of all worlds, high-end designers build hybrid clock networks to meet their design goals at a reasonable cost. A tree topology is known for its low cost and low power consumption. A mesh topology is known for its high performance and immunity to process variations. A hybrid clock network borrows from both worlds to construct a network with reduced power and minimum skew.

Figure 5.10 shows Intel's Itanium hybrid clock network. This clock network employs both custom design and automated synthesis engines in its design and optimization. The top-level clock network is composed of a tree driving a set of spines. The spines are manually designed and placed to improve the drive strength and reduce the skew to the low level clock trees that drive the clock registers. In addition, the spines are driven by delay-programmable clock drivers. The programmability of these drivers allow them to be tuned post-silicon to reduce the skew. The network of distributed spines drive two levels of clock cells: regional clock buffers (RCB) and local clock buffers (LCB). Regular clock trees are employed to balance RCBs and LCBs with the least amount of wiring. Typically to reduce the capacitive load and the wire delays of the clock networks driven by the RCBs and LCBs, fish-bone routing is used.

ASIC designs can not afford such a costly and elaborate clock network due to the lack of expertise and the tight time-to-market constraints that ASICs typically operate under.

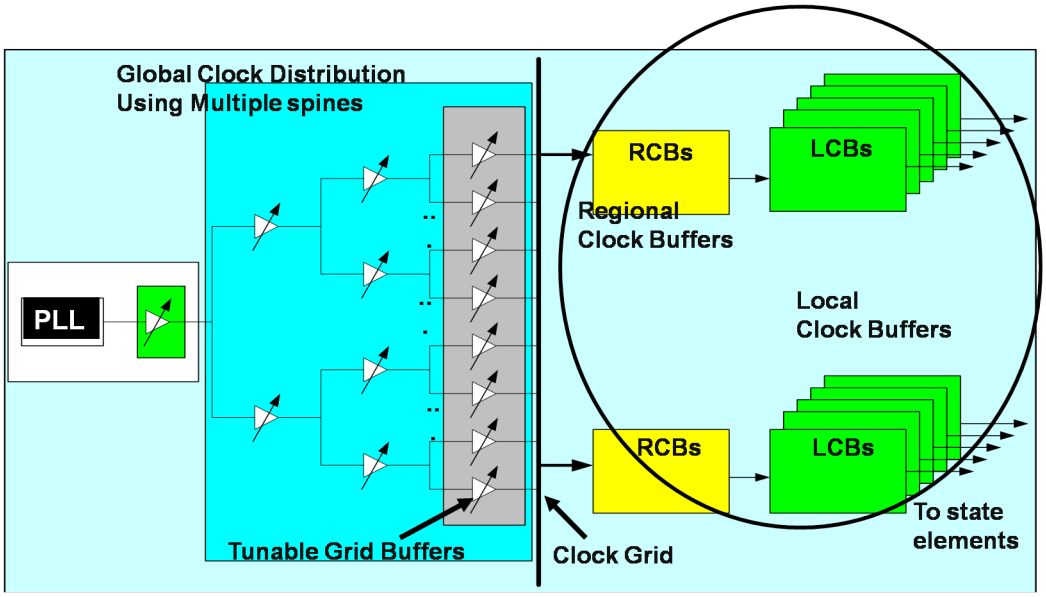



Figure 5.10. Itanium Clock Distribution Network

## 5.6 Package Clock Distribution

Recent work advocated routing the clock network in the package. The premise is that better skew and latency can be attained due to the the low resistivity of the package traces. However, as discussed above, inductance is a big cause of noise in the package, and careful design and optimization of the clock network is needed for such an approach to work.

## 5.7 Clock Skew Scheduling

As clock network design became more complex, and design convergence became harder, design emphasis shifted from designing minimum-skew networks to designing low-skew clock networks while reducing power and improving robustness. By utilizing the available skew, timing convergence of the design can be enhanced. There are two approaches to skew scheduling: useful skew and intentional skew.

- *Useful Skew* converts the present skew into a useful timing budget that can be allocated to that critical or near critical paths in the design. This is done by shifting the clock assertions such that STA will use the adjusted (relaxed) clock assertions when checking for timing violations and reporting critical paths.

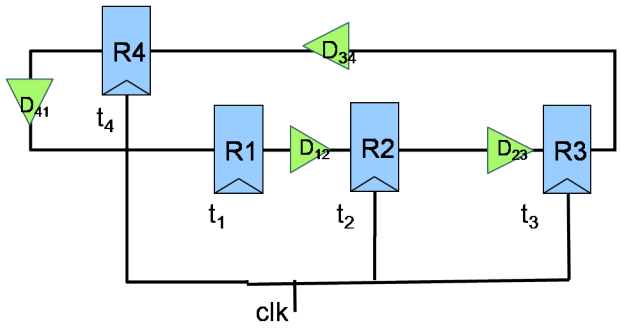



Figure 5.11. Useful Clock Skew relies on exploiting the skew present in the clock network to relax the timing assertions at the end points of the critical paths.

In Figure 5.11, $t_i$'s are the actual clock arrival times that are produced by clock tree synthesis. By shifting the timing assertions at the input of the flops, the path delays can take advantage of the available clock skew to optimize timing and reduce violations.

- *Intentional Skew* is a sequential optimization technique to design skew in as part of the logic/physical synthesis stages. The task becomes one of designing a clock network satisfying the skew constraints generated by the synthesis engine to optimize performance. This can be accomplished either by inserting intentional delay (buffers/inverters) on the clock paths that need to be delayed, or it can be accomplished by sizing the buffers/inverters to decrease or increase the delay along some paths. Although skew scheduling reduces the number of close-to-zero skew clock nodes, the actual physical implementation of the clock network becomes harder. Another advantage of this approach is to reduce the number of simultaneous switching clock cells by delaying the toggling of some of the

registers. This is desirable in order to reduce the peak current consumption of the clock network and reduce the noise injected into the power grid.

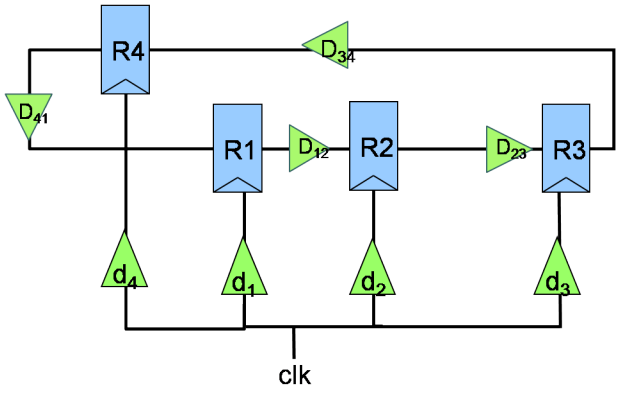



Figure 5.12. Intentional Clock Skew relies on designing intentional delays in the clock network to improve performance.

In Figure 5.12, $d_i$'s reflect the intentional delay added to the clock network to modify the arrival of the clock signal at the registers. [18] first formulated the problem as an LP to compute the $d_i$'s to optimize the clock cycle. Further work extended this work to include physical synthesis in the process of computing the intentional delays. [47] extended this work to latches as opposed to registers. Process variability could be considered as imposing another set of constraints on clock skew scheduling to improve the robustness of the clock network.

## 5.8 Optimization of Registers

Circuit optimization plays a key role in the performance and cost of the clock network. Selection and optimization of the type of latch or register to be used in the clock network has a great impact on the achieved skew and power. To properly assign the type of latch needed, timing analysis is performed to annotate the netlist with the correct path constraints and path slacks. Since different logic paths have

different skews, the fastest (most power consuming) latch is not warranted on every path. A trade-off can be made between power and performance on the less critical paths. Typically, such an optimization is not carried out as part of the back-end flow since changing the registers used in the RTL netlist is not encouraged. However, given that most of the power is in the last stage of the clock network, such circuit optimization is needed to reduce the power consumption of the clock network and to converge on design timing.

## 5.9 Clock Analysis and Sign-off Verification

The clock network is composed of both clock buffers and interconnect parasitics. For accurate analysis of the clock network, buffers are modeled as non-linear elements to capture the transient effects present during the switching of the clock buffers. The clock buffers do not switch at the same time due to the interconnect delay in the clock network as well as the variations in the interconnect parasitics and the clock buffers as a result of the process and environmental variations.

Many of the analysis engines and modeling decisions that were discussed in Chapter 4 related to power analysis are applicable here. The non-linearity of the clock buffers present complexity to the static timing analysis engine. STA engines employ a broad spectrum of delay models, some are as simple as switched-resistor models while others are as advanced as non-linear look-up tables, IV-curves and current-source based models.

The analysis of the clock topologies presented earlier such as grids/meshes and spines require special handling of the multi-driven nets. STA engines are not capable to accurately analyze these nets. The delay calculation of the multi-driven nets is

usually handled by Spice-like simulators to capture the non-linear and simultaneous switching nature of the clock drivers that drive these nets.

Due to the criticality of the clock network, for sign-off analysis, most high-end designers use Spice-like engines to analyze the clock network. The Spice engine calculates the cell delays and slews at the inputs and outputs of the clock buffers and the inputs of the clock registers. Based on the delay annotations produced by the Spice-like engine, the STA engine calculates the delays of the logic part of the design (cell and interconnect), and perform the timing checks needed to ensure proper functionality and accurate performance.

In this thesis, we employ this approach in the analysis and verification of the our clock network. We use Synopsys Nanosim (fast-Spice engine) to simulate the entire clock network. We load the simulation values into the design and annotate all the inputs and outputs of the clock buffers and the register inputs. This approach relieves the static timing analysis (STA) engine from doing any delay calculation on the clock signal; instead the STA engine reads the annotated delays produced by Synopsys Nanosim.

### 5.9.1 Clock Analysis and On-Chip Variations

If statistical analysis algorithms are not employed in the verification of the clock network, designers have to rely on worst case decisions to study the timing of the clock in the presence of process variations. Derating metrics are imposed on min and max (early and late) modes of the analysis. Typically 10-20% derating factors are used to slow the slow paths and speedup the fast paths in STA engines. Although this conservative approach makes converging on the timing of a design with very tight timing constraints difficult, it is the most common design approach adopted today.

However, this presents a problem when a cell is common between a clock path

and a data path. To account for process variations, the timing is de-rated for the min-max analysis during STA. This causes pessimism in the analysis since it is not realistic to have the cells on the common path exhibit early and late delay values at the same time (Figure 5.13). The pessimism factor is known as *clock re-convergence pessimism removal*, and special attention is made to remove it during static timing analysis.

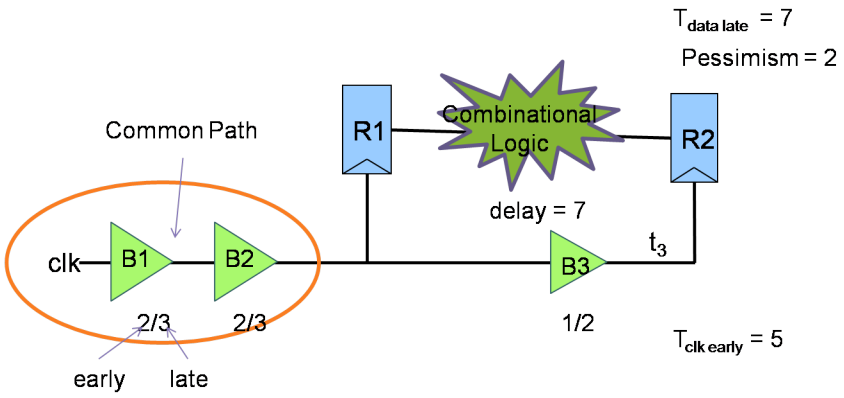



Figure 5.13. Common Path Pessimism Removal refers to removing the pessimism introduced during STA when derating a common path of the clock network between a pair of launching and capturing registers.

## 5.10 Clock Link Insertion

After constructing the zero-skew clock tree, further optimization might be needed to compensate for layout or process variation factors. [34] and others present a design technique that inserts links to transform the tree to a non-tree structure with a low wiring penalty as compared to a mesh network (Figure 5.14). The difficulty here lies in selecting the nodes to be shorted and what impact interconnect variations could have on the over-all network.

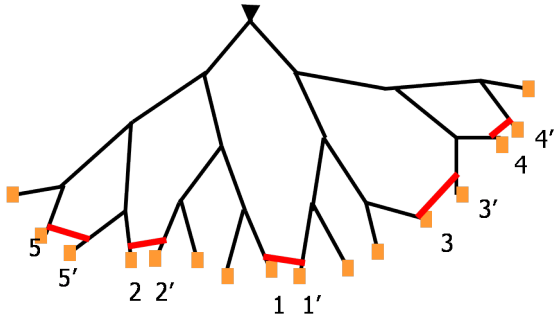



Figure 5.14. Clock Link Insertion to compensate for process variations effects. By shoring some nodes in the clock tree, skew is improved due to the improved driving strength to some nodes. Also, tolerance to process variations is enhanced due to the redundancy introduced in the clock network.

# Chapter 6

# Clock Mesh Design

## 6.1 Introduction

The number of clock registers in the design is growing at an astounding rate. This puts a huge burden on the clock network and makes balancing it a formidable task. As the die area increases, the clock network has to stretch over the entire area with long wires. These long wires require careful planning and optimization to control their delays. In addition, the clock buffers that are needed on these long wires tend to be large cells which consume a lot of power and present large blockages to the place and route engines.

For high-speed designs, careful design and planning of the top-level clock network is key to a successful tape-out. Typically, microprocessor designs choose a mesh implementation of the clocks due to their aggressive timing constraints. Traditional zero-skew or near-zero skew clock tree synthesis engines can not handle this task.

However, clock meshes have not been favored in ASIC design styles due to the design cost associated with them. Mesh topologies consume more power and area as compared to trees. Microprocessors, in general, enjoy a bigger design budget

for power and other resources as compared to ASICs. For example, microprocessor designs can dedicate special design teams with circuit design expertise to tackle the clock design problem. Also, they can afford to dedicate special routing layers for the clock if needed to insure robust and high-performance implementation. However, ASIC designs typically do not have this expertise available and can not afford such high costs due to the tight budget under which they operate.

One major attractive point of meshes is their tolerance to process variations due to the inherent redundancy in their topology. This is a very desirable factor for nanometer designs operating at multi-GHz frequencies. The tight design margins and increased susceptibility to process variations of nanometer ASIC designs will lead to a wider adoption of clock meshes.

In our proposed clock driven-design methodology, we chose a clock mesh as part of our predictable and robust clock network (Figure 6.1). Any other predictable and robust topology (distributed spines for example as was discussed in Chapter 5) can be utilized in this approach. We chose a hybrid clock network, composed of a mesh driven by a top-level clock tree. The mesh drives a set of latch clusters driven by local clock drivers (buffers or clock-gates). In this chapter, we expound on issues related to the design and optimization of clock meshes.

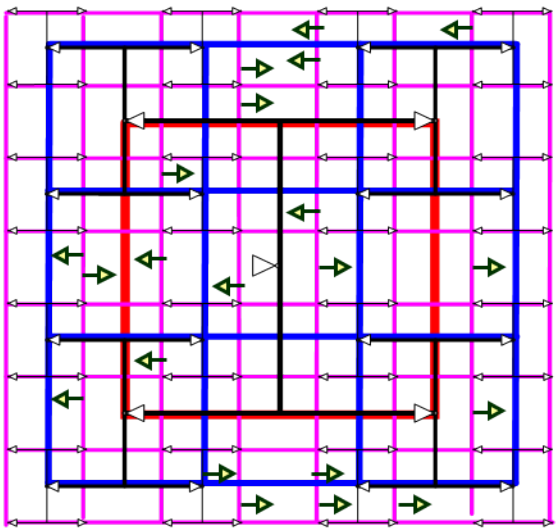


Figure 6.1. This figure shows a clock mesh driven by a 3-level H-tree. The buffers at each level of the H-tree are shorted together. Shorting the H-tree is not mandatory in designing a top-level clock network; however, designers may choose to do so to improve the skew. In this figure, clock buffers are inserted at each level in the H-tree. The clock buffers in the last stage of the H-tree constitute the mesh drivers. These mesh drivers drive a set of horizontal and vertical wire segments that make up the wiring of the clock mesh.

## 6.2 Design Cost

As mentioned, clock meshes consume a lot of power and wiring resources. Due to the large capacitive load presented by the routing segments of the mesh, a significant number of clock buffers is needed to drive the mesh with acceptable skew and slews. This in turn puts a burden on the top-level clock network to drive the large mesh drivers. Typically, although not necessarily, a tree topology is the topology of choice to drive the clock mesh. For custom clock networks, the tree is manually crafted and optimized. Most often, an H-tree is chosen to be the top-level clock network. Numerous simulation runs are carried out to balance and optimize the clock network. Due to the lack of automation in this regard, rarely are the top-level clock network

and the mesh optimized. In general, they tend to be over-designed to produce the desired skew and robustness at the expense of clock power and routing resources.

In ASIC designs, the top-level clock network could also be a synthesized tree that is built by a CTS engine. In this case, the goal would be to meet the skew and slews constraints at the inputs of the mesh drivers while reducing area and power if possible. We use this approach in some of the results that we report in Chapter 8. Recently, more attention is given to low-power clock tree synthesis due to the tight power budgets of the mobile applications. In these instances, the available automation tools sacrifice performance to meet the power budget.

## 6.3 Clock Mesh Design Goals

The design goals of a clock mesh implementation are shaped by the type of design, and the available timing and power budgets. Below are the most important design goals of designing a mesh:

1. *Skew* reduction is key to the top-level clock networks. Interconnect delays tend to dominate the cell delays, and with careful design, the clock is transmitted to the regional and local clock buffers with minimum delay variations.

2. *Slew* constraints need to be honored to insure proper switching of the cells. Faster slews mean smaller delays and faster switching. To reduce the short-circuit power in the clock buffers, it is essential that the clock buffers switch fast from the on-state to the off-state to reduce the time when both the P-channel and N-channel devices are partially turned on. When both P-channel and N-channel devices are partially on, a current known as *crowbar current* flows from the power supply pin to the ground pin and is responsible for the dissipated short-circuit power ($P_{short\_circuit} = I_{crowbar_{avg}} * V_{dd}$).

3. *Dynamic Power* is a major concern in clock design. As mentioned earlier, dynamic power of the clock network accounts for more than 40% of the total dynamic power consumption of the design. As the number of sequential elements grows, and the skew budget shrinks, the dynamic power of the clock grows as well. For ASIC designs, this is a serious issue because of their tight over-all power budgets.

4. *Noise* is a major issue in high-speed clock design. Since clock is the most frequently switching signal, it is one of the biggest noise aggressors. In addition, clock is the most sensitive signal due its high-speed switching. For high frequency designs, it is necessary to shield the clock from any aggressor to reduce any timing uncertainty. Careful design of the power network should be done to mitigate any $IR\ Drop$ or inductive noise concerns due to the fast switching of the clock network.

5. *Latency* is the delay of the longest path in the clock network from the clock root to any sequential element. When a clock domain with two or more clocks needs to be synchronized, the typically adopted approach is to delay (increase latency) the faster clock once both clocks are balanced separately for near-zero skew. Increasing latency means either increasing the wire length or inserting delay cells between the clock root and the lower buffer stages.

