Chair of Electronic Design Automation

Advanced Timing for High-Performance Design and Security of Digital Circuits

Li Zhang

Vollständiger Abdruck der von der
Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs
genehmigten Dissertation.

Vorsitzende/-r: Prof. Dr. phil. nat. Sebastian Steinhorst

Prüfende/-r der Dissertation:

1. Prof. Dr.-Ing. Ulf Schlichtmann

2. Assoc. Prof. Yiyu Shi, Ph.D.

Die Dissertation wurde am 17.04.2018 bei der Technischen Universität München
eingereicht und durch die Fakultät für
Elektrotechnik und Informationstechnik am 13.08.2018 angenommen.

Anhang I

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die bei der
Fakultät für Elektrotechnik und Informationstechnik

der TUM zur Promotionsprüfung vorgelegte Arbeit mit dem Titel:
Advanced Timing for High-Performance Design and Security of Digital Circuits

in Chair of Electronic Design Automation
Fakultät, Institut, Lehrstuhl, Klinik, Krankenhaus, Abteilung

unter der Anleitung und Betreuung durch: Prof. Dr.-Ing. Ulf Schlichtmann ohne sonstige Hilfe erstellt und bei der Abfassung nur die gemäß § 6 Ab. 6 und 7 Satz 2 angebotenen Hilfsmittel benutzt habe.

☒ Ich habe keine Organisation eingeschaltet, die gegen Entgelt Betreuerinnen und Betreuer für die Anfertigung von Dissertationen sucht, oder die mir obliegenden Pflichten hinsichtlich der Prüfungsleistungen für mich ganz oder teilweise erledigt.

☒ Ich habe die Dissertation in dieser oder ähnlicher Form in keinem anderen Prüfungsverfahren als Prüfungsleistung vorgelegt.

☒ Die vollständige Dissertation wurde in _____ veröffentlicht. Die promotionsführende Einrichtung
Chair of Electronic Design Automation, Department of Electrical and Computer Engineering

hat der Veröffentlichung zugestimmt.

☒ Ich habe den angestrebten Doktorgrad noch nicht erworben und bin nicht in einem früheren Promotionsverfahren für den angestrebten Doktorgrad endgültig gescheitert.

☐ Ich habe bereits am _____ bei der Fakultät für _____

der Hochschule _____
unter Vorlage einer Dissertation mit dem Thema _____

die Zulassung zur Promotion beantragt mit dem Ergebnis: _____

Die öffentlich zugängliche Promotionsordnung der TUM ist mir bekannt, insbesondere habe ich die Bedeutung von § 28 (Nichtigkeit der Promotion) und § 29 (Entzug des Doktorgrades) zur Kenntnis genommen. Ich bin mir der Konsequenzen einer falschen Eidesstattlichen Erklärung bewusst.

Mit der Aufnahme meiner personenbezogenen Daten in die Alumni-Datei bei der TUM bin ich

☒ einverstanden, ☐ nicht einverstanden.

Ort, Datum, Unterschrift

Acknowledgments

Firstly, I would like to express my sincere gratitude to Prof. Schlichtmann. He gave me the opportunity to do research in the Chair of Electronic Design Automation at TUM. Within the past three years, I have learned a lot from his attitudes and professionalism towards research and teaching. Every time I discussed with him, he always gave me constructive advice and inspired me to explore novel ideas. I also thank him for giving me a chance to be a teaching assistant. This precious experience helped me a lot in teaching and communicating with students. Furthermore, he kindly considered my academic career and motivated me to attend the project evaluation to gain relevant experience. I am grateful for his help and support very much.

I would also like to thank Prof. Sebastian Steinhorst and Prof. Yiyu Shi for serving as the committee members. I appreciate their effort and time very much.

Within the three years, I have been also cooperating with Dr. Bing Li on the research topic of timing of digital circuits. He not only taught me how to be an excellent researcher, but also helped open the door to a whole new research identity for me. In every discussion with him, he never took existing techniques for granted and motivated me to explore the scientific development in a comprehensive way. I thank him for his philosophy of "thinking outside the box", which helped me to tread on unknown frontiers of research.

During the three years, I also cooperated with other professors, including Professor David Z. Pan, Professor Jiang Hu, Professor Masanori Hashimoto, Professor Yiyu Shi, Professor Bei Yu. They gave me invaluable suggestions about our cooperation projects and I appreciate their help and support very much.

Furthermore, I would like to thank Dr. Helmut Gräb, Dr. Daniel Müller-Gritschneder, Susanne Werner, Hans Ranke, Tobias Baur and other colleagues in the institute for their help and support. It is a great pleasure to work with them.

Last but not least, I give my deepest gratitude to my parents and my husband for their endless patience in the challenging but fascinating three years.

Munich, January 2018

Grace Li Zhang

Contents

Chapter 1

Introduction

Integrated Circuits (IC) are making our lives more convenient in every aspect. They are important drivers of technology innovation and economic progress. As one of the core performance metrics, timing is used to reflect how fast ICs can operate. In digital ICs, the timing performance is determined by their clock frequency. For several decades, the timing performance enhancement of processors has been driven by the improvement of manufacturing technology and the increasing number of pipeline stages. For example, shrinking transistor sizes result in smaller propagation delays for combinational gates and with pipelining long paths can be partitioned into short paths to reduce the clock period. Unfortunately, such straightforward enhancements have now mostly come to a stop.

As the manufacturing technology advances into the nanometer era, the shrinking transistor size results in undesirable side-effects. One of the major concerns are the increasing manufacturing deviations from the nominal specifications, because it is extremely difficult to control the fabrication process accurately. The deviations of process parameters, e.g., gate length, oxide thickness and doping profiles, cause variability in electrical parameters of integrated circuit devices, such as V_{th}. Accordingly, delays of combinational gates and interconnects have variations.

In addition to the traditional variations of parameters between dies and wafers, a new type of variations, within-die variations, have become a non-negligible component of total variations. These variations may have a low correlation, so that even two transistors located side-by-side might vary in their performance significantly. Accordingly, the traditional worst-case analysis cannot handle them very well.

For several decades, the worst-case analysis flow has been adopted in IC industry. Worst-case analysis indicates the worst-conditions for the values of the process parameters. With these parameters, the performance of the worst-case circuit is determined. Because worst-case analysis is very efficient in terms of reducing design

effort, it has become the most widely used method for timing analysis and verification. However, this method chooses each process parameter value independently, and ignores the correlations among these process parameters. Thus the worst-case performance obtained with this method is extremely pessimistic. This pessimism forces designers to optimize designs that may have met timing specifications further. This overdesign consumes significant resources to improve circuit performance, leading to an increase of design cost.

Circuit aging, another challenge, refers to the degradation of MOS device characteristics. It degrades circuit timing performance over time, shortens circuit lifetime and introduces potential timing failures into circuits. As aging becomes more prominent, this phenomenon can no longer be ignored and must be addressed to ensure timing performance of circuits. The major physical mechanisms of device aging include Hot Carrier Injection (HCI) and Negative Bias Temperature Instability (NBTI). These effects cause an increase of threshold voltage V_{th} or a decrease of channel carrier mobility, leading to a loss of performance over time.

As the fabrication technology enters the nanometer era, variations in manufacturing and increasing fragility of devices require us to examine the concepts in the traditional timing paradigm carefully. For example, the combinational logic blocks in a circuit perform computation and sequential components are used to synchronize the logic computation in the traditional paradigm. This paradigm is suffering from the performance limit due to the barrier of flip-flops. A new timing paradigm is required to break these barriers to improve circuit performance further.

