Copyright
by
Anand Kumar Rajaram
2008

The Dissertation Committee for Anand Kumar Rajaram
certifies that this is the approved version of the following dissertation:

Synthesis of Variation Tolerant Clock Distribution Networks

Committee:

David Z. Pan, Supervisor

Joydeep Ghosh

Leon S. Lasdon

Michael Orshansky

Nur A. Touba

Synthesis of Variation Tolerant Clock Distribution Networks

by

Anand Kumar Rajaram, B.E., M.S.

DISSERTATION
Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN
December 2008

To my family - my parents, wife and brother.

Acknowledgments

First and foremost, I wish to thank my adviser Prof. David Z. Pan for all his help and support to me since Fall 2004. Without his constant encouragement, this dissertation would not have been possible. Though this is usually true for most graduate students and their advisers, it is especially true for me since I pursued my Ph.D. along with an out-of-town full-time job. It was Prof. Pan's empathy for my unique situation and his constant encouragement that helped me complete this dissertation. I have no words to thank him for all the help, encouragement and advice. I thank my committee members (in alphabetical order) – Prof. Joydeep Ghosh, Prof. Leon S. Lasdon, Prof. Michael Orshansky and Prof. Nur A. Touba – for being on my committee and for their insightful comments and questions. I thank my fellow UTDA group members (in alphabetical order) – Ashutosh Chakraborty, Minsik Cho, Tao Luo, Joydeep Mitra, Anand Ramalingam, Sean Shi, Peng Yu and Kun Yuan – who have helped me with many questions last few years. I thank the ECE Graduate Program Coordinator Melanie Gulick who helped me wade through all the necessary paper work related to my Ph.D. during the last few years.

I thank Texas Instruments for agreeing to let me pursue my Ph.D. in Austin while working full-time with them in Dallas. My special thanks to the Dallas DSP group members (in alphabetical order) – Sanjive Agarwala, Rajesh Annapillai, Raguram Damodaran, Anthony Hill, Arjun Rajagopal, Paul Wiley and others – who encouraged me both in work and in my research. I especially

thank Sanjive and Raguram who let me to work three days a week from Austin during Spring 2007 that enabled me to complete my course work. I am always grateful to them for their timely help.

I thank my parents who have been a source of inspiration and comfort to me throughout my life. It is their constant advice, blessings and good wishes that have made me what I am today and I can never hope to repay my infinite debt to them. I would also like to thank my younger brother Gokul for his numerous help to my work and research when we stayed together in Dallas. Though he is younger to me by a few years, he is much wiser in many ways and I have a lot to learn from him. I also wish to thank my in-laws for their blessings and good wishes for my studies last few years.

Last, but definitely not the least, I thank my wife Shanthi who has made it all worthwhile. Without her support and encouragement I would not have made it to the finish line. It was she who made the most personal sacrifices to help me complete my dissertation and I cannot thank her enough.

Anand Rajaram

University of Texas, Austin.

Synthesis of Variation Tolerant Clock Distribution Networks

Publication No. ____________________

Anand Kumar Rajaram, Ph.D.
The University of Texas at Austin, 2008

Supervisor: David Z. Pan

In the sub-65nm VLSI technology, the variation effects like manufacturing variation, power supply noise and temperature variation become very significant. As one of the most vital components in any synchronous VLSI chip, the Clock Distribution Network (CDN) is especially sensitive to these variations. The unwanted clock skews caused by the variation effects consume increasing proportion of the clock cycle, thereby limiting chip performance and yield. Thus, making the clock network *variation-tolerant* is a key objective in the chip designs of today.

In this dissertation, we propose several techniques that can be used to synthesize variation-tolerant clock networks. Our contributions can be broadly classified into following four categories: **(i)** Efficient algorithms for synthesizing link based non-tree clock networks. **(ii)** A methodology for synthesizing a balanced, variation tolerant, buffered clock network with cross-links. **(iii)** A comprehensive framework for planning, synthesis and optimization of clock mesh networks. **(iv)** A chip-level clock tree synthesis technique to address issues unique to hierarchical System-On-a-Chip (SOC) designs that are becoming more and more frequent today.

Depending on the performance requirements and resource constraints of a given chip, the above techniques can be used separately or in combination to synthesize a variation tolerant clock network.

Table of Contents

Acknowledgments **v**

Abstract **vii**

List of Tables **xiii**

List of Figures **xv**

Chapter 1. Introduction **1**

1.1 The Big Picture 1

1.2 Clock Network Design Objectives and Clock Network Types . . . 3

1.3 Thesis Organization 7

Chapter 2. Algorithms for Link Insertion in Clock Trees **9**

2.1 Link Insertion in Clock Trees 9

  2.1.1 Effect of Link Insertion on Skew Variability 10

  2.1.2 Skew Variability Between Arbitrary Nodes 11

  2.1.3 Link Insertion Scheme Overview 12

2.2 Review of Existing Link Insertion Methodologies 13

  2.2.1 Rule-based Node-pair Selection Algorithm 14

  2.2.2 Min-matching Based Node-pair Selection Algorithm 16

  2.2.3 Statistical Link Insertion 18

2.3 Rule-Delta Algorithm 20

  2.3.1 Parameter Determination 22

2.4 MST Based Link Insertion Algorithm with Rule Based Deletion . 27

  2.4.1 Parameter selection 31

2.5 Incremental Link Insertion 33

  2.5.1 Motivation for Incremental Link Insertion 33

  2.5.2 Automatic Link Selection 35

  2.5.3 The Algorithm 37

2.6 Application of Link Insertion Algorithms to Non-zero Skew CDNs 41

2.7 Experimental Results 43
    2.7.1 Experimental Setup 43
    2.7.2 Rule-Delta and MST Algorithms 44
    2.7.3 Run-time Comparison 47
    2.7.4 Automatic Parameter Determination Results 48
    2.7.5 Incremental Link Insertion Results 50
    2.7.6 Skew reduction results for non-zero skew clock networks . . 51
2.8 A note on selecting the appropriate link insertion algorithm 52

Chapter 3. Link Insertion for Buffered Clock Trees 54
3.1 Why Link Insertion for Buffered Clock Trees? 54
    3.1.1 Challenges in Buffered Clock Tree Link Insertion 55
        3.1.1.1 Chicken-egg problem 56
        3.1.1.2 Accuracy of delay model 56
        3.1.1.3 Short-circuit Power and Waveform Quality 57
    3.1.2 Existing clock tree synthesis algorithms 62
    3.1.3 Requirements of a Link Insertion Friendly Buffered Clock Tree Synthesis 63
3.2 Iterative Delay Evaluation and Backward Slew Propagation 66
    3.2.1 Iterative delay and slew evaluation 66
    3.2.2 Backward propagation of slew 68
3.3 Link Insertion Friendly Clock Tree Synthesis 69
    3.3.1 Subtree merging with backward slew propagation 70
    3.3.2 Balanced CTS algorithm 73
3.4 Link insertion flow 79
3.5 Sensitivity based Link Insertion 81
    3.5.1 Drawbacks of the Existing Approaches 81
    3.5.2 Sensitivity Based Algorithm 82
    3.5.3 Sensitivity Vector 83
    3.5.4 α-*Rule* for Buffered Clock Tree 87
    3.5.5 Sensitivity Based Link Insertion 89
    3.5.6 Advantages of Sensitivity Based Link Addition 91
3.6 Experimental Results 92
    3.6.1 Results for Balanced CTS and Link Insertion 92
    3.6.2 Results for Sensitivity Link Insertion 95

Chapter 4. Optimized Clock Mesh Network Synthesis 99
4.1 Clock Mesh Synthesis Problem 99
4.2 Mesh Planning and Synthesis 104
    4.2.1 Terms and Definitions 104
    4.2.2 Total wire-length as a function of Mesh size 105
    4.2.3 Skew as a function of Mesh size 107
    4.2.4 Mesh Optimization Friendly Buffer Placement/Sizing 112
4.3 Network Sensitivity Based Mesh Optimization 115
    4.3.1 Network Sensitivity Theory 115
    4.3.2 Accurate Buffer Modeling For Mesh Optimization 117
    4.3.3 Mesh Optimization Algorithm 119
    4.3.4 Buffer-resizing for Mesh Optimization 124
4.4 Wire Sizing for Reliability 124
4.5 Practical Considerations in the Use of MeshWorks 129
4.6 Experimental Results 131
    4.6.1 Experimental Setup 131
    4.6.2 Mesh Planning, Synthesis and Optimization Results 132
    4.6.3 Results of Wire-sizing for EM 137
    4.6.4 Results for Practical Issues in MeshWorks Usage 138

Chapter 5. Robust Multi-Corner Chip-level CTS 142
5.1 Chip-level Clock Tree Synthesis Problem 142
5.2 Motivation and Problem Formulation 145
    5.2.1 Significance of Clock Divergence Reduction 146
    5.2.2 Impact of Sub-block Clock Pin Location on Clock Divergence 147
    5.2.3 Multi-corner skew reduction problem 148
5.3 Clock Pin Assignment Algorithm for Clock Divergence Reduction 151
5.4 Multi-corner Skew Reduction Algorithm 154
5.5 Chip-Level CTS Algorithms 160
    5.5.1 Single-Corner DME based Approach 160
    5.5.2 Multi-Corner DME based Approach 161
    5.5.3 Greedy CCTS Algorithm 161
    5.5.4 Dynamic programming Based CCTS Algorithm 162
5.6 Practical Considerations in CCTS 167

5.6.1 Generalization of Pin Assignment Algorithm 167
5.6.2 Consideration of Blockages 168
5.6.3 Measuring Divergence 169
5.7 Experimental Results 169
5.7.1 Test-case Generation 170
5.7.2 Experimental Setup, Results and Discussions 172

Chapter 6. Conclusion **178**

Bibliography **179**

Vita **193**

List of Tables

2.1 Maximum skew variation (MSV), standard deviation (SD) and total wire-length (WL) of trees. The CPU time is the time for generating the tree using BST [17] code. 44

2.2 Skew variations and wire-length in terms of tree results shown in Table 2.1. Size of a tree+link network is the # of links. 46

2.3 Results for automatic parameter determination procedures. The HSPICE Skew variability and wire-length values are w.r.t. the tree values of Table 2.1. 49

2.4 Incremental link insertion results w.r.t. the tree values of Table 2.1. 51

2.5 Skew reduction for non-zero skew clock networks. The nominal skew (NS), maximum skew variation (MSV), standard deviation (SD) values for non-zero skew CDNs. The delay values given for the trees are absolute values (ps) and the values given for the non-trees are w.r.t. the corresponding tree. 52

3.1 Average power and sink skew for different values of skews between signals A and B in Figure 3.2 58

3.2 Skew variation and resource consumption results for the algorithm in [11] 95

3.3 Skew variation and resource consumption results for our new algorithms and algorithms in [14] w.r.t. results of [11] in Table 3.2 . 96

3.4 Size and skew variability information for buffered clock trees. WCS and SD in pico-seconds 97

3.5 Skew variation information for link based non-trees w.r.t. the results of trees shown in Table 3.4 97

4.1 Comparison of the different mesh optimization approaches – absolute values of buffer area (BA), wirelength(WL), power (PWR) and frequency (F_{max}). 134

4.2 Comparison of the different mesh optimization approaches – percentage value w.r.t. MM method. 135

4.3 Summary of optimization results from Table 4.1 and 4.2 for all test cases. 135

4.4 Results of EM violation fixed by wire-sizing. BA and WA (Wire-Area) are % reduction w.r.t. MM row in Table 4.1 and 4.2. $\#V_{em}$ denote number of EM violations. #itr denotes the number of EM fixing iterations. 137

5.1 Key test-case Generation Parameters. 171

5.2 Multi-corner delay, skew and clock divergence results for the different CCTS approaches. 176

5.3 Average & normalized delay, skew, clock divergence information along with Buffer Area(BA) and Wire Length(WL) results for the test-cases in Table 5.2. All normalization done w.r.t. results of 1-corner DME approach. 177

List of Figures

1.1 Working of a Synchronous Chip. 2

1.2 Different type of clock networks have progressively higher quality, robustness and cost. 5

2.1 An example of cross link based non-tree. 10

2.2 An example in which the rule based link-insertion algorithm might fail. 16

2.3 A bipartite graph model for selecting node pairs between two subtrees. Each subtree is further divided into a number of sub-subtrees for link insertion. An edge weight between two nodes is the shortest rectilinear distance between leaf(sink) nodes of two sub-subtrees. 17

2.4 An example of *The Law of Diminishing Returns* for link addition. As more links are added to a given tree, the extra skew reduction is limited 23

2.5 Effect of α on skew reduction and wire-length increase. Parameter β also exhibits similar behavior 24

2.6 An example of a new approach that might be better than the min-matching based link insertion. 28

2.7 The top-level algorithm of selecting node pairs for link insertion. . 29

2.8 MST based algorithm of selecting node pairs for link insertion. . . 30

2.9 A simple nontree where incremental approach might help. 34

2.10 The top-level incremental link insertion algorithm. 38

2.11 The method of picking the best link based in α value of the non-tree. 39

2.12 Illustration of link addition for non-zero skew clock trees. Link is added between equi-delay nodes A and P. 43

2.13 Run-time comparison between the different link insertion methods as a function of number of links inserted at $\gamma = 1$ level. 48

3.1 An example of link-based non-tree. (a) Unbuffered case. (b) Buffered case. 56

3.2 A simple example of link insertion in buffered clock trees. 57

3.3 Sink voltage waveform with 0ps skew between input signals A and B in Figure 3.2. 59

3.4 Sink voltage waveform with 50ps skew between input signals A and B in Figure 3.2. 59

3.5 Sink voltage waveform with 100ps skew between input signals A and B in Figure 3.2. 60

3.6 Sink voltage waveform with 150ps skew between input signals A and B in Figure 3.2. 60

3.7 Sink voltage waveform with 200ps skew between input signals A and B in Figure 3.2. 61

3.8 (a) An example of an unbalanced buffered clock tree; (b) An example of a balanced buffered clock tree. 64

3.9 (a) Definitions of transition times for nodes v and a. (b) A simple example of RC network. 67

3.10 Plot of x (ratio of RC and input slew) versus y (ratio of RC and output slew) of Equation (3.4). 70

3.11 An example of subtree merger using effective downstream capacitance 71

3.12 Procedure to evaluate the signal transition values of a node given the transition values of the child nodes. 73

3.13 The procedure to evaluate the effective downstream capacitance recursively. 74

3.14 The algorithm for selecting the subtrees to be merged. 76

3.15 The Merging cost for two subtrees. 78

3.16 Clock Tree with Variations on Each Segment 84

3.17 Node Pair Selection. 91

4.1 (a) A clock network with leaf-level mesh. (b) Leaf-level mesh driving clock sinks. 100

4.2 Examples of sparse and dense clock meshes. A dense mesh is likely to have shorter stubs. 106

4.3 Determining the right mesh size. 106

4.4 Three dominant skew components in a mesh - skew due to buffer delay imbalance, skew due to difference in distance from closest buffer and skew due to different stub length and load capacitance. 109

4.5 Plot showing the fidelity of the skew bound Equation(4.2). Though skew bound is not perfectly linear of actual worst case skew, it is monotonic. 110

4.6 The top-level algorithm of selecting the initial mesh size. 111

4.7 An example where the buffer insertion algorithm of [77] might not take the better choice. The shaded circles represent buffers of proportional size. 113

4.8 Network sensitivity theory can be applied for clock mesh optimization. 115
4.9 Accurate buffer used for clock mesh optimization. S is the size of a given buffer 118
4.10 Comparison of sink delays in SPICE obtained using buffers and the buffer model for a clock mesh test-case. 119
4.11 Simple example of network sensitivity based mesh optimization. . 120
4.12 Iterative wire-sizing flow to fix EM violations. 128
4.13 MeshWorks can be seamlessly applied for chips with blockages. . . 129
4.14 Mesh optimization result on a testcase with blockage. 139
4.15 Mesh optimization done on a testcase with two clocks on same floorplan. Clock-A is shown. 139
4.16 Mesh optimization done on a testcase with two clocks on same floorplan. Clock-B is shown. 140

5.1 A simple chip-level CTS example. The black circles represent the clock root for each sub-block. 146
5.2 Even for identical nominal skews, Case A is better than Case B because of lesser clock divergence and hence lesser skew variation. 147
5.3 Importance of clock pin assignment for sub-blocks. Case A and Case B differ in the clock pin location for block B, which affects CTS. If blocks A and B have critical paths between them, Case B will result in better yield because of reduced clock divergence between A and B. 148
5.4 Simple example illustrating difficulty of balancing two different IPs. The clock tree delays of the two blocks will scale differently across different corners due to different buffer sizes and interconnect lengths. 149
5.5 Buffer configuration used for multi-corner delay characterization. . 155
5.6 Multi-corner skew balancing heuristic. 158
5.7 Dynamic Programming based approach to chip-level CTS. The sub-steps are explained separately in Figures 5.8, 5.9, 5.10. . . . 163
5.8 Procedure to pick valid pairs for merger from a given set of sub-trees.164
5.9 Pre-eliminate procedure to eliminate very bad merging choices. . . 165
5.10 Post-eliminate procedure used to eliminate dominated sub-trees. . 166

Chapter 1

Introduction

1.1 The Big Picture

Electronic Design Automation (EDA) has been one of the key drivers of the remarkable cost reduction and performance improvements of VLSI chips in the last few decades. The focus of EDA is to build tools and algorithms that can be used to automate the design of complex electronic systems like computer processors and cell-phone chips. Physical Design Automation is one of the important steps of EDA in which the logical description of a chip is converted to the actual physical design that can then be manufactured to create the actual chips used in the electronic systems. This dissertation belongs to the area of Physical Design Automation.

Any VLSI chip can be divided into three logical component types. The first type of components do the computing that represent the actual work that needs to be done by the VLSI chip. Examples of this type include adders, multipliers etc. and are called *data elements* in general. The second component type stores both the input data for the data elements and also the result from them. Examples of this type include registers, latches and memory. The final component type synchronizes the overall operation of the chip such that the chip as a whole completes the necessary tasks in the expected amount of time. This third component is the focus of this dissertation.

In most VLSI chips, the synchronization is done using a clock signal. An ideal clock signal is a periodic binary signal that is either 0 or 1 at any given time. The clock signal is used as a reference with respect to which the data transfer between the individual registers in the chip takes place. An example is shown in Figure 1.1 in which the data from Register1 is available for use at the *launch* clock signal. This data is used to do the necessary computation by the data elements between Register1 and Register2 before the *capture* signal reaches the Register2. At that point, the data computed by the data elements get stored in Register2. This operation repeats every clock cycle. In most VLSI chips, there are hundreds of thousands of such registers interacting in a complex way to complete the required operations of the chip.

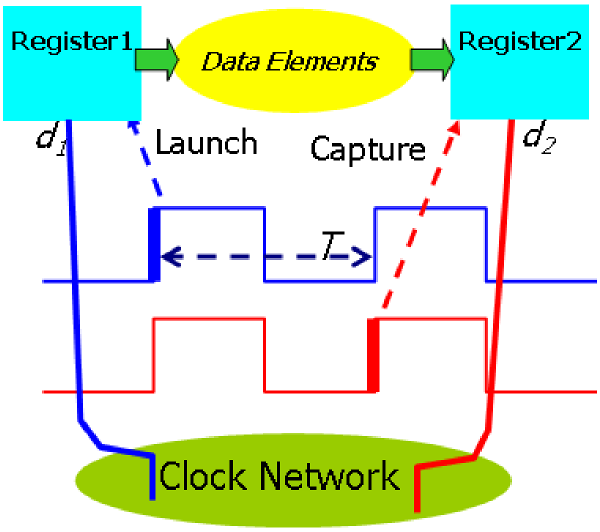


Figure 1.1: Working of a Synchronous Chip.

Clock Distribution Networks (CDNs) are the means by which the clock signals are transferred to every sequential element (registers, latches etc.) in the chip. Ideally, the clock signal that is transferred to every register will be identical. However, in reality the clock signals reaching the different registers will be different from each other due to many factors[1]. As a result, the quality of the clock signal is measured by the maximum time difference or *skew* between any two clock signals reaching any two registers in the chip. Since the complete chip depends on the clock signal as the reference, a higher clock skew means worse reference and thus worse overall chip performance. As a result, synthesis of high quality clock distribution networks is one of the most important steps in Physical Design Automation.

1.2 Clock Network Design Objectives and Clock Network Types

Due to the fact that CDN is one of the biggest and fastest switching nets in most VLSI chips, the quality of CDN can have tremendous influence on the overall operations of a chip[21]. This is especially true in the sub-65nm VLSI technology used today in which the variation effects like manufacturing variation [47], power supply noise [65], temperature variations [15] become very significant. The unwanted clock skews caused by these variation effects consume increasing proportion of the clock cycle, thereby limiting chip performance and yield. It is to be noted here that the skew variation can be a very significant portion of the *nominal* skew. For example, according to [42], just the interconnect variations can contribute to as much as 25% skew variation. Such a significant impact makes the traditional method of applying margins for variation both inefficient and risky. Thus, making the clock network *variation-tolerant* is a key

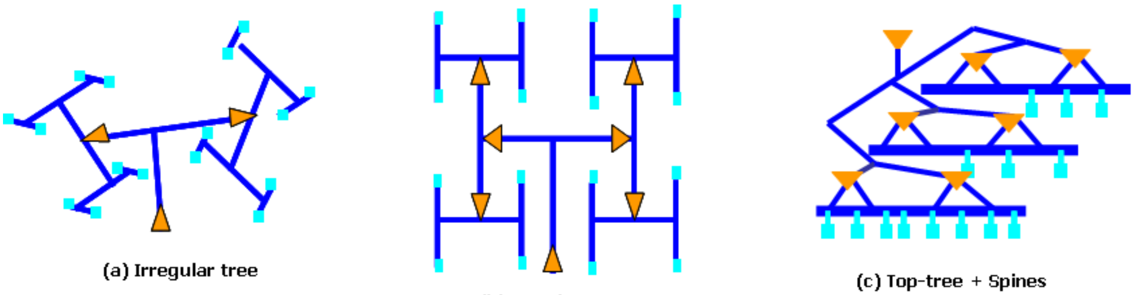
---

[1] Details discussed in the next sub-section

objective in the chip designs of today. Needless to say, variation tolerance should come at as little a cost as possible in terms of traditional metrics like delay, buffering/routing resources and power.

Though it is important for all synchronous chips to have a variation tolerant and low skew clock network with the use of minimum possible resources, the exact requirements for different chips differ drastically depending on the type of chip and its potential application. For example, for a chip that is designed to be used in a high performance computer, speed is the most important metric while power, though important, is secondary. However, for chips used in mobile devices, power is the key metric, while performance is secondary. Of course, there are various chips (and applications) that fall somewhere in-between the two extreme characterizations.

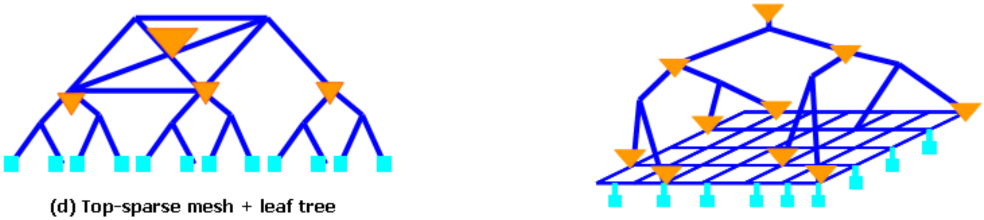
Figure 1.2 shows some of the existing types of CDNs. The *irregular tree* show in Figure 1.2-a is typically used in low power, low performance designs that do not require variation tolerant clock networks. Because of this, the main focus in their design is to keep both power and buffer/routing resources low. The *regular trees* or *balanced trees* in Figure 1.2-b is typically used in medium power/performance chips in which a reasonable variation tolerance is needed. As a result, it uses more power and resources compared to the irregular tree. The clock network type shown in Figure 1.2-c was used in Intel-Pentium$^{\text{TM}}$in which three *spines* were laid along the length of the chip and each was driven at multiple points, making it a non-tree clock network. This can significantly reduce the skew variation between the registers connected to a given spine. However, the power and resources used is likely to be higher than a balanced tree because of the presence of multi-driver nets and thick spines. The sparse top-level mesh shown in Figure 1.2-d was proposed in [67] in which a top-level sparse mesh drives a number of trees at the leak-level. The presence of redundancy at the top-level due to the mesh structure reduces the skew variation between the roots of the clock trees driven by the mesh. However, the skew within each tree driven by the mesh is still present. Finally, the type shown in Figure 1.2-e uses a complete leaf-level mesh that is driven at multiple points by a carefully tuned tree at the top-level. This type of clock network structure is used in high performance microprocessors like [64]. Because of the significant amount of redundancy in the clock network, the skew variation is drastically reduced. However, the redundancy implies a significantly higher power and resource requirement compared to the other types of clock networks.



(a) Irregular tree

(b) Regular tree

(c) Top-tree + Spines



(d) Top-sparse mesh + leaf tree

(e) Top-tree + Dense Mesh

Figure 1.2: Different type of clock networks have progressively higher quality, robustness and cost.

It may be noted that each of the five types of clock network structures discussed above has progressively higher performance at the cost of progressively higher power and resource requirements. The exact type of clock structure that is picked depends of the type of application. For example, for chips used in electronic toys, the irregular trees are sufficient as the performance requirement is very low. However, in the case of a processor used in supercomputers, the performance is the main goal and the power/resource requirements are secondary. This broad range of power vs. performance trade-off in different VLSI chips gets translated to a wide-range of requirements for different clock distribution networks. In broad terms, these requirements can be classified under the following four major types of clock networks:

- Small clock networks with strict low power/resources requirements
- Medium sized clock networks that can afford a small power increase
- High performance designs that have strict variation tolerance requirements. These designs can typically tolerate small losses in performance for significant gains in power.
- Hierarchical chip-level clock networks for System-On-a-Chip designs. SOC chips are built by integrating a number of different sub-chips together to form a big chip that can perform a vast array of tasks. For this type of clock networks, the focus is on integration of different components such that the complete system meets the performance requirements under all the extreme variation conditions.

1.3 Thesis Organization

In this work, we propose four categories of techniques that can be used to synthesize variation-tolerant clock networks. These four techniques correspond to the four major types of clock networks discussed above. They are:

- Efficient algorithms for synthesizing link based non-tree clock networks that can be used in small designs with strict resource constraints. Our algorithms can obtain clock networks that are significantly more variation-tolerant with little extra resources compared to the original clock trees.

- A methodology for synthesizing a balanced, variation tolerant, buffered clock network with cross-links. Our approach improves skew variability by 50% with small increase in buffer-area and wire-length. This method can be used in medium sized designs with strict resource constraints.

- A comprehensive framework for planning, synthesis and optimization of clock mesh networks. Clock meshes are typically used in high performance chips, such as microprocessors. Though they are variation tolerant, they are very resource intensive. Using our framework, we can achieve reductions of 26%, 19% and 18% in buffer area, wire-length and power respectively, without sacrificing variation tolerance. This makes clock meshes an attractive choice in high-performance ASIC designs, such as DSP cores, that cannot afford the same resources as a microprocessor.

- A chip-level clock tree synthesis (CTS) technique for System-On-a-Chip (SOC) chips. We specifically address issues that assume added importance in SOC-designs compared to traditional designs, like multi-corner skew and clock divergence. Our methods can achieve 14%-26%(19% on an average) reduction in the clock path divergence, 46% reduction in average multi-corner skews at the cost of 0.4% increase in buffer area and 0.2% increase in wire-length compared to existing methods.

Depending on the performance requirements and resource constraints of a given chip, any one of the above techniques can be used, either by itself or in combination with other techniques to synthesize a variation tolerant clock network. The rest of the dissertation is organized as follows: in Chapter-2, we present several novel link insertion algorithms that can be used to convert any given clock tree into a link-based non-tree clock network. The algorithms related to synthesis of link-based buffered clock networks are presented in Chapter-3. We propose our clock mesh network synthesis and optimization framework in Chapter-4. In Chapter-5, we present our multi-corner, chip-level clock tree synthesis algorithms. We conclude the dissertation in Chapter-6.

Chapter 2

Algorithms for Link Insertion in Clock Trees

In this chapter, we will first review the concept of link insertion in clock trees, followed by a brief review of previous link insertion algorithms. Next, we propose three different link insertion algorithms and discuss their results. We also discuss how the link insertion scheme can be used for non-zero skew clock networks. We conclude the chapter with a discussion on the use of the different link-insertion schemes under different contexts.

2.1 Link Insertion in Clock Trees

Systematic link insertion in clock trees for reducing skew variation was first proposed in [54] in which cross links were added in an existing clock tree, thereby making it a non-tree clock network. By suitably choosing the correct locations of the links, it was shown that significant reduction in skew variability can be achieved with very small increase in wire-length when compared to the original clock tree. An example of link-based non-tree is shown in Figure 2.1. The vital aspect of the link-based methodology is to determine the proper location of the cross links.

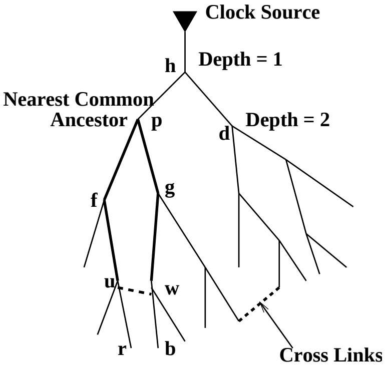


Figure 2.1: An example of cross link based non-tree.

2.1.1 Effect of Link Insertion on Skew Variability

The effect of inserting a link on skew between any two arbitrary points has been analyzed in detail in [54]. Here, we will review some of the important conclusions of [54] for the sake of clarity.

Skew Variability Between Link Endpoints: If a link is inserted between nodes u and w, then according to [54], the skew between u and w after link insertion is given by:

$$\hat{q}_{u,w} = \frac{R_l}{R_l + r_u - r_w}(q_{u,w} + \frac{C_l}{2}(R_{u,u} - R_{w,w})) \tag{2.1}$$

where, $q_{u,w}$ is the original skew between nodes u and w, $\hat{q}_{u,w}$ the final skew after the link insertion, C_l the link capacitance, R_l the link resistance, $R_{u,u}$ and $R_{w,w}$ the transfer resistances of nodes u and w. The above equation can be rewritten as:

$$\hat{q}_{u,w} = (\alpha q_{u,w} + \alpha\beta) \tag{2.2}$$

where $\alpha = \frac{R_l}{R_l + r_u - r_w}$ and $\beta = \frac{C_l}{2}(R_{u,u} - R_{w,w})$.

Since the effect of capacitance can be easily estimated and removed [54], Equation (2.1) gets reduced to:

$$\hat{q}_{u,w} = \frac{R_l}{R_l + r_u - r_w} q_{u,w} \tag{2.3}$$

Thus, from Equation (2.3), we can see that the final skew is a scaled value of the original skew with the scaling factor of $\frac{R_l}{R_l+r_u-r_w}$. It has been proved in [54] that the scaling factor is always less than 1, thereby proving that the skew between nodes u and w is always reduced as a result of link insertion. It has also been proved in [54] that inserting a link as close to sink nodes as possible is better in terms of skew variability reduction. For example, in Figure 2.1, when we want to reduce the skew between nodes r and b, then it is better for the link to be as close to the nodes r and b as possible.

2.1.2 Skew Variability Between Arbitrary Nodes

The effect of inserting a link between nodes w and u on skew between any two arbitrary nodes i and j (ignoring the effect of link capacitance, which can be considered separately) is given in [54] as:

$$\hat{q}_{i,j} = q_{i,j} - \frac{r_i - r_j}{R_l + r_u - r_w} q_{u,w} \tag{2.4}$$

Consider a clock tree $T = (V, E_T)$ as shown in Figure 2.1 with E_T being the solid lines. Let T_i denote the subtree rooted at node i . The node u is in a subtree $T_f \subset T$ and the node w is in another subtree $T_g \subset T$. The root nodes of T_f and T_g are the two child nodes of the node p. Node p is called the Nearest Common Ancestor (NCA) for the nodes u and w. According to [54], when a link is inserted between the nodes u and w the following three scenarios can arise:

Scenario 1: One of i and j is in subtree T_f and the other is in subtree T_g, for example, $i \in T_f$ and $j \in T_g$. In this case, the link addition introduces a correlation between the delays of nodes i and j, which results in reduced skew variability.

Scenario 2: Both i and j are in the same subtree T_f or T_g. In this scenario, the skew variability might increase. Since i and j are in the same subtree, their skew variation in original tree is usually not that considerable.

Scenario 3: One of i and j is in the subnetwork T_p rooted at the NCA node p for nodes u and w and the other node is disjoint with T_p. For example, i is in T_p like b and j is not in T_p like d in Figure 2.1. In this case, there is no predictable correlation between the delays of nodes i and j and so the skew might or might not get reduced.

Therefore, any link insertion scheme must be such that it increases the number of occurrences of scenario 1 while the number of occurrences of scenarios 2 and 3 must be reduced.

2.1.3 Link Insertion Scheme Overview

For a given link insertion algorithm, the overall approach taken in [54] towards building a non-tree clock distribution network is as follows.

1. Obtain initial clock tree from any of the available clock tree routing algorithms in the literature like [10, 17, 74].
2. Given the clock tree, select node pairs where cross links are to be inserted. The node-pairs must be selected in such a way that the scenario 1 discussed

in Section 2.1.2 is maximized and scenarios 2 and 3 minimized. This will make sure that the cross-link addition reduces the skew variability.

3. Since the addition of link capacitance might change the original skew of the clock tree, we need to tune the nodes of the clock tree such that the original skew is not altered. This can be done by considering only the effect of link capacitance on the link end points and tuning the nodes in a bottom-up fashion similar to the method in [75] so as to obtain the original skew after the addition of link capacitance.

4. Finally, the links are added to the selected node pairs. Since the effect of link capacitance has been already removed by bottom-up tuning, only the effect of link resistances will be present, which is very likely to reduce the skew variability.

In the above procedure, step 2 is the key step and has a tremendous influence on the quality of the final non-tree since selecting the wrong node pairs might result in worse skew variability. In the next section, we discuss the drawbacks of the existing link selection algorithms.

2.2 Review of Existing Link Insertion Methodologies

We will briefly review existing methods of [35, 54] and discuss their limitations. A more detailed explanation of the existing methods can be obtained from the corresponding papers.

2.2.1 Rule-based Node-pair Selection Algorithm

The rule based approach of [54] is derived directly from the Equation (2.1). In this method, three rules are defined for node pairs selection, which are given below:

α **rule**: The α value of any link is defined as $\alpha = \frac{R_l}{R_{loop}}$, where $R_{loop} = R_l + r_u - r_w$ is the total resistance along the loop of $p \rightsquigarrow u \rightsquigarrow w \rightsquigarrow p$. Only the links that have α value less than an upper bound value of α_{max} are allowed to be added. The lower the value of α, the lower the scaling of the original skew and the better the link for skew reduction.

β **rule**: The β value of any link is defined as $\beta = |\frac{C_l}{2}(R_{u,u} - R_{w,w})|$. An upper bound of β_{max} is placed on the β value of links that can be added. The β value determines the extra skew introduced into the original clock tree due to the link capacitance. The lower the value of β, the lesser the tuning required in the original tree.

γ **rule**: The γ value of a link is defined as the depth of the NCA of the linked nodes w.r.t the clock source. For the example in Figure 2.1, the NCA p of nodes r and b has depth 2 from the clock source. We call this depth of the NCA from the clock root as the level of the node pair and denote it using γ. While adding links, an upper bound γ_{max} is placed on the value of γ and only links with values lesser than γ_{max} is added. This rule is intended to make sure that the links added does not worsen the skew variation between any two pairs in the clock network significantly.

In this method, any link that has α, β, γ values less than the maximum value set by the user will be added to the tree. The lesser the value of α, β

and γ, the better the effectiveness of the link in skew variation reduction [54]. But choosing too low values for these variables will result in fewer links, thereby reducing the effectiveness of link insertion.

Merits: The rule based algorithm has the main advantage that the physical characteristics of the links to be inserted are considered before inserting the link in the clock network. Moreover, the run-time does not increase drastically as the number of links to be inserted and the clock tree size increases. Since a lengthy link will usually have high values of α and β, shorter links will be preferred. Shorter links will be much better in terms of the routability of the final non-tree. Also, when dealing with non-symmetric clock networks, the method will still work because the rules used does not assume any symmetry in the clock network.

Demerits: The rule based technique does not have any explicit control over the distribution of the links across the clock network. Generally speaking, we would like the links to be distributed across all the regions of the clock network so that the skew variation of all subtrees in the clock network is reduced. In the case of rule-based method, there is a possibility that the links are added only between a few subtrees thereby not controlling the skew variation in other subtrees. For example, in Figure 2.2, there are four different subtrees represented by A,B, C and D. It may happen that all the links might get added between the subtrees B and C only, leaving the skew between subtrees A and D completely uncontrolled. This situation can be avoided by making sure that links get added uniformly across all regions of the clock network.

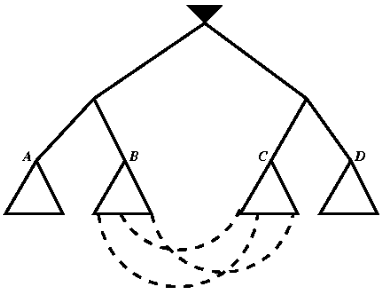


Figure 2.2: An example in which the rule based link-insertion algorithm might fail.