   In the past, reducing clock latency was not a priority as compared to reducing skew or power. However, one of the main causes for high latency is long wires which are not desired in nanometer designs due to their susceptibility to on-chip process variations. To build a more robust clock network that is tolerant of on-chip process variations, it is desirable to optimize the delay of the long wires and reduce the latency when possible.

## 6.4 Prior Work in Mesh Optimization

Design automation lacks reliable and high quality algorithms for the synthesis of meshes in general. Most mesh implementations are designed manually using iterative design methodologies. Simulation is the key engine of choice for designing and possibly optimizing the mesh network. In general, conservative design decisions are exercised to insure high performance and reliability of the clock network. These conservative decisions lead to an over-designed clock network, leaving a big opportunity for automation to optimize the power and area of the clock mesh.

There are two aspects to optimizing the clock mesh: 1) reducing the number of clock buffers, and 2) reducing the mesh wires. To ensure skew predictability, accurate circuit models for the buffers are needed to capture the non-linear switching effects as well as the non-linear behavior of the over all mesh circuit with its multiple drivers and receivers. In this section, we will elaborate on prior work related to optimizing the number of clock buffers using simple load-based delay models. Given the non-linear nature of the circuit, there is no guarantee that the resulting solution is sufficient to meeting the desired slew and skew constraints.

### 6.4.1 Clock Buffer Placement and Sizing

In [51], the clock mesh synthesis problem in terms of optimizing the mesh wiring, the number of clock drivers, their placement and sizing was formulated as a combinatorial optimization problem and modeled as a set-covering problem. Since the set-covering problem is known to be NP-complete, a heuristic is introduced to find a solution. A greedy algorithm is suggested to choose a set of buffers that is sufficient to cover (drive) the mesh nodes and registers. The following notations and conventions are introduced:

- $m$x$n$ denotes the dimension of the clock mesh. $I$ denotes the set of nodes in the mesh.

- Clock buffers of $B$ sizes $\{b_1, b_2, \ldots\ b_k\}$ in non-decreasing order. Buffer $b_i$ can drive a load of capacitance at most $c_i$.

- Buffer Mapping Function BM : $i\ \rightarrow\ j$ maps each node location $i \in I$ to $j \in B$. $BM(i) = \phi$ implies that the location $i$ has no buffer in it.

- $S = \{s_1, s_2 \ldots\ s_n\}$ denotes the set of clock sinks. Each sink $s_i$ is connected to node $i \in I$. The node $i$ in the mesh is referred to as the connection node of sink $s_i$. $d_{ij}$ denotes the minimum distance between node $i$ and $j$ in the mesh.

- Covering Region of the node for a particular buffer is defined as the set of nodes around the node in the 2-dimensional mesh such that the total capacitance of the nodes included in the covering region (including the mesh capacitance as well as the nodes that the mesh drives) is less than the maximum capacitance that the buffer can drive.

The problem is formulated as follows: Given a library of buffers, $B$, of different sizes, and a mesh of size $m$x$n$ segments, and a set of sinks placed in the design. Design a mesh such that:

1. Each sink $s_i$ has at least $k$ node locations such that for each such node location $j$, $d_{ij} \leq L_{max}$.

2. There is a buffer placed at node $n_j$.

3. There exists at least $l$ edge disjoint paths between $j$ and $i$.

$k$, $L_{max}$ (maximum delay between a sink and a nearby buffer) and $l$ are user defined constants. Let $CR_i^j$ denote the covering region of node $i \in I$ while driven by

buffer $j \in B$. That is $CR_i^j \in I$ for each $i \in I$ and $j \in B$. Let $SCR$ denote the super set of covering regions. We can draw the parallels between the above defined variables and the set covering problem: The set $I$ of node locations can be considered as the universe $\cup$. The covering regions $CR_i^j$ form the collection of subsets $S$. If the weight of a covering region $CR_i^j \subset S$ is measured by its buffer size $b_j$, then the objective is to minimize the weighted sum of subsets such that each node has at least one subset covering it. In equation 6.1, $x_i^j$ is an indicator variable that is set to 1 if buffer $b_j$ is selected or zero otherwise.

$$Min: \quad \sum_j \sum_i b_j x_i^j \tag{6.1}$$

$$\bigcup_{(i,j):x_i^j=1} CR_i^j \supseteq I \tag{6.2}$$

*Observation 1*: For any node $i \in S$, $CR_i^j \supseteq CR_i^l$ if $b_j > b_l$. The observation comes from the fact that a bigger buffer size can drive a bigger load (*Observation 1* in [51]).

*Lemma 1*: In any optimal solution $\Phi$, for any node $i$, there can be at most one buffer $j$ such that $x_i^j = 1$.

Proof: Direct consequence of the *Observation 1*. If there exists two buffers $j$ and $l$ such that $x_i^j = 1$ and $x_i^l = 1$ and $b_j \geq b_l$, then $CR_i^l$ can be removed from $\Phi$ without any loss of feasibility. This implies that $\Phi$ is not optimal and hence a contradiction (*Lemma 1* in [51]).

*Corollary*: In any solution $\Phi$, for any node $i$, if there are more than one buffer driving a node, one can pick the biggest buffer without losing feasibility (*Corollary 1* in [51]).

The set cover problem is implemented using a greedy algorithm (Algorithm 2). At the end of algorithm, the solution is pruned using *Corollary 1*.

**Algorithm 2** Greedy Set Covering algorithm for mesh buffer placement/sizing [51]

**Input:** $SCR = \bigcup CR_i^j$ // where $CR_I^j$ is covering region of node $i$ driven by buffer $j$

**Output:** $M$ = set of covering regions that are picked

1: $M \leftarrow 0$
2: **while** M does not cover I **do**
3: **for** each unpicked covering region $CR_i^j$ **do**
4: define $C_{eff} = \frac{b_j}{|CR_i^j - M|}$ // where $b_j$ denotes buffer $j$
5: **end for**
6: Pick set $C$ with least $C_{eff}$
7: $M = M \cup C$
8: **end while**
9: **for** each node $i \in I$, if there exists $j \in B$ and $l \in B$ **do**
10: both $CR_i^j$ and $CR_i^l$ are picked and $b_j > b_l$
11: drop $CR_i^l$ from the solution.
12: **end for**

The cost function $C_{eff}$ used in Algorithm 2 to pick a set of buffers does not account for the following two cases. First, it does not account for the mesh optimization (mesh wires removal) when the buffers are inserted. Second, Algorithm 2 favors large buffers over a set of smaller buffers although both cover the same area. This is because $C_{eff}$ of large buffers will always be smaller than that of a small buffer, and thus a large buffer that covers a set of nodes will be favored over a set of smaller buffers that cover the same set of nodes. This greedy decision does not account for the exponential attenuation of the signal as it travels in the mesh wires from the clock buffer to

farther mesh nodes. In [35], a new model for $C_{eff}$ (Equation 6.3) is used in the same greedy algorithm to remedy these issues.

$$C_{eff} = \frac{b_i}{b_T^2} * \frac{1}{N_{uncovered}} * \frac{1}{C_{load}^i} \tag{6.3}$$

where $b_i$ is the buffer under consideration, $b_T$ is the largest buffer in the library, $N_{uncovered}$ is the set of nodes that is not covered yet and can be covered by buffer $b_i$, and $C_{load}^i$ is the load capacitance at mesh node $n_i$ including the input capacitance of all the sinks that are connected to $n_i$. By changing $b_p$ in ($C_{eff} = \frac{b_p}{|CR_i^j - M|}$) to $\frac{b_p^2}{b_T^2}$ in Equation 6.3, the cost function forces the cost of several small buffers to be less than the cost of one big buffer even if the two solutions have the same area.

### 6.4.2 Mesh Wiring Optimization

The presence of the non-linear devices and the electrical loops make the analysis and optimization of the clock mesh a formidable task. This difficulty is reflected in the optimization of the mesh wiring as well.

The problem of mesh wiring optimization work was addressed in [51] and [35]. In [51], an algorithm based on Network Survivability, which is commonly used in the communication industry, is utilized. In a nutshell, to maintain certain redundancy in a communication network should a set of links fail, the algorithm tries to maintain a set of disjoint links between any pair of nodes. This has parallels in clock mesh design where the mesh is desired not just for its high-performance, but also because of its redundancy which makes it tolerant to process variations. To optimize the clock mesh while maintaining this redundancy, [20],[21] proposed an algorithm to optimize the wiring of a Steiner network while imposing certain constraints to ensure the redundancy. The problem is stated as follows:

*Given (a) Graph G = (V,E) (b) A cost function c for the edges and (c) A connectivity requirement function r : $V \rightarrow Z_+$, find a minimum cost sub-graph in G such that there exists at least r(u, v) edge disjoint paths for every ordered pair u, v ∈ V.*

In optimizing the wiring of the mesh, the goals are to remove the maximum number of mesh wires while keeping enough redundancy and maintaining low skew in the mesh. To ensure a skew target is met, every clock sink $s_i$ should be connected to at least $k$ mesh node locations such that for each such node location $j$, the delay $d_{ij} \leq L_{max}$, and a buffer is placed at mesh node $j$ (the mapping function of buffers to mesh nodes $BM(j) \neq \phi$). To ensure enough redundancy is present in the mesh for every sink $s_i$, there should exist at least $l$ edge disjoint paths between node $j$ and $i$.

The mesh reduction problem is transformed into Steiner Network problem by the following procedure (Algorithm 3):

1. Let the mesh be represented by a graph $G = (V, E)$.
2. Set connectivity requirement function $r(u, v) = 0$ for all $(u, v) \in V$.
3. For each clock sink $s_i \in$ S, identify $k$ closest mesh buffer locations (say) $T_i = (t_1, t_2, ...t_k)$.
4. Set $r(i, j) = l$ for all $s_i \in S$ and $j \in T_i$.

**Algorithm 3** Greedy Algorithm Based on Steiner Network Optimization for Mesh Reduction [51]

**Input:** $G = (V, E)$, and connectivity requirements

**Output:** $E' \subset E$ satisfies connectivity requirements

1: **for** $e \in E$, set $c(e) = 1$ **do**
2: **for** sink $s_i \in S$ **do**
3: Find $k$ closest buffers locations
4: Identify $l$ minimum cost disjoint paths (denoted by $P_i$) between $s_i$ and identified buffer locations
5: **for** e $\in P_i$ **do**
6: $E' \rightarrow E' \cup e, c(e) = 0$
7: **end for**
8: **end for**
9: **end for**
10: Output $E'$

In Step 1, Algorithm 3 starts with initializing the cost of all edges to unity. In Step 3, the $k$ closest buffers to a sink are identified which implicitly takes care of the the short path constraint (by means of $L_{max}$). This is due to the fact that if these closest buffers do not satisfy the $L_{max}$ requirement, it is easy to see that there exists no other buffer locations than can satisfy the constraint, and $L_{max}$ requirement should be relaxed. Further, because of the connectivity requirement, edges in the shortest pairs will be retained. In Step 4, the algorithm identifies edge disjoint paths between clock sinks and the closest $k$ mesh buffers. It is worthy to mention three points about identifying these paths: (a) The disjoint path requirement is between a clock sink and a particular mesh buffer and not across all the $k$ assigned buffers. For example, if a sink $a$ is assigned to buffers at locations $b$ and $c$, then we need to identify $l$ disjoint

paths between $a$ to $b$ (say $P_{ab}$) and $a$ to $c$ (say $P_{ac}$). While the paths within $P_{ab}$ and $P_{ac}$ are edge disjoint, they are allowed to share edges across each other. (b) Since it is cost driven, the cost of an added edge is set to zero in Step 6, and the algorithm tries to maximize the usage of edges which improves the quality of the solution and (c) Since the mesh graph has a very regular structure (planar grid), it is easy to identify the paths [51].

In [35], the authors propose a sensitivity-based algorithm to address the mesh wiring reduction. This work borrows from the work on *Equivalent Networks* to perturb an electrical network with no active devices in an efficient manner. To be able to calculate the sensitivities of a passive element with respect to all other elements in the network, the non-linear drivers have to be modeled with efficient and passive electrical networks. The passive models of the clock buffers are added to the electrical network of the mesh, and the optimization proceeds to decide which segments (resistance elements) can be eliminated while still honoring the design constraints.

## 6.5 Simulation-based Mesh Synthesis

The optimization of the clock mesh requires a delay-model that is amenable to efficient optimization. The desired delay model should be reasonably accurate and efficient. Due to the non-linear behavior of the clock drivers, it is not possible to carry out the optimization with a simple linear or quasi-linear model. Such an efficient model lacks the accuracy desired in designing and optimizing the most important signal in the design next to the power and ground.

Previous works have modeled the drivers with two-pole models to simplify the analysis process. However, such models either lacked enough accuracy or were too hard to integrate in an optimization loop. In our work, we decided not to simplify the

delay model at all but rather use spice-based models for maximum accuracy. We relied on the fast-spice engine to carry out the analysis in an efficient manner. Fast-spice engines make maximum use of partitioning of the circuit and analyze only parts of the circuit that are active at a certain time-step. These features make the analysis time two to three orders of magnitude faster than traditional spice simulators. Also, the resulting efficiency enables us to carry out the simulation of a large circuit composed of thousands of non-linear devices and 100Ks nodes which are typical of a clock mesh circuit. In addition, this flow will enable us to carry out simultaneous simulation of the clock and power/ground networks if desired [39].

Although the results of the previous works are preliminary, we mentioned them because we believe that they show promise in aiding better research of this problem. However, the above mentioned algorithms assume instantaneous switching of the clock buffers and rely on simple delay models. Hence, these algorithms do not capture the non-linearity of the mesh circuit, and thus do not provide guarantees in terms of predictability and robustness.

Figure 6.2 shows the different components of a clock mesh. A mesh is composed of a set of horizontal and vertical wires, a set of clock drivers to drive the mesh, and a set of mesh receivers. The mesh receivers in our flow are either a set of local clock drivers that drive the latch clusters, or existing clock-gating elements that were introduced earlier in the flow for power optimization.

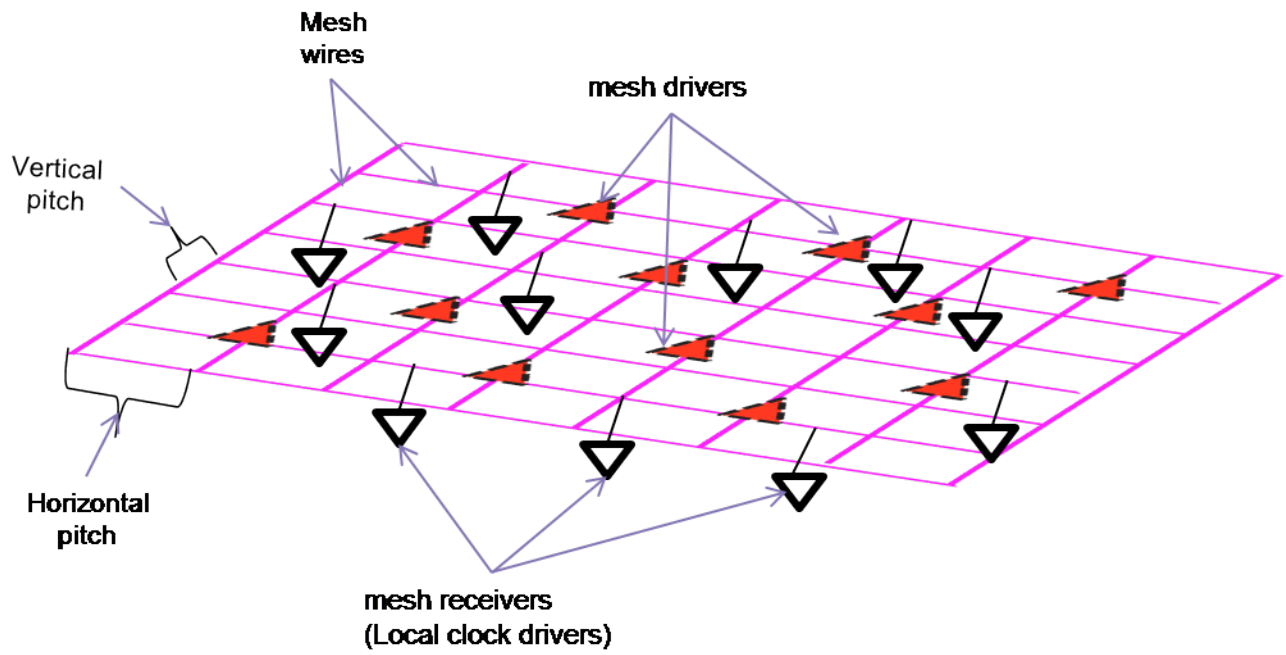



Figure 6.2. This figure shows the different components of a clock mesh. The mesh drivers are shown with dashed borders, while the mesh receivers are shown with solid black borders. To design a clock mesh, the locations of the mesh receivers should be given. The design of a clock mesh involves deciding on the number of clock drivers needed to drive the mesh, and the placement of these clock drivers. It also involves deciding on the horizontal and vertical pitches of the mesh.

In our work, we rely on an iterative simulation-based algorithm to decide on the number of the clock buffers and their placement. Also we compute the required mesh pitches (horizontal and vertical) needed to meet the skew and slews constraints. We do not do any mesh wire reduction in our work. Algorithm 4 describes the mesh synthesis algorithm used in this work. Algorithm 4 requires placement of the local clock drivers (leaves of the mesh) to be done at this stage. In Chapter 7, we will explain how we decide on the number and placement of the local clock drivers which in our flow drive the latch clusters. The decision on the placement of the local clock drivers relies on the relative and not the exact locations of the clock latches. We will show in Chapter 7 that once the local clock drivers that drive the latch clusters are

placed, they will be marked as fixed and will not move for the rest of the flow. The placement of the latches in the clusters are allowed to move as we will show later.

Algorithm 4 iteratively places a clock buffer template (set of clock buffers), connects it to the mesh (logical connectivity of the output pins of the buffers to the existing set of pins on the mesh net), routes the output pins of the clock buffers to the nearest horizontal or vertical mesh segments, extracts the resulting electrical network composed of the mesh drivers driving the mesh, the mesh receivers (existing local clock drivers) that were placed and given to the algorithm, and the mesh wires that were synthesized and laid out based on the computed horizontal and vertical pitches, and analyzes the resulting network using a fast-spice engine.

**Algorithm 4** Clock Mesh Synthesis: Part 1

**Input:** Mesh horizontal and vertical pitches ($P_h$, $P_v$), horizontal and vertical routing layers, widths of horizontal and vertical routing layers, clock buffers' slew constraints ($Slews_{target}$), mesh skew constraint ($skew_{target}$), an initial template of buffers $Template_{rxc}$, and placement of mesh receivers (local clock drivers).

**Output:** A routed mesh with a set of placed and connected clock drivers such that the mesh skew and slew constraints are met. Analysis of whether the slews and skew constraints are honored is carried out using Fast-spice simulation.

1: $row$ = Row($Template_{rxc}$) // returns the number of rows, $r$, in the template.

2: $col$ = Col($Template_{rxc}$) // returns the number of columns, $c$, in the template.

3: $Template_{current} = Template_{rowxcol}$;

4: $done = false$

5: **while** $done == false$ **do**

6: Synthesize and lay out the horizontal and vertical mesh wires.

7: Place the mesh drivers using buffer template $Template_{current}$ and connect them to the mesh net (Figure 6.1).

