Design and Optimization for Timing-Speculative Circuits

YE, Rong

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science and Engineering

The Chinese University of Hong Kong

April 2014

Thesis/Assessment Committee

Professor LYU Rung Tsong Michael (Chair)
Professor XU Qiang (Thesis Supervisor)
Professor LEE Pak Ching (Committee Member)
Professor XIE Yuan (External Examiner)

Abstract

As circuit non-idealities inevitably worsen with technology scaling, more design resource has to be incorporated to ensure integrated circuit (IC) timing correctness. Such worst-case-oriented design methodology results in pessimistic designs with considerable power and performance overheads, lessening the benefits provided by technology scaling.

Better-than-worst-case (BTWC) design methodology that allows reliability to be traded off against power and performance was proposed to dramatically improve the computation energy-efficiency. The basic idea behind BTWC design methodology is that, since circuit non-idealities mainly manifest themselves as infrequent timing errors on critical paths of the circuit, we can over-clock operating frequency and/or over-scale supply voltage of the chip to a critical point, where timing errors occur, and achieve error-resilient computations by performing timing error detection and correction. This approach is generally referred to as *timing speculation*, with which it is not necessary to guarantee "always correct" operations. Unfortunately, there is usually a "wall of critical paths" in the final implementation of a circuit caused by conventional worst-case-oriented design methodology, suggesting that, given a fixed circuit design, the effectiveness of timing speculation is limited by a fixed threshold beyond which the circuit performance/energy efficiency will drop significantly.

To address the above problem, this thesis first proposes to study the premises

and prospects of timing speculation by analyzing the minimum and maximum potential benefits that are achievable by timing speculation techniques. After answering the question posed by the conflict between conventional techniques and timing speculation, this thesis investigates multiple design and optimization problems in timing-speculative circuits. Firstly, as introducing timing speculation capability into circuits can naturally extend the flexibility of multi-supply voltage (MSV) designs to a new horizon, this thesis formulates the MSV design problem for timing-speculative circuits and develops a novel algorithm based on dynamic programming to solve it. Secondly, this thesis develops a general formulation of clock skew scheduling (CSS) problem for timing-speculative circuits, wherein timing error rate and its corresponding impact are explicitly considered, and proposes novel algorithms to tackle this problem. Finally, considering the impact of timing uncertainties caused by process variation and wearout effects, which is very difficult to be modeled and addressed at design stage, this thesis also develops a novel online clock skew tuning framework for timing-speculative circuits. By utilizing an elaborately-designed hardware architecture to collect timing error information and tune clock skews at runtime, variation effects can be effectively mitigated.

摘要

隨著半導體工藝技術的不斷進步 (technology scaling)，更多的設計資源不得不用於確保集成電路的時序正確性。這種“面向最壞情況”(worst-case-oriented) 的芯片設計方法導致了悲觀保守的芯片設計方案，增加了性能及功耗開銷，減少了工藝進步帶來的效益。

“優於最壞情況”(better-than-worst-case) 的芯片設計方法允許犧牲一定的芯片可靠性 (reliability) 來提高性能以及降低功耗，從而提高計算的能量效率 (energy efficiency)。“優於最壞情況”設計方法的核心思想在於放松對芯片可靠性的硬性需求。既然時序錯誤 (timing error) 在關鍵路徑中的發生頻率並不高，我們可以允許錯誤發生，從而節約用於防止錯誤發生所需要的高額開銷。而當錯誤發生時，再利用錯誤檢測和更正方法 (error detection and correction) 來消除錯誤造成的影響。這種無須保證計算過程永遠正確無誤的方法通常被稱作“時序推測”(timing speculation)。然而，不幸的是，由於傳統的“面向最壞情況”的設計方法往往導致芯片中存在所謂的“關鍵路徑壁壘”(wall of critical paths)，時序推測技術的有效性在一定程度上受限。

為了解決上述問題，我們首先研究了時序推測技術的前提與前景，也就是研究了如何估計時序推測技術能夠帶來的最小和最大效益。此外，我們也研究了時序推測芯片 (timing-speculative circuit) 中的若幹設計優化問題。首先，由於引入時序推測技術能夠提高多電壓 (multi-supply voltage) 技術的靈活性，我們闡述了時序推測芯片中的多電壓設計問題，並創造

性地提出了一種基於動態規劃 (dynamic programming) 的算法來解決這個問題。此外，我們提出了時序推測芯片中的時鐘差異規劃 (clock skew scheduling) 問題。在考慮了時序錯誤率 (timing error rate) 等因素的影響後，我們設計了新穎有效的方法來解決該問題。最後，鑒於工藝差異 (process variation) 和老化效應 (wearout effect) 對芯片時序的影響，而且這種影響很難在設計階段被消除，我們提出了一種實時的時序差異調整 (clock skew tuning) 架構。利用精心設計的硬件結構，我們可以實時地收集時序錯誤的信息，相應地調整時鐘差異，從而極大地減弱了時序不確定性對芯片性能的影響。

Acknowledgement

This thesis represents not only the end of my journey in obtaining my Ph.D. but also a starting point of my future work. Needless to say, it is a beautiful moment of my student experience and a milestone in my entire life.

This thesis has been seen through to completion with the support and encouragement of numerous people including my colleagues, friends, and families. At the end of my Ph.D. study, it is a pleasant task to express my thanks to all those who contributed in many ways to the accomplishment of this study and made this thesis possible.

My first debt of gratitude must go to my advisor, Professor Qiang Xu. This thesis could not have been finished without his insightful guidance and endless support. In the past four and a half years, he has always inspired my research with numerous ideas, leading a junior member of academia like me to a completely new world of research.

I am also extremely indebted to Professor Rakesh Kumar for providing his support and guidance that are particularly valuable during my half-year visit at UIUC.

I gratefully thank my committee members, Professor Michael R. Lyu and Professor Patrick P. C. Lee, for their constructive comments that significantly improve my thesis research. In particular, I could not forget my first chat with Professor Michael R. Lyu when I was still a fresh Ph.D. student. I would like to thank him

for sharing his insights that have always been helpful during my subsequent study.

I take this opportunity to thank Professor Yuan Xie from Pennsylvania State University for serving as my external marker and examining my thesis.

My colleagues and friends in 506 EDA Lab have consistently been sources of laughter, joy and support. To Lei Shi, I would like to thank him for teaching me the knowledge on machine learning. To Xiaoqing Yang, I would like to thank her for her comfort. To Yubin Zhang, I would like to thank him for his companion in my very rare physical activities and wish his lovely daughter a forever happy life. To Feng Yuan, I would like to thank him for his great help. The discussion with him could always overcome difficulties in my research. To Lin Huang, I would like to thank her for the enlightenment in my early research. She has served as a role model to me. To Xiao Liu, I would like to thank him for teaching me EDA tools. To Li Jiang, I would like to thank him for my first publication. I hope he would forget the "three cups" issue. To Jie Zhang, I would like to thank him for his kindness. Do remember our tacit agreement. To Zelong Sun, I would like to thank him for taking care of me when I was feeling ill. His various "theories" are always "misleading". To all my friends in 506 EDA Lab, Qiang Ma, Liang Li, Yan Jiang, Linfu Xiao, Jing Qin, Zaichen Qian, Guxin Cui, Xing Wei, Yi Diao, Xu He, Haile Yu, Yuxi Liu, Chenfei Ma, Qian Zhang, Ting Wang, Yannan Liu, Lingxiao Wei, and Fuyang Huang, sincerely thank you all for entering my life.

My special thanks go to my parents. I would remember your love and care for ever. Without you, nothing is possible. Last but not least, I convey special acknowledgement to my girlfriend, Hengsha. During the inevitable ups and downs, I believe it is my fortune to meet you and have you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preface

This thesis presents my research work in the past four and a half years, for the partial fulfilment of the requirements of my Ph.D. degree in Computer Science and Engineering at The Chinese University of Hong Kong (CUHK).

This thesis primarily consists of two parts. The first part (including Chapter 2) investigates the premises and prospects of timing speculation techniques, building up the fundamental of this thesis research and revealing its feasibility. The second part (including Chapter 3$\sim$5) studies the design and optimization problems for timing speculation that provide both offline and online solutions to realize a practical implementation of timing-speculative circuit. Every chapter in this thesis is largely self-contained. The notations defined in each chapter is applicable for that chapter only.

The first part of this thesis originates from the following work:

- **R. Ye**, F. Yuan and Q. Xu, "On the Premises and Prospects of Timing Speculation", submitted to ACM/IEEE Design Automation Conference (DAC), 2014.

The second part is mainly comprised of the following works:

- **R. Ye**, F. Yuan, H. Zhou and Q. Xu, "Clock Skew Scheduling for Timing-Speculative Circuits", submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD).
- **R. Ye**, F. Yuan and Q. Xu, "Online Clock Skew Tuning for Timing-Speculative Circuits", submitted to IEEE Transactions on Computers (TC).
- **R. Ye**, F. Yuan, Z. Sun, W.-B. Jone, and Q. Xu, "Post-Placement Voltage Island Generation for Timing-Speculative Circuits", Proc. ACM/IEEE Design Automation Conference (DAC), June 2013.
- **R. Ye**, F. Yuan, H. Zhou and Q. Xu, "Clock Skew Scheduling for Timing Speculation", Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE), Mar. 2012.
- **R. Ye**, F. Yuan and Q. Xu, "Online Clock Skew Tuning for Timing Speculation", Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2011.

In addition, there are a number of research works/publications that are not included in this thesis. They are:

- **R. Ye** and Q. Xu, "Learning-Based Power Management for Multi-Core Processors via Idle Period Manipulation", submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD).
- **R. Ye** and Q. Xu, "Energy-Efficient Design Techniques", Energy-Efficient Fault-Tolerant Systems, J. Mathew, A.R. Shafik and K.D. Pradhan eds., Springer, 2013, ISBN 978-1-4614-4192-2.
- **R. Ye**, T. Wang, F. Yuan, R. Kumar and Q. Xu, "On Reconfiguration-Oriented Approximate Adder Design and Its Application", Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2013.

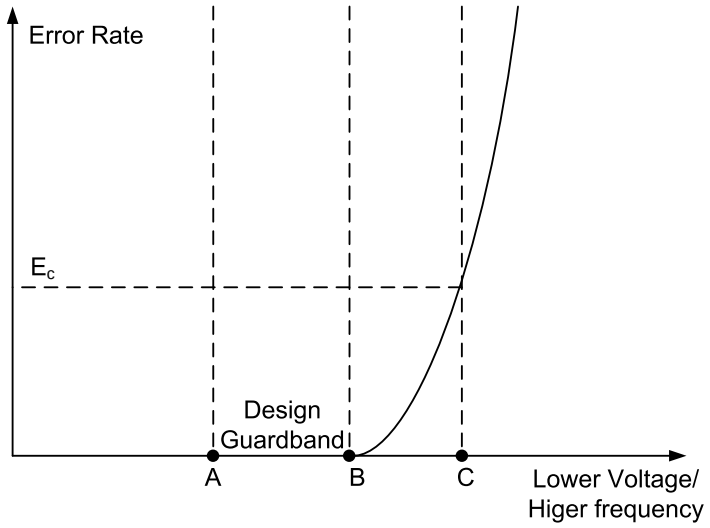
- **R. Ye** and Q. Xu, "Learning-Based Power Management for Multi-Core Processors via Idle Period Manipulation", Proc. IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), Jan. 2012. **(Best Paper Nomination)**
- Q. Zhang, F. Yuan, **R. Ye** and Q. Xu, "ApproxIt: An Approximate Computing Framework for Iterative Methods", submitted to ACM/IEEE Design Automation Conference (DAC), 2014.
- J. Zhang, F. Yuan, **R. Ye**, and Q. Xu, "ForTER: A Forward Error Correction Scheme for Timing Error Resilience", Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2013.
- Y. Liu, **R. Ye**, F. Yuan and Q. Xu, "Optimization for Timing-Speculated Circuits by Redundancy Addition and Removal", Proc. IEEE European Test Symposium (ETS), May 2013.
- Y. Liu, **R. Ye**, F. Yuan, R. Kumar and Q. Xu, "On Logic Synthesis for Timing Speculation", Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Nov. 2012.
- L. Huang, **R. Ye** and Q. Xu, "Customer-Aware Task Allocation and Scheduling for Multi-Mode MPSoCs", Proc. ACM/IEEE Design Automation Conference (DAC), pp. 387-392, June 2011.
- L. Jiang, **R. Ye** and Q. Xu, "Yield Enhancement for 3D-Stacked Memory by Redundancy Sharing across Dies", Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 230-234, Nov. 2010. **(Best Paper Nomination)**

1.2 Background

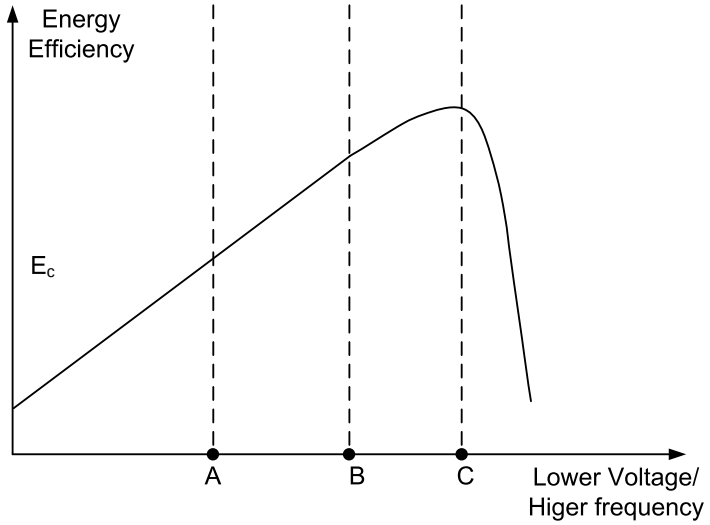
With the continuous downscaling of transistor feature size, a significant amount of research effort has been attracted by the increasingly severe uncertainties of the timing behavior of today's integrated circuits (ICs), mainly manifesting themselves as infrequent timing errors on speed-paths (i.e., critical or near-critical paths). There are multiple factors that contribute to this effect: (i) inevitable static process variation caused by manufacturing imperfection leads to the mismatch of timing performance between the designed value and the actual one; (ii) dynamic variations in supply voltage, temperature, and multiple-input switching cause varying circuit delay at runtime; (iii) circuit aging mechanisms such as hot carrier injection (HCI) and negative-bias temperature instability (NBTI) lead to gradual increase of circuit delay over its lifetime.

Facing these timing uncertainties, conventional IC designs try all means to achieve error-free computations, even under worst-case combinations of process, voltage, and temperature (PVT) variations and wearout effects [1, 2]. As the above circuit non-idealities inevitably worsen with technology scaling [3], more design guardband has to be incorporated to ensure IC timing correctness. Consequently, such worst-case-oriented design methodology results in pessimistic designs with considerable power and performance overheads [4], lessening the benefits provided by technology scaling. As can be seen in Fig. 1.1, even though a particular circuit may operate at point B without timing errors, during the design phase we have to conservatively let the circuit work at point A with lower frequency and/or higher supply voltage to ensure its timing correctness throughout its service life. The significant performance/energy difference between Point A and Point B is used as design guardband.

To address the above problem of pessimistic designs, *better-than-worst-case* (BTWC) design methodology that allows reliability to be traded off against power



(a) Error rate of timing-speculative circuits.



(b) Energy efficiency of timing-speculative circuits.

Figure 1.1: Motivation of timing speculation.

and performance was proposed to dramatically improve the energy efficiency of computations [5, 6]. The basic idea behind BTWC design methodology is that, since circuit non-idealities mainly manifest themselves as infrequent timing errors on critical paths of the circuit (if sufficient design guardband is not incorpo-

rated) [7], we can over-clock the operating frequency and/or over-scale the supply voltage of the chip to a critical point, where timing errors occur, and then achieve resilient computations (instead of error-free computations) by performing timing error detection and correction. This approach is generally referred to as *timing speculation* (TS). As can be seen in Fig. 1.1, a timing-speculative circuit (i.e., a circuit that is equipped with timing speculation capability) can operate at point C with much higher frequency or much lower supply voltage, thus greatly improving the circuit's energy-efficiency even after compensating the performance and energy penalties caused by timing detection and correction. Due to this significant benefit, timing speculation techniques have attracted lots of research interests from both academia and industry [8].

As the necessary components of timing speculation, both timing error detection and correction techniques (including error recovery and error masking) should be elaborately designed. These techniques presented in the literature would be discussed as follows.

1.2.1 Timing Error Detection

There are many timing error detectors presented in the literature (e.g., [9, 10]), and most of them are based on monitoring signal transitions on speed-paths for a specified clock period after the clock edge.

Without loss of generality, let us discuss one of the most representative timing speculation techniques, *Razor* [9, 10], to illustrate how error-resilient computations can be achieved with timing speculation. To detect timing errors on critical paths, the receiving ends of critical paths, referred to as *suspicious flip-flops* (SFF), are replaced with *Razor flip-flops* (RFFs), which includes a main flip-flop (FF), an additional shadow latch and some control logic (see Fig. 1.2). The main flip-flop latches the output signal at the clock edge with possible timing error, while the

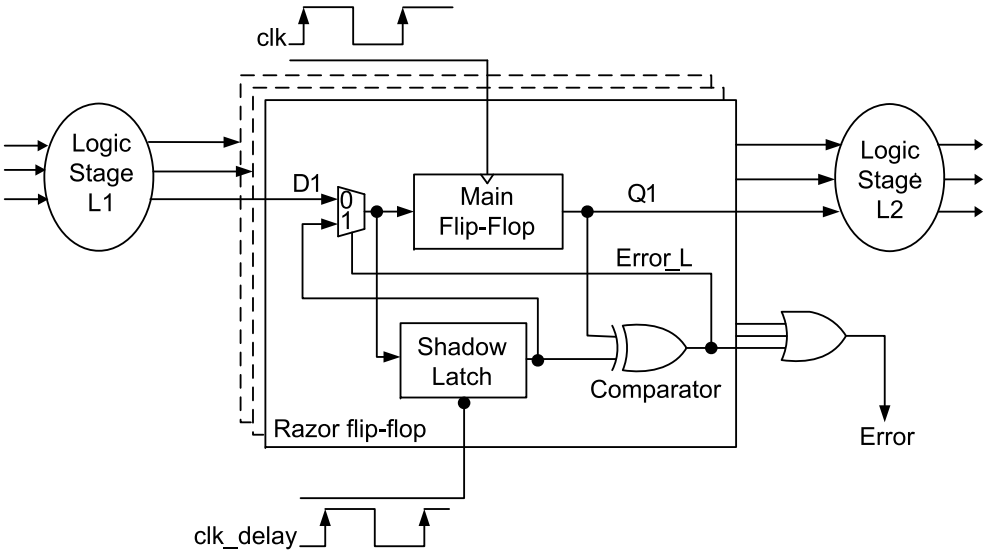


Figure 1.2: The design of Razor flip-flop.

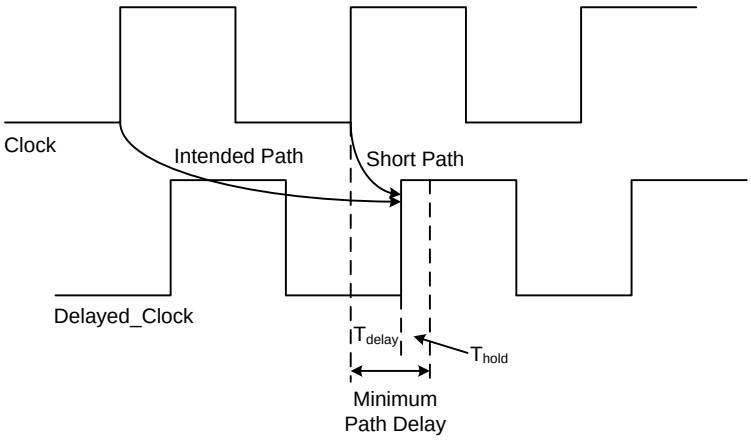


Figure 1.3: The short path problem of timing speculation.

shadow latch, controlled by a delayed clock signal, latches the signal a fraction of a clock cycle later, which guarantees to receive the correct value. Consequently, when the values of the shadow latch and the main flip-flop do not agree with each other, indicated by the comparator, a timing error is detected.

To detect timing errors on speed-paths as above, obviously, all the receiving ends of speed-paths, i.e., suspicious flip-flops, should be equipped with Razor-like

timing error detectors. There require, however, extra hardware resources to enable such double-sampling-based error-detection techniques. As discussed earlier, a certain checking period (or timing window) right after the clock edge is set to monitor late transition on suspicious flip-flops, and any transition in this timing window is regarded as timing error. It is therefore a critical issue to distinguish the late coming transitions caused by timing errors from those transitions generated by short paths, as shown in Fig. 1.3, so that every timing error detection is assured to be correct and reliable. Such a problem is known as *short path problem* of timing speculation, which can be solved by padding short paths [11] with buffers until the delay of a suspicious flip-flop is long enough to satisfy the hold time constraint of the delayed clock, for any path that connects to this suspicious flip-flop.

1.2.2 Timing Error Recovery

One widely-used error recovery scheme is to restore the system to a known-good pre-error state when timing error is detected. Razor [9] first implemented such a recovery scheme for timing errors with microarchitectural support. That is, when a timing error is detected in a Razor flip-flop, the processor pipeline is flushed and the correct result from the shadow latch is inserted back into the pipeline. Then, by replaying instructions (at possibly lower operating frequency), the processor operates correctly with little performance penalty. Furthermore, by taking timing error rate into consideration, voltage-scaling is utilized to allow processor to run robustly at the edge of minimum power consumption, with occasional timing error recovery for heavyweight computations. With the above techniques, Razor enables better than worst-case design by removing design guardband that is used to guarantee “always correct” operations, and has inspired a large amount of subsequent research work (e.g., [12–14]).

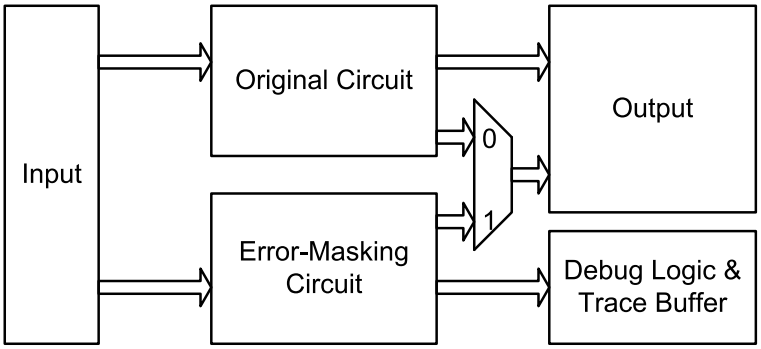


Figure 1.4: The conceptual framework of logical error masking.

1.2.3 Timing Error Masking

Albeit Razor-like timing speculation techniques are very effective for timing error correction in microprocessors with the help of instruction replay, they are very difficult, if not impossible, to be applied to general logic circuits, due to the high cost to checkpoint error-free states in them. It is therefore imperative to develop in-situ timing error correction techniques that are able to mask errors without any rollback. There are a few such techniques presented in the literature and they can be classified into two categories: logical error masking and temporal error masking.

1.2.3.1 Logical Error Masking

Logical error masking is based on synthesizing a circuit, referred to as *error-masking circuit*, that correctly predicts the outputs of the circuit upon application of inputs that sensitize the speed-paths of the circuit [15]. As demonstrated in Fig. 1.4, a redundant logic block is constructed to predict the outputs of the original circuit. With this exact sensitization constraint, the error-masking circuit tends to have more timing slack when compared to the original circuit, and hence is immune to timing errors. Albeit the idea is very interesting, to synthesize such

error-masking circuits, we need to obtain the characteristic function for the set of all speed-path activation patterns, which is only practical for small circuit blocks.

1.2.3.2 Temporal Error Masking

Temporal error masking techniques correct timing errors by delaying the arrival time of the correct data to the next logic level. There have been several temporal error masking techniques proposed in the literature (e.g., [16–18]).

To be specific, in [16] Kurimoto *et al.* proposed to stall the clock for one cycle after detecting a timing error to correct the circuit states. The assumption that the clock signal can be safely stalled within one clock cycle without corrupting circuit states, however, is usually impractical due to the fact that the latency involved in consolidating error signals can be much larger than the cycle time for reasonable-sized design. In [17], Hirose *et al.* used a delayed clock to re-sample and correct the data-path value by borrowing time from the next logic level when a timing error is detected. This technique assumes that all the sensitizable paths in the next logic level of a suspicious FF are non-critical and they can lend sufficient time for timing error correction, which is often not the case in practical circuits. To tackle this problem, Choudhury and Mohanram [18] proposed a so-called TIMBER technique that is able to recover from two-stage timing errors. Two kinds of sequential elements, namely TIMBER flip-flop and TIMBER latch, are implemented in this architecture to replace suspicious flip-flops with time-borrowing capability.

Albeit the above time-borrowing techniques are able to mask timing errors in suspicious flip-flops occurred during the checking period (i.e., the difference between the original clock signal and the delayed clock signal), the timing slack of their successive logic levels is reduced, and hence some initially non-suspicious flip-flops may become suspicious ones and need to be replaced by sequential elements with time-borrowing capability. Due to this propagation effect, the hard-

ware cost for such temporal error masking techniques can be quite high. In addition, there is clearly a trade-off between timing error detection capability and the area/power overhead of time-borrowing based error masking techniques. That is, higher timing error detection capability requires larger checking period, therefore increasing the number of suspicious flip-flops.

1.3 Motivation and Contributions

Timing error correction inevitably incurs extra performance loss and energy consumption. As can be observed in Fig. 1.1, further increase of frequency and/or decrease of voltage beyond point C will hurt system performance/energy efficiency, because too many system rollbacks are required to correct the massive timing errors. It is therefore very essential to optimize timing-speculative circuits by reducing timing error rate (TER) [12].

Various techniques have been presented for optimizing timing-speculative circuits in the literature. The key issue for this optimization problem is to reshape the path delay distribution of the circuit so that those frequently-sensitized timing paths are optimized to have more timing slack while the other paths are allowed to have less timing slack and even suffer from timing errors. EVAL [13] proposes a so-called high-dimensional dynamic adaptation technique that trades error rate for processor frequency by tilting, shifting, or reshaping the path distributions of various functional units. Blueshift [14] identifies and optimizes the most frequently sensitized critical paths by on-demand selective biasing and path constraint tuning. DynaTune [19] optimizes the most frequently sensitized critical paths of the circuit by assigning low threshold voltage to those critical gates that are strongly related to the occurrence of timing errors.

The above techniques are helpful for TER reduction, but one common limitation is that they conduct optimization on top of a given circuit netlist and hence

are not capable of manipulating the logic structure of the circuit. In [20], the authors attempted to conduct logic synthesis for BTWC designs. They constructed a simple timing error probability model and used it to guide the "balance" logic optimization step, which is a logic decomposition method initially used for delay minimization [21]. The effectiveness of this solution, however, is not very impressive from their experimental results, likely due to the lack of accuracy of the unvalidated timing error model and the simple strategy to include timing errors into optimization cost function only.

Unfortunately, there is usually a "wall of critical paths" (i.e., timing wall) in the final implementation of a circuit caused by the conventional worst-case-oriented design methodology. This is due to the nature of today's IC design and optimization flow, e.g., gates on those initially non-critical paths are often downsized to trade off gate delay for power and area, making many non-critical paths become critical. This suggests that, given a fixed circuit design, the effectiveness of timing speculation techniques is limited by a fixed threshold beyond which the circuit performance/energy efficiency will drop dramatically.

To address the above problem of "timing wall", this thesis first studies the premises and prospects of timing speculation, as it is not orthogonal to other circuit-level power optimization techniques. For example, given a circuit netlist, we could downsize those gates on non-critical paths for power reduction [22], which, however, increases the height of the timing wall with more speed-paths in the circuit. Alternatively, we could as well upsize those gates on frequently-sensitized speed-paths and turn them into non-critical paths for effective timing speculation. Since both methods can reduce power consumption and their impacts are interrelated, an interesting question is emerging. That is, *given a circuit netlist, whether the potential energy efficiency gains provided by timing speculation (over conventional circuit-level power optimization techniques) is significant enough to*

warrant the effort to make it timing-speculative? This fundamental problem, although important and relevant, has not been explicitly investigated in the literature, motivating this thesis to directly investigate it first.

After that, this thesis also presents and addresses a number of design and optimization problems to realize a practical implementation of timing-speculative circuit as follows.

Voltage Island Generation: Motivated by the fact that individual blocks of a circuit can have timing/power characteristics unique from the rest of the design, the concept of multi-supply voltage (MSV) design was leveraged to trade off power consumption and performance. Introducing timing speculation capability into circuits can naturally extend the flexibility of MSV designs to a new horizon. Consequently, this thesis formulates the post-placement MSV problem for timing-speculative circuits and develop a novel algorithm based on dynamic programming to solve it. The proposed algorithm can guarantee larger solution space is explored with similar algorithm runtime and meanwhile the optimization result is always improved step by step.

Clock Skew Scheduling: Clock skew Scheduling (CSS), treating clock skew as a manageable resource instead of design liability, is an effective technique to improve IC performance, by assigning intentional clock arrival times to flip-flops (FFs). This thesis develops a general formulation of clock skew scheduling (CSS) problem for timing-speculative circuits, wherein timing error rate and its corresponding impact are explicitly considered, and proposes novel algorithms to tackle this problem. The proposed algorithms are based on gradient-descent method (GDM) and steepest-descent method (SDM) that are classic techniques used to solve non-linear optimization problems.

Online Clock Skew Tuning: Considering the impact of process variation, which is really difficult to be modeled and addressed at design stage, this thesis

also develops a novel online clock skew tuning framework for timing-speculative circuits. Specifically, a novel hardware architecture is designed to collect timing error information and tune clock skews at runtime. By elaborately compensating process variation and/or aging effects, significant performance improvement can be achieved when compared to a fixed skew setting that is optimized at design stage.

1.4 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 investigates the premises and prospects of timing speculation by studying the minimum and maximum potential benefits that are achievable by conducting timing speculation. This work answers the question posed by the conflict between conventional techniques and timing speculation, and identifies which one is preferable for a given circuit netlist in terms of energy efficiency. Chapter 3 presents the post-placement voltage island design problem of timing-speculative circuits, wherein individual voltage island has its own supply voltage that is not necessary to satisfy timing constraint. Next, the offline clock skew scheduling and online clock skew tuning problems are detailed in Chapter 4 and Chapter 5, respectively. By explicitly considering timing error rate and its corresponding impact, assigning intentional clock skews is exploited to be an effective technique to improve timing speculation. Finally, Chapter 6 concludes this thesis and points out the directions of future research.

□ End of chapter.

Chapter 2

The Premises and Prospects

2.1 Introduction

Power minimization is a primary objective in the design of integrated circuits (ICs) nowadays, which is achieved with CMOS technology scaling and low-power design techniques at various levels. With techniques such as multi-threshold logic, gate sizing for power reduction, clock and power gating, dynamic power management, and dynamic voltage and frequency scaling (DVFS) already adopted in commercial IC products for power optimization [23], designers mainly rely on technology scaling to further improve the energy efficiency of VLSI circuits.

However, energy efficiency from aggressive technology scaling itself is showing diminishing improvements. This is because, even though the power consumption of individual devices continues to reduce with technology scaling, the increasing static and dynamic variation effects (e.g., manufacturing variability, temperature/voltage fluctuations, and circuit aging) associated with scaling [24, 25] require us to reserve increasingly large design guardband to ensure "always correct" operations, even under the worst-case combinations of the above variation effects.

Prior works suggest that the best and probably the only effective way to achieve

power reduction at design time is to eliminate waste whenever possible. In existing worst-case oriented designs, much energy is wasted to guarantee "*error-free*" computations. If we can over-clock the frequency and/or reduce the supply voltage of the circuit with correct computational results under nominal timing conditions while conducting online error detection and correction when timing errors occur under worst-case conditions, the potential circuit energy efficiency gain can be significant. Such *better-than-worst-case* (BTWC) design methodology [26] thus received lots of research attention, wherein the key enabling technique used to effectively tradeoff reliability with performance/power of the circuit to achieve "*error-resilient*" computations is called *timing speculation* (TS) [9, 12, 14, 19, 27, 28].