1.1 Contributions of This Work

Facing the challenges of process variations and aging, in this thesis, a post-silicon tuning technique is investigated to adjust the timing properties of chips after manufacturing. This method can introduce customized clock skews individually for each chip with respect to the results of process variations. Therefore, the chips with timing failures after manufacturing might be rescued to meet the performance requirements.

To apply this post-silicon technique, tuning components must be inserted into the circuit during the design phase. A trade-off must be made between area overhead and yield improvement. To solve this problem, this thesis proposes an iterative learning method [ZLL+18] to determine where to insert post-silicon tuning buffers during the design phase. This method learns the buffer locations with a Sobol sequence iteratively and reduces the buffer ranges afterwards with tuning concentration and buffer grouping.

In addition, chips must be tested after manufacturing to counter process variations. Previous methods rely on path-wise frequency stepping, which is very time-consuming and causes expensive test cost. To solve this post-silicon test problem, an efficient delay test framework [ZLS+18] is proposed in this thesis to reduce test cost by testing only representative paths with delay alignment while taking advantage of the tunable buffers in the circuit.

The presented post-silicon technique tunes the circuit with additional hardware, e.g., post-silicon tuning buffers, to counter process variations. Flexible timing properties of flip-flops can also be used to mitigate the effects of process variations. However, in traditional static timing analysis, the timing properties of flip-flops, e.g., setup time, hold time and clock-to-q delay, are characterized as constants. In reality, the relation between setup time, hold time and clock-to-q delay of a flip-flop is a continuous function. In this thesis, a piecewise model is proposed to characterize the relation of the timing properties of flip-flops [ZLS16c]. With this characterization, the timing performance of circuits is improved.

In the traditional timing paradigm, sequential components are used to synchronize logic computation. However, they limit the circuit performance in two regards.

Firstly, they have inherent clock-to-q delays and impose setup time constraints. Secondly, delay imbalances between flip-flop stages cannot be exploited since signal propagations are blocked at flip-flops instead of being allowed to propagate through flip-flops. If these components are removed from the circuit, the delay compensation between flip-flop stages can be achieved automatically. To break the confines of the traditional timing paradigm, in this thesis, a new timing model is proposed to optimize the timing performance of circuits [ZLHS18]. In this new model, sequential components and combinational gates are both considered as delay units. Thereafter, a timing optimization framework is introduced to allocate sequential components at necessary locations, while the functionality of the circuit is maintained.

In the traditional timing model, a netlist of a digital circuit carries all its design information. Therefore, reconstructing the original netlist to counterfeit chips becomes very easy for a third party who may produce the chips illegally. In this thesis, a timing camouflage method is proposed to invalidate the assumption that a netlist completely represents the function of a circuit [ZLY+18]. With the help of wave-pipelining paths, this method forces attackers to capture delay information from manufactured chips, which is a very challenging task because false paths are also introduced.

1.2 Organization of This Dissertation

The structure of this thesis is as follows. The basic concepts of static timing analysis are introduced in Chapter 2. Process variations and the state-of-the-art methods to mitigate these variations are described in Chapter 3. Post-silicon tuning techniques are explained in Chapter 4. The interdependency of setup time, hold time and clock-to-q delay is modeled to exploit the flexible timing properties of flip-flops in Chapter 5. Chapter 6 introduces the new timing model, with which the timing performance can be improved even beyond the traditional limit. Afterwards, a timing camouflage method to improve circuit security against counterfeiting is demonstrated in Chapter 7. In the end, this thesis is concluded in Chapter 8.

1.3 Summary

As the manufacturing technology advances into nanometer technology, process variations are becoming more and more pronounced. Therefore, the traditional worst-case design methodology to handle these variations is too pessimistic. As a solution, a post-silicon tuning technique is introduced to adjust the timing properties of chips after manufacturing. The challenges of this techniques are investigated in this thesis. Exploiting the flexible timing properties of flip-flops, as another technique to counter process variations, is presented. However, these techniques are still confined within the traditional timing paradigm, where the performance of a circuit has reached a limit. Therefore, a new timing model is also proposed to improve the circuit performance even beyond the traditional limit. To secure digital circuits, the traditional assumption that the netlist represents the function of a circuit is invalidated with the proposed timing camouflage method.

Chapter 2

Timing of Digital Circuits

As mentioned in the previous chapter, the timing performance is one of the core performances of digital circuits. In this chapter, the basic timing concepts of digital circuits and several existing timing optimization methods are reviewed.

2.1 Sequential Circuits

Generally, digital circuits can be classified into two categories, namely, combinational circuits and sequential circuits. Although combinational logic circuits constitute the infrastructure of digital circuits, sequential circuits are used more often in practice because they can provide more complex functions than combinational circuits.

In combinational circuits, the output is a function of only the present input. Sequential circuits involve memory elements along with combinational logic. Fig. 2.1 shows the typical structure of sequential circuits, where the output of memory elements is fed back into the input of combinational circuits. With this structure, the output of a sequential circuit is controlled by not only its current input but also the present state of the circuit. The memory elements store data at their inputs when the clock signal is valid.

Flip-flops and latches are basic memory elements in sequential circuits. Only at the predefined clock edges can the data at the input of a flip-flop be transferred to its output. Accordingly, a flip-flop is normally called edge-triggered. Different from flip-flops, the data at the input of a latch can be transferred to its output when the clock signal is active, high logic or low logic based on the type of a latch. Accordingly, a latch is called level-triggered. In this section, flip-flop based circuits are firstly introduced, and then latch based circuits are described.

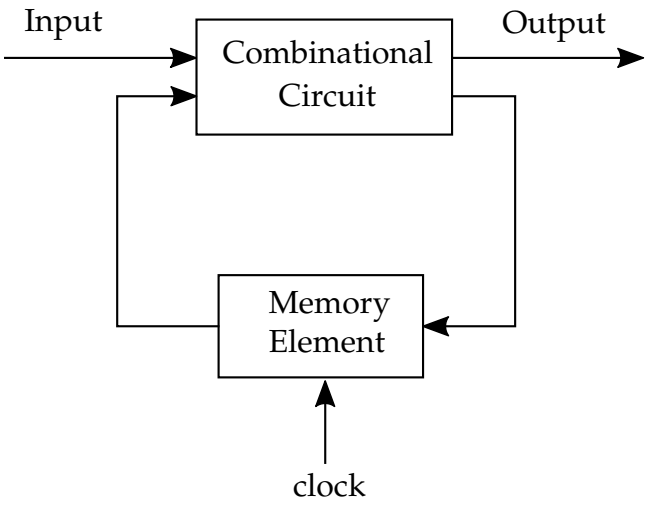

Figure 2.1: The structure of sequential circuits.

In sequential circuits, if flip-flops are used to store information, these circuits are called flip-flop based circuits. They are the most popular circuit type adopted in industry because of the their simple design and timing verification.

In flip-flop based circuits, flip-flops are used to synchronize signal propagations. Consequently, these propagations are blocked at flip-flops until a clock edge arrives. The data at the inputs of flip-flops are transferred to their outputs at each active clock edge, assumed as rising clock edge in the rest of this thesis. With flip-flops, combinational logic blocks are isolated naturally. In this way, designers only have to guarantee that the logic functions of combinational blocks are correct without having to worry about the interconnections between different stages. This design style reduces design efforts significantly.