2.2.2 Min-matching Based Node-pair Selection Algorithm

The min-matching based link insertion algorithm of [54] uses the bipartite min-matching algorithm for selecting the node pairs for link insertion. The main idea of this method is to divide the clock network into several regions and making sure that links are added to every region. This is done by dividing the clock network into two subtrees - the left subtree and the right subtree. Each subtree is further divided into k subtrees (denoted as sub-subtrees in the paper). The clock network is represented by a bipartite graph with the two main subtrees as the two sides and the sub-subtrees on either side as the nodes of the bipartite graph with total of $2k$ nodes. The edge weight of the bipartite graph between two nodes is fixed as the minimum rectilinear distance between any two pairs of sinks of those sub-subtrees. For example, in Figure 2.3, the edge weight between nodes 2 and b is the shortest rectilinear distance between any two pairs of sinks of the sub-subtrees 2 and b. Solving the min-matching of this bipartite graph will give us k links with minimum total wire-length. This will also make sure that every sub-subtree in the clock network is linked to another by a cross link. This idea is illustrated in Figure 2.3.

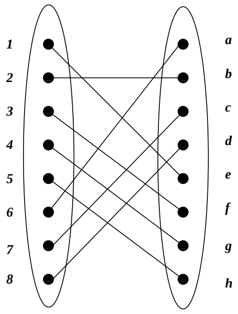


Figure 2.3: A bipartite graph model for selecting node pairs between two subtrees. Each subtree is further divided into a number of sub-subtrees for link insertion. An edge weight between two nodes is the shortest rectilinear distance between leaf(sink) nodes of two sub-subtrees.

Merits: The key advantage of the min-matching algorithm is that it guarantees even distribution of links across the clock network. As a result, this method out-performs the rule-based method in skew variation reduction. Also, because of the min-matching nature of the algorithm, it generally gives lesser total wire-length when compared to the rule-based method.

Demerits: The complexity of the min-matching algorithm (including the construction of bi-partite graph) is $O(n^3 + n^2m^2)$ where n is the number of nodes in bipartite graph and m is the maximum number of sinks within a sub-graph. This is an important disadvantage because the run-time will drastically increase as the size of the clock network increases. Also, since min-matching does not control the length of individual links, there is high probability of adding lengthy links. This might make the chip difficult to route. For example, in Figure 2.3, the links between the nodes 6 and a and nodes 1 and e may be very long. The main reason for lengthy links is that the min-matching algorithm allows a given subtree to be connected to exactly one subtree only. This problem will get worse as the number of links that we want to add increases. Thus, this method will hit a brick-wall in terms of the number of links that can be inserted at a particular level.

Another disadvantage in having lengthy links is that any problem in even one of the links will drastically affect the effectiveness of the non-tree for skew variability reduction. Possible situations in which this can happen are manufacturing defects in the links and unroutability of a link because of the initial clock tree routing. In such cases, having several small links is better than a few lengthy links. Another demerit of the min-matching algorithm is that it inherently assumes a reasonable amount of symmetry in the structure of the clock tree. Otherwise, it might not be very effective in reducing the skew variation. For example, when there are two subtrees on one side of the bipartite graph and five on the other side, then the min-matching algorithm will result in only two links, thereby leaving some of the subtrees without any cross links.

2.2.3 Statistical Link Insertion

The statistical link insertion method of [35] adds the links incrementally by updating the statistical values of the sink delays and using the information of the latest non-tree to select the next link to be inserted. This method has the advantage that every link insertion considers the effects of all the previously inserted links, unlike the methods of [54]. Another important advantage is that it does not require the user to select any empirical parameters for link insertion. However, the method of [35] suffers from the disadvantage that the run-time is very high even for clock networks of relatively small size. Also, the increase in run-time as a function of the clock network size is exponential. For example, clock networks with 74, 179 and 597 sinks take roughly 3, 50 and 655 minutes respectively[35]. As a result, the method of [35] cannot be used for bigger clock networks. Another point to be noted here is that the skew variation reduction as a result of link insertion is independent of the source of skew. This can be easily verified from Equation(2.2) in which the skew $q_{u,w}$ gets scaled down by the value of α irrespective of the source of $q_{u,w}$. Thus, considering all the independent variables like wire widths, load capacitance etc. (as is done in [35]) for every link insertion might be an overkill.

Based on the discussion in the above paragraphs, the requirements of an ideal node pair selection algorithm are:

- The algorithm must efficiently distribute the links across all the sections of the clock network. If an algorithm fails to achieve this, it might lead to high skew for the sinks in the region where links are not added. This is mainly because, an even distribution of links across all regions of the clock tree is likely to perform better when there is random variations across all parts of the clock tree. Also, an even distribution of links will have a better chance of evenly distributing the negative effects of any links that might result in scenarios 2 and 3.
- It must have a low complexity w.r.t. both the number of links added and the size of the initial clock tree.
- It must make sure that very lengthy links are avoided and the total wire-length is reduced. Reduction in the link length might be useful in easily routing the clock network and a low total wire-length will help reduce both area and power. We wish to point out here that we use the average link

length as an estimate of the routability of the link-based non-trees. Though this assumption need not be true always, shorter links are easier to add in general. Moreover, since we consider link addition as a post-detailed-routing step to reduce variation, we believe that it is better to have fewer, shorter links for a given skew variability reduction.

- The algorithm should be capable of handling asymmetric clock trees as well.
- The algorithm should avoid use of empirically chosen parameters to obtain a good non-tree.
- The effect of the previously selected links must be considered before selecting the subsequent links for insertion.

In the following sections, we present our “Rule Delta” algorithm , “MST with Rule Based Deletion” algorithm, and “Incremental Link Insertion” algorithms in which the drawbacks of the existing algorithms are addressed.

2.3 Rule-Delta Algorithm

The rule-delta algorithm is an improvement of the rule based node pair selection algorithm proposed in [54]. The key drawback of the rule-based algorithm is that it cannot guarantee an even distribution of links across all subtrees of the clock network. To overcome this drawback, we propose a new rule called δ rule in addition to the original α, β and γ rules. The δ rule is explained below.

δ **rule**: Let δ be a valid node level (level = depth of the node) from the clock source. According to the δ rule, no two links should have the same pair of ancestors at the δ level from the clock source. When this condition is added to the above rule-based link addition, we can control the distribution of the links among the different sub-trees of the clock tree. For example, in Figure 2.2, the problem that we would like to avoid is the crowding of the links between the subtrees B and C which leaves the subtrees A and D unconnected. When we apply the δ rule here with the value of $\delta = 2$, then no more than one link will be inserted between the subtrees B and C.

An important fact to be noted regarding the δ rule is that the value of δ must be greater than the γ value used. Otherwise, the number of links added will be significantly reduced, thereby reducing the effectiveness of the link insertion. When the value of δ is increased, more links will be added because of the increase in the number of permitted ancestor node-pairs for link insertion. This will allow the addition of sufficient number of links in a way that they get distributed across all sections of the clock network.

The advantages of Rule-Delta method are listed below:

- The 'rule-delta' algorithm can make sure that the links do not get crowded in the same regions of the clock network. By choosing a proper selection of the α, β, γ, and δ values, we can make sure that sufficient number of links get added across all sections of the clock network.
- Since the algorithm just performs a sweep for the different possible links based on the rule values, it is very efficient in terms of the run-time.
- The effectiveness of the links are assessed by the rules before adding it.

As a result, the situation of addition of lengthy links does not happen in practice. This is because the lengthy links will have a very high value of both α and β, which can be removed by using the appropriate bounds for the values of α and β.

- Since the rules are independent of the structure of the initial clock tree, it can handle even highly unbalanced clock trees.

It might be noted here that the last three advantages are shared by both the rule-based and the rule-delta algorithms. The rule-delta algorithm inherits all the advantages of the rule-based algorithm while overcoming the shortcomings.

2.3.1 Parameter Determination

Parameters α and β: An important observation we made from our experiments is that, when only one among the parameters α, β is changed, the skew variability reduction tends to follow the *Law of Diminishing Returns* w.r.t. the amount of extra wire-length added. In other words as more links are added to a given tree by changing only one parameter, the amount of extra reduction in skew variability diminishes. Figure 2.4 illustrates this fact for test case r3. The plot is obtained by varying only the parameter α, keeping all the other parameters constant, to increase the number of links added. We observe this behavior in all our test cases.

In order to determine the effect of α in reducing the skew variability, we obtain several link based non-trees and their skew variability by varying the value of α for a fixed set of values of other parameters like β, γ etc. We repeat this experiment for several sets of parameters. Figure 2.5 shows plots of relative skew

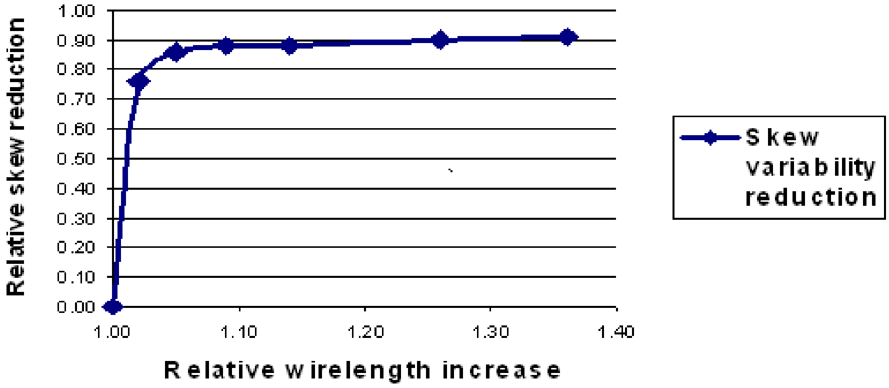


Figure 2.4: An example of *The Law of Diminishing Returns* for link addition. As more links are added to a given tree, the extra skew reduction is limited

reduction and wire-length increase (w.r.t. the values of tree) for three different sets of parameters. As seen from Figure 2.5, irrespective of the values of other parameters like β, γ and δ, the *Law of Diminishing Returns* holds. The only difference between the different sets of parameters is the point of attainment of the saturation value of skew reduction.

Thus, given the values of other parameters, we can obtain a good value of α based on the skew vs. wire-length trade-off of a particular design by obtaining a few non-trees with different α values and determining their skew variability. Theoretically, the value of α is always between 0 and 1. In practice, we found that the maximum value of α can be fixed at 0.3 irrespective of the clock tree size. This is because for links with *alpha* greater than 0.3 the length of the *link* becomes very comparable to the total length of the source to sink path. This statement can be easily verified from the definition of α in Section 2.2.1.

The parameter β also obeys the *Law of Diminishing Returns*. However, unlike α, the value of β can theoretically take any value. From the definition of β in Section 2.2.1, we can see that the value of β can be considered as the skew introduced by adding only the link capacitance in the original tree. From our experiments, we found that links with value of β of greater than 5% of the nominal delay never gets inserted because such links usually have very high value of α also. Also, a β value of more than 5% means that a lot of wire snaking might get introduced in the tuning step (step 3) of Section 2.1.3. In general, the maximum value of β can be selected as a fixed percentage of nominal delay by the user.

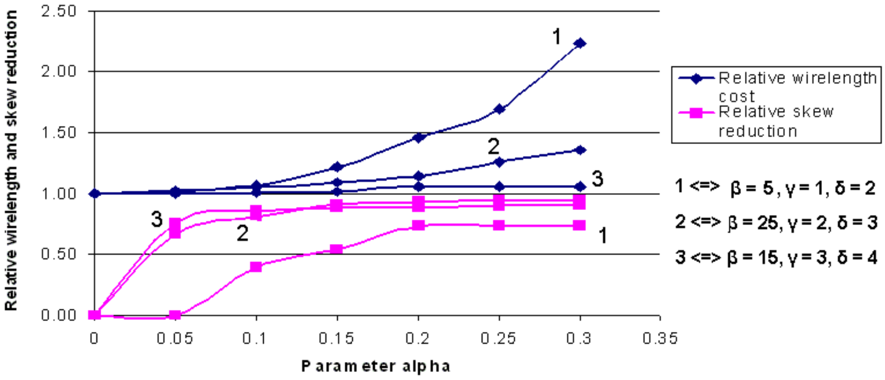


Figure 2.5: Effect of α on skew reduction and wire-length increase. Parameter β also exhibits similar behavior

Parameters γ and δ: Unlike the α and β parameters, the γ and δ parameters are integers. This is because these two parameters always denote the *depth* of some node in the original clock tree w.r.t. the clock source. Another key observation is that the total number of possible values of γ and δ parameters is finite because the *depth* of any node in a given clock tree is finite and is also known for a given clock tree. Thus, it is relatively easier to obtain the values of

γ and δ than α and β.

In order to get a good set of links, we found that the maximum value of γ should be much less than half the depth of the clock tree. This can be explained by the fact that lesser values of γ, more the probability of scenario 1 and lesser the chances the scenarios 2 and 3 discussed in Section 2.1.2. Thus, we can fix the value of γ to be from 0 to $\frac{CDN\_Max\_Depth}{2}$, where CDN_Max_Depth is the maximum depth of any node in the original clock tree from the clock source.

As explained in Section 2.3, the value of δ should be greater than the value of γ. Thus, the value of δ is also bounded between γ and CDN_Max_Depth.

Based on the above discussions, we can conclude that the values of the different parameters are bounded for all practical purposes. Also, the parameters α ans β obey the *Law of Diminishing Returns*, and parameters γ and δ are bounded integers with a small range. These properties can be used to perform an *intelligent search* of the solution space instead of a blind brute force search based on the theoretical bounds. At each step of the search, Elmore delay Monte Carlo analysis can be used to guide the selection of parameters. Since our non-trees are *tree like*, evaluating their delays and skews can be done efficiently using the delay evaluation procedures of [7]. As a result, the Elmore Monte Carlo analysis can be done quickly.

We have implemented a Perl routine that performs the intelligent search outlined above. The routine uses the Elmore delay Monte Carlo analysis with fewer trials to obtain the skew variability information of a given non-tree. This information is used to guide the parameter selection process. Using fewer Monte Carlo trials helps in reducing the overall run-time in selecting the parameters.

However, it might increase the probability of selecting a non-tree that is far from optimality when a full fledged HSPICE based Monte Carlo analysis is done. To overcome this disadvantage, we first select the top few non-trees based on the Elmore Monte Carlo results. HSPICE Monte Carlo simulations are performed on these selected non-trees to select the final non-tree to be used. This procedure is much faster than running lots of Elmore Monte Carlo trials to select a single non-tree and typically result in the same or very similar non-tree.

The different inputs required by this procedure are:

- $Minimal\_Skew\_Gradient$: It is defined as the minimum reduction in the value of skew per unit wire-length increase that is acceptable. This parameter can be set based on the wire-length cost vs. skew benefit requirements of the clock network. This parameter gives the user a direct control over when to stop adding links.
- The maximum values for the parameters α, β , γ and δ: the value of α is set at 0.3, the maximum value of β is set at a user defined percentage of the nominal delay. The limits of γ and δ are automatically obtained by the script based on the value of CDN_Max_Depth.

The procedure first constructs the non-tree with the minimal value of α, β, γ and δ parameters. The skew variability of this non-tree is obtained using Elmore Monte Carlo analysis. Next, the value of α is increased by a small value and a new non-tree and its skew variability are obtained. Since the parameter α obeys the *Law of Diminishing Returns*, we can use the steepest descent method [30] and the value of $Minimal\_Skew\_Gradient$ to determine the point at which an increase in alpha results in minimal skew variability reduction. The results of the non-tree at this point is stored and the procedure is repeated by changing the other parameters. This process is repeated until all the values of the parameters reach their maximum values. Since this procedure searches the entire solution space in an efficient manner, it is possible to automatically obtain a good nontree with reduced skew variability in a reasonable amount of time. The experimental validation for this procedure is presented in Section 2.7. Please note that this procedure is a lot faster than both manual tuning and also exhaustive search in the parameter space using only their theoretical bounds.

2.4 MST Based Link Insertion Algorithm with Rule Based Deletion

The MST based node pair selection algorithm is an alternative graph theoretical approach to the min-matching based node pair selection algorithm proposed in [54]. As pointed out in Section 2.2.2, one of the key demerits of the min-matching algorithm is the addition of lengthy links. One way to avoid adding lengthy links is to allow a given node to be connected to more than one node on the opposite side of the bipartite graph. Also, not more than one link should be allowed between any given pair of nodes to avoid crowding of links. When these two conditions are satisfied, it is likely that the selected links will not be too lengthy and at the same time, all the sub-subtrees are linked. This is illustrated in Figure 2.6 in which there are no extremely lengthy links, unlike in Figure 2.3.

To satisfy both these conditions, we propose to construct a MST in the complete bipartite graph. As in [54], the edge weight between any two nodes

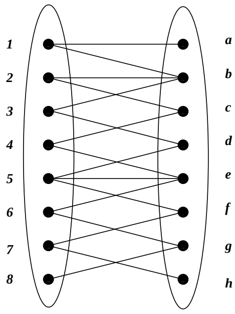


Figure 2.6: An example of a new approach that might be better than the min-matching based link insertion.

in the bipartite graph is given as the minimum rectilinear distance between any two pairs of sinks of those two sub-subtrees in the clock network. Constructing a MST from a complete bipartite graph will invariably allow multiple links for a few nodes in the graph. It will also avoid crowding of links between any two nodes. One possible problem that we might encounter in constructing a complete MST is that the total wire-length might become very high. To solve this problem, we selectively remove links from the selected MST edges based on values of the rules of α and β used in the rule based node pair selection algorithms.

Now, we will consider the complete flow of the MST-based link insertion algorithm. From the conclusions in Section 2.1.2, we know that we need to avoid scenario 3 so as to reduce the skew variability. Scenario 3 can be avoided by choosing node pairs between left child subtree and right child subtree of a node of depth 1 from the clock source. For example, in Figure 2.1, the links between subtree T_p and subtree T_d of depth 1 can avoid scenario 3, since there is no sinks outside of T_h. Node pairs for these links can be characterized by the depth of their NCA node h, which can be called γ level as in [54]. Therefore, node pairs with $\gamma = 1$ must be present to effectively reduce the skew variability.

```
Procedure: $Select\_Node\_Pairs(T_v)$
Input: Subtree $T_v$ rooted at node $v$
Output: Node pair set $P$
1. $l \leftarrow$ left child node of $v$
2. $r \leftarrow$ right child node of $v$
3. $P \leftarrow Pair\_Between\_Trees(T_l, T_r)$
4. If $Depth(v) == Max\_Depth$, return $P$
5. $P \leftarrow P \cup Select\_Node\_Pairs(T_l)$
6. $P \leftarrow P \cup Select\_Node\_Pairs(T_r)$
7. Return $P$
```

Figure 2.7: The top-level algorithm of selecting node pairs for link insertion.

The links inserted between subtrees T_d and T_p will improve skew variability between most sink pairs of those subtrees according to analysis of *scenario 1*. However, these links might worsen the skew variability between sinks pairs within T_d or T_p as discussed in *scenario 2*. This might harm the overall skew variability. This situation can be avoided by inserting links between sub-subtrees within subtree T_d or T_p. In other words, node pairs of $\gamma = 2$ need to be considered for link insertion. This procedure can be repeated recursively till γ is sufficiently large. The subtrees that correspond to large γ are mostly small and usually the skew variability inside is negligible. The main algorithm description on this recursive procedure is given in Figure 2.7.

The most important of all the steps in the algorithm in Figure 2.7 is step 3 in which the node pairs are selected between the two given subtrees. This step is illustrated in Figure 2.8.

The advantages of MST based link pair selection method are:

- Since the MST based method divides the clock network into subtrees, it can guarantee an even distribution of links across all the regions of the

```
Subroutine: $Pair\_Between\_Trees(T_l, T_r)$
Input: Two subtrees $T_l$ and $T_r$
Output: Node pair set $P$
A. Decompose $T_l$ into sub-subtrees $S_l = \{T_{l1}, T_{l2} .... T_{lk}\}$
B. Decompose $T_r$ into sub-subtrees $S_r = \{T_{r1}, T_{r2} .... T_{rk}\}$
C. For every pair ($T_{li}$, $T_{rj}$) between $S_l$ and $S_r$
D.     Weight($T_{li}$, $T_{rj}$) = Min distance between leaf
           pairs in $T_{li}$ and $T_{rj}$
E. Find the MST of the complete bipartite graph
      between $S_l$ and $S_r$.
F. Selectively remove the edges that have $\alpha$ and $\beta$
      values higher than the set limits of $\alpha_{max}$ and $\beta_{max}$
G. $P \leftarrow$ Selected links of MST.
H. Return $P$
```

Figure 2.8: MST based algorithm of selecting node pairs for link insertion.

clock network. This advantage is shared by both MST algorithm and the min-matching algorithm, unlike the rule-based algorithms.

- The overall complexity of the algorithm is $O(nlogn + n^2m^2)$ where n is the number of nodes in bipartite graph and m is the maximum number of sinks within a sub-graph. This is an important improvement over the $O(n^3 + n^2m^2)$ complexity of the min-matching based algorithm, especially for large clock networks.

- Since multiple links are allowed to be connected to a given node in the bipartite graph, the chances of adding a lengthy link is greatly reduced when compared to the min-matching algorithm.

- The MST based algorithm will work better than the min-matching based algorithm in the case of an unbalanced clock tree. This is because, when the two sides of the bipartite graph have unequal number of nodes, then some nodes will not be linked while using the min-matching algorithm. But in the case of the MST algorithm, all the nodes are guaranteed to be connected to at least one node on the opposite side of the bipartite graph.

- The use of rule based deletion makes sure that the physical characteristics of the links are taken into account before the addition of the link. This will make sure that bad links do not get added to the clock network.

2.4.1 Parameter selection

From the MST based algorithm described in Section 2.4, we see that the value of Max_Depth (used in Figure 2.7) for a given clock network must be specified. Also, for each of the levels, the value of k (used in Figure 2.8) that determines the number of nodes in the MST (and the number of links added) should be specified. One straightforward fact regarding these parameters is that all of them are integers. Also, the value of Max_Depth is bounded by the value of CDN_Max_Depth. Also, the value of Max_Depth determines the trade-off between the number of links and the occurrences of scenarios 1, 2 and 3 of Section 2.1.2. For higher values of Max_Depth, the links added increase the likelihood of occurrences of scenarios 2 and 3. However, selecting a very low values of Max_Depth will add too few links. In practice, the value of Max_Depth can be varied between 0 and $\frac{CDN\_Max\_Depth}{2}$. In all our test cases, the extra skew reduction was almost negligible for using any value close to or greater than $\frac{CDN\_Max\_Depth}{2}$ with an exponential increase in the additional cost. The exponential increase in cost can be explained by the fact that, for each level increase in the value of Max_Depth, the number of eligible links doubles.

In addition to selecting the value of Max_Depth, the value of parameter k at each level is to be determined. The total number of different values of the parameter k at each level is bounded by the value of CDN_Max_Depth. This is because, in order to obtain similar sub-subtrees for constructing the bipartite graph, it is easier to sub-divide the clock tree according to the existing topology of the clock tree. For example, lets us consider a *complete* binary tree without any loss of generality. For such a tree, it is easier to divide the left or right subtree into 4 sub-subtrees each instead of 5 sub-subtrees each. Thus, the different values that the parameter k can take at the topmost level is all possible values of 2^i where i is an integer value from 0 to $CDN\_Max\_Depth - 1$ (assuming that sink level is not considered for sub-sub tree generation). Thus, there are only CDN_Max_Depth different values of k at the topmost level. In general, for any given level *Depth*, the different values that k can take is given by 2^i where i is an integer value from 0 to $CDN\_Max\_Depth - Depth$. It maybe noted here that, the above conclusion of a fixed number of possible values of k is valid even for incomplete binary trees. The only difference is that the values of k need not be exact powers of 2.

Another useful observation is the relation between the k values of different levels. In general, it is better to have the highest value of k at the topmost level ($Depth = 0$) because, the links added at the topmost level maximize the probability of scenario 1 of Section 2.1.2. Similarly, for every successive level, the value of k can be fixed such that it is not more than the value of k used in the previous level. Thus, for the entire clock network, the different values of k forms a non-increasing series as the value of *Depth* increases. The above observations can be used to automatically determine the parameters for the MST based link insertion algorithm.

Based on the above discussion, we can see that for a given clock tree with known depth, the different set of possible non-trees using MST based link insertion algorithm is finite because all the parameters are tightly bounded integers. As a result, it is possible to quickly obtain a good set of parameters for a non-tree using a guided search similar to the one discussed in Section 2.3.1, with the changes only in the parameter determination part. The parameter determination for MST algorithm is much easier and quicker when compared to that of the rule-based algorithm. The experimental validation for this method is also presented in Section 2.7.

A note on choosing α, β for MST algorithm: We would like to point out here that in the case of the MST based algorithm, the values of parameters α_{max} and β_{max} are usually set to a very loose bounds to avoid very bad links. Unlike the rule-delta algorithm in which these parameters are used to select the links, in MST algorithm, their main purpose is to filter out very bad links.

2.5 Incremental Link Insertion

2.5.1 Motivation for Incremental Link Insertion

Though the two link insertion algorithms presented in Sections 2.3 and 2.4 address several drawbacks of the two algorithms of [54], all four of these algorithms have an important drawback in common. The drawback is that the effect of the links on each other is not considered. For example, in Figure 2.9, let us assume that *link a* is already selected for insertion and that before the insertion of *link a*, the *link c* had a lower α value when compared to *link d.* As a result, the rule-based algorithms will choose *link c* over *link d*. However, after inserting *link a*, the delays of all the nodes become correlated and the delay of every sink in the clock network is at least loosely correlated with the delays of every other sink. This is because, after inserting the first link *link a*, the values of r_u and $R_{u,u}$ used in Equation (2.2) will change for every sink, resulting in changed the values of the α and β for all possible links. Hence, the new values of α for *link d* might be lower than that of *link c* as a result of which it might be better to choose *link d*. In other words, even though *link c* is better than *link d* for the initial clock tree, *link d* might become better after the insertion of the first link.

Similarly, in the case of the graph theory based methods, the only physical characteristic of the link considered is the length of the link. However, for similar reasons as outlined in the previous paragraph, the link with minimal size might not be the best link to be added after an initial set of links have been inserted. As a result, any methodology that considers the effect of the previously selected links before selecting the next link might perform better in skew variability reduction. It can be noted here that though the algorithm of [35] does consider the effect of previously inserted links, it is extremely slow even for small test cases.

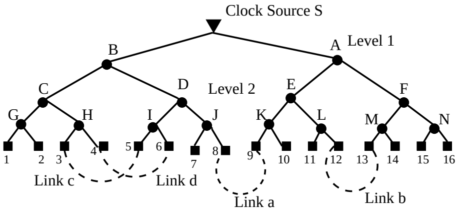


Figure 2.9: A simple nontree where incremental approach might help.

Another drawback shared by all four algorithms is that they require the user to empirically select the values of different parameters. Though the methods described in Sections 2.3 and 2.4 automates the parameter determination, a link insertion method without any empirical parameter selection might be better in terms of run time and efficiency. In summary, our objectives are:

1. The *effect of previously inserted links* on variability reduction must be considered before selecting the next link. This might help in selecting better places for link insertion.
2. *Automatic link selection* without requiring the users to empirically select the parameters is required. This will allow for a complete automation of link insertion and save the time spent in selecting the appropriate parameters.
3. Fast and Efficient incremental link insertion method will be required. This will allow us to insert links even in bigger clock networks.

Next, we present the details of our incremental link insertion algorithm, which satisfies the above three objectives.

2.5.2 Automatic Link Selection

Consider the Figure 2.9. The dashed lines represent the links and the solid lines represent the given clock tree. In the original tree, the nominal Elmore delay (w.r.t. the output node of buffer S) of sinks 1 and 16 are practically independent of each other. However, if the *link a* is added between the sinks 8 and 9, even the nominal delays of sinks 1 and 16 become somewhat correlated due to the fact that both the sinks now have more than one source to sink paths and both sinks share the $S \rightarrow A$ and $S \rightarrow B$ paths. It is because of this correlation that the skews of a link based non-tree due to random variation gets reduced.

Among the links shown in Figure 2.9, *link a* is usually the best because it links two sinks that are hierarchically farthest apart and physically closest. After the insertion of *link a*, the delays of all sinks are correlated with the delays of all the other sinks. The best position to insert a link will be in such a place where the correlation between the sink delays further increases. To the best of our knowledge, the only known way to reliably identify the correlation between the different sink delays is to perform Monte Carlo Analysis. This is because of the absence of any closed form solution to obtain the correlation between the delays of the sinks of a general (non-tree) RC structure. However, using Monte Carlo analysis to select each link incrementally might be very time consuming.

We propose to overcome this problem as follows. From the definition of α in Equation (2.2), we can observe that a link with the least value of α is usually among the best links to insert because α measures the scale by which the original skew gets reduced. Because of this, α value can be used as a measure of selecting the links. However, obtaining the α value for all possible links for a given non-tree is non-trivial and will require the computation of Elmore Delay for each sink pair (u, w) with $C_u = +1$ and $C_w = -1$. Thus, for n sinks, $O(n^2)$ evaluation of Elmore delays is necessary. Though the methods in [54, 60] make use of the α parameter, they obtain the α values only for the initial clock tree and they *never update* the values of r_u and r_w used in Equation (2.1). The initial values of r_i for any node i will be equal to the sum of all the resistance in the source$\rightarrow i$ path. After inserting the first link, because of the existence of multiple source to sink paths, there is no closed form expression to obtain the correct values of r_i. Thus, [54, 60] uses only the initial α values of the links which might be different from the values after inserting even the first link. If we can find a method to efficiently update the values of α for all possible links after each link insertion and select the next link based on the latest clock network structure, it will solve our objective 1. We describe such a method in Section 2.5.3 in which we make use of *equivalent* r_i, denoted by E_r_i which we use to evaluate the value of α defined in Equation (2.2).

It may be noted here that we update only the nominal values of circuit parameters, unlike [35] in which the statistical values like mean, variance are updated. By using only the nominal circuit delay values, we make use of the fact that skew reduction due to link addition is independent of the source of skew. Hence, we avoid time consuming steps such as the statistical method of [35] or Monte Carlo simulations to select links. Our objective 2 can be met by adding only the link with the least value of α each time and the stopping criterion can be based on the maximum wire-length increase that the user can accept.

2.5.3 The Algorithm

Figure 2.10 shows our top-level algorithm, which is an incremental equivalent to the one-step algorithms of [54, 60]. The key subroutines of the algorithm are $Pick\_Best\_Link$, $Node\_Tune$ and $Update\_Equivalent\_r_i\_Values$. The procedure $Pick\_Best\_Link$ picks the best link for a given non-tree (or the initial tree) based on the *latest* values of α. The $Update\_Equivalent\_r_i\_Values$ procedure *efficiently* obtain the correct values of α parameter of the latest non-tree. A point that may be noted here is that the method of [35] does not eliminate the adverse effect of the link capacitance after selecting a particular link. In this work, we use the $Node\_Tune$ procedure to efficiently eliminate the negative effect of link capacitance. As a result, the scaling effect of each link on unwanted skew can be expected to be much higher when compared to the method of [35]. This fact can be verified from Equation (2.1) in which the link capacitance might increase the final skew.

```
Procedure: $Insert\_Links(CDN_s)$
Inputs: Clock Network $CDN_s$ rooted at source $s$,
        MLC = Max_Link_Cost.
Initialization: Set L = {}, Link_Cost = 0.
Output: Link node pair set $L$.
1. $link = Pick\_Best\_Link(CDN_s)$.
2. If $Link\_Cost + len(link) > MLC$ Go To 8.
3. $L \leftarrow L + link; Link\_Cost += len(link)$.
4. Add link capacitance to selected node-pair n1,n2.
5. $Node\_Tune(CDN_s, n1, n2)$ to restore skew.
6. Add link resistance between nodes $n1, n2$.
7. $Update\_Equivalent\_r_i\_Values(CDN_s, link)$.
   Go To step 1.
8. Return $L$
```

Figure 2.10: The top-level incremental link insertion algorithm.

The *Pick_Best_Link* procedure used in step 1 of Figure 2.10 is described in Figure 2.11. The best link picked by the *Pick_Best_Link* procedure is added only when adding it does not increase the wire-length consumption of the clock network beyond a user-set wire-length bound. This is shown in the steps 2 and 3 of the Figure 2.10. In the step 4 of Figure 2.10, the link capacitances are added to the two sink nodes after which the *Node_Tune* step adjusts the locations of the internal nodes to restore the original skew. The *Node_Tune* procedure used in this method is slightly different from the similar method used in [54]. The difference comes because of the fact that only one link is added at a given point of time in the incremental link insertion. As a result, only the nodes in the source to sink path for the selected sink nodes need to be tuned. Also, since addition of link resistance between equi-delay nodes does not affect the nominal delays, the presence of link resistance can be ignored during this step.

```
Procedure: $Pick\_Best\_Link(CDN_s)$.
Input: Clock Network $CDN_s$ rooted at source $s$.
Output: The best link based on updated $\alpha$ value
of the non-tree(or the initial tree)
1. $l \leftarrow$ left child node of $s$.
2. $r \leftarrow$ right child node of $s$.
3. For all pairs $(i, j), i \in$ sinks(l) and $j \in$sinks(r).
     $link$ = Link node pair $(i, j)$ with min value of $\alpha$.
4. Return $link$
```

Figure 2.11: The method of picking the best link based in α value of the non-tree.

After the internal nodes have been tuned in the step 5 of the top-level algorithm, steps 6 and 7 add the link resistance and then update the values of E_r_i for each node i using the $Update\_Equivalent\_r_i\_Values$ subroutine. The different steps of the $Update\_Equivalent\_r_i\_Values$ subroutine are described below.

1. Remove all link resistances from the CDN_S, including the previously added links. This will make the structure a tree with only the link capacitances.

2. Set the values of all sink node capacitances to be +1 and all the internal node capacitances to be zero.

3. Evaluate the Elmore delays for all the sink nodes in the clock tree.

4. Using this initial Elmore delay values and the iterative procedure of [7], evaluate the final Elmore delays by iteratively adding all the link resistances. The final value of the Elmore delays of the sink nodes i gives the value of the *equivalent* r_i, denoted by E_r_i for each sink i.

The key idea in the above procedure is in the step 2 in which we set the values of all the sink capacitance to be +1 and all other internal node capacitance to be 0. Please note that the values of resistances are still maintained at their original values. The reasoning behind this procedure is explained below. We know that, for a tree structure, the value of r_i (used in Equation (2.1)) for any node i equals the total value of resistance of the *source* $\rightarrow$ i path. However, such a simple evaluation is not possible for a non-tree and we would like to get a similar parameter that captures the effect of equivalent resistance between the source and the sink node i for any given non-tree. One possible parameter that satisfies this requirement is the value of Elmore delay to a sink node i with all the sink capacitances values set to 1 and other capacitances set to zero. The Elmore delay so evaluated will be a weighted sum of terms involving the resistances of *all the source* $\rightarrow$ *i paths.* This value of Elmore delay will enable us to capture the effect of *equivalent* resistance between the source and the node i in an efficient manner.

The main advantage of the above procedure is that, for a given non-tree, a single evaluation of Elmore delays using the method of [7] will enable us to obtain the values equivalent to that of E_r_i value of every sink. Once we have the E_r_i values for all the sink nodes i, we can evaluate the α value of each possible link quickly using the E_r_i values instead of r_i values. This advantage is made use of in the *Pick_Best_Link* procedure. Thus, we can efficiently obtain the values of α for all possible links in a given clock network.

Given an initial clock tree and an upper bound on the total wire-length that the user wants to add, our top-level procedure incrementally chooses the best link based on the α value, tunes the locations of the internal nodes to eliminate the adverse effects of the link capacitance, obtains the values of E_r_i for all nodes i and uses them in the next iteration to choose the next best link. Thus we see that our incremental approach has addressed all the three objectives that we discussed in Section 2.5.1.

2.6 Application of Link Insertion Algorithms to Non-zero Skew CDNs

In this section, we demonstrate the use of the link insertion algorithms for reducing the skew variability of non-zero skew clock networks. Let us assume that the clock network shown in Figure 2.12 is a non-zero skew clock network. Without any loss of generality, we can assume that the required nominal delays at the nodes A and B are different from each other and that the nominal delay of B is higher than that of node A. In this case, the clock network has been designed to have a particular nominal skew between the two nodes A and B. If the nodes A and B are sequentially adjacent to each other, then this skew is called as *useful skew* [74]. A general requirement for the *useful skew* clock networks is that the skew between the sinks must remain very close to the *nominal useful skew* even under variation effects to avoid race conditions [21]. However, in a useful skew clock tree such as the one shown in Figure 2.12, the skew between sinks such as A and B might be high due to variation effects and might cause a race condition. This problem can be addressed effectively by adding a link between node A and a node P in the root→B path such that nodes A and P have the same nominal delays *after the link insertion.* Such a link addition is likely to keep the skew between nodes A and P as close to the nominal value of zero as possible. This is because the functioning of the link between nodes A and P is similar to the links between two zero skew sinks. Also, in most cases the length of path from node P to node B is small when compared to the root→B path. Because of this, the variation in the $P \rightarrow B$ path will be considerably less than the original variation in the root→B path. Thus, the skew between nodes A and B is likely to be close to the nominal skew after link insertion when compared to the tree.

The *post link-selection* steps for the non-zero skew CDNs are the same as the zero-skew clock networks, but for a minor, but important change. The change is in the tuning step in which the internal nodes are adjusted to restore the original skew considering the effect of link capacitance. For example, in Figure 2.12, let P' be the node in the original clock tree such that it has the identical delay as node A. Adding a link directly between nodes A and P' will change the delay values at both P' and B, thus changing the original skew between nodes A and B. However, the delay between the nodes P' and B will be the same as the original skew between A and B. Thus, to restore the original useful skew, the nodes in the root→ P' path much be tuned such that the node P' achieves the same delay as node A, *without changing the location of node* P'. This step is similar to the procedure employed in [12] to build non-zero skew trees. At the end of the tuning, the node P' will have the same delay as A and is the required node P. Please note that all our link insertion algorithms, including the ones proposed in [54], can be used to select node pairs for non-zero skew clock networks. We present the experimental verification for our approach in Section 2.7.