For a well-tuned circuit, there usually exists a large number of speed-paths (i.e., critical or near-critical paths) after timing and power optimization, which manifest themselves as a wall in the timing slack histogram, referred to as "*timing wall*" [29]. Such phenomenon, however, limits the effectiveness of timing speculation due to the performance/energy penalties associated with timing error correction [30]. That is, when errors occur in a timing-speculative circuit, the system needs to be rolled back to a pre-error state for re-computation (usually with slower frequency), which incurs both performance penalty and extra energy consumption. Consequently, a timing-speculative circuit needs to operate at a voltage-frequency combination with small timing error rate (TER). The timing wall basically dictates the threshold beyond which there are massive amount of timing errors and the associated penalties would outweigh its benefits. To mitigate this issue, a number of optimization techniques for timing-speculative circuits were proposed to reshape the circuit path delay distribution for effective timing speculation [11, 13, 31–33].

Timing speculation is *not* orthogonal to other circuit-level power optimization techniques. For example, given a circuit netlist, we could downsize those gates on non-critical paths for power reduction [22], which, however, increases the

height of the timing wall with more speed-paths in the circuit. Alternatively, we could upsize those gates on frequently-sensitized speed-paths and turn them into non-critical paths for effective timing speculation. As both methods can reduce power consumption and their impacts are interrelated, an interesting question is that, *given a circuit netlist, whether the potential energy efficiency gains provided by timing speculation (over conventional circuit-level power optimization techniques) is significant enough to warrant the effort to make it timing-speculative?* To the best of our knowledge, this problem, although important and relevant, has not been explicitly investigated in the literature.

As it is not possible to derive an optimal timing-speculative circuit for evaluation purpose, in this work, we try to answer the above question by studying the premises and prospects of timing speculation instead. To be specific, we develop novel algorithms to obtain the minimum and maximum potential benefits achievable with TS techniques which facilitate designers to explore preferred power optimization techniques. Experimental results on various benchmark circuits demonstrate the efficacy of the proposed methodology.

The remainder of this chapter is organized as follows. In Section 2.2, we present the preliminaries and motivation of this work. The general optimization problem for timing-speculative circuits is formulated in Section 2.3. The premise problem and prospect problem on the potential benefit of TS are then detailed in Section 2.4. Next, Section 2.5 presents our experimental results on various benchmark circuits. Finally, Section 2.6 concludes this chapter.

2.2 Preliminaries and Motivation

2.2.1 Timing Speculation

Circuit-level timing speculation, being able to detect timing errors online, react to the error quickly and recover from it by rolling back to a known-good pre-error state, is one of the most promising solutions to deal with process, voltage, and temperature (PVT) variations and aging effects.

Various online timing error detectors were proposed in the literature. Without loss of generality, let us consider the representative *Razor* flip-flop [9] to demonstrate how timing error detectors work. A Razor flip-flop, is comprised of a main flip-flop, a shadow latch and some control logic. The main flip-flop latches the output signal at the clock edge with possible timing error, while the shadow latch guarantees to receive the correct value, by latching the signal a fraction of a cycle later. Consequently, when the shadow latch and the main FF values do not agree, the timing error is detected. To make use of TS technique, it is necessary to replace all critical flip-flops that are driven by speed-paths of the circuit with Razor flip-flops (or other timing speculators).

For microprocessor pipelines, timing error recovery can be achieved with microarchitectural support [34]. That is, when a timing error is detected, the processor pipeline is flushed and the correct result from the shadow latch is returned back into the pipeline. Then, by replaying instructions (possibly at lower frequency), the processor is able to recover from timing errors [27].

2.2.2 Related Work

Various optimization techniques for timing-speculative circuits have been presented in the literature for TER reduction under a specified voltage-frequency combination. EVAL [13] uses a so-called high-dimensional dynamic adaptation

technique that trades error rate for processor frequency by tilting, shifting, or reshaping the path distributions of various functional units. Blueshift [14] identifies and optimizes the most frequently exercised critical paths by on-demand selective biasing and path constraint tuning. DynaTune [19] optimizes the most frequently-sensitized critical paths of the circuit by assigning low threshold voltage to those critical gates that are strongly related to the occurrence of timing errors. To mitigate the impact of timing wall, *Kahng et al.* [31, 32] proposed a slack redistribution strategy to achieve a gradual delay distribution that is able to better serve TS techniques.

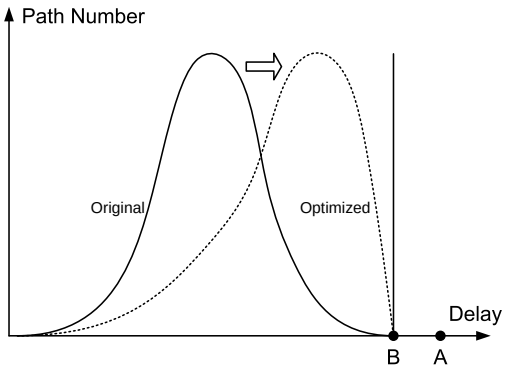
The above solutions conduct optimization for timing speculation with a fixed circuit netlist. Recently, several logic synthesis techniques were proposed [11, 20, 28]. By intentionally manipulating the circuit structure for timing speculation, the circuit energy efficiency can be further improved.

2.2.3 Motivation

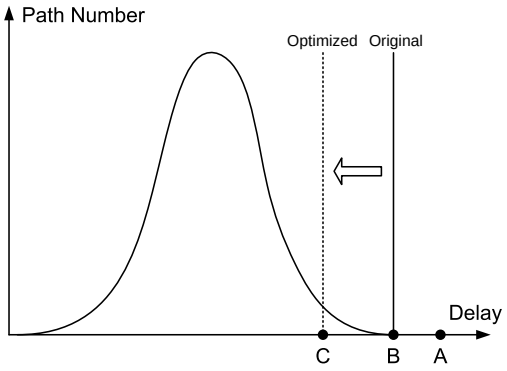
In Fig. 2.1, we plot the circuit path delay distribution under conventional power optimization and timing speculation, wherein the vertical lines represent the operational clock periods of the circuit.

The case of conventional power optimization is depicted in Fig. 2.1(a). Point B is the point where maximum path delay equals to operational clock period and hence has zero timing slack[1]. Considering variation effects, the actual operational clock period is usually set to be Point A, with the timing slack between Point A and Point B reserved as design guardband. By trading off delay and power consumption on non-critical paths, the path delay distribution is reshaped from the solid curve to the dotted one after power optimization, form a timing wall, as shown in Fig. 2.1(a). As for the TS case, because infrequent timing errors can be

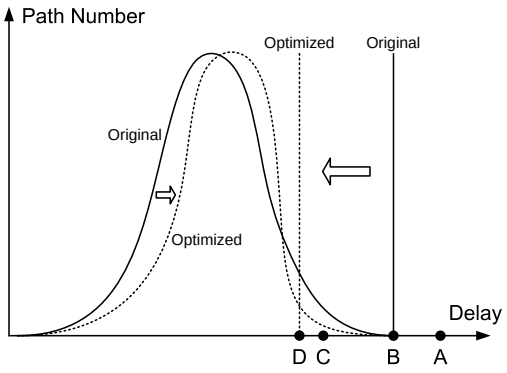
[1]We simply assume the setup time is zero here for simplicity.



(a) Conventional optimization.



(b) Timing speculation.



(c) Optimized timing speculation.

Figure 2.1: Path delay distributions for conventional power optimization and timing speculation.

detected and corrected in timing-speculative circuits, we can reduce operational clock period from Point B to Point C (see Fig. 2.1(b)). Clearly, such performance improvement can be easily translated into energy efficiency gain by simply scaling down supply voltage. By reshaping path delay distribution as shown in Fig. 2.1(c), we can further improve the energy efficiency of timing-speculative circuits.

Considering the non-trivial design effort to make a circuit timing-speculative, it is essential to evaluate the potential benefits at early design stage. In [12], Kruijf *et al.* presented a system-level analytical framework for timing speculation, wherein technology nodes, CMOS design styles (i.e., high-performance or low-power) and different fault recovery schemes are considered. While informative and inspiring, this general analytical framework lacks one important feature. That is, the potential energy efficiency gain provided by timing speculation may vary a lot for different circuits because it depends heavily on the given circuit structure. This important issue, however, is not taken into consideration in prior analytical works.

The above motivates us to study the minimum and maximum potential benefits achievable with TS techniques for a given circuit, which facilitates designers to make decisions on whether to adopt timing speculation or not, without paying much design effort to actually implement it.

2.3 General Problem Formulation

Before introducing how to obtain the minimum and maximum potential benefits of TS, let us first formulate the optimization problem for a given timing-speculative circuit.

Problem: *Given the netlist of a timing-speculative circuit, equipped with timing speculators such as Razor flip-flops [9], and a performance constraint f_c, determine the size w_i and the threshold voltage v_i of each gate G_i, and the supply*

voltage v_{dd} and operational clock frequency f of the entire chip, so that the energy consumption E_{total} is minimized under performance constraint.

As re-computation is needed when timing errors occur, the energy consumption of timing-speculative circuits is:

$$E_{total}(\vec{w},\vec{v},v_{dd},f) = P \cdot \frac{1}{f} \cdot (1 + error \cdot penalty) \quad , \tag{2.1}$$

where $\vec{w}$ is the vector whose element represents the size of each gate, $\vec{v}$ is the vector whose element represents the threshold voltage of each gate, P is the power function, *error* is the function of timing error probability, and *penalty* is the cost including both the cycles of wasted execution that must be discarded and the time spent on checkpointing and re-execution. Meanwhile, we need to ensure the performance constraint:

$$f_{eq} = \frac{f}{(1 + error \cdot penalty)} \geq f_c \quad , \tag{2.2}$$

where f_{eq} is the equivalent clock frequency considering performance penalty of timing error correction.

Note that, without loss of generality, we only consider the optimizations on gate size, threshold voltage and supply voltage in this work. The optimization objective function (see Eq. 2.1) requires the models of power consumption and timing error probability that have been discussed in Appendix 2.7.1 and Appendix 2.7.2. The proposed methodology, however, is applicable for any optimizations that have a closed-form objective function of energy consumption.

2.4 Premises and Prospects of Timing Speculation

As it is impossible to obtain an optimal solution for the problem defined in Section 2.3, we, instead, propose to investigate the minimum and maximum potential energy benefits of TS techniques. The minimum potential benefit establishes the

premise for timing speculation while the maximum potential benefit presents the prospects of timing speculation.

2.4.1 The Premises

The premise problem to calculate the minimum potential benefit can be tackled by conducting an effective optimization method onto the general formulation in Section 2.3. In this work, we develop a novel technique consisting of two stages to solve it. The first stage is based on gradient-descent method (GDM) [35] considering continuous solution space of parameter setting, while the second stage optimizes the discrete parameters with the help of steepest descent method (SDM) [36].

2.4.1.1 Exploring Continuous Space by GDM

GDM is a first-order optimization algorithm that utilizes the gradient vector $\nabla f(\vec{x})$ to determine the search direction for each iteration. The simplest and most famous GDM algorithm takes steps proportional to the negative/positive of the gradient (or, the approximate gradient) of the function at current iteration to minimize/maximize $f(\vec{x})$.

With the timing error probability discussed in Appendix 2.7.2, we can now compute the gradients of objective function (see Eq. 2.1) with respect to parameters. For the sake of clear presentation, we use $\vec{x}$ to represent all the parameters ($\vec{w}$, $\vec{v}$, v_{dd} and f) without distinguishing them from each other. As we would like to minimize energy consumption, we use the negative of the computed gradients to update the parameters at each iteration as follows:

$$x_\ell^{new} = x_\ell + \eta \cdot (-\frac{\partial E_{total}(\vec{x})}{\partial x_\ell}),\ \forall x_\ell \in \vec{x} = (x_1, \cdots, x_n)\ , \tag{2.3}$$

where η is the learning rate.

#	$f(\vec{x})$, the objective function for optimization
#	ε_g and $\varepsilon_{\vec{x}}$, the convergence tolerances
#	$\vec{x}^*$, the output of parameter setting
1.	Initialize $\vec{x}$
2.	**REPEAT** for each iteration
3.	**IF** $\| \nabla f(\vec{x}) \| < \varepsilon_g$
4.	Set $\vec{x}^* = \vec{x}$
5.	Break
6.	**ELSE**
7.	Compute $\vec{g} = -\nabla f(\vec{x})$
8.	Update parameters $\vec{x}^{new} = \vec{x} + \eta \cdot \vec{g}$
9.	**IF** $\| \vec{x}^{new} - \vec{x} \| < \varepsilon_{\vec{x}}$
10.	Set $\vec{x}^* = \vec{x}^{new}$
11.	Break
12.	**ELSE**
13.	Go for next iteration

Figure 2.2: The proposed GDM-based optimization in continuous space.

The algorithm flow of the proposed GDM-based method is described in Fig. 2.2. Firstly, we initialize the parameters $\vec{x}$ randomly. Then, we repeat the procedure (see Line 3 $\sim$ 13) to compute gradients and update parameters until the convergence criterion is satisfied. To be specific, we have two convergence criterion: (i) the gradient tolerance ε_g to determine whether the algorithm has arrived at a critical point, and (ii) the step tolerance $\varepsilon_{\vec{x}}$ to determine whether significant progress is achieved. Once either of them is satisfied, the optimization process is terminated; otherwise, it continues to compute gradients (See Line 7) and update parameters (see Line 8).

2.4.1.2 Exploring Discrete Space by SDM

The above GDM-based technique can effectively optimize timing-speculative circuits in continuous space. However, it is not practical to assume arbitrary continuous values are allowed for parameters such as gate size $\vec{w}$ and threshold voltage $\vec{v}$, since a look-up table gate model with only a few number of discrete parameter values is the standard in most industrial designs. The GDM-based technique is used to achieve the setting of supply voltage v_{dd} and clock frequency f. Its output on $\vec{w}$ and $\vec{v}$ would be further optimized by a novel SDM-based technique discussed as follows.

Given a set of discrete values for gate size and threshold voltage, we first discretize the output of GDM-based technique to the closest value within the set, providing an initial solution in discrete space. Then, we formulate a discrete search problem and resort to heuristic search algorithm based on SDM.

The solution representation is naturally described using the vector of parameter setting $\vec{x} = (x_1, \cdots, x_n)$, wherein each element is either gate size or threshold voltage of a certain gate. As for the move, it is simply defined as the change of x_i $(1 \leq i \leq m)$. This definition of move guarantees the completeness of traversing the entire discrete solution space.

To evaluate solutions during search process, we use the objective function defined by Eq. 2.1. It is worth noting that the change of x_i affects only a small part of the calculation of objective function. For example, when conducting gate sizing to a gate, only its preceding gates and itself are influenced. Therefore, we only have to update the calculation under influence, dramatically saving computational effort.

With the above definitions, this problem can naturally be solved by search algorithms (e.g., random search, simulated annealing) [36]. In this work, we resort to SDM, a discrete analogue of GDM, because it is typically able to converge in a

#	$\vec{x}_0$, initial solution generated by GDM-based method
#	$\vec{x}_c$ and $\vec{x}_n$, current solution and next solution
#	$metric(\cdot)$, the calculation of objective function
1.	Initialize setting $\vec{x}_c = \vec{x}_0$
2.	**REPEAT** for each iteration until convergence
3.	Initialize $\vec{x}_n = \vec{x}_c$
4.	**FOR** each neighbor solution $\vec{x}_i$ of $\vec{x}_c$
5.	**IF** $metric(\vec{x}_i) < metric(\vec{x}_n)$
6.	$\vec{x}_n = \vec{x}_i$
7.	**IF** $\vec{x}_n == \vec{x}_c$ **// No better solution found**
8.	Break
9.	**ELSE**
10.	$\vec{x}_c = \vec{x}_n$
11.	Go for next iteration

Figure 2.3: The proposed SDM-based search algorithm in discrete space.

few steps.

Basically, an SDM search starts at an initial state and takes search steps in solution space, reducing a given objective function with the maximum rate of descent. Instead of computing a gradient in GDM, the best move of SDM is determined using a local minimization. The algorithm flow is demonstrated in Fig. 2.3. After initialization, the search optimization repeats iteratively (see Line 3 $\sim$ 11). For each iteration, we find out the move leading to the neighbor solution with smallest objective value (see Line 4 $\sim$ 6). The search process continues until convergence or a termination condition is reached.

2.4.2 The Prospects

As we are to estimate the maximum potential benefit to find the prospect of timing speculation, we can simplify the original problem as long as the solution of the simplified problem will still be an upper bound of the original one.

2.4.2.1 Estimation Algorithm

The proposed algorithm to estimate the minimum energy consumption is described in Fig. 2.4. We start the estimation from the setting with the minimum power/energy, i.e., the setting with all the gate sizes set to the minimum allowed value and all the threshold voltage set to the maximum allowed value (see Line $1 \sim 3$). In such case, it is very likely to have rather large error probability that exceeds the error constraint specified by performance constraint in Eq. 2.2 , and hence owe an "error debt" that is defined as the difference between the current error probability and the specified error constraint (see Line $4 \sim 6$). To ensure the error constraint is satisfied and the eventually-estimated TS benefit is an upper bound, we have to guarantee that such "error debt" is paid off in the most energy-efficient manner. That is to say, we want to reduce error probability until it reaches error constraint at the minimum expense of energy increase.

To achieve the above, we define a metric *sensitivity* to describe the ratio of energy increase over error reduction due to the change of a certain parameter:

$$S = -\frac{\Delta E}{\Delta error} = \frac{E - E_0}{error_0 - error} \quad , \tag{2.4}$$

where E_0 and $error_0$ are the energy consumption and error probability with initial setting, and E and $error$ are the energy consumption and error probability after the parameter change. If we can ideally obtain the minimum sensitivity of each gate (see Line $7 \sim 10$) and then take action to the gates one by one in the sensitivity-ascending order until error debt is paid off (see Line $11 \sim 18$), it is definitely the

#	W_{min}, the minimum allowed gate width
#	V_{max}, the maximum allowed threshold voltage
#	S, sensitivity
1.	Initialize $w_i = W_{min}$ and $v_i = V_{max}$, $\forall i$
2.	Compute power consumption P
3.	Initialize estimated energy consumption $E = P/f$
4.	Compute error constraint $ec = (f/f_c - 1)/penalty$
5.	Compute error probability e with current setting
6.	Compute error debt $ed = e - ec$
7.	**FOR** each gate G_i
8.	Compute $G_i.S = \min(-\Delta E/\Delta error)$
9.	Record $G_i.E = \Delta E$ when achieving minimum S
10.	Record $G_i.error = -\Delta error$ when achieving minimum S
11.	**REPEAT** for each iteration until $ed \leq 0$
12.	**FOR** each gate G_i with $G_i.visited == 0$
13.	**IF** $G_i.S < S$
14.	$S = G_i.S$
15.	$m = i$
16.	$ed- = G_m.error$
17.	$E+ = G_m.E$
18.	Set $G_m.visited = 1$

Figure 2.4: The proposed algorithm to estimate the minimum energy consumption.

most energy-efficient way and the estimated value would be guaranteed to be a lower bound of TS energy consumption. Note that, the methodology of obtaining the minimum sensitivity of each gate would be detailed in the following.

2.4.2.2 Problem Simplification

One of the difficulties to efficiently obtain the minimum sensitivity of each gate is due to the complex impact of the parameter change. For example, the size of

a gate would influence not only its own load capacitance (and hence energy consumption) but also that of its preceding gates. More importantly, assuming there are k parameter choices for each gate, the size of solution space is k^n, exponentially increased with respect to gate number n. Obviously, such solution space is too large to efficiently explore. To tackle this problem, we propose to eliminate the influence between gates and simultaneously reduce solution space by simplifying the problem.

We simplify the calculation of energy consumption, on the basis of an observation that the gates impact each other's energy consumption through affecting the load capacitance. For instance, if we downsize a gate, all the load capacitances and hence the energy consumption of its preceding gates would be reduced. Therefore, when calculating the load capacitance of a gate, if we always use the minimum sizes of its succeeding gates no matter what sizes they actually have, the load capacitance would be affected by only the gate itself but not its succeeding gates any more. With such a model that intentionally underestimates load capacitances, the estimated value of energy consumption can be less than the actual one, ensuring that it is still a lower bound of TS energy consumption. Note that, using similar methodology we can also simplify the calculation of error probability.

With the above simplifications, the energy consumption and error probability of a gate are both influenced by the parameters of the gate itself. Consequently, the calculation of the minimum sensitivity becomes very easy, as we only need to consider the impact locally. Especially in the scenario of discrete space, as the size of solution space for each gate is only equal to its choice number k (typically, k can be about 100), we can even enumerate all the combinations of parameters to achieve its minimum sensitivity. It is worth noting that the size of solution space for the entire chip is now reduced to $k \times n$, only linearly increased with respect to gate number n.

2.5 Experimental Results

2.5.1 Experimental Setup

We conduct experiments on several large ISCAS'89 and ITC'99 benchmarks. Particularly, these ITC'99 benchmarks are the subsets of processors. We synthesize these circuits and obtain timing information using Synopsys EDA tools. To take process variation effects into consideration, we perform Monte Carlo simulation to inject gate-level delay variation following Gaussian distribution with standard deviation equal to 8%. Random inputs are used in our experiments and each simulation is performed with 100,000 cycles. All the experiments are conducted on a *2.8GHz* PC with *4GB* RAM.

For comparison, we provide two baseline solutions: (i) conventional technique without TS [37] , denoted as $CT_{baseline}$; and (ii) TS technique without conventional optimization [9], denoted as $TS_{baseline}$. The proposed optimization technique in Section 2.4.1 is denoted as TS_{opt} and the estimation algorithm in Section 2.4.2 is denoted as TS_{bound}. To equip some of the flip-flops as timing speculators, we can simply resort to a simple scheme that equips all the flip-flops whose timing slacks are less than 20% of operational clock period. The hardware cost to equip a timing speculator is assumed to be 10 gates. The *penalty* of error recovery in Eq. 2.1 is assumed to be 10 clock cycles according to [12].

2.5.2 Results and Discussion

We report the results on hardware cost and algorithm runtimes in Table 2.1. As can be seen, the average hardware costs for $TS_{baseline}$ and TS_{opt} to equip timing speculators are 5.32% and 5.66%, respectively. The hardware cost for TS_{opt} is a little higher than $TS_{baseline}$, but still within an acceptable range. The runtimes of TS_{opt} and TS_{bound} are all less than one hundred seconds, both listed in Table 2.1.

Bench.	TG#	TFF#	$TS_{baseline}$		TS_{opt}		RT_{opt} (s)	RT_{bound} (s)
			TTS#	Cost (%)	TTS#	Cost (%)		
s1494	680	6	2	2.94	2	2.94	10.91	0.21
s5378	3042	179	20	6.57	21	6.90	22.06	1.32
s9234	5866	228	22	3.75	25	4.26	23.65	1.99
s13207	8803	638	10	1.14	15	1.70	27.06	5.58
s15850	10470	597	89	8.50	92	8.79	33.25	9.67
s35932	18148	1728	144	7.93	151	8.32	39.25	15.42
s38584	21021	1426	168	7.99	179	8.52	44.05	21.24
s38417	24341	1564	91	3.74	115	4.72	48.22	18.38
b20	20226	490	121	5.64	116	5.42	45.13	13.65
b21	20571	490	119	5.47	125	5.73	52.56	15.16
b22	29951	735	153	4.86	156	4.95	57.88	19.41
AVERAGE				5.32		5.66		

TG#: total gate count; TFF#: total flip-flop count; TTS#: total timing speculator count;

Cost: hardware cost ratio for equipping timing speculators;

RT_{opt}: runtime of TS_{opt}; RT_{bound}: runtime of TS_{bound}.

Table 2.1: Hardware cost and algorithm runtimes.

To demonstrate the effectiveness of TS_{opt}, in Table 2.2 we present the energy consumptions[2] that have considered both the penalties of error recovery and the costs of equipping timing speculators. When compared to $TS_{baseline}$, TS_{opt} further reduces energy consumption by 0.306 on average. The improvement room between TS_{opt} and TS_{bound} is only 0.091, showing that the proposed optimization technique TS_{opt} is rather close to the "optimal".

Next, we take *s*38417, the largest benchmark circuit in ISCAS'89, as an example to demonstrate more details. In Fig. 2.5, we show the trends of both the energy metric estimated according to Eq. 2.1 and the actual energy consumption achieved by timing simulation, with respect to GDM/SDM iteration numbers. Ob-

[2]For clear presentation, all the energy consumptions are normalized to that of the case without any conventional and TS optimizations.

Bench.	$TS_{baseline}$	TS_{opt}		TS_{bound}	
		$En.$	Δ_1	$En.$	Δ_2
s1494	0.708	0.468	-0.240	0.420	-0.048
s5378	0.859	0.578	-0.281	0.506	-0.072
s9234	0.822	0.495	-0.327	0.375	-0.120
s13207	0.726	0.480	-0.246	0.346	-0.134
s15850	0.871	0.533	-0.338	0.445	-0.088
s35932	0.866	0.596	-0.270	0.458	-0.138
s38584	0.862	0.505	-0.357	0.405	-0.100
s38417	0.709	0.455	-0.254	0.342	-0.113
b20	0.810	0.525	-0.285	0.420	-0.106
b21	0.854	0.473	-0.381	0.400	-0.073
b22	0.825	0.436	-0.389	0.428	-0.009
AVERAGE	0.810	0.504	-0.306	0.413	-0.091

$En.$: energy consumption;

Δ_1: energy difference of TS_{opt} over $TS_{baseline}$;

Δ_2: energy difference of TS_{bound} over TS_{opt}.

Table 2.2: Energy consumptions of TS techniques with the cost of timing speculators included.

viously, the GDM and SDM methods optimizing towards the estimated metric can simultaneously reduce the actual energy consumption, proving their effectiveness. In Fig. 2.6 (a) and Fig. 2.6 (b), we present the delay distributions before and after the optimization by TS_{opt}, while the change of delay distribution between them is shown in Fig. 2.6 (c). As can be seen, after optimization the number of paths with delay in the range of $[0.5, 0.8]$ (see the solid ellipse) is substantially increased, while the number of paths with larger or smaller delay are decreased (see the dotted ellipses). This is because, TS_{opt} trends to shorten long paths for further over-clocking clock frequency (and then scaling down supply voltage for energy reduction) and prolong short paths for directly reducing energy. Such an observation on the change of delay distribution is consistent with the motivation

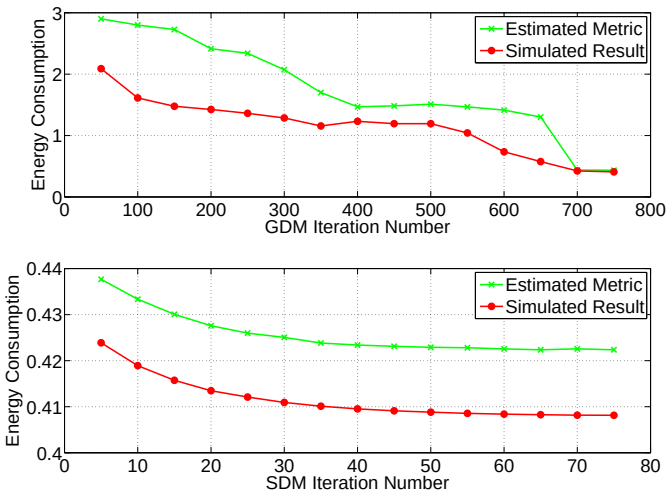


Figure 2.5: Energy consumption with respect to optimization iterations.

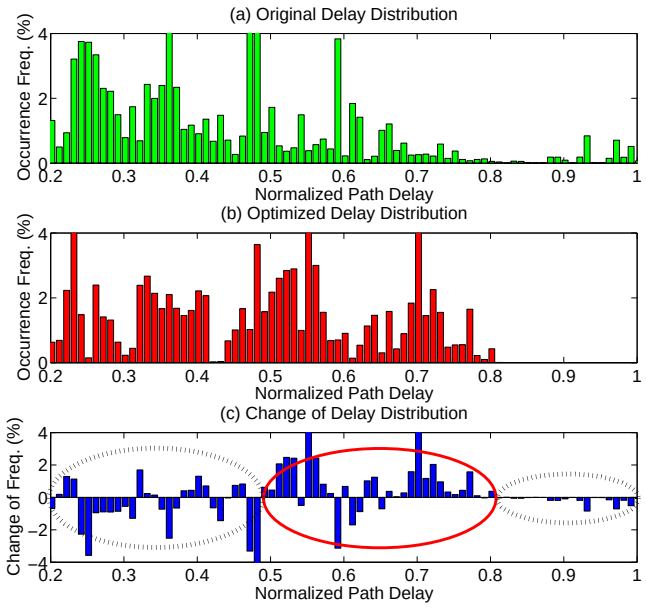


Figure 2.6: The change of delay distribution after optimization.

discussed in Section 2.2.3.

Bench.	$CT_{baseline}$	TS_{opt}		TS_{bound}	
		En.	*MB*	*En.*	*MPB*
s1494	0.522	0.468	0.053	0.420	0.102
s5378	0.544	0.578	-0.034	0.506	**0.038**
s9234	0.505	0.495	0.010	0.375	0.130
s13207	0.706	0.480	**0.227**	0.346	0.360
s15850	0.506	0.533	-0.027	0.445	**0.061**
s35932	0.525	0.596	-0.071	0.458	**0.066**
s38584	0.616	0.505	**0.111**	0.405	0.211
s38417	0.585	0.455	**0.130**	0.342	0.243
b20	0.573	0.525	0.048	0.420	0.153
b21	0.557	0.473	0.084	0.400	0.157
b22	0.544	0.436	**0.108**	0.428	0.117
AVERAGE	0.562	0.504	0.058	0.413	0.149

MB: The minimum benefit of TS;

MPB: The maximum potential benefit of TS.

Table 2.3: TS benefits in terms of energy consumption.

Finally, we take the energy cost, simply considered as same as hardware cost in Table 2.1, of equiping timing speculators into account to calculate the minimum benefit and maximum potential benefit. It can be found in Table 2.3 that, after considering such costs, the minimum benefit achieved by TS_{opt} is about 0.058 on average, while the maximum potential benefit estimated by TS_{bound} is about 0.149 on average. Assuming an amount of benefit η ($\eta = 0.1$) is considered to deserve design efforts by IC designers, we can conclude that: (i) TS is preferred for the optimization of the benchmarks $s13207$, $s38584$, $s38417$, $b22$, since they are proved by TS_{opt} to have a benefit more than 0.1; and (ii) conventional technique is preferred for the benchmarks $s5378$, $s15850$ and $s35932$, because their maximum potential benefits, indicated by TS_{bound}, are all less than 0.1. As for the other benchmarks, the proposed methodology, unfortunately, is not able to conclude the

applicability of TS on it, if the criteria $\eta = 0.1$ is given.

2.6 Conclusion

Timing speculation is a promising solution to combat the ever-increasing variation effects, but how much benefits can be provided is strongly related to the circuit structure itself. Considering the non-trivial design effort to make a circuit timing-speculative, for a given circuit, it is essential to evaluate the potential benefits at early design stage. In this work, we propose novel algorithms to study the premise and prospects of timing speculation to tackle this problem. Experimental results based on various benchmarks demonstrate the effectiveness of the proposed methodology.

2.7 Appendix

2.7.1 Power and Delay Models

The total power consumption of an electronic system is comprised of two components: dynamic power consumption and static power consumption [38].

Dynamic power [39] is due to the switching activities, manifesting as charging and discharging of the load capacitance. A widely-used model of dynamic power can be written as:

$$P_{dyn} = \alpha_s \cdot C \cdot v_{dd}^2 \cdot f \quad , \tag{2.5}$$

where α_s is switching activity, C is load capacitance, v_{dd} is supply voltage and f is clock frequency. To calculate dynamic power, one of the critical issues is to model load capacitance that is usually considered to consist of four components: (i) gate-drain capacitance of driver transistor; (ii) diffusion capacitance of driver transistor; (iii) gate capacitance of load transistors; and (iv) wiring capacitance depending on

the length and width of connecting wires. The details on these capacitances can be found in [22, 40].

Static power [39, 41] is primarily caused by subthreshold and gate-oxide leakage currents which are present even when no logic operations are performed. It can be modeled as:

$$P_{st} = v_{dd} \cdot I_{st} \quad , \tag{2.6}$$

where I_{st} is the cumulative leakage current of all the leakage mechanisms, especially the subthreshold and gate-oxide leakages that are described by:

$$\begin{aligned} I_{sub} &= K_1 \cdot W \cdot e^{-v/(nv_\theta)}(1 - e^{-v_{dd}/v_\theta}) \quad , \\ I_{ox} &= K_2 \cdot W \cdot (\frac{v_{dd}}{T_{ox}})^2 \cdot e^{-\gamma T_{ox}/v_{dd}} \quad , \end{aligned} \tag{2.7}$$

where I_{sub} is subthreshold leakage, I_{ox} is gate-oxide leakage, K_1, K_2, n and γ are experimentally derived parameters, W is gate width, v is threshold voltage, v_θ is thermal voltage, and T_{ox} is oxide thickness.