8: Complete the routing of the mesh by connecting the mesh drivers and mesh receivers (local clock drivers) to the nearest mesh segments. // special router is used

9: Extract the parasitics of the routed clock mesh and construct a spice circuit for the mesh that is composed of the mesh drivers (clock buffers), the mesh receivers (local clock drivers), and the parasitics (RC network) of the mesh wires.

10: Compute mesh skew and slews at the inputs and outputs of the clock cells (buffers and latches) using fast-spice simulation (Figure 6.4).

11: **if** $Skew_{mesh}$ and $Slews_{mesh}$ meet budgets **then**

12: $done = true$

13: **else**

Part 2 of Algorithm 4 continues on the second page.

**Algorithm 5** Clock Mesh Synthesis: Part 2 of Algorithm 4

14: **if** $Slews_{mesh}$ violate $Slews_{target}$ **then**

15: $row = row + 1$; $col = col + 1$; // Increase the number of clock buffers (mesh drivers) by incrementing the rows and columns of the buffer template.

16: $Template_{current} = Template_{rowxcol}$

17: **end if**

18: **if** $Skew_{mesh}$ violate $Skew_{target}$ **then**

19: $P_h- = \delta_h$; $P_v- = \delta_v$ // Reduce mesh wire pitches by $\delta_h$ and $\delta_v$; we use $\delta_h = \delta_v = 50\mu$ microns

20: **end if**

21: $done = false$

22: Remove the existing mesh.

23: **end if**

24: **end while**

Algorithm 4 relies on the following data to be provided by the designer: 1) initial buffer template (explained blow), initial horizontal and vertical mesh pitches, horizontal and vertical routing layers of the mesh and their widths, and the skew and slew constraints. The slew constraints are usually defined in the timing library of the design.

Algorithm 4 requires the locations of the mesh leaves (local clock drivers). Our proposed methodology automatically places the local clock drivers as will be explained in the next chapter.

Below, we explain the objectives of the different steps outlined in the Algorithm 4.

In Steps 1-3, the current buffer template is set to the initial template provided by the user. A buffer template is an array *row*x*col* such that *row* denotes the number

of horizontal divisions of the chip die area, and *col* is the number of vertical divisions of the chip die area. For example, if the buffer template provided is 2x2, then the die area is divided into four (2x2) bins, and a clock buffer is placed at the center of each bin. If the buffer template is 2x3, then the die area is divided into six (2x3) bins, and a clock buffer is placed at the center of each bin (Figure 6.3).

In Step 4, A Boolean flag *done* is set to *false* to indicate more iterations are needed.

In Step 5, Algorithm 4 iterates until the designed mesh meets the skew and slew constraints set by the user.

In Step 6, a set of horizontal and vertical wires are laid out. When placing the mesh wires based on the user-provided pitches, attention is made to make sure that the wires are placed properly such that no electric shorts are created with other pre-routes in the design. Pre-routes can be either power, ground, clock, or any other global signal.

In Step 7, the current buffer template is placed and logically connected to the clock mesh.

In Step 8, detailed routing of the mesh is carried out. This entails connecting the output pins of the placed mesh drivers to the nearest mesh segments. Also, the input pins of the mesh leaves (local clock drivers) are connected to the nearest mesh segments. It is important to keep wire length of these segments as short as possible to reduce their resistance and capacitance. This in turn will reduce any degradation in the mesh slews at the outputs of the mesh driver and at the inputs of the mesh receivers (local clock drivers).

In Step 9, parasitics extraction of the mesh wires is done, and a fully connected electrical network composed of mesh drivers, mesh receivers, and mesh wires is built. A spice circuit is constructed to be passed to the simulation engine.

In Step 10, the fast-spice engine is called to compute the mesh skew and the slews of the waveforms at the outputs of the mesh drivers and the inputs of the mesh receivers.

In Step 11, a check is made to see if the constructed mesh honors the skew ($Skew_{mesh}$) and the slew ($Slews_{mesh}$) constraints.

In Step 12, the Boolean flag *done* is set to *true* if the constraints are met. If by the end of this iteration the Boolean flag *done* is not set to *false*, then the algorithm terminates.

If the constraints are not met, then the algorithm proceeds to step 14. In Steps 14-16, a check is made if the slew constraints ($Slews_{mesh}$) are violated. If they are, then the current buffer template is replaced by a bigger template. In this work, we replace the current $row$x$col$ template by $(row + 1)$x$(col + 1)$ template. It is clear that if the slews are violated, either the pitch needs to be increased (makes the mesh more sparse) which might cause the skew to worsen, or the buffer count needs to be increased which would improve the slews and will not worsen the skew.

In Steps 18-20, a check is made if the mesh skew is violated. If so, then it is likely that the mesh is not dense enough. We choose to increase the mesh density by reducing the pitch by a $\delta$. In this work, $\delta$ is set to 50 microns.

In Step 21, *done* is set to *false* to indicate that another iteration is needed.

In Step 22, the current mesh is removed, and the algorithm starts again with either a new buffer template if the slews constraints were violated, or with a new pair of mesh pitches (horizontal and vertical) if the mesh skew is violated.

At the end of this algorithm, we will converge on a mesh design defined by a chosen buffer template and a pair of horizontal and vertical pitches. The clock buffers (mesh drivers) are placed and the whole mesh network is fully routed, simulated, and

annotated with the slews and arrival delays at the outputs of the mesh drivers and the inputs of the mesh receivers.

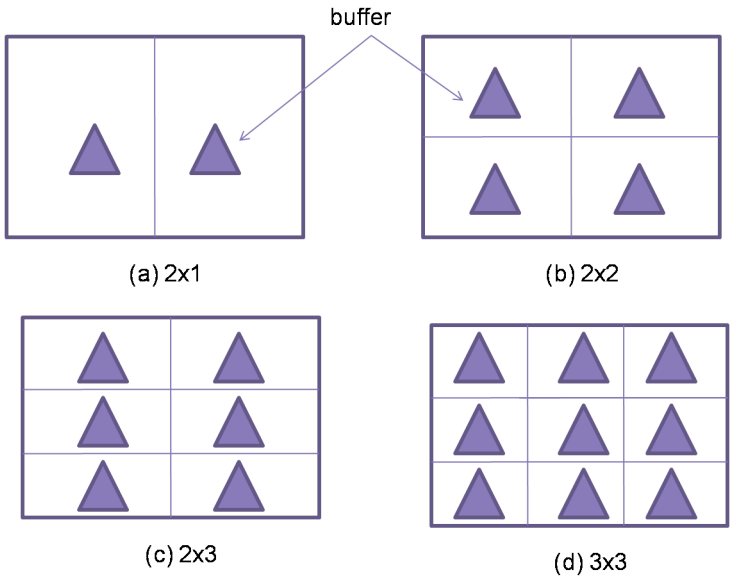



Figure 6.3. Clock buffer templates used in Algorithm 4. A template is an array of *row*x*col* buffers such that the die area is partitioned into *row* horizontal divisions followed by *col* vertical divisions. In figure (a), a 2x1 template is shown which partitions the die area horizontally into 1 partition (means no horizontal partitioning) followed by partitioning the die area into two vertical partitions . This creates only two partitions and a buffer is placed at the center of each partition. In figure (b), a 2x2 template which partitions the die area into two horizontal partitions followed by 2 vertical partitions; a buffer is placed at the center of each of the four partitions. In figure (c) a 2x3 template is shown that results in six buffers placed on the die area. In figure (d) a 3x3 template is shown which results in placing nine buffers on the die area.

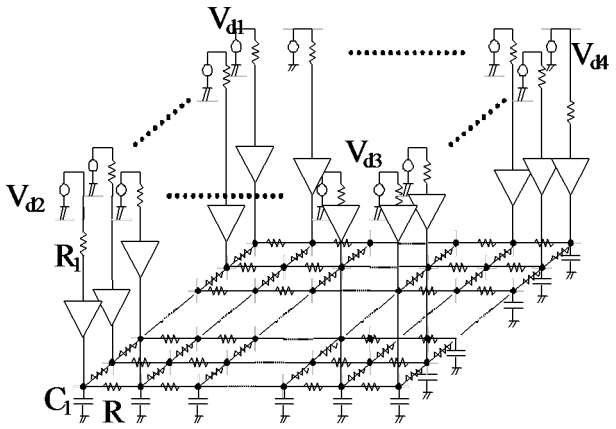



Figure 6.4. RC-network for the mesh. This network is composed of mesh drivers (clock buffers), mesh receivers (local clock drivers that drive the latch clusters), and the parasitics of the mesh wires. In this figure, the mesh receivers are modeled as decoupled capacitors to ground. Analysis of this network is carried out by a fast-spice engine.

## 6.6 Clock Mesh Tuning

In our work, the decision of sizing the clock buffers is not automated and left up to the designer. Algorithm 4 increases the number of buffers of the same size to meet the slews constraints, or reduces the horizontal and vertical pitches of the mesh wires. In Algorithm 4, the DRC-clean layout of the mesh wires (ensuring no shorts or opens as a result of laying out the mesh wires), the detailed placement of the clock buffers (to ensure no overlaps between the added clock buffers and the existing cells in the design), and the detailed routing of the entire clock mesh are automated. Once the mesh is designed, extraction and circuit simulation of the mesh network are carried out. At the end of the analysis, all timing annotations (arrival times and slews) are annotated on the input and output pins of the mesh cells (drivers and receivers).

Algorithm 4 limits itself to the buffer size given by the user. Also, this algorithm does not change the widths of the mesh wires. However, the designer may choose

to tune the clock mesh further or carry out what-if analysis by studying the QoR of the clock mesh by using a different buffer templates, adjusting the mesh pitches, changing the mesh routing layers, or adjusting the widths of the mesh wire segments. Based on the simulation results, the designer, for example, may increase the number of clock drivers to reduce the skew or improve the slews at the inputs of the mesh receivers. Alternatively, the analysis may show that the mesh is over designed, and a reduction of mesh wiring or buffers would reduce the consumed power without sacrificing performance. This process relies on the designer to draw on his past design experience and tune the clock mesh for satisfactory results. The designer can do what-if analysis or tune the clock mesh in one of two different ways:

1. The designer can do what-if analysis by providing Algorithm 4 with different buffer sizes, different wire widths, or different initial buffer template and pitches. This allows the designer to explore different mesh structures using different provided parameters.

2. The designer can use Algorithm 4 in an incremental mode. In this mode, the designer incrementally modifies the mesh designed by our algorithm. In this mode, the designer may choose to add some mesh wires in areas where the arrival delays or the slews at the inputs of the mesh receivers are unsatisfactory. The designer may also choose to add more buffers to beef up the driving strength of the mesh in certain areas of the design. Algorithm 4 utilizes a toolbox of functions as it carries out the mesh design. This toolbox contains functions to add mesh wires, to route the mesh network or to connect new buffers (drivers) to the mesh net. The designer has access to this toolbox to improve or tune the designed mesh.

Prior work showed that there is an exponential decay of the waveforms as they move away from the clock buffers to farther nodes on the mesh [11]. This factor

necessitates the placement of buffers close to the sinks. Buffers driving far mesh nodes that do not have near-by sinks is a waste of resources and power.

We emphasized that one of our main objectives is to build a reliable and robust top-level clock network to enable the clock driven design methodology to converge smoothly. This was the motivation for adopting the algorithm and techniques presented above. We will study other algorithms that tackled this problem and show promise in optimizing clock meshes as part of future work.

### 6.6.1 Mesh Shielding

It is critical that the clock network be shielded from any source of noise. As mentioned earlier, not only is the clock the biggest contributor to noise in the design due to the high rate of switching, but also it is the most sensitive signal since it is the gateway for any data transfer in the design.

The best way to shield the clock network is to concurrently design it with the power network, and have the clock wires be shielded by the neighboring power wires. This would enable reliable power design as well as robust clock. By favoring the placement of the power straps (wires) close to the clock straps (for mesh or other regular clock topologies), the clock wiring will be shielded from the neighboring aggressors.

## 6.7 Mesh Verification

The electrical verification of the clock mesh is carried out using a fast-spice simulator which can handle the non-linearity in the circuit. For uncompromised accuracy, a decision was made not to use any linearized or any other efficient close-form delay models. All work is carried out with detailed spice models. The use of the fast-spice simulator allowed us to enable what-if analysis and achieve good turn-around

time with excellent accuracy. We achieved insertion delay accuracy of within 1% as compared to spice results.

## 6.8 Summary

In this chapter, we elaborated on the design factors affecting clock meshes. We presented some promising prior works in this area that rely on simple delay models and ignore the non-linearity of the mesh circuit. Since one of our stated objectives of this work is skew predictability, we opted for a simulation-based approach that provides more accuracy and predictability. Our mesh synthesis algorithm (Algorithm 4) requires knowing the location of the local clock drivers that drive the latch clusters in the lowest level of the hybrid clock network. Once the local clock drivers are placed, the design of the clock mesh commences. Algorithm 4 relies on the designer to provide the initial buffer template and the widths of the mesh wires. It also relies on the designer to decide on which routing layers to use for the mesh implementation. Algorithm 4 does not automatically adjust the sizes of the buffers. It uniformly increases the size of the buffer templates (increases the number of inserted buffers) by incrementing the number of rows and columns in the buffer template by 1 if needed, (for example, it moves from template 2x2 to 3x3 if more buffers are needed). Although only either of the number of the rows or columns in the buffer template can be incremented at a time (for example moving from a template 2x2 to a template 2x3), Algorithm 4 does not take such a step, and it increments the *row* and *col* of the buffer template. At the end, the algorithm will converge on a mesh design that might consume more power and more area than an optimally designed mesh. In future work, we will tackle the problem of mesh optimization while employing more accurate delay-models and accounting for the non-linearity of the mesh circuit.

The contribution of Algorithm 4 is the automation of the decisions on the number

of the clock buffers needed to drive the mesh, and the horizontal and vertical pitches of the mesh. Our algorithm also automates the physical implementation of the clock mesh. The designer can utilize the toolbox of functions used by Algorithm 4 to further tune of the mesh or to carry out what-if analysis to explore different mesh structures using different design parameters.

The outcome of Algorithm 4 is a fully designed and laid out clock mesh with all timing annotations needed to build the top-level clock tree that is needed to drive the clock mesh. Since the mesh drivers are the leaves of the top-level clock tree, by deciding on the number of the mesh drivers and their locations, we would have put in place the needed components to design the top-level clock tree. Once the top-level clock network that drives the mesh is in place, the entire clock network is designed and laid out. This clock network will drive the implementation of the rest of the design as will be explained in the next chapter.

# Chapter 7

# Clock Driven Planning: Shells & Cores

## 7.1 Introduction

In high-speed designs, more than 40% of the total power is dissipated in the clock network [15]. In addition, a 10% IR drop translates to 8% degradation in timing accompanied with reliability issues due to on-chip variations [39],[48]. These problems lead to numerous design iterations and make it harder to attain the required quality of results (QoR) in a reasonable turnaround time. These factors necessitate a fresh look at the adopted design methodology especially with respect to designing reliable and predictable power and clock networks.

Since the clock network is the most frequently switching signal in the design, it consumes the most power. Extensive research and design efforts have focused on reducing the power consumption of the clock network as well as the noise induced as a result of the clock switching. Furthermore, with every new process technology generation, the design planning and physical synthesis tasks get harder and design

convergence gets worse. These problems lead to longer turn-around times and degrade the quality of results. In high-end ASIC designs, the design of a reliable and low-skew clock distribution network has become one of the most difficult tasks.

The previous chapters have been building up to this point so that we can present our proposed clock-driven design planning methodology in detail. After an introduction and an overview of the synchronous design paradigm in Chapter 2 and the discussion of the RTL-to-GDSII design flow in Chapter 3, we discussed some of the issues surrounding the design of the clock and power networks in ASICs. In Chapter 4, we elaborated on the issues and challenges that affect the design of the power network. In particular, we focused on issues related to the design and integrity of power and their impact on the clock network design. In Chapters 5 and 6, we elaborated on clock network design. We presented different clock topologies and expounded on algorithms and techniques for the design of a robust and low power multi-GHz clock network.

With the exception of one key topic, we have so far put in place the needed components to present our clock-driven design planning (CDP) methodology (Figure 7.1). The missing topic is how we place the registers prior to the implementation of the rest of the design. We have mentioned earlier that the placement of the registers is key to enabling the design of the clock network. We also explained why we propose the design of this clock network early in the design cycle and not wait until after cell placement, as is done in today's ASIC RTL-to-GDSII design flow. We also expounded on the tight relationship between the power network design and the clock network design.

This chapter presents the needed algorithms to design and plan a clock network prior to the physical optimization and placement of the design. In doing so, it designs a hybrid top-level clock network, and places the latches prior to the placement of the

logic cells. Our CDP methodology distinguishes between long wires and short wires during the placement of the latches. The objective of our methodology is to reduce the clock skew and increase the predictability and robustness of the clock network. It is important to emphasize that once the clock network is designed and implemented, later stages in the design flow will not affect the clock network performance and characteristics. This predictability carries over to a better design flow during physical optimization, placement and routing for the rest of the design. We ran this flow on three industrial designs and improved the skew between 10% and 20%. The degradation in wire length was less than 5%.

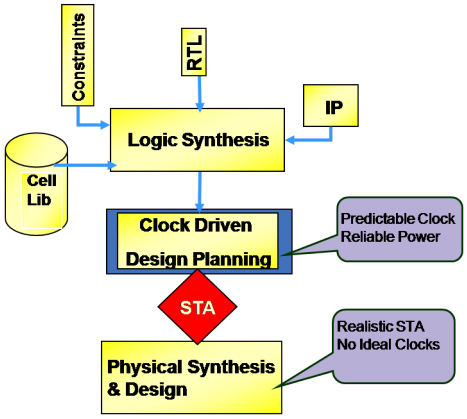



Figure 7.1. Clock Driven Design Flow. In this figure, clock design is carried out during the floorplanning stage. Tackling the clock design this early in the flow makes the clock design drive the rest of the physical synthesis and implementation stages. All optimization decisions made after the design of the clock network are carried out with accurate and detailed information about the clock latency and clock skew.

## 7.2 Prior and Current Work

Extensive research and design efforts have focused on designing a robust clock network which exhibits low skew, consumes less power, and reduces the clock-induced noise on the power grid. These efforts range from synthesizing clock-gates to stop

part of the clock to physically optimizing and tuning the clock network by sizing the clock drivers and reducing the clock wiring.

Traditional cell placement focused on reducing wire-length [27] and improving design timing [19]. The most important point to highlight about cell placement with respect to clock design is that cell placement algorithms make no distinction between logic cells and clock cells. To a placement engine, both clock and logic cells are points in space, and the main objectives are to reduce wire-length and improve design timing. Since most of the power in the clock network is consumed in its last stage where local clock drivers or clock gates drive clusters of latches, skew and power consumption can be improved by proper sizing of the clock gates and controlling their placement as well as the placement of the clock registers. However, this could have an adverse effect on the placement of the logic cells since registers are tightly coupled to logic cells.

Earlier work to tackle this deficiency focused on taking skew into account during placement [24]. In [52], a clock-driven placement algorithm was formulated to have the global placement take into account an independently designed and balanced clock tree.

