# DESIGN METHODOLOGIES FOR VARIATION-AWARE INTEGRATED CIRCUITS

A Dissertation

by

RUPAK SAMANTA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2008

Major Subject: Computer Engineering

# DESIGN METHODOLOGIES FOR VARIATION-AWARE INTEGRATED CIRCUITS

A Dissertation

by

RUPAK SAMANTA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

| | |
|---|---|
| Chair of Committee, | Jiang Hu |
| Committee Members, | Weiping Shi |
| | Jose Silva-Martinez |
| | Vivek Sarin |
| Head of Department, | Costas N. Georghiades |

December 2008

Major Subject: Computer Engineering

# ABSTRACT

Design Methodologies for Variation-Aware Integrated Circuits. (December 2008)

Rupak Samanta, B.E., Sambalpur University, Burla;

M.Tech., Indian Institute of Technology, Mumbai

Chair of Advisory Committee: Dr. Jiang Hu

The scaling of VLSI technology has spurred a rapid growth in the semiconductor industry. With the CMOS device dimension scaling to and beyond 90nm technology, it is possible to achieve higher performance and to pack more complex functionalities on a single chip. However, the scaling trend has introduced drastic variation of process and design parameters, leading to severe variability of chip performance in nanometer regime. Also, the manufacturing community projects CMOS will scale for three to four more generations. Since the uncertainties due to variations are expected to increase in each generation, it will significantly impact the performance of design and consequently the yield.

Another challenging issue in the nanometer IC design is the high power consumption due to the greater packing density, higher frequency of operation and excessive leakage power. Moreover, the circuits are usually over-designed to compensate for uncertainties due to variations. The over-designed circuits not only make timing closure difficult but also cause excessive power consumption. For portable electronics, excessive power consumption may reduce battery life; for non-portable systems it may impose great difficulties in cooling and packaging.

The objective of my research has been to develop design methodologies to address variations and power dissipation for reliable circuit operation. The proposed work has been divided into three parts: the first part addresses the issues related with power/ground noise induced by clock distribution network and proposes techniques

to reduce power/ground noise considering the effects of process variations. The second part proposes an elastic pipeline scheme for random circuits with feedback loops. The proposed scheme provides a low-power solution that has the same variation tolerance as the conventional approaches. The third section deals with discrete buffer and wire sizing for link-based non-tree clock network, which is an energy efficient structure for skew tolerance to variations.

For the power/ground noise problem, our approach could reduce the peak current and the delay variations by **50%** and **51%** respectively. Compared to conventional approach, the elastic timing scheme reduces power dissipation by $20\% - 27\%$. The sizing method achieves clock skew reduction of 45% with a small increase in power dissipation.

To my Parents and Wife Archana

# ACKNOWLEDGMENTS

There is a lifetime's worth of people who have undoubtedly influenced and encouraged me in many aspects of my life. I would like to thank all of them. I must limit myself to recognizing those in the present and most recent past.

First and foremost, I would like to express my deepest gratitude to my advisor Dr. Jiang Hu, for his invaluable academic guidance, patience and respect to my ideas and thoughts and timely advice on various aspects of my graduate student life. This research work would not have been possible without his support and guidance.

I would like to thank my committee members Dr. Weiping Shi, Dr. Jose Silva-Martinez and Dr. Vivek Sarin, for their valuable input and advice throughout the duration of my research. I would like to thank all the faculty members of computer engineering for their help during course projects or for my research. Special thanks to Dr. Weiping Shi for his course Algorithms for VLSI CAD which greatly helped me in honing my physical design skills and igniting my interest into the world of physical design. The students of the computer engineering group were equally helpful to my research effort. In particular, I'd like to thank Karandeep Singh, Rajesh Garg, Charu Nagpal, Nimay Shah, Shiyan Hu, Ganesh Venkataraman and Rohit Singhal. I really enjoyed working with them. I'd like to thank Saurabh Pradhan, Feroze Merchant, Nicholas Tsang and Manjunath Shamanna for being my mentors at Intel Austin during my internship. Many thanks to the FEC team members for the wonderful time I had during the internship.

I would like to thank all my friends who made my stay at Texas A&M very enjoyable. In particular, I would like to thank Bhoj, Karan, Ranjani, Charu, Nimay, Rajesh, Bhagi, Binoy, Rajmohan, Biswajit, Somnath, Gagan, Ashwin, Aditi and Vivek. Many thanks to Venkat, Swapna, Anuj and Sonal for the wonderful time in

Austin. I would like to extend my appreciation to a few friends who have been with me during all the difficult times of my life. Those who need special mention are Rakesh, Lala, Deba, Raju, Aditya, Manas and Suchismita. Lastly, I would like to thank the late Soyuz Priyadarshan for his invaluable advice during the initial years of my Ph.D.

I would like to thank my parents and my dear wife Archana, who have given me all their love and support and let me freely do whatever I want. Without them, there is no way I could possibly have accomplished this. Their understanding on the value of education is truly beyond my comprehension. I am just a lucky beneficiary. Along the way, my sister Rasmita has always shared her caring thoughts and encouraged me during the difficult times of my research. I would like to thank my Uncles (Kabuli Dada and Tuku Dada) and Aunts (Bina Khudi and Shanti khudi) for taking special care of my parents during my stay outside my hometown. Thanks to my cousins Runu Apa, Bhupen Nana, Aji Nana, Sanji Nana, Babuna Nana, Deepak, Dipu, Nandini Apa, Dudu, Puchi, Ruchi, Rima and the cute Jini. Finally, I would like to extend my appreciation to my grandmother and my late grandfather for their love and affection throughout my life.

And last, but not least, I would like to thank Dr. Madhav Desai and Dr. Ashok Pradhan for motivating me to consider higher study without which I would never have thought of pursuing the doctoral degree.

Thank you all!

TABLE OF CONTENTS

CHAPTER Page

# LIST OF TABLES

TABLE Page

# LIST OF FIGURES

FIGURE Page

# CHAPTER I

# INTRODUCTION

## A. Background and Motivation

The rapid growth of VLSI technology has been made possible by continuous scaling of CMOS devices to the ever smaller dimensions. The result of this rapid growth is high performance and low cost Integrated Circuits (ICs). In 1965, Gordon Moore predicted that the number of transistors on a chip would double every eighteen months. During the last three decades, semiconductor industries have closely followed the technology scaling trend of Moore's law. Intel is one of the leading players in the microprocessor/IC design and manufacture, which has kept pace with Moore's prediction. However, with the advent of nanometer regime, it has been difficult to meet Moore's prediction. One of the major challenges in designing fast and complex ICs is the shrinking device dimensions, that give rise to drastic variations in the device and design parameters leading to severe variability of the chip performance in the nanometer regime. The sources of variabilities can be categorized as follows [1]:

1. Manufacturing/Fabrication sources of variations
2. Environmental sources of variations

### 1. Manufacturing/Fabrication Sources of Variations

Manufacturing sources of variabilities are caused by processing and mask imperfections and reliability related degradations [1]. Until now it was sufficient to model the sources of variations as die-to-die variations. These variation models assume no

---

This dissertation follows the style of *IEEE Transactions on VLSI Systems.*

variability of parameters within a die. Such cases can be analyzed using classic Monte Carlo or the worst-case analysis. However, in deep sub-micron technology, there are significant within-die variations of device and interconnect parameters [1]. Within-die variations are dependent upon both the fabrication process and on the implementation of the ICs. These parameters can be spatially correlated or independent of each other.

## 2. Environmental Sources of Variations

The environmental sources include variations due to power supply voltage, noise coupling among nets and temperature fluctuation. These variations can be characterized as probability distribution and analyzed using Monte Carlo or the worst-case analysis. The variations are of low time constant and can vary within fraction of seconds.

- Supply voltage can vary from a nominal value during the operation of the chip. Moreover, the voltage variation is non-uniform across the entire chip. Since, supply voltage affects the drain current, the gate delay would vary across the chip with the variation in supply voltage.

- Signal nets can be affected by rising and falling signals of the neighboring nets due to its capacitive coupling with the neighbors. The signal net that gets affected by switching of the neighboring nets is called the victim net and the neighboring nets are called the aggressor nets. Coupling noise significantly impact the propagation delay of a signal net depending upon the direction of switching of its aggressor nets. The coupling noise is calculated using Miller coupling factor (MCF). With shrinking feature size and higher clock frequency, the coupling noise poses a serious threat to the delay variability.

- Temperature can vary throughout the chip depending upon the workload. Tem-

perature affects the mobility of the electrons and holes; higher the temperature lower is the mobility. This in turn affects the propagation delay of the gates. Thus, the propagation delay of a chip would vary depending upon its temperature variations.

Another challenging issue in nanometer IC design is the increase of power consumption. Over the past few years, low power IC design has been an important focus of research and development. The increase in power consumption is mainly due to the rapid technology scaling, enormous integration capacity and the mounting active and leakage power consumption. Moreover, the parasitic capacitances increase with the increase in number of devices on every technology scaling. The charging and discharging of these additional parasitic capacitances leads to soaring amount of power dissipation. With the clock frequency in modern ICs at 4 Ghz, the power dissipation is expected to be even worse. Thus, careful power planning and low power design techniques need to be adopted to reduce the ever increasing power demand of the chip.

## 3. Power Components in CMOS Logic

The power consumption in CMOS logic is usually estimated by

$$P_{total} = \alpha C V^2 f + V I_{leakage} + P_{short\_circuit} \tag{1.1}$$

Equation 1.1 consists of three components, namely dynamic power due to switching of the CMOS devices, static power due to leakage current and short-circuit power from power supply. Dynamic power depends upon activity factor ($\alpha$), load capacitance ($C$), supply voltage ($V$) and the frequency of operation ($f$). It is also called the active power of the chip. The leakage power consists of (1) subthreshold leakage (2)

gate leakage (3) substrate and junction leakage. The short-circuit power of CMOS logic is due to switching of input and output of the logic gate from one state to the other. During the input/output transition both the PMOS and NMOS of the CMOS logic are on for a certain period of time, thereby connecting the supply and ground through a resistive path. The short-circuit power is higher when the difference in slope between the input and output transition of the CMOS logic is greater. Traditionally, dynamic power is the dominant component of the total power consumption. However, due to aggressive scaling, the leakage components are growing at a faster rate and can not be neglected in nanometer VLSI design. The short circuit components are higher for bad designs where there is larger disparity between input and output slope of the logic gates.

## B. Contribution

The dissertation deals with two key aspects of the modern IC design - variation and power dissipation. We develop design methodologies to address the two key issues for reliable circuit operation. The proposed work has been divided into three parts: (1) Clock buffer polarity assignment for power noise reduction (2) Elastic timing scheme for energy-efficient and robust performance (3) Discrete buffer and wire sizing for link-based non-tree clock network.

### 1. Clock Buffer Polarity Assignment for Power Noise Reduction

As supply voltage reduces with VLSI technology scaling, the circuit performance becomes increasingly vulnerable to power and ground noise. This work aims to reduce the clock induced power/ground noise by a fine grained polarity assignment on an existing buffered clock tree. We use three existing algorithms for polarity assignment:

(1) partitioning (2) 2-coloring on minimum spanning tree (3) recursive min-matching. The fine granularity of assignment implies that even a very small region usually contains opposite polarity as long as there are more than one clock buffers. By doing so, the clock-induced power noise can be reduced remarkably.

## 2. Elastic Timing Scheme for Energy-Efficient and Robust Performance

The critical concern in nanometer IC design is the need to deliver high performance given ever-diminishing power budget and significant variation effects. The conventional approach of using safety margin to guard against low probability timing errors consumes power continuously. One of the alternatives to conventional safety margin based approach is the Razor technology [2], that eliminates such power inefficiency by using error detection and correction scheme. However, the error correction requires stalling/flushing of the pipeline, thus, is not preferred in real-time systems or finite state machines (FSM) with feedback loops. In this work, we propose an elastic timing scheme that can correct timing errors without stalling or flushing the pipeline. The main idea of this scheme is dynamic speed boosting. During normal operation, the circuit works with relatively low power consumption. When a timing error is detected, a few parts of the circuits are temporarily switched to a faster speed so that the timing deficit due to error is compensated. In order to minimize the overhead of the speed boosting, we incorporate dynamic clock skew shifting into the elastic timing scheme. Speed boosting and skew shifting should be applied in such a way that the overall power/cost overhead is minimized. We formulate and solve this problem by mixed integer programming. The management complexity of the elastic timing scheme is moderate and therefore not difficult to handle in practice. This timing scheme can also tolerate multiple simultaneous timing errors.

3. Discrete Buffer and Wire Sizing for Link-based Non-tree Clock Networks

The growing complexity of IC design can be attributed to two key issues: variability and power. Clock network is a subcircuit that deeply involves both the challenges. Link-based clock network has drawn people's attention for its appealing tradeoff between variation tolerance and power overhead. In this work, we investigate optimizing link-based non-tree clock network via buffer and wire sizing. Most of the previous work on buffer and wire sizing are either based on the Elmore delay, which is inaccurate for evaluation of skew in modern technology or handle continuous sizing. We will focus on discrete buffer and wire sizing with accurate delay models. Unlike continuous sizing, the discrete sizing rarely depends upon sensitivity analysis, which is valid over very small changes. Moreover, it is difficult to get fine-grained control for highly discrete problems. Using accurate model is also too expensive in a large solution space and may take large amount of time to converge to a solution. Thus, discrete sizing using accurate delay model is a very difficult problem. This difficulty is due to both the discreteness and model complexity. We made the following contributions to solve this difficult yet important problem:

- Support vector machine (SVM) is explored to handle the complex delay model issue and provide guidance for discrete optimization in large design space.
- We propose a two-stage hybrid optimization approach, which can significantly reduce clock skew.
- In the core part of the optimization, we introduce a technique to convert an integer nonlinear programming problem to an integer linear programming formulation.

- To the best of our knowledge, this is the first work on discrete clock network sizing using accurate delay model.
- To the best of our knowledge, this is also the first work on sizing link-based non-tree clock network.

## C. Organization

The remainder of this dissertation is organized as follows. Chapter II discusses power/ground noise reduction for buffered clock tree by performing polarity assignment of the clock buffers. We detail three different algorithms used for polarity assignment. Chapter III includes an elastic timing scheme for energy-efficient and robust IC design. We describe the new timing error correction scheme and the boosting techniques used for on-line error correction. Chapter IV explains the discrete buffer and wire sizing scheme for link-based non-tree clock network. We present a two stage hybrid optimization scheme for buffer and wire sizing. It also includes the description of support vector machine (SVM) used for guidance to our sizing algorithm. Chapter V summarizes our conclusions.

# CHAPTER II

# CLOCK BUFFER POLARITY ASSIGNMENT FOR POWER NOISE REDUCTION

Power/ground noise is a major source of VLSI circuit timing variations. This work aims to reduce clock network induced power noise by assigning different signal polarities (opposite switchings) to clock buffers in an existing buffered clock tree. Three assignment algorithms are proposed: (1) partitioning, (2) 2-coloring on minimum spanning tree and (3) recursive min-matching. A post-processing of clock buffer sizing is performed to achieve desired clock skew. SPICE based experimental results indicate that our techniques could reduce the average peak current and average delay variations by **50%** and **51%** respectively.

## A. Introduction

When the supply voltage decreases with VLSI technology scaling, circuit performance becomes increasingly vulnerable to power/ground noise [3, 4, 5]. This problem is exacerbated by the increase in frequency and large gate count in the scaled technologies. Based on an estimation in [6], a $0.1V$ power noise may cause 80% inverter delay variation at $45nm$ technology. A main culprit of power noise is clock network which keeps drawing huge current frequently from the power supply network [7, 8, 9, 10, 11]. Power/ground noise is acute at the beginning of clock cycle when flip-flops and the gates are switching simultaneously. In order to reduce the clock-induced power noise, a few works [7, 8, 9, 11] attempt to avoid simultaneous flip-flop switchings through clock skew scheduling. The common approach is to spread the computation across the entire clock period so that the peak of the power/ground noise occurring at the beginning of the clock cycle is distributed across the entire clock period. In [7, 8], the

flip-flops are grouped into buckets that are switched at different times. However, such an approach suffers from the limitation that the flip-flops within the same bucket still switch at the same instant of time. Moreover, the approaches [7, 8], do not consider the effect of clock skew scheduling on current profiles of combinational logic. The work [9], uses a graph based clock scheduling approach to minimize the peak current, hence the power supply noise. In [11], a circuit optimization technique called skew spreading is used to schedule the clock arrival time at each flip-flops such that peak current is reduced. In addition to performing the clock skew scheduling, the approaches [9, 11] also consider the effect of clock scheduling on the current profiles of combinational logic. The skew scheduling approaches [7, 8, 9, 11] are restricted by timing constraints of the combinational logic. Since the effectiveness of these approaches is dependent upon the available slack in the application, the power/ground noise result is expected to vary across different applications.

The use of on-chip decoupling capacitor to suppress the power/ground noise has been discussed in [12, 13]. The main idea is to use the charge stored in the decoupling capacitor to supply the switching transients. In another similar approach [14], the authors use a stub to suppress the power/ground noise. They attach a quarter length stub to the power supply line of the LSI chip. It acts as a band eliminate filter and suppresses power supply noise for a designed frequency.

Besides the flip-flops, the switchings of clock buffers also contribute greatly to the clock-induced noise. In a clock network of an industrial ASIC design, there could be dozens of thousands of clock buffers [15]. A recent work [10] proposes to use different signal polarities on clock buffers so that the roughly simultaneous same-direction switchings are replaced by a mixture of opposite-direction switchings. Signal polarity refers to whether or not a signal switches in the same direction as the clock source. The main idea of [10] is illustrated in Figure 1. In Figure 1(a),

all buffers have the same signal polarity and therefore they have either simultaneous rising switches, which draw large current from power ($V_{dd}$) network, or simultaneous falling switches which draw large current from ground ($V_{ss}$) network. In contrast, the application of opposite polarities as in Figure 1(b) decreases current withdraw since only a half of the buffers draw current from $V_{dd}$ while the others draw from $V_{ss}$ at the same time. Please note that polarity assignment to a buffer is different from selecting between inverting or non-inverting type for the buffer, although these two are related. By using different types of flip-flops, positive-edge or negative-edge triggered, both signal polarities can be accommodated at flip-flops with hardly any impact to the original circuit design. For the example in Figure 2, when the polarity of clock signal $t2$ is reversed from Figure 2(a) to Figure 2(b), the circuit timing is not affected if flip-flop FF2 is changed from positive-edge triggered to negative-edge triggered.

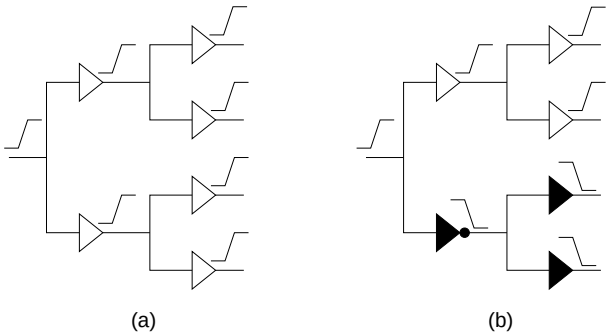


Fig. 1. All buffers in (a) have positive signal polarity and switch in the same direction. The dark buffers in (b) are assigned with negative polarity and switch in the direction opposite to the buffers with positive polarity.

When assigning polarities, the work of [10] partitions the clock sinks (flip-flops) into two subsets, one for positive polarity and the other for negative polarity. Then, two subtrees are constructed separately for the two subsets, i.e., one subtree has only positive polarity and the other subtree has only negative polarity. However, this

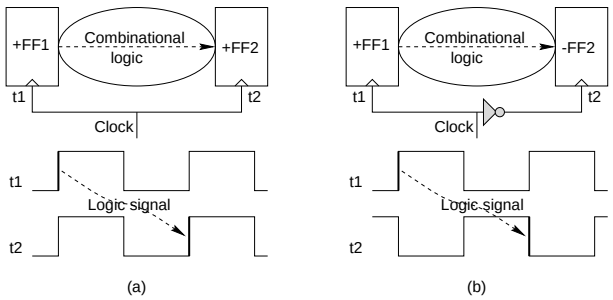


Fig. 2. (a) Positive-edge triggered FFs with original polarity (b) FF2 is changed to negative-triggered for the reversed polarity. The circuit timing is not affected if the skew t1 - t2 is maintained.

approach faces a dilemma considering the following two typical scenarios:

- If the two subsets are spatially separated from each other like in Figure 3(a), the two subtrees (one in solid lines and the other in dashed lines in Figure 3) are in two separated regions. Except the boundary region between the two subtrees, the power noise in a local area such as the shaded regions in Figure 3(a), is not reduced by the application of opposite polarities. This is because power noise is mostly a local effect.

- If the sink locations of the two subsets are intermingled, the approach of [10] results in two intermingled subtrees like Figure 3(b). In this scenario, the power noise in each local region can be reduced, but the wirelength of the clock network is increased greatly.