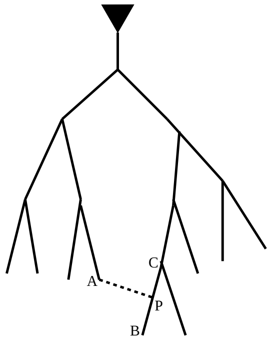


Figure 2.12: Illustration of link addition for non-zero skew clock trees. Link is added between equi-delay nodes A and P.

2.7 Experimental Results

In this section, we present the comprehensive experimental results for all the algorithms proposed in the paper. For the sake for clarity, the order of the results is the same as the order of presentation of the different algorithms. First, we discuss the common experimental setup for all the algorithms. Next we present the results of the rule-delta and MST based link insertion algorithms which is followed by results of our automatic parameter-selection schemes. This is followed by results of the incremental link insertion algorithm. Finally the results for link insertion in non-zero skew clock networks are presented.

2.7.1 Experimental Setup

To facilitate the comparison between the proposed algorithms and the algorithms of [54], we made sure that our experimental setup is identical to that of [54], i.e., r1-r5 benchmarks obtained from GSRC Bookshelf [26]. The variation factors considered in our experiments are also identical to [54] namely, the clock driver resistance, wire width and the load capacitance of all sinks. The driver resistance, wire width and the sink capacitance have ±15% variation following a normal distribution. This corresponds to a normal distribution with standard deviation of 5%. For all the clock networks, a Monte Carlo simulation of 1000 trials is performed using HSPICE. The maximum skew variation in the simulations is denoted by MSV and the standard deviation is denoted by SD. These values, along with values of wire-length and run-time are compared among clock trees, tree+links with algorithms of [54] and trees+links with our proposed algorithms. The size of benchmark circuits, skew variations and wire-length of clock trees are given in Table 2.1.

TC	# sinks	MSV	SD	WL	CPU(s)
r1	267	131	31	1320665	1
r2	598	406	79	2602908	3
r3	862	457	109	3388951	4
r4	1903	1390	380	6828510	12
r5	3101	3270	798	10242660	18

Table 2.1: Maximum skew variation (MSV), standard deviation (SD) and total wire-length (WL) of trees. The CPU time is the time for generating the tree using BST [17] code.

2.7.2 Rule-Delta and MST Algorithms

It has been shown in [54] that the min-matching algorithm performs better than the rule-based algorithm for all the test cases r1-r5. So, we compare our results only with the min-matching based link insertion from [54]. Table 2.2 compares the min-matching algorithm of [54] with the different link insertion algorithms. In the table, MM denotes the min-matching based link insertion of [54], RD denotes the rule-delta algorithm and MST denotes the MST based algorithm. It may be noted here that *all the values of MSV, SD and WL are reported in Table 2.2 are ratios* with respect to the results of the clock trees.

The results for the MM method in Table 2.2 are the best results of the min-matching algorithm in terms of skew variability reduction which we obtained from the algorithms of [54][1]. The RD rows gives the results for the "rule-delta" link insertion method for each test-case. The values of the parameters α, β , γ and δ are chosen empirically so as to obtain minimum skew variability. The values of α, β , γ and δ for the test-case $r1$ are 0.1, 21, 3 and 4 respectively. For all other test cases, namely $r2$ to $r5$, the values of α, β, γ and δ are 0.06, 100, 3, 5 respectively. The MST rows give the results of the "MST based link insertion" algorithm. We perform selective deletion of links from MST using the values of parameters $\alpha = 0.1$ and $\beta = 100$ for all the clock networks. The last but one column gives the Average Link Length (ALL) for all the three link insertion schemes.

The important observations from Table 2.2 are as follows:

- The new algorithms, RD and MST are able to achieve comparable or better skew variability reduction when compared to the min-matching algorithm of [54] with significantly lower wire-length. In all test cases, the average wire-length increase for the new algorithms was 5%, compared to the 15% cost increase of the min-matching algorithm of [54]. This can be observed from the 'WL' column.
- The decrease in the link-cost becomes more significant as the size of the clock network increases. For the test cases of r1 to r5, the wire-length costs

---

[1] In [1], the MM insertion algorithm was run on a given small number of links, but better MSV and SD may be obtained using the MM method if more links are allowed with more wire-length penalty.

Test-case	size	MSV	SD	WL	ALL	CPU(s)
MM-r1	22	0.14	0.15	1.15	9004	0.069
RD-r1	22	0.18	0.15	1.07	4688	0.007
MST-r1	24	0.13	0.14	1.07	4127	0.047
MM-r2	48	0.15	0.20	1.15	8296	0.095
RD-r2	40	0.12	0.11	1.07	4555	0.032
MST-r2	21	0.15	0.16	1.04	5701	0.075
MM-r3	64	0.16	0.19	1.14	7625	0.021
RD-r3	51	0.15	0.12	1.06	3987	0.067
MST-r3	41	0.18	0.13	1.05	4380	0.017
MM-r4	40	0.11	0.10	1.15	26289	0.860
RD-r4	71	0.13	0.10	1.04	4616	0.411
MST-r4	62	0.12	0.10	1.04	4956	0.79
MM-r5	72	0.09	0.08	1.15	22334	3.050
RD-r5	66	0.08	0.08	1.01	1862	1.945
MST-r5	94	0.09	0.08	1.01	1743	2.91
Avg.MM	49	0.13	0.14	1.15	14709	0.81
Avg.RD	50	0.13	0.11	1.05	3941	0.49
Avg.MST	48	0.13	0.12	1.04	4181	0.76

Table 2.2: Skew variations and wire-length in terms of tree results shown in Table 2.1. Size of a tree+link network is the # of links.

due to link insertion in our new algorithms (RD and MST) are reduced from around 7-8% to only 1.2% as the number of clock sinks increases from 267 to 3101. However, the previous best algorithm MM [1] has a 15% increase. This is mainly because of a drastic reduction in the average link length in the new algorithms. This in turn allows us to insert more links in the clock network, thereby reducing the skew variability more.

- The average length of link (ALL) for each of the three link insertion methods is given in the ALL column. In the case of the min-matching method, for bigger clock networks $(r4, r5)$, the average size of the links becomes significantly higher when compared to the smaller clock networks$(r1 - r3)$.

But in the case of the rule-delta algorithm and the MST based algorithm, there is no such significant increase in the average link size. Even in the case of smaller clock networks, the average link length for the proposed algorithms has significantly lower values when compared to the min-matching algorithm.

2.7.3 Run-time Comparison

One way of comparing the speed of the different algorithms is to find out how the run-times scale with the number of links inserted at the highest level (the links with $\gamma = 1$). This comparison is useful because, generally speaking, the more the number of highest level links, the better the skew variability reduction (because of increasing the occurrence of scenario 1). Figure 2.13 shows the run-time values of different algorithms as a function of number of links inserted at the highest level. From Figure 2.13 we can clearly see that the run-time of the min-matching algorithm increases drastically after a point when compared to the other two algorithms. This behavior can be explained as follows. In order to add k number of $\gamma = 1$ links, the number of nodes in the bi-partite graph required by the MST algorithm is $k+1$ while the min-matching requires $2k$ nodes. Thus, in terms of the number of links that needs to be added, the complexity of MST algorithm is $O(klog(k) + k^2m^2)$, while the complexity of min-matching is $O(k^3 + k^2m^2)$, where m is the maximum number of sinks within a sub-graph. Thus, we see that, for a given number of links to be added at a given level, the complexity of min-matching algorithm is considerably higher. Note that the non-trees of Table 2.2 are not restricted to a single γ level, unlike in this experiment.

Another way of comparing speed is to get the run-times of the different algorithms for nearly equal skew. This information can be obtained from the Table 2.2. From the Table 2.2, we see that the run-time for the MST algorithm is lesser than the min-matching algorithm for all the test cases. It can be noted here that since the algorithms might have to be run several times before a desired non-tree is obtained, the difference in the run-time per run might be important for large clock networks.

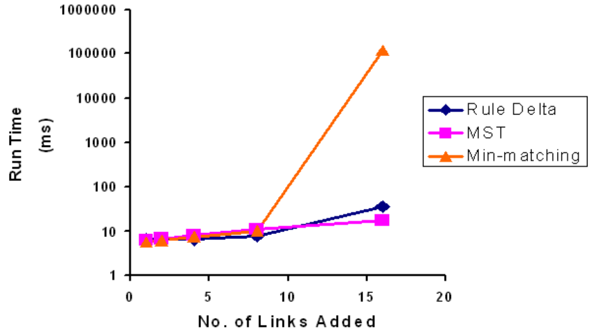


Figure 2.13: Run-time comparison between the different link insertion methods as a function of number of links inserted at $\gamma = 1$ level.

2.7.4 Automatic Parameter Determination Results

In order to verify the efficacy of our automatic parameter determination methods discussed in Sections 2.3 and 2.4, we use the same set of benchmark trees as used in Table 2.1. We compare the results of our automatic parameter determination with the best results obtained by us by manual tuning shown in Table 2.2. Table 2.3 shows the results of HSPICE Monte Carlo analysis for the non-trees obtained using our automatic parameter determination scripts for both rule-delta approach and MST based approach. The results are w.r.t. the trees shown in Table 2.1 and the skew reduction can be directly compared with the results of RD and MST given in Table 2.2. From the Table 2.3 we can see that the skew reductions obtained by our automatic parameter determination method are very close to that of the best results that we were able to get by manual parameter selection. The '# Trials' column in Table 2.3 shows the number of Elmore Monte Carlo trials required by our script to select the final non-tree.

From the Table 2.3, it is clear that determining a good set of parameters for the MST based approach is much quicker than for rule-delta approach. It may be noted here that, though the automatic parameter determination is much slower when compared to link addition algorithms, it is usually much faster than manual tuning. Also, for a given clock tree, link addition is only intended as a post-processing step aimed at reducing skew variability. As a result, the automatic parameter selection needs to be run only once.

Test-case	MSV	SD	WL	#Trials	CPU(s)
Auto-RD-r1	0.12	0.11	1.11	120	180
Auto-MST-r1	0.12	0.11	1.08	16	24
Auto-RD-r2	0.14	0.12	1.04	180	1260
Auto-MST-r2	0.14	0.12	1.06	16	112
Auto-RD-r3	0.12	0.13	1.09	150	3750
Auto-MST-r3	0.13	0.13	1.05	18	450
Auto-RD-r4	0.11	0.12	1.05	170	10540
Auto-MST-r4	0.10	0.12	1.02	24	1488
Auto-RD-r5	0.09	0.07	1.03	360	86400
Auto-MST-r5	0.07	0.07	1.03	36	8640
Auto-RD-Avg	0.116	0.11	1.075	196	20426
Auto-MST-Avg	0.112	0.11	1.073	22	2142

Table 2.3: Results for automatic parameter determination procedures. The HSPICE Skew variability and wire-length values are w.r.t. the tree values of Table 2.1.

2.7.5 Incremental Link Insertion Results

Table 2.4 shows the skew variability reduction of our incremental link insertion w.r.t. the variation results of the trees in Table 2.1. The important observations from Table 2.4 are as follows:

- The incremental link insertion achieves comparable reduction in skew variability to that of the existing algorithms with comparable increase in wire-length consumption.
- The incremental link insertion is very fast even for the biggest benchmark with 3100 sinks. Thus, our method is more efficient that the statistical link insertion of [35]. It is also much faster when compared to the automated version of the algorithms of [60] discussed in Sections 2.3 and 2.4.
- Since the incremental link insertion is a one-shot approach, a good non-tree is obtained in a single trial, thus avoiding the time consuming manual parameter selection and statistical methods of [35]. The amount of time saved by incremental link insertion can be seen by comparing the run-times of the Auto-MST and Auto-RD methods of Table 2.3 and the run-time of incremental link insertion of Table 2.4. For the biggest test case r5, the incremental link insertion selects a good non-tree in just 70 seconds when compared to the 8640 seconds taken by the Auto-MST method with a very similar variation reduction and wire-length increase.

Test-case	MSV	SD	WL	CPU(s)
Incremental-r1	0.18	0.17	1.10	0.5
Incremental-r2	0.18	0.2	1.06	3.0
Incremental-r3	0.12	0.13	1.05	6.0
Incremental-r4	0.13	0.14	1.02	12.0
Incremental-r5	0.09	0.07	1.02	70.0
Incremental-Avg	0.14	0.142	1.05	18.3

Table 2.4: Incremental link insertion results w.r.t. the tree values of Table 2.1.

2.7.6 Skew reduction results for non-zero skew clock networks

To verify our procedure of adding links to non-zero-skew clock networks discussed in Section 2.6, we modified our MST algorithm code so as to insert links in a nonzero skew clock network using the method described in Section 2.6. We used the BST [17] code from GSRC Bookshelf [26] with non-zero maximum skew of 200 ps to obtain the trees nzs-r1 to nzs-r5. These trees have a maximum skew of 200 ps between the different sinks and have non-zero skew for most sink pairs. As a result, we can treat them as useful skew trees for the purpose of our experiments. The sizes of these clock networks in terms of number of sinks is identical to that of the r1-r5 benchmarks that we have used for zero-skew link addition experiments. We run our MST algorithm on each of the trees and compare the skew variability and wire-length between the original trees and the corresponding non-trees. The results are summarized in Table 2.5. Please note that *the delay values given for the trees are absolute values and the values given for the non-trees are w.r.t. the corresponding tree.* We measure the skews for non-zero skew clock networks as the amount of deviation from the *nominal skew.* For example, for the test case nzs-r1, the value of 245 ps of MSV is the *unwanted and extra* skew due to variations, in addition to the nominal skew.

From Table 2.5, we can see that link insertion reduces the skew variation to a good extent in the non-zero-skew clock networks (roughly, by an average of more than 80%). However, the results are slightly less impressive when compared to the skew reduction for zero-skew clock networks. This can probably be explained by the fact that, in zero skew clock networks, since the links are directly added at the sinks, the skew due to load variation is controlled to an extent. However, for the non-zero-skew case, the loads of one of the sinks is not connected to the link directly. As a result, the variation of this load is not reduced even after link addition.

Test-case	NS	MSV	SD	WL	# Links
nzs-r1	200	245	70	307513	-
r1-NT	1.00	0.36	0.25	1.08	13
nzs-r2	200	693	195	638248	-
r2-NT	1.00	0.34	0.25	1.09	21
nzs-r3	200	958	241	833937	-
r3-NT	1.00	0.15	0.14	1.09	33
nzs-r4	200	3390	746	1904623	-
r4-NT	1.00	0.21	0.17	1.09	64
nzs-r5	200	5837	1503	2932095	-
r5-NT	1.00	0.11	0.09	1.09	68

Table 2.5: Skew reduction for non-zero skew clock networks. The nominal skew (NS), maximum skew variation (MSV), standard deviation (SD) values for non-zero skew CDNs. The delay values given for the trees are absolute values (ps) and the values given for the non-trees are w.r.t. the corresponding tree.

2.8 A note on selecting the appropriate link insertion algorithm

Since we have presented three different link insertion techniques in this chapter, a natural question that arises is how to choose between these methods. In this section, we give a few pointers on selecting the appropriate method. One conclusion that we can come to based on the experimental results is that, for large clock trees, incremental link insertion achieves good skew variability reduction while requiring very less run time compared to the other two methods. Also, the incremental approach does not require any parameter selection, making this a good choice for fully automated link insertion. Thus, the incremental link insertion method can be used when good results are required in a very short amount of time without much effort from the user. However, the best results in terms of skew standard deviation reduction are obtained by the RD and MST algorithms. But the runtime associated with choosing good parameters are much higher. Thus, these methods can be employed when skew variation reduction of 1-2% is critical, as is the case in very high speed designs and when higher computational cost is acceptable so as to gain a small increase in skew variability reduction.

Chapter 3

Link Insertion for Buffered Clock Trees

The main focus of this chapter is synthesis of buffered clock networks with cross-links. First, we will discuss why the link-insertion methods of the last chapter cannot be directly used for buffered clock network synthesis problem. This is followed by description of the method we propose to address this problem. We then propose a new sensitivity based link insertion scheme that is more suitable for use in buffered clock trees compared to methods discussed in last chapter. We conclude the chapter with experimental results that demonstrate the effectiveness of our methods.

3.1 Why Link Insertion for Buffered Clock Trees?

Link-based clock network [54, 60, 78] has been proposed as a possible method to reduce skew variability. However, [54, 60] address only unbuffered clock networks, making them impractical for even medium sized designs. The work of [78] attempts to solve the problem of constructing a link based buffered clock network in which special tunable buffers are used to obtain a low-skew (in SPICE) buffered clock network. Once a good low-skew buffered clock network is obtained, [78] uses the algorithms of [60] to select the links to be inserted. Finally, SPICE simulations are used to tune the buffers and internal node locations. However, the use of tunable buffers might result in increased area and power consumption. Moreover, the tunable buffers might not be available in all design libraries. Also, because of the use of SPICE for tuning, [78] is slow for even clock network of a few hundred sinks. In this work, we propose an efficient algorithm for synthesizing a link-based clock network in which we have attempted to overcome the above mentioned drawbacks of [78]. The important contributions of this work are:

- Our methodology uses ordinary buffer cells and does not require any special tunable buffer cells unlike [78].
- We adapt and modify the efficient & accurate delay evaluation method of [53] to consider the slew and resistive shielding effects during clock network synthesis, thereby avoiding the use of SPICE for constructing the clock network. This also helps us to overcome the inaccuracy of the simple Elmore delay model, which is used by most clock tree synthesis algorithms including the recent ones like [11, 73, 85]. Thus, our algorithm is both fast and accurate.
- We propose a new merging scheme for clock tree synthesis to obtain a *link insertion friendly* balanced clock tree.

Monte Carlo simulations in SPICE show that our algorithm can reduce skew variability by 50% on an average with no penalty in resources or runtime.

3.1.1 Challenges in Buffered Clock Tree Link Insertion

The algorithms used in [54, 60] is applicable only to unbuffered clock networks and cannot be applied to buffered case because of the reasons discussed

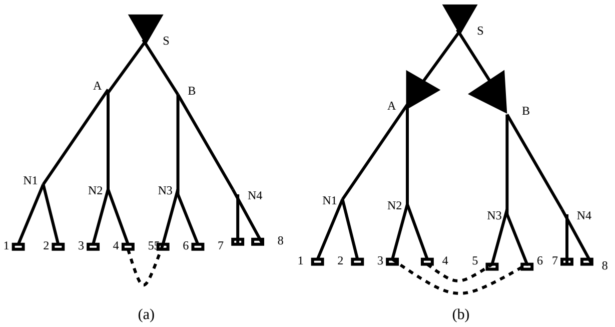


Figure 3.1: An example of link-based non-tree. (a) Unbuffered case. (b) Buffered case.

next.

3.1.1.1 Chicken-egg problem

Inserting a link in a buffered clock tree introduces a chicken-egg problem between the location of buffer driving the linked nodes, the input slew of the buffers and the downstream delays. For example, in Figure 3.1 (b), the link between nodes 4 and 5 increases the loads driven by the buffers A and B, which affects the locations and input slews to the buffers, which in-turn affects the delays from buffer A(B) to sink node 4(5). But, the skew between sink nodes 4 and 5 needs to be evaluated in order to select them for link insertion, thus completing the cyclic dependency. This fact has also been noted in [78].

3.1.1.2 Accuracy of delay model

Accurate delay models must be considered while inserting links in a buffered clock tree because of the following reasons:

- As shown in [54], a link will be beneficial only when it is inserted between two sink nodes with zero or near-zero skew . While Elmore delay has been shown to have good fidelity for the unbuffered clock trees w.r.t SPICE in [54, 60], the fidelity is poor for a buffered clock tree as demonstrated in [82].

- Adding links between two sinks driven by different buffers introduces the problem of multi-driver nets. For example, in Figure 3.1 (b), the link between nodes 4 and 5 has two drivers, namely buffers A and B. If the links are not selected considering accurate delays, then it is possible to insert links between nodes whose delay values are quite different. This might increase the nominal short circuit power of the clock network because of the virtual shorting of Vdd and Vss (Source and Ground) that might occur when one of the drivers gets turned much ahead of others or vice versa.

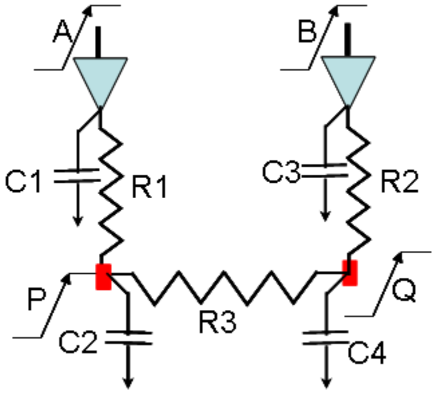


Figure 3.2: A simple example of link insertion in buffered clock trees.

3.1.1.3 Short-circuit Power and Waveform Quality

A key reason why link insertion in buffered clock trees is much more challenging than in unbuffered clock trees is the potential for significant increase in short-circuit power and for degradation of the waveform quality at the clock sinks. This problem can be easily understood using the following example. Consider the simplest case of one link between two buffers that were originally driving one sink each. After a link is added, the equivalent circuit can be represented as shown in Figure 3.2. Depending on the skew between the input signals A and B to the two buffers, the quality and power of the system can change drastically. To test this assertion, we picked sample values of $R1 = R2 = 100\ Ohms$ to represent the resistance between buffers and the sinks, $R3 = 2\ Ohms$ to represent the link resistance, $C1 = C3 = 5fF$ and $C2 = C4 = 6fF$ to represent the interconnect and sink capacitances. We then varied the skew between the two input signals A and B between $0ps$ to $200ps$ and analyzed the resulting waveforms and power using $HSPICE$. These results are presented in Figures 3.3, 3.4, 3.5, 3.6, 3.7 and in Table 3.1.

A to B Skew (ps)	Avg. Power (uW)	Sink Skew (ps)
0	71.085	0.019
10	75.334	0.014
20	75.563	0.008
30	82.373	0.004
40	92.533	0.019
50	102.42	0.03
75	126.49	0.121
100	155.86	0.31
125	187.59	1.12
150	219.84	0.83
175	250.31	0.49
200	283.26	0.71

Table 3.1: Average power and sink skew for different values of skews between signals A and B in Figure 3.2

From the Figures 3.3, 3.4, 3.5, 3.6, 3.7, we can see that the quality of the voltage waveform is degrading progressively. This is because of the fact that

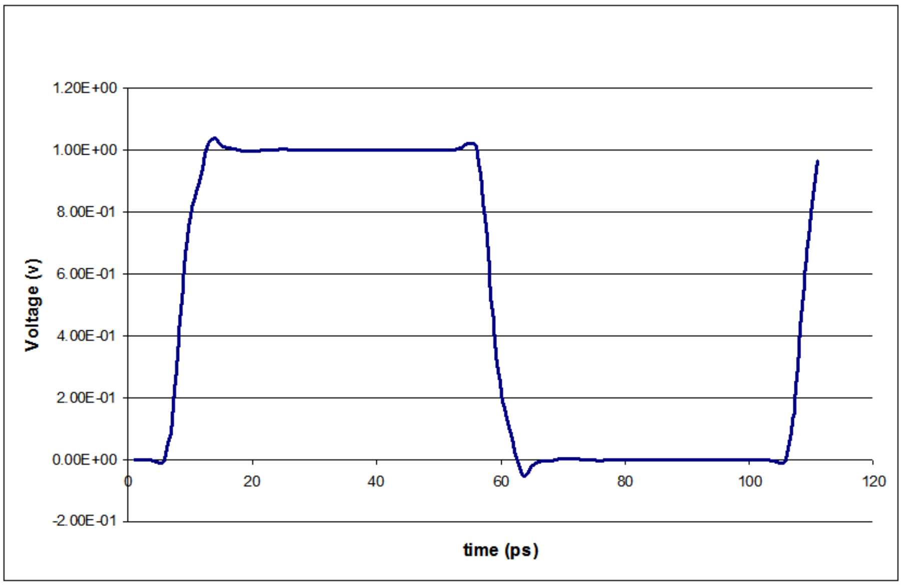


Figure 3.3: Sink voltage waveform with 0ps skew between input signals A and B in Figure 3.2.

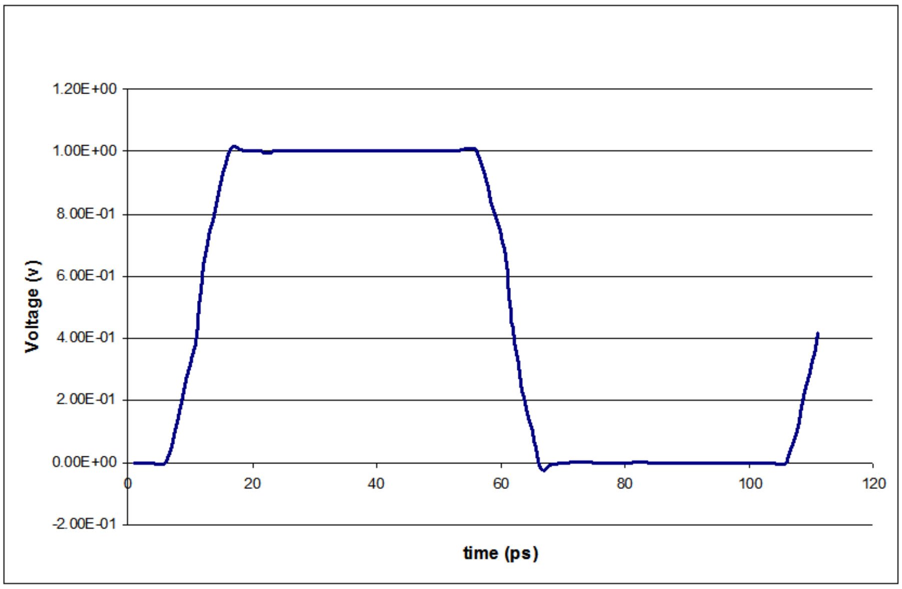


Figure 3.4: Sink voltage waveform with 50ps skew between input signals A and B in Figure 3.2.

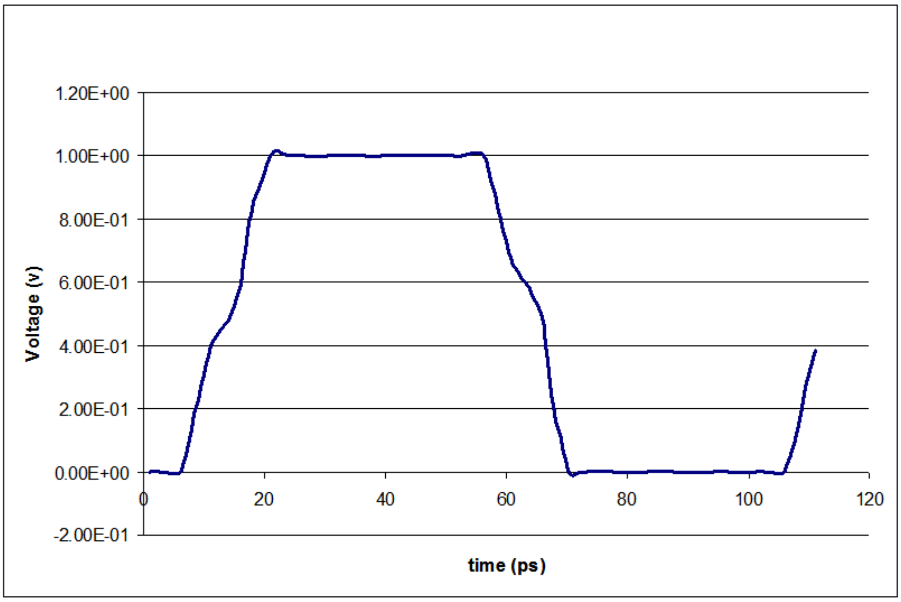


Figure 3.5: Sink voltage waveform with 100ps skew between input signals A and B in Figure 3.2.

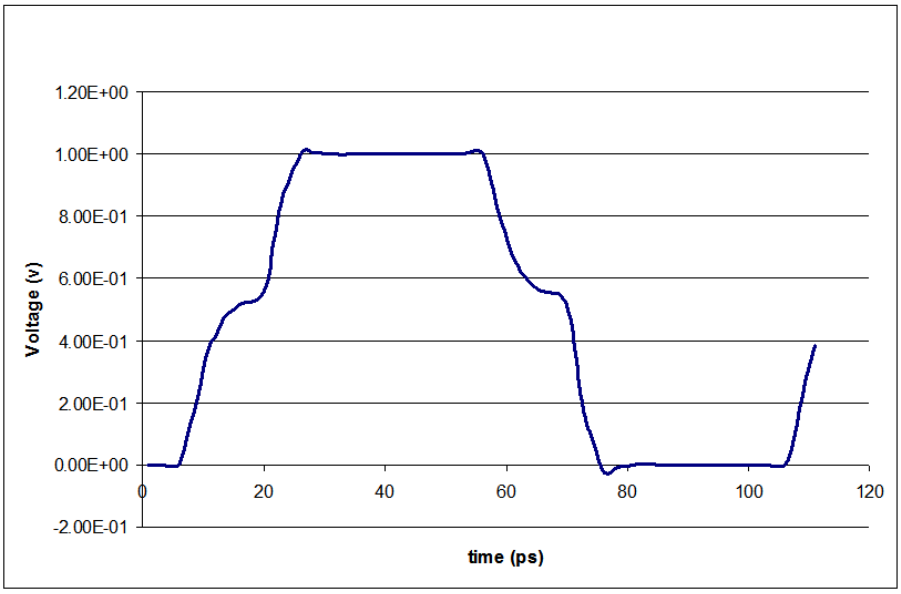


Figure 3.6: Sink voltage waveform with 150ps skew between input signals A and B in Figure 3.2.

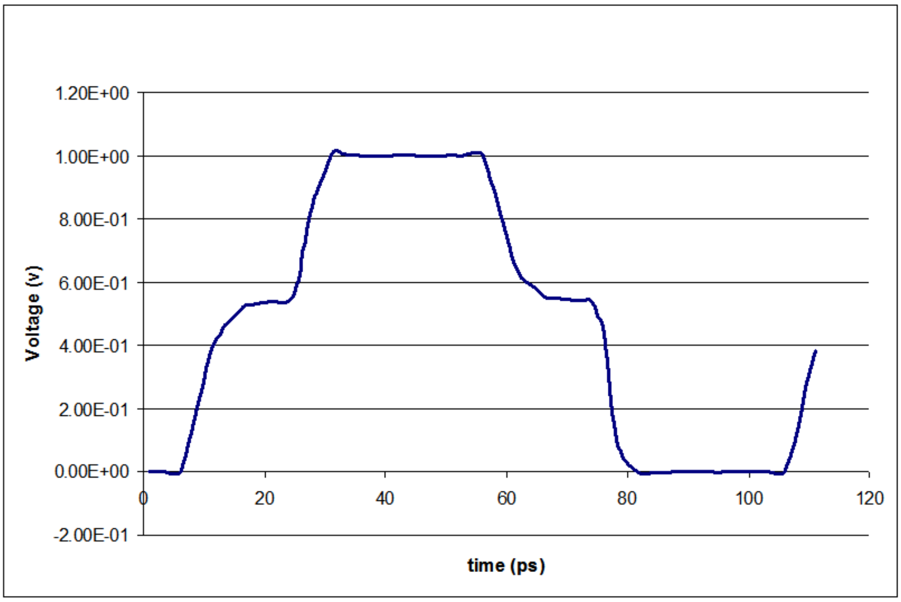


Figure 3.7: Sink voltage waveform with 200ps skew between input signals A and B in Figure 3.2.

when skew between the input signals increase, the two buffers are out of sync and so will not be on or off at the same time. This results in overloading of the buffer that has switched first and slowing the transition. This also results in significant increase in short-circuit power consumed over the same period of time as the two buffers will be in opposite states for longer period of time compared to the zero skew case. This can be verified from the average power numbers from Table 3.1. It may be noted here that in spite of the increase in power, the skew between the two sinks remains small due to the small value of link resistance. From this, we can conclude that a bad link addition between buffers that can potentially have high skews between them will result in both significant power increase and also waveform degradation that might result in unreliable operation of flops/registers.

Thus, it is clear from the above points that link insertion for buffered clock tree is a non-trivial problem. In order to insert links in a buffered clock tree, we required the buffered clock tree to have a good nominal skew in an accurate delay model. By making sure that the nodes have very similar accurate delay values, we not only make the skew variability better, but also reduce the nominal short-circuit power consumption.

3.1.2 Existing clock tree synthesis algorithms

One of the pioneering algorithms for clock routing was proposed in [75], in which a zero skew clock routing was obtained by recursively merging a pair of zero skew subtrees until a single clock tree is obtained. The zero-skew principle in [75] was extended in the DME algorithm [10] for wire length minimization. However, [10, 75] addressed only the problem of an unbuffered clock tree. The problem of constructing a zero skew buffered clock tree under Elmore delay model was solved in [14, 52]. The optimal clock buffering for a given topology and buffer library was solved in [16]. A heuristic for synthesizing a low-power buffered clock tree using the Elmore delay model was proposed in [80]. Buffer and wire sizing were done so as to reduce power and maintain the zero skew property under Elmore delay. None of the above clock tree synthesis algorithms consider clock signal slew *during* the synthesis of the clock tree.

To our best knowledge, [70] was the first work that explicitly considered slew during buffer insertion. But it assumes a *given* unbuffered clock tree, which may result in very sub-optimal solution compared to simultaneous buffering and clock routing as shown in [14]. In [82], a SPICE based, time domain based buffer/wire sizing has been proposed which results in greatly reduced skew in SPICE. To the best of our knowledge, [82] is the only work that aims to reduce the actual clock skew in SPICE. However, since it directly uses SPICE for tuning the clock network, the runtime may be prohibitively high. In [45], the important problem of clock buffer load imbalance is addressed. In the previous algorithms like [14], different clock buffers at a given level from the clock source may drive different loads. The methodology in [45] attempts to solve this problem by using a clustering approach. But such a clustering and load balancing approach usually results in excessive wire length due to wire snaking when the two clusters to be merged do not have similar target delays.

In the recent works of [73, 85], the problem for optimal buffer/wire sizing in clock network has been studied under the Elmore delay model. In [11], a new merging scheme has been proposed for prescribed skews which usually results in considerably less wire-length compared to the other algorithms. However, this algorithm results in highly unbalanced clock structure. The balanced clock structure issue has been addressed in [14, 45] at the cost of excessive total wire lengths compared to [11].

3.1.3 Requirements of a Link Insertion Friendly Buffered Clock Tree Synthesis

Based on our discussions in section 3.1.1, the requirements of a *link insertion friendly* clock tree synthesis algorithm are:

Accurate and fast delay model during synthesis: This requirement implies that the effect of slew and resistive shielding are to be considered, which will ensure that the sink node pairs are selected based on accurate skew values. It also implies that SPICE should not be used for the clock tree synthesis as it may increase the runtime considerably. However, as discussed in section 3.1.2, most of the existing buffered CTS algorithms, including the recent ones like [11, 73, 85], use Elmore delay or use SPICE for synthesis like [82].

Balanced clock tree without excessive wire snaking: We define a balanced clock tree as one in which identical buffers are inserted at a given level from the clock source. Also, a balanced buffered clock tree will have the same number of buffer levels from the source to each clock sink. Figure 3.8 (a) and (b) shows an example of unbalanced and balanced buffered clock tree respectively.

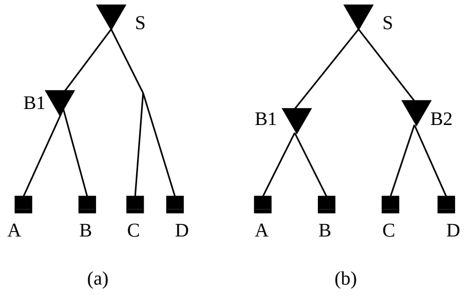


Figure 3.8: (a) An example of an unbalanced buffered clock tree; (b) An example of a balanced buffered clock tree.

A key advantage in having a balanced clock structure is that it will be much more tolerant to variation. For example, if the nominal delays from source to sinks in both Figure 3.8 (a) and (b) is 100 ps and if the nominal gate delay is 20ps. Under nominal conditions, both Figure 3.8 (a) and (b) will have identical skews. However, when variation effects become more and more severe, the scaling of the device delays and interconnect delays need not match. For example, due to certain change in operating temperature or voltage, the device delay doubles and interconnect delay becomes one half of the nominal values. Then the balanced structure is much more tolerant to the variation. In the UDSM technologies, it is becoming increasingly difficult to capture all the different variation effects. This further motivates us to use the balanced clock structure to improve tolerance to variations. Many recent clock tree synthesis algorithms like [73, 82, 85] use buffers of different sizes and tune them in such a way that the skew and delay targets are met at the nominal values of device and interconnect parameters. However, due to the PVT variations, significant skew can be generated in such clock trees.

Also, the total wire-length of the clock network should be as less as possible in spite of maintaining a balanced structure. The reason for this requirement is that most of the existing algorithms that obtain a balanced clock structure like [14, 45] achieve load balancing by clustering methods, which often result in excessive wire snaking. This is undesirable because excessive wire-length increases the total power and resource consumption. It may be emphasized here that, even thought the idea of a balanced clock tree is well known, to the best of our knowledge, there is no work that guarantees the balanced nature of the resulting clock tree without performing clustering.

A fact to be noted here is that none of the existing clock tree synthesis algorithms satisfy all the above requirements. Though each of the above requirements have been addressed in bits and pieces, to the best of our knowledge, there is no unified clock tree synthesis algorithm that addresses all the above issues in a systematic way. In this work, we propose such a unified clock tree synthesis methodology, which will result in a *link insertion friendly* buffered clock tree.