To guarantee the correct function of flip-flop based circuits, timing constraints should be satisfied for each pair of flip-flops. One of them is the setup constraint, which describes the scenario where the signal at the input of the source flip-flop should not propagate too slowly to arrive at the sink flip-flop. Another constraint requires that the signal arriving at the input of a flip-flop should not be too fast to affect the latching data at the flip-flop. These timing constraints will be explained further in Section 2.2.1.

Similar to flip-flops, latches can also be used as memory elements to design sequential circuits. This type of circuit is called latch based circuit. The design of latch

based circuits is much more difficult than that of flip-flop based circuits due to the inherent flexible property associated with latches. Accordingly, latch based circuits are only used in high-performance designs.

In this thesis, latches are assumed to be active when the clock signal is high. The data at the input of a latch is locked into this latch when the clock signal switches from high to low. Accordingly, the falling edge is called *latching edge* of a latch. On the contrary, the data at the input of a latch can propagate when the clock signal switches from low to high. The rising edge is called *enabling edge* of a latch. The phenomenon that the input signal changes result in immediate changes at the output is called latch transparency. This property of latch based circuits is the essential difference compared with that of flip-flop based circuits, where the path delay between pairs of flip-flops should be smaller than the clock period. Accordingly, timing analysis of latch based circuits is also more complicated than that of flip-flop based circuits.

2.2 Timing of Sequential Circuits

In this section, timing constraints and static timing analysis of sequential circuits will be described. These constraints are used to guarantee correct functionality of circuits. Firstly, timing constraints and analysis of flip-flop based circuits are introduced. Afterwards, those of latch based circuits are described.

2.2.1 Timing of Flip-flop Based Circuits

Fig. 2.2 illustrates the concept of timing constraints of flip-flop based circuits, where two flip-flops are connected by a combinational circuit. Assume that at the reference time 0, the signal at the input of FFi is transferred to its output and this signal starts to propagate to the input of FFj at the next stage. To guarantee the correct function of FFj, the data must be stable t_{su} time before it is latched by FFj at the next rising clock edge at time T. Therefore, the following inequation should be satisfied

$$t_{cq,i} + \overline{d}_{ij} + t_{su,j} \leq T \tag{2.1}$$

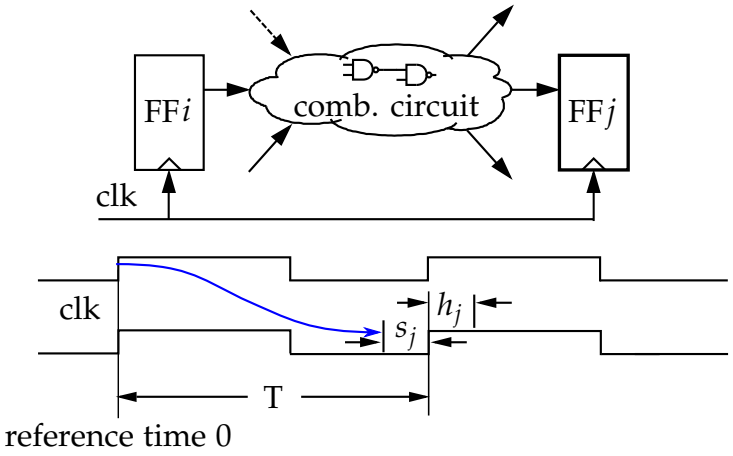

Figure 2.2: Timing constraints of flip-flop based circuits.

where $t_{cq,i}$ is the propagation delay from clock signal to the output of FFi, $\overline{d}_{ij}$ is the maximum delay of combinational paths between FFi and FFj, $t_{su,j}$ is the setup time of FFj and T is the clock period. For each pair of flip-flops in sequential circuits, (2.1) should be satisfied to guarantee the correct function.

To guarantee the data captured at FFj correctly, the data should also be stable t_h time after the next rising edge. t_h indicates that there should not be any change in the input data at FFj between the next rising edge T and T+t_h. Therefore, the earliest arrival time from FFi should satisfy the following inequation

$$t_{cq,i} + \underline{d}_{ij} \geq t_{h,j} \tag{2.2}$$

where $\underline{d}_{ij}$ is the minimum delay of combinational paths between FFi and FFj and $t_{h,j}$ is the hold time of FFj. Hold time constraints defined in (2.2) should be satisfied between all pairs of flip-flops in sequential circuits.

The maximum delay between all pairs of flip-flops in a sequential circuit determines the minimum clock period the circuit can work with. To calculate the maximum delay between a pair of flip-flops, the combinational circuit between two flip-flops should be traversed. Path-based and block-based traversals are two methods to calculate the maximum delay between pairs of flip-flops. The path-based method enumerates all paths from inputs to outputs of a circuit by summing the gate delays along a path together. Accordingly, it is suitable to evaluate circuits with a smaller number of paths. Instead of computing the delays of paths, the block-based method

visits each combinational gate in a circuit only once. Therefore, the runtime efficiency is much higher than that of the path-based method. In the following, the block-based timing analysis method is explained.

Algorithm 1 shows the pseudo code of the static timing analysis for flip-flop based circuits as an example of timing analysis algorithms. For simplification, all outputs of flip-flops are regarded as primary inputs and all inputs of flip-flops as primary outputs. Lines 6-14 initializes the arrival times at the primary inputs to be predefined values, e.g., the propagation delays of flip-flops. Thereafter, all combinational gates whose fanins are only primary inputs are appended to Q. The main loop (lines 15-27) processes these gates in iteratively. In every iteration, a combinational gate is taken from Q as the current gate. The arrival time of this gate is the maximum value of all input arrival times plus the pin-to-pin delay of this gate. Afterwards, the fanout gates are checked. Only when all the fanins of a fanout gate have been already visited can it be appended to Q. The main algorithm terminates when Q is empty, indicating the arrival times at the primary outputs of a circuit are already calculated. With the sum of the maximum arrival times at all primary outputs and setup time, the maximum delay of the circuit is obtained. This delay represents the minimum clock period without setup time violations for flip-flop based circuits.

The minimum delay at each primary output can be computed with the min operation in Algorithm 1. The minimum delay should be larger than the hold time of the flip-flops to guarantee the correct function when the circuit works with the maximum clock frequency obtained from Algorithm 1.

2.2.2 Timing of Latch Based Circuits

Timing of latch based circuits is more complicated than that of flip-flop based circuits because of the flexible timing properties of latches. Similar to flip-flops, the arrival time at the input of each latch must satisfy the setup time constraint. To formulate this constraint, we assume all arrival times of latches are with respect to the *local time zone*. In this thesis, the start time of the local time zone for each latch is assumed to be the time at which the clock signal switches from high to low, while the latch is in transparency when the clock signal is 1.