The gate delay can be modeled as follows:

$$Delay = \frac{K \cdot v_{dd} \cdot C}{W \cdot (v_{dd} - v)^\alpha} \quad , \tag{2.8}$$

where K and α are fitting parameters (see [38]), C is load capacitance, v is threshold voltage, and W is gate width.

2.7.2 Timing Error Probability

To solve the optimization problem defined in Section 2.3, one of the most critical problems is to calculate the timing error probability *error*, a function with respect to a number of variables (e.g., $\vec{w}$, $\vec{v}$, v_{dd}, and f) that can be expressed as:

$$error = 1 - \prod_{j=1}^{|V|} (1 - error_j) \quad , \tag{2.9}$$

where $error_j$ is the error probability of a flip-flop FF_j during one single clock cycle.

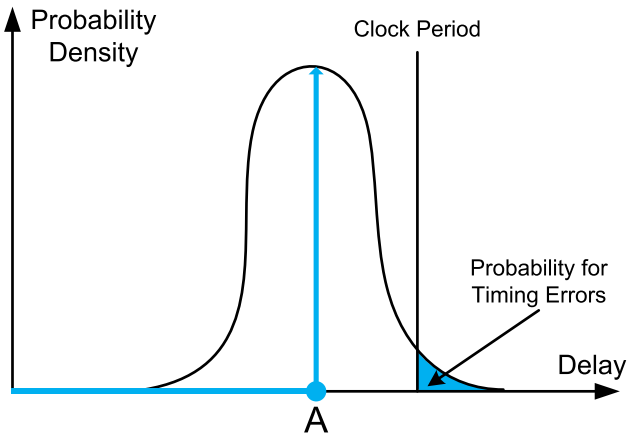


Figure 2.7: Timing error probability of a path considering variation effects.

As for the calculation of $error_j$, the error probability of each flip-flop, we need to know the path sensitization probability, which can be acquired by performing functional simulation of the circuit with representative workloads or calculated according to some models [28, 31].

As demonstrated in Fig. 2.7, when considering variation effects, the path delay (indicated by Point A) becomes a random variable [33], therefore the cumulative probability for path delay to exceed operational clock period is the error probability of this path. We can have

$$\begin{aligned} error_j &= \sum_{i \in B_j} \sum_{k} N_{ijk} \cdot Pr\{\widetilde{S}_{ijk} \leq 0\} \quad , \\ \widetilde{S}_{ijk} &\triangleq T - T_{setup} - \widetilde{D}_{ijk} \quad , \end{aligned} \tag{2.10}$$

where N_{ijk} is the probability for the k^{th} path from FF_i to FF_j to be sensitized, $\widetilde{D}_{ijk}$ is the corresponding path delay variable of the k^{th} path, $\widetilde{S}_{ijk}$ is the variable of timing slack as defined in Eq. 2.10, T is the operational clock period (i.e., $T = 1/f$), T_{setup} is the setup time of flip-flop, B_j is the set of flip-flops that serve as the beginners of the paths ending at FF_j, and $Pr\{\widetilde{S}_{ijk} \leq 0\}$ is the probability to have timing errors

in the k^{th} path. Note that, as the problem of hold time constraints can be solved by delay padding [42], we only consider the setup time constraints in this work.

To apply GDM, we need to ensure the path error probability $Pr\{\widetilde{S}_{ijk} \leq 0\}$ (see Eq. 2.10) to be differentiable. We express the impact of parameter change (e.g., downsizing gates, lowering supply voltage) to timing slack variable $\widetilde{S}_{ijk}$ as:

$$\widetilde{S}_{ijk} = \widetilde{S}^r_{ijk} + S^d_{ijk} \quad , \tag{2.11}$$

where $\widetilde{S}^r_{ijk}$ is the timing slack varaible under reference condition and S^d_{ijk} is the change of timing slack due to parameter change. Therefore, we have

$$Pr\{\widetilde{S}_{ijk} \leq 0\} = Pr\{\widetilde{S}^r_{ijk} \leq -S^d_{ijk}\} \quad , \tag{2.12}$$

implying that the increasing of timing slack on a path would reduce its error probability. By defining the Cumulative Distribution Function (CDF) of $\widetilde{S}^r_{ijk}$ as:

$$F_{ijk}(t) = Pr\{\widetilde{S}^r_{ijk} \leq t\} \quad , \tag{2.13}$$

where $t = -S^d_{ijk}$, the path delay probability becomes differentiable. Note that, similar to prior works (e.g., [33]), we assume path delay variable $\widetilde{D}_{ijk}$ and timing slack variable $\widetilde{S}^r_{ijk}$ to follow Gaussian Distribution.

□ End of chapter.

Chapter 3

Voltage Island Generation

3.1 Introduction

Motivated by the fact that individual blocks of a circuit can have timing/power characteristics unique from the rest of the design, the concept of multi-supply voltage (MSV) design was introduced to trade off power consumption and performance, and has attracted lots of interests from both academia and industry [43–51]. In MSV designs, circuits are partitioned into multiple ”voltage islands” and each island operates at a specified supply voltage that satisfies its performance requirement.

In conventional MSV designs, to meet the timing requirement of each voltage island, the corresponding supply voltage has to be high enough to drive the most timing-critical cell, even though the rest of cells may have much more relaxed timing requirements. Moreover, with the ever-increasing variation effects (e.g., process variation effects due to manufacturing imperfection and dynamic variation effects caused by voltage and temperature fluctuations) in nanometer technology, a large design guard band needs to be reserved to tolerate timing uncertainty. Due to the above, we have to be rather conservative when assigning voltages for each

island, reducing the possible power savings that can be achieved with MSV designs.

Recently, *timing speculation* (TS) techniques that allow the occurrence of infrequent timing errors and employ error detection and correction techniques to recover from them have emerged as a promising solution to achieve *error-resilient computing* [4, 7, 9, 12, 13]. Such "better than worst-case" designs allow the trade-off between reliability and performance/power, thereby being much more energy-efficient when compared with conventional "worst-case-oriented" designs. Intel [52] has recently demonstrated in their test chip that a timing-speculative microprocessor is able to achieve more than 30% throughput gain when compared to a conventional microprocessor design.

Introducing timing speculation capability into circuits can naturally extend the flexibility of MSV designs to a new horizon, since we do not need to guarantee "always correct" operations any longer and the voltage assignment of islands can avoid being dominated by certain sparse timing-critical cells. How to conduct MSV design for timing-speculative circuits is hence an interesting problem, which, to the best of our knowledge, has not been explored in the literature yet.

Motivated by the above, in this work, we formulate the MSV problem for timing-speculative circuits and develop a novel algorithm based on dynamic programming to solve it. The proposed technique naturally supports "recovery island" design methodology described in [53], wherein each island can recover independent of the rest of the circuit. Experimental results on various benchmark circuits demonstrate that the proposed technique is able to achieve significant power reduction when compared to exiting MSV design techniques.

The remainder of this chapter is organized as follows. In Section 3.2, we present the preliminaries and motivation of this work. The problem formulation and the corresponding algorithms are then detailed in Section 3.3 and Section 3.4,

respectively. Next, Section 3.5 presents our experimental results based on various benchmark circuits. Finally, Section 3.6 concludes this chapter.

3.2 Preliminaries and Related Work

3.2.1 Power and Delay Models

The total power consumption of an electronic system consists of two components: dynamic power and static power [38, 43]. Dynamic power is due to the switching activities, manifesting as charging and discharging of the load capacitance, while static power is primarily caused by subthreshold leakage current which is present even when no logic operations are performed. They can be described as

$$\begin{aligned} P_d &= \alpha \cdot C \cdot V_{dd}^2 \cdot f \quad , \\ P_s &= I_s \cdot V_{dd} \quad , \end{aligned} \tag{3.1}$$

where P_d is dynamic power, α is switching activity, C is load capacitance, V_{dd} is supply voltage, f is clock frequency, P_s is static power and I_s is the cumulative leakage current of all kinds of leakage mechanisms (refer to [54] for details). From Eq. 3.1, it can be found that both dynamic and static powers are strongly related to supply voltage V_{dd}. This observation gives the basic motivation to generate voltage islands to reduce power consumption.

However, with supply voltage scaling down, the gate delay of circuits will be increased as follows,

$$D = \frac{K_i \cdot V_{dd}}{(V_{dd} - V_t)^{\alpha}} \quad , \tag{3.2}$$

where D is gate delay, V_t is threshold voltage, K_i and α are fitting parameters as defined in [38]. That means, the power consumption of a circuit is reduced with voltage scaling down at the expense of timing performance.

3.2.2 MSV Design

A large amount of work has been devoted to MSV designs in the literature and they are applied in various design stages, e.g., floorplanning stage [43, 44], post-floorplanning stage [45], placement stage [46, 47], and post-placement stage [48–51].

As pointed out in [48], conducting region-based MSV design before placement based on their logic boundaries, while "natural", is usually far from optimal. Instead, by using placement proximity (instead of logical) information for MSV design, the acquired solution can achieve much better power savings. Motivated by this observation, the authors proposed to utilize dynamic programming (DP) to generate voltage islands considering placement proximity. While DP provides optimal results, the computational complexity and memory requirement to conduct it at fine-grained granularity is not acceptable for a reasonable-sized circuit. Consequently, a heuristic algorithm is used to partition the circuit into $p \times q$ coarse grids first and DP is conducted at the coarse-grained level. While being more efficient, the effectiveness of this technique is inevitably constrained by the heuristic partitioning algorithm. In [50], the authors investigated how to generate an initial voltage assignment considering the physical proximity of high voltage cells as the input of [48]. After that, to tackle the problem that the freedom of voltage assignment is limited by the amount of available slacks on timing-critical paths, [49] performed incremental placement to improve timing on these paths. All the above works try to generate voltage islands with the guarantee that the timing requirements of all cells are satisfied.

3.2.3 Timing Speculation

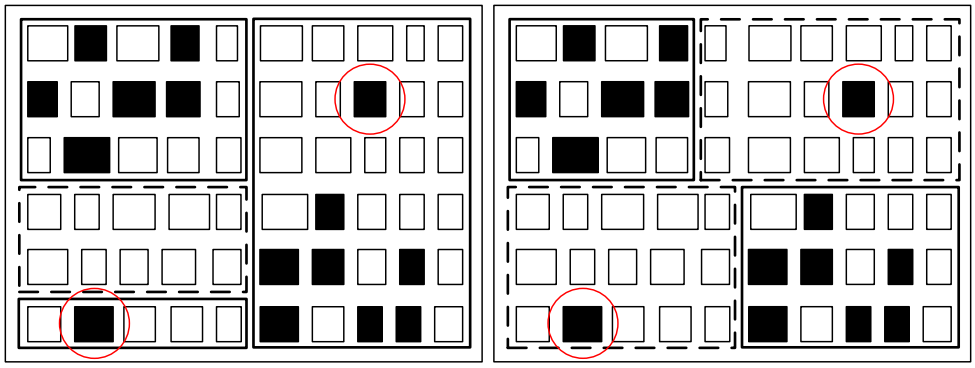
Circuit-level timing speculation technique, being able to detect timing errors at online stage, react to the error quickly and recover from it by rolling back to a

known-good pre-error state, has become one of the most promising solutions for variation-aware designs. Without loss of generality, let us discuss one of the most representative timing speculation techniques, Razor [9], to illustrate how resilient computation can be achieved with timing speculation. To detect timing errors on a critical path, the receiving end of the critical path, referred to as suspicious flip-flop, is replaced with Razor flip-flop (Razor-FF), which includes a main flip-flop (FF), an additional shadow latch and some control logic. The main flip-flop latches the output signal of the critical path at the clock edge with a possible timing error, while the shadow latch (controlled by a delayed clock signal) latches the signal a fraction of a cycle later, which guarantees to receive the correct value. Consequently, when the shadow latch and the main FF values do not agree, indicated by the comparator, timing error is detected. Then, by replaying instructions at lower frequency, the processor is able to recover from the timing error with a small re-execution cost.

Recently, Intel has demonstrated a timing-speculative microprocessor test chip in [52]. Their measurement results show that the resilient design enables 25% throughput gain over a conventional design by eliminating the guardband from circuit dynamic variations and an additional 7% throughput increase from exploiting the path-activation probabilities for timing error rate reduction. The above benefits have motivated a large amount of recent research efforts on design and optimization techniques for timing-speculative circuits (e.g., [55–58]).

3.2.4 Motivation

Fig. 3.1 presents an example MSV design, wherein the black cells have critical timing requirement and hence need high V_{dd} to eliminate timing errors, while the white cells with low timing requirement only need low V_{dd}. The solid-line rectangles represent the voltage islands assigned with high-V_{dd}, while the dashed-line



(a) MSV design without timing speculation (b) MSV design with timing speculation

Figure 3.1: An example to motivate MSV design for timing-speculative circuits.

rectangles represent those assigned with low-V_{dd}.

Assuming only four voltage islands are allowed in this MSV design, without timing speculation we can only have the voltage island design as shown in Fig. 3.1(a), leading to limited power savings. With timing speculation, however, we can have the MSV design as shown in Fig. 3.1(b) that has much less power consumption by allowing timing-critical cells residing in low-V_{dd} islands. In other words, with the capability of online timing error correction, the new MSV designs allow more aggressive voltage scaling, and the associated power savings are usually much higher than the power penalties paid to correct infrequent timing errors. The above considerations have motivated the MSV design for timing-speculative circuits investigated in this chapter.

3.3 MSV Design for Timing-Speculative Circuits

3.3.1 Problem Formulation

The MSV design problem for timing-speculative circuits investigated in this work can be formulated as follows:

Problem: Given

- A timing-speculative circuit $\mathcal{C}$, equipped with timing speculators, such as Razor [9];
- A circuit placement $\mathcal{P}$ with $m \times n$ grids , where each grid g_{ij} is placed at position (i, j);
- The probability function $F_{ij}(V_{dd})$[1] for timing errors to occur in grid g_{ij} with respect to V_{dd}, where V_{dd} is the supply voltage;
- The number of voltage islands N_{VI};
- The performance degradation constraint caused by re-execution, represented by throughput degradation ratio $\eta\%$;

to determine a circuit partitioning **P** and a voltage assignment **V** for voltage island generation, such that the power consumption P_{total} of targeted circuit $\mathcal{C}$ is minimized under the performance constraint.

As it is essential to conduct re-computation when timing errors occur, the power consumption of timing-speculative circuits is:

$$P_{total}(\mathbf{P},\mathbf{V}) = P(\mathbf{P},\mathbf{V}) \cdot (1 + error(\mathbf{P},\mathbf{V}) \cdot penalty) \quad , \tag{3.3}$$

where $P(\cdot)$ is the power function (including dynamic power P_d and static power P_s) of circuit $\mathcal{C}$ in one clock cycle after circuit partitioning **P** and voltage assignment **V** are given, $error(\cdot)$ is the error probability function, $penalty$ is the cost including both the cycles of wasted execution that must be discarded and the time spent on checkpointing and re-execution. Meanwhile, we need to ensure the performance

[1]The error probability function $F_{ij}(V_{dd})$ of each grid g_{ij} can be acquired by timing simulation of the targeted circuit with representative workloads.

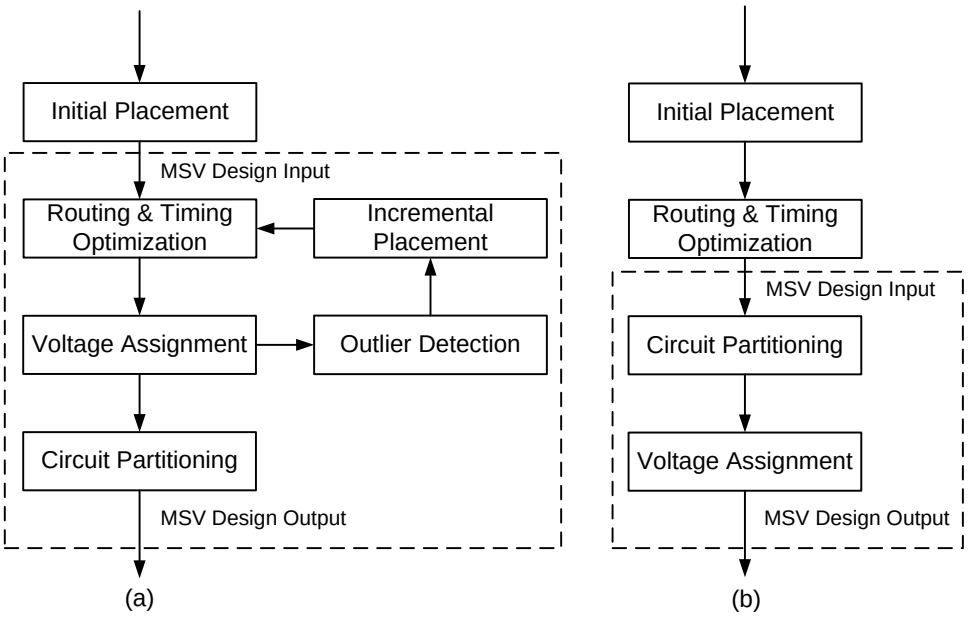


Figure 3.2: Design flow for MSV design.

constraint:

$$Th(\mathbf{P},\mathbf{V}) = \frac{1}{(1+error(\mathbf{P},\mathbf{V})\cdot penalty)} > 1-\eta\% \quad , \tag{3.4}$$

where $Th(\cdot)$ is the equivalent circuit throughput considering performance penalty for timing error correction.

Similar to prior works (e.g., [48–50]), we assume that only rectangular voltage islands are allowed, because voltage islands with arbitrary shapes generally lead to difficulty in power-supply network design. Note that, the hardware cost of MSV design (e.g., the overhead of voltage level shifters [44, 59]) is strongly related to the number of voltage islands, which is also considered in this work.

3.3.2 MSV Design Flow

Generally speaking, the MSV design process consists of two components: circuit partitioning and voltage assignment. For conventional “error-free” circuits, the effectiveness of MSV design is inevitably limited by the amount of available slack on each timing path. Consequently, the so-called ”outliers” (the circled cells in Fig. 3.1) would cause disproportionately expensive penalty to final voltage island generation. In order to mitigate this issue, previous work proposed an incremental placement method to eliminate these unwanted ”outliers” whenever possible. This design flow (see Fig. 3.2(a)) results in high design complexity, and its effectiveness is also not guaranteed.

For timing-speculative circuits, however, such ”outliers” can be naturally tolerated since circuit timing error rate is now a trade-off parameter (instead of a hard constraint) in design space. Consequently, it is not necessary to employ a complex incremental placement procedure. The simple design flow of our proposed methodology is depicted in Fig. 3.2(b), wherein we partition the circuit into voltage islands first and then assign voltages to them with timing error rate considerations.

3.4 Voltage Island Generation

3.4.1 Partitioning Model

How to partition a circuit into rectangular voltage islands has been well studied in [48, 60]. As described in Fig. 3.3, arbitrary partitioning allows any partitioning with rectangular tiles, slicing partitioning performs slicing through recursive cuts, and $p \times q$ partitioning cuts the circuit into $p \times q$ coarse grids. [48] proved that the optimal slicing partitioning result is a 2-approximation for the optimal arbitrary partitioning. As a special type of slicing partitioning, $p \times q$ partitioning is used

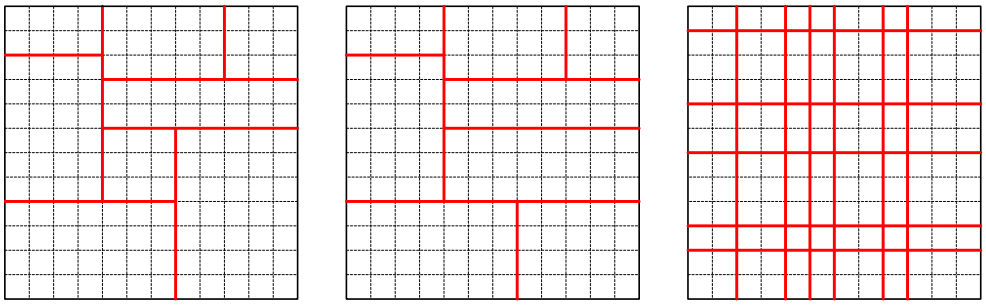


Figure 3.3: Three types of rectangular partitioning.

in [48] to provide the initial grids that are merged later to form voltage islands, which is also used in our work.

3.4.2 DP-Based Voltage Island Generation

To solve the proposed voltage island generation problem for timing-speculative circuits, we resort to a DP-based algorithm that enumerates all combinations of the horizontal and vertical cuts.

Given the error probability function $F_{ij}(V_{dd})$, we can have the power consumption of each grid g_{ij} considering power penalties,

$$P_{ij}(V_{dd}) = (P_d(V_{dd}) + P_s(V_{dd})) \cdot (1 + F_{ij}(V_{dd}) \cdot penalty) \quad , \tag{3.5}$$

where $P_d(\cdot)$ is dynamic power function and $P_s(\cdot)$ is static power function. By solving Eq. 3.5, we can easily obtain the optimal supply voltage V_{dd}^* for which the power consumption of g_{ij} has the optimal value P_{ij}^*.

Let an $m \times n$ array $\mathbf{A}$ with $A_{ij} = P_{ij}^*$ represent the optimal power consumptions of all the grids, and $R(x_1, y_1; x_2, y_2)$ represent a rectangular region covering the grids $\{g_{ij} | x_1 \leq i \leq x_2, y_1 \leq j \leq y_2\}$. For a region $R(x_1, y_1; x_2, y_2)$, we can just replace the power and error probability functions in Eq. 3.5 with the corresponding

$$W_s^*(R(x_1,y_1;x_2,y_2)) =$$
$$\min_{1\le t<s}\left\{\min_{x_1\le i<x_2, y_1\le j<y_2}\left\{\begin{array}{l} W_t^*(R(x_1,y_1;i,y_2))+W_{s-t}^*(R(i+1,y_1;x_2,y_2), \\ W_t^*(R(x_1,y_1;x_2,j))+W_{s-t}^*(R(x_1,j+1;x_2,y_2)) \end{array}\right\}\right\} . \tag{3.6}$$

terms of this region, and then use such an equation to describe the relationship between power and supply voltage. Similar to the case of a grid g_{ij}, we can also find out the optimal supply voltage V_{opt} for such a region. By denoting the optimal total power consumption of this region with all the grids in it driven by V_{opt} is P_R^*, we define the power wastage of a region $R(x_1,y_1;x_2,y_2)$ as,

$$W(R) = P_R^* - \sum_{g_{ij}\in R} P_{ij}^* \quad . \tag{3.7}$$

Therefore, we can have the power wastage of a partitioning $\mathbf{P} = \{R_i\}$ as follows,

$$W(\mathbf{P}) = \sum_{1\le i\le N_{VI}} W(R_i) \quad , \tag{3.8}$$

where N_{VI} is the specified voltage island number.

With the above definitions, we can have the recursion under slicing partitioning as shown in Eq. 3.6. A simple example is described in Fig. 3.4 to show the enumeration procedure. In the 9×5 grids with s islands allowed, we can choose an either vertical (e.g., $i=5$) or horizontal (e.g., $j=2$) cut to partition it, and allow t and $(s-t)$ voltage islands in the newly-cut rectangular regions, respectively. This enumeration ensures DP to find the optimal partitioning.

Note that, since the error probability functions of grids $\{g_{ij}\}$ and regions $\{R_i\}$ are fed into the DP solver as inputs to calculate the optimal power consumptions, we assume the error occurrences in different grids are independen[2]. This allows

[2]This is a simple approximation to reduce computational complexity, and its impact is

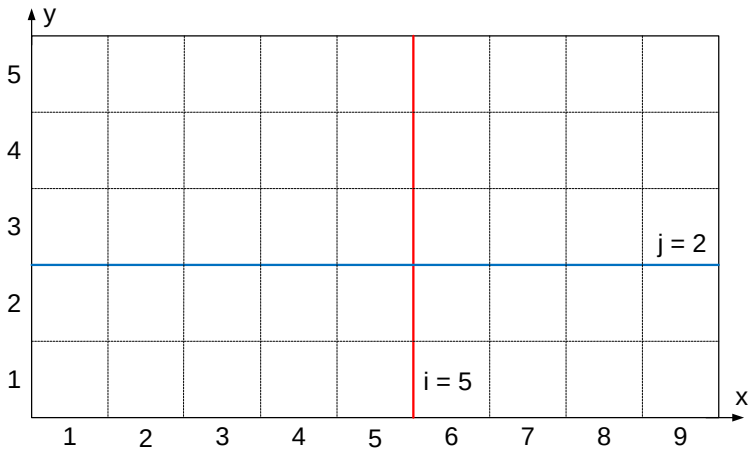


Figure 3.4: An example to show the enumeration process.

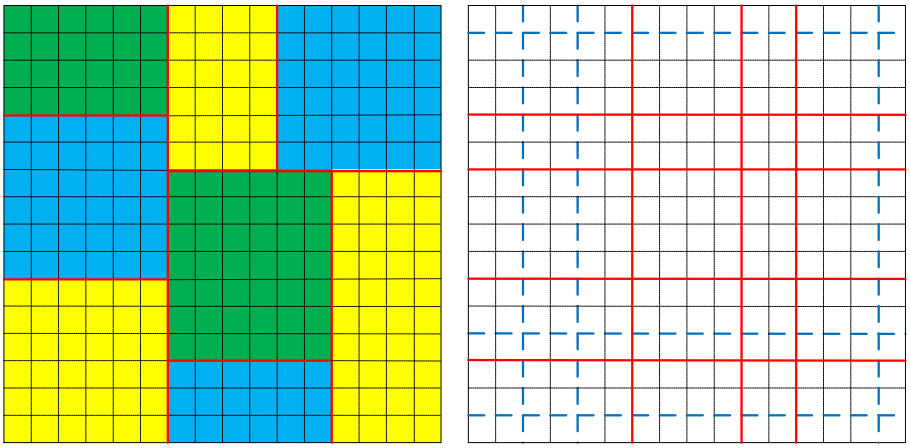
us to calculate the error probability function $F_R(V_{dd})$ of a region R, given the error probabilities of the grids $\{g_{ij}|g_{ij} \in R\}$. For example, we can calculate the error probability of a region R consisting of two regions R_1 and R_2 according to Eq. 3.9 as follows,

$$F_R = F_{R_1} + F_{R_2} - F_{R_1} \cdot F_{R_2} \quad . \tag{3.9}$$

3.4.3 Coarse Grid Reconstruction

The circuit partitioning problem under slicing partitioning can be solved by DP optimally [48]. However, the placement size $m \times n$ at the cell-level is usually too large to employ DP directly in practical applications. To avoid the huge time and memory costs, one intuitive and viable method is to partition the $m \times n$ grids into $p \times q$ coarse grids as shown in Fig. 3.3(c), and then apply DP to the coarse grids. Clearly, the effectiveness of the MSV design is limited by the heuristic coarse grid construction algorithm due to search space reduction. In [48], a heuristic-based partitioning algorithm according to [60] is used to construct the $p \times q$ coarse grids

reflected in our experimental results.



(a) Voltage islands (b) Coarse grid reconstruction

Figure 3.5: An example to show the coarse grid reconstruction process.

$$metric(L) = \sum_i (sd(R_i) - sd(R_{i1}) \cdot \frac{A(R_{i1})}{A(R_i)} - sd(R_{i2}) \cdot \frac{A(R_{i2})}{A(R_i)}) \cdot \frac{A(R_i)}{\sum_i A(R_i)} \quad . \quad (3.10)$$

before voltage island generation. With such fixed coarse grids, only a constrained MSV design solution space can be explored. Different from their solution, we propose a novel coarse grid reconstruction algorithm to explore more solution space by reconstructing coarse grids and applying DP iteratively.

As discussed in Section 3.4.2, given an array **A** consisting of many grids, DP can achieve an optimal solution for this array **A** if enough runtime is allowed. With this property, if we ensure the optimal voltage island design of the last $p \times q$ partitioning is still kept as a solution point in a newly-constructed coarse grids, it is guaranteed to achieve a solution not worse than the last one. Let us explain it using the following example.

Suppose we would like to generate 8 voltage islands based on a 16×16 placement and we decide to use 7×8 coarse grids to save runtime, we can perform any partitioning to divide this 16×16 placement into coarse grids and then use the DP

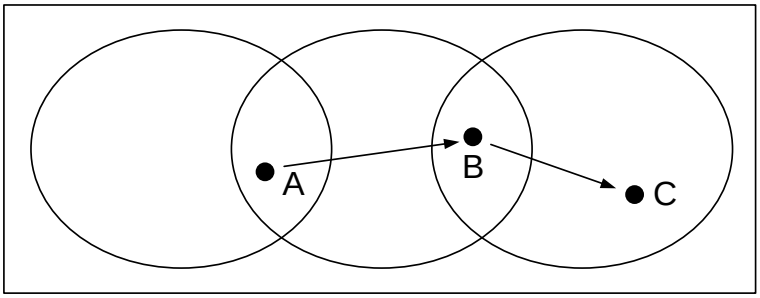


Figure 3.6: Solution space changes with iterative coarse grid reconstruction.

algorithm in Section 3.4.2 to generate voltage islands. By doing so, we can achieve an optimal voltage island design with the current 7×8 coarse grids. Without loss of generality, we assume the generated voltage island design[3] is the one depicted in Fig. 3.5(a). To construct a new 7×8 coarse girds for further exploration, it is obvious that we need to determine how to partition the 16×16 placement using 6 vertical lines and 7 horizontal. It is worth noting that, if we keep all the grid lines going through the boundaries of voltage islands as the new coarse grid lines (see the solid lines in Fig. 3.5(b)), we can make sure the current generated voltage islands (see Fig. 3.5(a)) is still achievable with newly-constructed coarse grids. In other words, given the 3 vertical lines and 4 horizontal lines that go through the boundaries of voltage islands, no matter how we assign the other 3 vertical lines and 3 horizontal lines (see the dashed lines in Fig. 3.5(b)) to partition the 16×16 placement, the voltage island design in Fig. 3.5(a) is one possible solution with the reconstructed coarse grids. As DP can always find out an optimal solution with given coarse grids, we should at least find a solution as good as the previous one and hence it is guaranteed to get a solution not worse than the design in Fig. 3.5(a) under the new 7×8 partitioning.

[3]The voltage islands are represented by rectangular blocks and plotted out using solid lines.

The above optimization process can be clarified using Fig. 3.6. The rectangle represents the entire solution space for DP to explore based on the original $m \times n$ fine-grained grids, and the ellipses represent the sub-spaces after partitioning into $p \times q$ coarse grids. Once the $p \times q$ coarse grids are obtained, we can use DP to achieve the optimal solution in the corresponding sub-space. Therefore, by reconstructing the sub-space and applying DP iteratively, we can get the optimal solution in each sub-space one by one: Point A, Point B, Point C, etc.

3.4.4 Reconstruction Algorithm

To keep the previous partitioning inside the reconstructed solution space, we would like to use those lines going through the boundaries of voltage islands as coarse grid lines. However, in most cases, there are still some vertical and horizontal lines (see the dashed lines in Fig. 3.5(b)) left to obtain a different $p \times q$ partitioning, which can be used to explore new solution space. We propose a heuristic-based algorithm to obtain new $p \times q$ partitionings, which selects $(p-1-p_0)$ vertical coarse grid lines out of $(m-1-p_0)$ candidate lines and $(q-1-q_0)$ horizontal coarse grid lines out of $(n-1-q_0)$ candidate lines. Here, p_0 and q_0 are the number of vertical and horizontal lines determined by the boundaries of voltage islands.

The proposed heuristic algorithm is based on the intuition that, for MSV design, it tends to group those grids with similar voltage requirement together, in order to achieve more power savings. In previous works (e.g., [48, 51]), voltages that guarantee no timing violations are chosen. However, for timing-speculative circuits, it is preferable to use the "optimal" voltage values obtained by trading off reliability with power (see Section 3.4.2). In this work, to support the proposed heuristic algorithm, we use an evaluation metric to reflect the similarity of the grids that are partitioned into the same islands and we tend to select those grid lines with larger metric values during the coarse grid line selection process.