Previous works on clock wire-length optimization were mostly concerned with the clock routing stage [25], or exploiting permissible bounds on clock skew [49]. Since clock routing depends on the placement of the clock registers, the register placement may greatly affect clock network wire-length. For a poor register placement, clock wire length reduction can be very limited. This is particularly true for prescribed skew routing that is popular for the sake of timing, SI, and power supply noise improvement.

In regards to early design and planning of the clock network, previous work focused on estimating the clock tree and its impact on the floorplanning of the design [56],[23]. In [23], a hierarchical clock planning methodology is proposed where a bal-

ancing algorithm based on Deferred-Merge and Embedding (DME) [17] is used to synthesize the top-level clock tree. The block-level clock networks are assumed to be H-tree topologies (Figure 7.2). To optimize the floorplan along with the clock design, Simulated Annealing is employed in two phases. In the first phase, the algorithm uses a cost function to reduce the chip-area and total wire-length (Equation 7.1). Once the initial floorplan is generated, the clock network is constructed, and the algorithm is executed again with a different cost function to take into account the cost of the clock network. The goals of the second phase are to reduce design area, logic wire-length, and clock wire-length (Equation 7.2).

$$Cost1 = Area + \alpha * Wire \tag{7.1}$$

$$Cost2 = Area + \alpha * Wire + \beta * clk\_tree\_depth + \gamma * clk\_tree\_wl \tag{7.2}$$

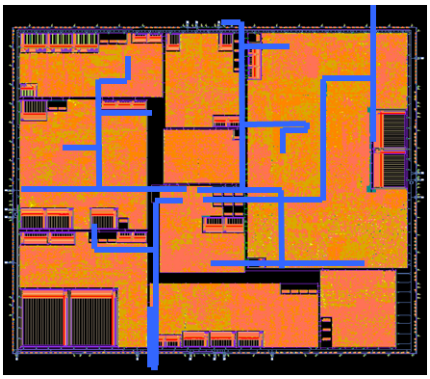


Figure 7.2. Simulated-annealing based floorplanning with clock tree estimation. This figure shows a floorplan with a top-level clock tree driving a set of blocks. In the first phase of the floorplanning, Simulated Annealing is carried out for top-level placement of the blocks without any estimation of the clock network. In the second phase of the floorplanning, a DME-based synthesis of the top-level clock tree is done to drive the clock pins on the blocks. Simulated Annealing is carried out again by adjusting its cost function to reflect the cost of the synthesized top-level clock tree. In this work, the lower-level clock networks of the blocks are assumed to be H-trees.

In [26], register placement is performed together with the standard cell logic in

a quadratic placement framework so that any negative impact on the traditional placement objectives can be minimized (Equations 7.3, 7.4). The idea is to place the clock latches along a Manhattan circle that covers the design. This will force all the clock registers to be equidistant from a common clock source that is assumed to be in the middle of the chip (Figure 7.3). In the recursive partitioning process, the center of gravity of the registers in each bin is constrained independently to lie along the edge of the Manhattan circle in that partition (Equation 7.5).

$$Min: \quad \Phi(X) = 1/2X^T CX + d_X^T X \tag{7.3}$$

$$s.t. \quad A_f^{(m)} X_f = u_f^{(m)} \tag{7.4}$$

$$A_g^{(m)} X_g = u_g^{(m)} \tag{7.5}$$

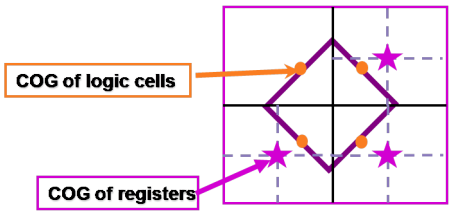



Figure 7.3. Two constraints vectors are added in the partitioning step of the Gordian-style formulation of cell placement. The first vector reflects the centers-of-gravity of the logic cells while the second one reflects the gravitation of the registers to the centers of gravity of the Manhattan circle segment in that partition [26].

A graph-based skew scheduling algorithm is performed during the placement to generate delay targets for registers and navigate the register locations. After the quadratic placement is done, a detailed placement phase is entered to finalize the placement of the design. However, due to the non-uniform distribution of the clock cells, it is not always possible to constrain the placement of the latches along the Manhattan circles. This causes divergence from the desired skew targets.

Recently, exotic clock network designs have emerged to overcome the challenges presented. These exotic networks target high-speed designs where peak performance and reduced power are important design factors [9],[8],[7]. Further discussion and analysis of these exotic networks are beyond the scope of this thesis.

In contrast to the above described efforts which carry out placement and physical optimization before clock synthesis, our method focuses on designing the clock first, and have the clock drive the implementation of the overall design as will be detailed in the next section.

## 7.3 Clock Driven Design Planning Methodology

As design complexity grows, confidence in achieving design convergence, meeting design constraints, and closing the design in an acceptable turn-around time is weakening. With the increasing cost of manufacturing, any design iteration that causes a change in the mask set is costly.

We devised a new methodology to tackle the clock network design early in the design cycle without incurring a big increase in the overall design costs. This facilitates carrying out the physical synthesis and implementation with accurate and detailed clock annotations. Figure 7.1 shows our proposed flow whose main objective is to seed the placement and physical optimization with a clock-driven plan. The design of the clock plan distinguishes between long and short wires in the clock network. The proposed flow is different from the traditional flow, which carries out placement and physical optimization before synthesizing a clock network.

By deferring clock network design until late in the design cycle, the traditional RTL-to-GDSII flow makes a strong assumption that the physical synthesis does not impede synthesizing a clock network which meets the skew and uncertainty con-

straints. Failing to implement the clock network that meets the desired constraints late in the design cycle means more costly design iterations.

The proposed designed methodology pre-designs a clock network without degrading the overall timing and congestion of the design. In addition, it reduces the switching power of the clock network, in large part because of the controlled implementation and size of the latch clusters.

The objectives of the proposed design methodology are as follows:

- Design a predictable clock network which meets a target skew. This eliminates the uncertainty of meeting a target skew when clock network synthesis is done after the logic is placed and optimized. The objective is to design a predictable clock network and not simply to reduce clock skew, although our results show skew improvements over custom clock mesh designs. This improvement can be used to improve timing of the design or to make up for any degradation in timing as a result of the latch placement.

- Although power reduction was not the original intent, we realized that power gains can be attained by better placement of the clock latches as compared to a traditional custom mesh flow.

- Predictable skew could come at a cost in area and timing. The objective is to reduce any potential overhead associated with the pre-planned network in terms of overall congestion and timing of the design.

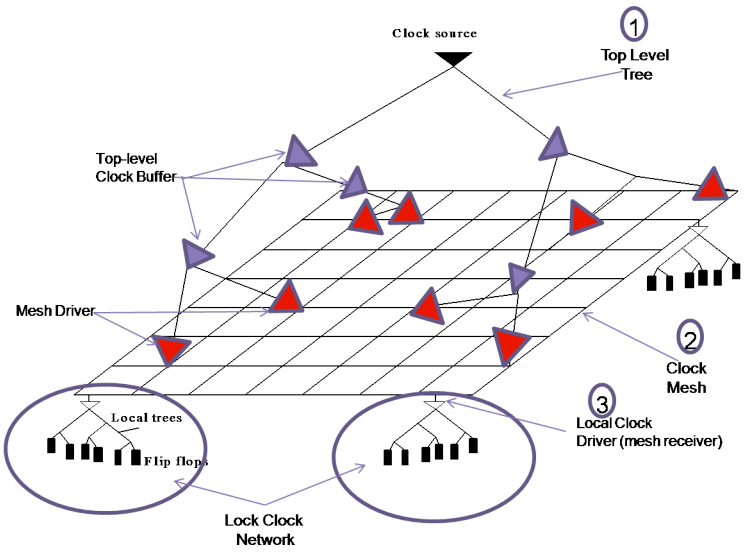



Figure 7.4. This figure shows the hybrid clock network that is built by our proposed CDP methodology. This hybrid clock network is composed of three parts. Part 1 is the top-level clock network that drives the clock mesh. This top-level clock network is either a synthesized tree topology or an H-tree. Part 2 is the clock mesh which drives the mesh. The leaves of the clock mesh are the local clock drivers which drive the latches in the clock clusters. Part 3 is the local clock network which is modeled as a 1-level tree that drives the latches in the clusters.

The hybrid clock network used in our methodology is composed of three parts (Figure 7.4):

1. Local network that is composed of the local clock trees in the latch clusters. In each cluster, a clock tree is built. The root of this tree is the root of the cluster which is the local clock driver. The leaves of this tree are the sinks in the cluster (latches). We will see later how we decide on the placement of the latches in order to place the local clock driver at the centroid of the cluster.

2. A clock mesh is the a multi-driven net. The leaves of this mesh are the local clock drivers that drive the local clock networks mentioned above. The drivers of the clock mesh will be synthesized using Algorithm 4.

3. The top-level clock tree is the tree that drives the clock mesh. The root of this

tree is the clock root defined in the overall design. The leaves of this clock tree are the mesh drivers that were designed and placed as part of the clock mesh design explained above using Algorithm 4.

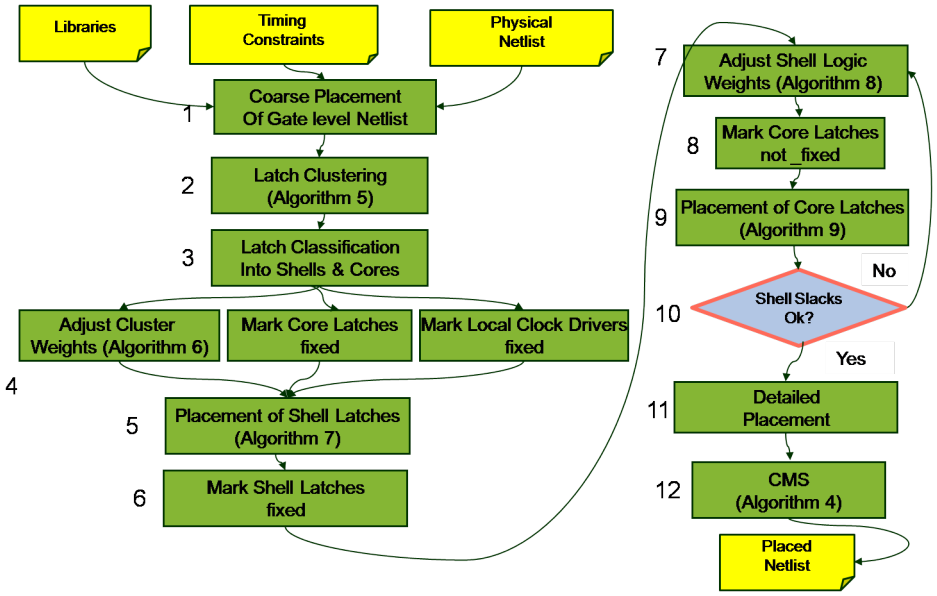



Figure 7.5. A Top-level flow of the proposed Clock-driven Design Planning Methodology (CDP). This flow shows the steps taken in carrying out the methodology. Each box is numbered so that we can easily refer to it in the high-level discussion below.

Figure 7.5 shows the steps taken to design the three different parts of the hybrid clock network proposed by our CDP methodology:

1. Coarse placement is carried out on the provided synthesized gate-level netlist. This placement provides estimates of the initial locations of the latches in the context of the entire design. Since the latches are tightly coupled to the logic cells, the initial placement allows us to identify the natural clusters of latches while taking into consideration the surrounding logic cells.

2. Latch clustering is performed based on the initial placement of the latches. The objective of this step is to identify the natural clusters in the design, and to

insert clock buffers (local clock drivers) to drive these clusters. Once we place the local clock drivers at the centroids of their respective clusters, we mark these local clock drivers as fixed, and they remain fixed for the rest of the flow. The locations of the latches will change in subsequent steps. However, since global placement is carried out on the entire design, the resulting initial locations of the latches will not change much due to their tightly-coupled logic connectivity. It is important to emphasize that once the locations of the local clock drivers are known, all information needed to design the top-level hybrid clock network is available since the local clock drivers are the leaves of the top-level clock network.

3. In order to classify the latches into two types: shells and cores, we construct a timing graph of the design. We traverse this graph starting from the roots of the clusters to classify the latches in each cluster. Although a connectivity graph can be used instead of a timing graph, that latter makes the job easier since in a timing graph, all arcs have their types marked properly. In a connectivity graph, no distinction is made between a *clock arc* versus a *data arc*. This information if available speeds up the task of the latch classification. At the end of this step, each cluster will have two sets of latches, a shell latch set, and a core latch set.

4. We will bias the placement of the latches in favor of the shell latches as opposed to the core latches since the shell latches drive the long wires. To do that, we mark the core latches as fixed based on the initial global placement carried out in Step 1. We allocate a portion of the over all clock skew budget to the latch clusters. This local skew budget defines a constraint on the placement of the latches in the cluster. We map this skew budget into an upper bound on the wire length of each cluster, and we compute a set of weights to guide the placer to honor the desired wire length in each cluster (Algorithm 7).

5. Coarse placement of the shell latches is carried out using the the computed weights in the previous step. At this point, the core latches are marked fixed based on their initial locations. The placement of the shell latches is done by setting conservative constraints on the wire length of the cluster clock nets (Algorithm 8).

6. Once the shell latches are placed, they will be marked fixed and will not change for the rest of the flow. The premise here is that since the core latches connect local logic with short wires, placing them to honor the local skew constraint will be an easier task than placing the shell latches.

7. To mitigate any poor placement of the logic cells between the shell latches (shell logic) as a result of the placement of the shell latches done in the previous step, we adjust the placement of the shell logic at the same time that we adjust the placement of the core latches (Algorithm 9).

8. Here we mark the core latches as *not_fixed*. This will free them to move around in the next placement phase. Up to this point, the core latches have been fixed in place based on the coarse placement done in Step 1. This decision is made since the core latches drive short wires inside their respective clusters, and thus the task of placing them close to the local clock drivers is easier than that of the shell latches. The core logic between the core latches is local logic and is capable of pulling the core latches closer to the roots of the clusters. In addition, the core latches are connected to the shell latches which have been placed as discussed above. This will add more forces on the core latches to be pulled to the clusters should they wander far from the roots (local clock drivers).

9. Coarse placement of the core latches and the shell logic commences based on the set of weights computed above. It is important to emphasize that the core latches are tightly coupled to the shell latches in their respective clusters. This

means that as we negotiate the placement of the core latches in such a way to improve the placement of the shell logic, the core latches will not wander away from the roots of the clusters (local clock drivers) (Algorithm 10).

10. At the end of each placement phase, we will check for timing violations. If they exist, we will adjust the weights of the violated nets and iterate over the placement of core latches and the shell logic (Steps 7-10) until we converge.

11. At this step, the core latches and the shell latches are placed and marked as fixed. Detailed placement of the design is carried out. The exact local skew achieved in the clusters is known. The design of the top-level clock network commences based on Algorithm 4 explained in the previous chapter.

12. Once the entire clock network is designed, detailed routing of the clock is done, and circuit simulation of the clock network takes place to annotate the clock network with the arrival times at the slews at the inputs and outputs of all clock cells (buffers, latches).

At the end of the flow proposed above, it is important to emphasize that the designed clock network along with the placement of the latches will not change as a result of the succeeding physical synthesis steps. In fact, it is this clock network and the placement of the latches that will drive the rest of the physical synthesis and implementation of the design. The detailed and accurate timing annotations of the clock network will provide the succeeding physical synthesis steps with the needed information to optimize the design and reduce the need for any worst case decisions. By designing the clock network early in the design cycle, our CDP methodology facilitates a more robust and convergent design implementation. In the following sections, we will explain in details the key steps of our flow (Figure 7.5) that were highlighted above.

### 7.3.1 Coarse Placement

Our proposed methodology starts by carrying out a coarse placement of the design. This step provides us with information about the geographical distribution of the latches in relation to other latches and logic cells. We are not interested in any optimization or detailed placement at this time. We are only interested in information that enables us to extract the natural clusters of the latches in the design. This step is very fast and does not consume more than few minutes on large designs.

### 7.3.2 Latch Clustering

It is our goal to design the clock network without being limited by the placement of the logic cells and registers that takes place late in the design cycle. Waiting until then removes many degrees of freedom in optimizing the clock network and may cause negative effects on timing and convergence of the design. However, the design of the clock network requires knowing the locations of the registers in the design and thus some way of finding or estimating the locations of these registers is needed.

Although it is possible to have the top-level clock network drive the latches directly, this would lead to a huge cost in terms of power and routing. In our flow, we opt to cluster the latches based on a coarse global placement. Then, based on the cluster characteristics, we select a set of local clock drivers with the proper drive strength to drive the clusters. We employed a clustering method based on the well-known *K-Means* clustering algorithm which is popular in statistical applications. We chose this algorithm for its simplicity, but other clustering algorithms can be employed. Since the location of the latches will change after we place the local clock drivers, we are interested in the global picture of where the local clock drivers should be placed and not in the exact location of the clock latches.

Algorithm 6 shows a pseudo code of the *K-Means* algorithm. The algorithm attempts to find the centers (means) of the natural clusters in the design. The objective is to minimize the total intra-cluster variance, or the squared error function as shown in Equation 7.6.

$$V = \sum_{i=1}^{K} \sum_{x_j \in S_i} (x_j - \mu_i)^2 \tag{7.6}$$

where $K$ denotes the number of clusters, and $\mu_i$ denotes the mean or centroid of the elements $x_i$ in cluster $S_i$.

To cluster the clock latches, an estimate is made of the wiring capacitance as a fraction $\beta$ of the load capacitance seen by a clock driver. Thus, for a local clock driver whose driving capacity is $C_{max}$, the number of local clock drivers, $K$, needed to drive $n$ latches, each with $C_g$ gate capacitance, can be computed (Equation 7.7).

$$K = \frac{n * C_g * (1 + \beta)}{C_{max}} \tag{7.7}$$

**Algorithm 6** Estimate of number of Local Clock Drivers needed to drive the mesh

**Input:** Placement of latches

**Output:** $K$ clusters of latches

1: Estimate number of clusters $K$ based on Equation 7.7
2: Pick latches to seed locations of means of clusters $S_{1..K}$ // means are centers of gravity of clusters
3: **while** means $\mu_{1..K}$ change **do**
4:  **for** each latch $l_i$ **do**
5:  Find nearest mean, $\mu_j$
6:  Move $l_i$ to $S_j$ // $\mu_j$ is mean of cluster $S_j$
7:  **end for**
8:  Re-compute means $\mu_{1..K}$ of the clusters $S_{1..K}$
9: **end while**

Keeping in mind that our objective is to re-place the latches to control (reduce) the skew, we are only interested in a reasonable partitioning of the latches. A more sophisticated partitioning algorithm can be used, but the $K-Means$ based algorithm is sufficient to capture the distribution of the latches as a result of the coarse placement done earlier in the flow.

## 7.4 Insertion of Local Clock Drivers

Once the natural clusters in the design are identified, clock buffers are inserted at the roots of these clusters. In some cases, clock-gating cells (clock gates) could already be present in the design. In that case, those clock gates are the drivers of their clusters. If there is any DRC violation (slew, load) for any cluster driven by a clock gate, then the clock gate is cloned and the cluster is partitioned further.

This step enables us to design the top-level clock network before cell placement. The placement of the latches will change later; however, the placement of the the local clock drivers will remain fixed throughout the rest of the flow.