Algorithm 1: Timing Analysis of Flip-flop Based Circuits.

```
   // variables
1  T_Min: minimum clock period (initialized to 0);
2  Q: FIFO-like queue of combinational gates to be visited;
3  n_*: primary inputs; m_*: primary outputs;
4  c_*: combinational gates; A_*: arrival times;
5  W_ij: the pin-to-pin delay of gate i;
   // algorithm initialization
6  foreach primary input n_i do
7      set arrival time of n_i;
8      mark n_i as visited;
9      foreach fanout gates c_j of n_i do
10         if all fanin of c_j are primary inputs then
11             append c_j to Q;
12         end
13     end
14 end
   // main loop
15 while Q is not empty do
16     c_i ← head of Q; A_i ← 0;
17     foreach fanin n_j or c_j of c_i do
18         A_t ← A_j + W_ji;
19         A_i ← max{A_t, A_i};
20     end
21     mark c_i as visited;
22     foreach fanout gates c_j of c_i do
23         if all fanin of c_j are visited then
24             append c_j to Q;
25         end
26     end
27 end
28 foreach primary output m_j in the circuit do
29     A_j ← arrival time of the fanin gate;
30     A_t ← A_j + t_su,j;
31     T_Min = max{T_Min, A_t};
32 end
```

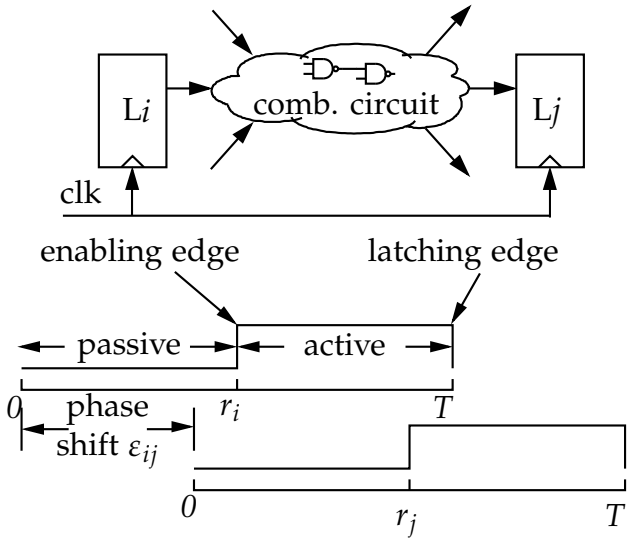

Figure 2.3: Local time zone and clock phase shift.

Fig. 2.3 illustrates a latch based circuit with two latches i and j and their clock phases. ε_{ij} is the *phase shift* of the two clock phases for latch i and j. The start propagation time of a data signal at the input of a latch is called *departure time* and represented as D_i, which is defined to the local time zone of latch i. The latest arrival time A_j is calculated as

$$A_j = D_i + t_{cq,i}/t_{dq,i} + \overline{d}_{ij} - \varepsilon_{ij} \tag{2.3}$$

where $-\varepsilon_{ij}$ transforms A_j to the local time zone of j. $t_{cq,i}$ is the propagation delay from clock signal to the output of latch i and $t_{dq,i}$ is the propagation delay from data to the output of latch i. $t_{cq,i}/t_{dq,i}$ is determined according to whether a data signal at the input of latch i starts propagation at the enabling clock edge or during the transparency.

In Fig. 2.3, the enabling clock edge of latch i is represented as r_i. Only after r_i can a data signal can propagate to the next latch stage. Accordingly, the latest departure time of i can be expressed as follows

$$D_i = \max\{A_i, r_i\}. \tag{2.4}$$

By substituting D_i in (2.3) with $\max\{A_i, r_i\}$, (2.3) can be expressed as follows,

$$A_j = \max\{A_i, r_i\} + t_{cq,i}/t_{dq,i} + \overline{d}_{ij} - \varepsilon_{ij}. \tag{2.5}$$

To guarantee the data signal at the input of the latch j to be latched correctly, this

input signal must be stable at least $t_{su,j}$ time before the latching clock edge. Accordingly, the following timing constraint for latch j should be met.

$$A_j + t_{su,j} \leq T. \tag{2.6}$$

The definition for hold time analysis can be explained similarly. The earliest arrival time a_j at latch j can be computed as

$$a_j = d_i + t_{cq,i}/t_{dq,i} + \underline{d}_{ij} - \varepsilon_{ij}. \tag{2.7}$$

Only after r_i can a data signal can propagate to the next latch stage. Accordingly, the earliest departure time of latch i can be expressed as follows

$$d_i = \max\{a_i, r_i\}. \tag{2.8}$$

Accordingly, the earliest arrival time at latch j becomes

$$a_j = \max\{a_i, r_i\} + t_{cq,i}/t_{dq,i} + \underline{d}_{ij} - \varepsilon_{ij}. \tag{2.9}$$

To guarantee the data to be latched correctly, the following hold time constraint should be satisfied

$$a_j \geq t_{h,j}. \tag{2.10}$$

The minimum clock period in (2.6) for latch based circuits cannot be identified directly because latch transparency makes the arrival times depend on each other. To deal with this dependency, a method was proposed in [SMO90b], where the constraints in (2.4) and (2.6) are relaxed to find the minimum clock period. The timing constraints in latch based circuits will be used in the Section 6 to perform timing optimization.

2.3 Timing Optimization Methods for Sequential Circuits

In this section, several existing timing optimization methods for improving the timing performance of sequential circuits are reviewed.

2.3.1 Gate Sizing

Gate sizing is one of the most popular methods for optimizing digital circuits. In gate sizing, scaling factors of combinational gates are selected to improve objectives such as clock frequency and area efficiency, while timing constraints between flip-flops are satisfied. It is usually adopted in the overall design flow to fix timing violations. For instance, it is used after placement to resize gates that violate the rules of maximum fanout or setup time requirements of flip-flops. After routing, it is used to fix setup and hold violations with the layout information.

Gate sizing has been investigated extensively in the literature. For example, [CCW98] introduces a fast and exact algorithm for simultaneous gate and wire sizing to minimize total area and propagation delay inside a circuit. In [OBH11], a Lagrangian Relaxation (LR) based formulation together with a graph model is proposed to optimize timing slacks and power consumption simultaneously. In [HKK+12], a meta-heuristic approach is developed to size logic gates that have the greatest impact on power-performance tradeoff. This method guarantees slack, capacitance and slew constraints throughout the optimization process. In addition, [LKLZ12] presents a framework for cell-type selection with a further extension of the traditional Lagrangian Relaxation to match discrete gate sizes, together with a min-cost network flow method to optimize power consumption.

2.3.2 Retiming

Retiming is another technique for optimizing digital circuits. It repositions the sequential components, e.g., flip-flops, in a circuit without moving the combinational portion. The objective of retiming is to find a circuit with the minimum number of sequential components for a given clock period. There are usually two variants in this method. The first one is minimizing the clock period without considering the number of sequential components in the final circuit. The second one is minimizing the number of sequential components in the final circuit with no constraints on the clock period.

Retiming has been investigated extensively for over a decade. In [LZ06], an efficient algorithm is proposed to retime sequential circuits under both setup and hold

constraints. The work in [HMB08] demonstrates a maximum-flow-based approach to minimize the number of flip-flops. In [WZB17], a new retiming method with a network-simplex algorithm is introduced for two-phase latch-based resilient circuits to reduce the overhead of normal and error detecting latches. Retiming for FPGA has been investigated in [SMB05] to meet architecture constraints such as avoiding flip-flops through carry chains to guarantee a correct circuit function.

2.3.3 Clock Skew Scheduling

Clock skew scheduling is a technique that intentionally introduces skews to sequential components, e.g., flip-flops, to reduce the clock period. Similar to retiming, clock skew scheduling can also balance the timing slacks between flip-flop stages to improve the circuit performance. Retiming achieves delay compensations at gate level. However, skew scheduling deals with delay imbalances at a finer level of granularity. Retiming and clock skew scheduling can be combined together to reduce the clock period. Intentional clock skews can be introduced statically with methods, such as different wire-interconnect lengths from clock signal to sequential components. After manufacturing, adjusting clock skews for each chip dynamically can be achieved with methods, such as post-silicon tuning, at the expense of area overhead.