Therefore, constructing two subtrees independently [10] either is ineffective for reducing local power noise or suffers from huge wirelength overhead. Moreover, the work of [10] evaluates only the peak current while neither power supply voltage noise nor the impact on delay variation is discussed.

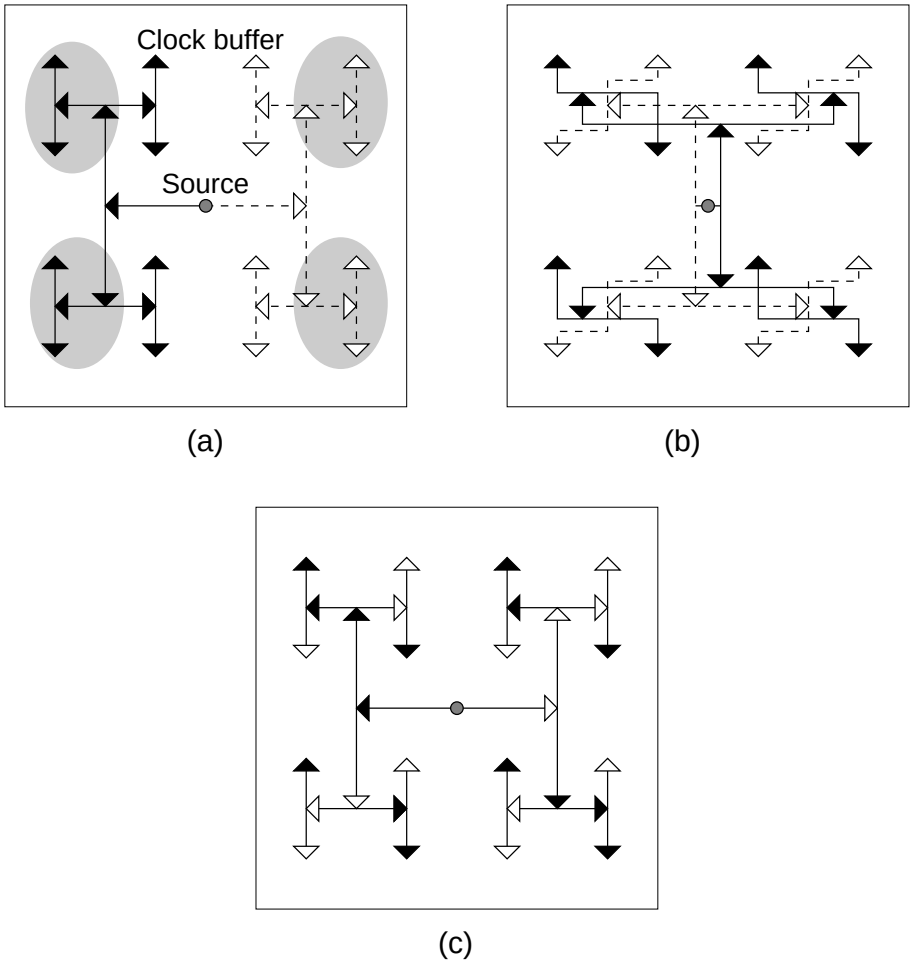


Fig. 3. Constructing two subtrees separately for opposite polarities either cannot reduce local power noise if the two subtrees are spatially apart like in (a), or results in huge wirelength overhead as in (b). We propose to perform fine–grained polarity assignment on an existing clock tree as in (c).

In this work, we propose to perform fine-grained clock buffer polarity assignment on an existing clock tree. We carry out a buffer type matching on the resulting clock tree to minimize the path unbalance that may arise due to buffer polarity assignment. Then, a clock buffer tuning is carried out to restore the clock skew altered by the polarity assignment. Three existing algorithms are used for polarity assignment: (1) partitioning, (2) 2-coloring on minimum spanning tree and (3) recursive min-matching. The fine granularity of the assignment implies that even a very small region usually contains opposite polarities as long as there are more than one clock buffers. By doing so, the clock-induced power noise can be reduced almost everywhere. Please note, our approach does not provide an alternative to the power noise reduction using clock skew scheduling [7, 8, 9, 11]. Our technique can be combined with these approaches to further improve the power noise results. Also, our technique complements the power noise reduction using the decoupling capacitors [12, 13, 14] and can reduce the stress on the decoupling capacitor network.

SPICE based experimental results indicate that our techniques could reduce the average peak current and average delay variations by 50% and 51% respectively.

## B. Impact to Delay Variation

Power/ground noise directly affects gate/buffer delay variation [5]. We present a first order analysis on the impact of clock buffer polarity assignment to gate/buffer delay variations. Without polarity assignment, i.e., with identical polarity for all clock buffers, all clock buffers have either simultaneous rising switchings, which cause decreased $V_{dd}$ and almost no disturbance to $V_{ss}$ (Figure 4(a)), or simultaneous falling switchings, which raise $V_{ss}$ but have negligible influence on $V_{dd}$ (Figure 4(b)). With polarity assignment, both $V_{dd}$ and $V_{ss}$ degrade but with less degree (Figure 4(c)).

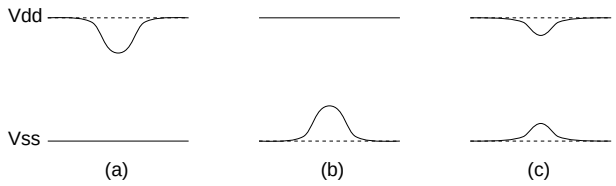


Fig. 4. Power noise in a local region when (a) all buffers have rising switches, (b) all buffers have falling switchings, and (c) half of the buffers rising while the others falling.

We define power noise $\Delta V_{dd}$ and ground noise $\Delta V_{ss}$ as

$$\Delta V_{dd} = \tilde{V}_{dd} - V_{dd} \qquad \Delta V_{ss} = \tilde{V}_{ss} - V_{ss}$$

where $V_{dd}$ and $V_{ss}$ are ideal voltage values, and $\tilde{V}_{dd}$ and $\tilde{V}_{ss}$ are the actual voltages considering noise. As in [5], the power/ground noise can be equivalently evaluated by *differential mode noise*

$$\Delta V_{dif} = \Delta V_{dd} - \Delta V_{ss}$$

and *common mode noise*

$$\Delta V_{com} = \Delta V_{dd} + \Delta V_{ss}$$

Table I. An example of power/ground noise for the three cases in Figure 4.

| Case in Figure 4 | $\Delta V_{dd}$ | $\Delta V_{ss}$ | $\Delta V_{dif}$ | $\Delta V_{com}$ |
|---|---|---|---|---|
| All rising (a) | -0.2 | 0 | -0.2 | -0.2 |
| All falling (b) | 0 | 0.2 | -0.2 | 0.2 |
| Half rising, half falling (c) | -0.1 | 0.1 | -0.2 | 0 |

In Table I, we list a rough numerical example of power/ground noise for the three cases in Figure 4. One can see that the polarity assignment does not change the differential mode noise but can reduce the common mode noise to nearly zero.

According to [5], the variation of rising delay and falling delay can be expressed as

$$\Delta t_{rise} = -A \cdot \Delta V_{com} - B \cdot \Delta V_{dif} \tag{2.1}$$

and

$$\Delta t_{fall} = C \cdot \Delta V_{com} - D \cdot \Delta V_{dif} \tag{2.2}$$

respectively, where $A, B, C$ and $D$ are all positive constants dependent upon device and technology parameters, the input transition time and the gate output load. The differential mode noise ($\Delta V_{dif}$) affects the delay to charge/discharge the capacitive load at the output of the gate. The larger the value of the differential mode noise faster is the charging and discharging of the capacitive load i.e. smaller is the gate delay. The common mode noise ($\Delta V_{com}$) contributes to modifying the effective switching threshold of the gate. For a positive common mode noise, the switching threshold of the $n/p$ transistors are higher than the threshold without noise i.e., the fall delay of the gate is larger with positive common mode noise. Similarly, for negative common mode noise the threshold of $n/p$ transistors are lower, thus, the rise delay of the gate is larger than without common mode noise. According to [5], delay variation of a gate is linearly dependent on differential mode and common mode noise. Thus, rise and fall delay variations are expressed as a linear combination of differential mode noise ($\Delta V_{dif}$) and common mode noise ($\Delta V_{com}$).

The three cases in Figure 4 result in approximately the same negative value of $\Delta V_{dif}$ which contributes to roughly the same amount of delay increase. The case of Figure 4(a) has more rising delay increase and less falling delay increase due to its negative common mode noise. Symmetrically, the case of Figure 4(b) has less rising delay increase and more falling delay increase. In contrast, the common mode noise from the case of Figure 4(c) is almost zero and therefore does not contribute

to the delay variation. We consider the example of Table I to find the worst case delay variation of a gate. For simplicity, $A$, $B$, $C$ and $D$ are assumed to be equal to 1. Substituting the values of $\Delta V_{dif}$ and $\Delta V_{com}$ for Figure 4(a) and Figure 4(b) in equation (2.1) or equation (2.2), the delay variation without polarity assignment:

$$\Delta t_{w/o,pol} = 0.2 + 0.2 = 0.4$$

The worst case delay variation of the gate due to polarity assignment can be found similarly by substituting values of $\Delta V_{dif}$ and $\Delta V_{com}$ for Figure 4(c) in equation (2.1) or equation (2.2):

$$\Delta t_{w/o,pol} = 0.0 + 0.2 = 0.2$$

Hence, clock buffer polarity assignment, which corresponds to Figure 4(c), can reduce the worst case delay variation compared to using identical polarity (Figure 4(b) and Figure 4(b)).

## C. Problem Formulation

Given a buffered clock tree with $n$ buffers, assign either positive or negative signal polarity to every buffer such that peak current reduction is maximized in any region of arbitrary size.

For a region including all of the clock buffers, this objective requires that roughly a half of the buffers have positive polarity and the others have negative polarity. For a small region containing only two clock buffers, this formulation requests one of them is positive and the other is negative.

## D. Polarity Assignment Algorithms

We have extended the application of three existing algorithms to solve the problem formulated in the previous section.

### 1. Partitioning

First, a graph $G = (V, E)$ is constructed with each node uniquely corresponding to a clock buffer and the node set $V$ covers all of the clock buffers. There is an edge between every pair of nodes, i.e., this is a complete graph. Then, a bi-partitioning [16] is performed on $G$ to partition $V$ into two disjoint subsets $V_+$ and $V_-$ such that $V = V_+ \cup V_-$ and $||V_+| - |V_-|| \leq 1$. The subsets $V_+$ and $V_-$ correspond to positive and negative polarities, respectively.

If two clock buffers are very close to each other, we prefer to separate them into different subsets (polarities). In a typical graph bi-partitioning [16], two nodes with a small edge weight in-between are more likely to be separated into two subsets. Thus, we let the weight of edge $(i, j)$ to be $d_{ij}$ which is the distance between node $i$ and $j$. Since a typical bi-partitioning algorithm minimizes the total weight of edges in the cut, an edge with small weight (or distance) has a large chance to be in the cut and its two end nodes are separated in different subsets. The complexity of bi-partitioning algorithm is $O(|V||E|)$ [16]. Since $G$ is a complete graph, the number of edges is proportional to $|V|^2$. Thus, complexity of bi-partitioning is $O(|V|^3)$.

### 2. 2-coloring on Minimum Spanning Tree

This is a very simple yet effective technique. First, a minimum spanning tree is generated for the nodes representing clock buffers. Again, each edge weight is defined as the distance between its two incident nodes. Then, a 2-coloring procedure is

applied on the minimum spanning tree. In 2-coloring, two end nodes of an edge are always assigned with different colors (or polarities). For a tree, there is always a feasible solution for 2-coloring and it can be found easily. Each color corresponds to a polarity. Since the minimum spanning tree algorithm chooses short edges, two nodes close to each other have opposite polarities. The minimum spanning tree is generated using greedy method. We iterate for $|V|-1$ times to construct the minimum spanning tree, where $|V|$ is the number of nodes in the graph $G$. During each iteration, we chose the shortest edge from the edges connected to the nodes of the spanning tree obtained in the previous iteration. Since graph $G$ is a complete graph, the total number of edges is proportional to $|V^2|$. Thus, the time complexity of choosing the shortest edge during each iteration is $O(|V|^2)$. Since we have $|V|-1$ iterations, the complexity of constructing the minimum spanning tree for the graph $G$ is $O(|V|^3)$. The 2-coloring on minimum spanning tree is $O(|E|+|V|)$ [17]. Thus, the complexity of polarity assignment due to 2-coloring on minimum spanning tree is $O(|V|^3)$.

### 3. Recursive Min-matching

A graph $G = (V, E)$ same as that in Section 1 is constructed. Performing min-matching (minimum weighted matching [18]) on this graph results in about $|V|/2$ matched node pairs. In a min-matching, the total weight of the edges between matched nodes is minimized among all possible matchings. Then, we force the two nodes (clock buffers) in the same pair to have opposite polarities. Since the min-matching algorithm normally selects pairs corresponding to small edge weight, the min-matching based polarity assignment tends to let two nearby buffers have opposite polarities.

However, requiring opposite polarities is not a complete assignment for a pair of buffers. For example, for a pair of clock buffers $(a, a')$, we can either let $a$ be positive

and $a'$ be negative (denoted as $(a_+, a'_-)$), or let $a$ be negative and $a'$ be positive $(a_-, a'_+)$. Both of the polarity permutations satisfy the constraint of being opposite. We denote the former as positive permutation $(a_+, a'_-)_+$ and the latter as negative permutation $(a_-, a'_+)_-$.

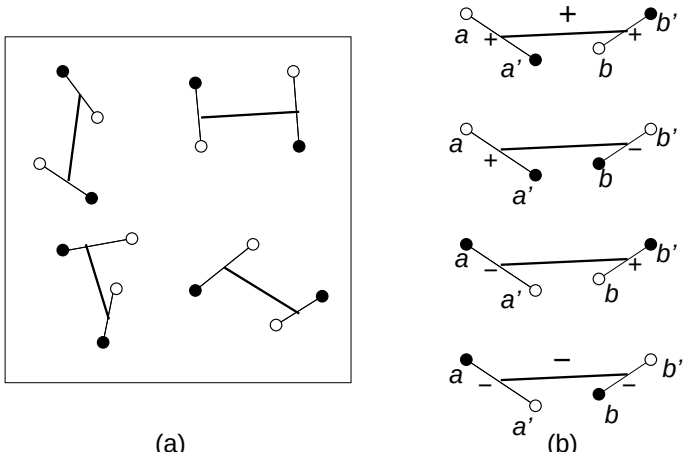


Fig. 5. Recursive min-matching.

The selection of polarity permutation is decided by performing another iteration of min-matching on the node pairs obtained in the first min-matching. In this iteration, the nodes of the graph is composed by the centroids of the node pairs matched in the previous iteration. Each edge weight is defined as the distance between corresponding centroids. If two node pairs $((a, a'), (b, b'))$ is selected to be matched in this iteration of min-matching, we have four different polarity permutations: (1) $((a_+, a'_-)_+, (b_+, b'_-)_+)$, (2) $((a_+, a'_-)_+, (b_-, b'_+)_-)$, (3) $((a_-, a'_+)_-, (b_+, b'_-)_+)$ and (4) $((a_-, a'_+)_-, (b_-, b'_+)_-)$. The notations for these permutations can be abbreviated as $++, +-, -+$ and $--$. These four cases are illustrated in Figure 5(b). It can be seen that $++$ and $--$ have no difference to the four clock buffers themselves. Similarly, $+-$ and $-+$ are equivalent to each other for the four buffers. But, $++$ and

$--$ are different from $+-$ and $-+$. For the example of Figure 5(b), it is obvious that permutation $++$ (at top) and $--$ (at bottom) are better than permutation $+-$ and $-+$ (in middle). Therefore, we choose $++$ and $--$ which are fully denoted by $((a_+, a'_-)_+, (b_+, b'_-)_+)_+$ and $((a_-, a'_+)_-, (b_-, b'_+)_-)_-$, respectively. The former is called positive permutation and the later is negative permutation. Now we have multiple node groups, each of which contains four nodes. The min-matching and polarity permutation selection can be repeated recursively on them till there is a single group containing all nodes.

We discuss how to decide polarity permutations for two matched node groups in general cases. Suppose two node groups $A$ and $B$ are matched. Each group has positive permutation $A_+$ and $B_+$ and negative permutation $A_-$ and $B_-$. We need to choose $++/--$ or $+-/-+$ for these two groups. Each polarity permutation can be evaluated by a score which is defined as follows. For a polarity permutation such as $++$, for each node $v$ in a group, we consider all nodes of the other group which are nearby, i.e., nodes within certain distance $D$ from $v$. If a nearby node of the other group has the same polarity as $v$, then the score of this polarity permutation is added by 1. Such score is counted for all nodes in one group. Finally, the polarity permutation with the smallest score is selected. For the example in Figure 5(b), the top $(++)$ and bottom $(--)$ permutations have score of 0 while the middle permutations ($+-$ and $-+$) have score of 1. The algorithm of the min-matching based polarity assignment is summarized in Figure 6.

The complexity of the minimum weighted matching algorithm is $O(|V|^3)$ [18], where $|V|$ is the number of nodes in the graph $G$. The recursive min-matching algorithm is called $|V|/2$ times to perform polarity assignment to each clock buffer. Thus, the complexity of the recursive min-matching algorithm is $O(|V|^4)$. During each iteration of the min-matching algorithm the number of nodes reduces to half than its

previous iteration. Thus, the actual number of nodes for most of the iterations is much smaller than $|V|$. Thus, the run time for the recursive min-matching algorithm is usually much faster than $O(|V|^4)$. We found the CPU run time for s35932 to be 0.603 sec, which is reasonable considering the size of the circuit. We implemented the data-structure in such way that, during each iteration the informations about the pair of matched nodes are stored in new memory location. The motivation behind this implementation is to reduce the execution time of the recursive algorithm. For this purpose, we need $2|V|$ extra memory nodes for the complete execution of the algorithm. The spatial complexity is acceptable considering the negligible run time of the recursive min-matching algorithm.

### E. Buffer Type Selection and Post Processing

After buffer signal polarity assignment, we need to choose either inverting or non-inverting type for each clock buffer. This procedure is straightforward. If a buffer has the same polarity as its parent buffer, it should use non-inverting type. Otherwise, an inverting type is applied.

In traditional clock tree designs, people prefer to use the same number of buffers on each source-sink path and use the same buffer type at each level [19]. This is illustrated in Figure 7(a). Such design can make clock skew robust to inter-die process variations. However, our buffer polarity assignment may result in different buffer types at a specific level (see Figure 7(b)). Therefore, we try to match the buffer types without affecting signal polarity in a post processing. After the buffer type matching, buffer sizing is performed to restore the original clock skew. Both the buffer type matching and buffer sizing are focused on flip-flops which are sequentially adjacent[1],

[1]A pair of flip-flops are sequentially adjacent if there is a pure combinational logic path in-between.

**Subroutine:** $Polarity\_Assignment(G)$

**Input:** Graph $G = (V, E)$

**Output:** Polarity assignment for each node $v \in V$

1. If $|V| \leq 1$, return
2. A set of node pairs $P \leftarrow Min\_Matching(G)$
3. For each pair $(u, v) \in P$
4. $p \leftarrow$ centroid of $(u, v)$
5. Select between $(u_+, v_+)/(u_-, v_-)$, $(u_+, v_-)/(u_-, v_+)$
6. If $(u_+, v_+)/(u_-, v_-)$ is selected
   $p_+ \leftarrow (u_+, v_+)$
   $p_- \leftarrow (u_-, v_-)$
7. Else
   $p_+ \leftarrow (u_+, v_-)$
   $p_- \leftarrow (u_-, v_+)$
8. $V \leftarrow V - \{u, v\} + \{p\}$
9. $E \leftarrow$ all pairs in $V$
10. $Polarity\_Assignment(G)$

Fig. 6. Algorithm of recursive min-matching based polarity assignment.

because the fundamental timing constraints - setup time and hold constraints, are mainly for sequentially adjacent flip-flops.

### 1. Buffer Type Matching

The buffer type matching is performed for a pair of sequentially adjacent flip-flops at a time. For such a pair $i$ and $j$, we check the paths from the source to $i$ and $j$. If there is any buffer type mismatch between the two paths at any level (like level 3 for $i$ and $j$ in Figure 7(b)), we swap the type of one mismatched buffers with the type of a nearby buffer while we try to maintain the signal polarity distribution unchanged. This procedure is repeated for every pair of sequentially adjacent flip-flops.