#	$\mathbf{S_p}$ and $\mathbf{S_q}$, the grid line sets for $p \times q$ partitioning;
#	$\lvert\mathbf{S_p}\rvert$ and $\lvert\mathbf{S_q}\rvert$, the set sizes; α, convergence tolerance.
1.	Initialize $p \times q$ partitioning randomly;
2.	**REPEAT** for each iteration
3.	Apply DP to generate voltage islands;
4.	**IF** $\lvert\Delta W_s^*\rvert < \alpha$
5.	Break;
6.	**ELSE** // reconstruct $p \times q$ partitioning
7.	Set $\mathbf{S_p} = \emptyset$, $\mathbf{S_q} = \emptyset$;
8.	Put the vertical lines of island boundaries $\rightarrow \mathbf{S_p}$;
9.	Put the horizontal lines of island boundaries $\rightarrow \mathbf{S_q}$;
10.	Use ε-greedy to select $(p-1-\lvert\mathbf{S_p}\rvert)$ lines $\rightarrow \mathbf{S_p}$;
11.	Use ε-greedy to select $(q-1-\lvert\mathbf{S_q}\rvert)$ lines $\rightarrow \mathbf{S_q}$;
12.	**END REPEAT**

Figure 3.7: The overall algorithm flow of proposed voltage island generation methodology.

Given a circuit partitioning $\mathbf{P}$, if the grid line L intersects n original islands $\{R_i | 1 \leq i \leq n\}$ to cut them into $2n$ new islands $\{R_{ij} | 1 \leq i \leq n, 1 \leq j \leq 2\}$, $metric(L_k)$ is defined as in Eq. 3.10, wherein $R_i = R_{i1} \cup R_{i2}$, $sd(R_i)$ is the standard deviation of all the optimal voltage values of the grids in region R_i, and $A(R_i)$ is the number of grids in it.

Note that, to avoid being trapped in local optimal points, we use ε-greedy to select the coarse grid lines for $p \times q$ partitioning. That is, we set up a probability parameter ε (e.g., $\varepsilon = 10\%$), and hence we have the probability of ε to select a grid line randomly, instead of the one with largest metric defined in Eq. 3.10.

The overall algorithm flow is summarized as shown in Fig 3.7. The optimization procedure is repeated until the terminal condition is satisfied (see Line 4). In each iteration, we use DP to get the optimal solution in the current solution space

Bench.	TG #	TFF #	T_{cp} (ns)	(m,n,p,q)	Island #	Cost(%)
38584	21021	1426	6.96	(20,20,10,10)	5	5.14
s38417	23949	1636	6.12	(20,20,10,10)	5	6.76
des_perf	155746	9105	13.7	(30,30,15,15)	10	6.63
ethernet	164912	10752	11.28	(30,30,15,15)	10	7.46
AVERAGE						6.50

TG #, total gate count; TFF #, total FF count; T_{cp}, the operating clock cycle period;

Island #, the specified voltage island number.

Table 3.1: Experimental setup.

(see Line 3) and then employ the proposed reconstruction algorithm to reconstruct the solution space to be explored in the next iteration (from Line 7 to Line 11).

3.5 Experimental Results

3.5.1 Experimental Setup

To evaluate the effectiveness of the proposed voltage island generation methodology, we conduct experiments on several large ISCAS'89 and IWLS'05 benchmark circuits. We synthesize these circuits on a 90nm technology, conduct physical design, and obtain timing information using commercial EDA tools. To take process variation effects into consideration, we perform Monte Carlo simulations to inject gate-level delay variations following Gaussian distribution.We conduct simulations with random inputs and each simulation is performed with 100,000 cycles. By performing simulation for representative workloads and recording error rates occurring in the grids under various operational clock periods, we achieve error probability function $F_{ij}(V_{dd})$ for each grid. We employ the power and delay models used in [38, 43, 54] in our experiments. All the experiments are conducted on a *2.8GHz* PC with *4GB* RAM.

We perform offline timing analysis with false paths excluded according to [61]

Bench.	$MSV_{baseline}$		$MSV_{reconstruction}$			MSV_{ts}				$MSV_{proposed}$					
	power	σ	*power*	σ	$\Delta_1(\%)$	*power*	σ	$\Delta Th(\%)$	$\Delta_2(\%)$	*power*	σ	$\Delta Th(\%)$	$\Delta_3(\%)$	$\Delta_4(\%)$	Runtime (s)
s38584	0.852	0.014	0.813	0.015	-4.58	0.793	0.014	-4.28	-6.92	0.689	0.016	-3.55	-13.11	-19.13	2.75
s38417	0.857	0.013	0.835	0.016	-2.57	0.825	0.018	-3.56	-3.73	0.781	0.020	-4.09	-5.33	-8.87	1.92
des_perf	0.862	0.019	0.806	0.017	-6.50	0.674	0.014	-5.52	-21.81	0.598	0.015	-7.03	-11.28	-30.63	17.35
ethernet	0.778	0.018	0.723	0.019	-7.07	0.631	0.015	-6.32	-18.89	0.581	0.012	-5.63	-7.92	-25.32	15.04
AVERAGE					-5.18			-4.92	-12.84			-5.08	-9.41	-20.99	

σ: standard deviation of *power*; Δ_1: *power* difference ratio between $MSV_{reconstruction}$ and $MSV_{baseline}$;

Δ_2: *power* difference ratio between MSV_{ts} and $MSV_{baseline}$; Δ_3: *power* difference ratio between $MSV_{proposed}$ and MSV_{ts};

Δ_4: *power* difference ratio between $MSV_{proposed}$ and $MSV_{baseline}$; ΔTh, performance degradation ratio.

Table 3.2: Results on the proposed reconstruction-based $p \times q$ partitioning.

and use the reported maximum path delay as the operating clock cycle period during timing simulation. For reasonable comparison, a widely-accepted voltage island generation algorithm proposed in [48] is used as the baseline solution and denoted as $MSV_{baseline}$. Because our proposed reconstruction-based $p \times q$ partitioning algorithm is also applicable for the non-TS voltage island generation problem in [48], we replace the corresponding $p \times q$ partitioning algorithm in [48] with ours and keep the rest of algorithm unchanged. This MSV design scheme is denoted as $MSV_{reconstruciton}$. We apply timing speculation directly to the MSV design of $MSV_{baseline}$, and denote this solution as MSV_{ts}. That means, in MSV_{ts} we keep the MSV design of $MSV_{baseline}$ and then perform timing simulation with different voltage assignments to obtain the error probability functions, so that we can achieve the "optimal" voltage assignment and power consumption considering timing speculation. Our proposed solution is denoted as $MSV_{proposed}$. The range of supply voltages allowed for voltage islands to operate is 0.7V to 1.0V in our experiments.

In timing-speculative circuits, we need to add timing error detectors to the receiving end of critical paths. A simple scheme is to transform all the FFs, whose

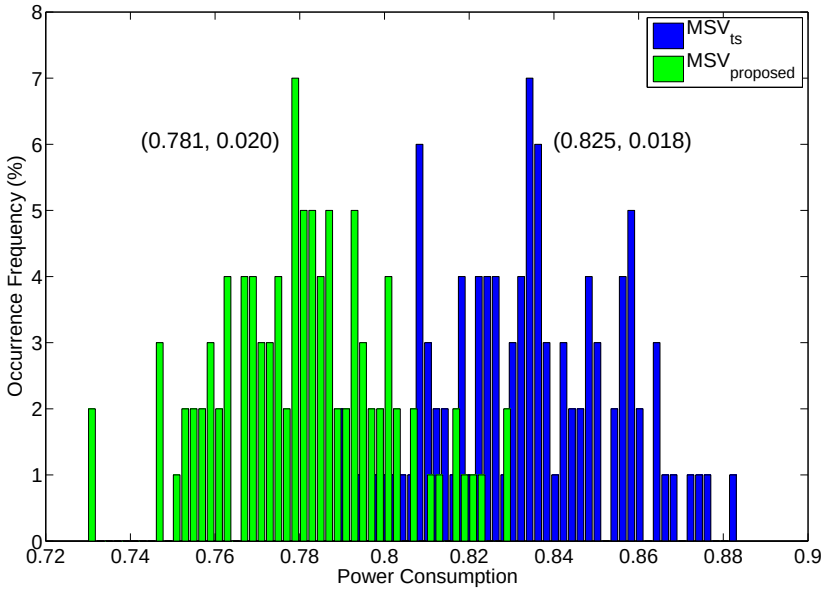


Figure 3.8: Monte Carlo simulation results.

maximum path delays are larger than β of the clock period (e.g., β = 80%), as Razor-FFs. Then, to avoid hold time violation on the shadow latch of Razor-FFs, we need to conduct short path padding and this is achieved by constraining paths that drive Razor-FFs with at least γ of the clock period (e.g., γ = 50%) during synthesis. In this work, once a voltage island design is generated, we perform timing analysis using timing information with voltage scaling considered and then set up Razor-FFs and conduct short path padding using the obtained path delays. Both of these hardware costs are accounted for in our experiments and β and γ are set to be 80% and 50%, respectively. The hardware cost for equipping each Razor-FF is assumed to be 10 gates. The *penalty* in Eq. 3.3 is assumed to be 10 clock cycles similar to prior works (e.g., [12]).

3.5.2 Results and Discussion

In Table 3.1, we report the operating clock period obtained by excluding false paths according to [61], the used parameters (m, n, p, q), the specified voltage island number and the hardware cost to enable timing speculation for each benchmark circuit. To be specific, we set up the values of (m, n, p, q) as $(20, 20, 10, 10)$ for small-scale circuits (e.g., $s38584$ and $s38417$) and as $(30, 30, 15, 15)$ for large-scale circuits (e.g., des_perf and $ethernet$). The average hardware cost to equip the circuits with TS capability (including timing speculator and short path padding cost) is about 6.5%.

To verify the effectiveness of the proposed voltage island generation methodology, we, first of all, perform Monte Carlo simulation to produce 100 sample chips with different variation patterns for each benchmark circuit. In Table 3.2, we report the average power consumption[4] and its standard deviation σ for $MSV_{baseline}$, $MSV_{reconstruction}$, MSV_{ts} and $MSV_{proposed}$, respectively. It is important to note that, the reported results includes the power overhead of MSV design (e.g., level shifters) and power penalties to correct timing errors.

As can be seen from Table 3.2, when compared to $MSV_{baseline}$, the proposed $MSV_{reconstruction}$ can achieve 5.18% power saving on average. This improvement comes from using our proposed $p \times q$ partitioning algorithm to replace the corresponding one in $MSV_{baseline}$ only, which demonstrates the effectiveness of our reconstruction algorithm. In other words, even for non-TS conventional circuits, our proposed solution lead to much more power-efficient MSV designs.

Besides, MSV_{ts} can achieve 12.84% power reduction on average when compared with $MSV_{baseline}$. This improvement reflects the efficacy of timing speculation itself, since in MSV_{ts} we just apply timing speculation directly to the MSV de-

[4]Each power value has been normalized by using the power consumption of the case without MSV design as unit value.

sign of $MSV_{baseline}$. Compared with MSV_{ts}, our proposed methodology $MSV_{proposed}$ can further achieve 9.41% power reduction on average, which reflects the efficacy of explicitly considering timing speculation during the MSV design process. The runtime of the proposed algorithm (see Fig. 3.7) is quite small.

$MSV_{proposed}$ achieve better results because (i) the proposed partitioning model and DP-based voltage island generation method facilitate to identify voltage islands with optimal supply voltages based on circuit slack distribution, which gives the first-level of power saving; (ii) once a voltage island has been formed, another level of power saving can be achieved by minimizing the timing error rates. Any voltage island with only a small number of critical paths (i.e., most circuit paths have relatively large slacks) can fully take advantage of this power saving while maintaining the performance. At the same time, we can observe that the power reduction ratios of these four benchmark circuits are quite different, and we attribute this phenomenon to the unique timing characteristic of each circuit. Generally speaking, if a circuit has gradually-decreasing path delay distribution, the benefit brought by timing speculation can be larger than that of those circuits with a sharply-declining delay distribution. This is because, in the latter case, a large number of paths may fail at the same time in the design when voltage overscaling exceeds a critical point, which causes a steep increase of timing error rate [62].

MSV_{ts} and $MSV_{proposed}$ would suffer from performance degradation caused by infrequent timing errors. We report this performance degradation in Table 3.1 and denote it as ΔTh, compared to the case that the circuit uses the maximum path delay as its operational clock period. On average, MSV_{ts} and $MSV_{proposed}$ have 4.92% and 5.08% throughput degradation, respectively. However, it is important to note that, for $MSV_{baseline}$ without timing error correction capability, designers usually have to reserve a large timing guard band (e.g., 15% of maximum path delay) to tolerate variation-induced timing uncertainty and hence system through-

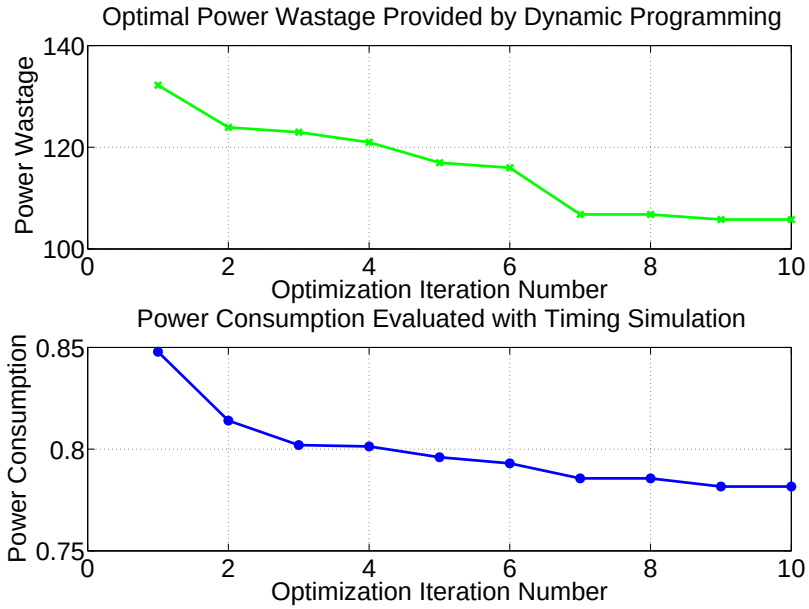


Figure 3.9: Power wastage and power consumption wrt. optimization iteration number.

put is degraded due to lower operational frequency [52]. From this perspective, if we consider the timing guardband existing in the non-TS solution $MSV_{baseline}$, the performance of MSV_{ts} and $MSV_{proposed}$ would be actually better than that of $MSV_{baseline}$.

To get more details of the proposed methodology, we take $s38417$ as an example in the following experiments. In Fig. 3.8, we show the results of MSV_{ts} and $MSV_{proposed}$ with process variation effects after performing Monte Carlo simulation. The corresponding mean value of power consumption and standard deviation for each case are depicted in the figure in the form of (μ, σ), which, again, demonstrates the benefits of $MSV_{proposed}$. In Fig. 3.9, we plot the curves to reflect the changes of both the power wastage provided by DP and the power consumption evaluated by timing simulation with error penalties taken into account. As can be

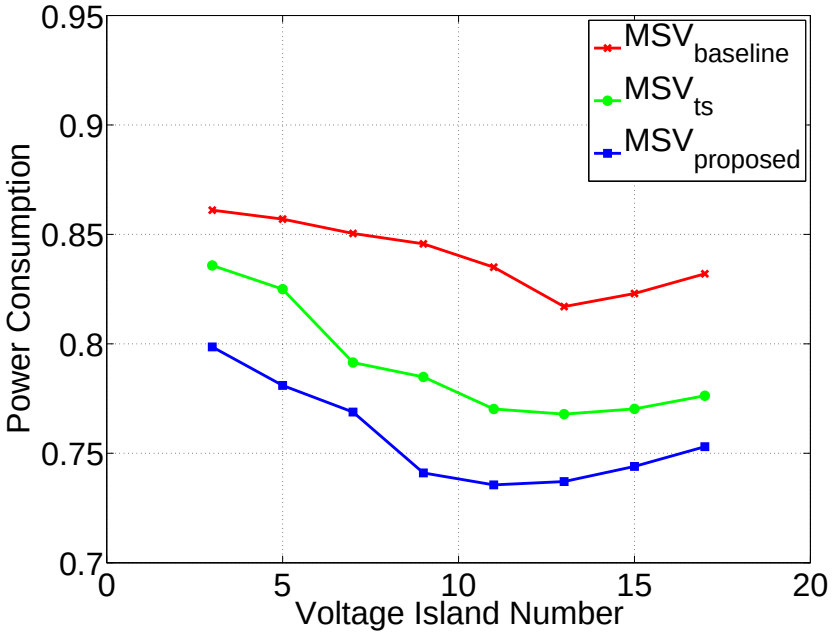


Figure 3.10: Power consumption wrt. voltage island number.

seen, the power wastage is decreased all the time, which proves the effectiveness of the reconstruction algorithm to explore new solution space and guarantee the power wastage to be optimized step by step, as discussed in Section 3.4.3. Note that, this can be used to trade off the algorithm runtime with optimization quality during design process. Moreover, with respect to the optimization iteration number, the two curves descends in the same manner. The similar trends of these two curves can prove the effectiveness of our proposed optimization process.

To investigate the effects with different specified voltage island number, we vary the number of islands and get the power consumption curves of $MSV_{baseline}$, MSV_{ts} and $MSV_{proposed}$ as described in Fig. 3.10. Clearly, with different number of voltage islands, $MSV_{proposed}$ always outperforms the other solutions. It can be also observed that, with increasing number of allowed voltage islands in the MSV design, the power savings of all these solutions increase in the beginning, but de-

crease in the end. This is because, more voltage islands allow fine-grained voltage assignments that satisfy the performance constraint of each individual island, leading to better power savings. However, more voltage islands also incur higher cost for the supporting circuitries (e.g., level shifters). Consequently, when the number is too large, the benefit provided with fine-grained voltage assignment cannot compensate the associated power cost.

3.6 Conclusion

Region-based MSV design has been used as an effective technique to reduce power consumption and attracted lots of research interests. However, all of the previous MSV works try to guarantee ”always correct” operations, which greatly limits the design flexibility. In this work, we formulate the MSV design problem for timing-speculative circuits, and propose a novel DP-based algorithm to generate voltage islands. Experimental results based on various benchmark circuits demonstrate that the proposed methodology is able to significantly reduce power consumption of timing-speculative circuits with acceptable performance degradation.

□ End of chapter.

Chapter 4

Clock Skew Scheduling

4.1 Introduction

Clock skew scheduling (CSS), which treats clock skew as a manageable resource instead of design liability, is an effective technique to improve IC performance, by assigning intentional clock arrival times to internal flip-flops (FFs). Earlier works in this domain (e.g., [63–66]) focused on clock period reduction to maximize IC timing performance. With technology scaling, process variation has become a serious concern for circuit design and the earlier performance-driven CSS solutions may result in low manufacturing yield due to timing uncertainty caused by process variation. Consequently, various yield-driven CSS techniques (e.g., [67–69]) were developed to maximize timing yield under a certain clock period.

Despite the different design objectives and optimization methods, a common design constraint of existing CSS techniques is that they need to guarantee the timing correctness of the circuit after skew adjustment, even in the worst case scenario. The ever-increasing process variation effects hence pose serious challenges for these techniques since a large timing guard band has to be reserved to tolerate timing uncertainty, leading to rather limited performance improvement room for

CSS techniques.

By allowing infrequent timing errors and achieving timing error resilience with error detection and correction techniques, timing speculation techniques such as Razor [9] enable highly energy-efficient "better than worst-case" designs, and hence have attracted lots of attention from both academia and industry [5, 7, 11–13, 70]. For circuits equipped with timing speculation capability (i.e., timing-speculative circuits), since there is no need to guarantee "always correct" operation in such designs, we can afford to have more aggressive skew optimization strategies to improve circuit performance without necessarily reserving large timing guard bands. Recently, a post-silicon clock skew tuning framework [71] has been proposed to manipulate timing slacks of different FFs according to collected timing error information at runtime. However, how to conduct pre-silicon clock skew scheduling at design stage for such timing-speculative designs, to the best of our knowledge, has not been studied in the literature yet. The pre-silicon clock skew scheduling work investigated in this chapter can be easily combined with the post-silicon clock skew tuning work.

Motivated by the above, in this chapter, we first develop a general formulation of CSS problem for timing-speculative circuits, wherein timing error rate and its corresponding impact are explicitly considered. We then propose a novel clock skew scheduling algorithm that consists of two phases to tackle this problem. The first phase is based on gradient-descent method (GDM), considering arbitrary continuous skew values can be assigned to FFs. Such assumption leads to a large number of skew values. In reality, it is very difficult, if not impossible, to implement so many different skews. CSS therefore has to be constrained to a limited number of skew values, and such an optimization problem is known as multi-domain clock skew scheduling (MDCSS) [72–76]. To seriously take this practical concern into consideration, in the second phase we develop a novel algorithm based on

steepest-descent method (SDM) that uses the output of the first phase as an initial solution. As shown in experimental results on various benchmark circuits, our proposed CSS algorithms are able to significantly reduce the overall timing error rate of the circuit, thus dramatically improving its throughput.

The remainder of this chapter is organized as follows. In Section 4.2, we present the preliminaries and related work. The CSS problem in timing-speculative circuits is then formulated in Section 4.3. The corresponding GDM-based CSS algorithm and SDM-based MDCSS algorithm are then detailed in Section 4.4 and Section 4.5, respectively. Next, Section 4.6 presents our experimental results on various benchmark circuits. Finally, Section 4.7 concludes this chapter.

4.2 Preliminaries and Motivation

4.2.1 Background

A synchronous circuit (see Fig. 4.1 for a simple example) with edge-triggered storage elements (e.g., flip-flops) can usually be modeled as a directed graph $G(V,E)$ as depicted in Fig. 4.2. In Fig. 4.1, the squares represent FFs and the circles represent combinational logics. In the timing constraint graph as shown in Fig. 4.2, each node $(v_i \in V)$ represents a FF and each arc $(e_{ij} \in E)$ represents the longest/shortest path from FF_i to FF_j.

Let s_i be the clock arrival time of v_i, and D_{ij} and d_{ij} be the maximum and minimum path delays of e_{ij} respectively, the setup-time and hold-time constraints of traditional CSS problem without timing speculation are as below:

$$\begin{aligned} s_i - s_j &\leq T_{cp} - T_{setup} - D_{ij} \quad , \\ s_i - s_j &\geq T_{hold} - d_{ij} \quad , \end{aligned} \tag{4.1}$$

where T_{cp} is the clock period, and T_{setup} and T_{hold} are the setup time and the hold time of FFs, respectively. In order to solve this CSS problem using a graph-

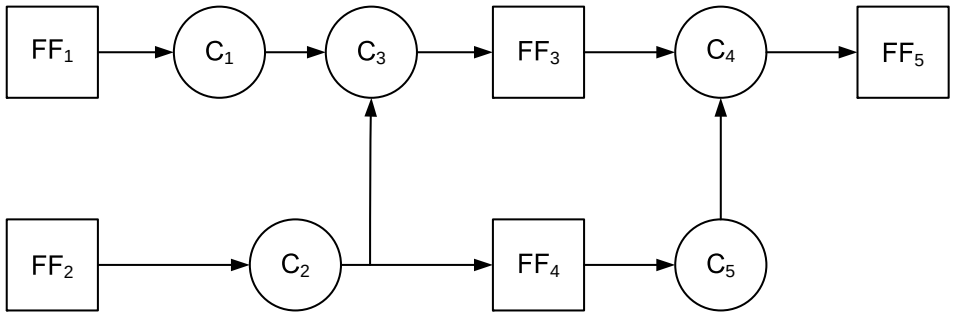


Figure 4.1: A simple example of synchronous digital circuit.

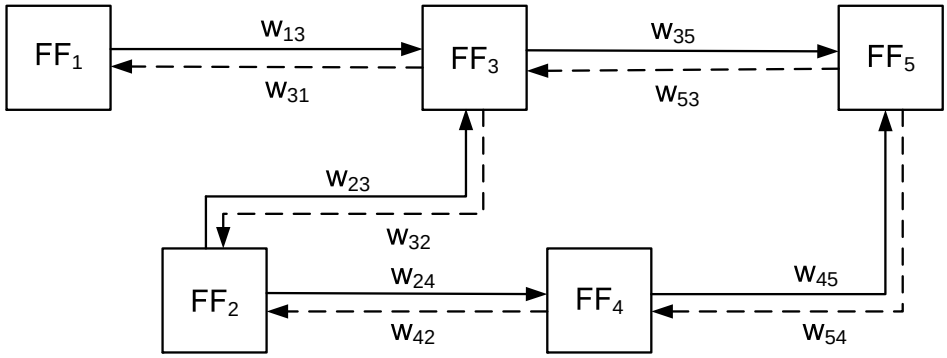


Figure 4.2: Timing constraint graph of circuit example.

theoretic approach, these timing constraints would usually be presented in timing constraint graph (see Fig. 4.2) by replacing the hold time constraint of the path from FF_i to FF_j with edge weight $w_{ji} = -(T_{hold} - d_{ij})$ and replacing the corresponding setup time constraint with edge weight $w_{ij} = T_{cp} - T_{setup} - D_{ij}$. The solid directed lines in Fig. 4.2 imply the setup time relation and the dashed directed lines imply the hold time relation[1].

[1]Interested readers may refer to [77] for more background about CSS.

4.2.2 Clock Skew Scheduling

The clock signals in synchronized sequential circuits can arrive at storage elements at different times, and such delay difference of clock signals is referred to as *clock skew*. Conventionally clock skews should be minimized when designing clock distribution network, however, properly assigning clock arrival times to FFs, known as CSS, can help solve the problem that the maximum achievable operational frequency is limited by the maximum datapath delay in the circuit. There has been significant effort to explore CSS to improve performance. Generally speaking, clock skew scheduling can be classified into two categories by different optimization objectives: performance-driven ones [63, 64] to achieve the highest operational frequency and timing yield-driven ones [65–69] to maximize yield under a given clock period.

Some early works [63, 64, 66] laid a foundation by formulating the CSS problem and solving it optimally. Huang *el al.* [42] argued that these optimal solutions do not achieve the lower bound of clock period, since the hold time constraints often limit the feasible clock period, and proposed delay insertion into the logic network as a post-processing step to solve the hold time violations and help improve the feasible timing schedule.

However, even after delay insertion, it is known that the minimum clock period of a synchronous circuit achievable through CSS is still limited by the uncertainties of the data-propagation times on local data paths, caused by the ever-increasing process variations. Considering this variability of critical path delays, some prior works (e.g., [63, 64]) allocated a safety margin with both upper and lower bounds to each feasible region of clock skews in advance to minimize clock period. Intuitively, since it seems more reasonable to target a skew close at the middle of skew feasible region, Kourtev and Friedman [65] proposed an original formulation of this problem and solved it using quadratic programming to mini-

mize the total least square of skews. Wei *et al.* [67] optimized clock slacks using an incremental slack distribution method to tolerate process variation. Albrecht *et al.* [66] modeled the variations on critical path delays with a single variable and transformed the yield optimization problem to a minimum mean cycle problem to guarantee safety margins. Tsai *et al.* [68] modeled the problem as a minimum cost-to-time ratio problem by introducing variance of path delay distribution into the feasible skew region to consider the statistical difference of critical path delays. Recently, Wang *el al.* [69] argued that all prior works cannot handle non-Gaussian critical path delays, and hence proposed a formulation of yield-driven clock skew scheduling technique under non-Gaussian variations.

As mentioned earlier, one practical concern regarding CSS is that due to the high vulnerability to process variations, a wide spectrum of dedicated clock skews is very difficult to implement in a reliabile manner [72–76]. Therefore, the concept of MDCSS that allows only a few number of clock skew domains and assigns each flip-flop onto one of the domains is first proposed in [72], whererin it is formulated as a mixed integer programming problem and solved by a SAT-based algorithm. In [75], the method based on multi-level clustering algorithm with a skew affinity metric is proposed to reduce the complexity. In [73], a heuristic algorithm is used to minimize the number of skew domains for a clock period with runtime reduction. In [76], an initial solution is given at first, and then iteratively optimized. In [74], the hardness of MDCSS problem is formally studied, and an optimal search algorithm is proposed together with pruning techniques to save the effort of feasibility check.

The above works all try to conduct CSS with the guarantee that there would never be timing errors to occur. With this serious promise, the traditional CSS has to reserve a rather large timing guard band, which limits the benefits brought by conducting CSS.

4.2.3 Timing Speculation

Circuit-level timing speculation technique, being able to detect timing errors at online stage, react to the error quickly and recover from it by rolling back to a known-good pre-error state, has become one of the most promising solutions to deal with the ever-increasing static and dynamic variation effects with technology scaling. With timing speculation, timing error is no longer the evil that has to be avoided at the cost of timing guard band. It provides designers the opportunity to trade off error rate with operational clock period.

Various techniques [9, 18, 78] are presented for online timing error detection. Without loss of generality, let us consider *Razor* flip-flop [9], one of the representative techniques, to demonstrate how timing error detectors work. A Razor-FF contains a main flip-flop, a shadow latch and some additional control logic, to detect timing errors. The main flip-flop latches the output signal at the clock edge with possible timing error, while the shadow latch guarantees to receive the correct value, by latching the signal a fraction of a cycle later. Consequently, when the shadow latch and the main FF values do not agree, a timing error is detected. To make use of timing speculation technique, it is necessary to replace all critical FFs that are driven by speed-paths (i.e., critical or near-critical paths) of the circuit with Razor-FFs (or other timing speculators).

For microprocessors, timing error recovery can be achieved with microarchitectural support [34]. That is, when a timing error is detected, the processor pipeline is flushed and the correct result from the shadow latch is returned back into the pipeline. Then, by replaying instructions (at possibly lower frequency), the processor is able to recover from the timing error [27].

Timing error recovery inevitably incurs some performance loss and extra energy consumption. Therefore, it is essential to reduce timing error rate (TER) to optimize timing-speculative circuits [12]. Various optimization techniques (e.g., [11,

13, 14, 19, 28]) have been presented for timing-speculative circuits in the literature. The key issue in this optimization problem is to reshape the path delay distribution of the circuit so that those frequently-exercised timing paths are optimized with more timing slack while other paths are allowed to have timing errors.

4.2.4 Motivation

As discussed above, in this work we formulate the CSS problem targeting timing-speculative circuits and present both a GDM-based CSS algorithm and a heuristic search-based MDCSS algorithm to maximize circuit performance. On the one hand, the proposed technique is motivated by the observation that a large design guard band needs to be reserved for the traditional clock skew optimization techniques, due to the increasing process variation. With timing speculation, we do not need to guarantee "always correct" operations any longer, dramatically increasing design flexibility and improvement room of CSS techniques. On the other hand, CSS can manipulate the timing slacks of different FFs, so that those frequently-sensitized critical FFs serving as receiving ends of critical paths can be better taken care of by allocating more timing slacks. By doing so, the error rate of such a timing-speculative circuit can be reduced and its performance would be improved.

Obviously, these two techniques can naturally complement each other to improve system performance, motivating this work to study how to effectively conduct CSS in timing-speculative circuits.

4.3 Problem Formulation

The CSS problem formulation for timing-speculative circuits is quite different from traditional one (e.g., [69]), because our optimization objective is not the absolute clock period and we need to differentiate the contributions of various circuit

paths to the system's overall timing error rate. We detail our problem formulation as follows.

Generally speaking, in timing-speculative circuits, we roll back the system once timing error is detected and then lower system frequency for a short while to re-compute the result in the failure cycle. Similar to [12], we trade off timing error rate with operational clock period to minimize the equivalent clock period T_{ecp} that can be expressed as follows:

$$T_{ecp} = (1 + error(T_{cp}, \vec{s}) \cdot penalty) \cdot T_{cp} \quad , \tag{4.2}$$

where T_{cp} is the operational clock period, $\vec{s}$ is the vector for skew setting, $error(T_{cp}, \vec{s})$ is the error cycle rate function with regard to T_{cp} and $\vec{s}$, indicating the probability for timing error to occur, and $penalty$ is the penalty due to error occurring. This penalty includes both the cycles of wasted execution that must be discarded when an error occurs and the time spent on checkpointing and re-execution.

To achieve the above optimization objective, if a certain T_{cp} is given, the problem in Eq. 4.2 can be equivalently developed as minimizing the following error probability:

$$error(\vec{s}) = 1 - \prod_{j=1}^{|V|} (1 - error_j(\vec{s})) \quad , \tag{4.3}$$

where $error_j(\vec{s})$ is the error probability of FF_j during one clock cycle, and $|V|$ represents the node number (i.e., FF number) in graph model of the circuit. By solving above optimization problem, we can finally select the best operational clock period T_{cp} to minimize the equivalent clock period as indicated in Eq. 4.2.

To calculate $error_j(\vec{s})$, the error probability of each FF, we need to know the path sensitization probability, which can be acquired by performing functional simulation of the circuit with representative workloads. Note that, we only need to record the sensitized delay information of FFs during this simulation process, but not all the sensitized path delays.