3.2 Iterative Delay Evaluation and Backward Slew Propagation

The work of [53] introduces a fast, accurate and iterative delay evaluation procedure which has the elegance and simplicity of Elmore delay with much improved accuracy. The method of [53] is mainly for delay analysis and cannot be directly applied for clock tree synthesis. This is because [53] uses a technique called *slew propagation* in which the slew is propagated from the signal source to the sinks. But, during bottom-up clock tree synthesis, the slew at the source in unknown and hence the method of [53] cannot be used. To overcome this, we solve the inverse of the *slew propagation* called *backward slew propagation* in which we propagate the slew targets in a bottom-up fashion, which can be applied during clock tree synthesis. In this section, we briefly review the iterative delay evaluation of [53] followed by the explanation of our backward slew propagation method.

3.2.1 Iterative delay and slew evaluation

Ideally, we would like to have a delay evaluation procedure that is as efficient and elegant as Elmore delay while accounting for resistive shielding and signal slew effects. The iterative delay estimation procedure of [53] is such a delay model, used in IBM's physical design closure tool. The procedure explicitly considers the signal slew in delay evaluation and accounts for the interdependence between the input signal slew of a node and the *effective* load seen by the node. However, the procedure is mainly for delay evaluation. In this paper, we extend it for the purpose of clock tree synthesis by introducing the notion of *required slew* similar to the concept of *required skew.*

Consider the Figure 3.9 of a simple RC network connecting nodes v and a. An input ramp voltage with a signal transition time of t_v is applied at the node v. The transition time at the output node of the RC segment, namely node a is given by t_a. According to Elmore delay, the total down stream capacitance *seen* by the node v is C. However, because of the resistive shielding effect of the resistance R, only a fraction of the capacitance C is actually seen by the node v, which is usually referred to by the name *effective capacitance* [53]. According to [53], the value of this effective capacitance is give as:

$$C_{eff} = K * C \tag{3.1}$$

where K is the scaling factor defined as:

$$K = 1 - 2x(1 - e^{-\frac{1}{2x}}), \quad where \quad x = \frac{RC}{t_v} \tag{3.2}$$

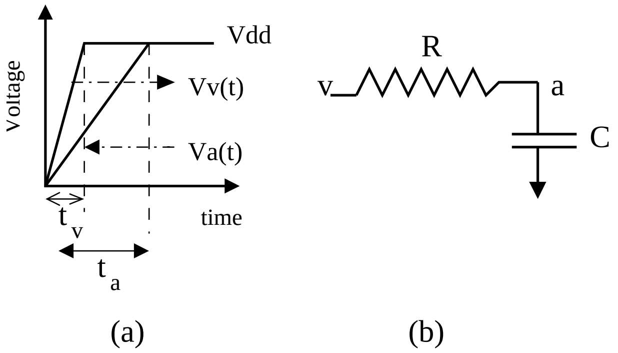


Figure 3.9: (a) Definitions of transition times for nodes v and a. (b) A simple example of RC network.

It should noted that the value of the effective capacitance seen by node v and the slew rate at v are interdependent. The output slew rate of the CMOS buffer depends on both the input slew *and* the load capacitance [53]. From Equations (3.1) and (3.2), the effective load capacitance seen by the buffer output depends on the slew at the buffer output. This factor introduces a chicken-egg problem which is addressed in [53] using an iterative delay evaluation technique.

3.2.2 Backward propagation of slew

In order to consider the node slews during the clock tree synthesis, we need to calculate the signal slew rate during the bottom-up topology generation phase of the DME [10, 11] algorithm. However, by definition, the slew rate at a child node can be calculated only when the slew rate at the parent node is known. For example, in Figure 3.9, the slew rate at node a can be obtained only when the slew rate at node v is known. An efficient method for obtaining the transition times at the nodes of the clock tree for a given transition time at the source node has been proposed in [53]. Considering the Figure 3.9, the transition time at node v is given as t_v. Given t_v and the R, C values, the transition time time at node a can be obtained using the method of [53] as:

$$t_a = \frac{t_v}{1 - x(1 - e^{-\frac{1}{x}})}, \quad where \quad x = \frac{RC}{t_v} \tag{3.3}$$

In order to consider the slew *during* clock tree synthesis, we would like to get an inverse of Equation (3.3). That is, we would like to get the value of t_v for a given value of t_a. Such an inverse expression will enable us to consider slew during the bottom up phase of clock tree synthesis. Such an inverse expression can be obtained as follows: define a new variable called y and using 3.3, we have:

$$y = \frac{RC}{t_a} = \frac{RC(1 - x(1 - e^{-\frac{1}{x}}))}{t_v}$$

which can be simplified to

$$y = x(1 - x(1 - e^{-\frac{1}{x}})) \tag{3.4}$$

The plot of Equation (3.4) is shown in Figure 3.10. As it can be seen from the plot, the value of y reaches a saturation point after the value of x reaches a value of roughly 20. The saturation value of y is 0.5, which can also be verified by applying the Taylor series approximation for the term $e^{-\frac{1}{x}}$ as $1 - \frac{1}{x} + \frac{1}{2x^2}$. Using this approximation in Equation (3.4) will reduce the value of y to 0.5. A key use of the above observation is that for a given value of x, there is an unique value of y and vice versa. Thus, when we are given the *required* slew value at output node, we can obtain the value of y, which can be used to uniquely determine the value of x, which in turn can be used to obtain the *required* input slew. In other words, if we have a slew requirement at the child node a in Figure 3.9, using that we can uniquely obtain the *required* slew value at the parent node v. This technique can be used to build a buffered clock tree with simultaneous slew considerations in a bottom-up fashion.

3.3 Link Insertion Friendly Clock Tree Synthesis

In this section, we introduce our link insertion friendly clock tree synthesis algorithm in which we have attempted to simultaneously consider all the requirements outlined in section 3.1.3. To the best of our knowledge, this is the first work in which all these factors are considered in a unified and systematic way. First, we

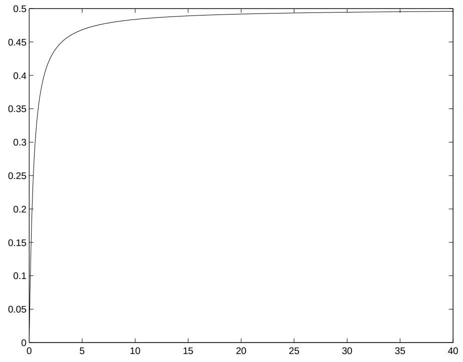


Figure 3.10: Plot of x (ratio of RC and input slew) versus y (ratio of RC and output slew) of Equation (3.4).

will consider the problem of merging two subtrees using the backward slew propagation algorithm of the previous section. Then we introduce our novel merging scheme which guarantees the construction of a perfectly balanced buffered clock tree while simultaneously reducing the wire-length and maintaining load balance.

The high level framework of our algorithm is similar to the DME based algorithms like [11, 14] in which the first step is the topology generation phase in which different subtrees are merged recursively based on a merging cost. After all the subtrees are merged into a single tree, a top down embedding is done to finalize the locations of the clock tree nodes.

3.3.1 Subtree merging with backward slew propagation

Consider Figure 3.11 in which two subtrees T_i and T_j (rooted at nodes i and j respectively) are to be merged to form a new subtree T_p with node p as the root. In the traditional merging, the lengths of segments $l_{p,i}$ and $l_{p,j}$ are determined in such a way that the Elmore delay from v to the sinks of both T_i and

T_j are identical. During this step, the entire downstream capacitance at nodes i and j are considered. However, the delay evaluation method of [53] considers only the *effective* capacitance at the subtrees T_i and T_j while determining the edge lengths. The delay from node p to nodes i and j are given as [53]:

$$D(p,i) = \frac{1}{2} rcl_{p,i}^2 + rl_{p,i}C_{eff1} \tag{3.5}$$

$$D(p,j) = \frac{1}{2} rcl_{p,j}^2 + rl_{p,j}C_{eff2}$$

where, r and c are the unit length resistance and capacitance, respectively. C_{eff1} and C_{eff2} are the effective downstream capacitance of nodes i and j respectively. It may be noted that for clock sinks, the value of effective capacitance is equal to the load capacitance.

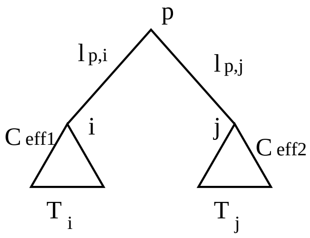


Figure 3.11: An example of subtree merger using effective downstream capacitance

In order to balance the *effective delays* of the two subtrees, the following equation must be satisfied:

$$D_i + D(p,i) = D_j + D(p,j) \tag{3.6}$$

where D_i and D_j are the delays from nodes i and j to their respective sink nodes. The edge lengths $l_{p,i}$ and $l_{p,j}$ can be obtained by solving Equation (3.6) with the condition that $l_{p,i} + l_{p,j} = L$, where L is the Manhattan distance between the nodes (or the merging segment of the nodes) i and j. Wire snaking can be used to match the delays if wire-lengths greater than L is required [11]. Once the appropriate segment lengths have determined, the *required slew* at the parent node p can be calculated using Equations (3.3) to (3.4).

Figure 3.12 explains this step in detail. A point to be noted regarding bottom-up transition time limit propagation is that, during merging of two subtrees with different transition time limits, two independent transition limits (one for each child node) can be obtained for the new root p, denoted by t_p^i and t_p^j in the Figure 3.12. Since the transition time limits are defined as the maximum signal rise time acceptable at a particular node, we pick only the tighter requirement of the two. Also, selecting the lesser transition time might impact the zero skew property within the subtree that has bigger transition time. However the effect of this is minimal based on our experimental experience.

Once the *required slew* information at the root node p is available, the effective downstream capacitance at node p can also be calculated as demonstrated in Figure 3.13.

Thus, using the algorithms of Figures 3.12 and (3.13), we can merge a give pair of subtrees and obtain the values of slew and effective downstream capacitance of the new subtree. In order for this method to be applied in a recursive fashion, the slew requirements at the clock sink nodes must be predefined by the user. This can be used during the bottom-up clock tree construction as shown in

```
Procedure: $FindSlew(T_p)$
Input: A subtree rooted at node $p$
Output: The signal transition time limit at $p$.
1. If $p$ is a sink
     $t_p$ = Transition time limit set by user.
     return.
2. $i = LeftChild(p); j = RightChild(p)$.
3. $t_i, t_j \leftarrow$ Transition time limit at nodes $i$ and $j$.
4. $R1 = rl_{p,i}; R2 = rl_{p,j}$.
5. $C1 = C_{eff1} + 0.5cl_{p,i}; C2 = C_{eff2} + 0.5cl_{p,j}$.
6. $y1 = \frac{R1C1}{t_i}; y2 = \frac{R2C2}{t_j}$ (Similar to eqn.2)
7. For $y1$ and $y2$, obtain the corresponding unique
     values of $x1$ and $x2$ using eqn.(3.4).
8. Using $x1$ and $x2$, obtain transition time limits
     at node $p$ w.r.t nodes $i$ and $j$ as:
     $t_p^i = \frac{R1C1}{x1}; t_p^j = \frac{R2C2}{x2}$
9. return $min(t_p^i, t_p^j)$
```

Figure 3.12: Procedure to evaluate the signal transition values of a node given the transition values of the child nodes.

the next section.

3.3.2 Balanced CTS algorithm

As discussed in section 3.1.3, one of the key disadvantages of several existing algorithms is the difficulty in getting a balanced clock tree without a wirelength penalty. We propose to address this key problem using a novel merging scheme, which is explained below.

In any merging scheme, node pairs to be merged are selected as per a cost function. In most of the traditional merging schemes like [10], node pairs that are physically closest are merged together with the intention of reducing the total wire length. But, as noted in [11], this might result in excessive wire snaking when the nodes to be merged do not have similar delays. The algorithm in [14]

```
Procedure: $FindEffectiveCapacitance(T_p)$
Input: A subtree rooted at node $p$
Output: The effective downstream capacitance at node $p$.

1. If $p$ is a sink
   $C_{eff}$ = Sink load capacitance
   return.
2. $i = LeftChild(p); j = RightChild(p)$
3. $C_{eff1}, C_{eff2} \leftarrow$ Effective downstream capacitance of $i$, $j$
4. $R1 = rl_{p,i}; R2 = rl_{p,j}$.
5. $C1 = C_{eff1} + 0.5cl_{p,i}; C2 = C_{eff2} + 0.5cl_{p,j}$.
6. $t_p$ = transition time limit of node $p$
7. $K1 = \frac{R1C1}{t_p}; K2 = \frac{R2C2}{t_p};$
8. $C_{eff} = K1C1 + K2C2 + 0.5c(l_{p,i} + l_{p,j})$; return.
```

Figure 3.13: The procedure to evaluate the effective downstream capacitance recursively.

selects the node pairs that result in the smallest delay after the merger. This generally results in a more balanced tree. However, the wire-length consumed is generally more. In [11], the pair that results in the minimal merging wire-length are merged. Since this is in some ways similar to the minimum spanning tree algorithm (which at each step selects the new edge with minimal cost), it results in much a lower wire-length when compared to the approaches of [10] and [14]. However, as noted in [11], it might result in an highly unbalanced clock tree. In our work, we modify the cost function of [11] such that a balanced structure is obtained while wire-length is also reduced.

Top level algorithm: The top-level steps involved in our buffer insertion flow are given below:

1. Initialize a list F as an empty list. This list will contain all the *flagged, unmerged* nodes. A *flagged* node is one that cannot be merged with any of the

other unmerged nodes without violating the limit on effective downstream capacitance (which is the maximum driving capability of the buffer used).

2. Initialize a list U with the set of all the sink nodes. This list will store all the *unmerged, unflagged* nodes.
3. While $(Sizeof(U) + Sizeof(F) > 1)$ Do
    (a) (T_i, T_j) = GetSubTreesToBeMerged(U) using steps in Figure 3.14.
    (b) If $(T_i, T_j) \neq$ NULL
        i. Merge the subtrees to get a new subtree T_k. Obtain the values of *required slew* and C_{effk} for node k using Figures 3.12 and 3.13.
        ii. Remove T_i, T_j from U.
        iii. Add T_k to list U.
    (c) else if ((T_i, T_j) = NULL) AND $(Sizeof(U) + Sizeof(F) > 1)$
        i. Insert buffers at all the nodes of F.
        ii. Update the values of delay, slew and effective downstream capacitance for all nodes $\in F$ using the delay characteristics of the buffer.
        iii. Move all the nodes in list F to list U and empty list F.
4. Perform top down embedding.

The key step in the above procedure is the step 3(a) which selects the node pairs to be merged. This step is explained in Figure 3.14. For node-pair selection,

```
**Procedure:** $GetSubTreesToBeMerged(U)$
**Input:** Set of all unmerged subtrees
**Output:** The two subtrees to be merged
1. $PairsFound = 0$
2. While (PairsFound $\neq 1$) AND ($Sizeof(U) > 1$) Do
   (a) $T_i$ = subtree with min root-sink delay in $U$
   (b) $MergingCost = \infty$
   (c) For each subtree $T_k \in U$ and $T_k \neq T_i$
       i. $cost = MergingCost(T_i, T_k)$ defined in Fig. 3.15
       ii. if $cost < MergingCost$
               $MergingCost = cost; T_j = T_k.$
   (d) if $MergingCost \neq \infty$
           PairsFound = 1
       else
           Remove $T_i$ from $U$; Add $T_i$ to $F$.
3. if $MergingCost \neq \infty$
       return ($T_i$, $T_j$)
   else
       Transfer the possible single node $\in U$ to list $F$.
       return $NULL$.
```

Figure 3.14: The algorithm for selecting the subtrees to be merged.

we use similar cost function as in [11] with an important change. In [11], a buffer will be inserted in a node *as and when* the node downstream capacitance exceeds a certain limit. But such an approach will result in an highly unbalanced clock tree.

In our algorithm, we insert buffers only when there is no node pair that can be merged without violating the *effective downstream capacitance limit.* To enforce this requirement, we maintain two separate lists - one called F which will have a list of $flagged$ nodes and another list called U in which we will store the list of $unflagged$ nodes. For node pair selection, we consider only the list U. If, for a particular node $i \in U$, we are not able to identify a suitable node pair for merger without exceeding the capacitance limit, we add that node to the list of *flagged* nodes F and remove i from U. We repeat the node-pair selection process until the list U becomes empty or contains a single element that cannot be merged with any other node. At that stage, we add buffers to *all the unmerged nodes* of F, update their delays, slews and effective downstream capacitance and transfer all the nodes to the list U. This cycle continues till there is only a single clock tree.

A point that may be noted here is that the $MergingCost$ algorithm of Figure 3.15 returns a value of ∞ when a possible merger of two node pairs i and j causes the effective capacitance limit to be violated. Thus, only node pairs that result in a node with lesser effective capacitance than the preset limit are merged.

Merits of our algorithm: An obvious advantage of the above procedure is that it will, by construction, result in a perfectly balanced clock tree. This is

```
Procedure: $MergingCost(T_i, T_j)$
Input: A pair of subtrees
Output: The merging cost of the subtree pair
1. Cost = Total wire length required to merge $T_i$ and $T_j$
2. EDSC = Effective downstream capacitance of the
     parent node assuming the merging of subtrees
     $T_i$ and $T_j$ using steps of Figure (3.13)
3. If $EDSC <$ Capacitance Limit
      return Cost
   else
      return $\infty$
```

Figure 3.15: The Merging cost for two subtrees.

because buffers are added only in the step 3(c) of the top-level algorithm in which *all* the unmerged nodes are buffered. As a result, the number of buffers from the clock source to *every* sink will be the same, thus satisfying one of the important objectives of our work.

A less obvious advantage of the proposed merging scheme is that, on the average, all the nodes to be *flagged* are mostly in the same ballpark as 1/2 times the effective capacitance limit used in the Figure 3.15. This results in similar equivalent capacitance loads for all the buffers at a given level. This helps to a great extent in reducing the actual SPICE skew. It may be noted here that works of [14, 45] also target the objective of balancing the loads for buffers at a given level. However, they obtain the balancing by adding excessive wire capacitance, which results in a big increase in total wire-length. In our scheme, since we merge nodes considering the wire length cost, our algorithm generally results in considerably lesser wire-length that [14, 45]. Our top-down embedding after obtaining the topology is identical to the DME algorithm [10].

3.4 Link insertion flow

We adopt the following approach to address the challenges discussed in section 3.1.1:

- The problem of inaccurate delay model and the chicken-egg relationship between link and buffer slew is addressed by using the iterative delay evaluation procedure of [53] and the bottom-up clock tree synthesis flow described in section 3.3.2.
- The problem of excessive nominal short-circuit power is addressed by considering both the spatial proximity of the nodes *and* the proximity in terms of delays during link node pair selection. This approach also makes sure that link addition does not affect the skew between other sink pairs adversely. More specifically, we use a modified cost function for the MST algorithm of [60] by making the link insertion cost as a weighted function of *both* link length and accurate delays (obtained using the algorithm of [53]) of the clock tree end points before link insertion.

The top-level algorithm of our link insertion for buffered clock tree is similar to the top-level algorithm for the unbuffered case with certain important differences like the use of accurate delay models and the use of link insertion friendly clock tree synthesis methodology. The major steps in constructing a linked buffered clock tree are:

1. Construct a balanced buffered clock tree using the flow described in Section 3.3.

2. Select the sink node pairs for link insertion using a modified form of algorithm in [60] with the cost function as weighted function of link length and proximity of accurate delays for the node pairs. The accurate endpoint delays are obtained using the algorithm of [53] for delay evaluation.

3. Since the endpoint locations are fixed, the capacitance value of each link can be calculated once the node pairs have been selected. Using the link capacitance values as extra load capacitance at the selected sinks, construct another buffered clock tree with the same topology as the one constructed in step(1). This new buffered clock tree will be equivalent to the first buffered clock tree *plus* the link capacitance.

4. Add the link resistances to the new clock tree built in step (3). The final result will be equivalent to the buffered clock tree constructed in the first step *plus* the link capacitance *and* link resistances. This is our final buffered, linked clock network.

It may be noted here that, even though the work of [78] has a similar objective of obtaining a link based buffered clock network, our approach differs from that of [78] in the following aspects:

- We use ordinary buffer cells unlike [78] which requires special tunable buffer cells.

- We use the iterative delay evaluation procedure of [53] during clock tree synthesis instead of SPICE. This makes our algorithm both fast and accurate.

- We propose a new merging scheme that results in a balanced buffered clock network that is inherently friendly for link insertion.

3.5 Sensitivity based Link Insertion

In this section, we propose our sensitivity based link insertion scheme that is very suitable for link insertion in buffered clock trees. The important contributions of this work are:

- We introduce the concept of delay sensitivity vector for the sink pairs. We use this to identify the sink pairs that are most susceptible to variation effects and insert links only between such sink pairs. This makes our algorithm very efficient and results in a robust, variation tolerant clock network.
- Our methodology is compatible to higher order delay models. Though we use Elmore delay in this paper to explain our algorithm, the same methodology can be easily extended to any higher order delay model.
- Any given variation model, like the systematic and random power supply variation, temperature variation etc. can be seamlessly integrated with our algorithm. In this work, we use random interconnect width variation as an example.

3.5.1 Drawbacks of the Existing Approaches

Sensitivity based link insertion is more suitable for buffered clock trees because of the following drawbacks of the existing algorithms:

- Most of the exiting works, except [35], do not use either delay or skew variation information while choosing links. Therefore, they might add ineffective links.

- The current methods are not compatible with the use of higher order delay models because all the parameters used for selecting the links are based on nominal Elmore delays.
- Also, the current algorithms cannot be modified for use with a given variation model. For example, if the IR drop variation information or the metal thickness variation information is available, the current methods cannot use them to choose between links that have similar parameter values, but are different only because of their variation context.
- Though the method of [35] does not suffer from some of the above drawbacks, it has very high run-time and complexity. For example, the method takes 2 minutes to add 10 links in a 74 pin clock network and more than 10 hours to add 20 links to a 597 sink clock network.

3.5.2 Sensitivity Based Algorithm

In this section, we propose the delay sensitivity based algorithm that addresses all the drawbacks of the existing algorithms. The basic idea behind sensitivity based link insertion is to insert links between the node pairs that are most susceptible to variation. This is motivated by the fact that a given link has the maximum beneficial effect on the node pairs between which it is inserted. The rest of this section presents the details of the algorithm. First, we introduce the concept of delay sensitivity vector for a given process variation model. This concept helps us to identify the link pairs that are the most susceptible to the variation effects. Following this, we prove that the α-rule proposed for unbuffered clock trees in [54] is also true for buffered clock tree case. Finally, we present our sensitivity based link selection method.

3.5.3 Sensitivity Vector

Consider Fig.3.16 which shows a typical clock tree. Let the variables x_1, x_2,. . . , x_n denote process variation variables that affect the delays and skew of the clock tree. Without any loss of generality, we can assume that these variables are independent from each other. If the variables are not independent from each other, then we can employ the method of [37] to create independent process variation variables. Let T_i denote the root to sink delay of any sink i. This can be expressed in terms of the process variation variables x_1, x_2,. . . , x_n as follows:

$$T_i = f_i(x_0, x_1, x_2, \ldots, x_n)$$

The first order Taylor expansion of T_i can be written as

$$\begin{aligned}\hat{T_i} &\approx T_{i,0} + \frac{\partial T_i}{\partial x_0}\Delta x_0 + \frac{\partial T_i}{\partial x_1}\Delta x_1 + \frac{\partial T_i}{\partial x_2}\Delta x_2 + \ldots \\ &\quad + \frac{\partial T_i}{\partial x_i}\Delta x_i + \cdots + \frac{\partial T_i}{\partial x_n}\Delta x_n \\ &= T_{i,0} + \sum_{k=0}^{n} \frac{\partial T_i}{\partial x_k}\Delta x_k\end{aligned}$$

where $T_{i,0}$ is the nominal delay of node i and $\Delta x_k = x_k - x_{k\_nominal}(k \in (0 \ldots n))$.

It may be noted here that, expressing the delay variation as a linear function of the random variables is quite accurate for all practical purposes [22]. In

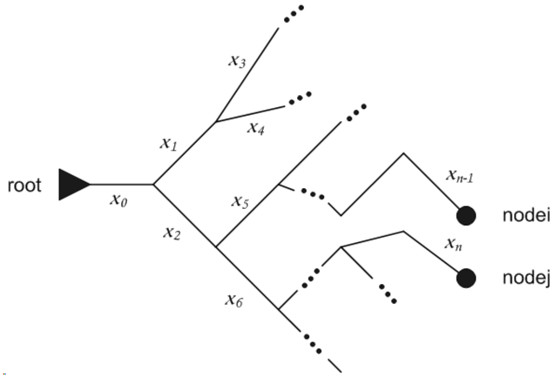


Figure 3.16: Clock Tree with Variations on Each Segment

other words, though the delay might be a non-linear function of the variables x_i, the variation in delay w.r.t. its nominal value can be treated as a linear function of changes in the random variables.

Assuming a zero nominal skew clock tree, $T_{i,0}$ is equal to $T_{j,0}$ for any two sinks i and j. Therefore, the skew between sinks i and node j in the presence of process variation can be expressed as:

$$\begin{aligned} q_{i,j} &= T_i - T_j \approx \hat{T}_i - \hat{T}_j \\ &= \begin{bmatrix} \Delta x_0 \Delta x_1 \Delta x_2 \ldots \Delta x_n \end{bmatrix} \begin{bmatrix} \frac{\partial T_i}{\partial x_0} - \frac{\partial T_j}{\partial x_0} \\ \frac{\partial T_i}{\partial x_1} - \frac{\partial T_j}{\partial x_1} \\ \vdots \\ \frac{\partial T_i}{\partial x_n} - \frac{\partial T_j}{\partial x_n} \end{bmatrix} \end{aligned} \tag{3.7}$$

Let us define the sensitivity vector for the skew between nodes i and j as:

$$S_{i,j} = \begin{bmatrix} \frac{\partial T_i}{\partial x_0} - \frac{\partial T_j}{\partial x_0} \\ \frac{\partial T_i}{\partial x_1} - \frac{\partial T_j}{\partial x_1} \\ \vdots \\ \frac{\partial T_i}{\partial x_n} - \frac{\partial T_j}{\partial x_n} \end{bmatrix} \tag{3.8}$$

Let $M_{i,j}$ be the *Magnitude* of $S_{i,j}$, we have

$$M_{i,j} = \sqrt{\sum_{k=0}^{n} (\frac{\partial T_i}{\partial x_k} - \frac{\partial T_j}{\partial x_k})^2}$$

Thus, the skew between any two sink nodes i and j can be expressed in terms of the sensitivity vector as:

$$q_{i,j} = \left[\begin{array}{c} \Delta x_0 \Delta x_1 \Delta x_2 \ldots \Delta x_n \end{array} \right] S_{i,j}$$

An important point to be noted here is that each element of the vector $S_{i,j}$ in equation (3.8) represents effect of one process related variable on the skew between sink nodes i and j. Similar to the work of [35], we also assume that the variables have a Gaussian distribution. Based on this assumption, we can regard the Δx_k ($k = 0, 1, \ldots, n$) as zero mean random variables with a known variance σ_k. Since a weighted sum of several independent Gaussian random variable is also Gaussian, the skew between sink nodes i and j, $q_{i,j}$ is also a Gaussian random variable. $q_{i,j}$ has a range of $E(q_{i,j}) - 3\sigma_{i,j} \sim E(q_{i,j}) + 3\sigma_{i,j}$, where the $\sigma_{i,j}$ is the standard deviation of the $q_{i,j}$. For a zero nominal skew clock tree, the expected value of skew is given as:

$$E(q_{i,j}) = (E(\sum_{k=0}^{n} (\frac{\partial T_i}{\partial x_k} - \frac{\partial T_j}{\partial x_k}) \Delta x_k)) = 0 \tag{3.9}$$

The range of $q_{i,j}$ is $-3\sigma_{i,j} \sim +3\sigma_{i,j}$. Similarly, the standard deviation of skew is calculated as

$$\sigma_{i,j} = \sqrt{\sum_{k=0}^{n} (\frac{\partial T_i}{\partial x_k} - \frac{\partial T_j}{\partial x_k})^2 \sigma_k^2} \tag{3.10}$$

where σ_k is the standard deviation of the independent Gaussian distribution variable Δx_k. Assuming that $\Delta x_k (k = 0, 1, \ldots, n)$ have the same standard deviation σ, we can rewrite the above equation as

$$\sigma_{i,j} = \sqrt{\sum_{k=0}^{n} (\frac{\partial T_i}{\partial x_k} - \frac{\partial T_j}{\partial x_k})^2} \cdot \sigma = M_{i,j} \cdot \sigma \tag{3.11}$$

The magnitude of standard deviation of skew between sinks i and j is indicative of how sensitive the skew between nodes i and j is to the variation effects. Therefore, we can conclude that the sink pairs that have higher value of $M_{i,j}$ are more sensitive to variation. Thus, by adding link only between sinks pairs that have the maximum value of $M_{i,j}$, we can greatly reduce the sensitivity to variations.

Obtaining Sensitivity Vector for a Generic Case: In the above derivation, we have assumed that the random variables Δx_k as Gaussian random variables with the same σ just for illustrative purpose. The concept of using the magnitude of sensitivity vector as a measure of skew sensitivity to variation is still applicable even if the random variables Δx_k had different non-Gaussian distributions.

Similarly, the same concept can also be applied with higher order delay models, including SPICE, by employing a method similar to the work of [81] to obtain the delay sensitivities. For example, we can obtain the sensitivity vector considering the effect of power-supply noise as follows. For a given buffer in the clock tree, any changes in the power-supply voltage on that buffer directly affects buffer delay and the delay of the segment directly driven by it. Also, it can have an effect on the input slews of next stage, thereby affecting the delay of the next stage slightly. In most practical cases, the effect of slews on delays can be ignored after one stage of buffers. The key point is that, for all practical purposes, we can bound the effect of small changes in power-supply voltage on a given buffer to a few segments near the buffer. Thus, we need not evaluate the delay sensitivities of all the segments in the clock tree for a given power-supply noise in a single buffer as most segment sensitivities can be assumed to be zero for all practical purposes. By extension, we can state that we can obtain the delay sensitivities of all the segments of a given clock tree w.r.t. the power-supply noise for all the clock buffers in an efficient manner. Once we know the delay sensitivities of each segment of the clock network, we can trivially obtain the sensitivities of the sink delays and thereby the sensitivities of skew between any two sinks. This method can be similarly extended to consider variation effects like L_{eff}, T_{ox}, interconnect thickness, etc. Also, the above method of obtaining sensitivities inherently considers the path sharing between different sinks. For example, when a given segment is common for two sinks, its effect will be cancelled out while obtaining the values of $S_{i,j}$ in Equation 3.8.

3.5.4 α-*Rule* for Buffered Clock Tree

The α rule that was proposed in [54] was motivated by the fact that it scales down the original skew as shown in equation 2.2. Further, it was shown in [54] that the value of α is always less than 1 for any link added between nodes driven by the same buffer. However, it was not clear whether the same α rule could be extended for links that connect nodes driven by different clock buffers. This situation can happen in a buffered clock network. In this section, we prove that the same rule is also valid for buffered clock networks where a given link can have two drivers.

The motivation for extending the α-rule for buffered clock tree is explained below. The magnitude of sensitivity vector, $M_{i,j}$, is a good measure of how much a the skew between a given sink pairs i, j is susceptible to variation effects. However, the value of $M_{i,j}$ does not measure how close two sink pairs are in terms of physical proximity. For example, two sets of sink pairs might have the same value of $M_{i,j}$, but one of the sink pairs might be physically close to each other while the other pair might be far apart. In such cases, we need to be able to add link between the sink pair that are close to each other to reduce the wire-length consumption. From the work of [54], we can conclude that lengthy link will have a high value of α. Thus, if we are able to choose links with low value of α, we will be able to have a good control over the increase in wire-length due to link insertion.

It is worth noting here that equation 2.2 was derived in [54] based on the following equation from [7]:

$$\hat{\mathbf{v}} = \mathbf{v} - \frac{v_i - v_j}{R + r_i - r_j}\mathbf{r} \tag{3.12}$$

The above equation gives the new voltage in all the nodes of any RC network when a resistance R is added between nodes i and j of the RC network. The $\hat{\mathbf{v}}$ is the vector of new node voltages and $\mathbf{v}$ is the original node voltages before adding the resistance R, v_i and v_j are the initial node voltages of i and j. The $\mathbf{r}$ vector gives the Elmore delays of all the nodes of the RC network when the capacitance at node i is +1 and at node j is -1 and all other capacitance values set at 0. v_i and v_j are the entries in the $\mathbf{r}$ for the nodes i and j.

The work of [7] also deals with situations when there are more than one voltage source by chopping up the network into sub-networks such that the each sub-network has only one source. After this, the different sub-networks are stitched together considering the fact that the final settling voltages of the different nodes at $t = \infty$ can be different from what it was when it was driven by only one source. However, in the case of a link driven by two different buffers, since the entire clock tree can be assumed to be on the same voltage domain, we can directly apply equation (3.12) for delay calculation purposes. As a result, the same α rule proposed in [54] for non-buffered clock trees is also valid for buffered clock trees. From [54], we know that lower the value of α, better the link is. Since the value of α is always less than 1, we can also state that a higher value of $(1 - \alpha)$ denotes a better link.

3.5.5 Sensitivity Based Link Insertion

Based on the last two sections, we can conclude that the effectiveness of a link between nodes i and j is proportional to $M_{i,j}$ and (1-α) where $M_{i,j}$ is the magnitude of the sensitivity vector and α is as defined in equation 2.2. Hence, we define the link sensitivity cost as follows:

$$SensitivityCost_{i,j} = (1 - \alpha_{i,j})M_{i,j} \tag{3.13}$$

Any node pair with large value of eq (3.13) can be regarded as a good candidate for link insertion. This is because if $M_{i,j}$ is high, it means that the sink pair i,j is very sensitive to variations. Similarly, if the value of α is low (or a high value of 1-α), the link can significantly reduce the final skew. Thus, we use the *SensitivityCost* as a measure of effectiveness of each link.

In this paper, as in [78], we follow the set of guidelines listed below for link insertion:

- Links are always inserted between zero nominal skew nodes.
- Node pairs are selected such that the buffers driving the links are not overloaded. Therefore, the selected sink nodes are relatively close to each other. This also makes sure that the links are not concentrated in a single place.
- Since the magnitude of the sensitivity vector is higher for sink pairs that do not share any common element, the method inherently selects pairs with considerably different source to sink paths.
- Short circuit risk in multi-driver nets is avoided by the method of [78].

The overall flow of the sensitivity based link insertion for buffered clock trees is shown below.

As shown in the algorithm, we first obtain the delay sensitivity of each clock segment under the given variation model. This step enables us to quickly get the skew sensitivities for all the sink pairs. Next, for every feasible node pair, we evaluate the value of the *SensitivityCost*. Finally, we select the top N links based on the value of *SensitivityCost* such that the total wire-length does not exceed the user specified maximum wire-length increase.

```
Procedure: Sensitivity based link insertion
Input: Buffered Clock Tree Node-List.
Output: Final link-based non-tree clock network.

1. Calculate delay sensitivity for all segments of the clock network.
2. For Each sink pair i, j do
     If Link distance ¡ Max link length AND no short circuit problem then
          Generate sensitivity vector's magnitude $M_{i,j}$
          Compute $(1-\alpha)M$
          $SensCost_{i,j} = M_{i,j} \cdot (1-\alpha)$
3. Sort $(1-\alpha)M$ and find the top $N$ links subject to wire-length constraint.
4. Add all link capacitance to the sink nodes.
5. Tune the locations of the internal nodes to restore original skew.
6. Add link resistances.
```

Figure 3.17: Node Pair Selection.

3.5.6 Advantages of Sensitivity Based Link Addition

The sensitivity based link insertion algorithm has the following merits when compared to the other existing algorithms:

- Because of the use of delay sensitivity, this method identifies the sink pairs that are most prone to the effects of variations.

- Links insertion can be modified based on a given variation model. For example, if the power-supply noise model is available, the works of [54, 56, 57, 60, 78] cannot use that information while choosing the links. However, since our scheme directly uses the variation information in the form of delay sensitivity, the links selected will be more effective.

- The work of [35], though can make use of given variation information, is extremely slow. This is mainly because [35] updates the values of mean and variance for all sink delays after every single link addition while considering all possible variation parameters simultaneously. However, since our method of obtaining the sensitivity information is a one-time process, our overall approach is much faster than the approach of [35].

- The calculation of delay sensitivity is compatible with higher order models including SPICE. Hence our algorithm can be applied even when high accuracy is required. In the case of using SPICE, we can employ a method very similar to the one used in [81] to obtain the delay sensitivities.
- Our algorithm does not involve the use of any empirical parameters such as α, β, γ, etc., as in [54, 57, 60, 78].

3.6 Experimental Results

3.6.1 Results for Balanced CTS and Link Insertion

In order to verify the variation tolerance of our new buffered clock tree and the linked buffered clock tree approaches, we run SPICE based Monte Carlo simulations (500 trials) considering both interconnect and device variations. We assume that interconnect width, load capacitance, device channel length and oxide thickness vary with a Gaussian distribution with $\sigma = 5\%$. We implemented our algorithms in C++ and experiments were run with a 3.25GHz, 2Gb memory Linux system. We use the same benchmarks as in [75].

It will be apt to compare our results with the results of [78] because of near-identical objective of our work and the work of [78]. However, the exact details of the tunable buffers in [78] were unavailable to us for direct comparison. As a result, we compare our results with the algorithms in [14] and [11]. We chose these two algorithms for comparison because the algorithm in [11] will result in a clock tree with greatly reduced wire length consumption because it is very similar to minimum spanning tree construction. So it can be a good benchmark to do the wire-length comparisons. The algorithm in [14], due to its balanced nature, is likely to yield a good and balanced clock tree with reduced skew variability. Thus, comparing our results with these two algorithms will give us appropriate benchmarks for both wire-length and skew. It may be noted that, for the major part, the code for our algorithms and our implementation for [11, 14] are identical *except* the merging schemes used. So the difference in runtime and results can be directly attributed to the different merging schemes and delay model.