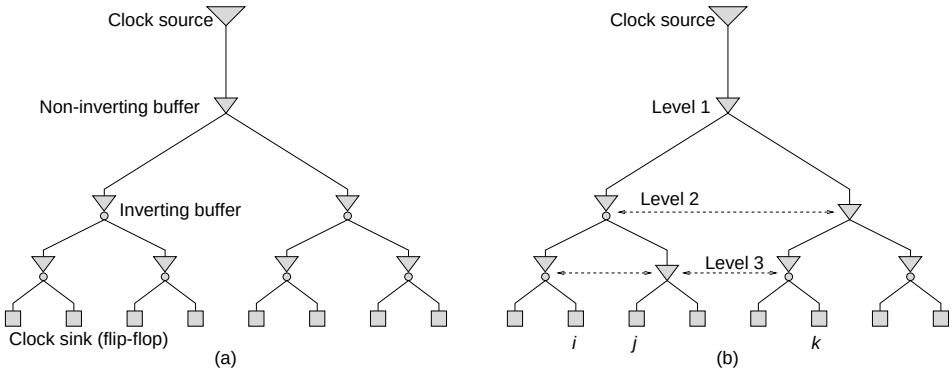


Fig. 7. (a) Original clock tree. (b) Buffer type mismatch occurs at level 2 and level 3 after polarity assignment.

The overall buffer type mismatch is evaluated by a mismatch-score defined as

$$MismatchScore = \sum_{i=1}^{n} criticality(i,j) \cdot diff(i,j)$$

where $n$ is the total number of sequentially adjacent pairs, $criticality(i,j)$ is the criticality between the sink pair ($i$,$j$) and $diff(i,j)$ is the number of buffer types mismatches between the paths from sinks $i$ and $j$ to the root of the tree. For exam-

ple, the $diff(i,j)$ and $diff(j,k)$ values for the clock tree of Figure 7(b) is 1 and 2 respectively.

The clock skew between flip-flop $i$ and $j$ has to be within a permissible range $[L_{ij}, U_{ij}]$ to satisfy setup time and hold time constraint. The size of the permissible range is represented by $P_{ij} = U_{ij} - L_{ij}$. The distance between $i$ and $j$ is denoted as $D_{ij}$. Then, the criticality for the pair $i$ and $j$ is estimated by [20]:

$$Criticality(i,j) = \alpha(\frac{P_{min}}{P_{ij}}) + (1-\alpha)(\frac{D_{ij}}{D_{max}})$$

where $\alpha \in (0,1)$ is the weight for permissible range, $P_{min}$ is the minimum permissible range among all flip-flop pairs and $D_{max}$ is the maximum distance among all pairs. This formula is based on the fact that the skew between a pair of flip-flops is critical if they have a small permissible range and/or they are far apart.

The algorithm for buffer type matching proceeds as follows. Initially, we calculate the most critical sink pair $i$ and $j$ that has the maximum product of $criticality(i,j)$ and $diff(i,j)$. The path from the two sinks to the root of the tree is is traversed in a bottom-up manner and the buffers at each level are compared subsequently. If a mismatch of the buffer type is found, the nearest neighbor of each buffer type is located. The mismatch-scores of the two buffers i.e. $MS(i)$ and $MS(j)$, are calculated assuming that the buffer has been swapped with its neighbor. The mismatch-scores of buffers are compared with the target mismatch-score ($MS$). The target mismatch-score is calculated once before the algorithm starts and it gets updated each time a buffer pair is swapped. If one or both of the mismatch-scores ($MS(i)$, $MS(j)$) are smaller than the target scores then, the buffer pairs that minimizes the target score by greater amount is selected. If the selected buffer pairs on swapping maintain the initial polarity distribution intact, we qualify the buffers for swapping. A pair of buffers can retain the initial polarity distribution only if they have opposite polarities before

swapping, then swapping of buffers would also cause a swapping of the polarities. Thus the initial polarity distribution is retained. However, if the swapped buffered nodes have same polarity, then the swap would cause both the polarities to invert and thus alters the initial polarity distribution. The swapping of buffer also affects polarity distribution of the subtree with the swapped buffers as its root. Thus we traverse from each swapped buffer in a top-down fashion and invert the buffer type of each buffer node in the subtree to maintain the original polarity.

If a buffer pair is qualified for swapping, we update the target mismatch-score with the updated scores i.e. either $MS(i)$ or $MS(j)$, and start over again by finding the critical most sink. On the other hand, if none of the buffer pair is qualified for swapping, then we continue the bottom-up traversal till next mismatch in the buffer type is found or the root node is encountered. If we reach the root then the next critical sink pair is selected and the same process is repeated for the selected sink pairs. If all the sink pairs are exhausted, then no further improvement in mismatch-score is possible and the algorithm stops.

### 2. Clock Skew Tuning

Since the original clock skew is changed due to the buffer type change in the polarity assignment, we run a clock skew tuning procedure after the buffer type matching to restore the original clock skew. This tuning procedure is same as [21] where the sizes of dummy capacitors are tuned toward desired clock skew. Therefore, the wirelength is not affected in this tuning. Although the area of dummy capacitance is increased, the buffer capacitance is often reduced when non-inverting buffers are replaced by inverting buffers in the polarity assignment. Hence, the overall capacitance is rarely increased as indicated in the experimental results.

## F. Experimental Results

The proposed procedure for power noise reduction was implemented in C on a Linux machine with 2 dual-core Intel Xeon processors of 3.2GHz and 8GB RAM. We performed experiments on two sets of benchmark circuits (a) ISCAS89 sequential circuits and (b) r1-r5 downloaded from GSRC Bookshelf [22]. The reason we employ ISCAS89 benchmark is that it has logic information and the reason for r1-r5 benchmark is that it is larger in size. The characteristics of the test cases are shown in Table II. The table indicates the number of clock sinks and the buffers for each test case.

For ISCAS89 benchmark circuits, the combinational logic gates were synthesized in Design Analyzer from Synopsys and placed using Silicon Ensemble from Cadence. The placement of the logic gates were done for 180nm library downloaded from the website [23]. The clock tree is then constructed using DME [24] and the clock buffers are placed similar as [19]. The logic gates were replaced by time-varying current sources connected between power and ground at grid points determined from the placement result of Silicon Ensemble. We replace the logic gates by current sources to make the power grid simulation feasible. The clock buffers are connected to the power and ground grids at locations determined from the clock tree construction. For SPICE simulation, we used 180nm model card obtained from [25] and $V_{dd}$ was set to 2.5V. We chose 180nm model card for SPICE simulation to maintain consistency between placement result and the SPICE simulation.

r1-r5 benchmark circuits were obtained from [22]. The buffered clock tree was generated using the algorithm in [21]. Since r1-r5 does not have logic gate information, we conducted experiments on a standalone clock tree. For SPICE simulation, we used 65nm BSIM4 model card obtained from [25] and set $V_{dd}$ to 1.0V.

Table II. Characteristics of testcases

| Case | # Sinks | # Buffers |
|---|---|---|
| S9234 | 135 | 20 |
| S5378 | 164 | 25 |
| S13207 | 503 | 77 |
| S38584 | 1426 | 235 |
| S35932 | 1728 | 286 |
| r1 | 267 | 37 |
| r2 | 598 | 171 |
| r3 | 861 | 59 |
| r4 | 1903 | 303 |
| r5 | 3101 | 441 |

To measure the effectiveness of our technique, we perform simulations to determine the peak current, power supply noise, delay variation, power consumption, total capacitance and global skew. We measure the above mentioned parameters for base case (initial clock tree with no polarity assignment), previous work [10] and the three algorithms proposed. For each parameters, we insert several sampling points in the circuit to measure the value during SPICE transient simulation. We record the worst case at each sampling point. For power noise, peak current and power consumption parameters, the sampling points are on the power grid. The sampling points for skew measurement is set at the sink locations. For delay variation, we introduce few logic gates into our simulation structure. The logic gates are connected to power and ground grids at points selected randomly.

The results for ISCAS89 and r1-r5 benchmark circuits are summarized in the Tables on pages 28 through 32. Each table consists of set of 5 columns. The first set of columns presents the results of the base case. The second set presents the previous work [10] . This is followed by a set of three columns that present the the three proposed algorithms i.e. Partition, MST and Matching. In each table, we report either normalized average or average for both the benchmarks separately. The

final row in each table shows the normalized average or average for r1-r5 benchmark circuits, the other row with normalized average is for ISCAS89 benchmark. The normalized average is used to compare our procedure with the base case as well as the technique described in [10].

Table III. Results for peak current (mA)

| Case | Base Case | | Previous Work | | Partition | | MST | | Matching | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Avg | Max | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| s5378 | 50.2 | 108.1 | 42.2 | 87.0 | 21.9 | 47.3 | 24.1 | 51.7 | 23.1 | 52.9 |
| s9234 | 41.30 | 69.4 | 34.6 | 60.1 | 22.0 | 40.1 | 19.1 | 33.4 | 18.0 | 32.0 |
| s13207 | 127.2 | 222.3 | 115.6 | 210.0 | 79.4 | 140.1 | 62.3 | 109.8 | 69.0 | 130.0 |
| s35932 | 95.1 | 154.1 | 90.3 | 180.5 | 53.0 | 92.4 | 52.4 | 85.5 | 51.4 | 94.4 |
| s38584 | 87.5 | 144.9 | 73.9 | 122.9 | 51.1 | 89.3 | 44.6 | 72.0 | 50.2 | 88.6 |
| Nor Ave. | 1.00 | 1.00 | 0.87 | 0.88 | 0.55 | 0.56 | 0.50 | 0.50 | 0.51 | 0.53 |
| r1 | 15.9 | 24.5 | 12.4 | 20.2 | 7.9 | 11.9 | 8.7 | 14.3 | 8.5 | 13.4 |
| r2 | 43.8 | 82.1 | 35.1 | 71.1 | 26.1 | 46.3 | 25.8 | 45.7 | 26.1 | 48.1 |
| r3 | 21.6 | 37.0 | 20.4 | 37.5 | 11.2 | 20.0 | 12.0 | 20.4 | 11.2 | 28.5 |
| r4 | 80.7 | 156.4 | 70.5 | 138.4 | 46.2 | 85.9 | 40.7 | 74.8 | 40.7 | 77.4 |
| r5 | 111.9 | 156.7 | 94.6 | 132.7 | 64.2 | 90.1 | 63.6 | 90.8 | 60.8 | 85.0 |
| Nor Ave. | 1.00 | 1.00 | 0.85 | 0.88 | 0.55 | 0.54 | 0.54 | 0.54 | 0.54 | 0.53 |

Table IV. Results for power noise (mV)

| Case | Base Case | | Previous Work | | Partition | | MST | | Matching | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Avg | Max | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| s5378 | 42.2 | 92.8 | 39.8 | 83.8 | 21.5 | 48.8 | 23.1 | 46.6 | 22.7 | 45.8 |
| s9234 | 34.3 | 69.1 | 27.5 | 62.5 | 20.1 | 50.7 | 16.4 | 50.1 | 15.5 | 50.1 |
| s13207 | 170.0 | 247.8 | 149.2 | 239.0 | 97.5 | 143.1 | 80.9 | 125.6 | 91.2 | 91.2 |
| s35932 | 169.6 | 298.0 | 155.4 | 295.2 | 92.1 | 182.4 | 88.8 | 167.7 | 85.2 | 167.5 |
| s38584 | 140.0 | 255.0 | 114.0 | 219.4 | 76.4 | 183.1 | 67.2 | 143.1 | 73.9 | 159.2 |
| Nor Ave. | 1.00 | 1.00 | 0.87 | 0.92 | 0.55 | 0.63 | 0.50 | 0.57 | 0.51 | 0.60 |
| r1 | 6.2 | 10.1 | 5.4 | 9.2 | 3.0 | 4.4 | 3.4 | 4.9 | 3.2 | 4.3 |
| r2 | 15.6 | 25.7 | 12.8 | 23.2 | 10.0 | 16.5 | 10.0 | 16.1 | 10.2 | 17.7 |
| r3 | 9.6 | 13.4 | 8.8 | 13.0 | 5.0 | 8.1 | 5.6 | 8.9 | 5.2 | 7.6 |
| r4 | 36.5 | 51.7 | 30.2 | 43.3 | 22.5 | 32.8 | 18.8 | 27.2 | 18.5 | 26.0 |
| r5 | 61.4 | 79.1 | 53.4 | 75.4 | 32.4 | 43.8 | 43.8 | 30.5 | 41.2 | 42.7 |
| Nor Ave. | 1.00 | 1.00 | 0.86 | 0.91 | 0.56 | 0.57 | 0.55 | 0.56 | 0.54 | 0.54 |

Table V. Results for ground noise (mV)

| Case | Base Case | | Previous Work | | Partition | | MST | | Matching | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Avg | Max | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| s5378 | 44.8 | 83.6 | 39.7 | 78.2 | 20.3 | 53.0 | 22.5 | 42.4 | 21.1 | 42.5 |
| s9234 | 31.3 | 63.2 | 27.4 | 60.1 | 19.5 | 39.6 | 14.5 | 39.3 | 13.4 | 39.4 |
| s13207 | 156.3 | 227.8 | 147.2 | 227.3 | 104.1 | 156.1 | 79.0 | 122.1 | 85.1 | 129.0 |
| s35932 | 154.0 | 295.0 | 152.0 | 290.0 | 87.7 | 174.5 | 88.1 | 176.7 | 87.8 | 184.9 |
| s38584 | 128.1 | 245.4 | 110.1 | 193.9 | 77.4 | 168.9 | 69.1 | 140.2 | 79.2 | 168.3 |
| Nor Ave. | 1.00 | 1.00 | 0.91 | 0.93 | 0.58 | 0.64 | 0.51 | 0.56 | 0.52 | 0.60 |
| r1 | 7.1 | 10.8 | 6.1 | 10.3 | 3.2 | 4.5 | 3.6 | 5.1 | 3.9 | 4.7 |
| r2 | 18.5 | 30.3 | 14.6 | 27.0 | 10.4 | 17.8 | 10.1 | 16.9 | 10.3 | 17.8 |
| r3 | 10.6 | 15.3 | 10.2 | 15.0 | 5.6 | 9.0 | 5.8 | 9.2 | 5.4 | 8.3 |
| r4 | 39.8 | 56.4 | 33.8 | 51.3 | 21.4 | 30.8 | 19.6 | 28.4 | 19.9 | 28.6 |
| r5 | 58.5 | 78.2 | 53.7 | 72.4 | 31.9 | 42.9 | 30.0 | 40.5 | 30.5 | 41.5 |
| Nor Ave. | 1.00 | 1.00 | 0.87 | 0.93 | 0.52 | 0.54 | 0.52 | 0.53 | 0.53 | 0.52 |

Table VI. Results for delay variation (ps)

| Case | Base Case | | Previous Work | | Partition | | MST | | Matching | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Avg | Max | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| s5378 | 0.72 | 0.96 | 0.52 | 0.73 | 0.33 | 0.36 | 0.28 | 0.30 | 0.25 | 0.29 |
| s9234 | 0.50 | 0.70 | 0.39 | 0.57 | 0.35 | 0.45 | 0.31 | 0.42 | 0.33 | 0.43 |
| s13207 | 1.60 | 2.10 | 1.40 | 2.00 | 1.05 | 1.20 | 0.80 | 0.91 | 0.81 | 0.93 |
| s35932 | 3.30 | 3.60 | 3.10 | 3.30 | 1.62 | 1.81 | 1.51 | 1.70 | 1.43 | 1.60 |
| s38584 | 2.80 | 2.90 | 2.56 | 2.71 | 1.55 | 1.70 | 1.41 | 1.59 | 1.50 | 1.70 |
| Nor Ave. | 1.00 | 1.00 | 0.84 | 0.87 | 0.57 | 0.53 | 0.49 | 0.46 | 0.50 | 0.48 |
| r1 | 0.41 | 0.48 | 0.38 | 0.40 | 0.21 | 0.24 | 0.19 | 0.23 | 0.22 | 0.25 |
| r2 | 1.29 | 1.33 | 1.23 | 1.32 | 0.75 | 0.77 | 0.80 | 0.83 | 0.76 | 0.78 |
| r3 | 0.53 | 0.6 | 0.42 | 0.47 | 0.28 | 0.33 | 0.27 | 0.32 | 0.25 | 0.30 |
| r4 | 2.00 | 2.16 | 1.95 | 2.10 | 1.05 | 1.15 | 1.02 | 1.06 | 1.10 | 1.21 |
| r5 | 2.98 | 3.24 | 2.71 | 2.95 | 1.54 | 1.62 | 1.65 | 1.80 | 1.70 | 1.82 |
| Nor Ave. | 1.00 | 1.00 | 0.91 | 0.90 | 0.53 | 0.53 | 0.53 | 0.53 | 0.54 | 0.54 |

In the data Tables III, IV, V, VI and VIII , we report the average and the maximum results among these worst case values from different sampling points. In the data Table IX, we report the resource consumption for both benchmark circuits. For each of the 5 cases in Table IX, we report average value of the total power consumption, total capacitance. The total capacitance is the sum of tuning capacitance and

Table VII. Results for nominal skew (ps)

| Case | Base Case | Previous Work | Partition | MST | Matching |
|---|---|---|---|---|---|
| s5378 | 4.9 | 15.7 | 16.0 | 17.7 | 20.0 |
| s9234 | 4.0 | 14.1 | 22.0 | 18.6 | 19.0 |
| s13207 | 10.2 | 12.4 | 15.5 | 18.3 | 19.8 |
| s35932 | 30.0 | 33.0 | 35.0 | 25.0 | 35 |
| s38584 | 28.0 | 35.0 | 29.0 | 35.0 | 32.0 |
| Ave. | 15.4 | 22.0 | 23.5 | 22.9 | 25.2 |
| r1 | 5.4 | 7.2 | 9.6 | 9.0 | 10.2 |
| r2 | 31.5 | 31.4 | 35.2 | 31.5 | 35.0 |
| r3 | 20.0 | 18.5 | 19.0 | 14.1 | 18.6 |
| r4 | 12.5 | 27.2 | 19.4 | 22.8 | 19.3 |
| r5 | 12.8 | 12.4 | 11.7 | 19.6 | 17.8 |
| Ave. | 16.5 | 19.3 | 19.0 | 19.4 | 20.2 |

Table VIII. Results for skew due to variation for ISCAS89 (ps)

| Case | Base Case | | Previous Work | | Partition | | MST | | Matching | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Avg | Max | Avg | Max | Avg | Max | Avg | Max | Avg | Max |
| s5378 | 48.1 | 93.6 | 50.0 | 94.6 | 46.4 | 84.8 | 50.1 | 92.0 | 49.8 | 90.5 |
| s9234 | 5.5 | 6.8 | 17.9 | 19.0 | 19.2 | 20.1 | 18.7 | 19.2 | 19.0 | 19.4 |
| s13207 | 76.2 | 116.8 | 72.7 | 105.1 | 76.0 | 115.5 | 75.4 | 109.2 | 72.9 | 106.1 |
| s35932 | 184.6 | 265.0 | 173.4 | 278.9 | 181.1 | 265.0 | 166.5 | 248.7 | 183.5 | 267.0 |
| s38584 | 129.5 | 182.4 | 173.4 | 193.1 | 133.2 | 199 | 123.5 | 177.8 | 136.4 | 189.3 |
| Ave. | 88.8 | 139.9 | 90.4 | 138.1 | 91.2 | 136.9 | 86.9 | 129.4 | 92.3 | 134.5 |

buffer capacitance. The CPU run time is reported in Table X. For CPU run time, we include the CPU run time to generate SPICE files for each of the algorithm and does not include the SPICE run time.

We post process the clock tree obtained (after assigning different polarities) to tune the skew by techniques suggested in [21]. By doing that, we bring the skew to be less than the required skew bound for all the test cases. The skew bound was set to $35psec$ for all the test cases. The skew results are reported in Table VII.

The skew due to variation was determined for ISCAS89 benchmark circuits. The result for skew was obtained by running 1000 Monte Carlo simulations for each case. The following parameters are varied (a) channel Length (b) threshold voltage and (c)

Table IX. Results for total resource consumption (power in mW and cap in pf)

| Case | Base Case | | Previous Work | | Partition | | MST | | Matching | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Pow | Cap | Pow | Cap | Pow | Cap | Pow | Cap | Pow | Cap |
| s5378 | 38.2 | 7.72 | 36.4 | 7.59 | 23.7 | 4.62 | 26.6 | 5.11 | 27.4 | 5.31 |
| s9234 | 31.8 | 6.10 | 30.2 | 5.71 | 25.0 | 4.86 | 22.3 | 3.81 | 21.0 | 3.46 |
| s13207 | 117.8 | 56.0 | 105.6 | 54.7 | 94.6 | 50.5 | 88.5 | 47.6 | 88.2 | 46.1 |
| s35932 | 526.0 | 129.2 | 548.0 | 157.3 | 480.0 | 97.9 | 443.6 | 130.0 | 471.0 | 127.2 |
| s38584 | 394.9 | 90.00 | 409.2 | 107.8 | 330.3 | 67.40 | 294.0 | 82.40 | 313.6 | 73.70 |
| Nor Ave. | 1.00 | 1.00 | 0.97 | 1.06 | 0.79 | 0.76 | 0.75 | 0.81 | 0.76 | 0.77 |
| r1 | 1.4 | 0.41 | 1.4 | 0.52 | 1.1 | 0.42 | 1.1 | 0.42 | 1.0 | 0.46 |
| r2 | 6.1 | 1.90 | 6.4 | 3.34 | 4.4 | 1.84 | 4.5 | 2.13 | 4.3 | 1.81 |
| r3 | 2.7 | 0.75 | 2.8 | 1.01 | 2.0 | 0.84 | 2.2 | 0.95 | 2.1 | 0.84 |
| r4 | 10.6 | 3.35 | 10.7 | 4.85 | 7.8 | 3.74 | 8.2 | 5.21 | 7.5 | 5.13 |
| r5 | 14.0 | 4.90 | 14.9 | 6.10 | 10.5 | 5.34 | 10.9 | 5.77 | 10.0 | 5.13 |
| Nor Ave. | 1.00 | 1.00 | 1.04 | 1.41 | 0.75 | 1.06 | 0.78 | 1.22 | 0.73 | 1.14 |

temperature. The above parameters were varied with mean as nominal value and a standard deviation of 3%. In Table VIII, we report the average and maximum values of the skew due to variation.