In [Fis90], clock skew scheduling is investigated to improve the circuit performance, e.g., minimizing the clock period or maximizing the safety margin for a given clock period. To tolerate process variations, in [NF96], a graph-based algorithm is proposed to incorporate process-dependent delays when determining the minimum clock period. [KK17] exploits the useful clock skew scheduling with adjustable delay buffers allocated at the flip-flops.

2.3.4 Wave-Pipelining

Wave-pipelining is another method to improve circuit performance, where logic waves are allowed to propagate through combinational paths without intermediate sequential components. Wave-pipelining provides a method to make the clock frequency of a circuit independent of the largest path delay, which limits circuit performance in traditional circuit designs. This technique has been explored in the

context of circuit design, where the numbers of waves on logic paths should be defined and their synchronization should be maintained by designers during the design phase. In wave-pipelining based design, logic design and timing cannot be handled separately. Consequently, it is incompatible with the traditional fully synchronous design.

As early as in [JC93], a linear method to minimize the clock period using wave pipelining is proposed. The work in [BCKL98] introduces the fundamentals of wave-pipelining and investigates wave-pipelined VLSI chips and CAD tools for designing wave-pipelined circuits. Recently, this method is also explored for majority-based beyond-CMOS technologies to improve the throughput of majority inverter graph (MIG) designs in [ZMT+17].

2.4 Summary

The clock frequency determines the timing performance of sequential circuits. In order to evaluate the timing performance, static timing analysis is commonly used. The clock period of the circuit is determined by the maximum delays between pairs of flip-flops and setup times without hold time violations. The static timing analysis methods are a fundamental part of the timing analysis considering process variations in the following chapters.

In the traditional timing paradigm, four methods are usually adopted to improve the timing performance of digital circuits. Gate sizing, retiming and clock skew scheduling can be used separately or jointly. The combined methods achieve an aggressive optimization by balancing the timing slacks with the smallest combinational delay in the library. Wave-pipelining provides a way to balance the timing slacks between several flip-flops stages. However, it is not compatible with the traditional timing paradigm.

Chapter 3

Background and Problem Description

Traditional static timing analysis evaluates the timing performance of a circuit by setting the process parameters to their worst/best corners. With the process variations more pronounced, it is intractable to analyze all possible corners. Furthermore, the corner-based analysis is pessimistic in evaluating the circuit performance. Circuit aging also affects the timing properties of circuits. To counter process variations and aging, a large timing margin is usually reserved. As aging effects becomes more prominent, this conservative approach is increasingly less feasible. In this chapter, variations and aging are discussed firstly. Thereafter, existing methods to alleviate them are investigated.

3.1 Process Variations and Aging

Process variations are deviations of process parameters, e.g., device gate length and oxide thickness, from their nominal specifications after manufacturing. These variations result from the fact that the semiconductor fabrication process cannot be precisely controlled. The deviations of process parameters cause variability in electrical parameters of integrated circuit devices and wires. Accordingly, delays of combinational gates and interconnects have variations. As the manufacturing technology advances into nanometer technology nodes, process variations are becoming more and more pronounced and affect circuit performance significantly.

The traditional method to deal with process variations is the worst-case design where delays of long paths are overestimated and those of short paths are underestimated. This method, however, leads to a large unnecessary margin reserved in circuits. With the increasing process variations, it cannot be applied any more due to the underestimation of circuit performance.

To reduce the pessimism in the worst-case design, process variations should be firstly investigated and modeled accurately instead of using the worst-case design method. Afterwards, the analysis of process variations can be incorporated into design optimization methods to improve yield and profit of circuits.

3.1.1 Sources of Process Variations

Generally, variations result from the inaccurate control of semiconductor manufacturing process. Various sources of variations affect characteristics of wafers and dies on them during the fabrication process. In this section, the sources of variations are analyzed from the perspective of integrated circuit devices and interconnects.

The variations of integrated circuit devices are typically categorized into two types: device geometry variations and device material parameter variations. The first type affect the physical geometric structure of devices. They include film thickness variations and lateral dimension variations. Film thickness variations occur primarily from wafer-to-wafer or die-to-die control and cause deviations of gate oxide thickness. Lateral dimension variations result from photolithography proximity effect, lens, mask, plasma etch dependencies, or photo system deviations [BN00]. They cause deviations of gate length, which is critical dimension (CD) because it affects gate delay primarily. The second type, device material parameter variations, are related to internal material parameters. For example, doping variations affect junction depth and dopant profiles, leading to variability in effective channel length. There also exist variations during the processes of deposition and anneal, resulting in variability in contact and line resistance.

Similar to devices, the variations of interconnects also include geometry variations and material parameter variations. Deviations in line width and spacing due to photolithography and etch dependencies impact line resistance and capacitance. In addition, the fluctuations during CMP process affect the thickness of metal lines. These fluctuations make electrical characteristics of interconnects, e.g., capacitance, differ from nominal specifications. There are also variations in material property. For example, metal resistivity varies from wafer to wafer and the dielectric constant is affected by the deposition process.

3.1.2 Categories of Process Variations

Generally, there are two groups of process variations: systematic and non-systematic variations. Systematic variations can be obtained after analyzing the design layout. Accordingly, they can be considered as deterministic values during the design process. Different from systematic variations, non-systematic variations can not be predicted before manufacturing. They are also called random variations, indicating the uncertainty of variations in physical parameters. Therefore, they are usually modeled as statistical variables during the design process.

Non-systematic variations can be further classified into die-to-die variations and within-die variations according to the spatial scales of variations. Die-to-die variations affect all devices and interconnects equally. For example, they cause gate length of all devices on one chip to vary in a similar way. In contrast, within-die variations affect each device on the same die differently. For instance, some devices on a die have a smaller gate length, whereas other devices on the same die have a larger gate length than design values.

Furthermore, within-die variations can be classified into spatially correlated and independent variations. The former exhibit spatial dependence. For example, some processes affect devices that are located closely to each other in a similar way. Accordingly, they can be modeled by establishing correlation between these variations. However, independent variations do not exhibit spatially dependent correlation. They are different for all devices. They result from the inaccuracy of manufacturing equipments and process control and happen in nearly every processing step.

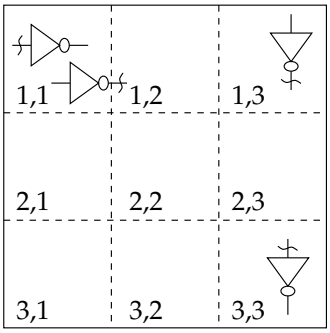

Figure 3.1: Uniform Grid Correlation Model [CS03].

3.1.3 Correlation Modeling

Process variations can be decomposed into different variables [SBC97]. Accordingly, they can be modeled by summing these decomposed variables. Die-to-die variations result in global correlation or die-to-die correlation for parameters of the devices and interconnects. Within-die variations demonstrate proximity correlation, indicating that the distance between two devices on the die determines the correlation [CCBC06]. For example, if the distance between two devices is large, the correlation between their parameters is small.