Since the results of [11] are expected to yield the minimum wire length and worst skew (because of its unbalanced clock trees), we use [11] as the baseline for comparing our results. The skew variation and resource consumption for [11] are shown in Table 3.2. While selecting the clock trees for different algorithms, we made sure that all of them meet the slew requirement of 100ps on the clock tree points. As a result, the skew across benchmarks of different sizes are comparable for a given algorithm. We also made sure that the clock tree with minimal resources that met the slew criterion was selected for each algorithm so as to ensure a fair comparison between the different algorithms.

Table 3.3 shows the results of our new algorithms and the algorithm in [14] *scaled* in terms of the results of [11] (All the columns except the # Buf and CPU have been scaled). The Method column specifies the method for which results have been given. We have identified the algorithm of [14] and [11] as CTS-[14] and CTS-[11] respectively. We identify our algorithms as CTS and Link+CTS.

The wire length consumption is shown under the column titled WL. The '# Buf' column gives the number of buffers for the particular clock tree. The NS, WCS and AS denote the 'Nominal Skew', 'Worst Case Skew' and 'Average Skew' in SPICE, the last two values obtained for 500 trials of Monte Carlo simulations in SPICE. The important observations from Table 3.3 are as follows:

- From column 2 of Table 3.3, it can be observed that our buffered clock tree results in comparable wire-length to that of [11] and much less wire-length than [14].
- As expected, the skew values for [11] is the worst among all the algorithms. Also, it can be observed that our buffer insertion algorithm produces consistently better results than [14] in terms of skew variability reduction.
- The linked, buffered clock network has the best skew variability reduction among all the algorithms. Also, the percentage of extra wire-length consumed for link insertion is small and drops heavily as the size of the clock tree increases. This proves the effectiveness of link insertion for buffered clock trees.
- The CPU time consumed for the algorithm [11] is the lowest while our algorithms yields comparable CPU times. When compared to the run times of [14], the run times of our buffer insertion algorithm and the linked buffered clock network algorithm are much faster. Most notably, our runtimes are much lower compared to those reported in [78] because [78] uses SPICE.

TC	WL	# Buf	NS	WCS	AS	CPU(s)
r1	25937	16	100	190	76	0.06
r2	34110	28	96	222	60	0.36
r3	34353	36	101	196	52	0.71
r4	55115	78	176	362	76	3.46
r5	109722	163	110	226	56	9.4

Table 3.2: Skew variation and resource consumption results for the algorithm in [11]

3.6.2 Results for Sensitivity Link Insertion

In order to verify the effectiveness of the sensitivity based link insertion, we should ideally compare our results with that of the works [35, 54, 56, 57, 60, 78]. Since the skew reduction in [35] was lesser than the works of [54, 56, 57, 60, 78] and also the benchmarks used were smaller, it is sufficient to compare our results with the best from the works of [54, 56, 57, 60, 78]. However, the work of [78] uses special tunable buffers and the work does not give details about the buffer. The works of [54, 56, 60] mainly focus on unbuffered clock trees as opposed to the focus on buffered clock trees in this work. Because of these reasons, we compare our results only with the work of [57].

Similar to [57], we use the benchmarks r1-r5 downloaded from GSRC Bookshelf [26]. We run SPICE based Monte Carlo simulations with 500 trials with interconnect width as a source of variation. We assume that the interconnect widths are Gaussian random variables with a σ value of of 5%. Please note that we use this process variation model only as an example. Any other variation can also be used in our method. We use the $180nm$ technology parameters for the buffers and interconnect parameters. We implemented our algorithm in C++ and all the experiments were run on a 3.25GHz, 2GB Linux system. Table 3.4

TC	Method	WL	# Buf	NS	WCS	AS	CPU
r1	CTS-[11]	1.0	16	1.0	1.0	1.0	0.06
	CTS-[14]	5.2	18	0.57	0.72	0.46	1.1
	Our CTS	0.8	18	0.37	0.49	0.23	0.08
	Link+CTS	1.1	18	0.41	0.45	0.16	0.18
r2	CTS-[11]	1.0	28	1.0	1.0	1.0	0.36
	CTS-[14]	7.5	36	0.91	0.95	0.91	14
	Our CTS	1.8	40	0.62	0.60	0.59	0.42
	Link+CTS	1.8	40	0.65	0.39	0.37	0.52
r3	CTS-[11]	1.0	36	1.0	1.0	1.0	0.71
	CTS-[14]	9.6	41	0.59	0.57	0.61	44
	Our CTS	1.1	45	0.49	0.54	0.51	0.78
	Link+CTS	1.3	45	0.51	0.40	0.32	0.88
r4	CTS-[11]	1.0	78	1.0	1.0	1.0	3.46
	CTS-[14]	12.4	85	0.56	0.55	0.47	509
	Our CTS	2.1	83	0.34	0.42	0.36	3.94
	Link+CTS	2.1	83	0.41	0.33	0.25	4.41
r5	CTS-[11]	1.0	163	1.0	1.0	1.0	9.4
	CTS-[14]	9.1	174	0.79	0.55	0.49	2009
	Our CTS	1.5	183	0.46	0.38	0.35	10.12
	Link+CTS	1.5	183	0.48	0.30	0.28	11.62

Table 3.3: Skew variation and resource consumption results for our new algorithms and algorithms in [14] w.r.t. results of [11] in Table 3.2

shows the details of the buffered clock trees used in our experiments. The clock trees were obtained using the algorithm of [14], which was also implemented using C++. The column denoted by WL gives the wire-length of the trees. The worst case skew and the standard deviation values for each of the clock trees is given under the columns of WCS and SD respectively.

Table 3.5 presents the skew variability information for the link-based non-trees obtained using the sensitivity based link insertion method. Please note that all the values given in Table 3.5 are in terms of the values of the trees shown in Table 3.4.

From Table 3.5, we can see that the sensitivity based link insertion has

TC	# of Sinks	# of Buf	WL	WCS	SD
r1	267	32	1632666	183.4	86.2
r2	598	66	3257889	343.0	69.2
r3	862	85	3949749	293.0	63.0
r4	1903	184	8243951	360.7	88.5
r5	3101	281	12281280	361.0	91.6

Table 3.4: Size and skew variability information for buffered clock trees. WCS and SD in pico-seconds

TC	WL	SD	WCS	CPU(s)
r1	1.07	0.55	0.61	0.04
r2	1.03	0.76	0.86	0.12
r3	1.10	0.59	0.70	0.18
r4	1.11	0.60	0.82	0.74
r5	1.07	0.64	0.74	2.23
Average	1.076	0.62	0.74	0.662
% Change	+7.6	-38.0	-26.0	-

Table 3.5: Skew variation information for link based non-trees w.r.t. the results of trees shown in Table 3.4

achieved a 26% reduction in worst case skew, 38% reduction in standard deviation at the cost of 7.6% increase in the total wire length when compared with the buffered clock trees. In comparison, the method of [57] has achieved a 22% reduction in worst case skew, 30% reduction in standard deviation at the cost of 15% increase in wire-length. Thus, the sensitivity based method has achieved higher skew variability reduction while reducing the total wire-length increase by 7% compared to [57]. This indicates that the sensitivity based method is able to insert links much more efficiently than the method of [57].

Another important fact to be noted in Table 3.5 is that our sensitivity based algorithm is orders of magnitude faster when compared with the work of [35]. The work of [35] is a good comparison for run-time because it is the only work among [35, 54, 56, 57, 60, 78] which can potentially make use of any given process variation model. Even for our biggest benchmark, the run time is in a few seconds when compared to several hours of run-time for the method of [35]. Thus, our sensitivity based algorithm is very fast and achieves better skew variability reduction using lesser routing resources.

Chapter 4

Optimized Clock Mesh Network Synthesis

4.1 Clock Mesh Synthesis Problem

Among different methods suggested for clock skew variation reduction, a leaf-level mesh with a top-level tree has been shown to be very effective in reducing skew variation in several commercial chips as noted in [49, 64]. The variation tolerance of a leaf-level mesh is a direct result of its high redundancy, with multiple source to sink paths for every sink. Figure 4.1 shows an example of a clock network with top-level tree and leaf-level mesh.

The high buffer area, routing resource and power requirements of a leaf-level clock mesh have historically restricted its use to a few high-end products like microprocessors [19, 25, 34, 48, 64]. However, with variation becoming a bigger issue at 65nm technology and below, even non-microprocessor chips might consider the use of a clock mesh to improve yield. Nevertheless, most non-microprocessor chips still cannot use a leaf-level mesh because of two reasons. First, as noted above, the resource requirements (wirelength, buffers and power) might be prohibitively high. Second, there is a lack of automatic mesh planning/synthesis and optimization tools to help achieve the design objectives without manual effort [63]. Since ASICs typically have much shorter turn-around times than microprocessor chips, they cannot afford to have a manually planned and optimized clock mesh. In fact, the lack of research on automated clock mesh synthesis was noted as early as in 2001 [21]. However, no comprehensive work has been done on this practically important topic in the literature, to our best knowledge. Even in the recent tutorial on clock distribution networks [49], no systematic method has been presented for mesh planning[1] or optimization. Thus, to make clock mesh a viable option for non-microprocessor chips, a fully automated framework for mesh planning, synthesis and optimizations is needed. Such a framework can enable chip teams to achieve a smooth tradeoff between performance (skew) and power (area).

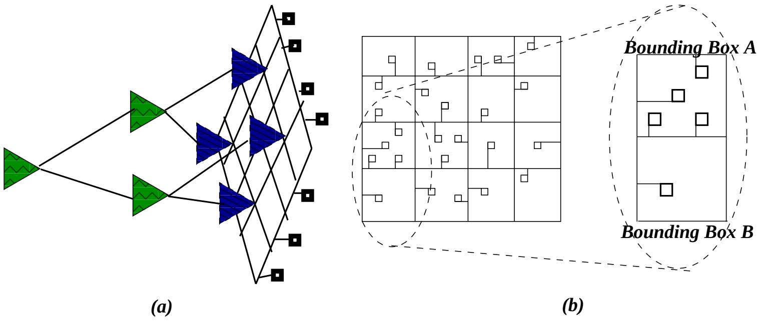


Figure 4.1: (a) A clock network with leaf-level mesh. (b) Leaf-level mesh driving clock sinks.

It may be noted that fully automated clock mesh planning/synthesis and optimization will be very useful to microprocessor chips as well. For example, automated mesh planning/synthesis can be used to get the preliminary clock mesh after which finer adjustments can be made manually. Similarly, mesh optimization can be performed on the individual *grid zones*[2] [49] to reduce power/resources used. The potential difference on the use of such automated methods between

---

[1] It is called *grid floor-planning* in [49].

[2] The individual sub-grids driving small zones of a chip.

microprocessor and other chips lies in their respective "*resource vs. skew*" tradeoff. While microprocessors might opt for maximum power reduction with a strict skew requirement, other chips might opt for minimum skew with a strict power/resource target.

Review of Existing Works: Next, we briefly review the existing works on clock mesh. The works of [25, 34, 48, 64] deal with custom/semi-custom mesh design and do not address the problem of automatic mesh synthesis/optimization. The works of [19, 77] perform optimizations on a *given* clock mesh. However the problem of obtaining a good initial clock mesh that can be optimized has not been addressed. The work of [67] deals with the synthesis of hybrid clock network with a top-level mesh and bottom-level tree. Since the size of the bottom-level trees are not negligible, they will still have considerable skew variation. The work of [19] performs clock mesh sizing considering only the nominal skew targets and ignores variation. Recent works [13, 62] present efficient methods for clock mesh analysis and they do not deal with clock mesh synthesis.

To the best of our knowledge, [77] is the first work that aims to achieve a "*variation tolerance vs. wirelength*" tradeoff in a clock mesh. Given a clock mesh and buffer library, [77] uses a set-cover formulation to obtain the minimum buffer resource to drive the mesh under slew constraints. Using this buffered mesh, [77] applies network survivability theory (used in computer networks) to remove some of the mesh segments without significantly affecting variation tolerance. The heuristic in [77] makes sure that every sink has at least a certain number of paths from certain number of buffers within a given distance in the clock mesh. The edges not present in short paths of any sink are removed, resulting in an optimized clock network. Though the work of [77] is efficient, it has a few key drawbacks as summarized below:

- It does not consider the problem of initial mesh planning/synthesis and relies on manually selected mesh for performing optimizations.
- The network survivability formulation ignores the non-uniform sink distribution and hence the effect of differential loading on buffer delays. However, non-uniform sink distribution is common in most designs [49, 64]. Also, the computer network model ignores the delay characteristics of a clock mesh which might result in incorrect optimizations.
- Electrical characteristics of mesh buffers, irrespective of their sizes, are ignored during the mesh optimization and all buffers are treated identically. Moreover the interaction between mesh reduction and buffering is ignored.
- It does not consider the reliability requirements of the clock tree after removing some of the edges in the initial mesh. As a result, the final mesh might have Electromigration (EM) violations. It may be noted that the work of [19] considers EM requirements as a constraint during mesh sizing. In the context of mesh optimization by removing mesh segments, the inverse problem of solving existing EM violation with minimum additional wire-area is needed.

In this work, we attempt to address all these drawbacks. The key components of our MeshWorks framework are:

- ***Mesh Planning and Synthesis:*** We propose a simple yet effective method that can aid in fast planning & synthesis of a buffered clock mesh for a given set of design constraints. Our method can help choose a good initial buffered mesh, which can be further optimized for power/resource reduction during refinement stage. The initial mesh so obtained can be further optimized either based on our algorithms or the existing algorithms of [19, 77].

- ***Mesh Optimization:*** We propose an efficient algorithm using *network sensitivity theory* to select the mesh edges that can be safely removed with little impact on skew variability. This formulation is more accurate than the work of [77] because the mesh delay sensitivities are directly considered during optimization.

- ***Buffer Modeling for Mesh Optimization:*** We propose an efficient buffer modeling method that is especially suitable for use during clock mesh optimization. We also present an efficient technique to resize the buffers after mesh optimization to reduce buffer area and power consumption.

- ***Wire Sizing for Reliability:*** For a given optimized mesh, we propose an effective heuristic that sizes relatively few mesh segments to meet the EM constraints.

The above contributions make MeshWorks the first comprehensive framework for complete automation of clock mesh networks synthesis and optimization. Experimental results suggest that MeshWorks can achieve significant resource reduction compared to the already optimized results of the work of [77] with similar worst case maximum operational frequency under variation. A preliminary version of this work was published in [58].

4.2 Mesh Planning and Synthesis

The mesh planning and synthesis problem can be stated as follows.

Given: Sink locations and load capacitance, buffer library, interconnect parameters, variation models, nominal/variational skew targets.

Problem: Obtain an initial clock mesh with minimum routing and buffering resources such that the given design constraints are likely to be satisfied. It shall be noted that our objective is not to get a final clock mesh, but to *quickly* get a good mesh that can further be optimized using the algorithm presented in Section 4.3.

4.2.1 Terms and Definitions

Here, we define a few common terms to facilitate our discussions.

- $S = \{s_1, s_2, ...s_N\}$ is the set of all N clock sinks, where s_i denotes the i^{th} sink.
- $B = \{b_1, b_2, ...b_T\}$ is the set of all T buffer sizes in the library with the *buffers numbered in non-decreasing order of size/drive strength.* For each buffer size b_p, the maximum load that can be driven under a given max-slew constraint Max_Slew is denoted by CL_p^{max}.
- Let $D_q(Cap)$ be the delay at the output of a buffer of size q $(1 \leq q \leq T)$ when it drives a load cap of value Cap.

- Let $IntDel(l, C)$ denote the delay when an interconnect of length l drives a load capacitance of value C.
- The leaf-level mesh, by definition, covers the entire chip area spanned by all the sinks. The X,Y dimensions of the chip area are given by X_{bound},Y_{bound}. Mesh size is defined by the number of horizontal, vertical segments denoted by m,n. Such a mesh will have $m * n$ nodes, numbered sequentially from 1 to $m * n$. Each clock sink s_i is attached to the nearest mesh node using an interconnect called *stub* of length ${L^i}_{stub}$.
- The buffers directly driving the mesh are called mesh buffers.

4.2.2 Total wire-length as a function of Mesh size

The total wire-length of the clock mesh along with the stubs can be written as :

$$L_{tot} = L_{mesh} + L_{stub} = m * X_{bound} + n * Y_{bound} + \sum_{i=1}^{N} {L^i}_{stub} \tag{4.1}$$

The wire-length of the mesh itself is a linear function of mesh size. Let us now consider the effect of increasing the mesh size on the sum of wire-lengths of all the stubs. As either m or n increases, a randomly chosen sink is more likely to have closer horizontal or vertical mesh segment. Since the number of stubs is constant, it is very likely that the total stub length decreases. In a sparse mesh, the mesh wire-length is less when compared to the dense mesh. However, the total stub wire-length is likely to be more for a sparse mesh than a dense mesh because each sink needs to be connected to the nearby mesh point using a longer interconnect. Figure 4.2 illustrates this fact with a simple example.

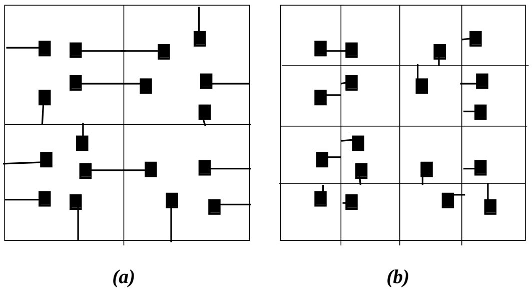


Figure 4.2: Examples of sparse and dense clock meshes. A dense mesh is likely to have shorter stubs.

Figure 4.3 shows how the wire-length changes as a function of mesh size in one of our test cases. The *key point* to be noted is that it is easy to get a plot similar to Figure 4.3 for a given set of sinks even though the shape of wire-length function might differ. From such a plot, choosing an appropriate mesh size or size range that fits our "wire-length vs. mesh-size" trade-off requirement is trivial.

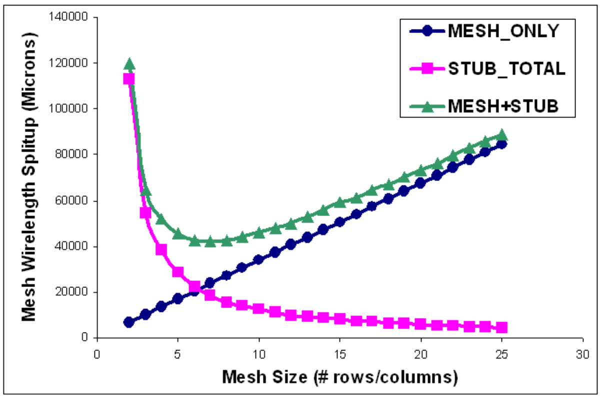


Figure 4.3: Determining the right mesh size.

4.2.3 Skew as a function of Mesh size

Skew variation is typically a decreasing function of mesh size because of two factors. First, the mesh itself becomes more dense, resulting in more redundancy, making it more tolerant to variations. Second, the length of the stub also decreases, resulting in reduction of the maximum possible uncontrolled delay variation. In general, skew in a given mesh can be expressed as a sum of three components as follows:

$$\begin{aligned} Sk_{bound} = {} & [Max(D_p(CL_p^{max})) - Min(D_q(CL_{q-1}^{max}))] + Delay(D_{max}) \\ & + IntDel(L_{stub}^{max}, C_L^{max}); \qquad \forall p, q : 1 \leq p, q \leq T \end{aligned} \tag{4.2}$$

where, $L_{stub}^{max} = Min(\frac{X_{bound}}{2n}, \frac{Y_{bound}}{2m})$ gives the maximum length of any stub when the chip area of dimension X_{bound}, Y_{bound} is divided equally into m rows and n columns, C_L^{max} gives the maximum value of sink load capacitance for the given set of sinks and D_{max} is the maximum distance between a sink and the nearest mesh buffer.

The first component in Equation(4.2) is the skew due to the differential loading/sizing of the mesh buffers. This is the difference between the maximum delay of any buffer in the library under its maximum loading condition and the minimum delay of any buffer in the library under the maximum loading condition of the *previous sized buffer* $(q-1)$. We can consider a load of CL_{q-1}^{max} to be a lower bound of the load for buffer b_q because the use of bigger sized buffer b_q when a smaller buffer b_{q-1} can be used will waste resources[3]. Thus, this term

---

[3] Refer to end of Section 4.2.3 on assumptions made in this regard.

gives a tight upper bound for the maximum skew that can be introduced in the mesh due to differential buffer loading. As stated in [64], uneven buffer loading is one of the most important reasons for the skew in a clock mesh and is typically the most dominant part of the skew.

The second component in Equation(4.2) is because of the difference in proximity of each sink to the buffer that is closest to it. Due to the redundancy of the mesh, this component will be usually small for a well driven mesh satisfying the slew requirements. If D_{max} is the maximum distance for a given buffered clock mesh, then maximum skew is equal to the delay in the segment itself. This corresponds to the worst case situation where a sink is located right next to a mesh buffer, while another is located at a distance of D_{max} from the same buffer with all other components being identical.

The third component in Equation(4.2) is due to the difference in the stub lengths and load capacitance. This component can be significant because it is uncontrolled by the redundancy of the mesh. It represents the worst case skew that can be caused when one of the sinks is located on the mesh itself and the other sink with maximum load capacitance is connected to the mesh using a stub of maximum length. Figure 4.4 illustrates the situation in which all the three factors discussed above might combine, resulting in maximum skew between two sinks shown. For the first case, a big buffer drives a big load capacitance that is located at a distance D_{max} from the buffer. For the second case, a small buffer drives a small capacitance located right next to it.

Among the skew components, the first component depends only on the buffer library and sets a practical limit on the skew obtainable using the given

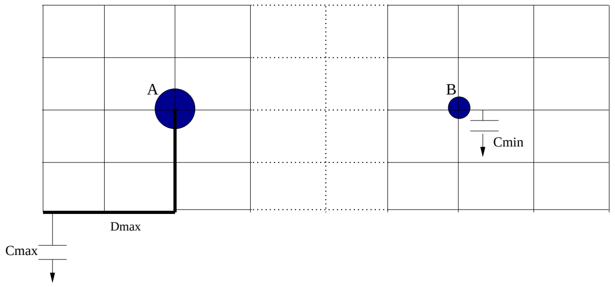


Figure 4.4: Three dominant skew components in a mesh - skew due to buffer delay imbalance, skew due to difference in distance from closest buffer and skew due to different stub length and load capacitance.

set of library buffers. The third component depends only on the mesh size and hence can be obtained for a given mesh size once we get the plots in Figure 4.3. However, to accurately evaluate the second skew component, the precise location of mesh buffers should be known. But buffer locations cannot be known unless we choose the mesh size. Thus, there is a chicken and egg problem in accurate estimation of the second component.

For a given set of library buffers and slew requirements, as the mesh is made denser, there will be addition of more mesh buffers to satisfy the slew requirements. Thus, for a randomly selected sink, the location of the nearest buffer is likely to be proportionately closer as we increase the mesh density. Another useful observation is that the value of L_{stub}^{max} scales in the same general way as the value of D_{max} as the size of the mesh is increased. Thus, we can approximate the value of D_{max} by a scaled factor of L_{stub}^{max} where the scaling factor is a function of the buffer library and the mesh buffer placement/sizing algorithm. The value of scaling factor can be estimated based on a few experiments and used for estimating the skew bound subsequently. Though this approach is an approximation and we can find corner cases where this observation need not be true, our experiments on several benchmark circuits show that this assumption is valid in practice. Also, the choice of buffer placement/sizing algorithm influences the accuracy of this approximation. For example, if the buffer placement/sizing is done in such a way that buffers are placed close to sinks, then the second factor can even be neglected from skew bound analysis. Our buffer placement/sizing algorithm, discussed in Section 4.2.4, enables us to achieve that.

Figure 4.5 shows the plot between the skew bound estimated using the above approximation and the accurate skew obtained by running SPICE Monte Carlo analysis on one of our benchmark circuits. As we can see, the skew bound, though not perfectly linear, is still monotonic w.r.t. the changes in the actual worst case skew and hence has high fidelity. We observe similar curves for all our other benchmark circuits.

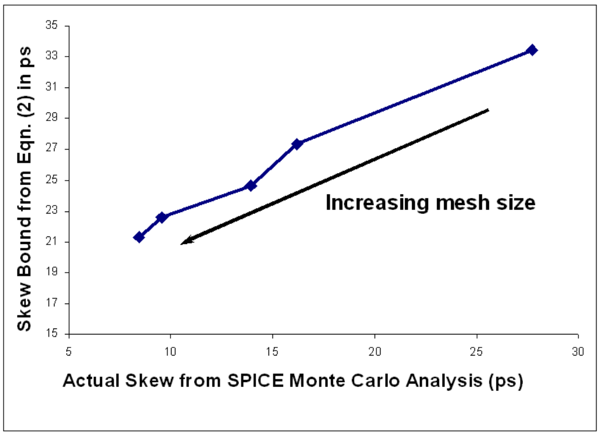


Figure 4.5: Plot showing the fidelity of the skew bound Equation(4.2). Though skew bound is not perfectly linear of actual worst case skew, it is monotonic.

Thus, Equation(4.2) can be used to get a high fidelity estimation of skew bound for a mesh of given size. Because of the closed form nature of this equation, skew bounds for a given mesh size can be estimated quickly under the assumptions discussed above. The steps to obtain the size of the initial mesh are summarized in Figure 4.6.

```
Procedure: $Obtain\_Initial\_Mesh\_Size$
Input: $L_{max}, L_{min}, S_{max} \Rightarrow$ Min, Max wire-length & Skew target.
1. Get $m, n$ such that total wire-length from Equation(4.1) is $\simeq L_{min}$.
2. Using the current $m, n$, obtain $L_{tot}$ using Equation(4.1).
3. If ($L_{tot} \geq L_{max}$)
      Print "Relax design constraints"
      Quit.
4. Obtain value of $Sk_{bound}$ from Equation(4.2)
5. If ( $Sk_{bound} \leq S_{max}$ )
      Return $m, n$. Stop.
   Else
      Increment values of $m, n$ by 1 and go to step 2.
```

Figure 4.6: The top-level algorithm of selecting the initial mesh size.

In practice, the value of S_{max} parameter used as input to Figure 4.6 should be chosen such that it is not too tight. This is because the value of Sk_{bound} obtained from Equation(4.1) is always pessimistic since it is a bound for the worst possible skew for a given mesh size.

A note on Equation(4.2): It may be noted that Equation(4.2) inherently makes the following assumptions:

- Several buffers of incrementally different sizes/drive strengths are available to make the target skew physically possible. As noted in [3], most practical libraries will have hundreds of different buffer sizes to choose from. Hence this assumption is valid in practice.

- The buffer placement/sizing is done such that the smallest buffer that can drive a given set of loads will be used. In other words, we assume that all the buffers in the library have a valid capacitance range, which is used to choose the smallest buffer for a given load. This assumption is also valid in practice as power/area reduction is a key objective of any clock network synthesis algorithm.

4.2.4 Mesh Optimization Friendly Buffer Placement/Sizing

The buffer insertion heuristic of [77] has two main drawbacks. First, the potential impact of buffer insertion on mesh optimization is not considered. This might result in buffer insertion at nodes that could have been optimized if the buffer were not present. Second, the cost function used in the set-cover formulation of [77] ignores the low-pass filter characteristics of an RC mesh[13, 62]. For an RC mesh, the attenuation of a ramp signal applied at a given node increases exponentially as a function of distance from the node. This attenuation is constant for a given clock frequency. Hence, inserting several small buffers distributed throughout the clock mesh instead of fewer big buffers might result in lesser buffer area and improve slew at the clock sinks. This is illustrated in Figure 4.7. The solution in Figure 4.7-a uses two smaller buffers to drive the same amount of load instead of one big buffer in Figure 4.7-b. Considering the attenuation characteristics of an RC mesh, the solution in Figure 4.7-a will result in lesser slew rate for a given buffer area. In other words, for a given slew requirement at the clock sinks, the solution in Figure 4.7-a will result in lesser buffer area. However, the work of [77] might randomly pick one of them.

To address the above drawbacks, we propose the following cost function

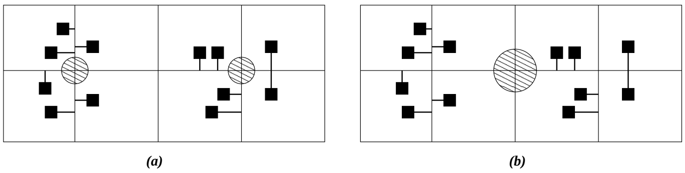


Figure 4.7: An example where the buffer insertion algorithm of [77] might not take the better choice. The shaded circles represent buffers of proportional size.

for the greedy set-cover algorithm of [77]. The cost of inserting a buffer of size p at node i of the clock mesh is given as:

$$Cost_i^p = \frac{{b_p}^2}{{b_T}^2} * \frac{1}{N_{uncov}} * \frac{1}{C_{Load}^i} \tag{4.3}$$

where, b_T is the biggest buffer in the given library, N_{uncov} is the number of uncovered nodes that can be covered by the buffer under consideration, C_{Load}^i is the value of capacitance at the mesh node i, including the capacitance of all the sink nodes attached to it. The advantages of using the above cost function are:

- Use of $\frac{{b_p}^2}{{b_T}^2}$ term instead of b_p term of [77] 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 total area. Thus, this cost function indirectly considers the RC attenuation effect of the mesh.

- The $\frac{1}{C_{Load}^i}$ term lowers the cost of adding a given buffer closer to the sinks even if coverage can be done from a farther node. This reduces the RC attenuation by placing the buffers closer to the sinks. Also, this makes the buffer locations *optimization friendly* as the edges connected to buffers are less likely to be removed because of the close proximity to the sinks. Section 4.3.4 has more details on this.

- Similar to the work of [77], the cost function is inversely proportional to the number of new, uncovered mesh nodes that can be covered by the buffer under consideration.

The other aspects of the set-cover formulation are same as in [77] and are omitted here due to page limit.

Impact of mesh buffer placement on top-level clock tree: The increased number of mesh buffer from the above buffer placement method might increase the wire-length of the top-level clock tree. However, this effect is compensated by two opposite effects, which is explained next. Comparing the situations in Figure 4.7 a and b, case-b will have fewer mesh buffers and hence lesser top-level wire-length. However, the capacitance of the single end point will be high because of bigger buffer. Also, the mesh optimization cannot remove the edges that connect the big buffer to the two clusters of sinks on either side, increasing the wire-length of the mesh. In contrast, the situation in case-a has more end points in the top-level, which can increase the top-level wire-length. However, the total pin capacitance is lower than in the first case because, to achieve comparable slew rates at the sinks, case-a can use smaller buffers with lesser total area. Also, several extra mesh edges can be optimized away, resulting in lesser mesh wire-length. Thus, case-a reduces both the top-level pin capacitance and the mesh wire-length at the cost of increasing the top-level mesh wire-length. Since the top-level wire-length addition is only a tree (which can be represented by sum of distances from the single big buffer to the small buffers that replace the big buffer) and since the potential optimization that can be done in the mesh can result in removal of several mesh edges, the above approach typically results in an overall reduction in total capacitance of the clock network.

4.3 Network Sensitivity Based Mesh Optimization

In this section, we will first review the concepts of network sensitivity theory that is at the root of our mesh optimization approach. Next, we present our efficient buffer model that is used during the mesh optimization. Finally, using these concepts, we present our network sensitivity based mesh optimization algorithm.

4.3.1 Network Sensitivity Theory

Given a RC network, network sensitivity theory aims to efficiently evaluate sensitivities of a given output parameter (voltage or current) to changes in the circuit parameters. A straight forward and inefficient method to obtain the sensitivities is to perturb each circuit parameter and observe the changes in the output. However, in the case of RC networks with no active elements, the sensitivities of a given output can be obtained w.r.t. *every* parameter in the network using the method of [36] without perturbing any circuit parameters.

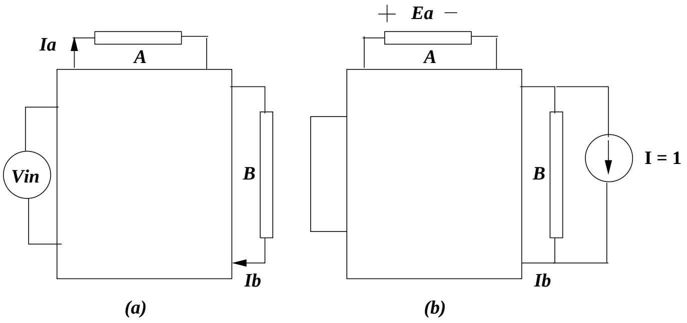


Figure 4.8: Network sensitivity theory can be applied for clock mesh optimization.

Consider the Figure 4.8 (from [36]) which shows a generic electrical network with 3 identified elements for illustrative purposes. The elements can be any of the passive components like R, C and L. Let I_A, I_B, I_C be the currents through these elements in the nominal circuit. The element V_{in} represents all the sources in the RC network. Let the voltage across element B be considered as the output of this network. According to [36], to obtain the sensitivities of the output voltage w.r.t. all the parameters of the circuit, *irrespective of the number of circuit parameters*, we need to construct an auxiliary network for the original network as follows: all the independent current sources are opened, voltage sources are shorted, a unit current source is applied across the element B and the voltages across all the components in the network are measured. According to [36], the relationship between the currents of the original network, element values and voltages in the auxiliary network is given as:

$$E_a = \frac{\partial E_b}{\partial A} * \frac{A}{I_A} \tag{4.4}$$

where, E_a is the voltage across any element in the auxiliary network, $\frac{\partial E_b}{\partial A}$ is of sensitivity of the output voltage E_b w.r.t. parameter A (the required value), and I_A is the current flowing through the element A in the original network. Thus, using only two simulations, the sensitivities of a given output w.r.t. all the network parameters can be obtained, irrespective of the number of parameters. Though the method of [36] is efficient when compared to the perturbation method, it still requires one simulation for each output. Thus, a direct application of this method is not practical for multi output networks, like clock networks. Our method to overcome this drawback is explained in Section 4.3.3.

4.3.2 Accurate Buffer Modeling For Mesh Optimization

The sensitivity calculation method of [36] can be applied only for a passive network. To apply the concepts of sensitivity theory to a clock mesh, all the clock buffer must be modeled using a combination of voltage/current sources and passive elements like resistors and capacitors. The typical switch resistance modeling of buffers is becoming increasingly inadequate to approximate the buffers in the sub 100nm technologies. This inadequacy is compounded by the inherent difficulty in modeling the effective capacitance of a mesh because of its multiple paths and multiple drivers that can possibly interact in highly non-linear fashion. Thus, we need a buffer model that is accurate, independent of the load and also captures the non-linear behavior of the buffers. The buffer model proposed in [66] satisfies all these requirements. The basic idea behind the work of [66] is the use of a two-pole approximation for modeling a buffer instead of the single pole approximation of a switched resistance modeling. As a result, it can be characterized almost independent of the load that it drives and captures the non-linear behavior of the buffers. An example of this model is shown in Figure 4.9 where S is the size of the buffer. The values of $R1, R2, C1, C2$ are obtained by using the OPTIMIZE function in HSPICE[29] to approximate the delay characteristics of a given buffer. In this work, we adapt the work of [66] to make it suitable for the problem approximating a library of buffers.

The work of [66] concentrates on modeling a single buffer. Though this can be trivially extended for a library of buffers, the values of the parameters $R1, R2, C1, C2$ can differ drastically based on the initial values used in the OPTIMIZE function of HSPICE. Ideally, we would want monotonic changes in the values of $R1, R2, C1, C2$ for monotonic changes in buffer sizes. This requirement is feasible under the assumption that a bigger buffer is used to drive proportionally bigger load under a given slew target. The monotonic property of $R1, R2, C1, C2$ parameters ensures that the any buffer resizing done with these models will be accurate. The monotonic characteristic of the RC parameters can be guaranteed by first obtaining a good approximation for either the smallest or the biggest buffer size in the library using large search space. For all the other buffers, the approximations are obtained by constraining the maximum or minimum values of $R1, R2, C1, C2$ to the values of the previous or next sized buffers using the OPTIMIZE function in HSPICE. From our experiments, we observed that this always preserves the monotonic nature of the parameters while resulting in accurate approximations.

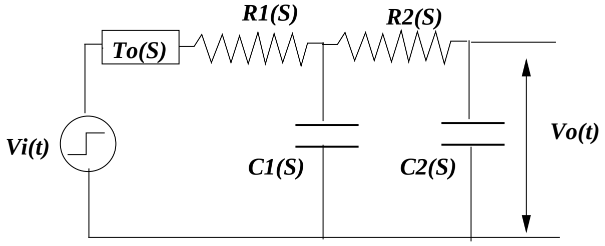


Figure 4.9: Accurate buffer used for clock mesh optimization. S is the size of a given buffer

Accuracy of the buffer model: Figure 4.10 compares the clock sink delays for one of our mesh test cases with original buffers and the buffer models. As seen from this figure, the two delay curves track well across the clock sinks. We observe similar results for all our test-cases. For all the test-cases, the error because of our buffer models is around 4% for delays and 1% for skews. Thus, any optimization done using our buffer models is likely to be accurate.

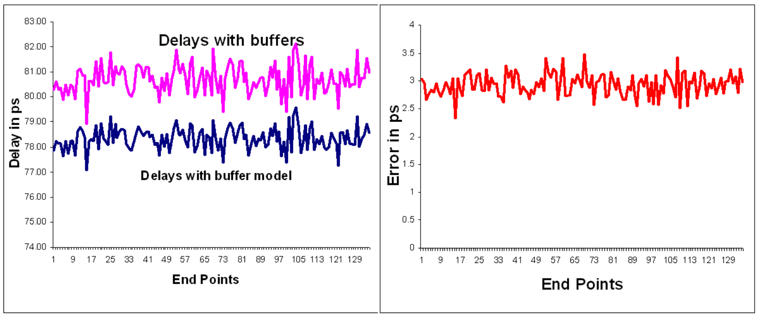


Figure 4.10: Comparison of sink delays in SPICE obtained using buffers and the buffer model for a clock mesh test-case.