The following observations could be drawn from the results:

- Our techniques clearly dominate the method suggested in [10] in terms of peak current, power supply noise, delay variation and power consumption.

- The reduction in peak current is significant, 46%-50% and 45%-47% respectively for ISCAS89 and r1-r5 benchmark circuits. In fact, in few cases it could lead up to more than 50% peak current reduction. Such high reductions in peak current have a direct positive impact on circuit reliability.

- The power supply noise and ground noise come down by 45-50% (45-46%) and 42-49% (47-48%) respectively for ISCAS89 (r1-r5) benchmark. For s38584, a power supply noise reduction of 52% is achieved. This result indicates that our algorithm is efficient in reducing power supply noise for even bigger circuits.

Table X. Results for CPU time (sec)

| Case | Base Case | Previous Work | Partition | MST | Matching |
|---|---|---|---|---|---|
| s5378 | 0.200 | 0.210 | 0.253 | 0.275 | 0.245 |
| s9234 | 0.190 | 0.195 | 0.240 | 0.256 | 0.245 |
| s13207 | 0.249 | 0.301 | 0.320 | 0.331 | 0.315 |
| s35932 | 0.545 | 0.712 | 0.745 | 0.81 | 0.803 |
| s38584 | 0.539 | 0.695 | 0.701 | 0.750 | 0.719 |
| Nor Ave. | 1.00 | 1.17 | 1.29 | 1.38 | 1.32 |
| r1 | 0.340 | 0.350 | 0.480 | 0.460 | 0.440 |
| r2 | 0.615 | 0.750 | 0.723 | 0.810 | 0.796 |
| r3 | 0.450 | 0.510 | 0.535 | 0.565 | 0.490 |
| r4 | 0.645 | 0.754 | 0.812 | 0.843 | 0.821 |
| r5 | 1.400 | 1.560 | 1.610 | 1.730 | 1.712 |
| Nor Ave. | 1.00 | 1.13 | 1.23 | 1.29 | 1.23 |

- The delay variation reduces by 43-51% and 46-47% respectively for ISCAS89 and r1-r5 benchmarks. The impact of our algorithm for delay variation reduction is higher for the bigger clock networks. This trend is encouraging as it indicates that our algorithm scales favorably for bigger nets.

- The total power consumption reduces by 21-25% (22-27%) for the three different algorithms. The power reduction can be attributed to the use of different buffer types in the polarity assigned clock tree.

- The total capacitance reduction is in the range 19% - 24% for ISCAS89 benchmark. However, for r1-r5 the capacitance increases in the range 6%-22%. The increase in capacitance for our case in r1-r5 is due to smaller buffer sizes used for 65nm technology. Thus the tuning capacitance is more than the total buffer capacitance.

- The nominal skew values for three different algorithms are reduced to that of the base case. The skew bound is set to 20ps for small and medium sized testcases,

the skew bound is 35ps for the large test cases. The only exception is for $r_2$ testcase. Since the skew of initial tree is poor for $r_2$, we chose the skew to be 35ps instead of 20ps bound.

- The skew due to variation is calculated for ISCAS89 benchmark circuits. The average and maximum value of skew in our case is almost same as that for the base case. Thus the proposed technique has similar variation tolerance as the base case. For few testcases our algorithm is found to reduce the skew due to variation compared to base case. As an examples for s35932, the average skew due to variation for MST algorithm is reduced by 10% compared to the base case. Thus, in addition to reducing peak current and power supply noise, in few cases, our algorithm can also provide more robustness to skew due to variation.

- Since the run-time is negligible, our technique offers the flexibility of trying all three approaches and picking the one that offers the best results.

# CHAPTER III

# ELASTIC TIMING SCHEME FOR ENERGY-EFFICIENT AND ROBUST PERFORMANCE

In this work, we propose an elastic timing scheme which can correct timing errors without stalling/flushing of the pipeline. This is achieved by dynamically boosting circuit speed and dynamic clock skew shifting when a timing error occurs. We formulate an optimization algorithm to minimize the cost overhead of boosting and skew shifting. Compared to conventional safety margin based approach, the elastic timing scheme can reduce power dissipation by $20\% - 27\%$ for ISCAS89 sequential circuits while retaining similar variation tolerance.

## A. Introduction

When the VLSI technology scales to $65nm$ and beyond, the endeavor for performance growth is seriously hampered by the fundamental limit on power density. Even worse, variation effects such as process, voltage and temperature variations, are increasingly significant and consequently entail extra timing and power budgets.

Conventionally, the variations are handled by guarding nominal circuit delay with timing safety margins. Because of the uncertainty in variations, designers have to use pessimistically large safety margins such that circuits are guaranteed to work properly under the worst case variations. Such over-design not only makes timing closure difficult but also causes excessive power dissipation. On one hand, the likelihood of the worst case or near-worst-case variations is very small. On the other hand, the power for maintaining the safety margins is consumed almost continuously. Evidently, the efficiency of such power usage is poor. Recently, statistical methods [26, 27, 28, 29] have been developed to reduce the pessimism of safety margins.

However, large portions of the safety margins are still retained to guard against at least the near-worst-case variations. Adaptive design techniques [30, 31] can compensate manufacturing process variations, but are not good at handling runtime dynamic variations such as supply voltage fluctuations.

The Razor technology [2] is a breakthrough work that largely eliminates the power inefficiency of safety margin based approaches. Instead of relying on safety margins, Razor achieves variation tolerance through in-situ timing error detection and correction. As a result, power is spent for the worst case or near-worst-case only when it occurs. The power overhead of the error detection and correction is considerably less than the power savings from the safety margin reduction. When correcting a timing error, the Razor pipeline has to be stalled and often flushed [2] via architectural approach. In real-time systems, however, pipeline stalls should be avoided as much as possible. For general sequential circuits with feedback loops, such as finite state machine, it is not obvious how to perform the pipeline flushing.

In this work, we propose an elastic timing scheme that can correct timing errors without stalling or flushing the pipeline. The key technique of this scheme is dynamic speed boosting. In normal operations, the circuit works with relatively low power consumption. When a timing error is detected, a few parts of the circuit are temporarily switched to a faster speed such that the timing deficit due to the error is compensated. In order to minimize the overhead of the speed boosting, we incorporate dynamic clock skew shifting into the elastic timing scheme. Speed boosting and skew shifting should be applied in such a way that the overall power/cost overhead is minimized. We formulate and solve this problem by mixed integer programming. The management complexity of the elastic timing scheme is moderate and therefore not difficult to handle in practice. This timing scheme can also tolerate multiple simultaneous timing errors. Compared to conventional safety margin based approach,

the elastic timing scheme can reduce power dissipation by $20\% - 27\%$ on ISCAS89 sequential circuits. At the same time, our approach retains similar variation tolerance as the conventional approach. Since the boosting technique is applied to only a small portion of each entire circuit, the overall area overhead is usually less than 5%. The short path constraint and metastability problem are handled in the same way as Razor [2].

## B. Background on Razor

Our idea of elastic timing scheme is inspired by Razor [2], which is a power-efficient technique for variation tolerance. The main idea of Razor is to restrain the power consumption to typical scenarios and handle the low probability timing errors with dynamic corrections instead of relying on safety margins. Since variation-induced timing errors occur with low probability, the power spent on the error correction is much less than that of maintaining safety margins. An important component in implementing this idea is the Razor flip-flop, which is depicted in Figure 8(a). Here, a traditional flip-flop is protected by a shadow latch, which can catch signals missed by the flip-flop. The clock signal arrival time at the shadow latch can be either equal to or later than that to the main flip-flop [2]. The signal miss - or timing error - can be detected by comparing the flip-flop output and the latch output. The shadow latch entails tight short path constraint and requires delay buffers to slow down the short paths. However, the overhead of the delay buffers is limited. It is reported in [2] that the total chip power overhead due to Razor flip-flops and delay buffers is only 2.9%. The metastability issue is also discussed in [2].

There are two error correction schemes suggested in Razor [2]: (i) centralized pipeline recovery, where the error signal stalls the clock network and the pipeline while

the corrected logic signal is resent from the shadow latch; (ii) distributed pipeline recovery which flushes and restarts the entire pipeline. The inventors of Razor [2] pointed out that the centralized pipeline recovery is difficult to implement in practice since it is sometimes infeasible to deliver the error (stall) signal from a Razor flip-flop to the clock control gate within one clock cycle. The distributed pipeline recovery is a more practical approach, however, it may cause pipeline stall for several clock cycles and is therefore not ideal for use in real-time systems. Moreover, pipeline flushing is not obviously feasible in circuits with feedback loops, such as finite state machines.

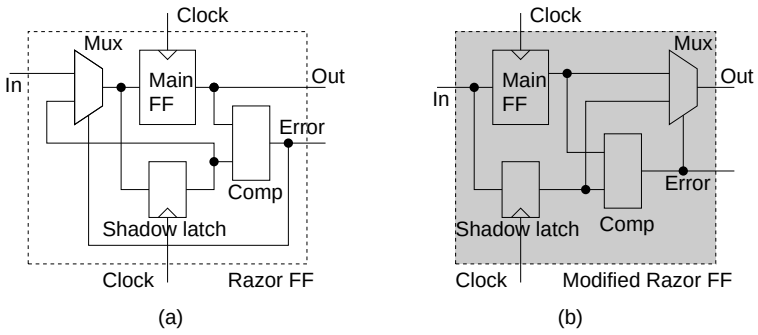


Fig. 8. (a) Razor flip-flop. (b) Modified Razor flip-flop.

## C. New Timing Error Correction Scheme

Our goal is to overcome the limitation of Razor so that its advantages can be utilized in general pipelined or sequential circuits. The focus of our approach is on a new timing error correction scheme - elastic timing scheme. The timing error detection part is inherited from Razor [2] which has already addressed the issues of delay padding for short paths and metastability. The key idea of the elastic timing is to let the correct signal directly chase after the incorrect signal so that the pipeline does not need to be stalled or flushed. In order to do so, we make a modification to the Razor flip-flop

structure as shown in Figure 8(b). After detecting a timing error, we re-send the correct logic signal to the output of the flip-flop instead of its input as was done in Razor. Then, the correct logic signal will be propagated through the combinational logic network in the same clock cycle as the incorrect logic signal which caused the timing error. Our intention is to let the corrected signal eventually overwrite the incorrect signal so that pipeline stall is no longer necessary.

## D. Dynamic Speed Boosting

In the proposed timing error correction scheme, the launch time of the corrected logic signal is later than usual because (1) the signal arrives at the shadow latch later than the active clock edge of the main flip-flop and (2) the signal is sent to the multiplexer output after the comparator delay. Therefore, its arrival time to the next flip-flop might be too late to be captured, i.e., the next flip-flop may still capture the incorrect logic signal. We propose to momentarily boost the speed of combinational logic circuit such that the late launch time is compensated. Since we do not use large safety margins, the saved timing budget allows the circuit to run at reduced speed and low power in normal operations. Hence, the circuit speed should have room for acceleration when timing errors occur. There are two options for speed boosting, which are introduced below and illustrated in Figure 9.

### 1. Dynamic Dual-$V_{DD}$

The circuit is supplied by a low $V_{DD}$ in normal operations and switched to a high $V_{DD}$ when timing errors occur. Conventional dual-$V_{DD}$ designs apply different $V_{DD}$ levels to different circuit blocks in a static manner. As long as the low $V_{DD}$ level is significantly higher than the threshold voltage of high $V_{DD}$ blocks, the circuit

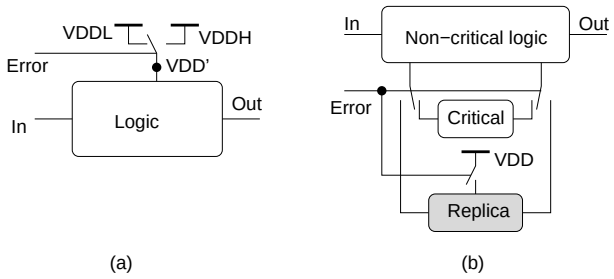


Fig. 9. Speed boosting by (a) dynamic dual-$V_{DD}$ and (b) dynamic fast lane.

functionality is not affected when a low $V_{DD}$ block directly drives a high $V_{DD}$ block. However, a low threshold voltage in high $V_{DD}$ blocks may cause high leakage current. Therefore, people use a relatively high threshold voltage for high $V_{DD}$ blocks and insert a level converting circuit between low $V_{DD}$ and high $V_{DD}$ domains to reduce the leakage power [32]. In our case, the level converting circuit is skipped because (1) it causes a delay penalty that degrades the effect of speed boosting, and (2) the power overhead at high $V_{DD}$ is very limited due to low probability of error occurrence. The switches between the power supply network and logic circuits can be implemented by sleep transistors as in power gating [33]. We performed SPICE simulation on ISCAS85 benchmark circuit C432 with the dynamic $V_{DD}$. The waveforms of supply voltage at $V'_{DD}$ of Figure 9 and the control ($\overline{Error}$) signal are shown in Figure 10. One can see that the switching between two different $V_{DD}$ can be carried out very quickly and smoothly. This issue is discussed in more details in Section F.

### 2. Dynamic Fast Lane

The timing critical part of the circuit is replicated and the replica can run at significantly faster speed compared to the original circuit. When a timing error is detected, the logic computation is switched from the original circuit to the fast replica. After

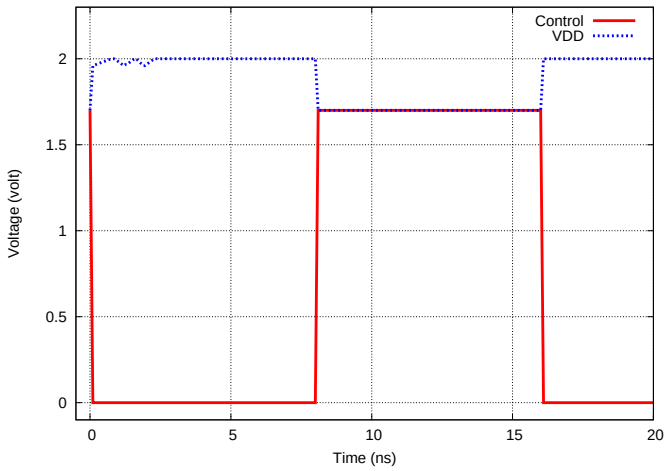


Fig. 10. The upper waveform is the dynamic supply voltage. The lower waveform is the control (error) signal that switches the $V_{DD}$ level.

the error is corrected, the computation is switched back to the original circuit. There are various techniques to obtain high speed for the replica: higher $V_{DD}$, lower $V_{th}$, forward body bias, larger and properly sized gates, or a combination of those. In normal operations, the replica idles and does not consume dynamic power. Its leakage power can be reduced by using power gating. However, the wake-up of power gated replica takes some time, often 2 or 3 clock cycles [33]. Therefore, only shallow sleep mode [33] can be applied for the power gating here so that the wake-up time is not too long. In addition, power gated replica should be structurally some distance away from where the error is detected, so that it has sufficient time to wake up before the corrected signal arrives. Although it seems that the area penalty of the replica approach is large, its overall impact to the entire chip area can be very limited if it is judiciously applied at only a few very critical places.

### 3. Power Reduction and Overhead

Among the above options for dynamic speed boosting, dynamic dual-$V_{DD}$ is relatively easy to use in practice and has small overhead. Although it requires two sets of power grid, the power routing overhead is not much different from that of distributed power gating [34] where a virtual power grid is built in addition to the original power grid. Since the boosting is applied to a small portion of an entire chip, the overall power routing overhead is quite limited. For the global level power delivery, the overhead due to dynamic dual-$V_{DD}$ is almost the same as that of existing dual-$V_{DD}$ designs [32]. The dynamic fast lane approach is more powerful on speed boosting as the replica circuit can be implemented with multiple acceleration techniques simultaneously. Evidently, the replica has relatively large area penalty. However, we apply the fast lane to only a very small portion of very critical circuit so that the overall cost overhead is still limited.

## E. Shared Boosting via Dynamic Skew Shifting

We strive for minimizing the area and power delivery overhead of the dynamic speed boosting. A key observation is that we do not need to make every logic stage boostable even if *every* logic stage has small timing slack. A boosting can be shared by multiple stages because an error does not have to be corrected immediately after it is detected. As long as the error can be corrected before it is propagated to the primary output of the entire chip or block, the functionality and timing budget of the chip or block are not affected.

In order to achieve the boosting sharing, we propose dynamic clock skew shifting such that the timing deficit resulted from a timing error can be transferred among different stages. Consider the example in Figure 11. If a timing error is detected at

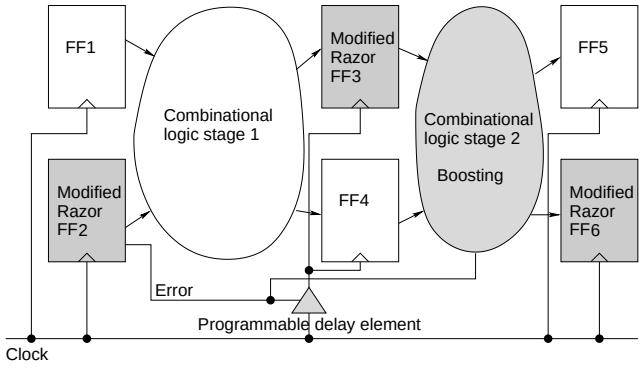


Fig. 11. Elastic timing via simultaneous skew shifting and speed boosting.

flip-flop FF2, we temporarily shift the clock signal arrival time at FF3 and FF4 to a later time. Then, the late launch times at the outputs of flip-flop FF3 and FF4 are compensated by boosting at logic stage 2. The skew shifting can be realized using programmable delay elements [35] where certain capacitive load can be dynamically connected or disconnected from the clock signal paths through pass transistors. The pass transistors are controlled by the error signals so that they can be turned on/off at runtime. Although the proposed boosting sharing can reduce boosting cost, it causes skew shifting cost which includes programmable delay elements [35] and the control interconnect. Therefore, we need to find the best tradeoff between the skew shifting cost and the boosting cost.

## F. Timing Control

In the elastic timing scheme, we need to manage the timing interaction among logic signals, the error signal and boosting switchings. This timing management should not be more complex than managing the interaction between logic signals and the clock signal in conventional designs.

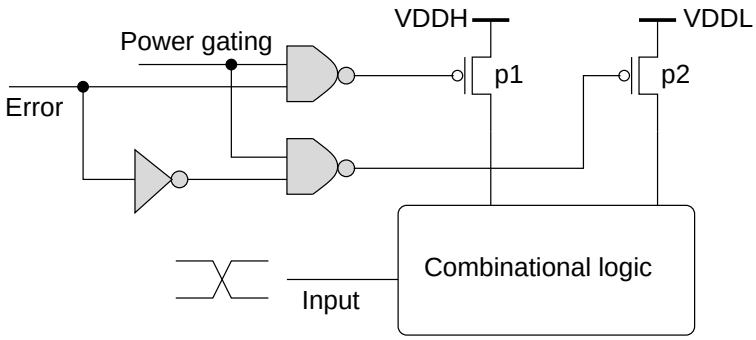


Fig. 12. Dynamic dual-$V_{DD}$ with power gating.

We first discuss the case of dynamic dual-$V_{DD}$ which is implemented by two sleep transistors (p1 and p2 in Figure 12). This involves the switchings of p1, p2 and the logic circuit and depends on the direction of mode change: from normal to boosting mode or from boosting to normal mode. The error signal arrival times at p1 and p2 are denoted as $d_1$ and $d_2$, respectively. The effects of different scenarios are listed in Table XI. Consider the case that the logic circuit is switching from normal mode to boosting mode. If $d_1 < d_2$, i.e., p1 is turned on before p2 is turned off, there is a short moment when both p1 and p2 are on. Then, there is a short circuit between $V_{DD,H}$ and $V_{DD,L}$. If $d_1 > d_2$, there is a moment that both p1 and p2 are off. If the logic circuit does not switch during this moment, the logic signals retain their levels like in a dynamic circuit. Otherwise, the logic switchings are paused and therefore their delays are increased. The cases when the circuit changes from boosting to normal mode are symmetric.