The widely deployed correlation model for within-die variations is proposed in [CS03]. In this model, uniform grids are used to partition the die area, illustrated in Fig. 3.1. A random variable is assigned for each grid cell. Therefore, n variables are assigned when there exist n grid cells. The correlations among the n random variables are computed with methods, e.g., [XZH07]. To reduce the complexity in handling correlated variables, principal component analysis (PCA) [Jol02] is deployed to make decomposition of the n correlated variables into linear combinations of independent random variables. After the decomposition, only the independent variables with large coefficients are maintained in the linear combinations. Accordingly, the number of variables modeling correlation can be reduced significantly. For example, a process parameter p after decomposition can be represented as follows

$$p = p_0 + p_g + \boldsymbol{p_l} + p_r \tag{3.1}$$

where p_0 is the nominal value of the process parameter. p_g models the die-to-die variation and is shared by all devices. $\boldsymbol{p_l}$ is the vector of independent random

variables after decomposition of correlated random variables. p_r is an independent variable modeling the purely random effect during manufacturing processes.

To evaluate the circuit performance under process variations, gate delays should be modeled as random variables based on the modeling of process variations. The correlation between process variations leads to correlated gate delays. For instance, the delays of two gates differ from the nominal specifications in a similar way when these gates are located nearby on the die. If their distance is large, delays of both gates demonstrate more randomness. In order to consider the effect of the correlation from process variations on the gate delays, these delays are represented as functions of process parameters.

The widely used delay description method uses linear functions [CS03, VRK+04]. Assume n process parameters affect gate delay. Therefore, a gate delay in this method is formulated as follows

$$D = \sum_{i=1}^{n} kp_0 + \sum_{i=1}^{n} kp_g + \sum_{i=1}^{n} \boldsymbol{k}\boldsymbol{p}_l + \sum_{i=1}^{n} kp_r \tag{3.2}$$

$$= D_0 + \sum_{i=1}^{m} d_i v_i + d_r v_r \tag{3.3}$$

where k and $\boldsymbol{k}$ are the coefficients and the coefficient vector. The gate delay in (3.2) can be generalized into the canonical linear delay form [VRK+04] as in (3.3), where m is the number of independent random variables after decomposition, v_i are independent random variables shared by all gate delays. v_r is the purely random variable specific for each delay. D_0 is the nominal value of the delay. d_i and d_r are the coefficients of the random variables. The correlations between gate delays are represented by sharing the same set of random variables v_i.

With the canonical delay model (3.3), arrival times can be propagated very fast with simple computations [VRK+04]. In the following chapters, this delay model is used to represent gate delays.

3.1.4 Circuit Aging

Another challenge that has emerged as the manufacturing process has shrunk to nanometer technology is circuit aging. Several aging phenomena affect the performance of transistors prominently. They are Hot Carrier Injection (HCI), Negative

Bias Temperature Instability (NBTI), Electromigration (EM), Timing Dependent Dielectric Breakdown (TDDB) and Positive BTI (PBTI).

The analysis of aging is complicated because aging is sensitive to many factors, such as process parameter variations, temperature, frequency and supply voltage. In the traditional method, a safety margin is reserved to alleviate the effect of aging. However, this method is increasingly infeasible because it is not a viable solution to tolerate a large timing margin as aging has become more pronounced. Therefore, it is desirable to analyze the aging effect of a circuit specifically.

To estimate aging effects, the traditional method adopts accurate transistor-level simulations. However, this method is not efficient for analyzing large circuits. To deal with this problem, researchers have investigated many methods to analyze HCI and NBTI and the factors influencing them. In addition, timing models and algorithms for aging analysis on gate level has been developed [BM09, CWT11, KKS06, KKS07, PKK+06, KBW+14, AKGH16, KME+16]. Among these methods, the AgeGate model [LGS09, LBS10, LBS12] is built on the canonical delay model [VRK+04] so that it can be integrated into standard timing signoff flows [KS15]. The aging analysis is accelerated further in [LBS14] while maintaining a good accuracy.

In the design phase, the aging effect still has to be considered statistically because both process variations and aging affect chips individually in the manufacturing process. To counter aging effects actively, post-silicon tuning techniques can also be applied to adjust the timing properties of individual chips. Such techniques include body bias tuning [KSB06, GLL+15], voltage control [And05, KCL+17, KLS+15] and clock tuning [NSG+06, TZC05, LN14].

3.2 Timing with Process Variations

With the increasing process variations at nanometer technology nodes, it is a challenging task to verify timing of circuits before manufacturing. In addition, the chips might not work with the designated clock period after manufacturing, leading to yield loss. To deal with these variations, methods from design phase to test phase have been proposed in the last several decades [SHJL16, LHS18]. In the following subsections, these methods are reviewed to show how they meet the challenges imposed by process variations.

3.2.1 Traditional Corner-based Design Method in Digital Circuits

Traditional static timing analysis method depends on specific process corners. Two letters are usually used to describe different corners, where the first letter represent the NMOS device and the second refers to the PMOS device. For example, SS represents a process corner with slow PMOS and slow NMOS. TT, SS, FF, SF and FS are commonly used five corners for performance analysis. Interconnect parasitics are extracted at multiple corners similarly. To guarantee the correct function of designs throughout the process range, hold time violations are verified at the FF corners and setup time violations are verified at the SS corner. Accordingly, corner-based timing analysis is pessimistic due to the overestimation of delays of long paths and underestimation of delays of short paths.

At submicron manufacturing technology nodes, process variations are becoming more pronounced. To handle these variations, an exponential number of corners are required as more complicated process effects occur. However, it is very difficult to analyze all possible corners. Therefore, some missing corners may cause failures of circuits after manufacturing. Furthermore, within-die variations cannot be ignored any more. The common method to deal with these variations is to use a predefined delay scaling factor for all circuit elements. For example, delay is increased for long-path analysis and is decreased for short analysis. However, this method causes overly pessimistic analysis because the factor is set to the worst-case within-die variations.

3.2.2 Statistical Static Timing Analysis

The demand for an effective modeling of process variations in timing analysis leads to the boom of research on statistical static timing analysis (SSTA). Instead of using deterministic values for gate or interconnect delays, SSTA models them as random variables with known probability distributions. Therefore, delay distributions are propagated to determine the probability distribution of circuit performance in SSTA. Existing research on SSTA includes [VRK+04, CS05, SS08, BCSS08, LKS+08, LCS09a, LCS+09b, LCS10, LCS11, LCS12, LCXS13, KLS+15, LS15]. They are generally classified into two categories: path-based statistical timing analysis and block-based statistical timing analysis.

In path-based statistical timing analysis, a set of paths are selected which have significant probability of being critical and their delay distributions are calculated by summing the delays of all gate and interconnect delays along these paths. The circuit delay is then estimated with a statistical maximum operation over all path delays. The method has the advantage of separating the computation of path delays and the statistical maximum operation over these path delays. However, it is not clear how to select paths with significant probability of being critical. Additionally, the number of these critical paths is large in well-balanced circuits, leading to expensive computation. Furthermore, it is not suitable for incremental timing analysis where changes in circuits should be dealt with incrementally and efficiently.

Instead of enumerating paths in a "depth-first" way, block-based statistical timing analysis traverses the circuit in a "breadth-first" manner. It propagates signal arrival times at circuit nodes using sum and max operations, resulting in a runtime that is linear with circuit size. Unlike path-based statistical timing analysis, block-based statistical timing analysis is computationally efficient and is capable of incremental analysis. However, it is not trivial to calculate the statistical maximum of two correlated arrival times.