Besides, considering process variation, the path delay becomes a random variable [69], therefore we have

$$\begin{aligned} error_j(\vec{s}) &= \sum_{i \in P_j} \sum_k N_{ijk} \cdot Pr\{s_i - s_j \geq \widetilde{W}_{ijk}\} \quad , \\ \widetilde{W}_{ijk} &\triangleq T_{cp} - T_{setup} - \widetilde{D}_{ijk} \quad , \end{aligned} \tag{4.4}$$

where N_{ijk} is the probability for the k^{th} path from FF_i to FF_j to be sensitized, $\widetilde{D}_{ijk}$ is the corresponding path delay variable of the k^{th} path, s_i is skew value of FF_i, $\widetilde{W}_{ijk}$ denotes a random variable as defined in Eq. 4.4, and P_j is the set of FFs that serve as the beginners of the paths from FF_i to FF_j. Note that, since the problem of hold time constraints can be solved by delay padding [42], we only consider the setup time constraints in this work. $Pr\{s_i - s_j \geq \widetilde{W}_{ijk}\}$ implies the probability for timing error to occur if the setup time constraint as specified in Eq. 4.1 is violated.

By defining the Cumulative Distribution Function (CDF) of $\widetilde{W}_{ijk}$ as

$$F_{ijk}(x) = Pr\{x \geq \widetilde{W}_{ijk}\} \quad , \tag{4.5}$$

where $x = s_i - s_j$, we have

$$error_j(\vec{s}) = \sum_{i \in P_j} \sum_k N_{ijk} \cdot F_{ijk}(s_i - s_j) \quad . \tag{4.6}$$

Note that, if we consider the path delay between FF pair as a variable following a certain distribution, we can have a general form for error probability calculation using an integral form, instead of the summation in Eq. 4.6,

$$\begin{aligned} error_j &= \sum_{i \in P_j} \int_{-\infty}^{+\infty} \mathbf{N}_{ij}(x) \cdot \mathbf{F}_{ij}(s_i - s_j, x) dx \quad , \\ \mathbf{F}_{ij}(s_i - s_j, x) &= Pr\{s_i - s_j > T_{cp} - T_{setup} - x\} \quad , \end{aligned} \tag{4.7}$$

where $\mathbf{N}_{ij}(x)$ is the probability function with regard to sensitized path delay x.

Based on Eq. 4.3 and Eq. 4.6, we can formulate the CSS problem in timing-speculative circuits as an optimization problem that aims at minimizing the fol-

lowing objective:

$$error(\vec{s}) = 1 - \prod_{j=1}^{|V|}(1 - \sum_{i \in P_j}\sum_{k} N_{ijk} \cdot F_{ijk}(s_i - s_j)) \quad . \tag{4.8}$$

The optimization targeting this objective is expected to reduce the error cycle rate of overall system so that it can improve circuit throughput.

Finally, we develop the above general CSS problem to an MDCSS one wherein clock skews are contrained to only a number of discrete values. This MDCSS problem in timing-speculative circuits can be formulated as follows:

Problem: Given

- A timing-speculative circuit, equipped with timing speculators, such as *Razor* [9];
- The lower bound and upper bound of skew range, s_{lower} and s_{upper};
- The skew domain number, K;
- The penalty to recover from a timing error, *penalty*;

to determine: (i) the skew set $\mathcal{C} = \{c_1, c_2, \cdots, c_K\}$ of K skew domains, and (ii) how to assign each FF to one of these domains, such that the overall error rate is minimized, i.e., the performance is maximized.

4.4 GDM-Based Skew Scheduling Algorithm

In this work, we first develop a novel GDM-based algorithm to solve the general CSS problem in timing-speculative circuits, serving as the first phase of the proposed CSS technique. The simplest version of GDM (refer to [35] for details) is usually formulated as the following unconstrained optimization problem:

$$\min_{\{\vec{s}\}} f(\vec{s}), \vec{s} \in \mathbb{R}^n \quad , \tag{4.9}$$

where $f(\vec{s})$ is a scalar objective function, $\mathbb{R}^n$ is the n-dimensional real Euclidean space, and $\vec{s}$ is a vector of n real components, $\{s_i | 1 \leq i \leq n\}$. It is assumed that $f(\vec{s})$ is differentiable so that the gradient vector $\nabla f(\vec{s})$ exists everywhere in $\mathbb{R}^n$. The solution of above problem is denoted as $\vec{s}^*$.

GDM is a first-order optimization algorithm that uses the gradient vector $\nabla f(\vec{x})$ to determine the search direction for each iteration. The simplest and most famous GDM algorithm is the *method of steepest descent*, which takes steps proportional to the negative/positive of the gradient (or, the approximate gradient) of the function at current iteration to minimize/maximize $f(\vec{s})$.

The targeted CSS problem with timing speculation in this work is how to determine clock skew setting and hence allocate timing slack as resource to the most critical paths to reduce error cycle rate of overall system and improve system throughput. This is strongly relevant to both sensitization probability and expected error probability for each timing critical/sub-critical path. From this viewpoint, GDM is just applicable to our targeted problem. How to apply GDM in targeted CSS problem will be detailed in the following.

4.4.1 Proposed Optimization Metric

From Eq. 4.2, we can see that the system error cycle rate must be reduced and kept within a small number to achieve a higher throughput, otherwise the penalty caused by error occurrence would greatly reduce the benefit of clock period decrease and even worsen the throughput.

Based on this observation, we can simplify the original optimization objective described in Eq. 4.8 by assuming $error_j(\vec{s})$, the error probability for each FF, is so small that there would not be more than one error occurring at the same time. Therefore, after expanding Eq. 4.3 we can ignore the high order terms and obtain

the following approximation,

$$error(\vec{s}) = 1 - \prod_{j=1}^{|V|}(1 - error_j(\vec{s})) \approx \sum_{j=1}^{|V|} error_j(\vec{s}) \quad . \tag{4.10}$$

This simplified optimization objective is used in our GDM-based skew scheduling algorithm as a metric to guide the reduction of system error cycle rate. After substituting Eq. 4.6 into Eq. 4.10, the proposed optimization metric is written as

$$error(\vec{s}) = \sum_{j=1}^{|V|} \sum_{i \in P_j} \sum_{k} N_{ijk} \cdot F_{ijk}(s_i - s_j) \quad . \tag{4.11}$$

4.4.2 Skew Constraint

In practice, the assigned skew values should be constrained in a specified range with a lower bound and an upper bound (e.g., $[s_{lower}, s_{upper}]$). This constraint requirement is beyond the basic unconstrained formulation of GDM in Eq. 4.9. In order to tackle this problem, we can use such a function,

$$S(x_i) = s_{lower} + (s_{upper} - s_{lower}) \cdot (1 - \frac{1}{1 + e^{\mu x_i}}) \quad , \tag{4.12}$$

to constrain the assigned skew values within specified range $[s_{lower}, s_{upper}]$ all the time. Here, μ is a parameter to control the function shape. As a result, by substituting Eq. 4.12 into Eq. 4.11, we obtain the optimization objective written as

$$error(\vec{x}) = \sum_{j=1}^{|V|} \sum_{i \in P_j} \sum_{k} N_{ijk} \cdot F_{ijk}(S(x_i) - S(x_j)) \quad , \tag{4.13}$$

where $\vec{x}$ is the new vector variable. Obviously, once $\vec{x}^*$ is determined, the skew value $\vec{s}^*$ can be easily calculated according to Eq. 4.12. The transformation from a constrained problem to an unconstrained version makes GDM applicable to targeted CSS problem.

4.4.3 Proposed Skew Scheduling Algorithm

After approximating optimization metric in Section 4.4.1 and applying the transformation in Section 4.4.2, it is now ready for GDM to determine the clock skew setting.

We compute the gradient of objective function. For $\forall\ \ell \in \{1,2,\cdots,n\}$,

$$\begin{aligned}\frac{\partial error(\vec{x})}{\partial S(x_\ell)} = &-\sum_{i\in P_\ell}\sum_{k} N_{i\ell k}\cdot F'_{i\ell k}(S(x_i)-S(x_\ell)) \\ &+\sum_{\ell\in P_j}\sum_{k} N_{\ell jk}\cdot F'_{\ell jk}(S(x_\ell)-S(x_j)) \quad ,\end{aligned} \tag{4.14}$$

$$\frac{dS(x_\ell)}{dx_\ell} = (s_{upper}-s_{lower})\frac{\mu e^{\mu x_\ell}}{(1+e^{\mu x_\ell})^2} \quad , \tag{4.15}$$

where $F'_{ijk}(\cdot)$ is the probability density function. Based on Eq. 4.14 and Eq. 4.15, we can finally get the gradient according to

$$\frac{\partial error(\vec{x})}{\partial x_\ell} = \frac{\partial error(\vec{x})}{\partial S(x_\ell)}\cdot\frac{dS(x_\ell)}{dx_\ell} \quad . \tag{4.16}$$

Since we would like to minimize the error cycle rate, we update the parameters using the negative of the computed gradient at each iteration as below,

$$x_\ell^{new} = x_\ell - \eta\frac{\partial error(\vec{x})}{\partial x_\ell} \quad , \tag{4.17}$$

where η is the learning rate that determines step size towards optimized solution point and maintains a balance between convergence speed and avoiding divergence. How to determine effective learning rate, in fact, has been studied by many machine learning works. As this problem is not the focus of this work, in our implementation we simply set η to be a typical value 0.05. By updating the parameter $\vec{x}$ periodically until the procedure converges, we can obtain the skew setting $\vec{s}^* = S(\vec{x}^*)$ according to the definition in Eq. 4.12.

The algorithm flow of the first phase for general CSS problem is described in Fig. 4.3. First of all, we initialize the variables $\vec{x}$ as same as the case with

#	$f(\vec{x})$, the objective function for optimization
#	ε_g and $\varepsilon_{\vec{x}}$, convergence tolerances
1.	Initialize $\vec{x}$
2.	**REPEAT** for each iteration
3.	**IF** $\parallel \nabla f(\vec{x}) \parallel < \varepsilon_g$
4.	Set $\vec{x}^* = \vec{x}$
5.	Break
6.	**ELSE**
7.	Set $\vec{g} = -\nabla f(\vec{x})$
8.	**FOR** the iteration i from 1 to $\|V\|$
9.	Update parameters $x_i^{new} = x_i - \eta \cdot g_i$
10.	**IF** $\parallel \vec{x}^{new} - \vec{x} \parallel < \varepsilon_{\vec{x}}$
11.	Set $\vec{x}^* = \vec{x}^{new}$
12.	Break
13.	**ELSE**
14.	Go for next iteration

Figure 4.3: The proposed GDM-based skew scheduling algorithm.

zero-skews. Then, we repeat the procedure (see Line 3 $\sim$ 14) to compute the gradient and update parameters until the convergence criterion is satisfied. To be specific, we have two convergence criterions: (i) the gradient tolerance ε_g (see Line 3 $\sim$ 5) to determine whether the algorithm has arrived at a critical point, and (ii) the step tolerance $\varepsilon_{\vec{x}}$ (see Line 10 $\sim$ 12) to determine whether significant progress is achieved. Once either of them is satisfied, the optimization process is terminated; otherwise, it continues to compute the gradient (See Line 7) and update parameters (see Line 9).

4.5 Multi-Domain Skew Scheduling Algorithm

In the second phase of our proposed CSS technique, we tackle the MDCSS problem using the output of the first phase as initial solution. This MDCSS algorithm is based on SDM, including two stages: (i) initial stage: based on the result given by our GDM-based CSS algorithm, we resort to K-means clustering [79] to achieve the skew set $\mathcal{S}$ and also the initial skew scheduling $\vec{s}_0$; and (ii) refinement stage: using $\vec{s}_0$ as the initial solution, we employ search-based algorithm to explore better solution in the discrete solution space.

4.5.1 Initial Stage

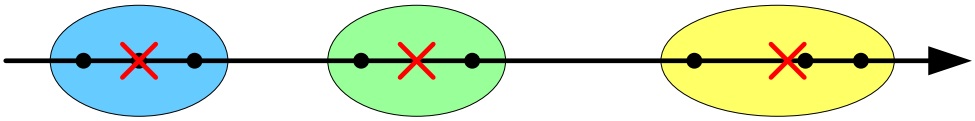


Figure 4.4: An example of grouping 8 one-dimensional sample points into 3 clusters by K-means clustering.

K-means clustering [79] is a method of clustering analysis that groups some given sample points into K cohesive clusters. As shown in Fig. 4.4, the 8 one-dimensional sample points represented by black spots are partitioned into 3 clusters, each of which has a red cross representing its center. Formally, given the n sample points $\mathcal{P} = \{p_1, p_2, \cdots, p_n\}$ and an integer K, K-means clustering tries to find K points $\mathcal{C} = \{c_1, c_2, \cdots, c_K\}$ as the centers of K clusters, such that the following objective function is minimized:

$$h(P,K) = \sum_{i=1}^{K} \sum_{p_j \in C_i} ||p_j - c_i||^2 \ , \tag{4.18}$$

where C_i is the set of points which are the closest to the center c_i, and $h(P,K)$ is the sum of the squared distances between all the points to their closest cluster center.

#	$\{s'_i \mid 1 \leq i \leq \|V\|\}$, the set of skews
#	$\{C_j \mid 1 \leq j \leq K\}$, the set of clusters (i.e., skew domains)
#	$\{c_j \mid 1 \leq j \leq K\}$, the set of cluster centers
1.	Initialize $\mathcal{C} = \{c_1, c_2, \cdots, c_K\}$ randomly
2.	**REPEAT** for each iteration until convergence
3.	Initialize $C_j = \varnothing, \forall 1 \leq j \leq K$
4.	**FOR** each skew element s'_i
5.	$j = \arg\min_t \|\|s'_i - c_t\|\|$
6.	Assign the skew s'_i to the cluster C_j
7.	**FOR** each cluster C_j
8.	Compute its center $c_j = (\sum_{p_i \in C_j} p_i)/\|C_j\|$

Figure 4.5: The proposed algorithm based on K-means at intial stage.

Accordingly, given the $|V|$ elements of the skew vector $\vec{s'} = (s'_1, s'_2, \cdots, s'_{|V|})$ generated by our GDM-based algorithm in continuous space as the sample points and the skew domain number K, we would like to find K skew values of skew domains that representatively describe the $|V|$ skew values and assign each sample point to a certain domain. Consequently, this problem can be naturally solved using K-means, as shown in Fig. 4.5. The cluster centers are used as the skews of K skew domains, and the skew of each FF is assigned as same as the skew of its associated domain.

4.5.2 Refinement Stage

After initial stage, we achieve the discrete skew set $\mathcal{C} = \{c_1, c_2, \cdots, c_K\}$ and an initial skew setting $\vec{s}_0$ in discrete space. To further improve the solution, we formulate the refinement stage of MDCSS problem as a discrete search problem and resort to a heuristic search algorithm based on steepest descent method [36].

4.5.2.1 Solution Representation and Move

An effective solution representation together with the corresponding move is very important for any discrete search algorithm. In refinement stage of the MDCSS problem, the solution representation is naturally described using the skew vector $\vec{s}$, wherein each element (e.g., s_i) is the skew value of a FF defined in the discrete skew set $\mathcal{C}$. As for the move, we simply define it as the skew change of a certain FF. When moving to a neighbor solution, we start with the original solution and choose a FF to change its skew value. This definition of move can guarantee the completeness of traversing the entire solution space.

4.5.2.2 Objective Function

To evaluate solutions during search process, we can simply use the metric defined in Eq. 4.11 as the objective function. However, we have such an observation that the change of the skew of a certain FF affects only a small part of the metric calculation. To clarify, let us consider the example demonstrated in Fig. 4.6. If the skew of FF_3 is changed, only the timing slacks between FF_3 and its neighbors (i.e., FF_1, FF_2 and FF_5) would be affected, consequently we only have to update the metric calculation related to these FFs.

Formally, if the skew of FF_ℓ is changed from s_ℓ to s'_ℓ, the metric difference between these two solutions should be:

$$\begin{aligned}\Delta = &\sum_{i\in P_\ell}\sum_k N_{i\ell k}\cdot(F_{i\ell k}(s_i - s'_\ell) - F_{i\ell k}(s_i - s_\ell)) + \\ &\sum_{\ell\in P_j}\sum_k N_{\ell jk}\cdot(F_{\ell jk}(s'_\ell - s_j) - F_{\ell jk}(s_\ell - s_j)) \ ,\end{aligned} \tag{4.19}$$

By calculating the metric of initial solution and extracting metric difference between two neighbor solutions according to Eq. 4.19, we can achieve the metric of each solution with much less computations, compared to computing Eq. 4.11 directly.

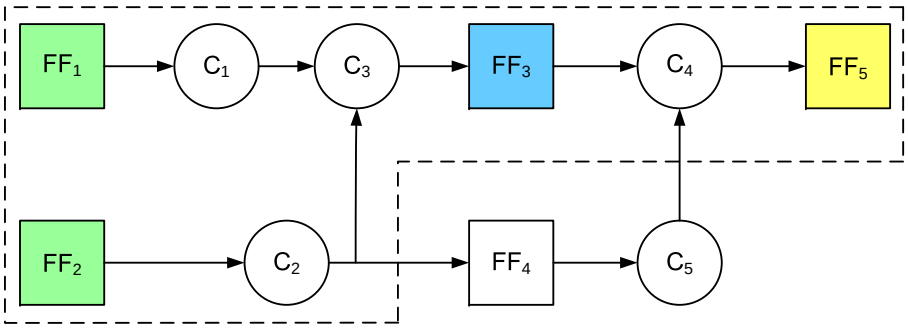


Figure 4.6: The change of the skew of a certain FF impacts only a small part of the metric calculation.

#	$\vec{s}_0$, initial solution generated by initial stage
#	$\vec{s}_c$ and $\vec{s}_n$, current solution and next solution
1.	Initialize skew setting $\vec{s}_c = \vec{s}_0$
2.	**REPEAT** for each iteration until convergence
3.	Initialize $\vec{s}_n = \vec{s}_c$
4.	**FOR** each neighbor solution $\vec{s}_i$ of $\vec{s}_c$
5.	**IF** $metric(\vec{s}_i) < metric(\vec{s}_n)$
6.	$\vec{s}_n = \vec{s}_i$
7.	**IF** $\vec{s}_n == \vec{s}_c$ // **No better solution found**
8.	Break
9.	**ELSE**
10.	$\vec{s}_c = \vec{s}_n$
11.	Go for next iteration

Figure 4.7: The proposed search algorithm based on SDM at refinement stage.

4.5.2.3 Heuristic Search Algorithm

With the above definitions of solution representation, move, and objective function, the refinement stage of MDCSS problem can be naturally solved by search algorithms (e.g., random search and simulated annealing) [36]. In this work, we

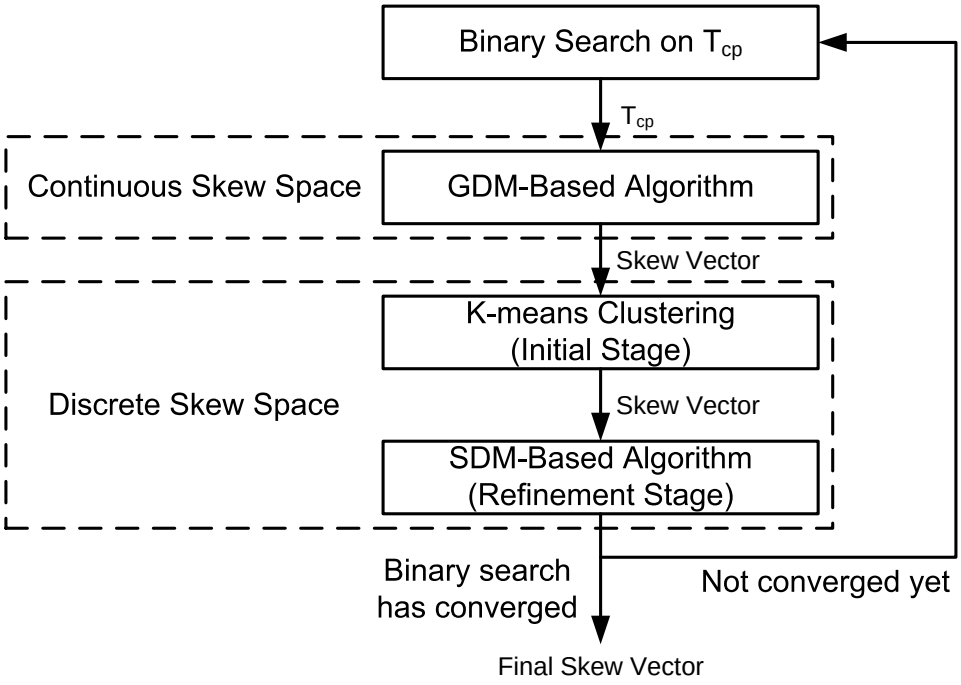


Figure 4.8: The overall flow of MDCSS algorithm.

simply employ steepest descent method (SDM), a discrete analogue of GDM in a space involving discrete components, since it is typically able to converge in a few steps.

Basically, an SDM search starts at an initial state and takes search steps in solution space, reducing a given objective function with the maximum rate of descent. Instead of computing a gradient in GDM, the best move of SDM is determined using a local minimization. As demonstrated in Fig. 4.7, after initialization, the search optimization repeats iteratively (see Line 3 $\sim$ 11). For each iteration, we find out the move leading to the neighbor solution with smallest metric (see Line 4 $\sim$ 6). The search process continues until convergence or a termination condition is reached.

4.5.3 Overall Flow of MDCSS Algorithm

The overall flow of our proposed MDCSS algorithm is depicted in Fig. 4.8. First of all, we perform binary search on the operational clock period T_{cp}. Then, for each selected T_{cp}, we use GDM-based algorithm to achieve a vector of skew setting in continuous skew space. K-means clustering is employed to group the elements of such a skew vector into clock skew domains and provide the initial solution. Finally, SDM-based algorithm is utilized to explore the discrete skew space and further improve the solution.

4.6 Experimental Results

4.6.1 Experimental Setup

To evaluate the effectiveness of the proposed CSS method, we conduct experiments on several large ISCAS'89, IWLS'05 and ITC'99 benchmarks. We synthesize these circuits and obtain timing information using Synopsys EDA tools. To take process variation effect into consideration, we perform Monte Carlo simulation to inject gate-level delay variation following Gaussian distribution with standard deviation equal to 8%. We conduct simulation with random inputs in our experiments and the simulation is performed with 100,000 cycles. All the experiments are conducted on a *2.8GHz* PC with *4GB* RAM.

For reasonable comparison, we provide a reference via offline timing analysis with false paths excluded according to [61], denoted as $CSS_{nonrazor}$. We apply timing speculation directly without CSS and denote this solution as CSS_{noncss}. That means, we assign all the FFs with *zero*-skew in CSS_{noncss}. A yield-driven CSS work [69] is used as the baseline solution for comparison and denoted as $CSS_{baseline}$. Our proposed CSS based on GDM is denoted as CSS_{gdm}, CSS with K-means clustering at initial stage is denoted as CSS_{kmeans} and CSS based on SDM

at refinement stage is denoted as CSS_{sdm}. All these solutions mentioned above with timing speculation select the operational clock period of circuits according to Eq. 4.2 to minimize the equivalent clock period T_{ecp} which considers the penalty for error occurring. Note that, for fair comparison, in $CSS_{baseline}$ we apply the algorithm in [69] to determine skew setting and then find out the optimal clock period since timing speculation capability is equipped. By doing this, the skew setting output by [69] would not be impacted by timing speculation, and its effectiveness is actually improved due to the tradeoff between performance and error rate. The comparison between $CSS_{baseline}$ and CSS_{gdm} can reflect the benefit of explicitly combining CSS with timing speculation, since CSS_{gdm} consider timing speculation capability in the optimization procedure of CSS technique explicitly.

Usually, which FF to be equipped as Razor-FF lies on its timing pressure. In the case without CSS, a simple scheme is to equip all the FFs that serve as the ends of critical paths as Razor-FFs. If the delay of a path is larger than β of clock period (e.g., $\beta = 80\%$), such a path is critical path. As for the case with CSS, because pre-silicon CSS discussed in this chapter is performed offline at design stage and its effect is just to manipulate timing slacks between FFs, we can simply set up those FFs, whose timing slacks are less than $(1-\beta)$ of clock period after conducting CSS, as Razor-FFs. In our experiments, the timing threshold β is set to be 80%, and the hardware cost for equipping each Razor-FF is assumed to be 10 gates. The *penalty* in Eq. 4.2 is assumed to be 10 clock cycles according to [12], and we sweep the operational clock period to select the one with best performance.

4.6.2 Results on GDM-Based Algorithm

In this subsection, we discuss the results on the GDM-based CSS algorithm (see Section 4.4).

We report the results on hardware cost and algorithm runtime of the proposed

Bench.	TG#	TFF#	CSS_{noncss}		$CSS_{baseline}$		CSS_{gdm}		RT_1 (s)
			$RFF\#$	Cost (%)	$RFF\#$	Cost (%)	$RFF\#$	Cost (%)	
s1494	680	6	2	2.94	2	2.94	3	4.41	0.17
s5378	3042	179	22	7.23	18	5.92	32	10.52	1.20
s9234	5866	228	22	3.75	25	4.26	18	3.07	1.15
s13207	8803	638	10	1.14	10	1.14	10	1.14	2.06
s15850	10470	597	89	8.50	89	8.50	81	7.74	3.33
s35932	18148	1728	144	7.93	144	7.93	144	7.93	13.76
s38584	21021	1426	168	6.90	180	7.39	182	7.48	16.30
s38417	24341	1564	87	4.14	106	5.23	108	5.14	10.29
wb_conmax	73233	3316	657	8.97	699	9.54	698	9.53	23.93
des_perf	154204	9105	668	4.33	887	5.75	870	5.64	46.83
ethernet	157841	10752	1553	9.84	1602	10.15	1584	10.04	34.87
b17	32326	1415	363	10.10	349	9.74	356	9.92	6.10
b18	114643	3320	653	5.39	676	5.57	678	5.58	25.00
b19	231364	6642	1383	5.64	1406	5.73	1398	5.70	31.54
b20	20226	490	192	8.67	189	8.55	193	8.71	13.59
b21	20571	490	170	7.63	209	9.22	192	8.54	13.81
b22	29951	735	198	6.20	215	6.70	207	6.46	17.10
AVERAGE				6.43		6.72		6.91	

TG#: total gate count; TFF#: total FF count; RFF#: Razor-FF count;

Cost: hardware cost ratio for equipping Razor-FFs;

RT_1: runtime of the proposed GDM-based algorithm.

Table 4.1: Experimental results on hardware cost and algorithm runtime with continuous skew space.

CSS technique CSS_{gdm} in Table 4.1. As can be seen, the average hardware cost for CSS_{noncss}, $CSS_{baseline}$ and CSS_{gdm} to enable timing speculation are 6.43%, 6.72% and 6.91%, respectively. Although the hardware cost after conducting CSS seems a little higher than CSS_{noncss}, the additional cost is still within an acceptable range. The runtime of CSS_{gdm} is listed in Column 10 of Table 4.1.

Next, we perform Monte Carlo simulation to produce 100 sample chips with

Bench.	$CSS_{nonrazor}$	CSS_{noncss}			$CSS_{baseline}$			CSS_{gdm}				
	$T_{ecp}(ns)$	$T_{ecp}(ns)$	$\sigma(ns)$	$\Delta_1(\%)$	$T_{ecp}(ns)$	$\sigma(ns)$	$\Delta_2(\%)$	$T_{ecp}(ns)$	$\sigma(ns)$	$\Delta_3(\%)$	$\Delta_4(\%)$	$\Delta_5(\%)$
s1494	2.72	2.38	0.068	-12.63	2.29	0.067	-3.73	2.19	0.070	-4.32	-7.89	-19.52
s5378	1.86	1.58	0.017	-15.21	1.36	0.024	-13.45	1.35	0.024	-1.13	-14.43	-27.44
s9234	4.58	4.05	0.086	-11.61	3.84	0.068	-5.16	3.39	0.064	-11.65	-16.21	-25.94
s13207	5.57	4.12	0.080	-25.97	4.09	0.081	-0.72	4.05	0.048	-1.07	-1.78	-27.29
s15850	6.22	5.28	0.035	-15.12	4.75	0.046	-9.95	4.55	0.063	-4.38	-13.89	-26.91
s35932	3.68	3.57	0.07	-2.99	3.53	0.071	-1.03	3.49	0.073	-1.13	-2.15	-5.07
s38584	6.96	6.39	0.145	-8.10	6.21	0.129	-2.87	6.07	0.106	-2.29	-5.09	-12.78
s38417	6.12	5.73	0.065	-6.48	5.17	0.062	-9.76	4.57	0.042	-11.56	-20.20	-25.36
wb_conmax	6.43	5.97	0.078	-7.20	5.53	0.063	-7.28	4.91	0.107	-11.21	-17.68	-23.60
des_perf	13.70	13.44	0.340	-1.90	12.23	0.309	-9.00	10.88	0.275	-10.99	-19.00	-20.54
ethernet	11.28	10.53	0.055	-6.65	10.28	0.05	-2.35	9.96	0.057	-3.11	-5.39	-11.68
b17	20.03	18.16	0.093	-9.33	16.83	0.092	-7.32	14.09	0.089	-16.28	-22.41	-29.65
b18	26.56	24.65	0.135	-7.19	23.64	0.133	-4.11	22.69	0.136	-4.02	-7.96	-14.57
b19	57.29	52.42	0.287	-8.49	49.64	0.242	-5.31	46.91	0.263	-5.49	-10.51	-18.11
b20	15.02	12.94	0.112	-13.83	12.25	0.115	-5.39	10.88	0.114	-11.13	-15.92	-27.55
b21	14.98	12.86	0.108	-14.14	11.98	0.106	-6.89	10.75	0.110	-10.26	-16.44	-28.25
b22	17.46	15.07	0.116	-13.71	14.63	0.114	-2.92	12.83	0.112	-12.26	-14.82	-26.50
AVERAGE				-10.62			-5.72			-7.19	-12.46	-21.81

T_{ecp}: mean equivalent clock period considering error penalty;

σ: standard deviation of equivalent clock period T_{ecp};

Δ_1: T_{ecp} difference ratio between CSS_{noncss} and $CSS_{nonrazor}$;

Δ_2: T_{ecp} difference ratio between $CSS_{baseline}$ and CSS_{noncss};

Δ_3: T_{ecp} difference ratio between CSS_{gdm} and $CSS_{baseline}$;

Δ_4: T_{ecp} difference ratio between CSS_{gdm} and CSS_{noncss};

Δ_5: T_{ecp} difference ratio between CSS_{gdm} and $CSS_{nonrazor}$.

Table 4.2: Experimental results on equivalent clock period T_{ecp} with continuous skew space.

different variation patterns for each benchmark. In Table 4.2, we report the equivalent clock period T_{ecp} and its standard deviation σ for CSS_{noncss}, $CSS_{baseline}$, and CSS_{gdm}, respectively. This clock period T_{ecp} has taken both the selected opera-

tional clock period and the penalty for error occurring into account. As can be seen, CSS_{noncss} can achieve 10.62% reduction in T_{ecp} in average when compared with $CSS_{nonrazor}$. This improvement reflects the efficacy of Razor technique itself without CSS, which is not the main contribution of this work. After applying $CSS_{baseline}$, a yield-driven CSS proposed in [69], another 5.72% improvement can be achieved in average, which shows that conducting CSS is beneficial for timing speculation. Compared with $CSS_{baseline}$, CSS_{gdm} can obtain 7.19% further improvement. This reflects the efficacy to consider timing speculation in CSS optimization procedure explicitly. The improvement for CSS_{gdm} over CSS_{noncss} and $CSS_{nonrazor}$ are demonstrated in Column 12-13 of Table 4.2.

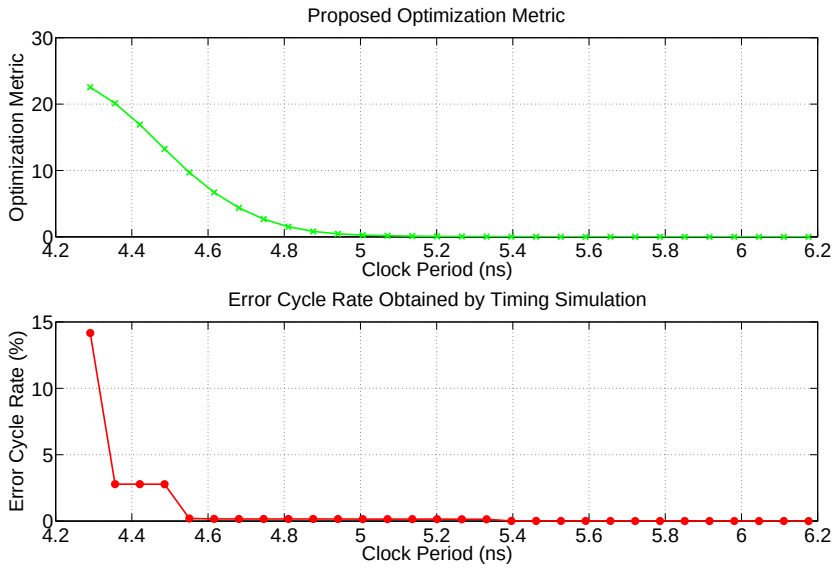


Figure 4.9: Experimental results of CSS_{gdm} with respect to the operational clock period.