Table XI. Sleep transistor timing in dynamic dual-$V_{DD}$ boosting

| | Logic switches | | No logic switching | |
|---|---|---|---|---|
| Mode change | $d_1 < d_2$ | $d_1 > d_2$ | $d_1 < d_2$ | $d_1 > d_2$ |
| Norm $\rightsquigarrow$ boost | $V_{DD}$ fight | Extra delay | $V_{DD}$ fight | No effect |
| Boost $\rightsquigarrow$ norm | Extra delay | $V_{DD}$ fight | No effect | $V_{DD}$ fight |

In order to minimize the short circuit between the two $V_{DD}$s, the time interval when both p1 and p2 are on should be minimized. This can be achieved by carefully placing p1/p2 and routing the error signal to p1/p2. The two sleep transistors should not be far from each other. The routing of the error signal to them needs to be performed like clock routing such that the skew $|d_1 - d_2|$ is minimized. If $|d_1 - d_2|$ is less than a few picoseconds and $V_{DD,H} - V_{DD,L}$ is not large, the $V_{DD}$ fight becomes indiscernible. If p1 and p2 are not far from each other, it is not difficult to make the skew within a few picoseconds. If the skew $|d_1 - d_2|$ is small, the time interval when both p1 and p2 are off is also small. Therefore, the resulting extra delay on the logic switchings is small and can be neglected. As a result, the logic signal and the error signal can switch at the same time, and the skew $|d_1 - d_2|$ should be minimized.

We tested the cases in Table XI through SPICE simulation on a combinational logic block which includes 10 gates in $90nm$ technology BPTM model [25]. Under $V_{DD,L} = 0.9V$, its path delay is about $175ps$ and its average power dissipation is $0.275mW$. We let this block switch from $V_{DD,L}$ to $V_{DD,H} = 1.1V$ and back. The input switches at the same time as either $d_1$ or $d_2$ and $|d_1 - d_2|$ varies within a range of $10ps$. Since p1 and p2 are not far from each other in placement, a skew constraint of $10ps$ is easy to satisfy in practice. For example, a $200\mu m$ long metal wire in $90nm$ technology BPTM model [25] has delay less than $1.2ps$ according to the Elmore delay, which is an upper bound for delay. Therefore, a $10ps$ skew constraint is easy to satisfy for clock sinks close to each other. The SPICE simulation results show that the path delay at most increases to $177ps$ during the $V_{DD}$ switching. In other words, the delay increase is no greater than 1.1%. The power dissipation can increase to $0.285mW$ which corresponds to 3.6% increase.

Now we discuss the requirement to the error signal timing. Consider the example of Figure 11 whose timing diagram is depicted in Figure 13. The clock arrival time

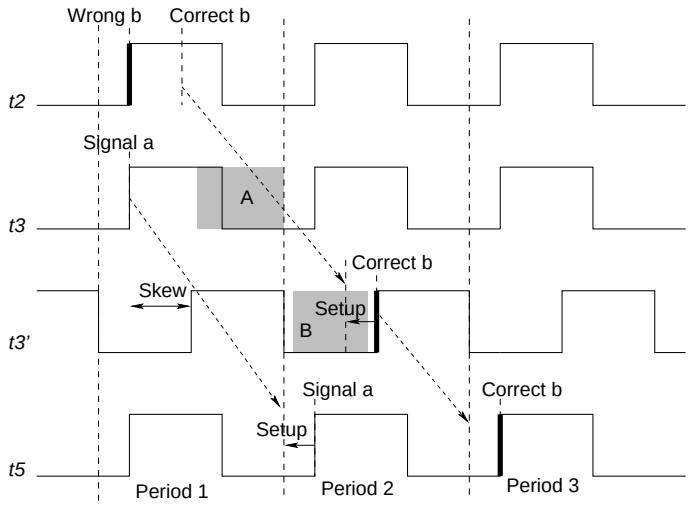


Fig. 13. Timing diagram for the pipeline in Figure 11. $t_3$ is the clock arrival time at FF3 in normal mode. $t_3'$ is the clock arrival time at FF3 after skew shifting. The error signal should arrive FF3 in shaded interval A and arrive logic stage 2 in shaded interval B.

to a flip-flop $i$ in normal mode is denoted as $t_i$. The corresponding time after skew shifting is $t_i'$. Assume a timing error is detected when FF2 is trying to catch logic signal $b$. In other words, a wrong signal for $b$ is caught by FF2 at its active clock edge, which is indicated by the thickened rising edge for $t_2$ in Figure 13. At the same time, logic signal $a$ is correctly captured by FF3. The correct signal $b$ is sent from FF2 at a later time. Then, the error signal should shift the clock signal arrival time $t_3$ to $t_3'$. The error signal can arrive FF3 within a time interval indicated by shaded region A in Figure 13. The error signal also switches logic stage 2 to boosting mode. This should happen after logic signal $a$ is captured by the next stage, for example, FF5 in Figure 11. The switching should be triggered before the corrected signal $b$ arrives logic stage 2. Hence, the allowed time interval for the error signal to arrive stage 2 is the shaded region B in Figure 13. In clock period 2 indicated in Figure 13, the error signal disappears at the output of FF2. As a result, $t_3'$ is switched back

to $t_3$ and logic stage 2 is restored to normal mode. If there is another timing error following signal $b$, the skew shift and the boosting mode are retained. It can be seen that there are certain constraints to the error signal time, but these constraints are not strict in general.

## G. Optimization for Minimizing the Overhead

When designing an elastic timing scheme, we first need to choose where to use modified Razor flip-flops (MRFF). If a combinational path has large timing slack even under low power implementation (low $V_{DD}$, high $V_{th}$ and small gate sizes), then MRFF is not needed there. Therefore, we use MRFF at the destination end of a combinational path only when its timing slack is relatively small. Next, we need to decide which logic stages should be boostable and where to use skew shifting. In this regard, we consider circuit timing constraints in addition to boosting and skew shifting cost. We wish to minimize the boosting and skew shifting cost subject to timing constraints. This problem can be formulated and solved as a mixed integer linear programming (MILP) which will be elaborated as follows.

We define some notations before describing the MILP formulation. The clock signal arrival time to flip-flop $i$ is denoted as $t_i$[1]. Each $t_i$ might be shifted by $x_i$, the value of which is decided by the optimization engine. Due to spatial and fabrication restrictions for programmable delay elements, $x_i$ is bounded in a range $[l_i, u_i]$. According to design rules, $x_i$ can take only discrete values. However, the number of the discrete values is plenty such that we can approximately treat $x_i$ as a continuous variable. We use a linear function $\alpha_i x_i$ to model the cost of skew shifting, although

[1]For a Razor based flip-flop, the clock signal arrival time at its shadow latch may be different from that at the main flip-flop [2]. Here we assume that the two times are equal to each other for the convenience of presentation without loss of generality.

other forms of cost functions can be accommodated in our optimization framework as well. Let $D_{ij,0}$ denote the maximum delay in the logic path between flip-flops $i$ and $j$ in normal mode. Assume there are $b_{ij}$ options for boosting the speed of this path. We use $D_{ij,l}(1 \leq l \leq b_{ij})$ to represent the maximum boosted delay when boosting option $l$ is implemented. The corresponding boosting cost is denoted as $\beta_{ij,l}$. These options are sorted in non-decreasing order of their cost. We also define decision variables $\{y_{ij,0}, y_{ij,1}, ..., y_{ij,b_{ij}}\}$ to choose among the options for boosting. If $y_{ij,0} = 1$, no boosting is implemented. Let $T$ denote the clock period. The delay overhead of error detection at flip-flop $i$ is $\Delta_i$, which includes the lateness of the signal arrival to the flip-flop input and the delay of comparator and multiplexer. The optimization for elastic timing scheme is aimed to minimizing boosting and skew shifting cost subject to timing constraints when timing errors occur. This is formulated as the following MILP problem.

$$\text{Minimize} \quad \sum_i \alpha_i x_i + \sum_{ij} \sum_l \beta_{ij,l} y_{ij,l} \tag{3.1}$$

$$\text{Subject to} \quad l_i \leq x_i \leq u_i, \forall i \tag{3.2}$$

$$t_i + \Delta_i + \sum y_{ij,l} D_{ij,l} \leq t_j + T + x_j, \forall \; path \; i \rightsquigarrow j \tag{3.3}$$

$$t_i + x_i + \sum y_{ij,l} D_{ij,l} \leq t_j + T + x_j, \forall \; path \; i \rightsquigarrow j \tag{3.4}$$

$$\sum y_{ij,l} = 1, \forall \; path \; i \rightsquigarrow j \tag{3.5}$$

$$y_{ij,l} \in \{0, 1\}, \forall \; path \; i \rightsquigarrow j, \forall l \tag{3.6}$$

The objective function (3.1) is to minimize the overall cost from skew shifting and boosting. The constraints of the above formulation is for the scenario where a timing error has occurred and the error correction procedure is triggered. Constraint (3.2) enforces the allowed skew shifting range. Inequality (3.3) is the long path constraint when a timing error occurs at MRFF $i$. Inequality (3.4) is the long path constraint

when clock skew is shifted at flip-flop $i$, which can be either a conventional flip-flop or an MRFF. The short path constraints are omitted for the sake of brevity and can be easily added. Constraint (3.5) ensures that no more than one option of boosting is selected for each logic stage. If power gating is applied with fast lane boosting, the boosting stage cannot be at immediate fanout of an MRFF. This constraint can be realized by prefixing the value of corresponding decision variable $y_{ij,l}$ to be 0. This mixed integer linear programming problem can be solved using existing solvers.

## H. Experimental Results

In the experiments, we compare the proposed elastic timing scheme with conventional safety margin based approach[2]. The comparison is focused on power dissipation while the clock periods in both approaches are chosen to be the same for each circuit. We make sure that all timing errors can be recovered by the elastic timing scheme under these clock periods. In the conventional approach, a relatively high $V_{DD}$ is employed to maintain the timing safety margin. In the elastic timing scheme, a relatively low $V_{DD}$ is used in its normal operations.

The experiments are carried out on ISCAS89 sequential circuits which include circuits with feedback loops. The numbers of logic gates and flip-flops for these circuits are listed in the second and the third column of Table XII. The $V_{DD}$ for the conventional timing scheme is $1.05V$. The power and delay of each gate are treated as random variables following Gaussian distribution. The nominal power

[2]It is difficult to compare our method with Razor for two reasons: (1) our work is targeted to general circuits, especially sequential circuits with loops, where the Razor technology is not directly applicable; (2) our error correction technique is at circuit level and the corresponding benchmarks do not have architectural level infrastructure for pipeline flushing like in Razor. We anticipate that the power savings from our technique is less than that of Razor since Razor is applied together with dynamic voltage scaling. Perhaps this is a price paid for our capability of non-stalling error correction.

and delay of each gate are computed based on $90nm$ technology BPTM model [25] and SPICE characterization. The standard deviation ($\sigma$) of each random variable is set to be 5% of its nominal value. Each gate delay is characterized by SPICE simulation. The longest path delay of each logic stage is obtained by static timing analysis. The clock period $T$ for each circuit is in column 4 of Table XII. It is decided by adding safety margin ($SM$) to the maximum variational path delay. If the longest path delay from flip-flop $i$ to flip-flop $j$ is $D_{ij}$ and its standard deviation is $\sigma_{ij}$, then $T = \max_{\forall \text{paths}}(D_{ij} + 3\sigma_{ij}) + SM$. The fifth column of Table XII shows the minimum timing safety margin for each circuit in term of $T$. The average power dissipation is displayed in the rightmost column.

Table XII. Results from conventional safety margin based timing scheme

| Case | #gates | #FF | $T(ps)$ | $SM$ | Power |
|---|---|---|---|---|---|
| S444 | 126 | 21 | 165 | $0.16T$ | 1.0 |
| S526 | 163 | 21 | 171 | $0.15T$ | 1.5 |
| S526n | 159 | 21 | 190 | $0.15T$ | 1.3 |
| S820 | 272 | 5 | 192 | $0.14T$ | 1.9 |
| S832 | 279 | 5 | 186 | $0.03T$ | 1.9 |
| S1423 | 535 | 74 | 567 | $0.11T$ | 1.3 |
| S9234 | 811 | 135 | 360 | $0.16T$ | 3.4 |
| S13207 | 1793 | 453 | 525 | $0.12T$ | 4.8 |
| S38584 | 8182 | 1285 | 518 | $0.14T$ | 21.7 |

In Table XIII, we report the results of the MILP solution for the elastic timing scheme. The second column lists the number of Modified Razor Flip-flips (MRFF) for each circuit. The mixed integer linear programming (MILP) problem (3.1-3.6) was solved using a public domain solver GLPK (GNU Linear Programming Kit) downloaded from $http://www.gnu.org/software/glpk/glpk.html$. This solver was run on a Linux machine with 2 dual-core Intel Xeon processors of 3.2GHz and 8G memory, and the corresponding CPU runtimes in seconds are reported in the third column of Table XIII. The MILP solution chooses to shift the clock skew for certain

flip-flops depending upon the objective function and constraints in (3.1-3.6). Column 4 and column 5 of Table XIII show the number of flip-flops with skew shift and the maximum amount of shift.

Table XIII. Results of MILP for elastic timing scheme (CPU time in *sec*)

| Case | #MRFF | MILP CPU | Skew shift | | Dual-$V_{DD}$ #gates | Fast lane #gates |
|---|---|---|---|---|---|---|
| | | | #shifts | Max | | |
| S444 | 1 | 0.03 | 3 (2%) | 10$ps$ | 41 | 0 |
| S526 | 6 | 0.01 | 10 (5%) | 14$ps$ | 52 | 0 |
| S526n | 3 | 0.04 | 8 (4%) | 15$ps$ | 91 | 9 (5%) |
| S820 | 5 | 0.01 | 0 (0%) | 0$ps$ | 86 | 0 |
| S832 | 4 | 0.01 | 1 (0%) | 13$ps$ | 121 | 0 |
| S1423 | 9 | 16.00 | 24 (4%) | 42$ps$ | 316 | 21 (3%) |
| S9234 | 12 | 1.00 | 13 (1%) | 25$ps$ | 180 | 0 |
| S13207 | 10 | 2.00 | 8 (0%) | 18$ps$ | 107 | 0 |
| S38584 | 2 | 41.00 | 6 (0%) | 25$ps$ | 296 | 0 |

Table XIV. Results of the elastic timing scheme (power in $mW$)

| Case | #Errors | Err protect margin | | Power | Reduction |
|---|---|---|---|---|---|
| | | Normal | Boosting | | |
| S444 | 25 | 0.12$T$ | 0.12$T$ | 0.8 | 20% |
| S526 | 0 | 0.11$T$ | 0.10$T$ | 1.2 | 24% |
| S526n | 28 | 0.11$T$ | 0.13$T$ | 1.0 | 22% |
| S820 | 9 | 0.48$T$ | 0.62$T$ | 1.4 | 27% |
| S832 | 7 | 0.11$T$ | 0.11$T$ | 1.4 | 25% |
| S1423 | 8 | 0.10$T$ | 0.08$T$ | 1.0 | 24% |
| S9234 | 4 | 0.11$T$ | 0.15$T$ | 2.5 | 25% |
| S13207 | 3 | 0.19$T$ | 0.04$T$ | 3.6 | 26% |
| S38584 | 0 | 0.19$T$ | 0.19$T$ | 15.8 | 27% |

For each logic stage, there are two boosting options: dynamic dual-$V_{DD}$ or dynamic fast lane. In normal mode, the circuits in the elastic timing scheme operate under $V_{DD}$ of $0.9V$. If a logic stage is implemented with dynamic dual-$V_{DD}$, its $V_{DD}$ switches to $1.1V$ in boosting mode. For dynamic fast lane, the fast replica is implemented with $V_{DD} = 1.1V$, low $V_{th}$ and forward body bias. Since no gate resizing is

performed for the fast replica, its area is approximately the same as its original circuit. Based on the results of MILP, each logic stage is implemented either without boosting, with dynamic dual-$V_{DD}$ boosting or dynamic fast lane boosting. In Table XIII, column 6 indicates the number of logic gates with dynamic dual-$V_{DD}$ boosting. Since the size of each logic stage is small, we use two sleep transistors as dual-$V_{DD}$ switches in a logic stage. Column 7 provides the numbers of replicated gates in dynamic fast lane.

The variation tolerance of the elastic timing scheme is tested through Monte Carlo simulation of 1000 times. In Table XIV, column 2 shows the number of timing errors occurred in the Monte Carlo simulations. All these timing errors are successfully corrected in the elastic timing scheme. The clock periods $T$ in the elastic timing scheme are the same as those in Table XII. We define **error protection margin** as the timing safety margin that prevents the error correction mechanism from failures in the worst case variations. This is different from the conventional safety margin which is to avoid any timing errors. However, both of them are utilized to ensure that a circuit can function properly under the worst case variations. The error protection margin of the normal mode is the minimum between two cases: (1) a logic path with Modified Razor Flip-flop (MRFF) at its destination, and (2) a logic path with conventional flip-flop at its destination. In case (1), the error protection margin is obtained by $\min_{\forall \text{paths}}(1.5T - D_{ij} - 3\sigma_{ij})$ where the $0.5T$ is from the shadow latch protection in MRFF. The error protection margin in case (2) is the same as the conventional approaches. The error protection margin of the boosting mode is the minimum difference between the two sides of inequality (3.3) and (3.4). The error protection margins in normal mode and boosting mode are shown in column 3 and column 4, respectively. The rightmost two columns of Table XIV display the average power dissipation, which includes the extra power of speed boosting and skew shift-

ing triggered by timing errors, and the power reduction compared to the conventional safety margin based approach. The following observations could be drawn from the results:

- Our approach has a clear advantage on power dissipation which is $20\% - 27\%$ less than the conventional approach.
- The MILP solver tends to choose more dual-$V_{DD}$ boostings than fast lane boostings. Sine fast lane boosting is more effective but more expensive, it is employed only in cases where dual-$V_{DD}$ boosting is not sufficient to correct timing errors.
- The area increase of our approach is limited. This can be observed from column 4 and 7 of Table XIII. Roughly speaking, a skew shift induces an extra gate of small size. Thus, the area increase compared to the conventional approach is rarely greater than 5%. Of course, we also need to consider the extra cost on sleep transistors, power grid for dual-$V_{DD}$ and forward body bias. However, these cost overhead have been acceptable in many industrial designs with power gating, multi-$V_{DD}$ and adaptive body bias. Our work is focused on the error correction scheme. The overheads on the error detection, such as Razor flip-flops and delay padding for short path constraints, are roughly the same as the original Razor and have been discussed in [2].
- The options of $V_{DD}$ levels in the elastic timing scheme is significantly less than that in Razor, which is applied with dynamic voltage scaling. Consequently, our control to the error rate is not as refined as in Razor. In some circuits, we did not meet any timing errors in the 1000-run of Monte Carlo simulation. However, the error rate showing in column 2 of Table XIV is never greater than 3%. This error rate is sufficiently low such that the extra power dissipation in boosting is not significant.

- The error protection margins in Table XIV are close to the safety margins in Table XII. This implies that the elastic timing scheme has similar variation tolerance as the conventional safety margin based timing scheme.

# CHAPTER IV

# DISCRETE BUFFER AND WIRE SIZING FOR LINK-BASED NON-TREE CLOCK NETWORKS

In this work, we investigate optimizing link-based non-tree clock networks through buffer and wire sizing. A two-stage hybrid optimization approach is proposed. It considers the realistic constraint of discrete buffer/wire sizes and is based on accurate delay models. We use SVM (Support vector machine) based learning for guidance to our optimization. SVM is efficient in estimating the cost of optimization and acts as a surrogate for expensive circuit-level simulation. Experimental results on benchmark circuits show that our sizing method can reduce clock skew by 45% on average with very small increase in power dissipation.

## A. Introduction

VLSI designs in nanometer regime are facing two simultaneous challenges: variability and power. Clock network is a sub-circuit that deeply involves both of the challenges. On one hand, clock skew, which is the difference between clock signal delays, is very sensitive to variations. On the other hand, clock network is a large power consumer due to its large fanout size and high switching frequency. A well-known approach for skew tolerance to variations is clock mesh [36]. However, clock mesh has large wire/power overhead and consequently exacerbates the power crisis. Recently, Rajaram, et al., proposed a new approach [37] that constructs non-tree clock network by inserting cross links in a clock tree. Such network is much more robust than clock trees yet causes much less wire/power overhead compared to clock mesh. Therefore, the link-based non-tree clock network provides an appealing tradeoff between robustness and power overhead.