## 7.5 Latch Classification: Shells & Cores

As designs get bigger, the number of latches increases. The large capacitance associated with the latches and the routing of the clusters puts a huge burden on the last stage of the clock network (local clock drivers). In our flow, the skew of the pre-designed hybrid top-level clock network down to the local clock drivers is designed, characterized, and fixed. This network is locked in so that it is not affected by placement or any other optimizations. The objective now is to place the latches driven by the local clock drivers in such a way that we minimize skew in each cluster, meet transition constraints at the outputs of the local clock drivers, reduce switching power, and reduce the impact of the pre-planning of the clock network and the latches on the placement and routing of the rest of the design. To accomplish this goal, we partition the design into two sets of latches: shells and cores (Figure 7.6).

- *Shell latches* are those which connect different latch clusters together. These latches are the interface elements which connect through some shell logic and wiring to shell latches of different clusters. The wiring which is associated with the shell logic could span long distances on the chip. Proper implementation of the shell latches and shell wires results in better skew and reduces the impact of the clock design on the rest of the logic. Pre-planning the shell latches to minimize any skew while still optimizing the long wires is a primary objective of this section.

- *Core latches* are those which only connect to logic inside each cluster. We can

infer from this locality that the wires inside the clusters are short. Therefore we can rely on the optimization engine to handle any potential timing problems affecting these local nets that could be caused by the pre-placement of the core and shell latches later in the flow.

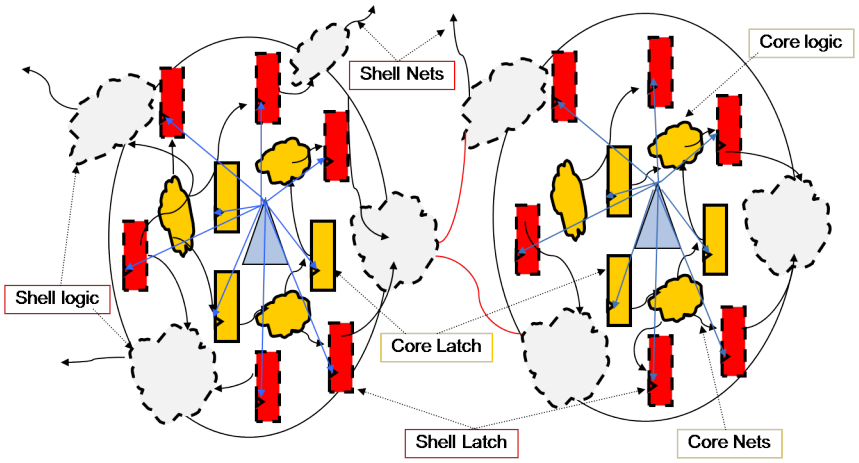



Figure 7.6. Partitioning of latches into shells and cores. This figure shows two latch clusters. At the center of each cluster is a local clock driver which drives all the latches in that cluster. Each cluster has two sets of latches: a shell latch set, and a core latch. A shell latch is shown with a dashed border, and a core latch is shown with a solid border. The clouds in the figure denote combinational logic. There are two kinds of logic shown: shell logic and core logic. Shell logic is the combinational logic that connects different clusters together (logic between the shell latches). The shell logic clouds are shown with dashed borders. Core logic is the combinational logic that connects latches inside a single cluster (logic between shell and core latches inside a single cluster). The core logic clouds are shown with solid borders).

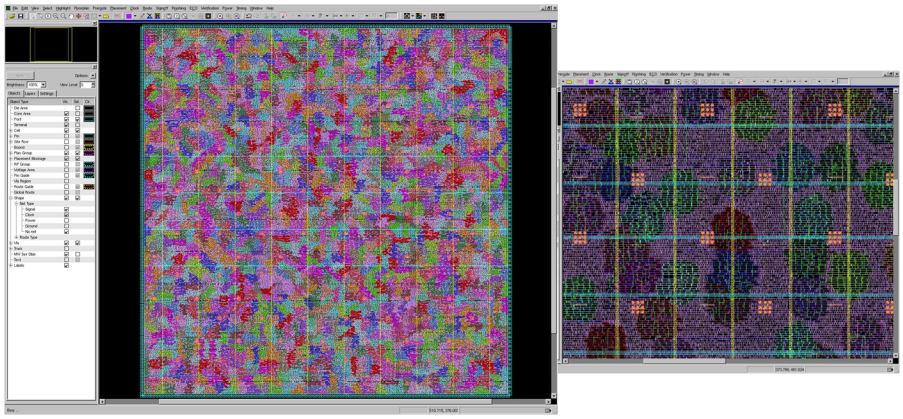


Figure 7.7. This figure shows the distribution of shells and cores in an actual design. This figure shows a colored map of the shells and cores of the various clusters in the design after global placement. Two colors are assigned for each cluster to differentiate between the shell and core latches. Given the large number of the clusters in the design, the colors are recycled to color all the clusters. The figure on the right is a blow-up of part of the figure on the left to highlight the shell/core distribution.

## 7.6 Latch Placement

In the proposed methodology, the shell latches are placed prior to the core latches. Since we are not interested in detailed placement at this stage, any time we need to do cell placement, we rely on the global (coarse) placer which is very efficient and provides the global information needed to place the latches. In these placement steps, the linear placement engine we use is biased to reflect whether the shells or the cores are being placed (Figure 7.7). The placement bias is computed based on the relationship between the net weights and the wire length as discussed next.

### 7.6.1 Weight Computation for Cluster Nets

Tsay and Koehl showed the relationship between the weight increase (Lagrangian multiplier) and the wire length increase in placement [50]. Instead of computing

the exact Lagrangian multiplier which requires the computation of the inverse of the connectivity matrix $A$, they formulated an efficient model (Equation 7.8) which estimates this relationship. This model is efficient enough to be used in the inner loop of P&R. This model enables the adjustment of the weights on the clock nets that connect the local clock drivers to the shell latches. Also, this model can be applied to bias the placement of the shell logic to reduce the wire length of the long interconnects that connect the clock clusters together.

$$\Delta W = \frac{-\Delta L/L + \Delta L}{1/W_{src} + 2/W_{sink} - 2W/W_{src}W_{sink}} \tag{7.8}$$

For a given global skew constraint for the entire clock network, we can allocate a portion of this skew budget ($Skew_{local}$) to the clock nets in the latch clusters which are at the last stage in the clock network. In order to place the shell and core latches such that the skew in the clusters does not exceed the budgeted $Skew_{local}$, we map the skew budget $Skew_{local}$ into a wire length $L_{new}$ using Equation 7.9. For a given 2-pin net with wire length $L$, we can calculate the change in wire length needed ($\Delta L = L_{new} - L$) to meet the desired wire length. We then map $\Delta L$ into a weight adjustment $\Delta W$ (Equation 7.8) so that we can guide the placement of the cells on this net.

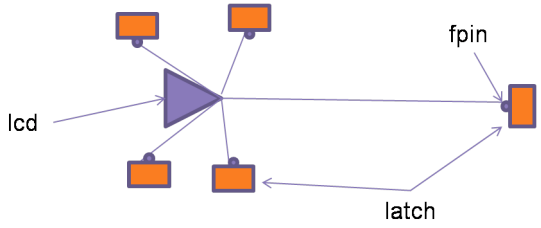



Figure 7.8. This figure shows a local clock driver (root of cluster) connecting the different latches in the cluster using a star-route model.

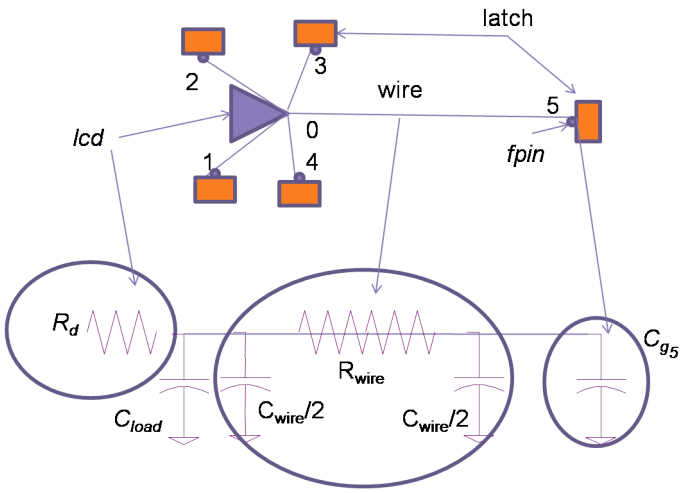



Figure 7.9. RC network of the edge connecting the local clock driver to the farthest latch in the cluster. In this figure, the local clock driver is modeled with an on-resistor $R_d$, $C_g$ is the the pin capacitance of $fpin$ (input of latch), and the connecting wire is modeled with a $1-\pi$ rc segment

$$D = R_d C_{load} + \frac{rc}{2} L^2 + rLC_{load} \tag{7.9}$$

Equation 7.9 shows the Elmore delay model we use to compute the delay of a net where D is the delay of the net, $r$ & $c$ are wire resistance and capacitance per unit length, $R_d$ is the driver's on-resistance, $C_{load}$ is the load capacitance, and $L$ is the length of the interconnect (Figure 7.9).

$$\frac{rc}{2} L^2 + rC_{load} L + R_d C_{load} - D = 0 \tag{7.10}$$

$$K = R_d C_{load} - D \tag{7.11}$$

$$L = \frac{\frac{-rC'_{load}}{2} + \sqrt{(\frac{-rC'_{load}}{2})^2 - \frac{rc}{2} * K}}{\frac{rc}{2}} \tag{7.12}$$

$$C_{load} = \sum_{i=1}^{5} C_{wire_{0\rightarrow\ i}} + \sum_{i=1}^{5} C_{g_i} \tag{7.13}$$

$$C'_{load} = C_{load} - C_{g_5} - C_{wire_{0\rightarrow 5}} \tag{7.14}$$

To carry out the placement of the shell and core latches, we bias the placement as shown in Algorithm 7. For simplicity, we assume a star-route model for the connections between the local clock drivers ($lcd$) and the latch pins ($fpin$) (Figure 7.8). This is conservative since it produces an upper bound on the wire length of any single connection between the local clock driver and the connected latch pins. Figure 7.9 shows the $RC$ network used to model the net in Figure 7.8. To meet a target skew ($Skew_{local}$) in the cluster, we use the delay estimated by Equation 7.9 to map $Skew_{local}$ to an upper bound ($l_{ub}$) on the wire length of the edges of the star-route. We pick the pin with the longest edge connecting it to the local clock driver $lcd$. If the edge connecting this pin ($fpin$) is longer than the upper bound $l_{ub}$, we adjust the weight on the cluster net as we explained above.

It is important to emphasize that we compute a net weight and not a pin-to-pin weight for the placement engine. This means that when we selected the farthest pin in the cluster net, the computed adjusted weight using Equation 7.8 will be a conservative estimate of the weight needed on the net to make sure all edges connecting the pins to the cluster net have a wire length $< l_{ub}$.

For a given a target local skew constraint $Skew_{target}$, we set $D$ in Equation 7.10 to $Skew_{local}$ to solve for the required wire length of the segment joining the local clock driver $lcd$ and the farthest pin $fpin$. This would result in an upper bound $l_{ub}$ on the wire length that the placer must honor when placing the latch corresponding to $fpin$.

If the delay $D$ is known, or an upper bound on the delay is given, the goal is to compute the wire length $L$ that would produce such a delay $D$. To do that, Equation 7.9 is rewritten as Equation 7.10. Equation 7.12 shows how we can convert a known delay $D$ to a wire length $L$ such that a net that has two nodes $lcd$ and $fpin$, and has a wire length of $L$ and load capacitance of $C_{load}$ will have $D$ as its interconnect delay. In Equation 7.14, $C_{load}$' is produced by subtracting from the total

load capacitance $C_{load}$ the wire capacitance of length $L$ and the input capacitance of the farthest pin $fpin$ ($C_{g_5}$ in Figure 7.9). This is done so that the parasitics of the edge in question ($lcd \rightarrow fpin$) are not counted twice.

**Algorithm 7** Algorithm to compute weight adjustment needed to compensate for the adjusted wire length of the cluster nets

**Input:** $ClusterClockNets$ N=$\{n_i\}$, coarse placement of latches, and target local skew $skew_{local}$

**Output:** Weights on cluster nets are set to reflect a desired wire length change. The placer is required to produce the desired wire length such that a local skew constraint $skew_{local}$ is met.

1: Translate $skew_{local}$ into a wire length upper bound constraint $l_{ub}$ based on Equation 7.12.
2: **for** $n_i \in N$ **do**
3: fpin = get_far_pin($n_i$) // farthest pin from root of cluster
4: Set $lcd_i$ to the local clock driver of net $n_i$
5: $dist_{lcd_i \rightarrow fpin}$ = manhat_dist($lcd_i$,$fpin$) // $lcd_i$ denotes local clock driver (root of cluster of $n_i$), and $fpin$ is the farthest cluster pin from the root $lcd_i$
6: $\delta = dist_{lcd_i \rightarrow fpin} - l_{ub}$
7: $\lambda$ = calc_wght($\delta$, $dist_{lcd_i \rightarrow\ fpin}$) // using Equation 7.8
8: $W_{n_i} = W_{n_i} + \lambda$
9: **for** $latch_j \in n_i$ **do**
10: **if** $latch_j$ is Shell Latch **then**
11: $n_j = ShellLogicNet$ connected to $latch_j$
12: $W_{n_j} = W_{n_j} + W_{n_i}$
13: **end if**
14: **end for**
15: **end for**

In Step 1 of Algorithm 7, we compute an upper bound on the wire length $l_{ub}$ of any segment between the root $lcd$ of the cluster (local clock driver) and a latch ($fpin$) based on a local skew constraint, $skew_{local}$ as explained above. In Step 3, we find the farthest pin $fpin$ away from the root of the cluster $lcd$. Using the farthest pin to compute an upper bound on the wire-length will result in a conservative estimate of the required wire length to meet the target $skew_{local}$. In Step 5, we compute the Manhattan distance from $lcd$ to $fpin$. In Step 6, we find the wire length deviation of $fpin$ from the $l_{ub}$ to meet the target local skew. In Step 7, we use Equation 7.8 to translate this wire length violation into a net weight to bias the placement. In Step 8, we add the computed weight to the existing net weight. To make sure we do not degrade the quality of the placement of the shell logic, we update the weights of the shell nets in Steps 10-13 to reflect the core nets weights.

Thus, the cluster nets are biased based on the weight factors during the placement of the latches. Once the latches are placed, the assumption of star routing is removed, and a balanced route is chosen to route the clusters.

### 7.6.2 Placement of Shell Latches

Shell latches of different clusters are connected with wires and logic which span long distances on the chip as compared to those that connect core latches. To constrain the placer in such a way as to shorten all clock nets would yield poor results in terms of timing and wire length for the rest of the logic. Thus a special arrangement is made for this shell logic and wires. In order to capture the effect of placing the shell latches on the rest of the design, their placement is done in the context of the placement of the whole design. Algorithm 8 details the steps executed to place the shell latches without degrading the quality of the overall placement.

**Algorithm 8** Placement of shell latches

**Input:** Placement of local clock drivers & initial placement of core latches // Core latches are placed in first coarse placement stage. They will be marked as fixed temporarily until the shell latches are placed. After that, the core latches will get re-placed along with the shell logic as explained in Algorithm 10.

**Output:** Placement of Shell Latches

1: **for** clock net $n_i$ **do**
2: Mark clock buffers on $n_i$ *fixed*
3: **end for**
4: **for** latch $l_i$ in Clock Latches **do**
5: **if** $l_i$ is a core latch **then**
6: Mark $l_i$ *fixed*
7: **end if**
8: **end for**
9: Set net weights on the cluster nets to compensate for wire length change needed to meet clusters skew (Algorithm 7). Adjust the net weights of the shell logic such that the placement of the shell latches does not degrade the quality of the placement of the rest of the logic (Algorithm 7).
10: Run global placement to place the shell latches.

In Step 2, Algorithm 8 starts by locking in place (fixing) the clock network up to the local clock drivers (roots of the latch clusters). In Step 5, all core latches are marked fixed as well. These will get re-placed later in the flow to improve the QoR. Since their placement does not affect the long wires, we will exploit their locations to improve the QoR while honoring the local skew constraint. In Steps 9 and 10, we use the weights computed in Algorithm 7 to bias the placement of the shell latches. We also adjust the weights of the shell logic nets that are directly connected to the

shell latches that are being placed. In Step 11, we run the global (coarse) placement again to update the placement of the shell latches.

### 7.6.3 Placement of Core Latches

Once placement of the shell latches is done, they are locked in place (fixed) so that the skew computed so far, based on the shell latches alone, is maintained. The next step is to place the core latches. As presented earlier, the core logic is connected via local (short) wires. The objective is to improve the skew and reduce the switching power of the clusters without affecting the pre-designed top-level clock network and the shell latches. This is an easier task compared to the previous step of placing the shell latches.

In this phase, we also try to undo any degradation in wire-length of the shell logic nets. To do this, we adjust the weights of the shell logic nets as discussed next.

### 7.6.4 Net Weight Adjustment for Shell Nets

To compensate for the change in wire length as a result of the placement of the latches, we bias the placement of the design based on Equation 7.15 as opposed to the model in Equation 7.8. To make sure that we do not violate any constraints on the shell nets, we compute the change in wire-length as a result of the placement of the shell latches. This is easily done since we know the wire-length of the shell nets after the first (initial) placement and after the second placement phase (shell placement). The weight allocation constrains the placement in favor of the shell nets as opposed to the core nets. This produced good results since placement is biased in favor of the long wires over the short wires. We should emphasize that the weight adjustment in

this section affects the shell logic nets and not the shell latches since the shell latches have already been placed to meet a target skew ($skew_{local}$).

$$wt_{ij} = \alpha * e^{\beta * \frac{l_{ij} - lu_{ij}}{lu_{ij}}} \tag{7.15}$$

In Equation 7.15 , $\alpha$ and $\beta$ are tuning constants to bias the placement algorithm. $l_{ij}$ is the current wire-length of the net in question. $lu_{ij}$ is the upper bound in the wire-length so that the skew target is met. If some nets still violate their wire-length bounds after the first iteration of the placer, the weights can be tightened further. This is accomplished by tuning $\beta$. $\beta$ reflects the iteration count of the placement algorithm. Equation 7.15 allocates successively higher weights for violating nets. In this work, we did not iterate over the placement since we achieved positive slack after the first iteration. An iterative approach could potentially provide further gains. However, for efficient planning of the shell logic, tighter integration with a placer is desired. This will be part of future work.