To get more details about CSS_{gdm}, we take $s38417$, the largest circuit in ISCAS'89, as an example in the following. In Fig. 4.9, we show the trends of both the proposed optimization metric in Eq. 4.11 and the error cycle rate obtained by

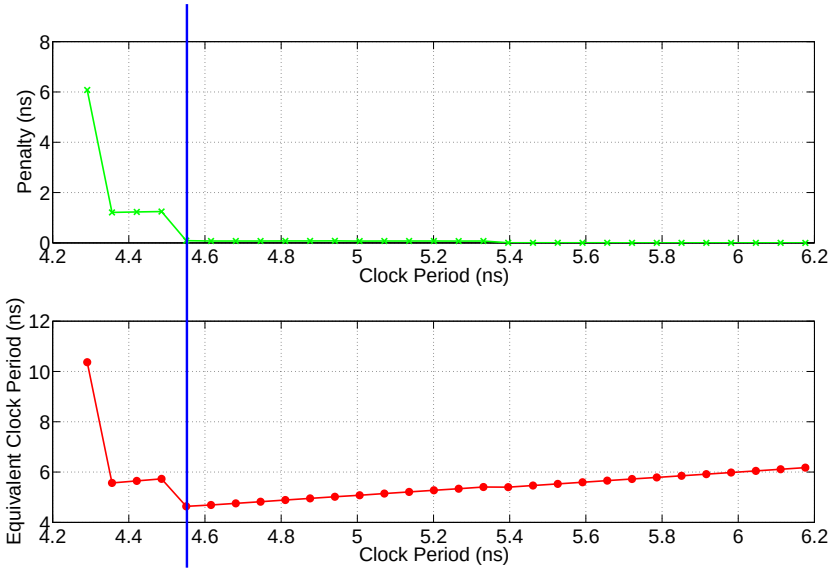


Figure 4.10: Performance penalty and equivalent clock period with respect to the operational clock period.

timing simulation with respect to the operational clock period. We also present the performance penalty due to timing errors and the equivalent clock period considering the penalty in Fig. 4.10. It can be found that with the scaling down of clock period, the penalty increases slightly at first, therefore the equivalent clock period decreases gradually. However, when the clock period scales down to a point (indicated by the vertical line in Fig. 4.10), the penalty starts to increase significantly, leading to the minimum point of the equivalent clock period. In such manner, CSS_{gdm} selects the operational clock period with a reasonable error cycle rate to minimize the equivalent clock period (i.e., maximize throughput). In Fig. 4.11, the trends of the proposed optimization metric and the error cycle rate with respect to GDM iteration number under the optimal operational clock period are plotted out. This figure shows how CSS_{gdm} optimizes the proposed metric and hence reduces the error cycle rate at the same time. The similar trends of these two curves

in Fig. 4.11 prove the effectiveness of our proposed optimization algorithm and metric.

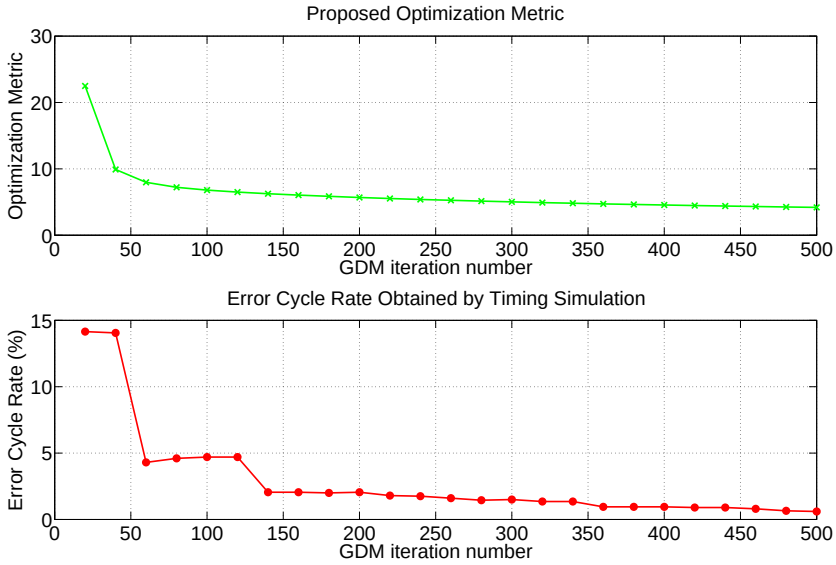


Figure 4.11: Experimental results of CSS_{gdb} with respect to the GDM iterations.

4.6.3 Results on MDCSS Algorithm

In this subsection, we discuss the results on the proposed MDCSS algorithm (see Section 4.5). In the experiments, the domain number is set to be 3.

We report the experimental results of MDCSS algorithm in Table 4.3. The hardware costs of CSS_{kmeans} and CSS_{sdm} are about 6.8%, almost the same with that of CSS_{gdm}. Compared to CSS_{gdm}, CSS_{kmeans} at initial stage suffers from 2.45% performance degradation. This is because CSS_{kmeans} clusters the skews generated by CSS_{gdm} into only a small number of skew domains, and assigns clock skews in a coarse-grain manner. Fortunately, by using CSS_{sdm} at refinement stage, we achieve 2.22% reduction in equivalent clock period, when compared to CSS_{kmeans}.

Bench.	CSS_{kmeans}				CSS_{sdm}					RT_2 (s)	RT_3 (s)
	$RFF\#$	Cost (%)	$T_{ecp}(ns)$	$\Delta_1(\%)$	$RFF\#$	Cost (%)	$T_{ecp}(ns)$	$\Delta_2(\%)$	$\Delta_3(\%)$		
s1494	2	2.94	2.20	0.26	3	4.41	2.2	0	0.26	0.01	2.03
s5378	30	9.86	1.36	0.52	27	8.88	1.35	-0.52	0	0.01	7.25
s9234	17	2.90	3.42	0.94	14	2.39	3.32	-2.88	-1.97	0.02	9.98
s13207	10	1.14	4.07	0.49	10	1.14	4.01	-1.48	-0.99	0.04	23.14
s15850	83	7.93	4.80	5.39	83	7.93	4.69	-2.29	2.97	0.52	53.91
s35932	139	7.66	3.55	1.74	135	7.44	3.51	-1.13	0.59	1.39	136.65
s38584	173	8.23	6.19	2.06	185	8.80	5.88	-5.05	-3.10	1.94	125.92
s38417	114	4.68	4.87	6.65	111	4.56	4.65	-4.60	1.75	1.86	144.53
wb_conmax	712	9.72	5.42	10.4	701	9.57	5.38	-0.83	9.48	2.09	234.18
des_perf	875	5.67	11.04	1.43	869	5.64	10.88	-1.41	0	2.09	438.12
ethernet	1629	10.32	10.30	3.38	1596	10.11	9.43	-8.45	-5.35	2.37	384.12
b17	361	10.05	14.24	1.03	359	10.00	14.09	-1.02	0	1.13	89.33
b18	664	5.47	23.14	1.97	675	5.56	22.79	-1.51	0.43	1.84	261.37
b19	1385	5.65	47.67	1.61	1389	5.66	46.96	-1.48	0.11	2.58	523.13
b20	198	8.92	11.01	1.19	192	8.67	10.83	-1.65	-0.48	1.16	97.36
b21	185	8.25	10.88	1.25	179	8.01	10.72	-1.54	-0.31	1.18	113.66
b22	203	6.35	13.01	1.37	198	6.20	12.76	-1.95	-0.61	1.22	108.73
AVERAGE		6.81		2.45		6.76		-2.22	0.16		

RFF#: Razor-FF count; Cost: hardware cost ratio for equipping Razor-FFs;

T_{ecp}: equivalent clock period considering error penalty;

Δ_1: T_{ecp} difference ratio between CSS_{kmeans} and CSS_{gdm};

Δ_2: T_{ecp} difference ratio between CSS_{sdm} and CSS_{kmeans};

Δ_3: T_{ecp} difference ratio between CSS_{sdm} and CSS_{gdm};

RT_2: runtime of the proposed MDCSS algorithm including both initial and refinement stages;

RT_3: total runtime of the overall design flow shown in Fig. 4.8.

Table 4.3: Experimental results of MDCSS algorithm with discrete skew space.

This is the benefit of our SDM-based search algorithm by exploring better solution in the discrete skew space. Taking this improvement into account, the overall performance degradation due to the constrained discrete skew space is reduced to only 0.16%. The runtime of our MDCSS algorithm including both initial and

refinement stages, as shown in Table 4.3, is very low. The total runtime of the overall design flow that is shown in Fig. 4.8 is also presented.

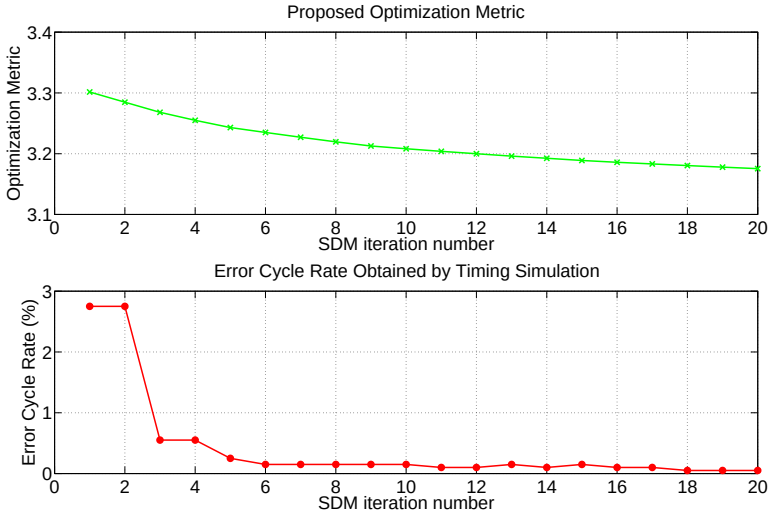


Figure 4.12: Experimental results of CSS_{sdm} with respect to the SDM iterations.

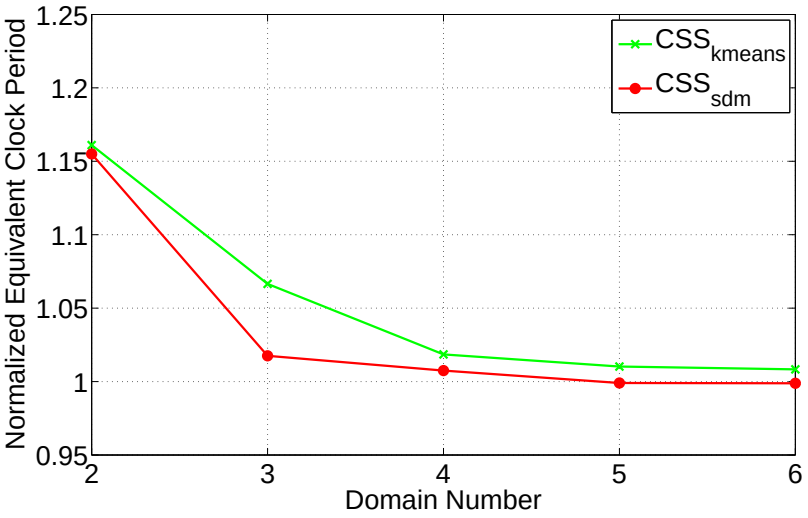


Figure 4.13: Experimental results with respect to the allowed domain number.

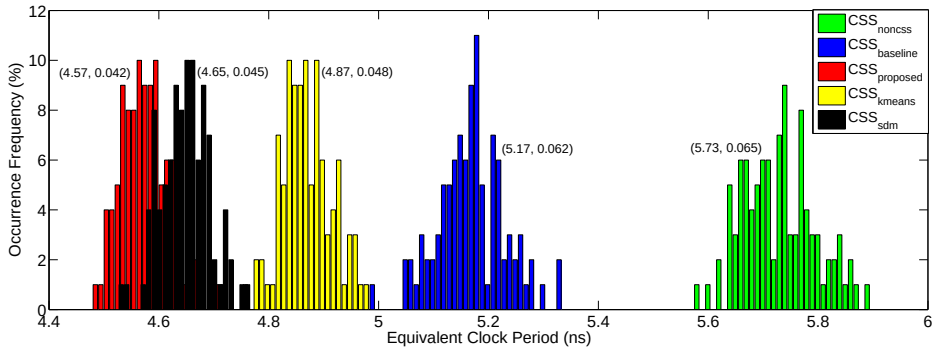


Figure 4.14: Experimental results to show variation effects.

Finally, we use *s*38417 as an example to demonstrate the details of our proposed MDCSS algorithm. In Fig. 4.12, we show the trends of both the proposed optimization metric and the error cycle rate when conducting CSS_{sdm}. As can be seen, CSS_{sdm} can effectively reduce the proposed metric that is used as optimization objective, and hence decrease the error rate iteratively. Besides, we vary the allowed domain number and plot out the equivalent clock periods normalized to that of CSS_{gdm} in Fig. 4.13. We find that, with the increase of domain number the clock periods of CSS_{kmeans} and CSS_{sdm} both decrease significantly at first and then keep almost constant. This is because, the design flexibility is limited in the case of small domain number. When domain number is large (e.g., 4 domains in *s*38417), the proposed methodology only further improves performance slightly with respect to the increase of domain number. In addition, in Fig. 4.14, we show the results with variation effects after performing Monte Carlo simulation. The corresponding mean of T_{ecp} and standard deviation σ are depicted in the form of (T_{ecp}, σ).

4.7 Conclusion

Clock skew scheduling has been exploited as an effective technique to improve timing performance and yield of ICs and attracted lots of research interest. However, with the ever-increasing process variations, a large timing guard band has to be reserved to tolerate variations and guarantee "always correct" operations. In this work, we formulate the CSS problem for timing-speculative circuits, and propose a GDM-based skew scheduling algorithm and a MDCSS algorithm to determine skew setting effectively. Experimental results on benchmark circuits demonstrate that the proposed CSS technique is able to significantly enhance circuit performance.

□ End of chapter.

Chapter 5

Online Clock Skew Tuning

5.1 Introduction

Clock skew optimization (CSO) [63], which treats clock skew as a manageable resource instead of design liability, has been exploited as an effective technique to improve the timing performance of integrated circuits (ICs), by assigning intentional clock arrival times to flip-flops (FFs) in synchronized sequential circuits. Earlier works in this domain [63–67] try to find a good clock schedule that maximizes the timing slack of all paths. Recently, with the introduction of tunable clock tree to combat process variation [80], researchers have also presented various post-silicon clock skew tuning techniques to improve circuit timing performance [68, 81–86].

All the above works ensure that circuits can always operate correctly, even in the worst case scenario. With the ever-increasing static process variation effects due to manufacturing imperfection and dynamic variation effects such as voltage and temperature fluctuations, however, there is an increasing uncertainty for circuit timing behavior. Consequently, a large guard band needs to be reserved when conducting clock skew optimization, leading to rather limited performance

improvement room for such conservative approaches.

The growing circuit timing uncertainties often manifest themselves as timing errors on speed-paths (i.e., critical or near-critical paths) of the circuit. Instead of embedding a large design guard band to guarantee "always correct" operation, timing speculation technique such as Razor [9] allows infrequent occurrence of timing errors and achieves timing error resilience by employing error detection and correction techniques. This "better than worst-case" design methodology enables the tradeoff between reliability and performance/power and hence can achieve much better energy efficiency when compared to that of traditional "worst-case" design methodology. It has thus received lots of research attention from both academia and industry [4, 5, 7, 9, 12, 13].

CSO and timing speculation techniques naturally complement each other and combining them together is able to achieve much better timing performance [57, 87]. On the one hand, with timing speculation, we do not need to guarantee "always correct" operation during clock skew optimization, which significantly enlarges the improvement room of skew optimization techniques. On the other hand, clock skew tuning can be used to manipulate the occurrence of timing errors in the circuit in such manner that those frequently sensitized paths are with larger timing slack and hence the overall timing error rate of the circuit can be reduced, resulting in better throughput of the circuit.

Motivated by the above, in this chapter, we propose a novel online skew tuning framework for timing-speculative circuits. To be specific, we develop novel hardware architecture to collect online timing error information and use it to guide our proposed clock skew tuning procedure to achieve better timing performance. Experimental results on various benchmark circuits demonstrate the effectiveness of our proposed methodology. Note that, our framework focuses on post-silicon skew tuning, and hence it can easily be combined with prior pre-silicon skew scheduling

works.

The remainder of this chapter is organized as follows. In Section 5.2, we present the preliminaries and motivation of this work. The proposed hardware architecture of online error collection and skew tuning is detailed in Section 5.3. The algorithms for block grouping and skew tuning are presented in Section 5.4 and Section 5.5, respectively. Next, Section 5.6 discusses our experimental results on various benchmark circuits. Finally, Section 5.7 concludes this chapter.

5.2 Preliminaries and Motivation

5.2.1 Pre-Silicon Clock Skew Scheduling

Generally speaking, pre-silicon clock skew scheduling can be classified into two categories by different optimization objectives: performance-driven ones [63, 64] to achieve the highest operational frequency and timing yield-driven ones [65–69, 81] to maximize yield under a particular clock period. Considering the variability of critical path delays, some prior works (e.g., [63, 64]) allocated a safety margin with both upper and lower bounds to each feasible region of clock skews in advance in order to minimize clock period. Intuitively, since it seems more reasonable to assign the skews close to the middle of the feasible region, Kourtev and Friedman [65] solved the targeted problem using quadratic programming to minimize the total least square of skews. Wei *et al.* [67] optimized clock slacks using an incremental slack distribution method to tolerate process variation. Albrecht *et al.* [66] modeled the variations on critical path delays with a single variable and transformed the yield optimization problem to a minimum mean cycle problem to guarantee safety margins. Tsai *et al.* [68] modeled the problem as a minimum cost-to-time ratio problem by introducing variance of path delay distribution into the feasible skew region to consider the statistical difference of critical path delays.

Recently, Wang *et al.* [69] argued that all prior works cannot handle non-Gaussian critical path delays, and hence proposed a formulation of yield-driven clock skew scheduling technique under non-Gaussian variations.

The above works try to assign a good clock schedule at design stage, relying on static timing analysis results and process variation models. These offline estimation/analysis techniques, however, cannot provide very accurate timing information and hence limit the effectiveness of pre-silicon clock skew scheduling solutions.

5.2.2 Post-Silicon Clock Skew Tuning

Recently, post-silicon tuning capability has been introduced to clock tree design to remove unintentional skews and boost timing yield under increasing process variations [82]. A representative example is Intel's dual-core Itanium processor, which places tunable delay buffers (TDBs) in the clock distribution network to cancel clock skew variations [80]. In the second level of clock network of Itanium processors, there exist about 15,000 clock vernier devices (CVDs) for post-silicon skew tuning. These TDBs can be programmed from the test access port (TAP).

To realize post-silicon skew tuning, the design of tunable delay buffers is important and various design schemes have been presented in the literature. In [83], a chain of inverters feeding a multiplexer was used to extract fine-grained timing information. In [84], two kinds of delay circuits, a tapped delay line and a mirror delay line, are used to provide tuning capability. One drawback of the above design is the significant power consumption caused by the continuous switching of inverter chains. To tackle this problem, a so-called programmable delay element [81] using a pair of inverters with a set of capacitive loads in between them is proposed. The loads, consisting of transmission gates and NMOS transistors, can be activated via control signals.

With the help of TDB designs, Takahashi *et al.* [85] proposed a post-silicon clock timing adjustment strategy, but how to program the tunable elements and determine their locations was not addressed. Tsai *et al.* [81] and Nagaraj *et al.* [86] combined a statistical timing driven skew scheduling algorithm with a post-silicon clock tuning scheme.

The above works rely on offline testing to obtain timing information and use it for post-silicon clock tuning. Path delay test generation, however, is an extremely difficult problem and the coverage is usually quite low. More importantly, with technology scaling, the discrepancy between circuits' timing behavior in functional mode and that in structural test mode has dramatically increased [88, 89].

Recently, Lak and Nicolici [90] made use of aging sensors to predict NBTI aging effects and correspondingly tuned clock skews to combat lifetime performance degradation. Although online timing information is provided by aging sensors, it still has to be guaranteed that there are no timing violations to occur. Due to the above, the effectiveness of existing post-silicon clock skew tuning techniques is also limited.

5.2.3 Timing Speculation

Circuit-level timing speculation technique, being able to detect timing errors at online stage, react to the error quickly and recover from it by rolling back to a known-good pre-error state, has become one of the most promising solutions to deal with the ever-increasing static and dynamic variation effects with technology scaling.

Various techniques have been presented for online timing error detection. Without loss of generality, let us consider the representative *Razor* flip-flop [9] to demonstrate how timing error detectors work. A Razor-FF, contains a main flip-flop, a shadow latch and some additional control logic, to detect timing errors.

The main flip-flop latches the output signal at the clock edge with possible timing error, while the shadow latch guarantees to receive the correct value, by latching the signal a fraction of a cycle later. Consequently, when the shadow latch and the main FF values do not agree, the timing error is detected. To make use of timing speculation technique, it is necessary to replace all critical FFs that are driven by speed-paths (i.e., critical or near-critical paths) of the circuit with Razor-FFs (or other timing speculators).

For microprocessors, timing error recovery can be achieved with microarchitectural support [34]. That is, when a timing error is detected, the processor pipeline is flushed and the correct result from the shadow latch is returned back into the pipeline. Then, by replaying instructions (at possibly lower frequency), the processor is able to recover from the timing error [27].

Timing error recovery inevitably incurs some performance loss and extra energy consumption. Therefore, it is essential to reduce timing error rate (TER) to optimize timing-speculative circuits [12]. Various optimization techniques (e.g., [11, 13, 14, 19, 28]) have been presented for timing-speculative circuits in the literature. The key issue in this optimization problem is to reshape the path delay distribution of the circuit so that those frequently-exercised timing paths are optimized with more timing slack while other paths are allowed to have timing errors.

5.2.4 Motivation

Targeting timing-speculative circuits, this work presents a novel online clock skew tuning technique to maximize circuit performance. The proposed technique is motivated by the following observations:

- A specific manufactured circuit has its unique characteristics (e.g., path delay distribution), which is difficult to estimate during design stage or costly to characterize with delay testing techniques accurately. Consequently, a

large design guard band needs to be reserved for conventional clock skew optimization techniques. With timing speculation, however, we do not need to guarantee "always correct" operation, which dramatically increases the flexibility and improvement room of clock skew optimization techniques.

- While existing timing speculation techniques are able to apply dynamic voltage/frequency scaling to achieve better energy-performance tradeoff using timing error rate information, this is conducted at the entire circuit level. With online clock skew tuning capability, we can manipulate timing error rate in a fine-grained manner so that those frequently sensitized paths are with larger timing slack, and hence reduce the overall timing error rate of the circuit.

The above motivates us to design a novel online clock skew tuning framework, as shown in Fig. 5.1, wherein we collect runtime timing error information and use it to guide our clock skew tuning process to achieve better circuit performance, as detailed in the following sections.

5.3 Design for Online Clock Skew Tuning

Timing-speculative circuits can detect and correct infrequent timing errors, but they do not support the collection of timing error information, let alone clock skew tuning. Consequently, we need to add additional circuitries into the design to achieve this objective.

5.3.1 Basic Tuning Block

To monitor and manage system timing behavior as shown in Fig. 5.1, the most aggressive design would be to record the timing errors occurred on each Razor-FF

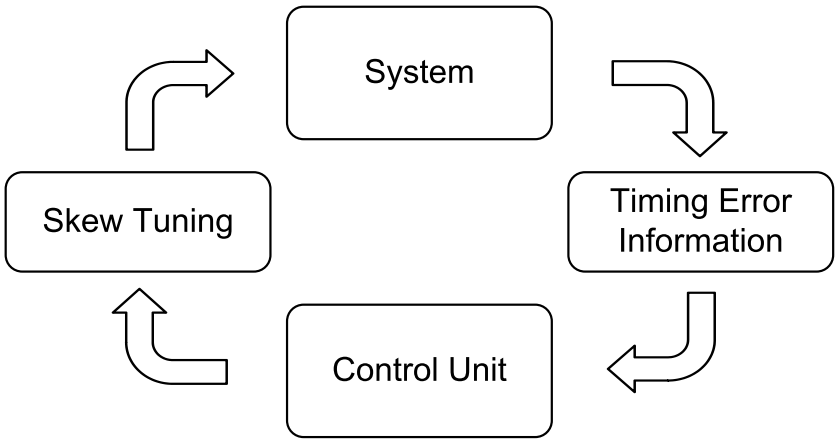


Figure 5.1: The proposed online skew tuning framework.

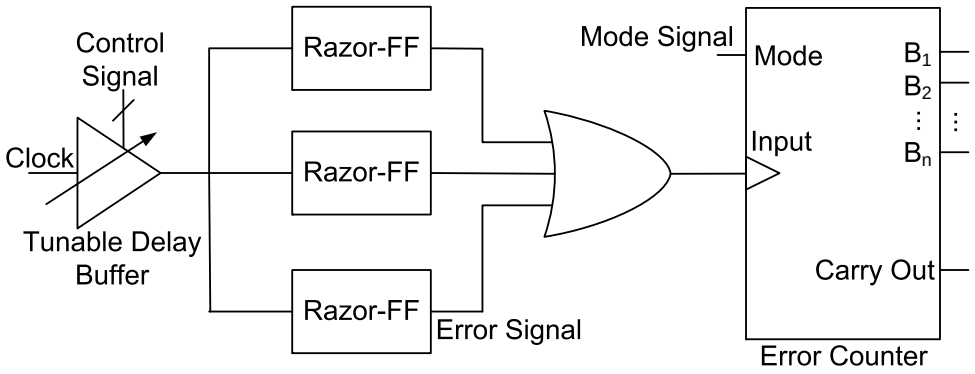


Figure 5.2: Conceptual basic tuning block.

and conduct clock skew tuning individually. Clearly, such design will introduce unaffordable hardware cost, and complicate the clock skew tuning procedure. A more practical approach is hence to group adjacent Razor-FFs together to form a basic tuning block, wherein we collect timing errors occurred in it together and we insert only one tunable delay buffer for each block to apply skew tuning.

The proposed grouping procedure is a bottom-up approach based on the physical layout of the synthesized clock tree structure (detailed in Section 5.4). Within each block, all FFs receive the same clock signal, whose skew is controlled by the

control unit (see Fig. 5.1). Note that, due to the above, clock skew tuning will only affect paths between blocks.

Without loss of generality, we assume that the error signal will be set as 1 once timing error is detected by the corresponding Razor-FF. These error signals are "ORed" together and connected to a counter (see Fig. 5.2). Since the system has to recover from timing error whatever how many errors are encountered in a single cycle, the actual concern during optimization is error cycle rate, rather than timing error number. Besides, the likelihood for multiple Razor-FFs in a block to have timing errors simultaneously is quite low. Therefore, our tuning block design can save area cost with little accuracy loss. The carry-out signal of counter can be used to indicate whether the counter is full.

5.3.2 Timing Error Collection and Clock Skew Tuning Mechanism

One challenging issue in the proposed framework is how to online collect timing error information from all the distributed blocks to the system-level control unit. The most straightforward solution would be to connect the error counter of every block to the control unit. This strategy, however, incurs unaffordable routing cost to the system.

To tackle this problem, we propose a serially shifting mechanism and the proposed architecture is depicted in Fig. 5.3. In this mechanism, the distributed error counters can be reconfigured to work as a shift register by adding some additional control logic. In other words, the error counter has two operational modes: counting mode and shifting mode. In counting mode, the number of timing errors occurred in the corresponding block is accumulated. Whenever an error counter is full (indicated by the carry out signal), the system-level control unit will receive

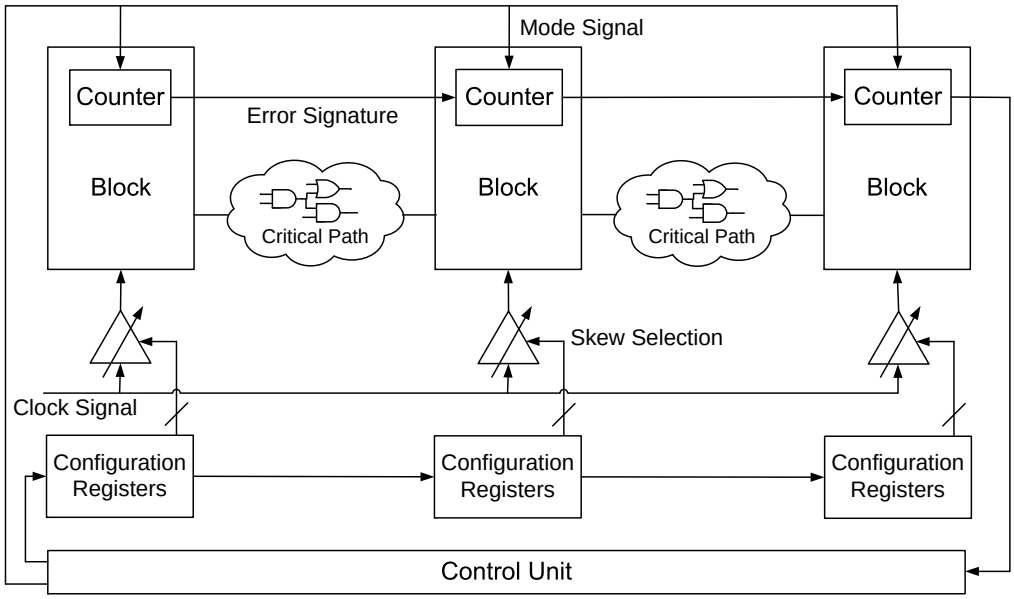


Figure 5.3: Block diagram for timing error collection and clock skew tuning.

a "full" signal from this block[1] and then set all the counters into shifting mode to collect timing errors from all blocks. In shifting mode, the FFs in all counters are serially linked as a shift register and their values are shifted out to control unit without disturbing the normal operation of the system.

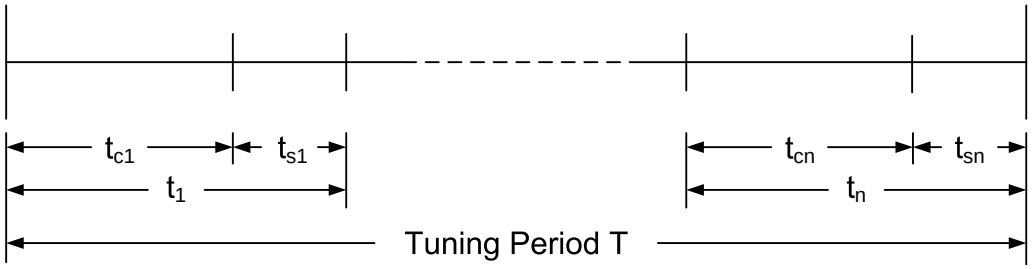


Figure 5.4: Counter mode during tuning period T.

[1]Carry out signals from all distributed blocks are "ORed" together and send to the control unit. To save routing cost, we can "OR" neighboring signals together in a staggered fashion.

With this mechanism, we are able to collect timing error information in a sampling way during a certain period T (referred as "tuning period" in the following) as shown in Fig. 5.4. The tuning period T can be divided into several time segments $t_1, t_2, \cdots, t_n$. Each time segment (e.g., t_1) has one subsegment for counting mode (e.g., t_{c1}) and one subsegment for shifting mode (e.g., t_{s1}). By recording the number of timing errors in each block during $t_{c1}, t_{c2}, \cdots, t_{cn}$ respectively, we simply estimate the total number of errors of each block during the whole tuning period T as follows:

$$EN_{total} = (\sum_i EN_i) \cdot T / (\sum_i t_{ci}) \ , \tag{5.1}$$

wherein EN_{total} is total number of errors number during tuning period T, EN_i is the number of timing errors during t_{ci}. Note that, with the above equation, timing error rate in shifting mode is assumed to be the same as that during counting mode. This is not entirely accurate. However, since the fraction of time for error counters to operate in shifting mode is usually very small, it leads to negligible accuracy loss.

At the end of each tuning period T, control unit determines the skew setting for each block based on the collected timing error information, and sends out the corresponding control signals to the tunable delay buffers equipped with each basic tuning blocks, using a similar shifting mechanism. In other words, control signals are serially shifted into the configuration registers and they are applied in parallel when ready for applying skew tuning.