Although SSTA has obtained extensive interest in recent years, it has not been used widely in industry. On one hand, it is too complicated, especially with realistic distributions instead of Gaussian distributions, which are widely used in SSTA. Furthermore, the effort for the required characterization of standard cell libraries can become significant. On the other hand, traditional deterministic STA is enhanced by

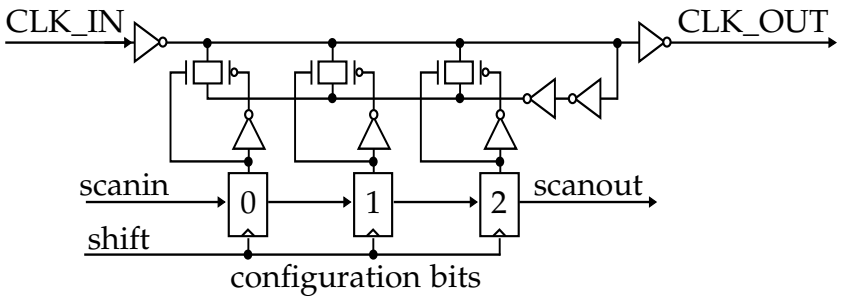

Figure 3.2: Post-silicon delay tunable buffer in [NSG+06].

incorporating sensitivities and correlation. Accordingly, the benefit of SSTA is still not certain.

3.2.3 Post-Silicon Tuning to Mitigate Process Variations

Post-silicon tuning is another technique to counter process variations. To apply this technique, tunable components are inserted into the circuit during the design phase. After manufacturing, chips with timing failures can be rescued by tuning buffers with respect to the effect of process variations, which become deterministic at this phase.

A widely used post-silicon tuning technique is clock tuning using delay buffers. Post-silicon tunable buffers have various structures [TRND+00, TKMH04, MFDN05, NSG+06]. The structure of the tunable buffer used in [NSG+06] is demonstrated in Fig. 3.2, where the three registers control the delay between the clock input CLK_IN and output CLK_OUT. During the design phase, tunable buffers of such type are inserted onto the clock paths of selected flip-flops related to potential critical paths. After manufacturing, the delay values of these buffers are adjusted through the test access port (TAP) to allot critical paths more timing slack by shifting clock edges toward stages with smaller combinational delays. Since critical paths might be different in individual chips due to process variations, post-silicon tunable buffers can be configured in each failed chip to counterbalance them efficiently. With post-silicon tuning, chips that failed to meet timing specifications can be revitalized.

In recent years, several methods have been proposed to determine buffer locations and evaluate the potential resulting yield improvement. A clock scheduling method is developed in [TBCS04] and tunable buffers are selectively inserted to balance the

skews resulting from process variations. In [TZC05] algorithms are proposed to insert buffers into the clock tree to guarantee a given yield, while minimizing the total area of these tunable buffers or the total number of them. In [KK17] the buffer allocation problem is solved with a graph-based algorithm under useful clock skew scheduling. Yield loss due to process variations and the total cost of tunable buffers are formulated together for gate sizing in [KS07]. A machine learning method is proposed in [YZLS17] to identify the locations of buffers. In [NK09], the placement of tunable buffers is investigated and a considerable improvement is observed when the clock tree is designed using the proposed tuning system. With the locations of tunable buffers known, the improved yield of the circuit can be evaluated efficiently using the method in [LCS11, LS15].

Since post-silicon tunable buffers take die area and require special treatment during physical design, the number of these buffers should be small to provide a good yield/profit improvement. This buffer insertion at the design phase is essentially a statistical optimization problem when process variations are considered. Previous methods [TZC05, KS08] solve this problem by path search or the cutting plane method. In these methods, yield values of different combinations of buffer locations are evaluated using Monte Carlo simulation. New combinations of buffer locations are then selected to evaluate according to the yield gradient. This is in fact a statistical extension of linear programming. Since Monte Carlo simulation is used at many branching points, this direct extension requires a large runtime to determine buffer locations, though the calculated buffer locations may still fall into a local optimum in the problem space due to the nature of path search.

After manufacturing, necessary information, e.g., delays of combinational paths, need to be extracted from chips with timing failures for post-silicon clock skew scheduling. In [LN14] an efficient post-silicon tuning method is proposed to search a configuration tree together with graph pruning and buffer grouping. The methods in [NK08, TGB09] measure path delays individually in manufactured chips and tune them accordingly. Furthermore, this post-silicon tuning technique has been applied for on-line adjustment to improve lifetime performance of a circuit in view of process variations and aging [YYX11, LN12]. Moreover, the method in [CDS+08] applies tunable buffers to compensate dynamic delay uncertainty induced by temperature variations. However, delay measurement in these methods is still performed by ap-

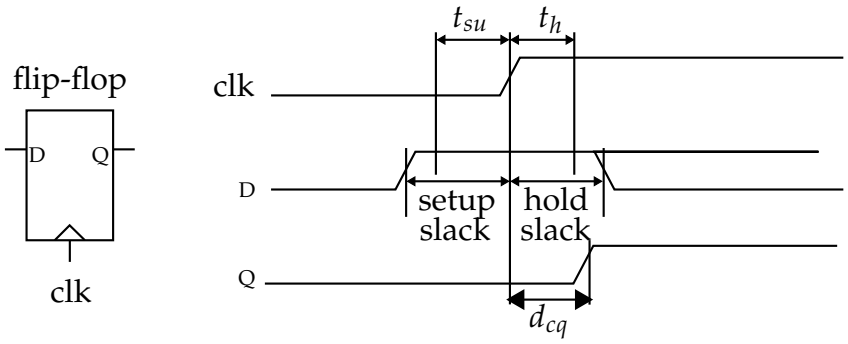

Figure 3.3: Setup time (t_{su}), hold time (t_h) and clock-to-q delay(d_{cq}) of a flip-flop. *Setup slack and hold slack* are defined as the distance from the latest and earliest signal switching to the active clock edge, respectively.

plying frequency stepping to individual paths [LN14,NK08,TGB09], which requires much time from expensive testers.

To apply the post-silicon tuning, a new technique will be introduced to insert tunable components into circuits during the design phase in Section 4.1. In Section 4.2, how to reduce the test cost to configure the tunable components will be presented.

3.2.4 Setup Time and Hold Time Characterization in Static Timing Analysis

Facing challenges from process variations, traditional definitions of timing concepts in STA should be examined. In STA, combinational circuit stages between flip-flops are assumed to be independent, and a flip-flop can work reliably if its setup time and hold time constraints are met [Pat90, Roe03]. With this assumption, only the worst/best corners (late and early modes) need to be verified with the largest/smallest combinational delays. However, in reality the relation between setup time, hold time and clock-to-q delay is a continuous function. Therefore, a smaller setup time allows the critical paths before a flip-flop more timing budget in case no timing violations are incurred at the next stage. These characteristics give the circuit a chance to balance the slacks between successive flip-flop stages. Therefore, the effect of process variations can be alleviated using the flexible timing characteristics of flip-flops. In this section, the simplified timing model of a flip-flop is reviewed. Afterwards, the interdependency between clock-to-q delays, setup

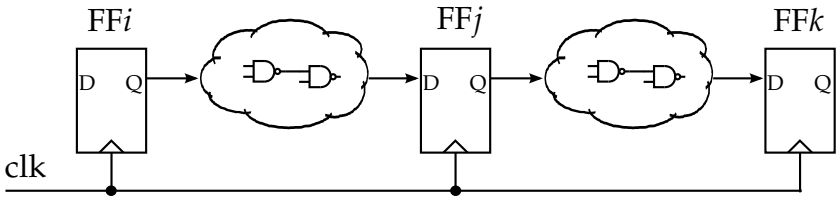

Figure 3.4: Circuit example with three flip-flops and two combinational circuits.

times and hold times is discussed.