Later, an improved link insertion algorithm was introduced in [38]. Incremental link insertion techniques were reported in [39, 40]. Link insertion for buffered clock networks were discussed in [21, 41]. Most of these works are focused on only link insertions while there is almost no work on optimizing clock networks with cross links. Is it necessary to further optimize a clock network after link insertion? The answer is "yes", especially for buffered clock networks. According to [37], if a cross link is inserted between two nodes with large skew, it may increase clock skew for some other nodes, i.e., it is better to insert links between nodes with zero skew. However, obtaining the zero skew in buffered clock trees is a very difficult problem by itself. The work of [21] suggests to minimize the skew by using special buffers that have additional capacitors inside. Since the capacitors are within cascaded buffers, changing their sizes can tune the buffer delay with very little influence to slew rate and capacitive load of other parts of the clock tree. As a result, skew minimization becomes a local tuning problem which is much easier to solve. The main drawback of this technique is that such special buffers are not common in practice and may require significant changes to design methodologies. Without using such buffers, link insertion in buffered clock trees often increases nominal clock skew as shown in the experimental results of [41]. Although skew variability becomes increasingly prominent, nominal skew is still essentially important to circuit design. According to [37], decreasing nominal skew may also enhance the effect of link insertion on skew variability reduction. Therefore, merely inserting links in buffered clock trees is not adequate and further skew optimization is still necessary.

This work investigates optimizing link-based non-tree clock networks via buffer and wire sizing, which is a common and effective technique for skew minimization, For clock trees, buffer sizing techniques were reported in [42, 43] and wire sizing methods were introduced in [19, 44, 45, 46]. Wire sizing algorithms were also developed for

clock mesh in [47, 48]. In [49, 15, 50], approaches for simultaneous clock tree buffer and wire sizing were proposed. These previous works are either based on the Elmore delay model [42, 46, 49], which is inaccurate for evaluating skew in modern technology, or handle only continuous sizings [45, 48, 15, 50]. The work of [44] does discrete sizing with accurate models, but it is limited to wire sizing only. In realistic designs, the requirement to clock skew accuracy is very rigorous. Clock network design is often closed by circuit simulations of SPICE level accuracy as opposed to static timer level accuracy for logic timing closure. On the other hand, the number of clock nets on a chip is far less than that of signal nets. Thus, the relatively long runtime caused by using accurate model is well justified for clock network design. Since link-based non-tree is targeted to ASIC designs, the sizes of buffers and wires are discrete. Moreover, the number of buffer and wire size options for clock network is often small. In other words, the buffer and wire sizing problem for ASIC clock networks is highly discrete. For highly discrete problems, rounding continuous solutions may result in significant errors [18, 51] and therefore direct combinatorial techniques are needed.

We will focus on discrete buffer and wire sizing with accurate delay models. This is a very difficult problem which has not been well studied. The difficulty is due to both the discreteness and model complexity. Unlike continuous sizing, which is usually guided by sensitivity information [45, 48, 15, 50], discrete sizing can rarely rely on sensitivity, which is valid mostly for very small changes. Moreover, it is very difficult to have fine-grained control to highly discrete problems. For example, changing buffer/wire size by one discrete step may result in a large change on skew and consequently a discrete optimization tends to have slow convergence. Unlike the cases using simple models, where many candidate solutions can be quickly evaluated and explored [49, 46], using accurate model is too expensive for systematic solution search. Even worse, non-tree networks are more complicated to analyze than clock trees.

We solve this difficult yet important problem using a two-stage hybrid optimization approach, which can significantly reduce the clock skew. Support vector machine (SVM) is integrated into the optimization scheme to handle the complex delay model issue and provide guidance for discrete optimization in large design space. To the best of our knowledge, this is the first work on discrete clock network sizing using accurate delay model. Experimental results on benchmark circuits show that our sizing method can reduce clock skew by 45% on average with very small increase on power dissipation. Considering that the granularity of buffer/wire sizes is coarse, 45% skew reduction is very significant.

## B. Skew Modeling via Statistical Learning

One of the significant hurdles in the optimization of large non-tree clock networks is the cost of circuit-level non-tree analysis. In order to assess the impact of design change on the network performance (e.g. the maximum clock skew) during optimization, a large non-tree clock network must be analyzed many times. In discrete sizing based optimization, the cost of clock network analysis is further exacerbated since relatively efficient incremental sensitivity analysis is unable to provide accurate prediction of clock skews over wide discrete gate and wire size ranges. Hence, an accurate yet efficient surrogate for costly circuit-level simulation is strongly desirable to facilitate efficient optimization.

To this end, we adopt powerful statistical learning techniques to empirically model the performance of a clock network (e.g. maximum clock skew) as a function of multiple gate and wire sizes. The type of learning algorithms that has a particular appeal for our problem is *Support Vector Machine*, or SVM [52, 53]. SVM is an appropriate choice for our clock skew empirical modeling as it is well suited for fitting

highly nonlinear and high-dimensional data. In contrast, widely used polynomial based fitting is only applicable to smooth local data.

For a set of $M$ training data set $(x_1, y_1), \cdots, (x_M, y_M)$, where, $x_i$ is an input pattern and $y_i$ is the output, the so-called $\varepsilon$-SVM regression attempts to construct a regression model $y = f(x)$ that has at most $\varepsilon$ deviation from the actual $y$ value for all the training data. The SVM regression model is in the form:

$$f(x) = \sum_{i=1}^{K} \alpha_i K(s_i, x), \tag{4.1}$$

where $s_i's$ are a set of $K$ (small) number of support vectors in the input space, $\alpha_i$ is a non zero co-efficient and $K(\cdot, \cdot)$ is a kernel function. The admissible kernel functions correspond to a dot product in certain input feature space. One particularly effective Kernel function is the radial-basis kernel function

$$K(a, b) = e^{\frac{-||a-b||^2}{2\sigma^2}}. \tag{4.2}$$

SVM machinery attempts to produce a *flattest* regression model according to the introduced margin $\varepsilon$ and thereby avoids over-fitting. The use of a small set of support vectors and suitable Kernel functions leads to the sparsity of the model and also makes it suitable for fitting highly nonlinear data in a high-dimensional input space, as required in our clock skew modeling task.

SVM model is integrated into our sizing algorithm to provide guidance for the optimization. Thus, we build an SVM model before performing the optimization for buffer and wire sizing. The SVM model is built by performing $n$ number of SPICE simulations. For each simulation, we determine the skew cost $Q = \sum q_{ij}^2$, where $q_{ij}$ is the clock skew between node $i$ and node $j$. The results of the $n$ simulations are used to build the SVM model by statistical learning. The sizes of buffers and wires for each SPICE simulation are determined randomly. Once the model is built, it can

be integrated into our optimization scheme and is efficient in estimating the skew cost during optimization. Building an SVM model can take significant amount of time. Thus, we consider running 120 SPICE simulations for statistical learning of SVM model to minimize the learning time of SVM. The accuracy of the SVM model compared to an accurate delay model like SPICE and the CPU time for building SVM model is discussed in experimental section.

## C. Sizing Algorithms

### 1. Overview

The goal of buffer and wire sizing is to minimize the global skew, which is the maximum clock delay difference among all pairs of clock sinks. In the big picture, our approach is to iteratively optimize a portion of the given clock network. Compared to simultaneously optimizing the entire network, the computation cost of our approach is more practical. Compared to iteratively optimizing a single element, such as a single buffer size, our approach is more efficient on finding good solutions from a global perspective.

The overall flow of our sizing method is shown in Figure 14. In this flow, an *OptimizationCore* is applied repeatedly. The procedure of *OptimizationCore* tells how to extract a portion from the clock network and how to optimize this portion. The *OptimizationCore* may be instantiated differently at different stages/steps of the flow. The pseudo code of generic *OptimizationCore* is given in Figure 15. The optimization in our algorithm is carried out from the clock source to the sinks in a level-by-level manner. Figure 16 illustrates the levels in a part of a clock network. In each iteration, only one level is optimized such that the skew among leaf nodes of that level is minimized. This strategy is in the same spirit of delay balancing [19], which can assist the overall skew minimization and avoid the risk of short circuit between dif-

**Procedure:** $Sizing(\mathcal{G})$

**Input:** clock network $\mathcal{G}$

**Stage 1:**

Remove link resistors

For each level

Step 1: $OptimizationCore(\mathcal{P}_1, e_1, k_1, \mathcal{E}_1)$

Step 2: $OptimizationCore(\mathcal{P}_1, e_{1,2}, k_2, \mathcal{E}_1)$

**Stage 2:**

Add link resistors back

For each level

Step 1: $OptimizationCore(\mathcal{P}_2, e_1, k_1, \mathcal{E}_2)$

Step 2: $OptimizationCore(\mathcal{P}_2, e_{2,2}, k_2, \mathcal{E}_2)$

Fig. 14. Overall optimization flow.

ferent buffers [21]. The optimization at each level is started with a SPICE simulation. In order to improve the computation efficiency, the optimization is performed on a set of components instead of the entire level. Component $\mathcal{P}$ in the *OptimizationCore* is defined as subtree or subtree with links, which is a buffered subtree with the buffer at root of the subtree. For instance, there are four subtrees at level $i+1$ in Figure 16. The parameter $e$ indicates what elements in the selected components should be sized: buffer, wire and/or link. We select $k$ subtrees associated with the maximum delay and another $k$ subtrees associated with the minimum delay among the leaf nodes of this level. For the example in Figure 16, if $k = 1$, we choose the left-most subtree and the right-most subtree for optimization. Since the main goal of clock network sizing is to minimize clock skew, we extract $k$ components associated with the maximum (minimum) delay and try to reduce (increase) their delay.

In our optimization, the objective is formulated as minimizing a skew cost function $Q = \sum q_{ij}^2$, where $q_{ij}$ is the clock skew between node $i$ and node $j$. It is shown in [50] that the quadratic objective function not only addresses the overall skew but also leads to solutions with improved robustness. In order to quickly evaluate the impact on the objective function $Q$ from sizing, an SVM model is built to model $Q$ for the extracted components. Although it takes significant time to build the SVM model, this model is very quick on estimating $Q$ and is called frequently in the optimization. Therefore, using SVM can help to reduce the overall runtime. We restrain the size of the extracted portion so that the quality of the SVM model is justified. After the SVM model is obtained, optimization $\mathcal{E}$ is performed on the extracted portion.

| **Procedure:** $OptimizationCore(\mathcal{P}, e, k, \mathcal{E})$ |
|---|
| **Input:**<br>$\mathcal{P}$: definition of component<br>$e$: types of elements to be sized<br>$k$: the number of components for optimization<br>$\mathcal{E}$: optimization engine |
| 1. Run SPICE simulation<br>2. $S \leftarrow k$ components $\mathcal{P}$ associated with max delay<br>$S \leftarrow S \cup k$ components $\mathcal{P}$ associated with min delay<br>3. Build SVM model over $S$<br>4. Size element types $e$ in $S$ using $\mathcal{E}$ |

Fig. 15. Optimization core.

The overall flow (Figure 14) includes two stages. The main difference between them is that the network is relaxed to a tree for the optimization in stage 1 while

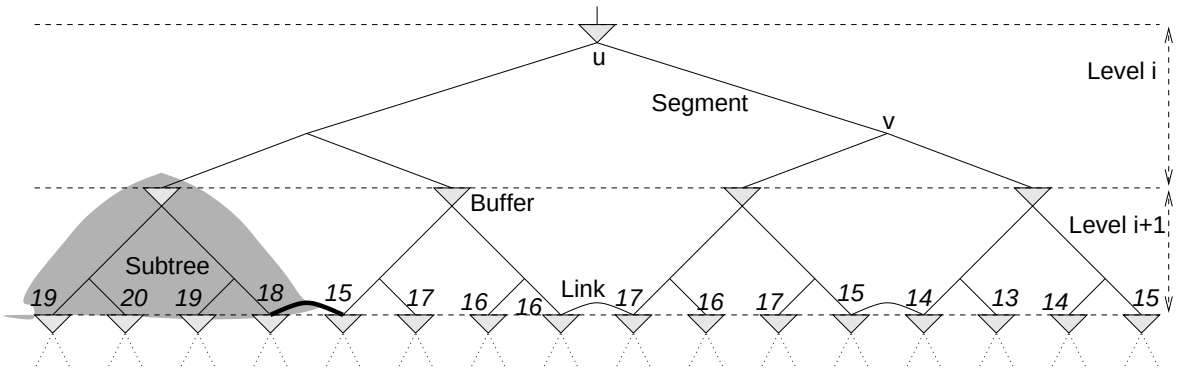


Fig. 16. Illustration of definitions. The number besides each buffer is the clock signal arrival time to the buffer input.

stage 2 optimizes the complete network. The motivation for the relaxation in stage 1 is for the convenience of divide-and-conquer. The reason that we are able to do the relaxation by simply removing the link resistors is based on an important conclusion in [37]. According to [37], if a link resistor is inserted between two nodes with zero skew, then this resistor does not affect skew between any two nodes in the network. Since the links are usually inserted between leaf nodes [21, 41], after skew minimization in stage 1, the effect of adding link resistors back is not large. In other words, the improvement obtained in stage 1 can be mostly retained even after adding link resistors back. The details of stage 1 and stage 2 will be elaborated in subsequent sections.

## 2. Optimization Stage 1

### a. Overall Optimization Flow

At the beginning of stage 1, the link resistors are removed from the clock network while the link capacitances are retained. After the removal, the topology of the clock network becomes tree, which is usually a balanced tree due to the requirement from

link insertion [21, 41]. There are two steps in the optimization of stage 1: step 1 is more on a coarse and global level while step 2 is more on a fine-grained and local level. The component $\mathcal{P}_1$ for stage 1 is defined as a subtree, which is a buffered subtree with the buffer at the root of the subtree. In step 1, the elements $e_1$ to be sized include the buffers and the wires of the selected subtrees. Since step 1 is on a relatively coarse level, the wires of each subtree is sized uniformly. In other words, all wire segments of a subtree are sized with the same width. In Figure 16, a wire segment is illustrated between node $u$ and $v$ and the shaded subtree contains 6 wire segments. In the wire sizing of step 1, only one variable is needed for each subtree. Since there are $2k_1$ subtrees selected for step 1, there are $2k_1|B| + 2k_1|W|$ integer variables: where $|B|$ and $|W|$ are the number of buffer and wire width choices available. The main reason for the uniform wire sizing is to reduce the number of variables of each subtree so that more subtrees can be selected for the optimization. In step 2, $2k_2$ subtrees are selected and the sizing is performed at a relatively fine granularity level. The elements $e_{1,2}$ to be sized in step 2 are the individual wire segments of the selected subtrees. For each subtree, the number of elements (wire segments) in step 2 is significantly larger than that of step 1. Therefore, the value of $k_2$ is usually smaller than $k_1$ so that the total number of elements (variables) in step 2 is limited. One can see that the focuses of step 1 and step 2 are complement with each other. Step 1 is mostly for a coarse level sizing while step 2 is more for fine tuning.

The optimization engine $\mathcal{E}_1$ of entire stage 1 is an ILP (Integer Linear Programming) based local search. The objective is to minimize the overall skew cost $Q = \sum q_{ij}^2$, where $q_{ij}$ is the clock skew between leaf node $v_i$ and $v_j$. In the optimization of $\mathcal{E}_1$, the value of $Q$ is estimated according to the SVM model, which is built before $\mathcal{E}_1$ is called.

The pseudo code for the optimization engine $\mathcal{E}_1$ of stage 1 is outlined in Figure 17.

**Procedure: Optimization engine** $\mathcal{E}_1(S)$

**Input:** A set $S$ of components to be sized

```
1. While ( improve ) {
2.     Partition S into a set G of m groups
3.     Obtain average leaf node delay t_ave of each group
4.     Sort groups in G in non-decreasing order of t_ave
5.     For i = 1 to m/2 {
6.         While ( improve )
7.             Increase t_ave of g_i ∈ G by δ
8.         While ( improve )
9.             Decrease t_ave of g_{m-i+1} ∈ G by δ
10.    }
11.}
```

Fig. 17. Optimization engine $\mathcal{E}_1$ for stage 1.

The *improve* in line 1, 6 and 8 is true when the value of the objective function $Q$ is reduced according to the SVM model. Each group resulted from the partition of line 2 consists of one or several subtrees. Usually, we group those subtrees close to each other and with similar leaf delays together. The loop from line 5 to line 10 is to minimize the inter-group skew, which is the difference of the average leaf node delay between two groups. The basic idea is to gradually increase the average delay if a group originally has small average delay (line 7). By the same token, if a group originally has large average delay, the average delay is gradually decreased (line 9). We focus on the inter-group skew because we observed that intra-group skew is usually very small as long as the initial clock tree is not poorly designed.

b. ILP Formulation

The increase or decrease of the average delay (line 7 and line 9 of Figure 17) is achieved by sizing the elements of the group through Integer Programming. We describe the Integer Programming formulation for the case of increasing the average delay. The method for decreasing the average delay is the same. The main constraint is to make the average delay increase by $\delta$ with a maximum error of $\pm\tau$. The Elmore delay model is employed here since it has analytical expressions which can be easily incorporated in the formulation. Although the Elmore delay is sometimes inaccurate, the corresponding solution is accepted only if it indeed reduces the overall skew function $Q$ according to the SVM model. Harnessed by SVM validation, the Elmore based solution should generally move in a reliable direction.

A straightforward formulation leads to an Integer Nonlinear Program problem, which is very difficult to solve. We propose a technique to convert this nonlinear problem into an ILP (Integer Linear Programming) formulation, which is significantly easier than the nonlinear version. Let $b$ and $w$ denote buffer and wire size, respectively. There are two kinds of decision variables. Variable $x_{i,b} = 1$ if size $b$ is assigned to buffer $i$. Similarly, variable $y_{j,w} = 1$ indicates that size $w$ is assigned to wire $j$. Please note that $j$ is a subtree in step 1 but is a segment in step 2. We attempt to increase the delay of each leaf node $l$ by $\delta$ with at most $\pm\tau$ error. The straightforward integer programming formulation is given as follows.

$$\text{Minimize} \quad \tau \tag{4.3}$$

*Subject to* :

$$\delta - \tau \leq \ \Delta t_l \ \leq \delta + \tau, \quad \forall \text{ leaf node } l \tag{4.4}$$

$$\sum_{\forall b} x_{i,b} = 1, \quad \forall \text{ buffer } i \tag{4.5}$$

$$\sum_{\forall w} y_{j,w} = 1, \quad \forall \text{ wire } j \tag{4.6}$$

$$x_{i,b} \in \{0,1\} \ \ \forall\, i,b, \qquad y_{j,w} \in \{0,1\} \ \ \forall\, j,w \tag{4.7}$$

where $\Delta t_l$ is the delay increase of the leaf node $l$ depending on the sizing and $\tau$ is the error in the delay increase. Constraint (4.4) ensures that the delay of each leaf node is increased by $\delta$ with at most $\pm\tau$ error. Constraint (4.5) and (4.6) make sure that only one size is assigned to each buffer and wire. Please note the objective function in this formulation is to minimize the error in the delay increase i.e., minimize $\tau$.

If we express the delay increase in constraint (4.4) in term of the Elmore delay, it is not difficult to find that it is a nonlinear function with respect to the decision variables. Therefore, this formulation results in an Integer Nonlinear Programming problem, which is very difficult to solve directly. Next, we will introduce a technique to convert this nonlinear problem into an ILP formulation, which is significantly easier to solve.

We use a very simple example to illustrate how to convert the nonlinear delay expression into linear function of newly introduced decision variables. In this example, there is one buffer driving one wire segment. There are two options $b_1$ and $b_2$ for the buffer size and two options $w_1$ and $w_2$ for the width of this wire segment. The Elmore delay of the buffer can be expressed as:

$$\begin{aligned} t_B = {} & x_1 R_{b,1} y_1 C_{w,1} + x_1 R_{b,1} y_2 C_{w,2} \\ & + x_2 R_{b,2} y_1 C_{w,1} + x_2 R_{b,2} y_2 C_{w,2} \end{aligned} \tag{4.8}$$

$$x_1 + x_2 = 1 \tag{4.9}$$

$$y_1 + y_2 = 1 \tag{4.10}$$

where $x$ variables are for selecting buffer sizes and $y$ variables are for selecting wire widths. $R_{b,1}$ and $R_{b,2}$ are the buffer output resistances for the sizes $b_1$, $b_2$. $C_{w,1}$ and

$C_{w,2}$ are the wire capacitances for widths $w_1$, $w_2$. Since each of the terms in the delay expression $t_B$ is product of $x$ and $y$ variables, the delay is a non-linear function. We introduce a new set of decision variables to replace $x$ and $y$.

$$\begin{aligned} z_{b1,w1} &= x_{b,1} y_{w,1} \\ z_{b1,w2} &= x_{b,1} y_{w,2} \\ z_{b2,w1} &= x_{b,2} y_{w,1} \\ z_{b2,w2} &= x_{b,2} y_{w,2} \end{aligned} \tag{4.11}$$

Then, the delay expression becomes

$$\begin{aligned} t_B = z_{b1,w1} R_{b,1} C_{w,1} + z_{b1,w2} R_{b,1} C_{w,2} \\ + z_{b2,w1} R_{b,2} C_{w,1} + z_{b2,w2} R_{b,2} C_{w,2} \end{aligned} \tag{4.12}$$

$$z_{b1,w1} + z_{b1,w2} + z_{b2,w1} + z_{b2,w2} = 1 \tag{4.13}$$

$$z_{bi,wj} \in \{0,1\} \quad i = 1,2 \quad j = 1,2 \tag{4.14}$$

Now the delay is a linear function with respect to the new variables $z$. After performing such conversion to (4.4)-(4.7), the original Integer Nonlinear Programming problem becomes ILP formulation and can be solved by existing ILP solver.