4.3.3 Mesh Optimization Algorithm

To minimize the mesh wire-length without significantly affecting the variation tolerance, the mesh segments that are not critical for variation tolerance should be removed. Consider Figure (4.11) where a mesh drives several clock sinks. In this case, the edges shown in dashed lines can be safely removed without significantly affecting the skew characteristics of the original mesh because they are far away from all the clock sinks. Similarly, we would like to have a dense mesh in places where the clock sink distribution is high and a sparse mesh in locations where the density is much lower.

In this work, we attempt to achieve the above objectives using network sensitivity theory as explained below. Let Del_i denote the delay of a sink s_i and let a mesh segment connected between mesh nodes p and q be denoted by $Seg(p,q)$. The delay sensitivity for the sink s_i w.r.t. width $W(p,q)$ of the mesh segment $Seg(p,q)$ can be expressed as:

$$\frac{\partial Del_i}{\partial W(p,q)} = \frac{\partial Del_i}{\partial R(p,q)} * \frac{\partial R(p,q)}{\partial Wp,q} + \frac{\partial Del_i}{\partial C(p,q)} * \frac{\partial C(p,q)}{\partial Wp,q} \tag{4.5}$$

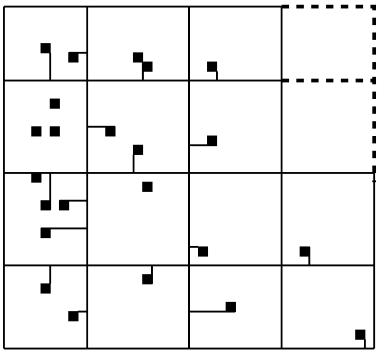


Figure 4.11: Simple example of network sensitivity based mesh optimization.

In the above equation, the terms $\frac{\partial Del_i}{\partial R(p,q)}$ and $\frac{\partial Del_i}{\partial C(p,q)}$ are the values of delay sensitivity w.r.t. the resistance and capacitance of the mesh segment. There is no closed form expression for evaluating these terms. The terms $\frac{\partial R(p,q)}{\partial Wp,q}$ and $\frac{\partial C(p,q)}{\partial Wp,q}$ are the changes in resistance and capacitance values of the mesh segment as a function of width. These terms can be easily obtained though the relationship between interconnect width and resistance/capacitance values. For the simple case of $R(p,q) = \frac{R0(p,q)}{W}$, and $C(p,q) = C0(p,q) * W$, these expressions are $-\frac{R0(p,q)}{W^2}$ and $C0(p,q)$ respectively. In order to select mesh edges for removal, we first quantify the effect of removing each edge by defining the following cost function for each mesh segment $Seg(p,q)$:

$$Cost(p,q) = Max\left(\frac{\partial Del_j}{\partial W(p,q)} - \frac{\partial Del_k}{\partial W(p,q)}\right) * W(p,q) \tag{4.6}$$

$$\forall j,k \in sinks\ S.$$

The above cost function approximates the maximum change in skew in the entire mesh when a given segment is removed. The criticality of each mesh segment w.r.t. variation tolerance will be proportional to the value of cost function. The basic idea of our approach is to remove the segments that have a low cost function, resulting in an optimized mesh. However, the following sub-problems must be solved for efficient application of network sensitivity towards solving mesh optimization problem:

- As stated in Section 4.3.1, the method of [36] is inefficient for clock network which has many output (sink) nodes.
- The method of [36] can be used only to obtain voltage sensitivities and cannot be directly used to obtain delay sensitivity.
- The sensitivities for a given segment assumes that all the other mesh segments are held constant.

The above sub-problems can be easily solved when the following key observations are considered:

- The RC mesh network behaves as a low-pass filter [13, 62], in which the attenuation of a ramp input signal applied at a given node increases exponentially as a function of distance from the source node. As a result, the delay sensitivities of a given set of closely located sinks will be almost the same w.r.t. most clock segments. This assumption is not valid for mesh segments located close to the sinks. However, such mesh segments are less likely to be optimized out. We can use this observation to drastically reduce the number of output nodes (sinks) for which delay sensitivity needs to be evaluated.

- According to [22], the effects of most variation can be modeled using linear approximation without any significant effect on the accuracy. As a result, we can obtain the delay sensitivity terms of Equation(4.5) using Elmore delay without accuracy loss.

- The Elmore delay sensitivities can be obtained efficiently by evaluating the voltage sensitivities of the DC equivalent network of the original mesh network. The DC equivalent circuit can be obtained by shorting all voltage sources and replacing all the capacitances by current sources of equal magnitude [6]. The node voltages in this circuit represent the Elmore delays of the original mesh networks and the voltage sensitivities are the sensitivities of the Elmore delays.

- The sensitivities of several output voltages in the DC equivalent circuit can be evaluated efficiently by reusing the results of *LU* factorization. This is because the only change made in solving the different auxiliary networks for different output nodes is the location of the unit current source [6].

- The analysis efficiency can be further improved by exploiting the sparse nature of the nodal admittance matrix for most RC mesh networks [6].

Using the above observations, the value of cost function of Equation(4.6) can be evaluated for each mesh edge efficiently.

Overall Mesh Optimization Algorithm:

1. Identify the different sink clusters such that sinks in each cluster are closely located.

2. Obtain an approximate circuit by merging all the sinks in each cluster into a single *merged* sink with capacitance equal to the total capacitance of all the merged sinks. The resulting mesh will be a good approximation of the initial mesh as far as sensitivity calculations are concerned and will have far fewer end points compared to the original mesh.

3. Replace all the mesh buffers with the accurate buffer model values presented in Section 4.3.2.

4. Obtain Elmore delay sensitivities of every *merged* sink w.r.t. all the mesh segments by efficient reuse of the results of LU factorization and making use of sparse matrix methods. Using the delay sensitivities, obtain the $Cost(p, q)$ for each mesh segment as defined in Equation(4.6).

5. Sort mesh segments in increasing order of $Cost(p, q)$ value and remove the required number of segments to satisfy the wire-length reduction target. The mesh segments are selected such that no two removed mesh segments are at a distance of N nodes from each other. This requirement makes sure that any interactions between the mesh segments removed is negligible. A higher N implies fewer edge removal and lesser modeling error because of the interactions between the mesh edges.

4.3.4 Buffer-resizing for Mesh Optimization

A key drawback of [77] is that the optimized mesh uses the same buffer placement/sizing as the initial mesh. This can result in buffer area and power wastage. In this work, we propose an efficient buffer resizing heuristic to reduce the buffer area/power for a given optimized mesh. The main steps in our approach are:

1. For each clock buffer, obtain the rectangular covering region in the mesh where the total capacitance (including sink capacitance) is less than buffer load limit under the given slew constraint.
2. For each buffer that has an overlap with another buffer, consider resizing to the previous sized buffer such that the total covering region for all clock buffers is maintained.
3. Repeat this process till there exists no buffer that can be sized down without reducing the total coverage.

The amount of buffer area reduction obtained by the above heuristic is proportional to the reduction in mesh wire-length. However, the proportional reduction in power is likely to be less because the redundant buffers in the optimized mesh were driving light loads.

4.4 Wire Sizing for Reliability

As noted in [8, 38, 39, 68, 69, 71], electromigration (EM) is increasingly becoming a significant issue in the deep sub-micron IC designs. In general, EM is relevant to clock mesh because of the significant current flowing in it. EM is especially relevant to clock *mesh optimization* because removing any mesh segment can potentially increase the current density in a nearby segment. This problem has been implicitly addressed in our framework to a partial extent. During mesh optimization, we make sure that *no two removed mesh segments are at a distance of N nodes from each other.* This was primarily done for making sure that the interactions between mesh segments is negligible in terms of variation tolerance. The same step also helps in reducing potential EM violations because the edges removed are not very close to each other. As a result, any increase in current density because of mesh optimization will not be concentrated in a small region of the mesh. However, new EM violations can still happen because the above method does not directly measure current density. Thus, we need a systematic approach to address any EM violations that might occur due to mesh optimization. This section proposes an efficient method to address this issue systematically.

According to the empirical model developed in [4], the Mean Time To Failure (MTTF) of a wire considering EM issue is given as:

$$MTTF = \frac{C}{J^n} * exp(\frac{E_a}{k.T}) \tag{4.7}$$

where C and n are empirical constants, J is the average current density, E_a is the activation energy for the electromigration mechanism, k is the Boltzmann constant, and T is the temperature. Thus, the only parameter that can be adjusted during mesh optimization is the current density. The current density can be adjusted either by controlling the mesh edges that are removed during optimization or by wire-sizing after mesh optimization. Accurate implementation of the first method requires analysis of the mesh for EM violations after removing every mesh segment and hence is very costly in terms of run-time. The second method of wire-sizing is better because only mesh segments with EM issues in the optimized mesh need to be sized. As a result, we choose the second strategy. The complete details are described next.

The central fact used in our wire-sizing scheme is based on the observation that EM depends primarily on the asymmetric bidirectional currents as described in [71]. Thus, to a large extent, the value of current density J in a given mesh segment can be derived from the average DC current in it. This fact is also used in the work of [19] in which the average DC current can be easily computed from node voltages of the equivalent RI network of the mesh RC network. The equivalent RI network of a given RC network is obtained by replacing all capacitors with current sources of equal value and retaining the same resistance values. Thus, from [19], for a given pair of nodes i and j, the average current in the mesh segment between nodes i and j is given as:

$$I_{ij} = \frac{2V_{DD}}{T_{clk}} * (\frac{T_i - T_j}{R_{ij}}) \tag{4.8}$$

where i, j are the two nodes connected by the mesh segment under consideration, the factor 2 assumes a 50% duty cycle clock, V_{DD} is the supply voltage, T_{clk} is the time period of the clock signal, T_i and T_j are the first order (single pole) approximations of voltage at nodes i and j respectively and R_{ij} is the value of resistance of the mesh segment. The values of first order approximations of voltages at the mesh nodes can be obtained [19] as:

$$T = [T_0\, T_1\ \ldots\ T_n]^T = G^{-1}\mathbf{C} \tag{4.9}$$

where G is the $N\ X\ N$ admittance matrix of the mesh network and $\mathbf{C}$ is $[C_0\ C_1\ \ldots\ C_n]^T$ is the vector of node capacitances. Thus, the average current and hence the current density of a given mesh segment can be efficiently obtained from Equations (4.8) and (4.9).

We wish to point out here that the main difference between our work and the work of [19] is in the way we use Equation (4.8). The work of [19] uses the equation to identify the minimum sizes for mesh segments while down-sizing mesh segments to save area. However, we use the equation to guide our up-sizing of mesh segments to solve the EM issues in the optimized mesh. In other words, [19] uses the equation to *prevent* EM issues while recovering area, while we use it to *solve* EM issues that arise after our mesh optimization. Since our mesh optimization ensures that the optimized mesh itself is variation tolerant, any increase in mesh widths for fixing EM violations is likely to make the mesh more variation tolerant. However, in [19], variation tolerance is not considered.

Our overall wire-sizing scheme for solving EM violations is shown in Figure 4.12. In essence, we iteratively identify all the mesh segments with current density issues using Equation (4.8) and then increase the widths in linear proportion to the magnitude of violation. The linear increase is motivated by the fact that, for a given current, the current density reduction is directly proportional to the width increase. The iterations continue till all the violations are fixed. Since all the mesh segments with EM problems are sized-up in each iteration, the number of mesh segments with more EM violations after a given iteration is usually very small. Hence the total number of iterations for a given optimize mesh is also very small. This is also confirmed by our experimental results.

```
Procedure: $Fix\_EM\_Violations$
Input: Optimized mesh, Max. Current Density $J_{max}$.
Output: Final mesh with EM violations fixed.
1. Get $J_{avg}$ for each mesh segment using Eqn. (4.8), (4.9)
2. $N\_violators$ = # mesh segments with $J_{avg} > J_{max}$
3. while($N\_violators > 0$)
    (a) For (all segments with violations)
        (i) $scale = \frac{J_{avg}}{J_{max}}$
        (ii) $Width_{new} = Width_{old} * scale$
    (b) Get $J_{avg}$ for all mesh segments using Eqn. (4.8), (4.9)
    (c) $N\_violators$ = # mesh segments with $J_{avg} > J_{max}$
4. Output optimized mesh with new widths
```

Figure 4.12: Iterative wire-sizing flow to fix EM violations.

Integration in MeshWorks Flow: The wire-sizing based solution for solving EM violations is a natural addition to the rest of the MeshWorks flow since the input to this step is the optimized mesh from mesh optimization step. One potential issue that might arise because of the wire-sizing is that the increase in mesh capacitance might result in slew violations due to overloading of the mesh buffers. As a result, we might need one more round of mesh buffer sizing, which might in turn trigger new EM violations. Thus, in theory, the loop between wire-sizing and buffer sizing might not be closed. However, this does not happen in practice. The main reason is that the mesh buffering/sizing problem, which is based on set-cover problem, is an NP-complete problem. Thus the actual solution from the greedy mesh buffering/sizing is always sub-optimal and hence results in buffer sizes such that mesh is slightly over driven. Thus, as long as the increase in capacitance of mesh segments due to wire-sizing is small, buffer sizing will not be required. This analysis is confirmed by our experimental results presented in

Section 4.6.3.

4.5 Practical Considerations in the Use of MeshWorks

In this section, we discuss some additional issues and extensions of MeshWorks.

Blockages: An important issue to be considered during clock network synthesis of most designs is the presence of blockages. When clock *trees* are used, using a synthesis algorithm that is blockage aware is absolutely essential. Otherwise, a buffer or an internal clock node might be moved significantly after clock tree synthesis, thereby potentially changing the intended skews significantly. However, the MeshWorks framework can work seamlessly even for chips with blockages. For example, consider the Figure 4.13 that shows a simple example of a chip with a single blockage with a full clock mesh laid on the top. The semi-circular pin shown represents the location of the clock pin of the blockage. Here, we are assuming that the blockage is a hard-macro that has a clock pin. The analysis that follows is equally valid when it is any other type of blockage and when there are multiple blockages.

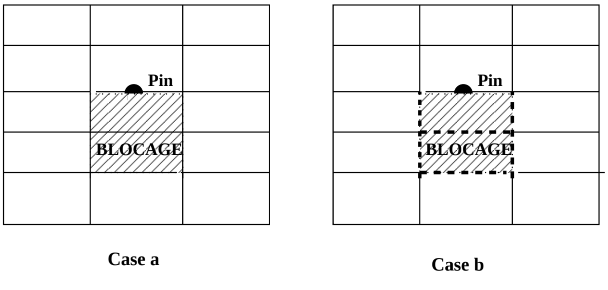


Figure 4.13: MeshWorks can be seamlessly applied for chips with blockages.

In this case, the mesh optimization problem is identical to the one obtained by replacing the blockage with its clock pins to be connected to the mesh. Since the area of the blockage will not have any other clock sinks, the mesh segments within this area *will naturally get optimized away*. This is shown in *case-b* of Figure 4.13 where the dashed lines indicate that the corresponding mesh segments have been removed. Another simple way to address this issue is by using a pre-processing step in which any clock mesh segment that is overlapping with the blockages can be simply deleted. One caveat to be noted while doing this step is that the blockage edges need not overlap exactly with the mesh edge locations. In such a situation, a simple modification of the local mesh segments to avoid overlaps can be undertaken without impacting the overall applicability of MeshWorks optimization framework.

Multi-Clock Floorplans: One of the main reasons why clock meshes are not used even in high performance ASICs is that they typically require multiple clocks to interact heavily and so they will have sinks of multiple clocks interspersed in the same floorplan. As a result, using a mesh structure for the clocks will require two separate meshes covering the entire floorplan, which is obviously unaffordable due to power/resource constraints. However, our clock mesh optimization scheme can recover most of the unnecessary clock mesh segments even if starting from complete meshes. This can make the use of clock meshes in multi-clock floorplans a viable option.

Highly uneven load distribution: The practically significant issue of uneven load distribution in different parts of a large chip can be addressed effectively using the MeshWorks framework. Such a situation can happen in reality when different IPs from different vendors are merged to create large System-On-a-Chip designs. Even in situations like this, the MeshWorks framework can be used effectively. One straightforward method is to start the mesh optimization with a dense mesh that will work for the most dense region of the chip. Since our method will automatically optimize away unnecessary edges that do not contribute to skew variation tolerance, the mesh segments in the regions with light load distribution will be optimized away naturally. Another method is to divide the entire chip area into several regions of vastly different flop densities and use mesh works independently on each of them. Finally, each of the optimized sub-meshes can be connected with each other using the minimum number of mesh segments. This last approach is similar to the method described in the recent tutorials on clock distribution networks [49] in which the chip area is divided into several *grid zones* which differ in loading and density.

Experimental results to verify the working of our mesh optimization scheme under each of the above issues are presented in Section 4.6.4.

4.6 Experimental Results

4.6.1 Experimental Setup

We use the results of the recent work of [77] as it has the closest objective to that of ours. To make a valid comparison, we obtained the results of the method of [77] from the authors for *our buffer library and slew constraints.* The number of buffer types used in [77] was 4, much lesser than what is available/used in most practical libraries/designs [3]. Also, the nominal slew constraint used in [77] was 150ps, which is 15% of even a GHz clock frequency. As mentioned in [64], a slew of around 10% of the clock frequency is common considering all process corners. Also, clock nets are typically well buffered to maintain tighter slews than signal nets. Considering these facts, we used 12 different buffer sizes with max-capacitance limit ranging from 60fF to 300fF and a nominal slew constraint of 75 ps for all the different methods. All *other* experimental conditions are identical to [77]. In particular, we use the same 65-nm technology parameters and transistor models from *bptm* [27] and same set of benchmark circuits. Also, we modeled the effects of variation on the top-level clock tree in the same way as in [77] by modeling the input arrival time for the mesh buffers by a random variable with a maximum variation range of 50ps. Other variation parameters considered are buffer channel lengths, interconnect width, power supply variation and sink load capacitance variation. The above parameters are varied with 5% standard deviation around the nominal value. The spatial correlation in variation is accounted by the method of Principal Component Analysis [9].

4.6.2 Mesh Planning, Synthesis and Optimization Results

The complete results of different mesh optimizations are shown in the Table 4.1 and 4.2. Table 4.3 shows the average improvement for all 6 test cases in Table 4.1. The different mesh optimization approaches of our work and that of [77] are compared w.r.t. the manually selected mesh used in [77]. According to the authors of [77], the mesh sizes were chosen in such a way that a target nominal skew is obtained with minimum mesh wire-length. This manual mesh is denoted by "MM" in our tables. We directly compare the effectiveness of our optimization algorithms with the method of [77] by performing optimizations on this initial mesh [4]. The mesh optimization method of [77] is denoted by "MO[77]" and our network sensitivity based approach, along with buffer resizing is denoted by "NSMO" (Network Sensitivity Mesh Optimization). In order to measure the effectiveness of our mesh planning & synthesis approach, we obtain the best mesh chosen by the algorithm in Figure 4.6 for the same set of benchmark circuits and design constraints. This approach is denoted by "MP&S" (Mesh Planning and Synthesis) in the tables. Finally, we run our optimization algorithm on the mesh obtained from our mesh planning and synthesis algorithm. This approach is denoted by "MPSO" in our tables. The columns under "%Red" are the relative reductions w.r.t. "MM".

The different parameters in Table 4.1 are buffer area (BA), total wire-length (WL), power (PWR) and mean/standard deviations of skew (μ_{sk}, σ_{sk}) considering variations (obtained by SPICE Monte Carlo simulations). The last column in Table 4.1 gives the worst case maximum frequency, F_{max}, at which the clock network can be run in the presence of $\mu_{sk} + 3\sigma_{sk}$ skew variation assuming the *ideal* target frequency to be 1GHz. Similar to [77], we also use the percentage reduction in max frequency under variation as the measure of variation tolerance instead of changes in skew. This enables us to directly compare the power/area vs. frequency trade-off. Instead, if we directly consider the increase in skew, even a change from a skew of 1ps to 2ps will be a 100% change but it does not convey the actual trade-off between frequency of operation and resources.

The key observations from the Tables 4.1, 4.2 and 4.3 are:

---

[4] Only the mesh itself is identical to the one used in published results of [77] and not the buffer placement, buffers and slew constraint used. Also, the wire-length reported in [77] did not include the stub wire-lengths.

Case	Method	Size	BA	WL	PWR	μ_{sk}	σ_{sk}	F_{max}
(#Sinks)			μm^2	mm	(mW)	(ps)	(ps)	MHz
s9234 (135)	MM	9X9	36.5	44.1	9.8	8.8	2.3	984
	MO[77]	9X9	36.5	40.9	9.2	15.7	4.4	971
	NSMO	9X9	35.2	32.1	8.3	14.5	2.6	977
	MP&S	7X7	35.0	42.0	9.3	18.1	4.0	970
	MPSO	7X7	31.4	33.6	8.0	29.8	5.5	955
s5378 (165)	MM	10X10	38.6	46.8	10.4	7.0	2.0	987
	MO[77]	10X10	38.6	39.4	8.9	19.5	6.1	963
	NSMO	10X10	38.2	32.6	7.9	30.8	2.1	964
	MP&S	8X8	36.7	44.2	9.9	11.1	2.8	980
	MPSO	8X8	30.1	31.0	6.7	35.2	5.8	949
s13207 (500)	MM	30X30	155.9	175.5	39.7	7.7	1.7	987
	MO[77]	30X30	155.9	131.6	31.1	13.7	2.6	978
	NSMO	30X30	115.0	122.9	30.3	17.0	3.2	974
	MP&S	13X13	98.3	116.7	26.2	12.9	2.3	980
	MPSO	13X13	90.2	82.8	20.6	27.1	2.3	966
s15850 (566)	MM	30X30	167.4	191.5	43.2	7.6	1.6	987
	MO[77]	30X30	167.4	127.6	31.1	24.5	3.9	964
	NSMO	30X30	123.5	109.2	27.1	17.2	2.8	974
	MP&S	15X15	113.9	137.2	30.7	10.4	2.1	983
	MPSO	15X15	101.2	84.0	22.0	23.5	4.0	965
s38584 (1426)	MM	40X40	381.4	455.3	101.6	8.6	1.5	986
	MO[77]	40X40	381.4	345.9	79.3	15.3	1.9	979
	NSMO	40X40	342.6	318.7	77.3	21.7	3.5	968
	MP&S	25X25	303.5	367.5	82.0	12.4	2.2	981
	MPSO	25X25	295.8	256.5	65.2	37.2	4.9	950
s35932 (1728)	MM	40X40	449.9	543.8	121.0	9.1	1.3	986
	MO[77]	40X40	449.9	437.2	97.3	19.6	3.0	971
	NSMO	40X40	400.2	380.6	80.7	26.7	1.2	970
	MP&S	26X26	387.3	459.7	103.0	13.5	1.8	981
	MPSO	26X26	350.1	349.4	73.5	31.0	3.5	960

Table 4.1: Comparison of the different mesh optimization approaches – absolute values of buffer area (BA), wirelength(WL), power (PWR) and frequency (F_{max}).

Mesh Optimization: Our network sensitivity based optimization (NSMO) yields consistently better results than the approach of [77] for identical starting mesh. On an average, our approach yields 14.25%, 8.69% and 7.78% extra reduction in buffer area, wire-length and power respectively with 0.01% improvement in F_{max}. This proves the effectiveness of our mesh optimization approach.

Mesh Planning: Our mesh planning and synthesis algorithm (MP&S) is effec-

Case (#Sinks)	Method	Size	BA %	WL %	PWR %	F_{max} %	CPU (s)
s9234 (135)	MM	9X9	0.0	0.0	0.0	NA	NA
	MO[77]	9X9	0.0	7.2	6.1	1.2	0.4
	NSMO	9X9	3.7	27.2	14.7	0.6	6.2
	MP&S	7X7	4.3	4.8	4.5	1.3	0.1
	MPSO	7X7	14.0	23.8	18.4	2.9	5.8
s5378 (165)	MM	10X10	0.0	0.0	0.0	NA	NA
	MO[77]	10X10	0.0	15.7	14.0	2.3	0.4
	NSMO	10X10	1.0	30.2	23.6	2.3	9.0
	MP&S	8X8	4.8	5.4	5.0	0.6	0.1
	MPSO	8X8	22.1	33.8	35.4	3.7	7.0
s13207 (500)	MM	30X30	0.0	0.0	0.0	NA	NA
	MO[77]	30X30	0.0	25.0	21.4	0.8	3.0
	NSMO	30X30	26.2	29.9	23.6	1.3	26.1
	MP&S	13X13	36.9	33.5	34.0	0.7	0.1
	MPSO	13X13	42.1	52.7	47.8	2.0	22.0
s15850 (566)	MM	30X30	0.0	0.0	0.0	NA	NA
	MO[77]	30X30	0.0	33.3	28.1	2.3	3.0
	NSMO	30X30	26.2	42.9	37.2	1.2	32.1
	MP&S	15X15	31.9	28.3	29.0	0.4	0.1
	MPSO	15X15	39.5	56.1	48.9	2.2	28.9
s38584 (1426)	MM	40X40	0.0	0.0	0.0	NA	NA
	MO[77]	40X40	0.0	24.0	21.9	0.7	8.6
	NSMO	40X40	10.1	30.0	23.9	1.8	100.2
	MP&S	25X25	20.4	19.2	19.2	0.5	0.2
	MPSO	25X25	22.4	43.6	35.8	3.7	81.6
s35932 (1728)	MM	40X40	0.0	0.0	0.0	NA	NA
	MO[77]	40X40	0.0	19.6	19.5	1.5	9.4
	NSMO	40X40	11.0	30.0	33.3	1.6	120.1
	MP&S	26X26	13.9	15.4	14.8	0.5	0.2
	MPSO	26X26	22.1	35.7	39.2	2.7	98.5

Table 4.2: Comparison of the different mesh optimization approaches – percentage value w.r.t. MM method.

Method	% BA Red	% WL Red	% PR Red	μ_{skew} Avg.	σ_{skew} Avg.	F_{max} MHz	% F_{max} Red
MO[77]	0.00	22.94	21.11	18.09	3.72	971.58	1.53
NSMO	14.25	31.63	28.89	21.35	2.60	971.65	1.52
MP&S	20.74	19.88	19.84	13.13	2.58	979.55	0.72
MPSO	26.92	42.53	39.80	30.66	4.38	958.05	2.90

Table 4.3: Summary of optimization results from Table 4.1 and 4.2 for all test cases.

tive in choosing a good initial mesh. In most cases, the quality of this initial mesh is close to the final, optimized results of [77]. Also, the size of the initial mesh obtained from our mesh planning approach is significantly smaller when compared to the manually obtained mesh of[77] for the bigger test-cases. This illustrates the importance of having a good methodology to obtain an initial mesh.

Combined Mesh Planning and Optimization: By performing our network sensitivity based optimizations on the mesh selected by our mesh planning algorithm (MPSO), we are able to achieve, on an average, 26.90% buffer area reduction, 19.59% wire-length reduction and 18.69% power reduction with less than 1.5% reduction in the worst case maximum frequency.

Run times: The runtimes for our mesh planning and synthesis algorithm are negligible even for our biggest test case. This can help in quick selection of a good mesh avoiding manual selection of initial mesh. Our mesh optimization approach has longer run time compared to the approach of [77]. However, since our optimization is to be done only on a good mesh obtained by the mesh planning algorithm, the overall time taken for getting an optimized mesh is considerably reduced because of the elimination of manual mesh size selection process.

Resources vs. Frequency trade-off: From Table 4.3, we see that the bigger the reduction in power, buffer area and wire-length, the higher is the skew degradation, which is expected. But what is noteworthy is that the degradation in skew is insignificant because it results in less than 3% reduction in F_{max} (w.r.t. the original mesh of [77]) while achieving significant reduction in power and area.

Case	Before EM Fix			After EM Fix			Difference			#itr
	$\#V_{em}$	%BA	%WA	$\#V_{em}$	%BA	%WA	$\%V_{em}$	%BA	%WA	
s9234	1	14.0	23.8	0	14.0	23.24	100	0	0.56	1
s5378	0	22.1	33.8	0	22.1	33.8	100	0	0.00	0
s13207	11	42.1	52.7	0	42.1	51.09	100	0	1.60	2
s15850	7	39.5	56.1	0	39.5	55.36	100	0	0.73	1
s38584	11	22.4	43.6	0	22.4	42.95	100	0	0.64	2
s35932	33	22.1	35.7	0	22.1	33.78	100	0	1.91	4

Table 4.4: Results of EM violation fixed by wire-sizing. BA and WA (Wire-Area) are % reduction w.r.t. MM row in Table 4.1 and 4.2. $\#V_{em}$ denote number of EM violations. #itr denotes the number of EM fixing iterations.

4.6.3 Results of Wire-sizing for EM

To demonstrate the effectiveness of our wire-sizing scheme for meeting EM requirements, we first obtain the target current density value. The existing literature [8, 39, 68, 69] has a wide range of values for the current density that can be used for copper interconnects. For example, the works of [69],[39],[68] and [8] recommend values of $90mA/\mu m^2$, $16mA/\mu m^2$, $8mA/\mu m^2$ and $160mA/\mu m^2$ respectively. In order to be conservative, we set 90% of the most aggressive value of $8mA/\mu m^2$ as our current density target. Thus, any mesh segment with a current density value above $7.2mA/\mu m^2$ is treated as a single EM violation. Thus, the goal is to ensure that no mesh segment in the final optimized mesh exceeds this current density requirement. Next, we use the methodology described in Section 4.4 to identify and fix the violations found in the optimized mesh from the MPSO rows of Table 4.1 and 4.2. The complete results for this procedure is shown in Table 4.4.

From the results of Table 4.4, we can see that our method is able to fix all EM violations with an average 1% increase in total wire-area. Please note that all the results in Table 4.1, 4.2 uses minimum wire-widths and the increase in wire-area in Table 4.4 is from the wire-sizing.

It may be noted here that we do attempt to resize mesh buffers after mesh segment sizing to ensure we do not overload the mesh buffers. However, in all our testcases, we found this to be unnecessary as the amount of extra loading on the mesh buffers was very small. As a result, there was no change in the buffer area even after fixing all the EM violations.

4.6.4 Results for Practical Issues in MeshWorks Usage

Next, we will discuss the testcase generation and experimental results related to the issues discussed in Section 4.5.

Blockages: In order to test if our mesh optimization works as expected on a testcase with blockage, we first created a testcase as described next. We picked a rectangular floorplan and randomly generated the sink locations in it. The center of the floorplan was assumed to have a rectangular blockage and so any random sinks located in the smaller square was removed. A complete 10X10 mesh was selected arbitrarily for this floorplan. Buffering & mesh optimization were done on this complete mesh without giving any explicit information about the presence of the blockage to them. The final mesh edges post mesh optimization are shown in Figure 4.14. Visual inspection on the Figure confirms that all mesh segments that were present in the blockage area have been optimized away. This demonstrates that our mesh optimization scheme can be directly applied in the presence of blockages.

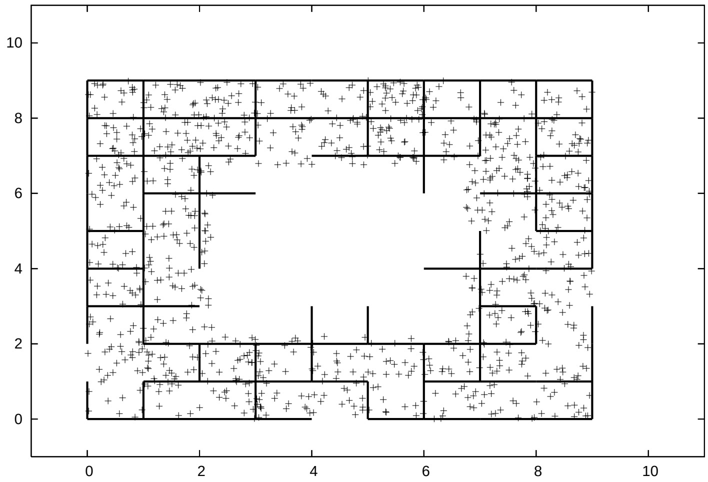


Figure 4.14: Mesh optimization result on a testcase with blockage.

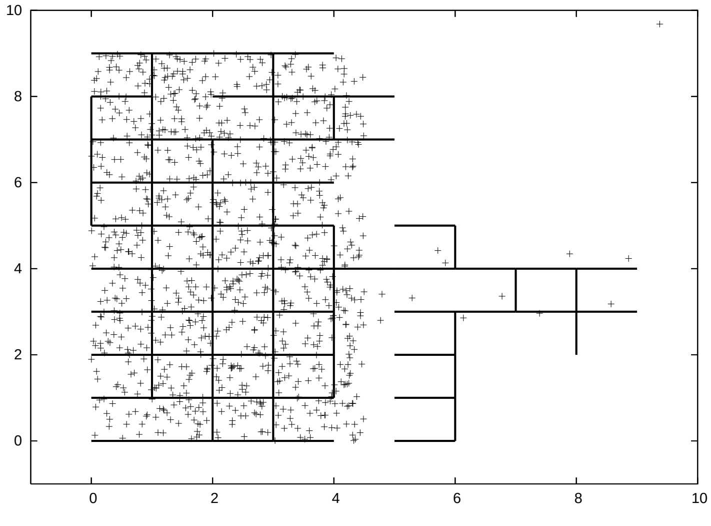


Figure 4.15: Mesh optimization done on a testcase with two clocks on same floorplan. Clock-A is shown.

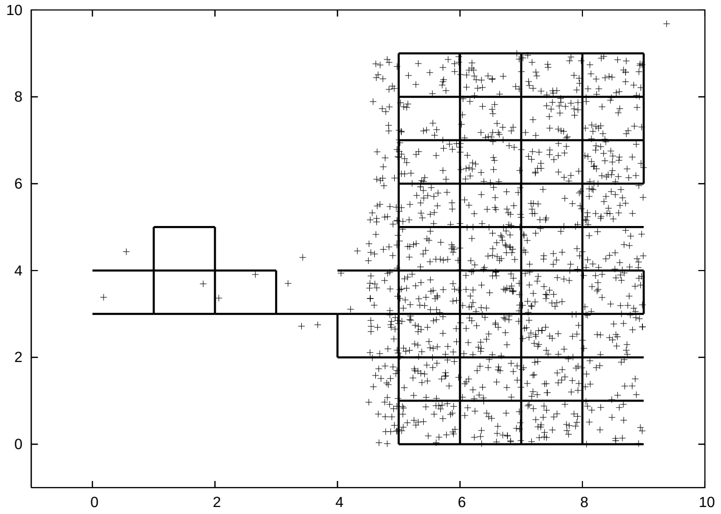


Figure 4.16: Mesh optimization done on a testcase with two clocks on same floorplan. Clock-B is shown.

Multiple Clocks: We generated our multi-clock testcase as follows. A rectangular floorplan was selected and clock sinks were randomly generated with constraints such that sinks on the left side of the square were assigned to Clock-A and the sinks on the right side of the square were assigned to Clock-B. Exceptions to this rule were allowed with a small probability when the Y-coordinate of the sinks was between a selected band (between 30% and 50% of the maximum Y distance). This setup enabled us to get a floorplan such that majority of sinks belonging to Clock-A were located on the left-side with a few on the right side of the block and vice-versa. This imitates the conditions in many ASIC designs where registers belonging multiple clocks interact.

Given this testcase, two sets of buffering and mesh-optimization were done on a 10X10 mesh, one for Clock-A and another for Clock-B. The final mesh edges after mesh optimization for these clocks are shown in Figures 4.16 and 4.15. As expected, the mesh for Clock-A has most of the mesh segments on the right side removed, except for the segments attached to the few Clock-A sinks on the right side and vice-versa for Clock-B. This demonstrates that our mesh optimization can reduce resource utilizations on multi-clock floorplans even if the initial mesh covers the entire floorplan.

Highly Uneven Load Distribution: As the results from the previous experiment demonstrated, our mesh optimization scheme works as expected even when the clock sinks are distributed to one side of the chip predominantly over the other. This can be directly inferred from both Figures 4.16 and 4.15 considering as individual testcases. Thus, our mesh optimization scheme can be used even in cases with highly uneven distribution of clock sinks.

Chapter 5

Robust Multi-Corner Chip-level CTS

5.1 Chip-level Clock Tree Synthesis Problem

A System-on-a-Chip (SOC) design can be defined as "an IC, designed by stitching together multiple stand-alone VLSI designs to provide full functionality for an application" [61]. In today's 65nm/45nm VLSI technologies, SOC designs have become increasingly common and the trend is expected to continue in the future [32]. An attractive feature of SOC designs is the ability to reuse a given sub-component in multiple chips. The level of reuse can be different from IP[1] to IP. At one extreme of the reuse spectrum are hard-IPs where the exact transistor-level layout is reused in several designs. At the other end are the soft-IPs which go through the physical design/timing closure process from scratch so as to integrate the sub-block with the rest of the chip.

Most SOC physical design closure is done in a hierarchical fashion [61]. In such a methodology, the chip consists of several logical and physical partitions that are timing closed independently [1, 2, 32, 61]. This trend is only expected to accelerate as SOC chips become bigger and more functionality gets added to them. To complete the full chip, different sub-blocks should be integrated along with the glue logic and chip-level timing closure should be done. This

---

[1] We use the word IP or sub-block to denote the individual sub-component used in SOC designs. They are also referred to as *core* in some literature [61].

chip-level timing closure includes the chip-level CTS (CCTS) step in which a chip-level clock tree is synthesized to drive all the block-level clock trees. CCTS is driven by two main objectives. The primary objective is that the full clock tree, which includes the chip-level and all the block-level clock trees, should be balanced and have less skew. This requirement is an absolute necessity in SOC design as it avoids data mismatch as well as the use of data lock-up latches [61]. Satisfying this requirement is relatively easy when considering only the nominal delay corner. However, timing closure in most practical chips involve verifying timing across several corners (referred to as design corners) that represent several global variation effects such as fab-to-fab, wafer-to-wafer, die-to-die variation, global voltage and temperature variations [1, 2, 61]. This implies that the clock trees should also be balanced with small skews across all the design corners. The different sub-blocks of an SOC typically include several hard IPs as well as soft-IPs that are timing-closed independently by different individuals/teams, possibly using different methodologies, tools, and library cells. In such cases, achieving good skews for the entire clock tree of the chip across all the design corners is a very challenging task. This is primarily because of the possible difference in the way the delays and skews of the different sub-clock-trees scale, either because of difference in the clock structures or the relative significance of cell and interconnect delays.