At runtime, control unit periodically determines intentional skews to manipulate timing slacks of different blocks using a heuristic approach for each tuning period T. The online tuning algorithm is detailed in Section 5.5, and it can be implemented using either hardware or software.

5.4 Tuning Block Formation

5.4.1 Problem Formulation

Since each tuning block is equipped with only one tunable delay buffer, we would like to have all the FFs within a block to lie in the same subtree[2] from the viewpoint of clock tree structure, so that we can simply insert the TDB in the root node of this subtree. The grouping problem studied here is formulated as follows:

Problem: **Given**

- A circuit with timing speculation and its critical path distribution[3];
- The corresponding clock tree $\mathbf{T} = (\mathbf{N}, \mathbf{E})$ with tree height h, wherein each node in $\mathbf{N} = \{N_{ij} : i = 1, \cdots, h; j = 1, \cdots, m_i\}$ represents a buffer ($i \neq h$) or FF ($i = h$), m_i is node number in tree level i, and $\mathbf{E}$ is the set of directed arcs representing precedence relationship;
- A pre-defined parameter β to constrain the total number of blocks;

to determine a grouping plan with maximum system controllability satisfying $n_b/n_f < \beta$, wherein n_b is the total number of blocks after grouping and n_f is total FF count. Here, system controllability is defined as $(1 - P_{inside}/P)$, wherein P_{inside} represents the number of critical paths inside blocks and P is the total number of critical paths in the whole circuit. Larger controllability means that more critical paths are outside of blocks after grouping, whose delays can be affected by clock skew tuning. How to obtain a grouping outcome with reasonable balance between system controllability and hardware cost should be explored.

[2]The concept "subtree" used in the context of clock tree means the same concept with "block", which will not be explicitly explained in the following.

[3]The receiving ends of critical paths are implemented as Razor-FFs.

5.4.2 Grouping Algorithm

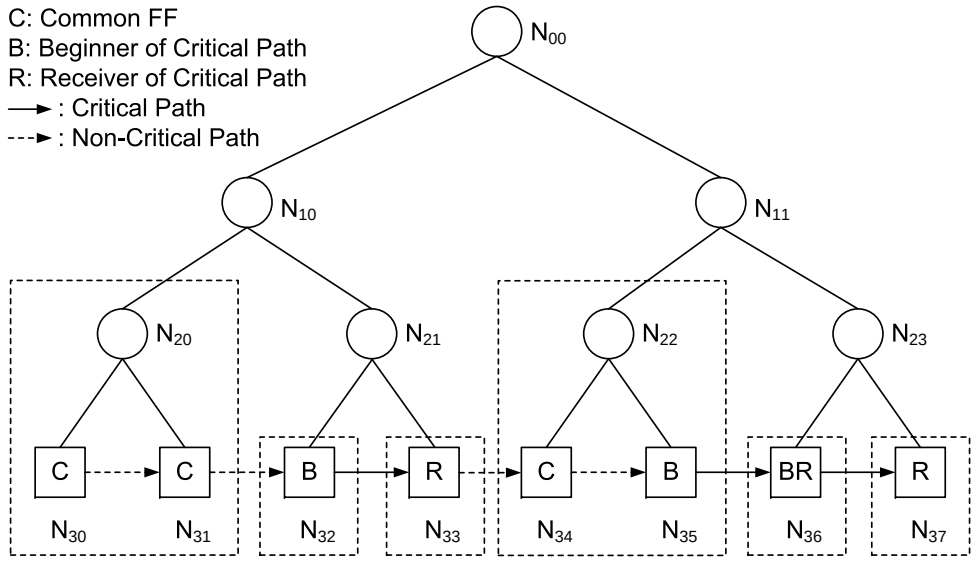
The proposed grouping procedure is a bottom-up approach based on the physical layout of the synthesized clock tree structure. To explain our grouping algorithm better, we use a simple example shown in Fig. 5.6 to present the key idea. In Fig. 5.6, the non-leaf nodes (represented as circles) are buffers in clock tree, while leaf nodes (represented as squares) are FFs. The grouping algorithm flow is detailed in Fig. 5.5.

#	T_{ij}: the subtree using N_{ij} as root
#	**T**: the set of subtrees
#	**B**: the resulting set of blocks
1.	Initialize $p = 0$
2.	**Repeat**
3.	$p++$
4.	Initialize all leaf nodes as T_{hj}
5.	$\mathbf{T} \leftarrow \{T_{hj} : j = 1, \cdots, m_h\}$
6.	**For** tree level iteration i from $(h-1)$ to 1
7.	**For** node index iteration j from 1 to m_i
8.	**If** all the children subtrees of $T_{ij} \in \mathbf{T}$ **AND** $critical\_path\_number(T_{ij}) < p$
9.	$\mathbf{T} \leftarrow$ Include(T_{ij})
10.	**Else B** $\leftarrow$ Include(all children subtrees of T_{ij})
11.	Exclude all the B-type and C-type blocks from **B**
12.	**Until** $n_b/n_f < \beta$

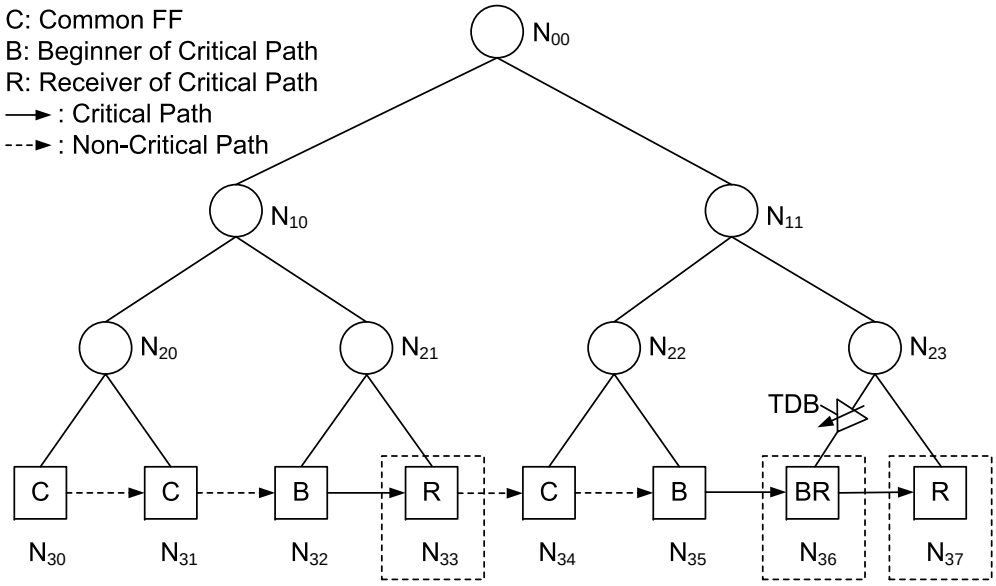
Figure 5.5: The proposed grouping algorithm.

First of all, we initialize all the leaf nodes as subtrees (Line 4∼5), and then try to merge them together to form larger subtrees (Line 6∼10). The decision on whether to merge subtrees or not depends on the critical path count in the merged subtree. Assuming we only perform merging when the merged subtree has no

critical path inside, this grouping scheme will finally exclude all the critical paths out of blocks, and hence clock skew tuning is able to take effects to all the critical paths and have the strongest controllability in manipulating timing slacks.



(a) B-type and C-type blocks included.



(b) B-type and C-type blocks excluded.

Figure 5.6: Tuning block formation: an example.

Such an aggressive grouping scheme may lead to a large number of blocks and hence introduce unaffordable hardware cost. To trade off system controllability and hardware cost, we allow a certain number (specified as parameter p) of critical paths inside subtrees. Consequently, the criterion used to decide merging becomes whether the number of critical paths within a subtree is less than a parameter p (see Line 8): if the critical path count in the new subtree is less than p, we merge its children subtrees together; otherwise, we have to treat its children subtrees as different blocks. By doing so, in the case with $p = 1$, we obtain subtrees without any critical path inside them as shown in Fig. 5.6 (a), where the subtrees are outlined by the dash line rectangles. We can see that the two ends of any critical path are separately included in different subtrees.

Block Type	FF Type				Equipment	
	BR-type FF	R-type FF	B-type FF	C-type FF	Tuning Mechanism	Observing Mechanism
BR-type Block	√	◯	◯	◯	Yes	Yes
R-type Block	×	√	◯	◯	No	Yes
B-type Block	×	×	√	◯	No	No
C-type Block	×	×	×	√	No	No

√: contained; ×: not contained; ◯: do not care.

Table 5.1: Categorization for tuning block formation.

However, it is not necessary to collect timing errors and add tunable delay buffer for every block shown in Fig. 5.6 (a). For the ease of discussion, let us first define four types of FFs according to their relationships to critical paths: (i) BR-type FFs, serving as both beginner and receiver of critical path; (ii) R-type FFs, serving as only receiver of critical path; (iii) B-type FFs, serving as only beginner of critical path; (iv) C-type FFs, common FFs, serving as neither beginner nor receiver of critical path. Based on this, we define four types of subtrees/blocks as listed in Table 5.1 according to the FF types these blocks contain. For example, if a certain block contains no BR-type FF but R-type FF inside, it is defined as

R-type block no matter whether it has B-type or C-type FF.

Based on this categorization, we have the following observation: for those FFs that are not included in BR-type blocks, their optimal skews can be easily determined offline at design stage. For example, as shown in Fig. 5.6 (a), since N_{33} is only an R-type FF, tuning its skew backwardly to the greatest extent within constrained maximum skew (defined in Section 4.2) must be beneficial for itself and at the same time does not worsen its successor N_{34}. As for B-type FF N_{35}, tuning forwardly within maximum skew constraint can benefit its successor N_{36} but never worsen itself. Therefore, for FFs in R-type/B-type blocks, there is no need to tune their skews at runtime since their optimal skew setting can be obtained at design stage. C-type blocks are not of concern and hence are ignored for skew tuning.

With the above, we can conclude that only BR-type block needs to be equipped with both timing error collection and tuning capabilities, R-type blocks requires to have timing error collection capability only, while the other two types do not need to be observed or tuned at runtime. Therefore, we can get the final set of tuning blocks as shown in Fig. 5.6 (b) (again, for the case with $p = 1$).

By incrementing parameter p, we can obtain grouping outcome with less blocks until satisfying the pre-defined requirement $n_b/n_f < \beta$. To get proper grouping faster, a binary search method can be used to set the parameter p, instead of incrementing its value by one each time.

5.5 Clock Skew Tuning Algorithms

5.5.1 Tuning Algorithm for Tackling Process Variation

Our proposed clock skew tuning technique for tackling process variation is comprised of two phases: (i) offline phase (see Fig. 5.7) and (ii) online phase (see

Fig. 5.8). By taking online timing error information into consideration, online clock skew tuning can optimize the circuit's timing performance based on the variation characteristics of a specific chip and the actual path sensitization of applications.

5.5.1.1 Tunable Skew Setting

Before introducing our algorithms in detail, let us discuss how to setup the skew tuning step for each block first. To guarantee that there exist no silent errors (i.e., timing errors occurring on those FFs that are not protected with Razor-like detectors), we constrain the maximum tunable skew values of TDBs to ensure enough timing margin is kept for non-Razor FFs. If the timing threshold γ (e.g., $\gamma = 80\%$ of clock cycle period) is used to set up Razor-FFs, the maximum tunable skew is

$$S_{max} = [(1-\gamma)\cdot T_c - \delta]/2\ , \tag{5.2}$$

wherein T_c is the operational clock cycle period and δ is a safety margin. Note that, in reality, it is very difficult, if not impossible, to implement a large number of tunable delay values into clock tree. Consequelty, in this work only five tuning skew levels[4] as shown in Table 5.2 are used in our implementation. Initially, the skew levels of all the TDBs are set to be 3.

Skew Level	1	2	3	4	5
Skew Value	$-S_{max}$	$-S_{max}/2$	0	$S_{max}/2$	S_{max}

Table 5.2: Tunable skew set.

5.5.1.2 Proposed Algorithm

As discussed in Section 5.4, for all those FFs that are not included in BR-type blocks after grouping procedure, their optimal skew tuning can be simply deter-

[4]Tuning forwardly/backwardly means decreasing/increasing the skew level.

#	**C**: the set of BR-type blocks, $\mathbf{C} \subset \mathbf{B}$
#	C_i: BR-type block, $C_i \in \mathbf{C}$
#	n_f: total FF count
#	**Offline Phase**
1.	**For** the iteration i from 1 to n_f
2.	**If** $FF_i \notin C_j$, for $\forall C_j$
3.	**If** FF_i is B-type FF
4.	Set skew level of $FF_i = 1$
5.	**Else** if FF_i is R-type FF
6.	Set skew level of $FF_i = 5$

Figure 5.7: The proposed algorithm in offline phase.

mined at design stage, as shown in Fig 5.7.

The most important part of proposed tuning algorithm is online phase (see Fig. 5.8), a heuristic approach targeting the FFs in BR-type blocks beyond the above offline phase.

It is, however, rather difficult to determine the skew setting of BR-type blocks, because any action taken to a block is possible to be a double-edged sword: either worsen the block itself to benefit the blocks as the receivers of critical paths starting from this block or improve itself to worsen the receivers. Consequently, we need to determine it at online phase (see Fig. 5.8). To tackle the above problem, we first define the *benefit* for block k as below,

$$benefit(k) = \sum_{i \in U_k} (error(i) - error(i)^{new}) \ , \tag{5.3}$$

wherein $error(i)$ is error rate of block i, and U_k is the block set including block k itself and all the blocks as the receivers of critical paths starting from block k. If $benefit(k) \geq 0$, the tuning action to block k in last tuning period is considered to be beneficial, otherwise it is not beneficial.

```
#   C: the set of BR-type blocks, C ⊂ B
#   C_i: BR-type block, C_i ∈ C
#   Online Phase
1.  Initialize C_i → skew = 3, for ∀C_i
2.  Initialize error(C_i) after 1st tuning period T_1, for ∀C_i
3.  For ∀C_i : error(C_i) = 0
4.      Set C_i → tunable = true
5.      C_i → skew − −
6.  Repeat for each tuning period T_{k_1} (k_1 = 2,3,···)
7.      For ∀C_i : error(C_i) = 0
8.          If C_i → tunable = true
9.              If benefit(C_i) ≥ 0
10.                 If C_i → skew ≠ 1
11.                     C_i → skew − −
12.                 Else set C_i → tunable = false
13.             Else C_i → skew + + // cancel last action
14.                 Set C_i → tunable = false
15. Until C_i → tunable = false, for ∀C_i : error(C_i) = 0
16. Set C_i → tunable = true, for ∀C_i : C_i → skew ≠ 5
17. Select C_i with largest error(C_i) AND C_i → tunable = true
18. C_i → skew + +
19. C_old ← C_i
20. Repeat for each tuning period T_{k_2} (k_2 = k_1 + 1,...)
21.     If benefit(C_old) < 0
22.         C_old → skew − − // cancel last action
23.         C_old → tunable = false
24.     Select C_i with largest error(C_i) AND
        C_i → tunable = true
25.     C_i → skew + +
26.     C_old ← C_i
27. Until C_i → tunable = false, for ∀C_i
```

Figure 5.8: The proposed algorithm in online phase.

First of all, we consider those blocks that do not have timing errors during the last tuning period (Lines 1∼15). Since these blocks do not encounter errors, it is very likely that they have extra timing budgets. Therefore, we tune their skews forwardly until their skews have been the lowest level "1" or the tuning is judged to be not beneficial any longer. If $benefit(k) \geq 0$ (Line 9), we consider it to be beneficial and keep the applied action to block k in last tuning period; otherwise, the applied action is canceled (Line 13) and the block is marked as "untunable" (Line 14). The above tuning process decreases the timing budgets of blocks without errors, while relaxing the timing stress of blocks with errors.

After that, our focus is changed to those blocks with errors (Line 16∼27). For each tuning period, we first evaluate whether the applied action in last tuning period is beneficial or not (Line 21). If not, we will cancel this action (Line 22) and mark this block as "untunable" (Line 23); otherwise, we keep the action, select the "tunable" block with largest error rate (Line 24) and tune its skew backwardly by one level (Line 25). This procedure is repeated periodically.

The proposed online tuning algorithm can be seen as a greedy heuristic, which tries to handle the block with the highest error rate each time. The mechanism to cancel those tuning actions that do not bring any benefits can guarantee to reduce the timing error rates of the overall system step by step. Note that, this skew tuning process tries to minimize timing error rate with a given clock period. We resort to binary search to sweep operational clock period and find a better solution. For each clock period, the online phase of tuning algorithm is repeated.

5.5.2 Tuning Algorithm for Mitigating Aging Effects

The above tuning algorithm is able to tackle static variation effects (e.g., process variation) at time0 (fresh after manufacturing). However, because the online phase of tuning algorithm (see Fig. 5.8) would terminate when all the blocks are marked

as "untunable" in a certain number of tuning periods, unfortunately it cannot handle the variation effects that do not exisit originally at time0 but accumulate after the termination of tuning algorithm in the long term, such as aging effects caused by Negative Bias Temperature Instability (NBTI) [91, 92]. Without loss of generality, we take NBTI-induced delay degradation into consideration and further develop the tuning algorithm to mitigate it. Note that, the proposed tuning algorithm can actually be utilized to tackle any other long-term variations.

5.5.2.1 NBTI Effects

NBTI, occurring with PMOS devices stressed under nagative gate voltage and resulting in the increase of PMOS threshold voltage, has emerged as a major threat to system reliability with technology scaling, since it can lead to delay degradation as high as 10% within three years under 90nm technology [91].

There have been a large number of research works (e.g., [91, 92]) trying to tackle NBTI problems. For example, Vaidyanathan *et al.* [92] modeled the impact of intrinsic NBTI variability on pipeline delay and proposed to conduct clock skew tuning (also known as dynamic cycle time stealing) to handle intrinsic NBTI variability. As indicated in [92], by inserting TDBs and tuning their skews after experiencing ten-year NBTI degradation, timing guard band to tolerate NBTI effects can be reduced by 30%. However, because the aim of [92] is simply to motivate the benefit of skew tuning for such a problem, in fact there is no skew tuning algorithm provided in [92].

5.5.2.2 Proposed Algorithm

An intuitive method to tackle NBTI problem is to repeatedly execute the tuning algorithm discussed in Section 5.5.1, triggered by the events that timing error rate of a tuning period exceeds a threshold due to delay increase of critical paths. Such

a simple scheme, however, requires many tuning periods for tuning algorithm to converge, leading to much more penalties for correcting errors that occur during the tuning process before convergence. Instead, a novel algorithm that tunes the skews is designed to mitigate NBTI effects, as detailed in the following.

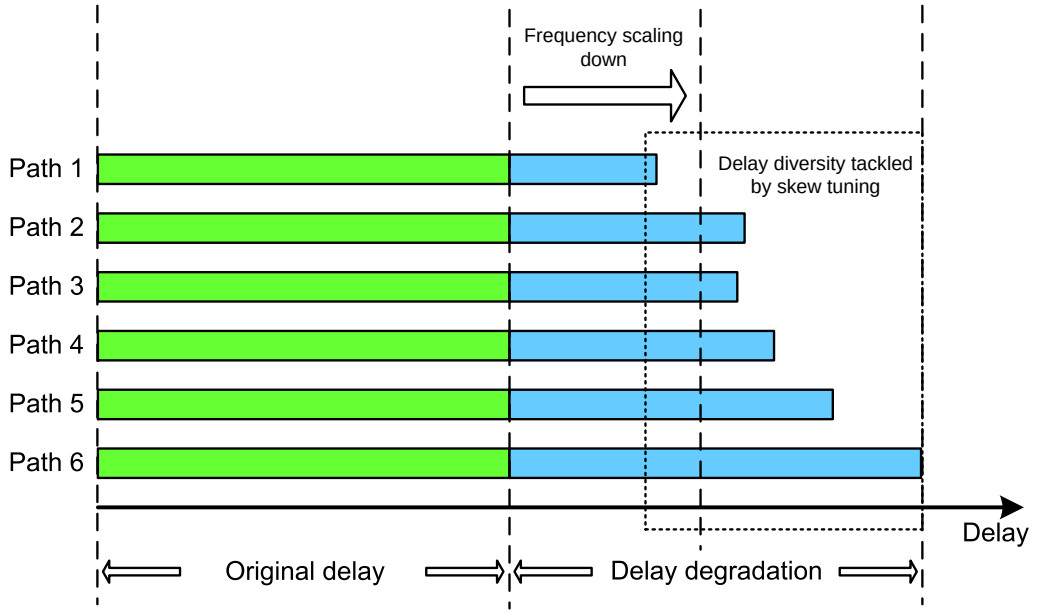


Figure 5.9: Delay degradation of critical paths.

As demonstrated in Fig. 5.9, the critical paths suffering from NBTI effects have delay degradation represented by the blue bars. Clearly, these paths may experience different aging environments (e.g., temperature, voltage, duty cycle), causing diverse delay degradations. Such diversity calls for skew tuning technique to re-allocate timing slacks of critical paths.

Besides, as discussed earlier the skew setting of TDBs are not arbitrary but actually constrained to some discrete values with a maximum tunable skew, which confines the flexibility of skew tuning to some extent. Suppose delay degradation of critical paths is so significant that it exceeds the maximum tunable range, skew tuning may not work at all. Consequently, to mitigate NBTI effects our skew

tuning algorithm should explicitly take clock frequency scaling into account. As shown in Fig. 5.9, frequency scaling can relax timing to a point with common delay degradation, and then skew tuning is performed based on new operational frequency to tackle the diversity of delay degradation.

Due to the above, we develop our tuning algorithm as described in Fig. 5.10. Firstly, we initialize the skews using the output generated by online phase algorithm described in Fig. 5.8. Secondly, we scale down clock frequency (see Line 2$\sim$4) step by step and monitor the change of error rate via the propose error collection mechanism. If error rate with lower frequncy is decreased, we keep current frequency; otherwise, we terminate frequency scaling and start to perform skew tuning. Next, we consider those blocks without timing errors (see Line 5$\sim$16). Their skews are tuned forwardly one by one. Error rates of the blocks are observed to calculate $benefit(C_i)$, the criteria determining last action should be cancelled or not. Finally, different from online phase algorithm, we handle the blocks with increasing error rate (see Line 17$\sim$28). This is because, since skew tuning in last tuning period has effectively balance timing slacks of critical paths and the delay degradation is simply an additional delay, skew tuning in current tuning period, in fact, needs to tackle the newly-emerged errors only. Consequently, we calculate the increase of error rate of each block:

$$\Delta error(C_i) = error(C_i)^{new} - error(C_i) \quad , \tag{5.4}$$

select the block with largest change of error rate, and tune its skew backwardly. If this action is observed to be not beneficial in next tuning period, we would cancel it. By doing so, tuning algorithm requires much less tuning periods to converge when compared to online phase algorithm.

```
#   C: the set of BR-type blocks, C ⊂ B
#   C_i: BR-type block, C_i ∈ C
#   f: operational clock frequency
#   Δf: decrease of clock frequency in each scaling
1.  Initialize C_i → skew, for ∀C_i

#   Frequency Scaling
2.  Repeat for each tuning period T_k3 (k_3 = k_2 + 1, ...)
3.      Scaling down frequency f^new = f − Δf
4.  Until error(C)^new > error(C)

#   Skew Tuning
5.  Set C_i → tunable = true, for ∀C_i : C_i → skew ≠ 1
6.  Select C_i with error(C_i) = 0 AND C_i → tunable = true
7.  C_i → skew − −
8.  C_old ← C_i
9.  Repeat for each tuning period T_k4 (k_4 = k_3 + 1, ...)
10.     If benefit(C_old) < 0
11.         C_old → skew + + // cancel last action
12.         C_old → tunable = false
13.     Select C_i with error(C_i) = 0 AND
        C_i → tunable = true
14.     C_i → skew − −
15.     C_old ← C_i
16. Until C_i → tunable = false, for ∀C_i
17. Set C_i → tunable = true, for ∀C_i : C_i → skew ≠ 5
18. Select C_i with largest Δerror(C_i) AND C_i → tunable = true
19. C_i → skew + +
20. C_old ← C_i
21. Repeat for each tuning period T_k5 (k_5 = k_4 + 1, ...)
22.     If benefit(C_old) < 0
23.         C_old → skew − − // cancel last action
24.         C_old → tunable = false
25.     Select C_i with largest Δerror(C_i) AND
        C_i → tunable = true
26.     C_i → skew + +
27.     C_old ← C_i
28. Until C_i → tunable = false, for ∀C_i
```

Figure 5.10: The proposed skew tuning algorithm to tackle aging effects.

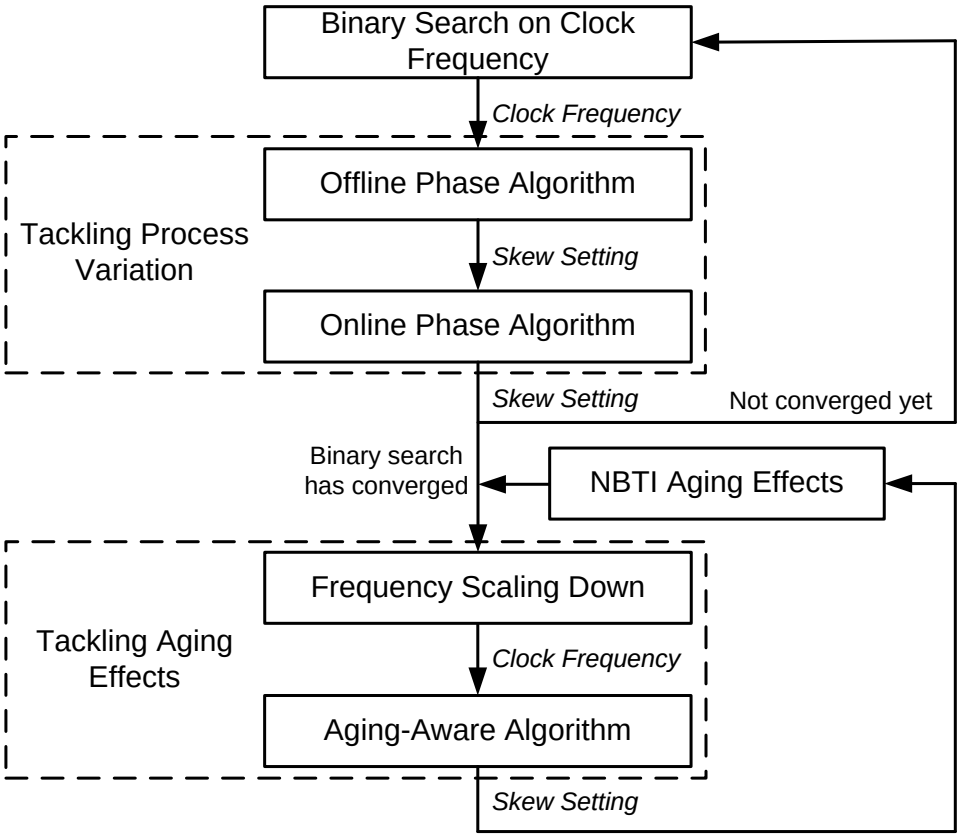


Figure 5.11: The overall flow of skew tuning algorithm.

5.5.3 Overall Flow

The overall flow of our proposed skew tuning algorithm is depicted in Fig. 5.11. First of all, we perform binary search on the operational clock frequency. Then, for each selected frequency, we repeatedly use offline phase and online phase algorithms to optimize skew setting, until binary search has converged. After that, due to NBTI aging effects, delay degradation fails the original skew setting and leads to the increase of error rate. To tackle this problem, we scale down clock frequency in a heuristic manner and then perform the proposed aging-aware algorithm.

5.6 Experimental Results

5.6.1 Experimental Setup

To evaluate the effectiveness of the proposed online skew tuning methodology, we conduct experiments on several large ISCAS'89 and IWLS'05 benchmarks. We synthesize these circuits, generate clock tree structures, and obtain timing information using Synopsys EDA tools. To take process variation effect into consideration, we perform Monte Carlo simulation to inject gate-level delay variation following Gaussian distribution with standard deviation equal to 5%. NBTI-induced delay degradation is injected according to [91]. We consider 10 aging periods, each of which has 3×10^7 seconds.

We set the top 5% longest paths as critical paths and treat their receivers as Razor-FFs, and we assume with the help of error recovery mechanism, we can roll back the system once timing error is detected and we can lower system frequency for a short while to re-compute the result in the failure cycle. Similar with [12], we can trade off timing error rate with clock cycle period to achieve a higher throughput according to the following equation,

$$\min_{T_c}[(1 + error(T_c) \cdot penalty) \cdot T_c] \,, \tag{5.5}$$

wherein T_c is clock cycle period, $error(T_c)$ is the percentage of cycles to have timing errors, and $penalty$ is the penalty due to error cycle occurring. This penalty includes both the cycles of wasted execution that must be discarded when an error occurs and the time spent on checkpointing and re-execution. According to [12], we assume the penalty to be 10 cycles.

We assume there are five discrete tuning levels and we conduct simulation with random inputs in our experiments. The tuning period for the proposed algorithm is set as 100,000 cycles. Note that, longer period can be used in practical applications. The experiments are conducted on a *2.8GHz* PC with *4GB* RAM.

Bench.	TG#	TFF#	RB#	BRB#	Cost(%)	Runtime(s)
s38584	21021	1426	4	15	2.62	0.010
s38417	23949	1636	5	22	3.30	0.026
des_perf	155746	9105	0	39	0.83	2.458
ethernet	164912	10752	17	36	0.76	9.070

TG#: total gate count; TFF#: total FF count;

RB#: R-type block count; BRB#: BR-type block count;

Cost: hardware cost for equipping R-type and BR-type blocks.

Table 5.3: Experimental results on hardware cost.

To provide a reasonable baseline reference, we utilize the Useful Skew Technique of IC Compiler to optimize the skews during the CAD process. The minimum clock cycle period reported by IC Compiler considering non-Razor case is denoted as $CSO_{nonrazor}$. The optimal clock cycle period with reasonable error cycle rate, selected according to Equation 5.5, is denoted as $CSO_{baseline}$ and used as the baseline solution. Note that, any design-phase clock skew scheduling algorithm can be combined with our post-silicon skew tuning framework as an initial solution. The offline phase of our proposed skew tuning algorithm in Section 5.5.1 is denoted as $CSO_{offline}$ and the online phase is denoted as CSO_{online}. The aging-aware skew tuning algorithm proposed in Section 5.5.2 is denoted as CSO_{aging}.

5.6.2 Results and Discussion

5.6.2.1 Results on grouping algorithm

In Table 5.3, we present the result for our grouping algorithm. The parameter β used to constrain the total number of blocks (see Section 5.4) is set to be 2% for small scale benchmarks (*s38584* and *s38417*), and 0.5% for large benchmarks (*ethernet* and *des_perf*). Assuming skew tuning algorithm is implemented with software, the proposed architecture (see Fig. 5.3) is implemented and the addi-

Bench.	$CSO_{nonrazor}$		$CSO_{baseline}$				$CSO_{offline}$				CSO_{online}				
	$CP(ns)$	$Th.(MHz)$	$CP(ns)$	$Err.(\%)$	$Th.(MHz)$	$\Delta_1(\%)$	$CP(ns)$	$Err.(\%)$	$Th.(MHz)$	$\Delta_2(\%)$	$CP(ns)$	$Err.(\%)$	$Th.(MHz)$	$\Delta_3(\%)$	$\Delta_4(\%)$
s38584	8.24	121.36	7.86	0.042	126.69	4.40	7.72	0.047	128.93	1.76	7.31	0.081	135.70	5.25	7.11
s38417	7.93	126.10	7.59	0.097	130.49	3.48	7.47	0.122	132.26	1.36	7.17	0.131	137.67	4.09	5.50
des_perf	15.52	64.43	14.38	0.064	69.10	7.24	14.2	0.047	70.09	1.44	13.25	0.062	75.01	7.01	8.55
ethernet	13.52	73.96	12.64	0.093	78.38	5.98	12.34	0.088	80.33	2.48	11.45	0.106	86.42	7.58	10.25

CP: clock period; $Err.$: error cycle rate under selected clock period; $Th.$: throughput;

Δ_1: throughput difference ratio between $CSO_{baseline}$ and $CSO_{nonrazor}$; Δ_2: throughput difference ratio between $CSO_{offline}$ and $CSO_{baseline}$;

Δ_3: throughput difference ratio between CSO_{online} and $CSO_{offline}$; Δ_4: throughput difference ratio between CSO_{online} and $CSO_{baseline}$.