Since the solution of ILP solver is in term of $z$, we need to do a post processing to find buffer size and wire width from the values of $z$. For example, if the ILP solution is $z_{b1,w1}$, and the other variables being zero, then the buffer size is selected to be $b_1$ and the wire width is $w_1$. There is a price that we have to pay for the nonlinear-linear conversion. That is the increase in number of decision variables. On the first order, the increase is about $(|B| + |W|) \cdot |W|$ per constraint in (4.4) where $|B|$ is the number of buffer sizes and $|W|$ is the number of wire width options. Since our work is targeted to realistic cases that have very few buffer and wire options, the variable

increase caused by the nonlinear-linear conversion is limited. Moreover, we carefully restrict the size of each group being optimized. Therefore, the runtime of solving the ILP is still reasonable.

### 3. Optimization Stage 2

At the beginning of stage 2, the link resistors are added back and therefore the clock network topology becomes a non-tree. The definitions of component and element in stage 2 are similar as those in stage 1 except that the cross links are included. The component $\mathcal{P}_2$ for stage 2 is defined as a subtree and associated links. In Figure 16, one example of $\mathcal{P}_2$ is the shaded subtree plus the thickened link. The optimization of stage 2 also consists of two steps, which are similar to the two steps in stage 1. Same as in stage 1, the elements $e_1$ to be sized in step 1 of stage 2 are buffers and wires of the selected subtrees. Again, all wire segments in the same subtree are sized uniformly. Please note that links are not included in the elements of step 1. The elements $e_{2,2}$ to be sized in step 2 are the individual wire segments and the cross links incident to the selected subtrees.

Since the network topology in stage 2 is non-tree, which is not friendly to ILP formulation, we design the optimization engine $\mathcal{E}_2$ as a group migration based heuristic like Kernighan-Lin partitioning algorithm [54]. Same as in stage 1, the objective of $\mathcal{E}_2$ is to minimize the overall skew cost $Q = \sum q_{ij}^2$, where $q_{ij}$ is the clock skew between leaf node $v_i$ and $v_j$. We define a **move** as either sizing-up or sizing-down an element, which can be either a buffer, a wire segment or a link. The size change in a move is restricted to be a single discrete step. For instance, for a wire with initial width of $2\times$, a move can be a change to $1\times$ or $3\times$, but not $4\times$. If the objective function equals $Q_{i-1}$ before move $i$ and becomes $Q_i$ after move $i$, then the **gain** of this move is defined as $g_i = Q_{i-1} - Q_i$. For moves from 1 to $l$, the **cumulated gain** is $G_l = \sum_{i=1}^{l} g_i$.

| **Procedure: Optimization engine** $\mathcal{E}_2(S)$ |
|---|
| **Input:** A set $S$ of components to be sized |
| 1. While ( true ) { |
| 2. $S' \leftarrow S$ |
| 3. While $S' \neq \emptyset$ { |
| 4. Find move $i$ with max gain $g_i$ |
| 5. $e_i$ = the element sized in move $i$ |
| 6. $S' \leftarrow S' - \{e_i\}$ } |
| 7. Find $l$ such that cumulated gain $G_l$ is maximized |
| 8. If $G_l > 0$, make the $l$ moves on $S$ |
| 9. Else break } |

Fig. 18. Optimization engine $\mathcal{E}_2$ for stage 2.

The pseudo code of the optimization engine $\mathcal{E}_2$ is given in Figure 18. It has two nested loops: inner loop line 3-6 and outer loop line 1-9. In the inner loop, the move that maximizes the gain is found in each iteration. Although these moves are greedy, they are only temporary moves. In the outer loop, the cumulated effect of multiple moves are checked. A group of temporary moves are indeed implemented only if they can maximize the cumulated gain and the cumulated gain is positive. In this procedure, almost all of the computations on gains are based on SVM model except step 8, which is based on SPICE. Since step 8 is the last step to decide whether or not to take the $l$ moves, this decision must be validated by the most reliable model. Step 8 is outside the inner loop, therefore, the number of calls to SPICE is limited.

## D. Experimental Results

We performed experiments on two sets of benchmark circuits (a) ISCAS89 sequential circuits and (b) r1-r5 downloaded from GSRC Bookshelf [22]. These circuits are first synthesized using SIS [55] and then placed by an academic placer mPL [56]. After placement, the locations of the clock sinks are available. The clock tree construction and link insertion are performed according to [21]. The numbers of sinks, buffers and links in the clock networks are listed in Table XV.

Table XV. Testcases

| Case | # Sinks | # Buffers | # Links |
|---|---|---|---|
| S9234 | 135 | 20 | 21 |
| S5378 | 164 | 25 | 30 |
| S13207 | 503 | 77 | 69 |
| S15850 | 566 | 81 | 86 |
| S38584 | 1426 | 235 | 50 |
| S35932 | 1728 | 286 | 143 |
| r1 | 267 | 37 | 44 |
| r2 | 598 | 171 | 31 |
| r3 | 861 | 59 | 45 |
| r4 | 1903 | 303 | 156 |
| r5 | 3101 | 441 | 114 |

The device model and wire parasitics are obtained from $90nm$ BPTM model [57]. The $V_{DD}$ voltage level is 1.0 volt. In our library, we have four buffer types with 16X, 24X, 32X and 48X of the minimum buffer size, respectively. In realistic clock network synthesis, it is very common that people use only a few options of very large buffers. The wire size options include 1X, 2X and 3X of the minimum wire width. In the initial network, all buffers take the size of 24X and all wires have 2X width. The values of $k_1$ and $k_2$ (see Figure 14) are 10-15 and 4-5, respectively.

Our algorithm of buffer and wire sizing is implemented in C language. The integer linear program (ILP) is solved using a public domain solver GLPK (GNU Lin-

ear Programming Kit) downloaded from *http://www.gnu.org/software/glpk/glpk.html*. The binaries for the Support Vector Machine (SVM) is obtained from *http://svmlight.joachims.org*. All of the experiments are carried out on a Linux machine with 2 dual-core Intel Xeon processors of 3.2GHz and 8G of memory.

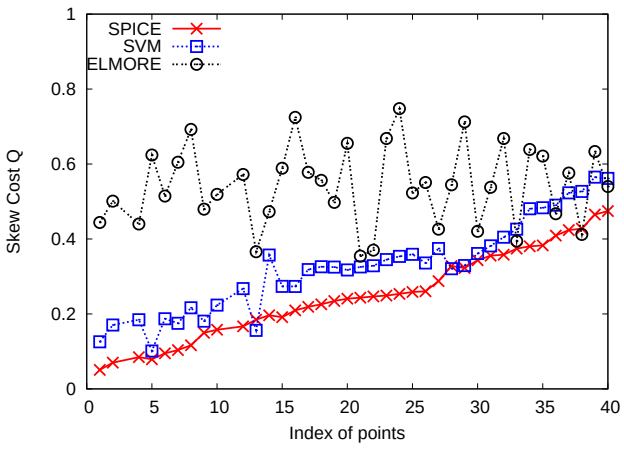


Fig. 19. Comparison of SPICE, SVM model and Elmore delay.

We first run experiment to validate the SVM model. The values of objective function $Q = \sum q_{ij}^2$ from the Elmore delay, SVM model and SPICE simulations are compared in Figure 19. In the Elmore delay computation, each buffer is modeled by input capacitance, output resistance and intrinsic delay, i.e., RC model. These results are from 40 data points (sizing solutions) on a portion of circuit s13207. These data are sorted according the $Q$ value from SPICE. It can be seen that the $Q$ value from SVM approximately follows the trend of SPICE results, although it gives some small errors occasionally. In contrast, the Q values estimated from the Elmore delay have much greater deviations from the SPICE result. Moreover, these deviations are not monotone so that the Elmore delay can hardly capture the general trend of the SPICE delay. The main reason is that the simple RC buffer model in the Elmore delay computation cannot reflect the significant nonlinear behaviors of the devices.

According to this comparison, it is evident that the SVM model is superior to the Elmore delay when applied to guide the skew optimization.

Since, there is no previous work on simultaneous buffer and wire sizing for non-tree clock networks, we compare the following approaches:

- Tree+Link: The initial link based non-tree obtained according to [21].
- Tree+Link+Elmore sizing : This result is obtained from sizing the initial non-tree using our proposed algorithm except that the sizing is guided by the Elmore delay. We show this result to see the effect of using SVM compared to the Elmore delay.
- Tree+Link+SPICE sizing: The result is obtained by using the proposed algorithm and a SPICE simulation based guidance for sizing the initial link-based non-tree clock network. We show the result to see how good is the SVM based sizing compared to an accurate delay model like SPICE.
- Tree+Link+SVM sizing: This is the result from our proposed algorithm.

Table XVI. Results of global skew for non-tree clock network (global skew in ps)

| | Tree+Link | Tree+Link+Elmore | | Tree+Link+SPICE | | Tree+Link+SVM | |
|---|---|---|---|---|---|---|---|
| Case | Skew | Skew | Skew Ratio | Skew | Skew Ratio | Skew | Skew Ratio |
| s9234 | 33.9 | 28.2 | 0.83 | 22.1 | 0.65 | 20.5 | 0.60 |
| s5378 | 23.0 | 24.3 | 1.06 | 14.7 | 0.64 | 3.0 | 0.13 |
| s13207 | 162.0 | 166.7 | 1.03 | 110.7 | 0.68 | 111.0 | 0.68 |
| s15850 | 103.5 | 103.2 | 0.99 | 63.4 | 0.61 | 63.2 | 0.61 |
| s38584 | 212.0 | 211.0 | 0.99 | 116.9 | 0.55 | 115.1 | 0.54 |
| s35932 | 172.0 | 169.0 | 0.98 | N/A | N/A | 129.0 | 0.75 |
| Avg | 117.7 | 117.1 | 0.98 | N/A | N/A | 73.6 | 0.55 |
| r1 | 16.1 | 16.2 | 1.00 | 10.1 | 0.62 | 8.2 | 0.50 |
| r2 | 143.3 | 145.0 | 1.01 | 89.8 | 0.62 | 96.0 | 0.66 |
| r3 | 164.8 | 164.4 | 0.99 | 118.5 | 0.72 | 110.0 | 0.66 |
| r4 | 334.3 | 331.2 | 0.99 | 172.7 | 0.52 | 198.0 | 0.59 |
| r5 | 256.0 | 253.2 | 0.98 | N/A | N/A | 202.0 | 0.79 |
| Avg | 182.9 | 182.0 | 0.99 | N/A | N/A | 122.8 | 0.64 |

Table XVII. Results of resource consumption for non-tree clock network (power in mW and total cap in pF)

| | Tree+Link | | Tree+Link+Elmore | | Tree+Link+SPICE | | Tree+Link+SVM | |
|---|---|---|---|---|---|---|---|---|
| Case | Pow | Cap | Pow | Cap | Pow | Cap | Pow | Cap |
| s9234 | 1.83 | 1.46 | 1.71 | 1.54 | 1.84 | 1.56 | 1.83 | 1.47 |
| s5378 | 2.00 | 1.54 | 1.96 | 1.67 | 2.15 | 1.64 | 2.30 | 1.76 |
| s13207 | 12.50 | 7.80 | 12.20 | 7.91 | 12.80 | 7.83 | 13.00 | 8.08 |
| s15850 | 10.80 | 10.26 | 10.20 | 10.43 | 11.80 | 10.85 | 11.30 | 8.63 |
| s38584 | 37.40 | 34.20 | 34.40 | 36.84 | 37.10 | 35.35 | 38.50 | 34.55 |
| s35932 | 76.70 | 54.55 | 75.80 | 53.29 | N/A | N/A | 84.00 | 55.01 |
| Nor Avg | 1.0 | 1.0 | 0.98 | 1.01 | 1.04 | 1.02 | 1.04 | 1.02 |
| r1 | 2.70 | 1.73 | 2.60 | 1.91 | 2.73 | 1.88 | 2.80 | 17.13 |
| r2 | 8.30 | 5.19 | 8.70 | 5.22 | 8.45 | 5.31 | 8.50 | 5.69 |
| r3 | 14.30 | 10.72 | 13.70 | 10.78 | 15.10 | 10.56 | 15.50 | 11.25 |
| r4 | 31.80 | 27.11 | 30.12 | 27.72 | 33.30 | 26.98 | 32.60 | 27.83 |
| r5 | 73.30 | 64.43 | 72.90 | 60.24 | N/A | N/A | 75.30 | 60.611 |
| Nor Avg | 1.0 | 1.0 | 0.97 | 1.04 | 1.03 | 1.03 | 1.05 | 1.01 |

Table XVIII. Results of CPU time for non-tree clock network (min)

| Case | Tree+Link+Elmore | Tree+Link+SPICE | Tree+Link+SVM |
|---|---|---|---|
| s9234 | 4.2 | 18.2 | 9.8 |
| s5378 | 3.1 | 13.3 | 10.9 |
| s13207 | 10.8 | 262.0 | 41.0 |
| s15850 | 11.2 | 923.1 | 39.7 |
| s38584 | 35.2 | 1986.4 | 193.6 |
| s35932 | 33.6 | N/A | 227.7 |
| Nor Ave. | 0.26 | 8.6 | 1.0 |
| r1 | 4.1 | 59.2 | 21.2 |
| r2 | 5.8 | 100.5 | 44.5 |
| r3 | 6.2 | 413.5 | 79.8 |
| r4 | 20.5 | 2190.0 | 267.2 |
| r5 | 36.7 | N/A | 280.5 |
| Nor Ave. | 0.13 | 4.6 | 1.0 |

To measure effectiveness of our approach, we perform simulations to determine global skew, skew due to variation and resource consumption i.e., power consumption and total capacitance. All of the results on skew and power are obtained using SPICE simulation. The results for ISCAS89 and r1-r5 benchmark circuits are summarized in

Table XIX. Results of skew due to variation for non-tree clock network (average and maximum skew in ps)

| | Tree+Link | | | Tree+Link+Elmore | | | Tree+Link+SVM | | |
|---|---|---|---|---|---|---|---|---|---|
| Case | Max | Avg | SD | Max | Avg | SD | Max | Avg | SD |
| s9234 | 84.0 | 40.9 | 13.3 | 80.9 | 35.8 | 13.4 | 58.6 | 25.4 | 11.3 |
| s5378 | 78.2 | 29.1 | 14.5 | 77.4 | 27.8 | 14.2 | 56.8 | 15.1 | 9.6 |
| s13207 | 293.6 | 191.8 | 30.7 | 80.9 | 35.8 | 30.9 | 237.6 | 142.1 | 25.5 |
| s15850 | 217.9 | 142.1 | 24.22 | 292.1 | 192.1 | 25.1 | 178.1 | 111.2 | 19.2 |
| s38584 | 393.3 | 251.0 | 36.5 | 220.9 | 141.2 | 33.4 | 237.9 | 142.1 | 28.5 |
| s35932 | 316.0 | 241.6 | 29.1 | 363.0 | 234.9 | 27.2 | 249.8 | 188.6 | 21.9 |
| Nor Avg | 1.0 | 1.0 | 1.0 | 0.98 | 0.95 | 0.98 | 0.74 | 0.66 | 0.78 |
| r1 | 51.5 | 28.6 | 8.27 | 50.7 | 21.4 | 8.7 | 34.2 | 18.5 | 6.2 |
| r2 | 254.8 | 156.7 | 24.3 | 258.6 | 154.5 | 24.2 | 189.2 | 111.3 | 19.3 |
| r3 | 311.0 | 218.2 | 30.6 | 306.1 | 211.1 | 30.7 | 243.0 | 134.3 | 24.2 |
| r4 | 475.5 | 368.1 | 36.3 | 484.9 | 372.7 | 36.5 | 348.5 | 248.44 | 30.2 |
| r5 | 440.2 | 320.4 | 34.2 | 428.2 | 311.5 | 32.2 | 329.0 | 243.0 | 26.9 |
| Nor Avg | 1.0 | 1.0 | 1.0 | 0.99 | 0.94 | 0.99 | 0.73 | 0.68 | 0.79 |

the Tables: XVI, XVII, XVIII, XIX, XX and XXI. Tables XVI and XVII consists of 4 columns. The first set of columns presents the results of the initial tree. The second and third sets of columns shows the results for Tree+Link+Elmore sizing and Tree+Link+SPICE sizing respectively. The final set of columns present the results of the proposed algorithm. In Table XVIII, we show results for Tree+Link+Elmore sizing, Tree+Link+SPICE sizing and Tree+Link+SVM sizing. Table XIX consists of three sets of columns, one each for Tree+Link, Tree+Link+Elmore sizing and Tree+Link+SVM sizing. Table XX, shows the result of skew after both stages of optimization in our sizing algorithm. Table XXI, reports the time for SVM modeling. In tables XVII and XIX, we report normalized averages for both the benchmarks separately. The final row in each table shows the the normalized average for the r1-r5 benchmark circuits, the other row with normalized average is for ISCAS89 benchmark. The normalized average is calculated with respect to the results of the initial tree. In tables XVI, XX and XXI, we show the average values. The average or normalized

Table XX. Results of per stage skew improvement for Tree+Link+SVM sizing (skew in ps)

| | Tree+Link | Tree+Link+SVM+1st stage | | Tree+Link+SVM | |
|---|---|---|---|---|---|
| Case | Skew | Skew | Skew Ratio | Skew | Skew Ratio |
| s9234 | 33.9 | 31.2 | 0.92 | 20.5 | 0.60 |
| s5378 | 23.0 | 9.0 | 0.39 | 3.0 | 0.13 |
| s13207 | 162.0 | 130.0 | 0.81 | 111.0 | 0.68 |
| s15850 | 103.5 | 110.3 | 1.06 | 63.2 | 0.61 |
| s38584 | 212.0 | 163.8 | 0.77 | 115.1 | 0.54 |
| s35932 | 172.0 | 168.0 | 0.97 | 129.0 | 0.75 |
| Avg | 117.7 | 102.4 | 0.82 | 73.1 | 0.55 |
| r1 | 16.1 | 15.9 | 0.98 | 8.2 | 0.51 |
| r2 | 143.3 | 115.0 | 0.80 | 96.0 | 0.66 |
| r3 | 164.8 | 146.7 | 0.89 | 110.0 | 0.66 |
| r4 | 334.3 | 270.8 | 0.81 | 198.0 | 0.59 |
| r5 | 256.0 | 249.5 | 0.97 | 202.0 | 0.78 |
| Avg | 182.9 | 160.1 | 0.89 | 122.8 | 0.64 |

average is used to compare our sizing algorithm with the initial tree and the trees obtained using Elmore delay and SPICE based sizing. The SPICE based sizing could not finish optimizing for s35932 and r5 benchmarks. This is indicated by N/A in each table. Thus, we do not calculate the average values for this case.

The results of skew, which is the maximum clock delay difference among all pairs of sinks, are listed in data table XVI. In order to see the impact of our algorithm more clearly, the skew ratio, which is the ratio of skew before the sizing over the skew after the sizing, is also included. The results on resource consumption are included in Table XVII. The power consumption is the total average power consumed by each circuit and is listed in columns 1, 3, 6 and 9 respectively. The total capacitance for each test case, includes both buffer and wire capacitance, is reported in columns 2, 4, 7 and 10 respectively. The CPU time is the time for optimization and is reported in Table XVIII. The initial tree is obtained from [21], thus, it does not have a column for CPU time. The normalized average is thus calculated with respect to our sizing.

Table XXI. Results of CPU time for SVM modeling for Tree+Link+SVM sizing (CPU time in min)

| Case | CPU Time for Optimization | CPU Time for SVM | Ratio(SVM/Total) |
|---|---|---|---|
| s9234 | 9.8 | 8.4 | 0.86 |
| s5378 | 10.9 | 6.6 | 0.66 |
| s13207 | 41.0 | 22.5 | 0.55 |
| s15850 | 39.7 | 32.3 | 0.81 |
| s38584 | 193.6 | 75.6 | 0.39 |
| s35932 | 227.7 | 92.3 | 0.40 |
| Avg | 87.1 | 39.8 | 0.60 |
| r1 | 21.2 | 20 | 0.94 |
| r2 | 44.5 | 43.7 | 0.98 |
| r3 | 79.8 | 55.2 | 0.69 |
| r4 | 267.2 | 71.5 | 0.26 |
| r5 | 280.5 | 205.4 | 0.73 |
| Avg | 138.6 | 79.2 | 0.72 |

The skew due to variation was determined for Tree+Link, Tree+Link+Elmore sizing and Tree+Link+SVM sizing. The result for skew is obtained by running 1000 Monte Carlo simulations for each case. The following parameters are varied (a) Channel Length (b) Threshold voltage and (c) Transistor and Wire widths. The above parameters are varied with mean as nominal value and a standard deviation of 5% is used for parameters (a), (c) and 3% for parameter (b). In Table XIX, we report the maximum and average values of the skew due to variation. We also report the standard deviation (SD) in each case to show the spread from the mean value.