Another important objective for chip-level CTS is to minimize the clock divergence (see Section 5.2.1 for detailed explanation) for the IPs that interact significantly with each other. This helps to minimize the maximum possible skew variation between the critical timing paths between the IPs and thus improves the overall yield. The clock divergence reduction also helps in faster timing closure in real designs as most clock tree analysis algorithms [83] consider the fact that process variations in common part of the clock tree do not affect the skew between a given register pair. Reducing the clock divergence between IPs can be a trivial problem when either the number of IPs are very small or when they do not interact significantly. Unfortunately, both these conditions do not apply to the SOC designs of today which have a significant number of IPs which interact in a complex way [32, 61]. Thus, clock divergence reduction between interacting IPs is a difficult problem to solve in today's SOC designs.

In many complex chips, CCTS work is completely custom/manual [1, 2] so as to achieve the precise skew and divergence objectives. Though a custom tree has the advantage of achieving the exact results intended, it is often very time consuming. Also, as the complexity and size of SOC designs increase, custom/manual chip-level CTS will become increasingly difficult. Thus, fully automated methods to address the CCTS problem in today's SOC designs are needed. In this work, we attempt to address the CCTS problem. The key contributions of our work are:

- A 0-1 Quadratic Programming based clock pin relocation scheme for soft-IPs to reduce chip-level clock divergence.
- An effective method to reduce the chip-level clock tree skews simultaneously across different PVT corners.
- A dynamic programming based CCTS algorithm that simultaneously reduces clock divergence and multi-corner skew.

To our best knowledge, the above contributions make the first comprehensive solution to the CCTS problem for complex SOC designs. Experimental results on several test-cases indicate that our methods achieve 10%-31% (20% on average) reduction in the clock path divergence compared to existing CTS works. This clock divergence reduction directly translates to an increase in timing yield (for a given chip) or a reduction in chip power and area (for a given target yield). Our methods achieve these significant clock divergence reduction with as little as 0.5% increase in overall clock buffer area and wire-length.

The rest of the chapter is organized as follows: in the next section, we present the problem definition and also give some simple examples as to why the chip-level CTS problem is complicated compared to a CTS on a flat block. The main algorithms of our work is presented in Sections 5.3, 5.4 and 5.5. In Section 5.6, we discuss a few practical issues related to the application of our CCTS algorithms. Finally, Section 5.7 describes the experimental setup that we have used to verify the effectiveness of our approach.

5.2 Motivation and Problem Formulation

In this section, we will first discuss the significance of clock divergence, the effect of clock pin assignment on clock divergence and multi-corner skew reduction using a few simple examples after which we will formulate the chip-level CTS problem.

Figure 5.1 shows a simple example of a chip-level CTS problem. The sub-blocks shown might be either hard-IPs or soft-IPs. In the case of hard IPs, the clock pin location and the clock tree itself will be fixed. For soft-IPs, CTS will be done as a separate step along with block-level timing closure and then integrated at the chip-level.

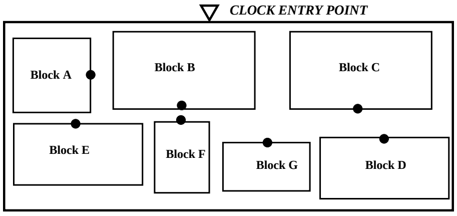


Figure 5.1: A simple chip-level CTS example. The black circles represent the clock root for each sub-block.

5.2.1 Significance of Clock Divergence Reduction

The significance of reducing clock divergence between registers in *timing-critical* paths is well known [21]. For a given overall delay, the lesser the divergent delay between the such register-pairs, the lesser is the value of maximum skew that can be seen between them. This is because any variation in the common clock path will not impact the skew between the register pair. This is illustrated in Figure 5.2. In this example, assuming all other conditions are same, Case A is better for timing yield in the presence of variation because skew variation in Case A is limited only to the variations in last clock net. However, in case B, since the last buffer is not shared, the magnitude of possible skew variations increases, thereby impacting the timing yield in the presence of variations.

Significance of clock divergence reduction in CCTS: The same principle of clock divergence reduction discussed above is also applicable at the chip-level where different sub-blocks interact with each other instead of register pairs. In some cases, clock divergence reduction between specific sub-blocks might

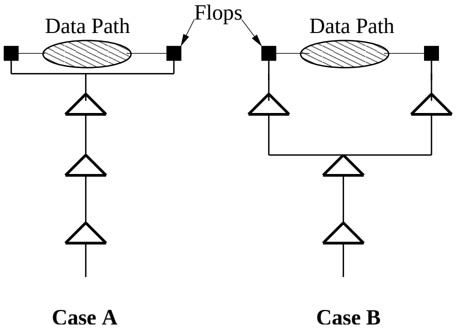


Figure 5.2: Even for identical nominal skews, Case A is better than Case B because of lesser clock divergence and hence lesser skew variation.

be extremely important to ensure good timing yield. For example, when the clock tree divergence between two heavily interacting IPs is high, it might result in significant skew variation between all the register pairs between the IPs. If some of these register pairs were already timing-critical, the increased skew variation will only exacerbate the situation, thereby affecting the timing yield.

5.2.2 Impact of Sub-block Clock Pin Location on Clock Divergence

Unlike hard IPs, the clock pins of the soft-IPs can be changed specific to a given chip and floorplan. This additional flexibility for the soft-IPs can be effectively used towards clock divergence reduction between critical IPs. Figure 5.3 shows a simple example where the clock pin assignment might make a difference in clock divergence reducing. In this example, blocks A and B are assumed to have critical paths between them. Thus, the pin assignment in Case B is better since it reduces the clock divergence (and hence the maximum clock skew under variation) between the flops in the critical path.

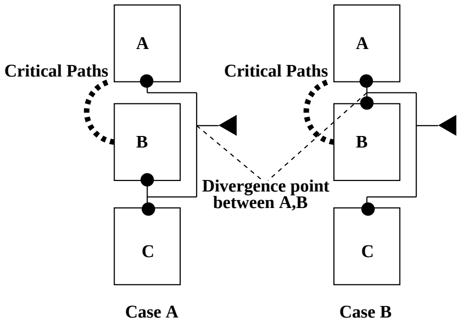


Figure 5.3: Importance of clock pin assignment for sub-blocks. Case A and Case B differ in the clock pin location for block B, which affects CTS. If blocks A and B have critical paths between them, Case B will result in better yield because of reduced clock divergence between A and B.

5.2.3 Multi-corner skew reduction problem

Once the clock pin location for the soft-IPs are determined and CTS is done on all sub-blocks, the next step is chip-level CTS. In the chip-level CTS problem, each of the sub-trees shown in Figure 5.1 are assumed to have full clock trees in them with fixed clock input pins. In addition, we also know the delay/skew of each of the clock trees across all the PVT corners. This information will be necessary for balancing the chip-level clock tree across all PVT corners. To understand the difficulty in reducing the skews at the chip-level across multiple design corners, consider Figure 5.4 where only two sub-blocks are present. The squares in the sub-blocks represent clock sinks. The left-side block has bigger buffers with longer interconnects and the right-side block has smaller buffers with shorter interconnect. Let us assume that both sub-clock-trees have identical delays in the nominal corner. However, their delays across different design corners will be different, mainly because of the difference in the interconnect lengths and buffer sizes. To balance these two sub-clock-trees across all corners, the chip-level clock tree should be built such that the differences in the delays, *across all corners*, between the sub-clock-trees gets exactly (or nearly) compensated at the chip-level. In our example, we can attempt to do this by driving the left-side block with small buffers and short interconnect and the right-side block with bigger buffer and longer interconnect as shown in Figure 5.4[2]. However, even in this case, non-zero skew across corners will still exist unless the structure is completely symmetric, which is rarely possible in real designs. In most SOC designs, there will be several sub-blocks having clock trees with significant differences in their size, structure, buffer sizes used and interconnect lengths. Thus, synthesizing a chip-level clock tree that can simultaneously reduce the skew across all corners by accounting for these differences while not significantly increasing the overall delay is a challenging problem.

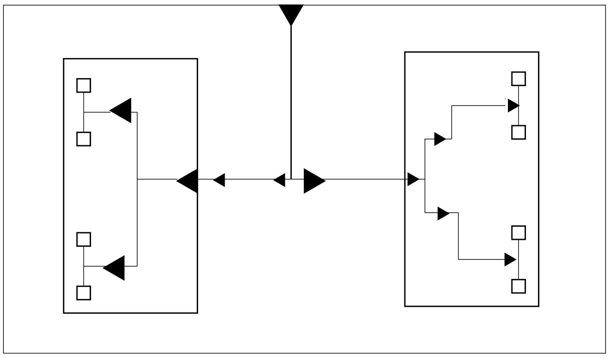


Figure 5.4: Simple example illustrating difficulty of balancing two different IPs. The clock tree delays of the two blocks will scale differently across different corners due to different buffer sizes and interconnect lengths.

---

[2] Please note that the figure is illustrative in nature and assumes that all buffers have load capacitance according to their drive strengths.

Problem Formulation: We formulate the overall CCTS problem into the following two sub-problems:

1. ***Given:*** Chip-level floorplan and criticality of clock divergence between all block pairs.

   Problem: Select the clock pin locations of all soft-IPs to reduce clock divergence between critical IP pairs.

2. ***Given:*** All information from the previous step and also information on sub-clock-tree delays/skews across all corners for each sub-block.

   Problem: Obtain a chip-level clock tree such that the skews and delays across all corners are reduced, while simultaneously reducing the weighted sum of clock divergence between all the IP pairs. The value of weight for a given IP pair is directly obtained from the number and timing criticality of paths between them. In general, the more paths a given IP pair and the higher the criticality of those paths, the higher the value of the weight for that pair.

***Trade-off between divergence reduction and delay reduction*:** In some cases, we might be able to achieve lesser clock divergence by increasing the overall delay of the clock tree and vice-versa. One simple way to quantify this Trade-off is to use a scaling factor that will determine the percentage of delay increase that can be tolerated for a given reduction in clock divergence. Using this scaling factor, we can define the overall cost as follows:

$$Cost = x * Max\_Delay + (1 - x) * DIV\_COST, \qquad (5.1)$$

where $DIV\_COST = \sum_{\forall i,j} W_{i,j} * \left({D_i}^F + {D_j}^F - 2 * {D_{i,j}}^C \right)$;

In the above equations,

- x: variable with value between 0 to 1 to quantify delay and divergence Trade-off.
- Max_Delay: maximum delay to any sink in the entire clock tree.
- DIV_COST: Clock divergence cost between all IPs pairs.
- i,j: The block numbers, with $1 \leq i, j \leq N, i \neq j$;.
- $W_{i,j}$: Criticality of clock divergence between blocks i,j.
- ${D_i}^F$: Avg. delay from clock root to the flops in block i.
- ${D_{i,j}}^C$: The maximum *shared or common* delay between any two block pair i, j.
- All the delay information are w.r.t. the nominal corner values.

Thus, the objective of the CCTS problem is to get a chip-level clock tree that can minimize the above cost function while simultaneously reducing the skews across all corners.

5.3 Clock Pin Assignment Algorithm for Clock Divergence Reduction

Given a floorplan and criticality of clock divergence between all sub-block pairs, the clock pin assignment aims to identify the location of all the clock pins of each soft-IP *even before any CTS is done on them* [3]. We restrict the possible clock pin locations to the mid points of one of the four sides of each block. This minimizes the distance between the clock pin and the farthest register and can result in reduced clock tree delay. When the flop distribution is not uniform within a given block or when there are multiple clocks present in a given block, we locate each clock pin such that it divides the sink distribution it drives into roughly two equal halves, either in the horizontal or vertical direction. Under this assumption, clock pin assignment problem can be formulated as follows:

Let B_i denote the sub-block number i where $1 \leq i \leq N$. Let $W_{i,j}$ denote the criticality of the paths between blocks i and j. Also, let all the four possible clock pin assignments for a given block be denoted by ${B_i}^1$, ${B_i}^2$, ${B_i}^3$, ${B_i}^4$ where the pin locations are numbered starting from bottom point and proceeds in the clockwise direction. Let the pin selection for a given block be denoted by a set of four variables: ${x_i}^1$, ${x_i}^2$, ${x_i}^3$, ${x_i}^4$. Each of these variables can be either 0 or a 1 and the sum of the four will always be 1 to make sure exactly one of the four locations are selected. Using these definitions, the problem of clock pin assignment for clock divergence reduction can be formulated as:

$$\begin{aligned} Minimize &: \sum {x_i}^p * {x_j}^q * W_{i,j} * Top\_Level\_Dist({B_i}^p, {B_j}^q) \\ s.t &: \sum {x_i}^p = 1, \quad {x_i}^p \in \{0,1\} \\ where &: 1 \leq i,j \leq N, i \neq j; \quad 1 \leq p \leq 4; \quad 1 \leq q \leq 4; \end{aligned} \tag{5.2}$$

In the above equations,

---

[3] This step needs to be done during the floor-planning stage of the chip design and before the timing closure of the individual sub-blocks starts

- i and j denote block numbers.
- p and q denote one of the four pin locations on a given block.
- $Top\_Level\_Dist({B_i}^p, {B_j}^q)$ represents the Manhattan distance between pin location p of B_i and q of B_j.

The conditions that each of the variables ${x_i}^p$ should be either 0 or 1 and that the sum of all the variables for a given block should exactly be 1 makes sure that exactly one pin location is selected for each block. The cost function being minimized is the weighted sum of distances between all the clock pins of all block pairs where the weight is the criticality of the paths between a given block pair. Minimizing the distance between two pins will directly increase the chances of clock delay sharing between the two blocks. The only variables in the above optimization problem are ${x_i}^p$ and since they can only take values of either 0 or 1, the above problem is a 0-1 Quadratic Programming problem. Though this problem is NP-hard, efficient heuristics are available to solve this problem [28].

Impact of pin assignment on block-level delay: The above formulation ignores the impact of clock pin assignment on the block-level clock tree delays. For example, if all the registers are concentrated on one side of the sub-block[4], then having the clock pin on that side will reduce the block-level clock tree delay. Since the above formulation does not consider this effect, it might end up increasing the overall delay or even clock divergence. However, the formulation can be made to account for block-level clock tree delays by introducing an

---

[4] This can happen in real designs when the block has multiple input clocks.

additional weighting term of the form ${K_i}^p$ that denotes the criticality of assigning the pin location p for block i with regards to the block-level clock tree. For example, if all four sides are equally acceptable for the block-level CTS of block i, then the value of ${K_i}^p$ will be identical for all four values of p. If on the other hand, we want to make a particular pin location more likely, we can increase the corresponding scaling factor. The relative values for these factors may be obtained by a weighted sum of distances of all the registers from each of the four pin locations.

5.4 Multi-corner Skew Reduction Algorithm

In this section, we will address the problem of merging any two sub-clock-trees such that their combined skews across all the corners are reduced. This problem can be divided into two categories. In the first, the clock pins are located very close to each other *and* their delays across all corners are very similar. In this case, the multi-corner skew balancing is trivial since it is possible to merge the clock pins with just interconnect without adding an extra buffer level. In the second case, the clock pins are far apart *and/or* they have significantly different delays across the corners. In such situations, we need to add one or more single-fanout [5] buffer stages (with appropriate buffer sizes/interconnect lengths) to the root of the sub-trees to reduce their combined skews across corners. In summary, to reduce the multi-corner skew between any two sub-trees for the non-trivial situation, we need a method to select the appropriate number of buffer stages and the size/lengths of the buffers/interconnects to be used to merge the clock pins

---

[5] Single-fanout because we are merging only the two sub-trees. A two fanout buffer will mean a successful merger!

of the two sub-blocks. In future discussions, we call the selection of appropriate buffer size/interconnect length as selection of an appropriate *buffer configuration*. Figure 5.5 shows an example of a *buffer configuration*.

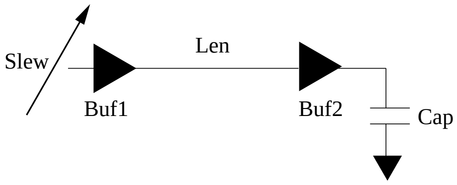


Figure 5.5: Buffer configuration used for multi-corner delay characterization.

Special Properties of CCTS Problem: To solve the problem of picking the right buffer configurations for multi-corner skew reduction, following special properties of CCTS problem can be exploited:

- Unlike CTS on a flat design, the CCTS problem will have just a hand full of end points (clock pins of sub-blocks) that are much more spread apart in distance than typical registers. This is because the number of sub-blocks in a typical SOC will be orders of magnitude lesser than the number of flops in the whole design.
- As a result, the typical fanout for a buffers in the chip-level clock tree will be considerably less compared to the block-level clock trees. In most practical cases, this can be as low as 1 or 2.

Steps to Choose Buffer Configurations for Multi-Corner Skew Reduction: In order to distinguish between the different buffer configurations and select the right set of configurations to achieve multi-corner skew reduction, we can follow the following steps:

- Restrict the maximum fanout for any chip-level clock buffer to just 1 or 2. The clock power/area penalty due to this restriction will be negligible because the fanout of most buffers is expected to be small anyway. Also, the number of chip-level clock buffers will be small compared to the total number of buffers of all the sub-blocks combined.

- The fanout restriction drastically reduces the number of possible buffer configurations, enabling us to do the multi-corner delay characterization for each configuration quite easily. For example, in Figure 5.5, the input slew (in 5ps increments), buffer type of the driver, interconnect lengths (in 25um increments) and the load buffer type are the variables. Since this is a simple circuit, the complete multi-corner delay characterization of all possible configurations *and* across all corners typically takes just a few minutes. This is similar to the typical cell delay characterizations used in ASIC designs, with the added explicit variables of interconnect length and load cell being driven.

- The next step is to get what we define as *cross-corner delay ratios* (CCDR) for every buffer configuration as described here. For each buffer configuration, we normalize (divide) the delays across all N corners with the nominal-corner delay of that configuration. After the normalization, each buffer configuration will have a *vector* of N numbers, corresponding to N corners, called its CCDR. Obviously, the ratio number corresponding to the nominal corner will always be 1. This normalization helps us to compare the *relative* cross-corner scaling of different buffer configurations and choose the appropriate one for merging any given sub-tree pair. For example, if a buffer driving a 500um interconnect has delays of 50, 100 and 200 ps in the fast, nominal and slow corners respectively, then the delay ratios for this configuration will be (0.5, 1.0, 2.0). If another buffer driving a 300um load has a delay ratio of (0.4, 1.0, 1.8) in the fast, nom and slow corners, then we can conclude that the second configuration relatively speeds up the fast and slow corners than the first configuration.

The concept of CCDR described above is used in our multi-corner sub-tree merging heuristic shown in Figure 5.6. The basic idea behind this heuristic can be explained by a simple example. Lets A and B be two sub-trees that we want to balance across three corners - fast, nominal and slow. Let the delays for the two blocks in the three corners be A(50,100,200) and B(40,100,220) respectively[6]. If the two clock trees are merged using a *zero-nominal-skew* chip-level clock tree, then the merged tree will have zero skew at nominal corner, but higher skews at the fast and slow corners. In order to achieve good skews across all three corners, we should build the chip-level tree such that $del\_to(A, nom) \simeq del\_to(B, nom)$ and $del\_to(A, fast) < del\_to(B, fast)$ and $del\_to(A, slow) > del\_to(B, slow)$, where $del\_to(A, nom)$ etc. represent the chip-level clock-tree delay to the clock pin of A in the nominal corner. Chip-level clock trees with such precise cross-corner delay scaling requirements can be constructed by selecting the buffer configurations with appropriate CCDR. This is the key idea behind our multi-corner sub-tree balancing heuristic shown in Figure 5.6.

---

[6] Such differences in delay scaling across corners can happen when different clock buffer types, CTS tools/methodologies are used in the two blocks.

```
Procedure: $Multi\_Corner\_Subtree\_Balance(S_A, S_B)$
Input: Location and delay information for both sub-trees $S_A$ & $S_B$.
Output: New sub-tree $S_C$ combining $S_A$ & $S_B$.
1. Get CCDRs of $S_A$, $S_B$ w.r.t. nominal corner.
2. Set $Sub\_trees\_not\_close = 1$
3. While ( $Sub\_trees\_not\_close == 1$ )
   (i) Pick sub-tree with min nom-corner delay, denoted by $S_P$.
       Let $S_Q$ be the sub-tree with max nom-corner delay.
   (ii) Select the best buffer config. to be added to $S_P$ such that the $CCDR$
       of $S_P$ after adding the buffer config. moves closest to the
       CCDR value of the max-nom delay sub-tree without exceeding its delay
       significantly(i.e. by more than biggest buffer delay). Since CCDRs are
       vectors, the closeness is defined by $L^2$ norm between them.
   (iii) Update delay and CCDR information for $S_P$.
   (iv) If skew across corners between sub-trees less than limit
           $Sub\_trees\_not\_close = 0$
4. Using selected buffer configurations, identify the Manhattan ring
   within which each sub-tree's new root can be located.
5. If (Two Manhattan rings intersect)
     Any point within intersecting area can be the new root.
     Do wire-snaking to ensure preservation of skews/delays.
  Else
     Select closest points on the rings to reduce wire-length.
     Merge them by constructing a simple symmetric (0 skew) tree.
6. Name the new sub-tree as $S_C$ and return $S_C$.
```

Figure 5.6: Multi-corner skew balancing heuristic.

In the above procedure, we first pick the sub-tree, denoted by S_P, with lesser nominal corner delay and recursively add buffer configurations at its root such that the CCDR of the new sub-tree moves closer to the other sub-tree. This process is repeated till the delays of both the sub-trees are fairly close to each other across all corners. At this point, the exact configurations to be added at the roots of both sub-trees A and B to minimize their cross-corner skew are available. However, the location of the merging point of the two sub-trees is still not yet fixed. For a given sub-tree, the total lengths of all the interconnects added with buffer configuration gives the radius of the Manhattan ring within which its root pin is to be located. If the Manhattan rings of both sub-trees intersect, then any point within the intersection can be selected as the root with appropriate wire-snaking. If the Manhattan rings do not overlap, it means that though the two sub-trees have similar delays, we need to add more buffer levels to physically merge them. To achieve this, we identify the closest points/segments on the two Manhattan rings and merge them with a perfectly symmetric tree. This will ensure that the multi-corner skew balancing already completed between the two sub-trees is not affected. It may be added here that exact location of the parent node after merger can be *deferred* in the same manner as in the DME algorithm.

It shall be noted that the above multi-corner sub-tree balancing procedure inherently assumes the following:

- The skew targets across corners are bigger than at least a buffer delay across corners. Otherwise, the skew condition in line 3-(iv) of the algorithm will never be met and the loop will go on indefinitely.
- All the buffer sizes used at the block-level CTS are available for use at

the chip-level CTS. Otherwise, there might be some buffer sizes that scale differently from others across corners which can not be compensated at the chip-level.

It may be noted here that the above procedure is suitable only in the limited context of chip-level CTS and is inefficient in terms of buffer resources for CTS on a sea-of-gates design. Since the number of sub-blocks will be several orders of magnitude less than the number of flops in the design, the chip-level CTS can afford to adopt the above approach.

5.5 Chip-Level CTS Algorithms

In this section, we discuss four different chip-level CTS algorithms with varying degrees of complexity. Please note that only the *dynamic programming based algorithm* is newly proposed in this work. The other three algorithms are simple modifications of existing CTS works used for comparison. We discuss all four CCTS algorithms here for the sake of completeness. Please note that except for the first algorithm (Single-Corner DME) the other three algorithms use the multi-corner skew balancing method of Figure 5.6. Also, the pin-assignment scheme proposed in Section 5.3 may or may not be used with the following CCTS algorithms.

5.5.1 Single-Corner DME based Approach

This algorithm is a direct application of existing CTS algorithms to the CCTS problem in which only nominal corner delays are used. The algorithm recursively merges sub-tree nodes which are the nearest neighbors in a manner similar to that of well known CTS algorithms [10, 17, 20, 75]. If a given node cannot be merged with any other sub-tree without violating the slew limits, a buffer is added on top of the node to extend the possible merging region for the sub-tree. The buffer sizes for merging two sub-trees are chosen in such a manner to reduce the total amount of interconnect added. For example, if a given delay can be achieved using buffers of different sizes and interconnect lengths, then the option that uses the minimum interconnect length is chosen. This approach simultaneously reduces both the wire-length and clock buffer area. *The results from this approach will be used as the baseline for rest of the algorithms.*

5.5.2 Multi-Corner DME based Approach

This approach is identical to the single-corner approach with one key difference: the consideration of multi-corner skews. During the process of merging two sub-trees, the method described in Figure 5.6 is used instead of using only the nominal corner delay. At each step, the sub-trees that are closest to each other are merged recursively till only one node remains. *The results from this approach will be used to do the cost Vs. benefit analysis of multi-corner skew reduction.*

5.5.3 Greedy CCTS Algorithm

This algorithm is a simple modification of the work of [11] in which every sub-tree merger is done to minimize the cost (wire-length/buffer area) of that merger. In our modification, the merging cost as defined by Equation(5.1) instead of wire-length. During each iteration, the merging cost of all possible pairs are evaluated and only the best pair is selected for the actual merger. The selected pair is then merged using the multi-corner skew reduction method of Figure 5.6. This is done repeatedly till all the different sub-trees are merged. This is a classic greedy approach to reduce both clock tree delay and divergence. As with most greedy approaches, this method might not result in the best tree possible. However, it is typically fast. *Since [11] is one of the best algorithms for prescribed-skew CTS, the results from this approach will help us determine if existing prescribed skew (useful skew) CTS algorithms can be modified for solving the CCTS problem.*

5.5.4 Dynamic programming Based CCTS Algorithm

Our dynamic programming based CCTS algorithm, shown in Figure 5.7, follows the same general outline of typical dynamic programming solutions[18]. As with any dynamic programming based approach, two key aspects of our algorithm are optimal solution of sub-problems and effective pruning of inferior sub-solutions to avoid exponential run-time.

For subsequent discussions, we use the following terminologies. An *active sub-tree* is one that has not yet been eliminated/pruned from subsequent merging operations. The list of active sub-trees represent the current list of sub-solutions to CCTS problem. A *new* sub-tree in the list of active sub-trees is one that has not gone through even a single round of mergers with other active sub-trees.

During initialization phase shown in step 1 of Figure 5.7, all the clock pins of sub-blocks are marked as *new* and *active sub-trees.* Step 2 of Figure 5.7 is the core part of our algorithm in which we iteratively combine existing sub-trees to progressively get bigger sub-trees, eventually getting one or more solutions that

```
Procedure: Dynamic_Programming_Top_Level_CTS
Input: Location and delay information for all blocks.
Output: Chip-level clock tree with min delay & clock divergence.
1. Initialize
  a. Mergers_Completed = 0
  b. Active_SubTrees = Clock Pins of all blocks
  c. For each subtree ∈ Active_SubTrees
       Status(subtree) = new.
2. While ( Mergers_Completed == 0 )
  a. Valid_Pairs = Pick_Valid_Pairs(Active_SubTrees)
  b. Eliminated_Pairs = Pre_Eliminate(Valid_Pairs)
  c. Valid_Pairs = Valid_Pairs - Eliminated_Pairs
  d. Generate_Cost for merger of each Valid_Pairs using Eq.(5.1)
  e. Potential_SubTrees = Active_SubTrees + Valid_Pairs
  f. Eliminated_SubTrees = Post_Eliminate(Potential_SubTrees)
  g. New_Additions = Potential_SubTrees - Eliminated_SubTrees
                     - Active_SubTrees
  h. For each subtree ∈ Active_SubTrees
       Status(subtree) = old.
  i. For each subtree ∈ New_Additions
       Status(subtree) = new.
  j. Active_SubTrees = Potential_SubTrees - Eliminated_SubTrees
  k. if (No. of New_Additions == 0)
       Mergers_Completed = 1
3. Pick sub-tree in Active_SubTrees with all
blocks and min delay and clock divergence.
```

Figure 5.7: Dynamic Programming based approach to chip-level CTS. The sub-steps are explained separately in Figures 5.8, 5.9, 5.10.

drive all target clock pins. In each iteration of step 2, we pick the set of valid sub-tree pairs that can be merged to create new sub-trees that have not been considered so far. It is essential from run-time perspective that we consider only the new sub-tree pairs that we have not yet eliminated or created. The steps to ensure this are outlined in Figure 5.8. Essentially, we mark sub-trees that have gone through one round of merging with each other as *old* as in step 2-h in Figure 5.7. As a result, any merger between two *old* nodes is invalid as it would have happened in one of the previous iterations. Also, any merger between two solutions that have overlapping list of target clock pins is also invalid as it is physically infeasible. For example, if the clock pin of a particular block A is present in two sub-solutions, merging them will mean that the same pin should be merged first with two nodes at the same time. So any such merger is invalid. Thus, the procedure of Figure 5.8 returns the complete list of *valid* new sub-trees pairs that can be considered for merging.

```
Procedure: $Pick\_Valid\_Pairs(Active\_SubTrees)$
Input: All active sub-trees.
Output: All pairs of sub-trees that are valid for merger.
1. Valid_Pairs = {}; Number sub-trees from 1 to $N$.
2. For $i = 1$ to $N$
     For $j = i$ to $N$
       Sub-tree pair considered: $S_i$ and $S_j$
       If (Status($S_i$) == new OR Status($S_j$) == new ) AND
       If (No overlap between $S_i$ and $S_j$ on block clock-pins driven)
         Valid_Pairs → Valid_Pairs + $(S_i, S_j)$
3. Return Valid_Pairs
```

Figure 5.8: Procedure to pick valid pairs for merger from a given set of sub-trees.

Once the full list of valid sub-tree pairs is available, the pre-elimination procedure shown in Figure 5.9 weeds out sub-trees that can be safely removed from consideration even before actually merging them. For example, if roots of two sub-trees are located very far away from each other or when their delays differ significantly, then it is probably a good choice to eliminate the pair because merging them will cause a significant increase in overall delay of the clock tree. A caveat is that a subtree-pair cannot be eliminated if there are no other alternative merging for the two subtrees involved.

```
Procedure: $Pre\_Eliminate(Valid\_Pairs)$
Input: All Valid Pairs of sub-trees.
Output: All very bad merging choices.
1. Eliminated_Pairs = {}. Number Valid_Pairs 1 to $V$.
2. For $i = 1$ to $V$
     Let $S_A$ and $S_B$ be sub-trees of the pair $i$.
     If (dist($S_A, S_B$) > $Dist\_Threshold$) OR
     If (delay_diff($S_A, S_B$) > $Delay\_Threshold$)
       Mark Pair $i$ for potential elimination.
3. For each Pair $i$ marked for potential elimination
     Let $S_A$ and $S_B$ be sub-trees of the pair $i$.
     If ($S_A$ and $S_B$ have merging pairs not marked for elimination)
       Eliminated_Pairs → Eliminated_Pairs + Pair $i$
3. Return Eliminated_Pairs
```

Figure 5.9: Pre-eliminate procedure to eliminate very bad merging choices.

After the pre-elimination step, the procedure in Figure 5.7 updates the list of valid pairs by removing the eliminated pairs. Next, each of the eligible sub-tree pairs are merged using the multi-corner sub-tree balancing algorithm of Figure 5.6 at the end of which, each sub-tree will have a specific cost as defined by Equation(5.1). The clock pins not driven by a given sub-tree is ignored for the purpose of cost evaluation and the sub-tree root is treated as the clock root. This is because only sub-trees that drive the same set (or a full sub-set) of clock pins are compared in their cost for the purpose of subsequent elimination.

Two sub-trees that drive different sets of clock pins will never be directly compared for elimination and so they having different roots for the cost evaluation does not matter. After the cost generation step is complete, a new set, called *Potential_SubTrees*, is created by merging the existing active sub-trees and the valid pairs for which merging cost is available. The post elimination step done on this set of sub-trees is explained in Figure 5.10.

Procedure: *Post_Eliminate*(*Potential_SubTrees*)

Input: Full list of potentially valid subtrees.

Output: Sub-treesPairs that can be pruned because of existence of other dominating sub-trees.

```
1. Eliminated_SubTrees = {};
   Number Potential_SubTrees sub-trees from 1 to $N$.
2. For $i = 1$ to $N$
     For $j = i$ to $N$
       P $\rightarrow$ List of all sub-block clock pins driven by sub-tree $i$.
       Q $\rightarrow$ List of all sub-block clock pins driven by sub-tree $j$.
       If (P $\subseteq$ Q AND cost(P) $\geq$ cost(Q))
         Eliminated_SubTrees $\rightarrow$ Eliminated_SubTrees + P;
       If (Q $\subset$ P AND cost(Q) $>$ cost(P))
         Eliminated_SubTrees $\rightarrow$ Eliminated_SubTrees + Q;
3. Return Eliminated_SubTrees
```

Figure 5.10: Post-eliminate procedure used to eliminate dominated sub-trees.

In the post elimination procedure of Figure 5.10, all the existing sub-trees are compared with each other to check for inferior solution. A sub-tree P is inferior if there exist another sub-tree Q that covers the same set (or a super-set) of clock pins covered by sub-tree P, but has same or lower merging cost. Once the inferior solutions are identified, they are removed from the list of active sub-trees that will be considered for the next round of sub-tree mergers. This is similar to the classic pruning condition used in dynamic programming approaches. After the post elimination step, the next step is to mark the existing active sub-trees as *old* (Step 2-h in Figure 5.7). This is because all possible valid sub-tree pairs from these have been evaluated already and they should not be repeated again. However, the newly created sub-trees should be allowed to merge, with each other and also with old sub-trees. So the newly created sub-trees are marked as *new* (Step 2-i in Figure 5.7). The termination of the iterations in Figure 5.7 occurs when there are no new sub-trees that were added in any given loop. When this happens, it implies that there are no two sub-trees that can be merged to form a valid pair. The final step of the algorithm involves picking the best solution in terms of merging cost that drives all the clock pins.

5.6 Practical Considerations in CCTS

In this section, we address some of the key practical issues that might arise during the application of our CCTS algorithms.

5.6.1 Generalization of Pin Assignment Algorithm

In Section 5.3, the 0-1 Quadratic Programming problem was formulated assuming that the clock pins can be located in only the mid-points of the four sides. However, in many situations, this assumption might not be valid. For example, the clock pin can be located at the mid point of the block and can be reached by using top metal layers at the chip-level. Further, there is no need for the clock pin to be at the center of the block or at the mid-points of the sides. The location might be skewed away from the midpoint to either reduce the block-level delay or to locate the clock pin closer to other sub-block clock pins that have timing critical paths with the sub-block. In the most generic case, a given sub-block can have multiple candidate clock-pin locations on each of the four sides and also candidate locations on the top of block. This situation can be easily handled by introducing two constant weight factors for each candidate location. One new factor should account for the estimated sub-block clock delay for each candidate location. This factor should increase proportionally w.r.t. the estimated delay of sub-block clock tree for the candidate pin location. The second factor should consider the potential routing layer difference that might arise when clock-pin locations on top of the sub-block are considered. Also, another straightforward modification to be done in 5.2 is that the variables p and q that represent the number of candidate pin locations should be changed to account for the new candidate pin locations. Thus, the original formulation in Section 5.3 can still be applied even under such general situation.

5.6.2 Consideration of Blockages

A key requirement of any chip-level CTS algorithm is that it works in the presence of blockages. For example, the chip might have several modes with different clocks. In such cases, the sub-blocks for one clock are to be treated as blockages while synthesizing the clock trees for the other clocks.

We would like to note here that all algorithms presented in our approach to the CCTS problem can be applied even for chips with blockages. For example, the clock pin assignment algorithm can be made blockage aware by measuring the distance between any two candidate pin locations using a *blockage aware global router* instead of a Manhattan estimate. Similarly, the multi-corner subtree balancing heuristic presented in Figure 5.6 can be modified by using the global router based distance instead of Manhattan distance. Since the dynamic programming algorithm internally uses the multi-corner heuristic, that can also be used in the presence of blockages. Thus, we see that all the components of our CCTS approach are generic in nature and can be used even in the presence of blockages.

5.6.3 Measuring Divergence

Reducing the clock divergence is one of the primary focus of this work. However, measuring clock divergence is not as clearly defined as some of the other clock metrics like skew and delay. In this section, we would like explain briefly as to how clock divergence can be measured for a given clock tree. In simple cases such as the one in Figure 5.2, clock divergence is measured simply by the amount of clock delay that is not shared by the two clock sinks. If an additional clock sink is added to this two-sink example and a new timing path is introduced, then we have to consider the clock divergence for the new timing path also. Since the original and new timing paths might not have the same timing criticality, we can introduce relative weights between them so that the total clock divergence is a weighted sum of the two separate clock divergence values. This concept can be easily extended as more clock sinks and timing paths are added. Similarly, clock divergence at the chip-level can be measured as the weighted sum of clock divergence between the sub-clock trees. The weight used for a pair of sub-clock-trees will be proportional to the timing criticality of all the paths between the pair.

5.7 Experimental Results

In this section, we first discuss the details of our test-case generation process. This is followed by detailed explanation of our experimental setup and discussion of our results.