**Algorithm 9** Algorithm to adjust weights of shell logic to compensate for movement of shell latches.

**Input:** List of paths $P = \{p_i, i = 1 \dots k\}$ between shell latches whose slacks $slack_{p_i} < slack_{thr}$, $\beta$ which is the iteration count in Algorithm 10

**Output:** slacks of $P$ meet slack budgets

1: **for** $p_i \in P$ **do**
2: $\delta\ slack_{p_i} = slack_{p_i}$ - $slack_{thr}$
3: budget_slack() // uniform partitioning of $\delta\ slack_{p_i}$ of path $p_i$ with $n$ nets such that each net receives a slack budget $\delta\ slack_{n_i} = \frac{\delta\ slack_{p_i}}{n}$
4: **for** shell net $n_j \in p_i$ **do**
5: $\delta\ l$ = wire_length_change($n_j$, $\delta\ slack_{n_j}$) // Equation 7.12
6: $\delta\ w$ = weight_adjust($\delta\ l$, $l_{n_j}$, $\beta$) // $lu_{ij} = \delta\ l$ and $l_{ij} = l_{n_j}$ in Equation 7.15
7: $\delta\ W_{n_j} = \min(\delta\ W_{n_j}, \delta\ w)$
8: **end for**
9: **for** $n_i$ on $p_i$ **do**
10: $W_{n_i} = W_{n_i} + \delta\ W_{n_i}$
11: **end for**
12: **end for**

Algorithm 9 shows the steps followed to optimize the shell logic nets. To mitigate any timing issues and reduce any impact on the timing of the design, we compile a list of critical or near-critical paths in the design whose slacks fall below a certain threshold $slack_{thr}$. The goal is to compute, for these paths, a set of weights to bias the placement and compensate for the movement of the shell latches. Since the shells have already been placed, the goal is to negotiate the placement of the logic to mitigate any impact on the timing of the shell logic nets.

In Step 2, we compute the slack violation of path $p_i$. In Step 3, we uniformly

partition the slack budget among the nets of the critical path. In Step 5, we translate the budgeted slack violation into a wire-length constraint using a 2-pin net model and employ the Elmore delay model to estimate the delays. In Step 6, we update the net weight based on Equation 7.15. In Step 7, to err on the conservative side as we update the placement, since a critical net could be shared by many critical paths, we only record the minimum weight-adjustment computed for the shell net over all critical paths on which the net appeared. In Step 10, we update the weights of the shell nets to guide the next global placement phase.

If, based on the timing analysis, some of the shell nets violate timing, we budget the slack (violation), and do another coarse placement to improve the shell nets. A weight factor is assigned to the nets on the critical paths, and these weights are cumulative over a set of coarse placement runs. This means a critical net that continues to violate timing will see its weight increasing and will get more attention in the succeeding placement steps. Although timing analysis of the design at this stage is coarse, given that the design has not been detailed placed and optimized, the global information produced by the coarse placer in terms of proximity and timing provide guidance to the criticality of the shell nets.

**Algorithm 10** Iterative placement algorithm to place the core latches and the shell logic. The placement of the core latches and the shell logic are iteratively modified to meet slack constraints on a set of critical paths.

**Input:** Placement of Shell Latches

**Output:** Placement of Core Latches and Shell Logic such that no timing violation exists on shell nets

1: $\beta = 1$ // initialize iteration counter
2: $done = False$
3: **while** $done == False$ **do**
4: Coarse placement of the design
5: Compute list of paths $P = \{p_i, i = 1 \dots k\}$ such that $slack_{p_i} < slack_{thr}$ // slacks are computed by a Static Timing Analysis engine
6: Adjust weights of nets on critical paths using Algorithm 9 // list of paths $P$ with slack violations, $\beta$, and $slack_{thr}$ are passed to Algorithm 9
7: $done = True$
8: **for** each $p_i \in P$ **do**
9: **if** $slack_{p_i} < slack_{thr}$ **then**
10: $done = False$
11: $Break$
12: **end if**
13: **end for**
14: $\beta$ += 1 // increment the iteration counter
15: **end while**

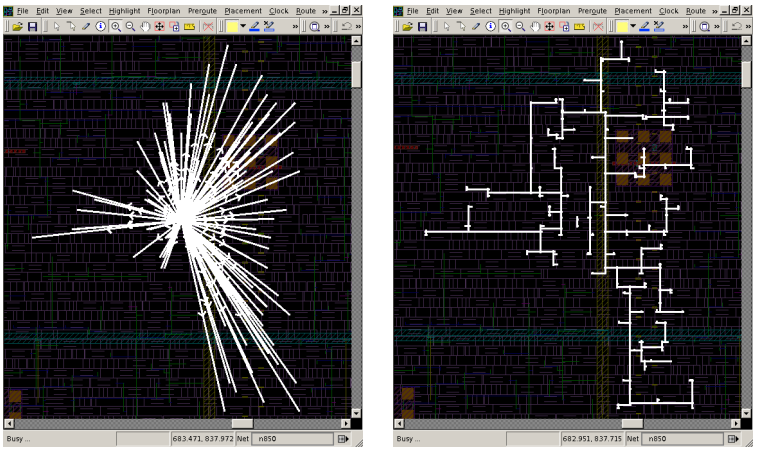
Algorithm 10 shows the top-level placement algorithm that is used to place the core latches and the shell logic. At this point, the shell latches have been placed and marked as fixed so that they do not move in any subsequent placement stage. Algorithm 10 relies on the freedom of moving the core latches and the shell logic to

mitigate any timing issues that could exist as a result of the placement of the shell latches. The core latches are re-placed without violating the local skew constraint $skew_{local}$ in the clusters.

In Step 1, we initialize the iteration counter. This counter plays an important role in computing a weight for the violating net (Equation 7.15). The weights on the violating nets will grow with every successive iteration. In Step 2, we initialize a Boolean flag *done* to *false*. In Step 3, the algorithm will continue to iterate over the placement of the shell logic cells and the core latches until the constraints are met. In Step 4, a coarse placement is carried out. In Step 5, static timing analysis of the design is carried out, and a list of paths whose slacks fall below a slack threshold $slack_{thr}$ is compiled. In Step 6, the weights of the critical nets which belong to the list of critical paths $P$ are adjusted as shown in Algorithm 9. In Step 7, the Boolean flag *done* is set to *true*. In Step 8, the algorithm iterates over the critical paths to check if timing violations still exist. In Step 9, a check is made if the slack of the path is below the slack threshold $slack_{thr}$. If so, then *done* is set to $false$ in Step 10 to indicate that another iteration is needed. In Step 11, the loop of Step 8 is terminated. In Step 14, the iteration counter $\beta$ is incremented, and another iteration commences.

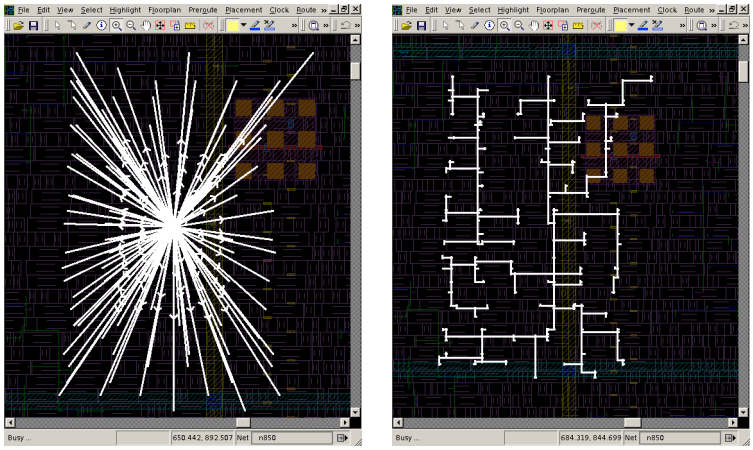
Figure 7.10 shows a sample local clock net (cluster net) in the CTS flow and the CDP flow. In Figures (a) and (b), a fly-net and a detailed-route of a sample net are shown as a result of the CTS flow. In Figures (c) and (d), the same net is shown after the latch placement stage in the CDP flow. An improvement of 16.5% in the wire length and a reduction of 17% in the wire capacitance are achieved as result of the controlled placement of the latches connected to this cluster net. Without the controlled placement of the latches in the clusters, no change to the top-level clock network can fix the local skew in the clusters. The only way to improve the local skew in the clusters is to either size up the local clock drivers which result in bigger power consumption of the clock network, or to re-place the latches to reduce the wire

length in the clusters as is done in the CDP flow. It is important to emphasize that the CDP methodology carries out this controlled placement early in the design flow before the placement of the logic cells. If the CTS flow were to change the placement of the latches after the logic cells are placed to fix this local skew problem, this would invalidate all the timing and optimization of the design and result in costly design iterations.



(a) Fly-net of net *n850* in the CTS flow

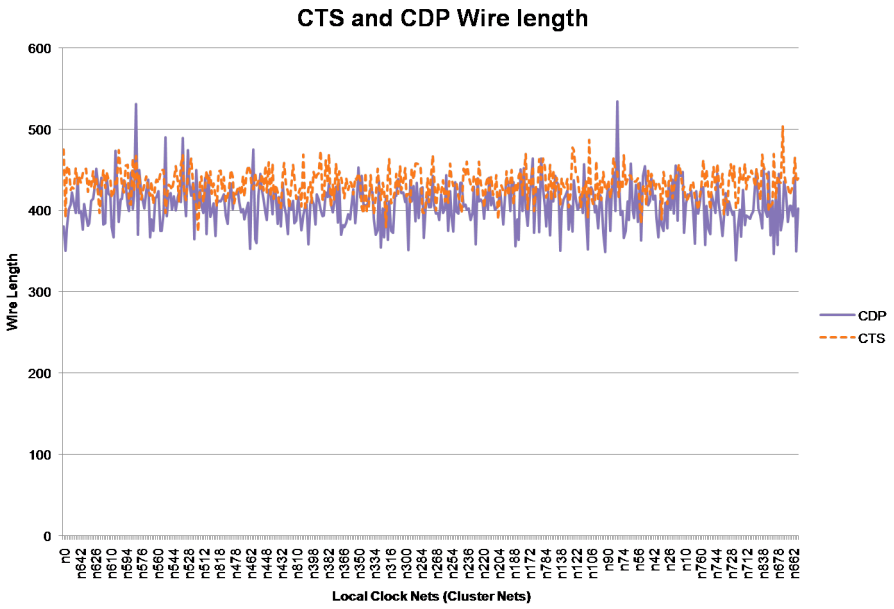
(b) Detailed-routing of net *n850* with a wire-length of 431 microns in the CTS flow



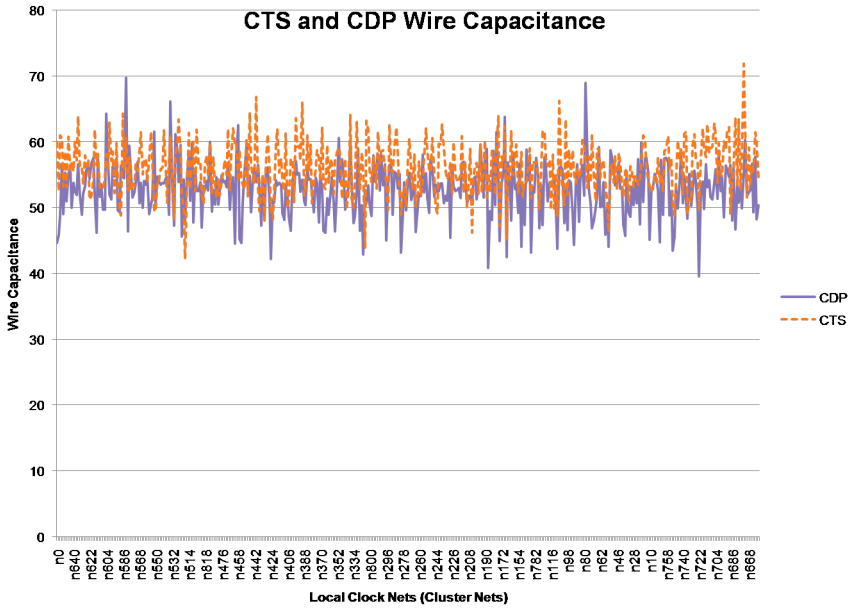
(c) Fly-net of net *n850* in the CDP flow

(d) Detailed-routing of net *n850* with a wire length of 359.5 microns in the CDP flow

Figure 7.10. This figure shows the placement of the latches with respect to placement of the local clock drivers. The picture in figure (a) shows a fly-net of a sample net (*n850*) in design *D1* which shows the placement of the latches in the CTS flow. The picture in figure (b) shows a detailed routing of net *n850* in the CTS flow. The pictures in figures (c) and (d) show a fly-net and a detailed-routing of the same net in the CDP flow. By controlling the placement of the latches in the clusters, the CDP flow provides a reduction of 71.5 microns (16.5%) in the wire length as compared to the CTS flow. This reduction in the wire length reduced the wire capacitance by 17% from 54.15nF in the CTS flow to 44.68nF in the CDP flow.




(a) Comparison of wire length of the cluster nets in design $D1$ between the CDP and CTS flows. The dashed line denotes the CTS data.




(b) Comparison of wire capacitance of the cluster nets in design $D1$ between the CDP and CTS flows. The dashed line denotes the CTS data

Figure 7.11. Graph (a) shows a comparison of the wire length between the CTS flow and the CDP flow. The results of the CDP flow show a reduction of as much as 26.2%. Graph (b) shows a comparison of the wire capacitance in the cluster nets between the CTS flow and the CDP flow.

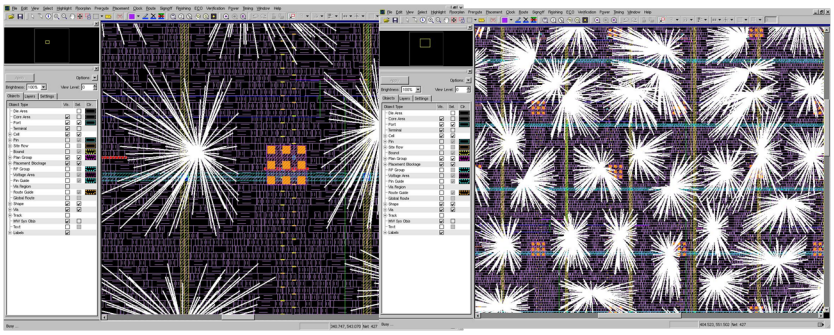


Figure 7.12. Placement of shell and core latches after CDP clustering. The highlighted clusters show how the latches gravitated towards the local clock drivers (centers of the clusters) while still accounting for their inter-cluster connectivity. The left picture is a blow-up of one of the clusters in the right picture.

## 7.7 Top-level Clock Network Design

At this stage, the latch clusters have been identified and placed. In addition, local clock drivers have been inserted to drive the latch clusters. The local clock drivers are the leaves of the top-level clock network. Since the local clock drivers have already been placed, all required information to synthesize the top-level clock network (Part 1 and 2 in Figure 7.4 is available at this stage.

The objective is to design a reliable top-level clock network which meets timing constraints prior to the placement and routing stages of the back-end phase. This top-level network can be a custom designed circuit or a synthesized one. In this work, we have used both a synthesized tree topology and a custom-designed H-tree topology to drive a clock mesh. Some of the experiments presented in Chapter 8 use a hybrid network with an H-tree topology to drive the clock mesh while others use a synthesized tree topology to drive a clock mesh. Although clock meshes are usually the design choice for microprocessors today, more ASIC designs are opting for a mesh implementation because of its tolerance to on-chip process variations (OCV). In addition, meshes, although they cost more in terms of power and wiring, produce lower-skew clock networks. Since the clock network is the biggest consumer of switch-

ing power in the design, a predictable and reliable clock network allows us to design a reliable power network early on without having to settle for a conservative worst-case design and incur a big penalty in terms of power and routing resources.

### 7.7.1 Clock Mesh Design

The design of the clock mesh is carried out using Algorithm 4 that was presented in the previous chapter. Estimating the number and sizes of the mesh drivers is done as a function of the load-capacitance composed of the clock gates, local clock drivers (LCDs), and the wiring capacitance due to the horizontal and vertical straps of the clock mesh (Figure 7.13). Since a mesh is a multi-driven net, its analysis is not straight-forward. AWE-like methods would not produce the desired accuracy since they can only handle linear RLC(K) circuits and cannot handle the non-linear driver models of the clock drivers. In nanometer designs, design margins and tolerance are very tight. We desire to move away from worst-casing any decision if possible. Furthermore, due to the sheer number of devices involved in the mesh analysis, we use a fast-spice simulator to model the active devices in the clock network with their non-linear models to be able to get simulation results in an efficient and accurate manner. The simulation times of the clock networks for the industrial designs studied in this work are typically a few minutes. Thus, what-if analysis can be carried out to efficiently optimize the top-level clock network before we lock it in and continue the rest of the design flow. Figure 7.14 shows the topology of the clock network that is presented to the fast-spice simulation engine.

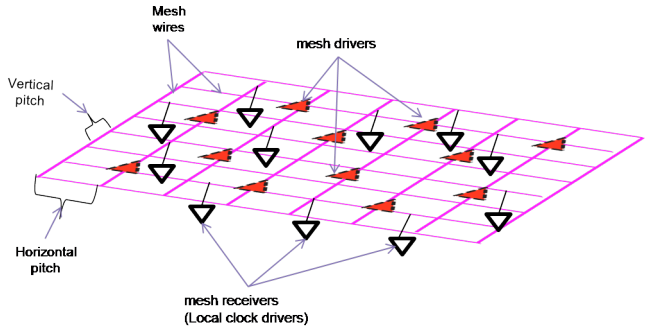



Figure 7.13. This figure shows the different components of a clock mesh. The mesh drivers are shown with dashed borders, while the mesh receivers are shown with black borders. To design a clock mesh, the locations of the mesh receivers should be given. The design of a clock mesh involves deciding on the number of clock drivers needed to drive the mesh, and the placement of these clock drivers. It also involves deciding on the horizontal and vertical pitches of the mesh.

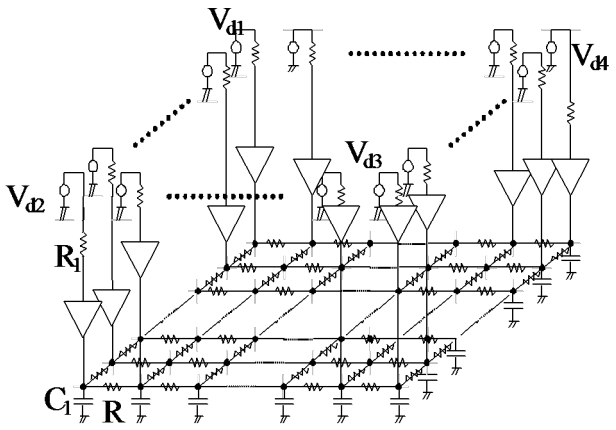



Figure 7.14. RC-network for the clock mesh shown in Figure 7.13. The mesh drivers are modeled as non-linear elements. Detailed spice models are used to capture the non-linearity of the mesh drivers. The gate capacitance of the mesh receivers are used to model them as decoupled capacitors to ground. The mesh wiring is modeled as an RC circuit. Analysis of this circuit is done using a fast-spice algorithm.

### 7.7.2 Top-level Clock Tree

Once the mesh is designed, all information needed to design a top-level clock tree is available. We mentioned that our intention is to design a predictable and robust

clock network. Any clock topology that meets these requirements will fit in our flow. In this work, we have experimented with both a synthesized top-level clock tree using a classical CTS engine as well as a custom-built H-tree to drive the mesh. Once the top-level clock network is designed, all timing annotations up to the inputs of the mesh receivers are updated to reflect the top-level clock network.

## 7.8 Detailed P&R

At this stage, the clock network is designed and placed. We mark all clock cells (buffers, clock gates, and latches) as fixed so that no further placement stage modifies their locations. We then carry out a detailed placement on the entire design so that the placement of the logic cells reflect the clock design. Any further physical synthesis and optimization step can continue with exact and accurate clock annotations.