We have two stages of optimization in our approach. Table XX, presents the skew improvement obtained after each stage of optimization separately. The second and third columns report the skew and the skew ratio after both stages of optimization. The skew ratio is calculated with respect to the skew of initial tree. We report these data to show that most of the skew improvements occur during optimization stage 2 of our sizing algorithm. The CPU time taken for SVM modeling is shown in data table XXI. In Table XXI, the second column shows the CPU time for SVM modeling,

the third column shows the ratio, which is the ratio of time taken for SVM modeling over total optimization time. The following observations could be drawn:

- On average, the skew reduction due to our algorithm compared to the initial tree is 45% for ISCAS89 benchmark and 36% for r1-r5. Considering that the granularity of buffer/wire sizes is coarse, 45% (36%) skew reduction is very significant.

- On average, the skew from our sizing using SVM is 44% (35%) smaller than sizing using Elmore delay for ISCAS89 (r1-r5) benchmark. This shows the impact of our algorithm compared to using inaccurate delay model like Elmore delay.

- Our sizing increases the power consumption by a little amount. The total capacitance, is almost not changed by the sizing.

- The average skew due to variation in our algorithm is 34% and 32% smaller than the initial tree for ISCAS89 and r1-r5. The maximum skew reduces by 26% and 27% respectively. The reduction in average and maximum skew compared to Elmore delay sizing are 29% (26%) and 24% (26%) for ISCAS89 (r1-r5) benchmark. This clearly indicates that our approach has more variation tolerance compared to initial tree and the tree obtained after Elmore delay sizing.

- On average, the standard deviation of skew due to variation in our case is smaller than the initial tree by 22%. Compared to Elmore delay sizing it is smaller by 21%. This would lead to better yield of design and less design effort.

- Our sizing algorithm achieves similar skew improvements compared to SPICE based sizing. On average, the CPU runtime for SPICE based optimization is 8.6

and 4.6 times larger than runtime for our sizing algorithm. For larger circuits, the runtime for SPICE based sizing is either unmanageable or does not complete optimizing after a long period of simulation. For s38584, the CPU runtime for optimization is more than 33 hours, r4 takes more than 36 hours to finish optimizing. The test cases s35932 and r5 could not finish optimizing even after simulating for a week. Thus, our algorithm provides an equal or better solution compared to accurate delay model like SPICE with smaller optimization time.

- From data table XX, it can be observed that most of the skew improvements occur during the second stage of optimization. The first stage of optimization is for producing a better initial tree so that the second stage can easily converge to a better solution.

- From the data table XXI, the CPU time taken for SVM modeling is 60% and 72% of the total CPU time of optimization for both the benchmark circuits. Thus, our approach takes smaller time once the SVM model is built.

Although our sizing algorithm is designed mostly for link-based non-tree networks, it can be applied to clock trees as well. The results on trees are given in Tables XXII, XXIII and XXIV. To the best of our knowledge, there is no previous work on discrete buffer/wire sizing with accurate delay models, even for clock trees. Therefore, we perform comparisons for global skew and resource consumption similar to those for non-tree. One can observe trends similar to the cases of link-based non-tree. Compared to the initial tree, our sizing algorithm can reduce the global clock skew by 39% and 33% on average for ISCAS89 and r1-r5 benchmarks. The skew reduction compared to the Elmore delay sizing on an average is 42% and 29% for ISCAS89 and r1-r5 respectively. The impact of our sizing on power consumption is small. Our sizing almost does not change the total capacitance. The CPU time

for optimizing trees is usually less than that for non-trees. One may observe that the clock skews of trees are not much different from that of non-trees. This is because the main advantage of link insertion is on robustness to variations instead of the nominal skew.

Table XXII. Results of global skew for clock tree network (global skew in ps)

| | Tree | Tree+Elmore | | Tree+SPICE | | Tree+SVM | |
|---|---|---|---|---|---|---|---|
| Case | Skew | Skew | Skew Ratio | Skew | Skew Ratio | Skew | Skew Ratio |
| s9234 | 32.1 | 37.6 | 1.17 | 24.3 | 0.75 | 20.2 | 0.63 |
| s5378 | 38.6 | 39.1 | 1.01 | 20.2 | 0.52 | 21.7 | 0.56 |
| s13207 | 175.0 | 178.0 | 1.01 | 130.9 | 0.75 | 117.3 | 0.67 |
| s15850 | 102.1 | 101.0 | 0.99 | 68.3 | 0.67 | 61.5 | 0.60 |
| s38584 | 221.5 | 219.2 | 0.99 | 139.2 | 0.63 | 118.2 | 0.53 |
| s35932 | 172.0 | 170.0 | 0.98 | N/A | N/A | 120.1 | 0.69 |
| Avg | 123.6 | 124.2 | 1.03 | N/A | N/A | 76.5 | 0.61 |
| r1 | 55.2 | 44.6 | 0.80 | 41.0 | 0.74 | 36.1 | 0.65 |
| r2 | 129.8 | 131.7 | 1.01 | 78.2 | 0.60 | 87.5 | 0.67 |
| r3 | 168.3 | 167.5 | 0.99 | 112.4 | 0.66 | 110.4 | 0.65 |
| r4 | 342.5 | 337.4 | 0.98 | 202.4 | 0.60 | 211.0 | 0.61 |
| r5 | 281.0 | 279.1 | 0.99 | N/A | N/A | 215.1 | 0.76 |
| Avg | 195.4 | 192.1 | 0.96 | N/A | N/A | 132.1 | 0.67 |

Table XXIII. Results of resource consumption for clock tree network (power in mW and total cap in pF)

| | Tree | | Tree+Elmore | | Tree+SPICE | | Tree+SVM | |
|---|---|---|---|---|---|---|---|---|
| Case | Pow | Tot Cap | Pow | Tot Cap | Pow | Tot Cap | Pow | Tot Cap |
| s9234 | 1.98 | 1.30 | 1.96 | 1.30 | 2.06 | 1.39 | 2.25 | 1.35 |
| s5378 | 1.83 | 1.11 | 1.68 | 1.10 | 1.66 | 1.12 | 1.83 | 1.12 |
| s13207 | 11.23 | 5.93 | 11.15 | 5.93 | 11.83 | 6.05 | 12.70 | 6.15 |
| s15850 | 9.98 | 6.33 | 10.10 | 6.43 | 10.98 | 6.29 | 10.56 | 6.38 |
| s38584 | 33.80 | 18.65 | 32.90 | 18.94 | 33.91 | 18.40 | 35.30 | 19.04 |
| s35932 | 76.71 | 29.55 | 69.00 | 29.02 | N/A | N/A | 84.00 | 30.02 |
| Nor Avg | 1.0 | 1.0 | 0.96 | 1.01 | 1.03 | 1.01 | 1.07 | 1.01 |
| r1 | 2.55 | 1.47 | 2.70 | 1.47 | 2.60 | 1.48 | 2.90 | 1.45 |
| r2 | 7.95 | 5.19 | 8.42 | 5.21 | 8.23 | 4.10 | 8.60 | 5.69 |
| r3 | 14.10 | 10.72 | 14.40 | 10.76 | 13.90 | 10.63 | 15.10 | 11.25 |
| r4 | 33.80 | 16.17 | 29.67 | 15.85 | 30.20 | 15.92 | 31.20 | 16.17 |
| r5 | 67.60 | 33.32 | 69.20 | 33.37 | N/A | N/A | 70.06 | 33.75 |
| Nor Avg | 1.0 | 1.0 | 1.007 | 0.99 | 0.98 | 0.95 | 1.04 | 1.02 |

Table XXIV. Results of CPU time for clock tree network (min)

| Case | Tree+Elmore | Tree+SPICE | Tree+SVM |
|---|---|---|---|
| s9234 | 3.5 | 12.1 | 8.5 |
| s5378 | 3.9 | 22.7 | 9.2 |
| s13207 | 11.6 | 252.5 | 35.0 |
| s15850 | 11.7 | 910.0 | 32.5 |
| s38584 | 34.4 | 1742.4 | 179.0 |
| s35932 | 32.5 | N/A | 216.2 |
| Nor Ave. | 0.32 | 9.78 | 1.0 |
| r1 | 5.6 | 82.1 | 19.5 |
| r2 | 4.9 | 174.1 | 38.6 |
| r3 | 6.35 | 433.3 | 76.3 |
| r4 | 30.1 | 1528.4 | 262.9 |
| r5 | 44.2i | N/A | 272.3 |
| Nor Ave. | 0.16 | 5.05 | 1.0 |

# CHAPTER V

## CONCLUSION

We presented several techniques to address two keys issues in nanometer IC design: (1) variability and (2) ever diminishing power budget. These techniques have been tested thoroughly using extensive experimentation.

Clock buffer polarity assignment has been shown to be effective in reducing power/ground noise for clock tree network. We propose techniques to reduce the clock network induced power supply noise by assigning different polarities to the clock buffers in an existing clock tree. We detail three different algorithms for the same problem. Experimental results indicate significant reduction in peak current, power supply noise and delay variations. Such reductions in peak current and delay variations lead to more reliable operation of the chip.

We propose an elastic timing scheme which can correct timing errors induced by variations. The correction is performed at runtime and does not cause pipeline stall. Using this scheme can reduce timing safety margins and corresponding power dissipation without sacrificing robustness. Compared to conventional safety margin based approach, the elastic timing scheme can reduce power dissipation by $20\% - 27\%$ on ISCAS89 sequential circuits while retaining similar variation tolerance. Through judicious usage, the area overhead of our approach can usually be controlled within 5%.

Link-based non-tree clock network is an energy-efficient structure for skew tolerance to variations. This work investigates buffer and wire sizing for link based non-tree clock network. In order to handle the difficulty of accurately and efficiently estimating skew in non-tree, a Support Vector Machine based model is employed as a surrogate. The realistic constraint on discrete buffer and wire sizes is considered. A

two-stage hybrid approach is developed to solve the discrete sizing problem, which is normally more difficult than its continuous counterpart. Experiments on benchmark circuits indicate that our method can reduce clock skew significantly according to SPICE simulations.

# REFERENCES

[1] S. R. Nassif, "Design for variability in DSM technologies," in *Proceedings of First International Symposium on Quality Electronic Design*, March 2000, pp. 451-454.

[2] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning DVS processor using delay-error detection and correction," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, April 2006.

[3] R. Saleh, S. Z. Hussain, S. Rochel, and D. Overhauser, "Clock skew verification in the presence of IR-drop in the power distribution network," *IEEE Transactions on Computer-Aided Design*, vol. 19, no. 6, pp. 635–644, June 2000.

[4] G. Bai, S. Bobba, and I. N. Hajj, "Static timing analysis including power supply noise effect on propagation delay in VLSI circuits," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 2001, pp. 295–300.

[5] L. H. Chen, M. Marek-Sadowska, and F. Brewer, "Buffer delay change in the presence of power and ground noise," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 461–473, June 2003.

[6] S. S. Sapatnekar and H. Su, "Analysis and optimization of power grids," *IEEE Design and Test of Computers*, vol. 20, no. 3, pp. 7–15, May-June 2003.

[7] A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska, "Clock skew optimization for ground bounce control," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 1996, pp. 395–399.

[8] L. Benini, P. Vuillod, A. Bogliolo, and G. De Micheli, "Clock skew optimization for peak current reduction," *Journal of VLSI Signal Processing*, vol. 16, no. 2/3, pp. 117–130, June/July 1997.

[9] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao, "Power supply noise suppression via clock skew scheduling," in *Proceedings of the IEEE International Symposium on Quality Electronic Design*, March 2002, pp. 355–360.

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

[11] Z. Yu, M. C. Papaefthymiou, and X. Liu, "Skew spreading for peak current reduction," in *Proceedings of the ACM Great Lakes Symposium on VLSI*, March 2007, pp. 461-464.

[12] M. Grazino, G. Masera, G. Piccinini, and M. Zamboni, "Automated power supply noise reduction via optimized distributed capacitors insertion," in *Proceedings of the IEEE Sothwest Symposium on Mixed-Signal Design*, February 2001, pp. 167-172.

[13] S. Zhao, K. Roy, and C.-K. Koh, "Decoupling capacitance allocation for power supply noise suppression," in *Proceedings of the ACM International Symposium Physical Design*, April 2001, pp. 66-71.

[14] T. Nakura, M. Ikeda, and K. Asada, "Preliminary experiments for power supply noise reduction using stubs," in *Proceedings of the IEEE Asia-Pacific Conference on Advanced System Integrated Circuits*, August 2004, pp. 286-289.

[15] K. Wang, Y. Ran, H. Jiang, and M. Marek-Sadowska, "General skew constrained

clock network sizing based on sequential linear programming," *IEEE Transactions on Computer-Aided Design*, vol. 24, no. 5, pp. 773–782, May 2005.

[16] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: a survey," *Integration: the VLSI Journal*, vol. 19, no. 1-2, pp. 1–81, August 1995.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*. Cambridge, MA: The MIT Press, 2001.

[18] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Mineola, NY: Dover Publications Inc., 1998.

[19] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage, "Skew and delay optimization for reliable buffered clock trees," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 1993, pp. 556–562.

[20] G. Venkataraman, C. N. Sze, and J. Hu, "Skew scheduling and clock routing for improved tolerance to process variations," in *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference*, January 2005, pp. 594–599.

[21] 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 *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design*, November 2005, pp. 591–595.

[22] GSRC Bookshelf, "BST: Bounded-Skew Clock tree Routing," 2000 [online], Available: *http* : *//vlsicad.ucsd.edu/GSRC/bookshelf/slot/BST*.

[23] VLSI Computer Architecture Research, Stillwater, OK, "FreePDK: Unleashing VLSI to the Masses," 2005 [online], Available: *http* : *//vcag.ecen.okstate.edu/projects/scells/download/*.

[24] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, and A. B. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 39, no. 11, pp. 799–814, November, 1992.

[25] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit simulation," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, May 2000, pp. 201–204.

[26] H. Chang and S. S. Sapatnekar, "Statistical timing analysis under spatial correlations," *IEEE Transactions on Computer-Aided Design*, vol. 24, no. 9, pp. 1467–1482, September 2005.

[27] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 2004, pp. 331–336.

[28] S. Raj, S. B. K. Vrudhula, and J. Wang, "A methodology to improve timing yield in the presence of process variations," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 2004, pp. 448–453.

[29] M. Mani, A. Devgan, and M. Orshansky, "An efficient algorithm for statistical minimization of total power under timing yield constraints," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 2005, pp. 309–314.

[30] J. W. Tschanz, S. G. Narendra, R. Nair, and V. De, "Effectiveness of adaptive supply voltage and body bias for reducing impact of parameter variations in low power and high performance microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 5, pp. 826–829, May 2003.

[31] T. Chen and S. Naffziger, "Comparison of adaptive body bias (ABB) and adaptive supply voltage (ASV) for improving delay and leakage under the presence of process variation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 5, pp. 888–899, October 2003.

[32] S. H. Kulkarni, A. N. Srivastava, and D. Sylvester, "A new algorithm for improved VDD assignment in low power dual VDD systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, August 2004, pp. 200–205.

[33] K. Agarwal, K. Nowka, H. Deogun, and D. Sylvester, "Power gating with multiple sleep modes," in *Proceedings of the IEEE International Symposium on Quality Electronic Design*, March 2006, pp. 633–637.

[34] C. Long and L. He, "Distributed sleep transistor network for power reduction," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 9, pp. 937–946, September 2004.

[35] J.-L. Tsai, D. H. Baik, C. C.-P. Chen, and K. K. Saluja, "A yield improvement methodology using pre- and post-silicon statistical clock scheduling," in *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design*, November 2004, pp. 611–618.

[36] 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," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, May 2001.

[37] A. Rajaram, J. Hu, and R. Mahapatra, "Reducing clock skew variability via cross links," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 2004, pp. 18–23.

[38] A. Rajaram, D. Pan, and J. Hu, "Improved algorithms for link based non-tree clock network for skew variability reduction," in *Proceedings of the ACM International Symposium on Physical Design*, April 2005, pp. 55–62.

[39] D. Lam, C.-K. Koh, Y. Chen, J. Jain, and V. Balakrishnan, "Statistical based link insertion for robust clock network design," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 2005, pp. 588–591.

[40] A. Rajaram and D. Z. Pan, "Fast incremental link insertion in clock networks for skew variability reduction," in *Proceedings of the IEEE International Symposium on Quality Electronic Design*, March 2006, pp. 79–846.

[41] A. Rajaram and D. Z. Pan, "Variation tolerant buffered clock network with cross links," in *Proceedings of the ACM International Symposium on Physical Design*, April 2006, pp. 157–164.

[42] A. Vittal and M. Marek-Sadowska, "Power optimal buffered clock tree design," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 1995, pp. 497–502.

[43] J. G. Xi and W. W.-M. Dai, "Buffer insertion and sizing under process variations for low power clock distribution," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 1995, pp 491–496.

[44] Q. Zhu and W. W.-M. Dai, "High-speed clock network sizing optimization based

distributed RC and lossy RLC interconnect models," *IEEE Transactions on Computer-Aided Design*, vol. 15, no. 9, pp. 1106–1118, September 1996.

[45] S. Pullela, N. Menezes, and L. T. Pileggi, "Moment-sensitivity-based wire sizing for skew reduction in on-chip clock nets," *IEEE Transactions on Computer-Aided Design*, vol. 16, no. 2, pp. 210–215, February 1997.

[46] J.-L. Tsai, T.-H. Chen, and C. C.-P. Chen, "$\varepsilon$-optimal minimum-delay/area zero-skew clock tree wire-sizing in pseudo-polynomial time," in *Proceedings of the ACM International Symposium on Physical Design*, April 2003, pp. 166–173.

[47] M. P. Desai, R. Cvijetic, and J. Jensen, "Sizing of clock distribution networks for high performance CPU chips," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 1996, pp. 389–394.

[48] H. Su and S. S. Sapatnekar, "Hybrid structured clock network construction," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, November 2001, pp. 333–336.

[49] I-M. Liu, T.-L. Chou, A. Aziz, and D. F. Wong, "Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion," in *Proceedings of the ACM International Symposium on Physical Design*, April 2000, pp. 33–38.

[50] M. R. Guthaus, D. Sylvester, and R. B. Brown, "Clock buffer and wire sizing using sequential programming," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 2006, pp. 1041–1046.

[51] O. Coudert, "Gate sizing for constrained delay/power/area optimization," *IEEE Transactions on VLSI Systems*, vol. 5, no. 4, pp. 465–472, December 1997.

[52] V. N. Vapnik, *Statistical Learning Theory.* New York: John Wiley & Sons Inc., 1998.

[53] T. Joachims, "Making large-scale SVM learning practical," in *Advances in Kernel Methods: Support Vector Learning*, B. Scholkopf, C.J.C. Burges, A.J. Smola, Eds. Cambridge, MA: MIT-Press, 1998.

[54] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, no. 2, pp. 291–307, February 1970.

[55] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, *SIS: a system for sequential circuit synthesis.* Memorandum no. M92/41, ERL, University of California, Berkeley, May 1992.

[56] "CPMO-constrained placement by multilevel optimization," Computer Science Department, State University, Los Angeles, (2005). [online]. Available: http://ballade.cs.ucla.edu/cpmo/.

[57] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit simulation," in *Proceedings of the IEEE Custom Integrated Circuit Conference*, May 2000, pp. 201–204.

# VITA

Rupak Samanta received his B.E. (hons) in electrical engineering from Sambalpur University, Burla, India in 2000. He completed his M.Tech. degree from Indian Institute of Technology (IIT), Mumbai in the year 2003. Rupak worked as a Design Engineer in Agere Systems between February 2003 and July 2004. He worked as an intern at Intel Austin from August 2007 to July 2008. His research interests include low power design, variation-tolerant circuit design and physical design. His works have been published in prestigious conferences and journals including IEEE Transactions in VLSI Systems (TVLSI), International Conference on Computer-Aided Design (ICCAD), International Symposium on Physical Design (ISPD), International Symposium on Quality Electronic Design (ISQED) and International Symposium on Defect and Fault Tolerance in VLSI Systems (DFTS).

Mr. Samanta may be reached at Department of Electrical and Computer Engineering, C/O Dr. Jiang Hu, Texas A&M University, College Station, Texas 77843-3128. His email is rupaksamanta@gmail.com.

The typist for this dissertation was Rupak Samanta.