5.7.1 Test-case Generation

To test the effectiveness of our algorithms, we need several chip-level SOC test-cases. Since obtaining test-cases from actual SOC chips is not feasible for us and since there are no known CCTS work in the literature, we generate random test-cases using the data available on SOC chips in the literature [1, 2, 32, 61].

Defining SOC Chip's Physical Attributes: First, we define reasonable *ranges* for the following variables: chip size, number of sub-blocks, size range of the blocks, aspect ratio range for block/chip and chip density. Using these, we generate random chip-level floorplans such that the chip size, number of blocks etc. are all within the selected ranges. We also make sure that the chip density (the ratio of the chip covered by the all blocks) is within limits and that there are no overlaps between the blocks. Each sub-block is marked as a hard or soft IP randomly with probabilities of 0.2 and 0.8 respectively.

Generating Timing Criticality Data: To generate a realistic timing criticality information between block pairs, we consider how the chip-level floorplan is done. A key objective of floorplanning step is to ensure blocks that have a lot of interaction with each other are located close to each other. However, when the interaction between the IPs become complex, placing all the blocks that interact right next to each other becomes impossible. Also, blocks that are very far away from each other rarely have a significant number of critical paths between them. To closely resemble this, we generate the criticality information randomly such that the maximum value on the random number generated remains constant till a certain distance, after which it reduces gradually. Thus, the probability of having a highly critical pair close to each other is higher than having them on the opposite ends of the chip. This process enables us to maintain an important flavor of realistic designs even in randomly generated test-cases.

Parameter	**Value**
Chip Size	0.5x0.5cm^2 to 2.5x2.5cm^2
Chip Utilization	70% to 90%
Aspect Ratio	0.7 to 1.3
Hard-IP Probability	0.2
# Slew Limit Range	90ps to 110ps
Technology	65nm

Table 5.1: Key test-case Generation Parameters.

Generating Sub-block Pin Assignments: The next step in test-case generation is to obtain clock pin assignment and . This is done in two ways to produce two flavors of test-cases. First, we use the pin assignment step of Section 5.3 to get one set of test-cases. Next, we randomly pick the clock pin location for all sub-blocks to get a second set of test-cases with identical floorplan as first set, the only difference being the clock pin locations. The result comparison between these two sets will tell us the effectiveness of our clock pin assignment step.

Generating Sub-block CTS Data: The final step in test-case generation is to mimic the block-level CTS done on the different IPs. This should be done in such a way as to account for the potential differences in the sub-clock-trees due to the difference in the individuals/teams, methodology, cell libraries, etc. We accomplish this by randomly selecting the clock sink density for each block within a pre-selected range, thereby selecting the number of sinks. This number is rounded off to the nearest power of 2 and the number of H-tree levels to drive these flops is obtained. Next, we select a random slew range from a tight range of valid slew. Finally, we recursively choose a random buffer size and use that to drive the H-Tree in a bottom-up fashion to meet the selected slew limit. The procedure stops at the clock root of the block. Because of the use of different buffer sizes and slightly different slew limits, the above procedure mimics, to a good extent, the situation that arises commonly in most SOC designs. Hence testing our chip-level algorithms on these test-cases will verify the effectiveness of our approach. Table-5.1 shows some of the key parameters for our test-case generation script.

5.7.2 Experimental Setup, Results and Discussions

We use the 65nm model cards from [27] for generation of delays across corners. We use three device corners (Nom, FF, SS) to generate the nominal, fast and slow corners. For simplicity, we did not consider other global variations like Voltage, Temperature and Interconnect. As more and more corners are added, the single-corner CTS will be even worse compared to our multi-corner algorithms. In summary, the skew reduction results that we are presenting here are very conservative gains by using our algorithm. The improvements can be expected to be much higher when more variations are considered. Also, adding these variation effects will not change the nature of results on clock divergence reduction as it uses only the nominal corner delay as a guidance for minimizing the divergence.

The four algorithms described in Section 5.5 are run on both sets of test-cases generated above – the ones with random pin placement as well as the ones with pin placement from the Quadratic Programming based approach. Since the two test-case sets are identical in all manner other than the pin locations, a direct comparison of the results from these two sets will indicate the effectiveness of clock pin placement method in reducing clock divergence. Also, we run each of the four CCTS algorithms on all test-cases irrespective of their clock pin placement method. This will be used to compare the effectiveness of the four CCTS algorithms. We generate 5 random test-cases with unique floorplans and different sizes within the ranges given in Table 5.1. Each of these test-cases will have two flavors depending on the pin assignment strategy. Some of the acronyms used in Table 5.2 and Table 5.3 are explained next. TC denotes the Test Case for the results. PAM denotes the Pin Assignment Method used in the test-case. This can either be the Quadratic-Programming (QP) based method or random pin assignment method (RND). The four CCTS algorithms described earlier are abbreviated as: 1C-DME (Singe-Corner DME approach), MC-DME (Multi-corner DME approach), MC-GRD (Multi-corner Greedy algorithm) and MC-DyP (Multi-corner Dynamic Programming algorithm). The divergence values given are weighted sum of clock divergence between all IP pairs. The weights are proportional to the timing criticality of all the paths between the sub-block pairs. The multi-corner delay, skew and divergence results in Table 5.2 have been averaged and normalized in Table 5.3 along with the data on buffer area and wire-length for each test-case and algorithm. Based on the results in Table 5.2 and 5.3, we observe the following:

- Skew across corners using our multi-corner skew reduction algorithm is significantly better than the single corner approach. Though the nominal skew itself is comparable in many cases, the skews in other corners are lesser in the multi-corner CTS methods. The average reduction in skew in fast/slow corner ranges from 16ps to as much as 64 ps in some cases. This is 1.6 % to 6.4% of cycle time even for a 1GHz clock and will be much more crucial for faster clocks. This proves the effectiveness of out multi-corner skew reduction method.

- Comparing the normalized average results of Table 5.3, we see that QP pin assignment method gives better clock divergence and skew reduction compared to the random pin placement for all the different CCTS algorithms. This demonstrates the effectiveness of our pin-assignment method.
- For a given pin assignment method, our dynamic programming based CCTS algorithm achieves an average 19% divergent delay reduction and almost halves the multi-corner average slew. This has been achieved with with less than 0.4% increase in buffer area and wire-length.
- Overall, when compared to a single-corner DME based approach with random pin placement, the dynamic programming based approach with QP based approach achieves an average clock divergence reduction of more than 20% with negligible impact on buffer area, wire-length and overall delay.

It may be noted that we have measured the effectiveness of our algorithms using divergent delay reduction and not yield improvement or skew variation reduction between critical paths. Both the latter metrics require precise timing path information between all IP pairs, which is not available for our randomly generated test-cases. Nevertheless, since our final clock skew divergence number is a weighted sum of pair-wise divergent clock delays, with weights being proportional to timing criticality of IP pairs, clock divergence reduction directly translates to a increase in timing yield for a given chip. Similarly, for a given yield, clock divergence reduction translates into reduction in closure time/effort and resources (people/buffer/interconnect/power) used to achieve the target yield.

			Delay(ns)			Skew(ps)			Divergence(ps)		
TC	PAM	meth.	Nom	Slow	Fast	Nom	Slow	Fast	Nom	Slow	Fast
TC1	RND	1C-DME	2.47	3.10	1.98	3.46	114.21	75.81	40.74	51.17	32.76
		MC-DME	2.49	3.08	2.04	64.50	80.36	52.59	38.33	47.43	31.34
		MC-GRD	2.44	3.03	2.00	47.21	53.23	68.46	36.06	40.44	32.66
		MC-DyP	2.46	3.04	2.01	43.43	52.49	28.58	35.39	43.80	28.94
	QP	1C-DME	2.51	3.17	2.02	17.97	110.71	88.13	41.17	51.81	33.01
		MC-DME	2.56	3.17	2.10	33.64	49.69	28.53	39.43	48.72	32.29
		MC-GRD	2.44	3.03	2.00	47.21	53.23	68.46	35.57	39.84	32.26
		MC-DyP	2.51	3.11	2.06	19.61	13.55	17.70	35.05	43.40	28.72
TC2	RND	1C-DME	3.04	3.84	2.43	55.58	128.53	158.65	530.69	667.43	425.47
		MC-DME	3.09	3.83	2.53	88.03	129.90	65.57	525.98	651.59	429.31
		MC-GRD	3.03	3.76	2.46	111.13	163.36	122.69	567.45	643.83	509.32
		MC-DyP	3.01	3.72	2.47	93.52	127.87	91.77	427.66	526.09	347.61
	QP	1C-DME	3.02	3.82	2.43	57.92	126.21	157.03	536.63	673.94	430.24
		MC-DME	3.12	3.87	2.57	104.84	143.90	82.39	538.35	666.54	440.58
		MC-GRD	3.02	3.73	2.46	95.71	136.70	134.66	550.14	623.01	494.70
		MC-DyP	3.01	3.73	2.47	83.13	102.05	77.65	398.11	489.95	323.53
TC3	RND	1C-DME	1.45	1.82	1.16	33.22	84.95	52.88	238.11	297.24	191.20
		MC-DME	1.45	1.78	1.17	82.53	78.48	66.07	234.28	287.93	189.85
		MC-GRD	1.44	1.77	1.16	82.39	89.17	82.10	226.30	255.95	201.70
		MC-DyP	1.43	1.75	1.15	68.52	89.42	82.53	183.36	225.31	148.40
	QP	1C-DME	1.44	1.80	1.14	48.02	77.41	70.17	213.28	266.79	170.61
		MC-DME	1.42	1.74	1.14	87.80	115.26	77.92	214.84	265.11	174.04
		MC-GRD	1.45	1.78	1.17	67.79	83.40	48.85	216.33	243.49	193.72
		MC-DyP	1.45	1.79	1.17	56.68	92.32	36.40	166.53	205.05	135.55
TC4	RND	1C-DME	2.10	2.66	1.66	20.48	125.17	73.80	96.69	121.73	76.93
		MC-DME	2.07	2.56	1.67	68.48	80.75	56.35	99.07	122.39	79.86
		MC-GRD	2.07	2.55	1.65	80.09	93.74	57.34	94.04	105.90	84.01
		MC-DyP	2.15	2.65	1.73	29.37	29.89	32.08	87.05	107.45	70.02
	QP	1C-DME	2.10	2.65	1.66	19.49	118.84	66.63	103.26	130.36	82.06
		MC-DME	2.13	2.63	1.71	53.42	67.24	36.70	98.47	121.52	79.08
		MC-GRD	2.07	2.55	1.65	80.09	93.74	57.34	92.46	103.95	82.72
		MC-DyP	2.15	2.65	1.73	24.74	27.29	25.52	86.58	106.90	69.65
TC5	RND	1C-DME	2.40	3.02	1.93	21.34	154.38	136.90	383.75	480.56	307.99
		MC-DME	2.38	2.95	1.94	67.18	121.11	118.53	382.86	472.86	312.27
		MC-GRD	2.39	2.96	1.95	57.24	88.74	99.22	386.62	438.88	346.09
		MC-DyP	2.38	2.95	1.95	89.76	113.12	94.04	311.94	385.59	255.20
	QP	1C-DME	2.40	3.02	1.92	29.69	147.85	142.73	374.87	469.92	300.69
		MC-DME	2.43	3.01	1.99	79.31	131.16	106.97	376.63	465.00	306.87
		MC-GRD	2.39	2.96	1.95	57.24	88.74	99.22	370.88	419.40	333.25
		MC-DyP	2.45	3.05	2.02	22.03	50.71	41.45	322.09	399.00	263.64

Table 5.2: Multi-corner delay, skew and clock divergence results for the different CCTS approaches.

			Delay(ps)		Skew(ps)		Div. (ps)		BA		WL	
TC	PAM	Meth.	Avg	Nor.	Avg	Nor.	Avg	Nor.	um^2	Nor.	m	Nor.
TC1	RND	1C-DME	2.52	1.00	64.50	1.00	41.5	1.00	1.49	1.000	7.3	1.000
		MC-DME	2.54	1.00	65.82	1.02	39.0	0.93	1.49	1.001	7.3	1.001
		MC-GRD	2.49	0.98	56.30	0.87	36.3	0.87	1.50	1.003	7.3	1.002
		MC-DyP	2.50	0.99	41.50	0.64	36.0	0.86	1.49	1.002	7.3	1.001
	QP	1C-DME	2.57	1.00	72.27	1.00	42.0	1.00	1.49	1.000	7.3	1.000
		MC-DME	2.61	1.01	37.29	0.51	40.1	0.95	1.49	1.002	7.3	1.002
		MC-GRD	2.49	0.97	56.30	0.77	35.8	0.85	1.49	1.003	7.3	1.002
		MC-DyP	2.56	0.99	16.95	0.23	35.7	0.85	1.49	1.002	7.3	1.001
TC2	RND	1C-DME	3.11	1.00	114.25	1.00	541.2	1.00	5.05	1.000	26.4	1.000
		MC-DME	3.15	1.01	94.50	0.82	535.6	0.99	5.06	1.001	26.5	1.001
		MC-GRD	3.09	0.99	132.39	1.15	573.5	1.06	5.08	1.007	26.6	1.004
		MC-DyP	3.07	0.98	104.39	0.91	433.7	0.80	5.07	1.003	26.5	1.002
	QP	1C-DME	3.09	1.00	113.72	1.00	546.9	1.00	5.04	1.000	26.4	1.000
		MC-DME	3.19	1.03	110.38	0.97	548.4	1.00	5.06	1.002	26.5	1.001
		MC-GRD	3.07	0.99	122.36	1.07	555.9	1.01	5.07	1.005	26.5	1.003
		MC-DyP	3.07	0.99	87.61	0.77	403.8	0.73	5.06	1.002	26.5	1.001
TC3	RND	1C-DME	1.48	1.00	57.02	1.00	242.1	1.00	1.01	1.000	5.3	1.000
		MC-DME	1.46	0.98	75.69	1.32	237.3	0.98	1.01	1.002	5.3	1.001
		MC-GRD	1.45	0.98	85.63	1.50	227.9	0.94	1.02	1.009	5.3	1.005
		MC-DyP	1.44	0.97	80.16	1.40	185.6	0.76	1.02	1.007	5.3	1.003
	QP	1C-DME	1.46	1.00	65.20	1.00	216.9	1.00	1.00	1.000	5.3	1.000
		MC-DME	1.43	0.98	93.66	1.43	218.0	1.00	1.01	1.003	5.3	1.001
		MC-GRD	1.47	1.00	66.68	1.02	217.8	1.00	1.01	1.009	5.3	1.004
		MC-DyP	1.47	1.00	61.80	0.94	169.0	0.77	1.01	1.006	5.3	1.002
TC4	RND	1C-DME	2.14	1.00	73.15	1.00	98.4	1.00	0.84	1.000	4.1	1.000
		MC-DME	2.10	0.98	68.53	0.93	100.4	1.02	0.84	1.004	4.1	1.002
		MC-GRD	2.09	0.97	77.06	1.05	94.6	0.96	0.85	1.013	4.1	1.007
		MC-DyP	2.18	1.01	30.44	0.41	88.1	0.89	0.84	1.007	4.1	1.005
	QP	1C-DME	2.14	1.00	68.32	1.00	105.2	1.00	0.84	1.000	4.1	1.000
		MC-DME	2.16	1.01	52.45	0.76	99.6	0.94	0.84	1.005	4.1	1.001
		MC-GRD	2.09	0.97	77.06	1.12	93.0	0.88	0.84	1.010	4.1	1.005
		MC-DyP	2.18	1.01	25.85	0.37	87.7	0.83	0.84	1.005	4.1	1.003
TC5	RND	1C-DME	2.45	1.00	104.21	1.00	390.7	1.00	2.86	1.000	14.5	1.000
		MC-DME	2.42	0.98	102.27	0.98	389.3	0.99	2.87	1.003	14.5	1.002
		MC-GRD	2.43	0.99	81.73	0.78	390.5	0.99	2.89	1.010	14.6	1.006
		MC-DyP	2.43	0.99	98.97	0.95	317.5	0.81	2.88	1.008	14.6	1.004
	QP	1C-DME	2.45	1.00	106.76	1.00	381.8	1.00	2.86	1.000	14.5	1.000
		MC-DME	2.48	1.01	105.81	0.99	382.8	1.00	2.87	1.004	14.5	1.002
		MC-GRD	2.43	0.99	81.73	0.76	374.5	0.98	2.88	1.009	14.6	1.005
		MC-DyP	2.51	1.02	38.07	0.35	328.2	0.86	2.87	1.005	14.5	1.002
Avg	RND	1C-DME	-.	1.00	-	1.00	-	1.00	-	1.000	-	1.000
		MC-DME	-	0.99	-	1.01	-	0.98	-	1.002	-	1.001
		MC-GRD	-	0.98	-	1.07	-	0.96	-	1.008	-	1.005
		MC-DyP	-	0.99	-	0.86	-	0.82	-	1.005	-	1.003
	QP	1C-DME	-	1.00	-	1.00	-	1.00	-	1.000	-	1.000
		MC-DME	-	1.00	-	0.93	-	0.98	-	1.003	-	1.001
		MC-GRD	-	0.98	-	0.95	-	0.94	-	1.007	-	1.004
		MC-DyP	-	1.00	-	0.53	-	0.81	-	1.004	-	1.002

Table 5.3: Average & normalized delay, skew, clock divergence information along with Buffer Area(BA) and Wire Length(WL) results for the test-cases in Table 5.2. All normalization done w.r.t. results of 1-corner DME approach.

Chapter 6

Conclusion

In this dissertation, we have proposed several effective methods of obtaining variation tolerant clock networks for different types of VLSI chips. The various link insertion algorithms presented in Chapter-2 can be used effectively for small ASIC chips that have strict area and power requirements, thereby gaining significant skew variability reduction with a negligible increase in wire-length and power. The link-based buffered clock network synthesis algorithms presented in Chapter-3 can be used in medium size/performance ASIC chips that can afford to increase power to a small extent to gain significant skew variation reduction. Both high performance ASICs and power constrained microprocessors can use the MeshWorks methodology presented in Chapter-4 to significantly reduce the power and resources needed by a leaf-level clock mesh without sacrificing the variation tolerant nature of the mesh. Finally, large SOC designs can use the algorithms in Chapter-5 to obtain a robust, multi-corner chip-level clock tree that can significantly reduce the impact of variation on chip performance/yield by reducing the clock divergence between critical sub-chips. Thus, each of the methods proposed in this work can be used in different contexts towards the goal of synthesizing variation tolerant clock distribution networks.

Bibliography

[1] S. Agarwala, A. Rajagopal, A. Hill, M. Joshi, S. Mullinnix, T. Anderson, R. Damodaran, L. Nardini, P. Wiley, P. Groves, J. Apostol, M. Gill, J. Flores, A. Chachad, A. Hales, K. Chirca, K. Panda, R. Venkatasubramanian, P. Eyres, R. Veiamuri, A. Rajaram, M. Krishnan, J. Nelson, J. Frade, M. Rahman, N. Mahmood, U. Narasimha, S. Sinha, S. Krishnan, W. Webster, Due Bui, S. Moharii, N. Common, R. Nair, R. Ramanujam, and M. Ryan. A 65nm c64x+ multi-core dsp platform for communications infrastructure. *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 262–601, Feb. 2007.

[2] S. Agarwala, P. Wiley, A. Rajagopal, A. Hill, R. Damodaran, L. Nardini, T. Anderson, S. Mullinnix, J. Flores, H. Yue, A. Chachad, J. Apostol, K. Castille, U. Narasimha, T. Wolf, NS. Nagaraj, M. Krishnan, L. Nguyen, T. Kroeger, M. Gill, P. Groves, B. Webster, J. Graber, and C. Karlovich. A 800 mhz system-on-chip for wireless infrastructure applications. In *VLSID '04: Proceedings of the 17th International Conference on VLSI Design*, page 381, Washington, DC, USA, 2004. IEEE Computer Society.

[3] C.J. Alpert, R.G. Gandham, J.L. Neves, and S.T. Quay. Buffer library selection. *Computer Design, 2000. Proceedings. 2000 International Conference on*, pages 221–226, 2000.

[4] I. A. Blech. Electromigration in thin aluminum films on titanium nitride.

Journal of Applied Physics, 47(4):1203–1208, 1976.

[5] K.D. Boese, A.B. Kahng, B.A. McCoy, and G. Robins. Near-optimal critical sink routing tree constructions. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(12):1417–1436, Dec 1995.

[6] M. Celik, L. Pileggi, and A. Odabasioglu. *IC Interconnect Analysis*. Springer Publishers, 2002.

[7] P.K. Chan and K. Karplus. Computing signal delay in general rc networks by tree/link partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 9(8):898–902, Aug 1990.

[8] C.W. Chang, C.L. Gan, C.V. Thompson, K.L. Pey, W.K. Choi, and M.H. Chua. Joule heating-assisted electromigration failure mechanisms for dual damascene cu/sio2 interconnects. *Physical and Failure Analysis of Integrated Circuits, 2003. IPFA 2003. Proceedings of the 10th International Symposium on the*, pages 69–74, July 2003.

[9] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 621, Washington, DC, USA, 2003. IEEE Computer Society.

[10] T. H. Chao, Y. C. Hsu, J. . Ho, and A.B. Kahng. Zero skew clock routing with minimum wirelength. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 39(11):799–814, Nov 1992.

[11] R. Chaturvedi and J. Hu. Buffered clock tree for high quality ic design. In *ISQED '04: Proceedings of the 5th International Symposium on Quality Electronic Design*, pages 381–386, Washington, DC, USA, 2004. IEEE Computer Society.

[12] R. Chaturvedi and J. Hu. An efficient merging scheme for prescribed skew clock routing. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(6):750–754, June 2005.

[13] 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 *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 939–946, Washington, DC, USA, 2005. IEEE Computer Society.

[14] Y. Chen and M. D. F. Wong. An algorithm for zero-skew clock tree routing with buffer insertion. Technical report, Austin, TX, USA, 1995.

[15] M. Cho, S. Ahmedtt, and D. Z. Pan. Taco: temperature aware clock-tree optimization. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 582–587, Washington, DC, USA, 2005. IEEE Computer Society.

[16] J. Chung and C. K. Cheng. Optimal buffered clock tree synthesis. *ASIC Conference and Exhibit, 1994. Proceedings., Seventh Annual IEEE International*, pages 130–133, Sep 1994.

[17] J. Cong, A. B. Kahng, C. K Koh, and C.-W. A. Tsao. Bounded-skew clock

and steiner routing. *ACM Trans. Des. Autom. Electron. Syst.*, 3(3):341–388, 1998.

[18] T. T. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms.* MIT Press, Cambridge, MA, USA, 1990.

[19] M. P. Desai, R. Cvijetic, and J. Jensen. Sizing of clock distribution networks for high performance cpu chips. In *DAC '96: Proceedings of the 33rd annual conference on Design automation*, pages 389–394, New York, NY, USA, 1996. ACM.

[20] M. Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *DAC '93: Proceedings of the 30th international conference on Design automation*, pages 612–616, New York, NY, USA, 1993. ACM.

[21] E.G. Friedman. Clock distribution networks in synchronous digital integrated circuits. *Proceedings of the IEEE*, 89(5):665–692, May 2001.

[22] A. Gattiker, S. Nassif, R. Dinakar, and C. Long. Timing yield estimation from static timing analysis. *Quality Electronic Design, 2001 International Symposium on*, pages 437–442, 2001.

[23] M. R. Guthaus, D. Sylvester, and R. B. Brown. Clock buffer and wire sizing using sequential programming. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 1041–1046, New York, NY, USA, 2006. ACM.

[24] D. Harris and S. Naffziger. Statistical clock skew modeling with data delay variations. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(6):888–898, 2001.

[25] R. Heald, K. Aingaran, C. Amir, M. Ang, M. Boland, A. Das, P. Dixit, G. Gouldsberry, J. Hart, T. Horel, Wen-Jay Hsu, J. Kaku, Chin Kim, Song Kim, F. Klass, Hang Kwan, Roger Lo, H. McIntyre, A. Mehta, D. Murata, S. Nguyen, Yet-Ping Pai, S. Patel, K. Shin, Kenway Tam, S. Vishwanthaiah, J. Wu, Gin Yee, and Hong You. Implementation of a 3rd-generation sparc v9 64 b microprocessor. *Solid-State Circuits Conference, 2000. Digest of Technical Papers. ISSCC. 2000 IEEE International*, pages 412–413, 2000.

[26] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/.

[27] http://www.eas.asu.edu/ptm.

[28] http://www.mathworks.com/products/optimization/description5.html.

[29] http://www.synopsys.com/products/mixedsignal/hspice/hspice.html.

[30] H. Th. Jongen, K. Meer, and E. Tries. *Optimization Theory.* Springer Publishers, 2004.

[31] A. Kapoor, N. Jayakumar, and S. P. Khatri. A novel clock distribution and dynamic de-skewing methodology. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 626–631, Washington, DC, USA, 2004. IEEE Computer Society.

[32] M. Keating and P. Bricaud. *Reuse methodology manual: for system-on-a-chip designs.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[33] C.-K. Koh. Power supply noise suppression via clock skew scheduling. In *ISQED '02: Proceedings of the 3rd International Symposium on Quality*

Electronic Design, page 355, Washington, DC, USA, 2002. IEEE Computer Society.

[34] N.A. Kurd, J.S. Barkarullah, R.O. Dizon, T.D. Fletcher, and P.D. Madland. A multigigahertz clocking scheme for the pentium(r) 4 microprocessor. *Solid-State Circuits, IEEE Journal of*, 36(11):1647–1653, Nov 2001.

[35] W.-C. D. Lam, J. Jam, C.-K. Koh, V. Balakrishnan, and Y. Chen. Statistical based link insertion for robust clock network design. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 588–591, Washington, DC, USA, 2005. IEEE Computer Society.

[36] J. Leeds and G. Ugron. Simplified multiple parameter sensitivity calculation and continuously equivalent networks. *Circuit Theory, IEEE Transactions on*, 14(2):188–191, Jun 1967.

[37] Z. Li, X. Lu, and W. Shi. Process variation dimension reduction based on svd [circuit simulation]. *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, 4:IV–672–IV–675 vol.4, May 2003.

[38] J. Lienig and G. Jerke. Electromigration-aware physical design of integrated circuits. In *VLSID '05: Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, pages 77–82, Washington, DC, USA, 2005. IEEE Computer Society.

[39] M.H. Lin, Y.L. Lin, G.S. Yang, M.-S. Yeh, K.P. Chang, K.C. Su, and Tahui Wang. Comparison of copper interconnect electromigration behaviors in various structures for advanced beol technology. *Physical and Failure Analysis of Integrated Circuits, 2004. IPFA 2004. Proceedings of the 11th International Symposium on the*, pages 177–180, July 2004.

[40] S. Lin and C. K. Wong. Process-variation-tolerant clock skew minimization. In *ICCAD '94: Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design*, pages 284–288, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[41] Y. Liu, X. Hong, Y. Cai, and X. Wei. Reliable buffered clock tree routing algorithm with process variation tolerance. *ASIC, 2003. Proceedings. 5th International Conference on*, 1:344–347 Vol.1, Oct. 2003.

[42] Y. Liu, S. R. Nassif, L. T. Pileggi, and A. J. Strojwas. Impact of interconnect variations on the clock skew of a gigahertz microprocessor. In *DAC '00: Proceedings of the 37th conference on Design automation*, pages 168–171, New York, NY, USA, 2000. ACM.

[43] B. Lu, J. Hu, G. Ellis, and H. Su. Process variation aware clock tree routing. In *ISPD '03: Proceedings of the 2003 international symposium on Physical design*, pages 174–181, New York, NY, USA, 2003. ACM.

[44] E. Malavasi, S. Zanella, Min Cao, J. Uschersohn, M. Misheloff, and C. Guardiani. Impact analysis of process variability on clock skew. *Quality Electronic Design, 2002. Proceedings. International Symposium on*, pages 129–132, 2002.

[45] A.D. Mehta, Y-P. Chen, N. Menezes, D.F. Wong, and L.T. Pilegg. Clustering and load balancing for buffered clock tree synthesis. *Computer Design: VLSI in Computers and Processors, 1997. ICCD '97. Proceedings., 1997 IEEE International Conference on*, pages 217–223, Oct 1997.

[46] M. Mori, H. Chen, B. Yao, and C. K. Cheng. A multiple level network approach for clock skew minimization with process variations. In *ASP-DAC '04: Proceedings of the 2004 conference on Asia South Pacific design automation*, pages 263–268, Piscataway, NJ, USA, 2004. IEEE Press.

[47] S.R. Nassif. Modeling and analysis of manufacturing variations. *Custom Integrated Circuits, 2001, IEEE Conference on.*, pages 223–228, 2001.

[48] G. Northrop, R. Averill, K. Barkley, S. Carey, Y. Chan, Y.H. Chan, M. Check, D. Hoffman, W. Huott, B. Krumm, C. Krygowski, J. Liptay, M. Mayo, T. McNamara, T. McPherson, E. Schwarz, L.S.T. Siegel, C. Webb, D. Webber, and P. Williams. 609 mhz g5 s/399 microprocessor. *Solid-State Circuits Conference, 1999. Digest of Technical Papers. ISSCC. 1999 IEEE International*, pages 88–89, 1999.

[49] D. Z. Pan and S. Tam. Tutorials on dfm routing and clock distribution. 2007.

[50] L. T. Pillage, R. A. Rohrer, and C. Visweswariah. *Electronic Circuit and System Simulation Methods*. Mcgraw-Hill Publishers, 1998.

[51] S. Pullela, N. Menezes, and L.T. Pillage. Reliable non-zero skew clock trees using wire width optimization. *Design Automation, 1993. 30th Conference on*, pages 165–170, June 1993.

[52] S. Pullela, N. Menezes, and L.T. Pillage. Low power ic clock tree design. *Custom Integrated Circuits Conference, 1995., Proceedings of the IEEE 1995*, pages 263–266, May 1995.

[53] R. Puri, D. S. Kung, and A. D. Drumm. Fast and accurate wire delay estimation for physical synthesis of large asics. In *GLSVLSI '02: Proceedings of the 12th ACM Great Lakes symposium on VLSI*, pages 30–36, New York, NY, USA, 2002. ACM.

[54] A. Rajaram, J. Hu, and R. Mahapatra. Reducing clock skew variability via cross links. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 18–23, New York, NY, USA, 2004. ACM.

[55] A. Rajaram, B. Lu, W. Guo, R. Mahapatra, and J. Hu. Analytical bound for unwanted clock skew due to wire width variation. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 401, Washington, DC, USA, 2003. IEEE Computer Society.

[56] A. Rajaram and D. Z. Pan. Fast incremental link insertion in clock networks for skew variability reduction. In *ISQED '06: Proceedings of the 7th International Symposium on Quality Electronic Design*, pages 79–84, Washington, DC, USA, 2006. IEEE Computer Society.

[57] A. Rajaram and D. Z. Pan. Variation tolerant buffered clock network synthesis with cross links. In *ISPD '06: Proceedings of the 2006 international symposium on Physical design*, pages 157–164, New York, NY, USA, 2006. ACM.

[58] A. Rajaram and D. Z. Pan. Meshworks: an efficient framework for planning, synthesis and optimization of clock mesh networks. In *ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation*, pages 250–257, Los Alamitos, CA, USA, 2008. IEEE Computer Society Press.

[59] A. Rajaram and D. Z. Pan. Robust chip-level clock tree synthesis for soc designs. In *DAC '08: Proceedings of the 45th annual conference on Design automation*, pages 720–723, New York, NY, USA, 2008. ACM.

[60] A. Rajaram, D. Z. Pan, and J. Hu. Improved algorithms for link-based non-tree clock networks for skew variability reduction. In *ISPD '05: Proceedings of the 2005 international symposium on Physical design*, pages 55–62, New York, NY, USA, 2005. ACM.

[61] R. Rajsuman. *System-on-a-Chip: Design and Test.* Artech House, Inc., Norwood, MA, USA, 2000.

[62] S. M. Reddy, G. R. Wilke, and R. Murgai. Analyzing timing uncertainty in mesh-based clock architectures. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 1097–1102, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[63] P. J. Restle. Personal communication.

[64] P.J. Restle, T.G. McNamara, D.A. Webber, P.J. Camporese, K.F. Eng, K.A. Jenkins, D.H. Allen, M.J. Rohn, M.P. Quaranta, D.W. Boerstler, C.J. Alpert, C.A. Carter, R.N. Bailey, J.G. Petrovick, B.L. Krauter, and B.D.

McCredie. A clock distribution network for microprocessors. *Solid-State Circuits, IEEE Journal of*, 36(5):792–799, May 2001.

[65] R. Saleh, S.Z. Hussain, S. Rochel, and D. Overhauser. Clock skew verification in the presence of ir-drop in the power distribution network. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(6):635–644, Jun 2000.

[66] M. Shao, M. D. F. Wong, H. Cao, Y. Gao, L-P. Yuan, L-D Huang, and S. Lee. Explicit gate delay model for timing evaluation. In *ISPD '03: Proceedings of the 2003 international symposium on Physical design*, pages 32–38, New York, NY, USA, 2003. ACM.

[67] H. Su and S. S. Sapatnekar. Hybrid structured clock network construction. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pages 333–336, Piscataway, NJ, USA, 2001. IEEE Press.

[68] C. M. Tan, A. Roy, A.V. Vairagar, A. Krishnamoorthy, and S.G. Mhaisalkar. Current crowding effect on copper dual damascene via bottom failure for ulsi applications. *Device and Materials Reliability, IEEE Transactions on*, 5(2):198–205, June 2005.

[69] J. Tao, N.W. Cheung, and C. Hu. Electromigration characteristics of copper interconnects. *Electron Device Letters, IEEE*, 14(5):249–251, May 1993.

[70] G.E. Tellez and M. Sarrafzadeh. Minimal buffer insertion in clock trees with skew and slew rate constraints. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(4):333–342, Apr 1997.

[71] A. Todri and M. Marek-Sadowska. A study of reliability issues in clock distribution networks. In *ICCD'08: Proceedings of the 26th International Conference on Computer Design*, pages 77–82, 2008.

[72] J. L. Tsai, D. Baik, C. C. P. Chen, and K. K. Saluja. A yield improvement methodology using pre- and post-silicon statistical clock scheduling. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design*, pages 611–618, Washington, DC, USA, 2004. IEEE Computer Society.

[73] J. L. Tsai, T. H. Chen, and C.C.-P. Chen. Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(4):565–572, April 2004.

[74] C. W. A. Tsao and C. K. Koh. Ust/dme: a clock tree router for general skew constraints. *ACM Trans. Des. Autom. Electron. Syst.*, 7(3):359–379, 2002.

[75] R.-S. Tsay. Exact zero skew. *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 336–339, Nov 1991.

[76] D. Velenis, E.G. Friedman, and M.C. Papaefthymiou. A clock tree topology extraction algorithm for improving the tolerance of clock distribution networks to delay uncertainty. *Circuits and Systems, 2001. ISCAS 2001. The 2001 IEEE International Symposium on*, 4:422–425 vol. 4, May 2001.

[77] G. Venkataraman, Z. Feng, J. Hu, and P. Li. Combinatorial algorithms for fast clock mesh optimization. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 563–567, New York, NY, USA, 2006. ACM.

[78] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness, and C. Alpert. Practical techniques to reduce skew and its variations in buffered clock networks. In *ICCAD '05: Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pages 592–596, Washington, DC, USA, 2005. IEEE Computer Society.

[79] C. Visweswariah. Death, taxes and failing chips. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 343–347, New York, NY, USA, 2003. ACM.

[80] A. Vittal and M. Marek-Sadowska. Low-power buffered clock tree design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(9):965–975, Sep 1997.

[81] K. Wang and M. Marek-Sadowska. Buffer sizing for clock power minimization subject to general skew constraints. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 159–164, New York, NY, USA, 2004. ACM.

[82] K. Wang and M. Marek-Sadowska. Clock network sizing via sequential linear programming with time-domain analysis. In *ISPD '04: Proceedings of the 2004 international symposium on Physical design*, pages 182–189, New York, NY, USA, 2004. ACM.

[83] V. Wason, R. Murgai, and W. W. Walker. An efficient uncertainty- and skew-aware methodology for clock tree synthesis and analysis. In *VLSID '07: Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference*, pages 271–277, Washington, DC, USA, 2007. IEEE Computer Society.

[84] T. Xue and E. S. Kuh. Post routing performance optimization via multi-link insertion and non-uniform wiresizing. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 575–580, Washington, DC, USA, 1995. IEEE Computer Society.

[85] J. T. Yan, C. W. Wu, K. P. Lin, Y. C. Lee, and T. Y. Wang. Iterative convergence of optimal wire sizing and available buffer insertion for zero-skew clock tree optimization. *Circuits and Systems, 2004. Proceedings. The 2004 IEEE Asia-Pacific Conference on*, 1:529–532 vol.1, Dec. 2004.

Vita

Anand Kumar Rajaram was born in Pammal, a suburb of Chennai, India. He completed his entire schooling in Sri Shankara Vidhyalaya, Pammal, and later he joined the College of Engineering Guindy, Anna University in August 1998. He earned his Bachelor of Engineering degree in Electrical and Electronics Engineering from Anna University with distinction in May 2002. He moved to Texas A&M University, College Station, in the Fall of 2002 where he earned his Master of Science degree in Electrical Engineering in August 2004. During the Summer of 2004, Anand worked as an Intern in Zenasis Technologies Inc. Since August 2004, he has been a Ph.D. student doing research on Variation Tolerant Clock Networks under the guidance of Prof. David Pan in the Electrical and Computer Engineering Department of University of Texas at Austin. Anand also worked fulltime in the Dallas DSP group of Texas Instruments, Dallas, since August 2004. He recently joined Magma Design Automation, Austin, Texas. Anand has received two Best Paper Nominations, one in DAC 2004 and another in ASP-DAC 2008.

Permanent address: 7 Babu Street, Krishna Nagar, Pammal,
Chennai-75, India 600075

This dissertation was typeset with LaTeX[†] by the author.

[†] LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.