## 7.9 Power Optimization

Our methodology facilitates further optimization and tuning of both power and clock networks. Since clock is the biggest consumer of power in the design, it is necessary to make sure that the clock network is not over designed. Although it is known that there is a trade-off between power and performance, careful design of the clock network can reduce the dynamic power without sacrificing the performance.

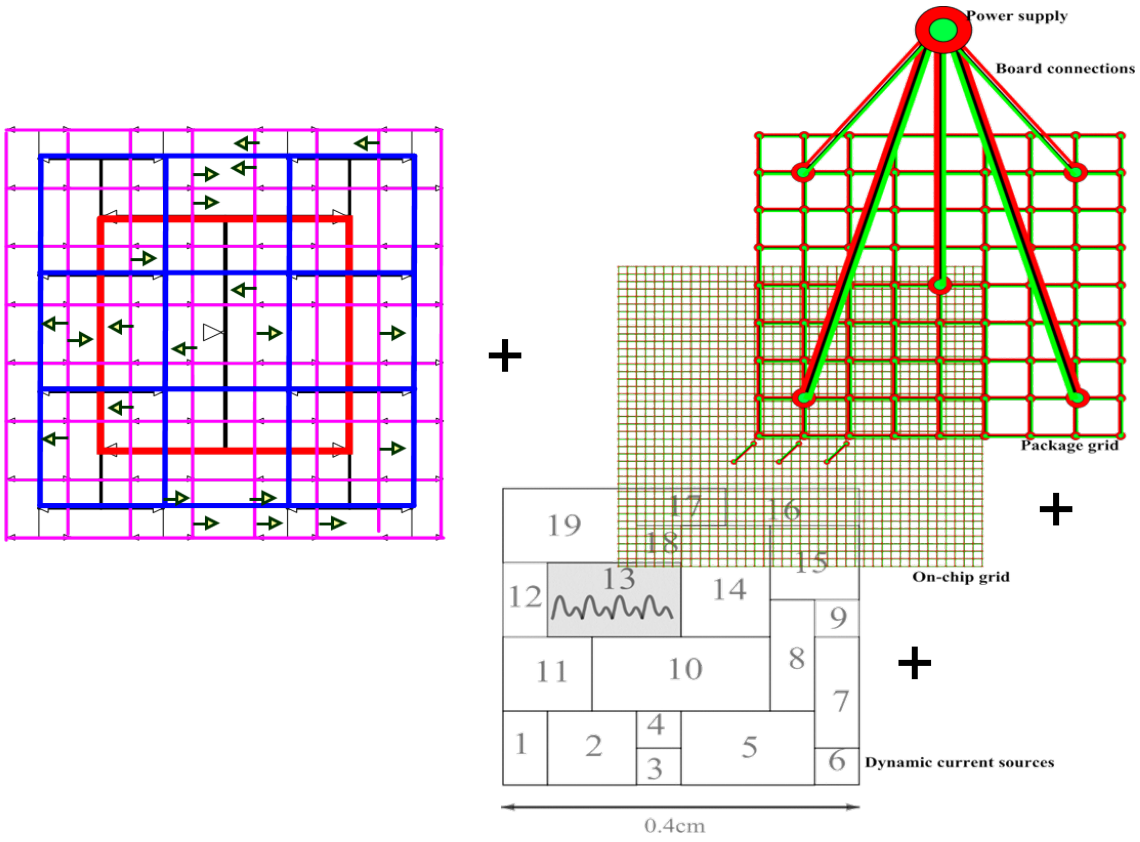



Figure 7.15. Once the clock network (shown at the top left) is designed and optimized using our CDP methodology, simultaneous design and analysis of both power and clock networks can be carried out before we proceed further in the flow. By estimating the current sources of the logic blocks in the design (bottom right), and by extracting the $RC$ network of the clock mesh (top-left), the $R(L)C$ network of the package power network (top-right), the $R(L)C$ network of the on-chip power grid (middle right), we can carry out simultaneous simulation of the power and clock networks to highlight any weakness in either of the networks.

Any reduction in the capacitive load seen by the clock network will result in savings in the dissipated power ($P = \frac{1}{2}\alpha\, C_L V^2 f$). By Clustering and placing the latches early in the design cycle, our proposed design methodology (CDP) can further reduce the portion of the dynamic power that is dependent on the placement of the local clock drivers and the sizes and placement of the register clusters. This added degree of freedom protects the design of the clock network from over-constraining the sizes of the clusters to account for the wire delay and capacitive loading of an already placed and optimized design.

We do not carry out any simultaneous optimization of the power and clock networks in this work. However, by designing the entire clock network early in the design cycle, we have put in place all necessary components to carry out the simultaneous design and optimization of both power and clock networks (Figure 7.15)

## 7.10 Design Tuning

The CDP flow described above is composed of manual and automated steps. Further tuning of the design can be carried out to improve performance, reduce power, and reduce the impact on the over all design.

The mesh can be further tuned by inserting more straps to improve the insertion delays at the inputs of the mesh receivers. Also, the clock buffers and clock gates can be sized up or down and detail-placed with minimum perturbation to the clock network.

The fact that CDP tackles the design of the clock network early in the design cycle based on coarse placement and not later when the design is placed and optimized makes any design tuning of the clock network less costly with respect to design cost (power) and design convergence (timing).

# Chapter 8

# Experimental Results

## 8.1 Introduction

In order to gauge the added value of the placement of the registers prior to the placement of the rest of the logic in the design, we will compare our CDP flow with the existing clock tree synthesis (CTS) flow as well as a custom mesh synthesis (CMS) flow. All experiments were carried out using Synopsys IC Compiler and Synopsys Nanosim fast-spice simulator.

## 8.2 Design Setup

The goal of these experiments was to show any performance gains that resulted from the CDP flow, beyond those of using a clock mesh flow. Since the CDP flow employs a clock mesh in the top-level network, we compare the CDP flow against a custom mesh flow. This will subtract out any gains in skew reduction by the CDP flow due to using a mesh. We ran three industrial designs, two of which use a 65nm process technology, and the third uses a 90nm process technology. The three designs

Table 8.1. Experiments Statistics

| Designs | Cells | Nets | FFs |
|---|---|---|---|
| D1 | 165832 | 166105 | 42665 |
| D2 | 115745 | 126831 | 15941 |
| D3 | 443467 | 452401 | 122102 |

have 42K, 16K, and 122K registers respectively. Table 8.1 shows the details of the three designs. We compare the skews and power dissipation in the three different clock topologies/flows. The CTS network is a traditional clock tree synthesized to minimize skew and reduce power. The CMS network is a hybrid network, where a clock-tree drives a clock mesh which in turn drives a local tree. The CDP is a hybrid clock network that was designed using the CDP methodology.

We would expect the CTS network to fare better than the other two networks in terms of power since it uses the least wiring, and thus need the smallest number and sizes of clock buffers. We also would expect CMS to consume the most power but provide a better skew compared to CTS. The objective of the CDP flow is to produce a skew comparable to CMS, and power comparable to CTS. In addition, since CDP designs the clock network up front before the implementation of the design it will have much higher predictability, and more freedom in optimization.

### 8.2.1 CTS Flow

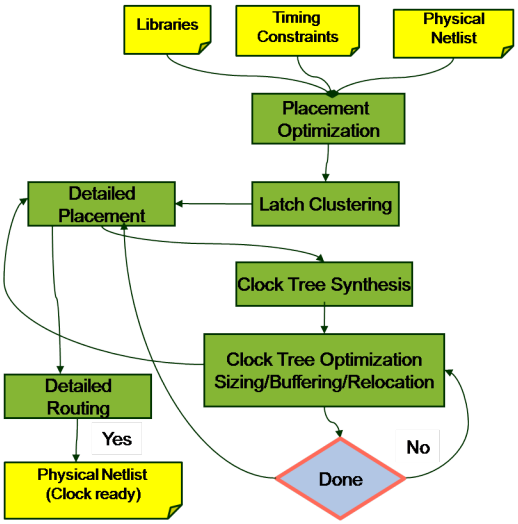



Figure 8.1. This figure shows a classical clock tree synthesis flow. In this flow, clock design is tackled late in the design cycle after the placement-based optimization stage is done. This would provide the locations of the latches which are needed to carry out the synthesis of the clock network.

Figure 8.1 details the ASIC design flow needed to synthesize a clock tree. Since the traditional ASIC RTL-GDSII flow defers the clock tree synthesis until after the cell placement is done, the CTS flow starts with a placed netlist and executes the following steps:

- *Placement-based Optimization*: Earlier, we discussed the importance of placement and global routing to any physical synthesis step. This is due to the dominance of the interconnect delays over cell delays. Thus, this step is concerned with physical optimization operations such as high-fanout synthesis, repeater insertion, gate sizing and gate duplication. All these optimization steps are based on a globally placed and routed netlist. The objective of this step is to tackle any timing or design violation and update the netlist to reflect the design change. This would minimize further changes to the netlist later in the design flow that would negatively affect the synthesized clock tree.

- *Latch Clustering*: Due to the large number of sequential elements in the design, clustering is a mandatory step to help the clock tree synthesis engine cope with the complexity of the network. Also, since most of the dynamic clock power is in the last stage of the clock network, careful implementation of this step is needed to reduce the dynamic power while honoring the clock constraints. There is a large body of research and development that deals with partitioning or clustering the clock network. The merits of these algorithms depend on the constraints which they honor as well as run-time.

  When clock-gating elements (clock gates) are present in the design, special attention has to be done during clustering. Sequential elements which share the same clock-enable signal can be grouped and partitioned as needed based on the design and physical constraints. Each duplicated (cloned) or merged (de-cloned) clock-gate constitutes a register cluster. For ease of balancing the clock network and reducing the load capacitance seen by the top-level clock network, sequential elements that are not connected to any clock gate or local clock driver can be clustered separately.

  The clustering algorithm honors the slew and load constraints at the output of the clock gates or the inserted repeaters. The algorithm also takes into account the intra- and inter-cluster skew when placing and sizing the clock gate or the inserted buffer.

- *Clock Tree Synthesis*: The roots of the latch clusters serve as leaves for the clock-tree synthesis. This step builds a tree topology in a bottom-up fashion from the leaves to the clock root. Chapter 5 provides further details on this step.

- *Clock Tree Optimization*: This step deals with sizing up or down the clock buffers/gates to meet the latency, skew, or slew constraints. Also, this step

Table 8.2. Skew & Power results of CTS flow. Column 2 shows the skew results of the CTS flow, and column 3 shows the dynamic clock power.

| Design | Skew (ps) | Switching Power (mw) |
|---|---|---|
| D1 | 147 | 72.6507 |
| D2 | 91 | 163.84 |
| D3 | 200 | 121.3290 |

tackles issues related to low power optimization. In addition, clock buffers can be inserted inside the clusters to help balance the over all clock tree. Also low-power constraints are tackled by this step. Buffer relocation is also carried out so that the clock cells are positioned at the desired locations. Detailed placement follows the optimization step to legalize the clock cells.

Table 8.2 shows the skew and power results produced by the CTS flow.

### 8.2.2 CMS Flow

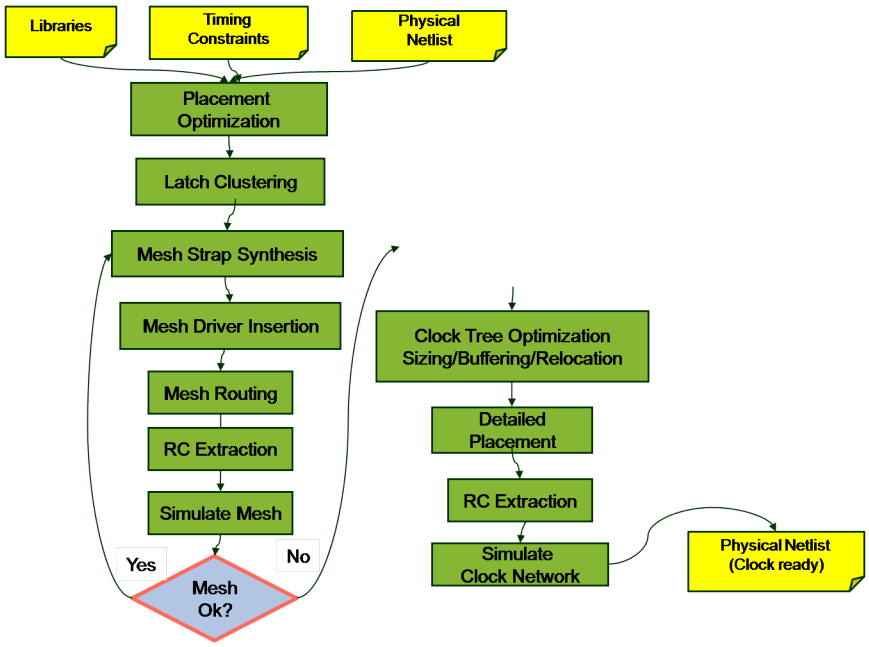



Figure 8.2. This figure shows the clock mesh synthesis flow. In this flow, the mesh is designed after the placement and optimization of the design are done. This would provide the locations of the latches and the local clock drivers.

Figure 8.2 details the design flow needed to construct a custom clock mesh. Since the traditional ASIC RTL-GDSII flow defers the clock tree synthesis until after the cell placement is done, the CMS flow starts with a placed netlist and executes the following steps:

- *Placement-based Optimization*
- *Latch Clustering*: CMS uses the same clustering algorithm used in the CTS flow. Although latches can tap into the mesh directly, this would result in a lot of wiring (stubs) that connects the latches to the nearest mesh strap. As a result, the design would incur a congestion penalty as well as a bigger dynamic power consumption due to the added capacitance of the wiring stubs. However, for designs that are sensitive to on-chip variations, sequential elements may

connect directly to the mesh to reduce the wire-length. This would produce a network that is more immune to process variations albeit at the cost of more power.

- *Mesh Strap Synthesis*: This step decides on horizontal and vertical pitches of the mesh. An *m*x*n* mesh is synthesized and laid-out while honoring any obstructions present in the design.

- *Mesh Drivers Insertion*: This step is concerned with designing and planning a buffer-configuration that is capable of driving the synthesized mesh and producing a minimum-skew clock network. The placement and sizes of the clock buffers are decided by the designer. What-if analysis using fast-spice simulation is used to improve the buffer-configuration and optimize their locations..

- *Mesh Routing*: connecting the drivers and receivers to mesh requires different routing algorithms than those used in signal routing. Comb-like routing is used where the mesh drivers and receivers connect to the nearest mesh strap using short stubs .

- *Mesh Simulation*: Simulation is carried out by a fast-spice engine (Synopsys Nanosim). Any simulator would do provided it has enough capacity and the needed performance.

- *Top-level Clock-tree Construction*: The goal is to synthesize a minimum-skew clock tree from the clock root to the mesh drivers. The mesh drivers have timing annotations which reflect the down-stream network composed of the mesh and the local clock trees. The actual algorithm for building this clock tree is the same as the one outlined under the CTS Flow.

- *Top-level Clock-tree Optimization*: The same optimization algorithm used in the CTS flow is used here.

Table 8.3. CMS skew and wire-length numbers. The table shows the number of local clock drivers needed in column 2. In column 3 and 4, we show CMS skew results and skew reduction as compared to the CTS flow. In column 5, we show the change in wire-length as compared to the CTS flow.

| Design | Local Clock Drivers | Skew(ps) CMS | Skew Change | WL Change |
|---|---|---|---|---|
| D1 | 427 | 58 | -13.7% | 8% |
| D2 | 1468 | 28 | -7% | 0.4% |
| D3 | 1022 | 86 | -9.3% | 10% |

In Table 8.3, we compare the skew and wire-length results of the CMS and the CTS flows.

### 8.2.3 CDP Flow

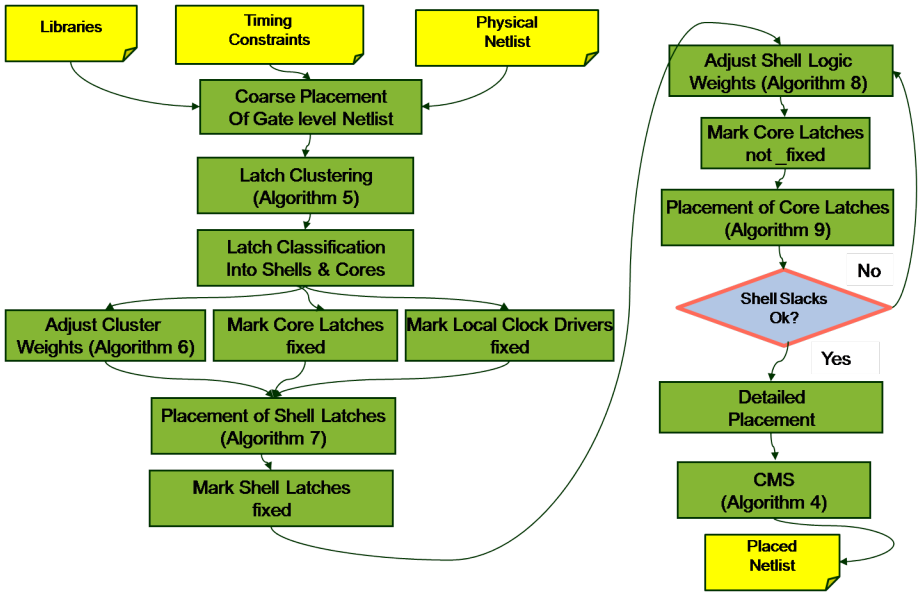



Figure 8.3. This figure shows our proposed Clock-driven Design Planning Latch Placement Flow. This flow tackles the clock design task early in the design cycle before the logic cells have been placed. After this flow is done, the placement of the latches and the local clock drivers is known, and the design of the top-level clock network can commence.

Figure 8.3 shows the steps followed in the CDP flow. The major change in this flow as compared to the CTS and CMS flows is that the clock design problem is tackled early in the design flow before the logic is placed and optimized. This is an important distinction as described earlier. The CDP flows performs the following steps:

- *Coarse Placement*: The placement produced in this step allows us to cluster the latches based on their rough locations. In addition, this placement provides an estimate of the wire-length for the cluster nets.
- *Latch Clustering*: A clustering algorithm based on K-means algorithm is executed to partition the latches into clusters each driven by a local clock driver.
- *Latch Classification*: This step walks a timing graph and classifies the latches into two sets: shells & cores.
- *Adjustment of Cluster Nets Weights*: In this step, a delay (skew) budget is translated into a wire-length budget. The desired wire-length budget is translated into a net weight to bias the global placement to produce the desired change in wire-length.
- *Coarse Placement*: As a result of this step, the core latches are placed and marked fixed, and the local cluster skew budget is attained.
- *Adjustment of Shell Logic Nets' Weights*: This step tries to balance the placement of the core latches with the slack of the shell logic nets. Subsequent movement of the core latches is permitted to reduce any impact on the timing of the shell logic while honoring the local skew budget of the clusters.