Table 5.4: Experimental results on tackling process variation.

tional hardware cost to enable online tuning is presented in Column 6 of Table 5.3. As can be seen, the costs are within 4% for small benchmark circuits and within 1% for larger ones, and they are strongly related to the number of basic tuning blocks in the circuit. The runtime to finish grouping procedure is illustrated in Column 7.

5.6.2.2 Results on tackling process variation

Firstly, we consider one particular instance of the benchmark circuits (i.e., under a specific variation pattern). In Table 5.4, *CP* is the optimal clock period selected according to Equation 5.5. $Err.$ is the corresponding error cycle rate under selected clock period *CP*, and $Th.$ is the corresponding throughput. We can see that, for *s*38584, the baseline $CSO_{baseline}$ can obtain 4.40% throughput improvement compared with $CSO_{nonrazor}$, a non-Razor solution optimized by IC Compiler. This improvement reflects the efficacy of Razor technique, which is not the main contribution of this work. With error information collection and clock skew tuning capabilities, offline phase $CSO_{offline}$ has 1.76% improvement compared with $CSO_{baseline}$, and online phase CSO_{online}, again, can achieve extra 5.25% improvement compared with $CSO_{offline}$. Similarly, the results of other benchmark circuits

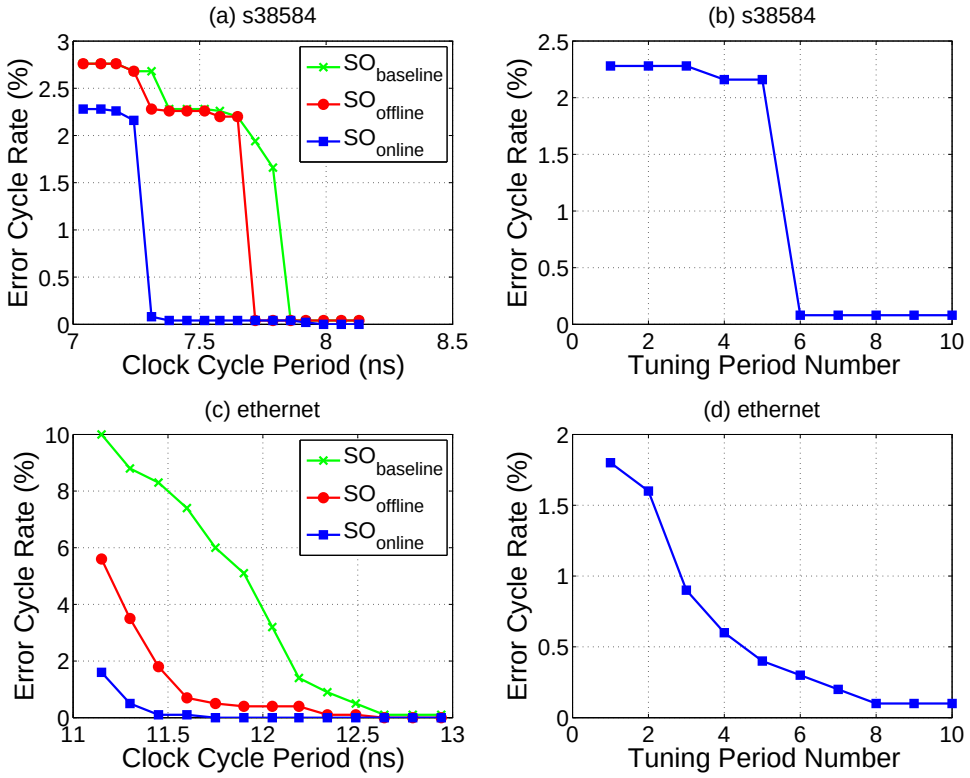


Figure 5.12: Error cycle rates with respect to clock cycle period and tuning period number.

are also listed in Table 5.4, and we can see that relatively larger improvement can be achieved with larger benchmark circuits.

To have a closer look into the clock tuning process, we present experimental results in Fig. 5.12, using benchmark circuits *s*38584 and *ethernet* as the representative of small and relatively larger benchmarks, respectively. Fig. 5.12 (a) and Fig. 5.12 (c) show that the three curves in each figure have similar trend with respect to increasing cycle period, while the lowest error cycle rate is always achieved by CSO_{online}. Using the skew setting of $CSO_{offline}$ as initial state, CSO_{online} tunes the skews periodically (see Section 5.5 for details). The error cycle

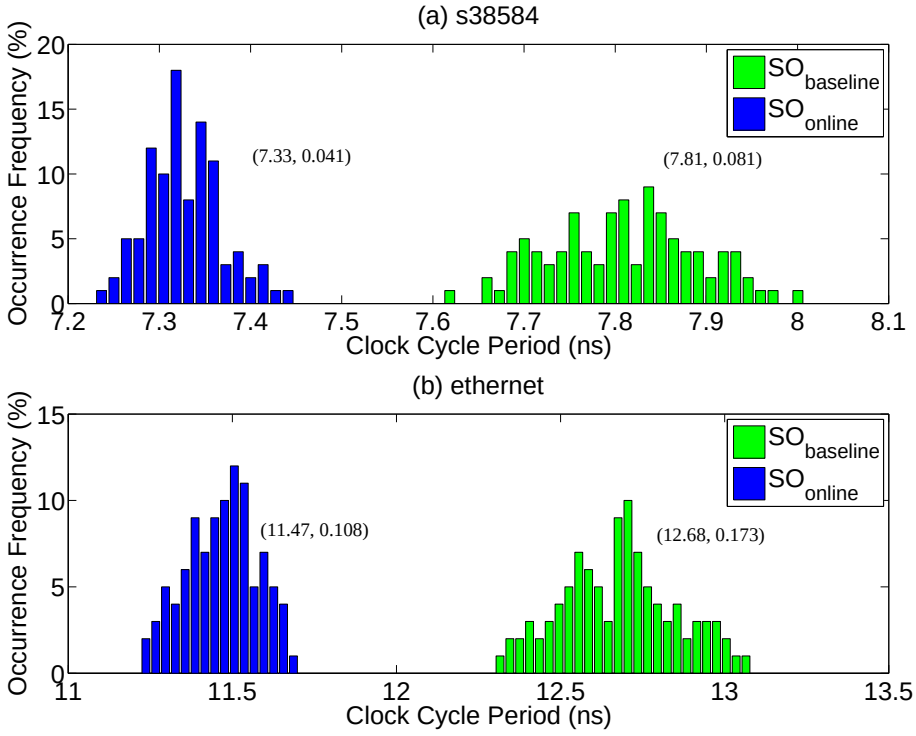


Figure 5.13: Variation effects.

rates of first 10 tuning periods under the optimal clock period selected by CSO_{online} are illustrated in Fig. 5.12 (b) and Fig. 5.12 (d) to present the change of error cycle rate during online tuning. One notable finding from these figures is that proposed skew tuning can achieve similar decreasing effect of error cycle rate with that of clock period increasing. In other words, by tuning skews we can achieve the error cycle rate as low as the case with laxer clock period.

We conduct Monte Carlo simulation to produce 100 sample circuits[5] with different variation patterns. Fig. 5.13 indicates that the mean of selected clock period after applying CSO_{online} is much smaller than that of $CSO_{baseline}$, thanks to

[5]Due to computational complexity in our experiments, we cannot afford to have more Monte Carlo simulations.

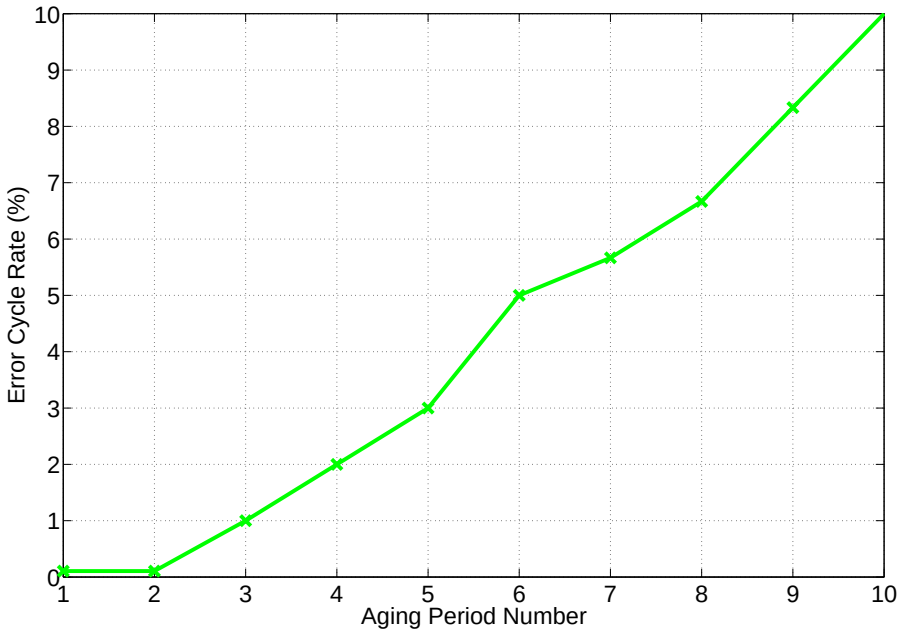


Figure 5.14: Error cycle rate with respect to aging period number.

clock skew tuning. More importantly, with the same process variation distribution, the clock period distribution of CSO_{online} is with much smaller standard deviation when compared to $CSO_{baseline}$. This is expected, because, unlike offline solutions that can at best optimize the circuit according to a given process variation model, online clock skew tuning takes advantage of the knowledge on each individual chip by timing error collection and facilitates to obtain an optimized chip-specific skew assignment. The corresponding mean clock period and standard deviation of each case are indicated in Fig. 5.13 in the form of (μ, σ).

5.6.2.3 Results on mitigating aging effects

We take the largest benchmark *ethernet* as an example to demonstrate how NBTI affects the performance of timing-speculative circuits. Given the skew setting that

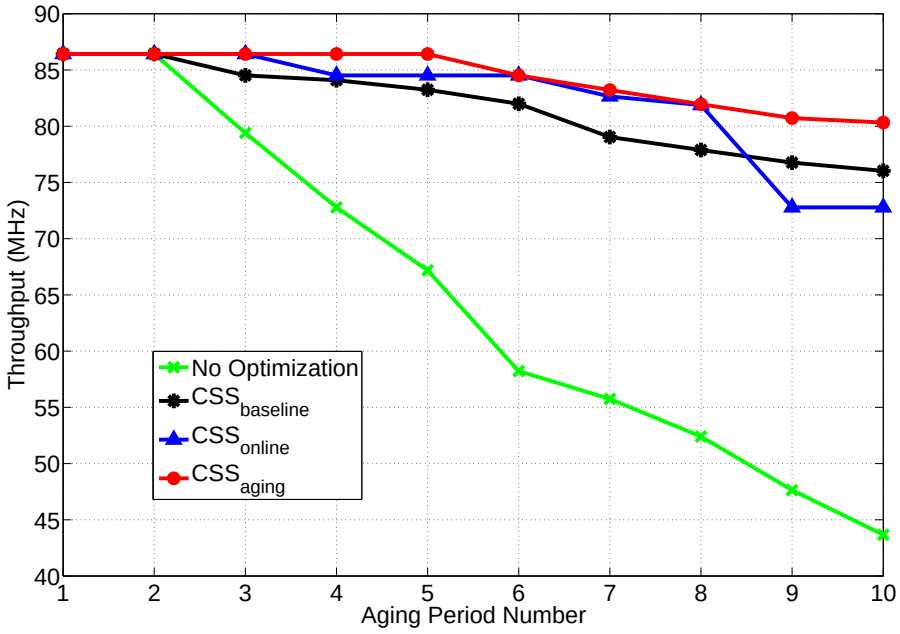


Figure 5.15: Throughputs with respect to aging period number.

is generated by the online phase in Section 5.5.1 and then fixed without any optimization technique during all the aging periods, the error cycle rate is significantly increased with respect to aging period number, as shown in Fig. 5.14. We can see that, after 10 aging periods the error rate is as high as 10%, motivating this work to mitigate aging effects.

Next, we show the throughput degradation of *ethernet* in Fig. 5.15. The green curve without optimization technique is presented as a reference of throughput degradation with respect to aging period number. $CSO_{baseline}$ is repeatedly scaling down frequency for each aging period, while CSO_{online} is repeatedly conducting online phase algorithm without frequency scaling. As can be seen, all the three techniques can achieve significant throughput improvement, and CSO_{aging} outperforms the others. When comparing $CSObaseline$ and CSO_{online}, it is interesting to

Bench.	$CSO_{baseline}$	CSO_{online}		CSO_{aging}		
	$Th.(MHz)$	$Th.(MHz)$	$\Delta_1(\%)$	$Th.(MHz)$	$\Delta_2(\%)$	$\Delta_3(\%)$
s38584	124.15	126.51	1.90	128.49	3.50	1.57
s38417	130.02	118.41	-8.93	131.14	0.87	10.75
des_perf	62.96	59.16	-6.03	67.86	7.78	14.70
ethernet	81.64	82.29	0.80	84.28	3.24	2.42
AVERAGE			-3.06		3.85	7.36

Δ_1: throughput difference ratio between CSS_{online} and $CSS_{baseline}$;

Δ_2: throughput difference ratio between CSS_{aging} and $CSS_{baseline}$;

Δ_3: throughput difference ratio between CSS_{aging} and CSS_{online}.

Table 5.5: Experimental results on mitigating aging effects.

note that CSO_{online} is better than $CSO_{baseline}$ in the case of smaller aging period number and worse in the case of larger aging period number. This observation justifies our argument that skew tuning can tackle the diversity of delay degradation while frequency scaling can tackle the common delay degradation. When delay degradation exceeds the maximum tunable range of TDBs, skew tuning would not work.

Finally, we show the average throughput of four benchmarks within 10 aging periods in Table 5.5. As can be seen, when compared to $CSO_{baseline}$, CSO_{online} has even worse throughput, again justifying our argument earlier. CSO_{aging} can achieve 3.85% and 7.36% throughput improvement when compared to $CSO_{baseline}$ and CSO_{online}, respectively.

5.7 Conclusion

Clock skew optimization is a widely-used technique to improve circuit timing performance, in which we assign intentional clock arrival times to FFs in synchronized sequential circuits. Traditionally, a large timing guard band needs to be reserved due to various variation effects. In this work, with the support of elab-

orately designed hardware architecture, we propose an online clock skew tuning framework for timing-speculative circuits. By observing the occurrence of timing errors at runtime and tuning clock skews accordingly, the proposed technique is able to achieve much better timing performance when compared to existing clock skew optimization solutions, as demonstrated in our experimental results.

□ End of chapter.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

With the continuous downscaling of transistor feature size, the ever-increasing timing uncertainties caused by various variation effects often manifest themselves as infrequent timing errors on speed-paths. Timing speculation, a better-than-worst-case design methodology, serves as one of the most promising solutions to mitigate the variation effects in nanometer technologies and has attracted a significant amount of research effort. However, as timing speculation is not orthogonal to conventional worst-case-oriented design techniques, the "timing wall" built by conventional techniques would significantly limit the effectiveness of timing speculation to a critical point beyond which a massive number of timing errors will occur, leading to considerable performance degradation and/or energy loss.

As a consequence, this thesis first investigates the premises and prospects of timing speculation by studying the minimum and maximum potential benefits that are achievable by conducting timing speculation. This work specifically answers the question posed by the conflict between conventional techniques and timing speculation, and is able to identify which methodology is preferable for a given

circuit netlist in terms of energy efficiency. As timing error correction incurs non-trivial performance/energy overhead, it is very essential to reshape the delay distribution of critical paths in timing-speculative circuits to reduce timing error rates. This thesis also investigates multiple design and optimization problems, including post-placement voltage island design, offline clock skew scheduling and online clock skew tuning, in order to realize a practical implementation of timing-speculative circuit.

After conducting the above research work, we explore the potential of timing speculation in performance improvement and energy reduction by trading off reliability against performance and power. As reliability problem is becoming increasingly critical in nanometer technologies, this thesis research provides a promising alternative for the development of future electronic industry.

6.2 Future Work

To realize practical timing speculation, there are still several important topics that should be explored in the future.

First, one of the fundamental problems for timing speculation is how to synthesize timing-speculative circuits from the ground up without significant impact on the original design flow. Particularly, as the delay distribution of a circuit after optimizations is changed dramatically, the originally non-suspicious flip-flops can become suspicious ones that need to be replaced with timing speculators. This problem requires iteratively re-synthesizing the timing-speculative circuits and may lead to a significant increase of design time.

Second, with the ever-increasing process variations, especially in the context of dark silicon that poses much severer variation effects, a larger speculation window is necessary to assure the functionalities of current timing error detectors. However, larger speculation window dramatically increases the hold time constraint

and also the associated overheads. How to design more effective and efficient timing speculation techniques is therefore an interesting and important problem in the near future.

Last but not least, timing error correction requires additional hardware that are very likely to be complex and application-specific. As the system has to roll back the system to recover from timing errors in pipelined structures or mask timing errors in general non-pipelined circuits, effective and efficient timing error correction mechanisms should also be investigated for general-purpose applications.

□ End of chapter.

Bibliography

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, V. De, Parameter variations and impact on circuits and microarchitecture. In *Proc. Design Automation Conference (DAC)*, 2003.

[2] D. Frank, R. Puri, D. Toma, Design and CAD challenges in 45nm CMOS and beyond. In *International Conference on Computer-Aided Design (ICCAD)*, 2006.

[3] S. Borkar, Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2005.

[4] T. Austin, V. Bertacco, D. Blaauw, T. Mudge, Opportunities and challenges for better than worst-case design. In *Proc. Asia South Pacific Design Automation Conference (ASP-DAC)*, 2005.

[5] T. Austin , V. Bertacco, Deployment of better than worst-case design: solutions and needs. In *Proc. International Conference on Computer Design (ICCD)*, 2005.

[6] R. Hegde , N. R. Shanbhag, Energy-efficient signal processing via algorithmic noise-tolerance. In *Proc. International Symposium on Low Power Electronics and Design (ISLPED)*, 1999.

[7] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, J. Torrellas, VARIUS: A model of process variation and resulting timing Errors for microarchitects. In *IEEE Transactions on Semiconductor Manufacturing*, 2008.

[8] T. Austin. Diva: A reliable substrate for deep submicron microarchitecture design. In *Proc. International Symposium on Microarchitecture (MICRO)*, 1999.

[9] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, T. Mudge Razor: a low-power pipeline based on circuit-level timing speculation. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2003.

[10] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull, D. T. Blaauw, RazorII: In situ error detection and ccorrection for PVT and SER tolerance. In *IEEE Journal of Solid-State Circuits*, 2009.

[11] Y. Liu, F. Yuan , Q. Xu, Re-Synthesis for Cost-Efficient Circuit-Level Timing Speculatio. In *Proc. Design Automation Conference (DAC)*, 2011.

[12] M. de Kruijf, S. Nomura, K. Sankaralingam, unified model for timing speculation: evaluating the impact of technology scaling, CMOS design style, and fault recovery mechanism. In *Proc. International Conference on Dependable Systems and Networks (DSN)*, 2010.

[13] S. Sarangi, B. Greskamp, A. Tiwari, J. Torrellas, EVAL: utilizing processors with variation-induced timing errors. In *Proc. International Symposium on Microarchitecture (MICRO)*, 2008.

[14] B. Greskamp, L. Wan, U. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen, C. Zilles, Blueshift: Designing processors for timing speculation from the ground up. In *Proc. International Symposium on High Performance Computer Architecture (HPCA)*, 2009

[15] M. R. Choudhury, K. Mohanram. Masking timing errors on speed-paths in logic circuits. In *Proc. Design, Automation, and Test in Europe (DATE)*, pp.87-92, 2009.

[16] M. Kurimoto, H. Suzuki, R. Akiyama, T. Yamanaka, H. Ohkuma, H. Takata and H. Shinohara. Phase-adjustable error detection flip-flops with 2-stage hold driven optimization and slack based grouping scheme for dynamic voltage scaling. In *Proc. Design Automation Conference (DAC)*, pp.884-889, 2008

[17] K. Hirose, Y. Manzawa, M. Goshima, S. Sakai. Delay-compensation flip-flop with in-situ error monitoring for low-power and timing-error-tolerant circuit design. In *Japanese Journal of Applied Physics*, vol.47, issue.4, pp.2779-2787, 2008.

[18] M. Choudhury, V. Chandra, K. Mohanram and R. Aitken. TIMBER: Time borrowing and error relaying for online timing error resilience. In *Proc. Design, Automation, and Test in Europe (DATE)*, pp.1554-1559, 2010.

[19] L. Wan , D. Chen, Dynatune: Circuit-level optimization for timing speculation considering dynamic path behavior. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2009.

[20] J. Cong , K. Minkovich, Logic synthesis for better than worst-case designs. In *Proc. International Symposium on VLSI Design, Automation and Test*, 2009.

[21] J. Cortadella, Timing-driven logic bi-decomposition. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2003.

[22] M. Borah, R.M. Owens, and M.J. Irwin. Transistor sizing for low power CMOS circuits. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol.15, no.6, pp.665-671, Jun 1996.

[23] J. Rabaey. Low Power Design Essentials. *Springer*, 2009.

[24] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[25] D. Frank, R. Puri, and D. Toma. Design and CAD Challenges in 45nm CMOS and beyond. In *Proc. ICCAD*, pp. 329–333, 2006.

[26] T. Austin and V. Bertacco. Deployment of better than worst-case design: solutions and needs. In *Proc. ICCD*, pp. 550–555, 2005.

[27] K. Bowman, J. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S.-L. Lu, T. Karnik, V. K. De, Energy-efficient and metastability-immune timing-error detection and instruction-replay-based recovery circuits for dynamic-variation tolerance. In *Proc. International Solid State Circuits Conference (ISSCC)*, 2008.

[28] Y. Liu, R. Ye, F. Yuan, R. Kumar, Q. Xu, On Logic Synthesis for Timing Speculation. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2012.

[29] X. Bai, C. Visweswariah, P. N. Strenski, and D. J. Hathaway. Uncertainty-Aware Circuit Optimization. In *Proc. DAC*, pp. 58-63, 2002.

[30] J. Patel. CMOS Process Variations: A Critical Operation Point Hypothesis. *Online Presentation*, 2008.

[31] A. B. Kahng, S. Kang, R. Kumar, J. Sartori, Slack redistribution for graceful degradation under voltage overscaling. In *Proc. Asia South Pacific Design Automation Conference (ASP-DAC)*, 2010.

[32] A.B. Kahng, S. Kang, R. Kumar, and J. Sartori. Designing a processor from the ground up to allow voltage/reliability tradeoffs. In *Proc. HPCA*, pp.1-11, 2010.

[33] R. Ye, F. Yuan, H. Zhou, Q. Xu, Clock Skew Scheduling for Timing Speculation. In *Proc. Design, Automation, and Test in Europe (DATE)*, 2012.

[34] R. Sproull, I. Sutherland, C. Molnar, The counterflow pipeline processor architecture. In *IEEE Design & Test of Computers*, 1994.

[35] J. A. Snyman. Practical mathematical optimization: An introduction to basic optimization theory and classical and new gradient-based algorithms. *Springer*, 2005.

[36] M. C. Vanier and J. M. Bower. A Comparative Survey of Automated Parameter-Search Methods for Compartmental Neural Models. In *Journal of Computational Neuroscience* , pp. 149-171, 1999.

[37] J. Hu, *et al.* Sensitivity-guided metaheuristics for accurate descrete gate sizing.. In *Proc. ICCAD*, 2012.

[38] D. Sengupta, R. A. Saleh. Application-Driven Voltage-Island Partitioning for Low-Power System-on-Chip Design. In *IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[39] N. Weste and D. Harris. CMOS VLSI Design: A Circuits and Systems Perspective. 4th edition, Addison Wesley, March 2010.

[40] M.J. Rabaey, A. Chandrakasan, and B. Nikolic. Digital integrated circuits. 2nd edition, Pearson, New Delhi, India, 2003.

[41] N.S. Kim, *et al.* Leakage current: Moore's law meets static power. In *Computer*, vol.36, no.12, pp.68,75, Dec. 2003.

[42] S. Huang, C. Cheng, C. Chang, and Y. Nieh. Clock period minimization with minimum delay insertion. In *Proc. DAC*, pp. 970-975, 2007.

[43] Q. Ma, E. Young. Multivoltage Floorplan Design. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits Systems*, 2010.

[44] W. K. Mak, J. W. Chen. Voltage Island Generation under Performance Requirement for SoC Designs. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2007.

[45] W. Lee, H. Liu, Y. Chang. An ILP Algorithm for Post-Floorplanning Voltage-Island Generation Considering Power-Network Planning. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2007

[46] B. Liu, Y. Cai, Q. Zhou , X. Hong. Power Driven Placement with Layout Aware Supply Voltage Assignment for Voltage Island Generation in Dual-Vdd Designs. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2006.

[47] L. Guo, Y. Cai, Q. Zhou, X. Hong. Logic and Layout Aware Voltage Island Generation for Low Power Design. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2007.

[48] H. Wu, M, Wong, I. Liu, Y. Wang. Placement-Proximity-Based Voltage Island Grouping Under Performance Requirement. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2007.

[49] H. Wu, M. Wong. Incremental Improvement of Voltage Assignment. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[50] H. Wu, M. Wong, I. Liu. Timing-Constrained and Voltage-Island-Aware voltage assignment. In *Proc. Design Automation Conference (DAC)*, 2006.

[51] R. Ching, E. Young, K. Leung , C. Chu. Post-Placement Voltage Island Generation. In *International Conference on Computer-Aided Design (ICCAD)*, 2006.

[52] K. Bowman, J. Tschanz, C. Wilkerson, S. Lu, T. Karnik, V. De, S. Borkar, Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance. In *IEEE Journal of Solid-State Circuits*, 44(1): 49–62, 2009.

[53] V. Kozhikkottu, S. Dey, A. Raghunathan, Recovery-based design for variation-tolerant SoCs. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 826-833, 2012.

[54] K. Roy, S. Mukhopadhyay, H. Mahmoodi-Meimand. Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits. In *Proceedings of the IEEE*, 2003.

[55] B. Greskamp, et al. Blueshift: Designing processors for timing speculation from the ground up. *Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 213-224, 2009.

[56] A. B. Kahng, et al. Slack redistribution for graceful degradation under voltage overscaling. *Proc. IEEE/ACM Asia South Pacific Design Automation Conference (ASP-DAC)*, pp. 825-831, 2010.

[57] R. Ye, F. Yuan and Q. Xu. Online clock skew tuning for timing speculation. *Proc. ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 442–447, 2011.

[58] Y. Liu, et al. On logic synthesis for timing speculation. *Proc. ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 591–596, 2012.

[59] J. Lin, W. Cheng, C. Lee and R. C.J. Hsu, Voltage island-driven floorplanning considering level shifter placement. In *Proc. IEEE/ACM Asia South Pacific Design Automation Conference (ASP-DAC)*, pp. 443-448, 2012.

[60] S. Muthukrishnan, T. Suel, Approximation Algorithms for Array Partitioning Problems. In *Journal of Algorithms*, Volume 54, Issue 1, pp. 85-104, Jan. 2005.

[61] F. Yuan and Q. Xu, On Timing-Independent False Path Identification. In *Proc. ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 532-535, 2010.

[62] J. Sartori and R. Kumar, Architecting Processors to Allow Voltage/Reliability Tradeoffs. In *Proc. ACM International Conference on Compilers Architecture and Synthesis for Embedded Systems (CASES)*, pp. 115-124, 2011.

[63] J. P. Fishburn. Clock skew optimization. In *IEEE Transactions on Computers*, vol.39, pp.945-951, July 1990.

[64] R. B. Deokar and S. S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 407-410, 1994.

[65] I. S. Kourtev and E. G. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 239-243, 1999.

[66] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Cycle time and slack optimization for VLSI-chips. In *Proc. IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, pp. 232-238, 1999.

[67] X. Wei, Y. Cai, and X. Hong. Clock skew scheduling under process variation. In *Proc. IEEE International Symposium on Quality Electronic Design (ISQED)*, pp. 237-242, 2006.

[68] J. L. Tsai, D. H. Baik, C. C. P. Chen, and K. K. Saluja Yield-driven, false-path-aware clock skew scheduling. In *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 214-222, 2005.

[69] Y. Wang, et al. Timing yield driven clock skew scheduling considering non-Gaussian distributions of critical path delays. In *Proc. IEEE/ACM Design Automation Conference (DAC)*, pp. 223-226, 2008.

[70] K. Bowman, et al. Circuit techniques for dynamic variation tolerance. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 4-7, 2009.

[71] R. Ye, F. Yuan, Q. Xu, Online Clock Skew Tuning for Timing Speculation. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, 2011.

[72] K. Ravindran, A. Kuehlmann, and E. Sentovich. Multi-domain clock skew scheduling. In *Proc. of the IEEE/ACM International Conference on Computer-aided Design*, pp. 801-808, 2003.

[73] M. Ni, and S. O. Memik. A fast heuristic algorithm for multidomain clock skew scheduling. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 18, No. 4, pp. 630-637, 2010.

[74] L. Li, Y. Hu, and H. Zhou. Optimal multi-domain clock skew scheduling. In *Proc. of the 48th annual Design Automation Conference*, pp. 152-157, 2011.

[75] J. Casanova and J. Cortadella. Multi-level clustering for clock skew optimization. In *Proc. of the 2009 International Conference on Computer-Aided Design (ICCAD '09)*, pp. 547-554, 2009.

[76] Y. Zhi, W. Luk, H. Zhou, C. Yan, H. Zhu, X. Zeng. An efficient algorithm for multi-domain clock skew scheduling. In *Proc. Design, Automation & Test in Europe (DATE)*, pp. 1-6, 2011.

[77] I. S. Kourtev, B. Taskin, and E. G. Friedman. Timing optimization through clock skew scheduling. *Springer*, 2009.

[78] C. Metra, M. Favalli, and B. Ricco. On-line detection of logic errors due to crosstalk, delay, and transient faults. In *Proc. IEEE International Test Conference (ITC)*, pp. 524-533, 1998.

[79] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. In *ACM Computing Surveys (CSUR)*, 1999.

[80] P. Mahoney, E. Fetzer, B. Doyle, S. Naffziger. Clock distribution on a dual-core, multi-threaded Itanium-family processor. In *Proceedings of IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 292-293 Vol. 1, 2005.

[81] 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 IEEE/ACM International Conference on Computer-aided Design*, pp. 611-618, 2004.

[82] A. Chakraborty, K. Duraisami, A. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino. Dynamic thermal clock skew compensation using tunable delay buffers. In *Proceedings of IEEE Transactions on Very Large Scale Integration Sysmtes*, pp. 639-649, June 2008.

[83] A. DeHon. In-system timing extraction and control through scan-based, test-access ports. In *Proceedings of International Test Conference (ITC)*, pp. 350-359, 1994.

[84] Y. Elboim, A. Kolodny, and R. Ginosar. A clock-tuning circuit for system-on-chip. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 616-626, August, 2003.

[85] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. A post-silicon clock timing adjustment using genetic algorithms. In *Digest of Technical Papers of International Symposium on VLSI Circuits*, pp. 13-16, 2003.

[86] K. Nagaraj and S. Kundu. An Automatic Post Silicon Clock Tuning System for Improving System Performance based on Tester Measurements. In *Proceedings of IEEE International Test Conference (ITC)*, pp. 1-8, 2008.

[87] R. Ye, F. Yuan, H. Zhou and Q. Xu. Clock skew scheduling for timing speculation. In *Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 929-934, 2012.

[88] E. Alpaslan, J. Dworak, B. Kruseman, A.K. Majhi, W.M. Heuvelman, and P. van de Wiel. NIM- A Noise Index Model to Estimate Delay Discrepancies between Silicon and Simulation. In *Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pages 1373–1376, 2010.

[89] S. Sde-Paz and E. Salomon. Frequency and Power Correlation between At-Speed Scan and Functional Tests. In *Proceedings IEEE International Test Conference (ITC)*, paper 13.3, 2008.

[90] Z. Lak, and N. Nicolici. In-system and on-the-fly clock tuning mechanism to combat lifetime performance degradation. In *Proceedings IEEE/ACM International Conference on Computer-Aided Design*, pages 434-441, 2011.

[91] Y. Wang, H. Luo, K. He, R. Luo, Y. Xie, H. Yang. Temperature-aware NBTI Modeling and the Impact of Input Vector Control on Performance Degradation. In *International Conference on Design Automation and Test in Europe (DATE)*, 2007.

[92] B. Vaidyanathan, A. S. Oates, Y. Xie. Intrinsic NBTI-Variability Aware Statistical Pipeline Performance Assessment and Tuning. In *International Conference on Computer-Aided Design (ICCAD)*, 2009.