Table 8.4 shows the results of the CDP flow. We show statistics related to the classification of the registers into shells and cores. We also show the skew and wire

Table 8.4. Shell/Core Stats. In column 2, the number of local clock drivers needed is shown. In column 3 and 4, we show the results of Latch Classification into shells and cores. In columns 5 and 6, we show skew results. Column 7 shows the skew reduction. In column 8, we show the change of wire-length in our flow.

| Design | Local Clock Drivers | Shell FFs | Core FFs | Skew(ps) (CMS) | Skew(ps) (CDP) | Skew Change | WL Change |
|---|---|---|---|---|---|---|---|
| D1 | 427 | 20774 | 22679 | 58 | 50 | -13.7% | 8% |
| D2 | 1468 | 16492 | 2364 | 28 | 26 | -7% | 0.4% |
| D3 | 1022 | 59646 | 64330 | 86 | 78 | -9.3% | 10% |

Table 8.5. This table shows that the slew rates at latch inputs for the CMS and CDP flows are practically identical.

| Design | Transition(ps) min/typical/max (CMS) | Transition(ps) min/typical/max (CDP) |
|---|---|---|
| D1 | 133/113/102 | 132/109/101 |
| D2 | 63/74/82 | 63/75/85 |
| D3 | 78/87/189 | 74/87/177 |

length results produced by the CDP flow and how they compare to the CMS flow. In Table 8.5 we show that the transition times computed at the inputs of the clock registers in both the CDP and CMS flows are virtually the same. In Table 8.6, we compare the clock power results produced by both the CTS and the CDP flows.

## 8.3 Summary

Table 8.4 shows clock skew results of the CMS and CDP flows. The results show skew reductions of our proposed flow between 7% and almost 14%. Table 8.5 shows the transition times (slew rates) at the inputs of the latches in both flows are practically identical. Table 8.6 shows that the skew results of CDP are improved over the CTS flow by as much as 70%. Also in Table 8.6, we show switching power results of CDP and CTS. Although the CDP flow as presented here relies on a clock mesh

Table 8.6. Skew & power comparison of CDP and CTS flows. Column 2 shows the skew results of the CTS flow, and column 3 shows the skew reduction due to CDP. Columns 4 and 5 show the switching power numbers, and the last column shows the power difference between CTS and CDP.

| Design | Skew(ps) (CTS) | CDP/CTS Skew ratio | Switching Power (mw) (CTS) | Switching Power (mw) (CDP) | CDP/CTS Power ratio |
|---|---|---|---|---|---|
| D1 | 147 | .605 | 72.6507 | 80.8737 | 1.11 |
| D2 | 91 | .692 | 163.84 | 151.63 | .926 |
| D3 | 200 | .57 | 121.3290 | 115.7502 | .954 |

as part of the hybrid top-level network, we still managed to reduce the power in two of the three cases. These gains in power are due to better placement of the latches. In the CTS flow, conservative decisions are made on the sizes of the clusters. These decisions are constrained by the placer which does not differentiate between logic cells and clock cells. In our flow, we can afford to increase the size of the clusters (provided we do not violate any electrical constraints of the inserted local clock drivers) and still achieve low skew. The results shown are based on fully placed and optimized designs, and the clocks are detailed routed.

# Chapter 9

# Conclusion

We have introduced a new clock driven design methodology (CDP) to enable design of low-power and low-skew clock networks. The goal of the proposed design methodology is to overcome the new challenges that accompanied the downscaling of the process technology below 100nm. We have discussed the different forces that played a role in shaping the existing design methodologies and flows, and we have emphasized that some of the barriers for engineering a more productive and higher quality design methodology are barriers due to legacy issues and not due to technological issues.

The goals of our CDP design methodology are to enable the design of a predictable clock network that meets the design constraints. We emphasize skew predictability over skew reduction, although we strive to reduce the clock skew as demonstrated in the reported results. Also, our proposed methodology reduces the power consumption of the clock network due to the controlled clustering and placement of the clock registers early in the design cycle. In this chapter, we review and highlight the contributions of our work.

## 9.1 Revisiting the ASIC Design Flow

In Chapters 2 and 3, we have emphasized the importance of revisiting the ASIC design flow to face the growing design challenges. The goal of introducing the different components of the flow along with the different design decisions needed at every step is to highlight the existing flaws in the RTL-to-GDSII flow along with the challenges we face in closing on issues related to timing, signal integrity, reliability, and clock and power integrity.

We have elaborated on the design of the clock network and the reasons the ASIC design flow defers it until after the cell placement.

## 9.2 Robust Power & Clock Planning

Our proposed CDP design methodology advocates a fresh look at the design, planning, and optimization of the two most important signals in the design: power and clock. We promoted a holistic view for the solutions related to nanometer issues surrounding the power and clock design. We have shown that the design of power and clock does not have to be a chicken-and-egg problem, but rather with careful design of a robust top-level clock network which in our case is a mesh topology driven by a tree topology, along with careful placement of the clock registers early in the design cycle, the major consumer of power in the design ($> 40\%$ of total power) is designed and optimized. This early design of the clock network facilitates the design of a robust power grid without worst case decisions and ad-hoc assumptions about the clock network.

In Chapter 4, we presented the most important issues related to power planning. We emphasized the symbiotic relationship between power planning and clock planning. We stressed that although the ASIC design flow treats power and clock as

two separate problems, they are not and need each other for successful design of a robust implementation of both networks without relying on worst case decisions. We also presented the tight coupling of noise issues between the package and die, and we discussed the relationship between a robust power design and its direct impact on a robust clock design.

In Chapter 5, we presented clock network design and the various clock topologies that are used in both custom designs and ASICs. We highlighted the pros and cons of these topologies. We also highlighted the design constraints that drive clock network design, and we discussed different techniques for reducing the power consumption of the clock network. In our proposed methodology, we choose a hybrid implementation for the clock network. The hybrid clock network in our flow is composed of a top-level clock tree or H-tree which drives a clock mesh. The clock mesh in turn drives local clock clusters which have one-level clock trees. In Chapter 6, we discussed in details the design and implementation of a clock mesh. We discussed the algorithms needed for a successful implementation of a clock mesh, and we discussed the gaps in the existing design approaches that require further research. We elaborated on the iterative nature of designing these networks and the reliance on circuit simulation for verifying them.

One important point that we repeatedly visited and emphasized is that our design methodology enables the simultaneous design and optimization of both the power and clock networks. We have discussed the tight design margins that exist in nanometer designs, and we have discussed how worst case decisions can not be relied on to converge on a design and meet its tight constraints.

## 9.3 Clock Driven Design Methodology

In Chapter 7, a novel methodology (CDP) and algorithms that enable the clock network to drive the implementation of the rest of the design are presented. CDP partitions the clock registers into two classes: *shells* and *cores*. These in turn partition the logic into shell logic and core logic. In doing so, CDP differentiates between the long wires and short wires that are driven by the clock registers. Algorithms and heuristics are presented to place the *shell* registers and *core* registers before the placement of the logic cells and improving the QoR in terms of skew and power.

We have changed the existing ASIC design flow by making the clock network drive the implementation of the rest of the logic. Our main contribution is devising new algorithms for the placement of the latches prior to the physical implementation of the design. This contribution enables the implementation of both the clock and power networks early in the design cycle before placement of the logic cells. Given that power and clock are the two most critical signals in the design, their implementation is reflected on the implementation of the rest of the logic. A successful implementation of the clock and power will undoubtedly lead to robust implementation of the rest of the design.

We presented a methodology and algorithms for *predictable skew and low power* register placement. We discussed register clustering and emphasized the importance of clustering to reduce the dynamic power consumption of the clock. We showed how we bias the placement of the *shell* registers which drive long wires against that of the *core* registers which drive short wires. We then showed how the placement of the clock registers drives the placement of the rest of the logic.

In Chapter 8, we discussed in details the design of experiments. We presented three design flows: CTS, CMS, and CDP. CTS is a classical ASIC clock-tree-synthesis flow where the implementation of the clock network is deferred until the placement

and optimization of the design are done. We highlighted the flaws in such a flow that relies on worst-case decisions and ad-hoc assumptions in designing the power and clock networks. We discussed the impact of such a flow on design convergence.

In the CMS design flow, we drew on our experience with custom design to highlight how a top-level clock network is designed and tuned for high-performance designs. The design of such a network comes at a cost in power and area. We have elaborated on why ASICs can not afford to adopt such a methodology due to lack of resources, and due to the cost in terms of power, design time, and verification.

We showed our clock-driven design flow, CDP, and the different steps that are needed to place the registers and drive the placement and routing of the rest of the logic. We presented results on three industrial design cases that have between 16K and 122K latches.

We have compared our design methodology (CDP) against those of CTS and CMS. CDP showed improvement over CTS in skew of more than 70% as well as savings in power in some cases of as much as much as 8%. For the comparison with CMS, CDP showed skew improvements of as much as 14% in some cases without a large penalty in wire length.

# Bibliography


[1] A. Agarwal, V. Zolotov, and D. Blaauw, "Statistical clock skew analysis considering intradie-process variations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 1231– 1242, August 2004.

[2] H. Bakoglu, *Circuits, Interconnects, and Packaging for VLSI.* Addison-Wesley, 1990.

[3] J. R. B. Black, "Electromigration failure modes in aluminium metallization for semiconductor devices," in *Proceedings of the IEEE*, vol. 57, September 1969, pp. 1587–1594.

[4] S. Borkar, "Parameter variation and impact on circuits and microarchitecture," in *Proceedings of the Design Automation Conference*, 2003.

[5] S. Boyd, L. Vandenberghe, A. E. Gamal, and S. Yun, "Design of robust global power and ground networks," in *Proceedings of ACM International Symposium on Physical Design*, 2001, pp. 60–65.

[6] Y. Cao, Y.-M. Lee, T.-H. Chen, and C. C.-P. Chen, "Hiprime: Hierarchical and passivity reserved interconnect macromodeling engine for rlkc power delivery," in *Proceedings of the Design Automation Conference*, 2002, pp. 379–384.

[7] S. C. Chan, P. J. Restle, N. K. James, and R. L. Franch, "A 4.6 ghz resonant global clock distribution network," in *IEEE ISSCC Digest of Technical Papers*, February 2004.

[8] S. C. Chan, K. L. Shepard, and P. J. Restle, "1.1 - 1.6ghz distributed differential oscillator global clock network," in *IEEE ISSCC Digest of Technical Papers*, February 2005.

[9] ——, "Uniform-phase, uniform-amplitude, resonant load global clock distributions," *IEEE Journal of Solid State Circuits*, vol. 40, pp. 102–109, January 2005.

[10] H. Chen and D. Ling, "Power supply noise analysis methodology for deep-submicron vlsi chip design," in *Proceedings of the 34th annual conference on Design automation*, 1997, pp. 638–643.

[11] H. Chen, C. Yeh, G. Wilke, S. Reddy, H. Nguyen, W. Walker, and R. Murgai, "A sliding window scheme for accurate clock mesh analysis," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, vol. 6, Novemebr 2005, pp. 939–946.

[12] P.-Y. Chen, K. Ho, and T. Hwang, "Skew aware polarity assignment in clock tree," in *Proceedings of the IEEE/ACM international conference on Computer-Aided Design*, 2007, pp. 376–379.

[13] J. Cullum, A. Ruehli, and T. Zhang, "A method for reduced-order modeling and simulation of large interconnect circuits and its application to peec models with retardation," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, pp. 261–273, April 2000.

[14] A. Devagan and S. Elassaad, "Pacakge-chip co-design tutorial," *Proc. IEEE International Conference on Compuier Design*, 2005.

[15] D. E. Duate, N. Vijaykrishnan, and M. J. Irwin, "A clock power model to evaluate impact of architectural and technology optimization," *IEEE Transactions on VLSI Systems*, vol. 10(6), pp. 844–855, Dec. 2002.

[16] R. Dutta and M. Marck-Sadowska, "Automatic sizing of power ground (p/g) networks in vlsi," in *Proceedings of ACM/IEEE Design Automation Conference*, 1989, pp. 783–786.

[17] M. Edahiro, "Minimum path-length equidistant routing," in *IEEE Asia-Pacific Conference on Circuits and Systems*, December 1992, pp. 41–46.

[18] J. P. Fishburn, "Clock skew optimization," in *IEEE Transactions on Computers*, vol. 39, 1990, pp. 945–951.

[19] W. Hou, X. Hong, W. Wu, and Y. Cai, "Path-based timing-driven quadratic placement algorithm," *Proc. IEEE/ACM Asia and South Pacific Design Automation Conference*, pp. 745–748, 2003.

[20] K. Jain, "A factor 2 approximation algorithm for the generalized steiner network problem," in *IEEE Symposium on Foundations of Computer Science*, 1998, pp. 448–457.

[21] H. Kerivin and A. R. Mahjoub, "Design of survivable networks," *Networks*, pp. 1–21, April 2005.

[22] P. Larsson, "Resonance and damping in cmos circuits with on-chip decoupling capacitance," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 45, pp. 849–858, August 1998.

[23] C.-H. Lee, C.-H. Su, C.-Y. Huang, S.-H. Lin, and T.-M. Hsieh, "Floorplanning with clock tree estimation," in *Proceedings of the Annual Conference on Design automation*, 2005, pp. 6244–6247.

[24] Y. Liu, X. Hong, Y. Cai, and W. Wu, "Cep: A clock-driven eco placement algorithm for standard-cell layout," *International Conference on ASIC*, pp. 118–121, 2001.

[25] Y. Lu, X. Hong, W. Hou, W. Wu, and Y. Cai, "Combining clustering and partitioning in quadratic placement," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 4.720 – 4.723, 2003.

[26] Y. Lu, C. Sze, X. Hong, Q. Zhou, Y. Cai, L. Huang, and J. Hu, "Register placement for low power clock network," *Proceedings of the ASP-DAC 2005*, vol. 1, pp. 588– 593, January 2005.

[27] K. J. M., G. Sigl, F. M. Johannes, and K. J. Antreich, "Gordian: Vlsi placement by quadratic programming and slicing optimization," *IEEE Transactions on CAD*, vol. lO(3), pp. 356–365, March 1991.

[28] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proceedings of Custom Integrated Circuit Conference*, 2001, pp. 223–228.

[29] R. Newton, "Presentation of 1996 phil kaufman award to dr. carver mead," *Archives of EDA Consortium*, 1996.

[30] Y. Nieh, S.-H. Huang, and H. S.-Y., "Minimizing peak current via opposite-phase clock tree," in *Proceedings of IEEE/ACM Design Automation Conference*, June 2005, pp. 182–185.

[31] A. Odabasioglu, M. Celik, and L. T. Pilleggi, "Prima: Passive reduced-order interconnect macromodeling algorithm," *IEEE Transactions on Computer-Aided Design*, vol. 17, pp. 645–654, August 1998.

[32] R. Puri, L. Stok, and S. Bhattacharya, "Keeping hot chips cool," in *Proceedings of the 42nd annual conference on Design automation*, 2005, pp. 285 – 288.

[33] J. M. Rabaey, A. Candrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, 2002.

[34] A. Rajaram, J. H., and R. Mahapatra, "Reducing clock skew variability via crosslinks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, pp. 1176–1182, June 2006.

[35] A. Rajaram and D. Pan, "Meshworks: an efficient framework for planning, synthesis and optimization of clock mesh networks," in *Proceedings of the 2008 conference on Asia and South Pacific design automation*, 2008.

[36] A. Ruehli and H. Heeb, "Circuit models for three-dimensional geometries including dielectrics," *IEEE Trans. Microwave Theory Techniques*, vol. 40, pp. 1507–1516, July 1992.

[37] A. Ruehli, "Equivalent circuit models for three-dimensional multiconductor systems," in *IEEE Transactions on Microwave Theory and Techniques*, vol. MTT-22, March 1974.

[38] T. Sakurai and A. Newton, "Alpha-power law mosfet model and its applications to cmos inverterdelay and other formulas," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 584–594, April 1990.

[39] R. Saleh, S. Z. Hussain, S. Rochel, and D. Overhauser, "Clock skew verification in the presence of ir-drop," *IEEE Transactions on Computer Aided Design*, pp. 635–644, June 2000.

[40] R. Samanta, G. Venkataraman, and J. Hu, "Buffer polarity assignment for power noise reduction," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2006, pp. 558–562.

[41] S. Sapatnekar2003 and H. Su, "Analysis and optimization of power grids," *IEEE Design & Test of Computers*, vol. 20, pp. 7–15, May-June 2003.

[42] Y. Shin and T. Sakurai, "Power distribution analysis of vlsi interconnects using model order reduction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 739–745, June 2002.

[43] J. Singh and S. Sapatnekar, "Congestion-aware topology optimization of structured power/ground networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, pp. 683–695, May 2005.

[44] H. Su, K. Gala, and S. Sapatnekar, "Analysis and optimization of structured power/ground networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, pp. 1533–1544, November 2003.

[45] X. D. S. Tan and C. J. R. Shi, "Fast power/ground network optimization based on equivalent circuit modeling," in *Proceedings of ACM/IEEE Design Automation Conference*, 2001, pp. 550–554.

[46] X. D. S. Tan, C. J. R. Shi, and J. C. Lee, "Reliability-constrained area optimization of vlsi power/ground networks via sequence of linear programmings," *Transactions on Computer-Aided Design*, vol. 22, pp. 156–161, December 2003.

[47] B. Taskin and I. Kourtev, "Delay insertion method in clock skew scheduling," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, 2006, pp. 651–663.

[48] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Proceedings of the 35th Design Automation Conference Proceedings*, 1998, pp. 732–737.

[49] C.-W. A. Tsao and C.-K. Koh, "Ust/dme: a clock tree router for general skew constraints," in *Proceedings of the IEEE/ACM lnternational Conference on Computer-Aided Design*, 2000, pp. 400–405.

[50] R.-S. Tsay and J. Koehl, "An analytic net weighting approach for performance optimization in circuit placement," *Design Automation Conference*, vol. 28, pp. 620–625, 1991, article.

[51] G. Venkataraman, Z. Feng, H. J., and P. Li, "Combinatorial algorithms for fast clock mesh optimization," in *Proceedings of ICCAD*, 2006, pp. 79–84.

[52] N. Venkateswaran and D. Bhatia, "Clock-skew constrained placement for row based designs," *Proc. IEEE International Conference on Computer Design*, pp. 219–220, 1998.

[53] J. M. Wang and T. V. Nguyen, "Extended krylov subspace method for reduced order analysis of linear circuits with multiple sources," in *Proceedings of the Design Automation Conference*, 2000, pp. 247–252.

[54] T. Wang and C. C. Chen, "Optimization of the power/ground network wire-sizing and spacing based on sequential network simplex algorithm," in *Proceedings of International Symposium on Quality Electron. Design*, 2002, pp. 157–162.

[55] X. Wu, X. Hong, Y. Cai, C. Cheng, J. Gu, and W. Dai, "Area minimization of power distribution network using efficient nonlinear programming techniques," in *Proceedings of the 2001 IEEE/ACM international Conference on Computer-aided Design*, 2001, pp. 153–157.

[56] J.-S. Yim, B. S.-O., and C.-M. Kyung, "A floorplan-based planning methodology for power and clock distribution in asics," in *Proceedings of the 36th Annual Conference on Design automation*, 1999, pp. 766–771.

[57] M. Zhao, R. V. Panda, S. S. Sapatnekar, T. Edwards, R. Chaudhry, and D. Blaauw, "Hierarchical analysis of power distribution networks," in *Proceedings of the Design Automation Conference*, 2000, pp. 481–486.