In STA, a flip-flop is characterized by its setup time, hold time, and clock-to-q delay. These characteristics can be explained using the example in Fig. 3.3. If the latest signal switching at the input of the flip-flop is t_{su} time before it is sampled by the active clock edge, and if the earliest signal switching happens t_h time later than the active clock edge, the data at the input of the flip-flop can be latched correctly. In this case, the delay from the active clock edge to the output of the flip-flop (Q), clock-to-q delay d_{cq}, is characterized as a constant. If either of the two constraints is violated, the flip-flop is considered to not work properly, and a timing error is reported.

Timing constraints of digital circuits can be explained further using Fig. 3.4, where three flip-flops are connected by combinational circuit blocks. When an active clock edge is generated, all flip-flops are triggered at the same time. In traditional STA, timing constraints are defined with setup time, hold time and constant clock-to-q delays. For flip-flops i and j, the timing constraints can be written as

$$d_{cq,i} + \overline{d}_{ij} + t_{su,j} \leq T \tag{3.4}$$

$$d_{cq,i} + \underline{d}_{ij} \geq t_{h,j} \tag{3.5}$$

where $d_{cq,i}$ is the delay of flip-flop i, $\overline{d}_{ij}$ ($\underline{d}_{ij}$) is the maximum (minimum) delay of the combinational circuit between flip-flops i and j, $t_{su,j}$ ($t_{h,j}$) is the setup (hold) time of flip-flop j, and T is the clock period.

The traditional setup time of a flip-flop is characterized by moving the signal switching at the input of the flip-flop to the active clock edge gradually. As the signal switching approaches the active clock edge, the delay of the flip-flop starts to increase. When this delay reaches a given metric, e.g., 110% of the minimum delay of the flip-flop characterized with very large setup and hold slacks, the time difference

between the signal switching and the active clock edge is defined as the setup time. Hold time is characterized similarly by moving the signal switching after the active clock edge to the clock edge gradually. Accordingly, this increased delay of the flip-flop, 110% delay in this case, is used as the clock-to-q delay in timing analysis.

The characterization process above guarantees that the circuit works properly if the constraints (3.4) and (3.5) are met for each pair of flip-flops, because the reserved setup time from latest signal switching to the active clock edge and the reserved hold time from the clock edge to the earliest signal switching together guarantee that the clock-to-q delay of the flip-flop does not exceed the characterized delay. This guarantees further that the timing constraints of the next flip-flop stage, e.g., between flip-flop j and k in Fig. 3.4 can be verified by assigning the clock-to-q delay of the flip-flop j to the characterized constant delay. In this way, the timing dependency between consecutive flip-flop stages is hidden from designers.

The timing model of a flip-flop with setup time, hold time and constant clock-to-q delay described above is a significant simplification for timing analysis and optimization. As a result, the combinational circuits between flip-flops can be analyzed and optimized independently without considering time borrowing as in designs using level-sensitive latches. This efficiency in timing analysis is very important because front-end circuit design usually undergoes many analysis-optimization iterations.

The simplified flip-flop model, however, sacrifices circuit performance for the sake of execution efficiency. In order to simplify the clock-to-q delay to be a constant, the latest signal switching must be t_{su} earlier than the clock edge. Otherwise, the circuit is not considered to work properly. In reality, however, the flip-flop may still latch the input data correctly if the arrival time of a signal is late, although the clock-to-q delay may become larger than the constant delay. If the arrival time of the input signal is too close the clock edge, the flip-flop may finally enter metastability and thus fail to work properly. Similarly, if the hold time constraint is violated, the flip-flop also works with a larger delay in a feasible region, before the signal change is too close to the clock edge. For a given signal, we refer to the distance from the signal switching at the input of a flip-flop to the clock edge as *setup slack*, and the distance from the clock edge to the signal switching as *hold slack*, as illustrated in Fig. 3.3. Consequently, setup time and hold time in the traditional definition are

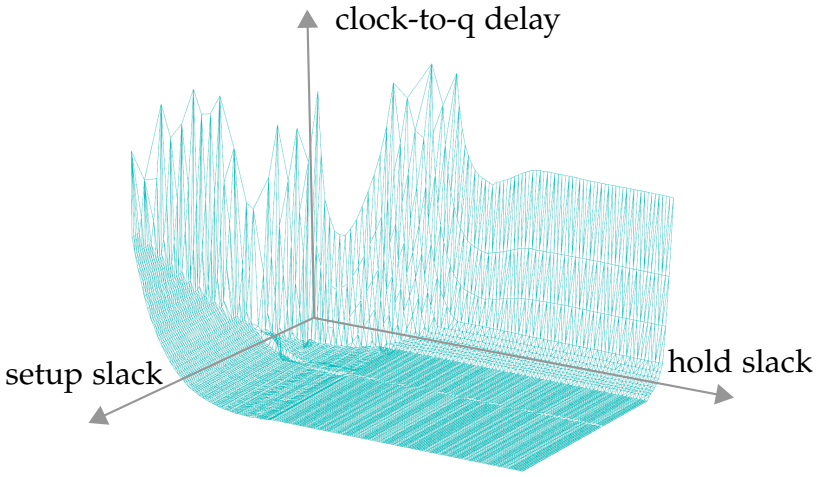

Figure 3.5: The three dimensional clock-to-q delay surface of a 45nm flip-flop with respect to setup slack and hold slack.

actually also slacks at which the clock-to-q delay of the flip-flop is characterized as a constant.

The relation between clock-to-q delay, setup slack and hold slack can be demonstrated using the simulated delay surface of a 45nm flip-flop in Fig. 3.5. In this example, we can see that when the setup and hold slacks are large, the delay of the flip-flop is a constant. If these slacks are reduced enough, the clock-to-q delay becomes larger, until the flip-flop finally enters metastability. In traditional STA, the flip-flop is assumed to work in the area with a constant clock-to-q delay. This simplification does not take advantage of the feasible region beyond the setup time and hold time, so that the circuit performance may be underestimated.

If the flip-flop is allowed to work in the region with the setup slack smaller than setup time, the clock period for the critical path of the circuit can be smaller, because the latest signal switching can arrive at the ending flop-flip later. Accordingly, the delay of this flip-flop becomes larger, but the increased delay only affects the combinational path in the next stage. In case the delay of the combinational path of the next stage is not large, no timing violation appears. Consequently, the critical path receives more timing budget for signal propagation, leading to an improved circuit performance. This delay compensation is very similar to designs with level-sensitive latches, or designs using intentional clock skew scheduling. The advantage

of this phenomenon is that the performance increase comes from a more accurate timing analysis, without invoking the complete design-optimization flow or timing ECO. Therefore, timing analysis considering this delay compensation is specially useful in late stages of design flow such as timing signoff.

As mentioned earlier, when setup slack or hold slack becomes smaller, the delay of the flip-flop increases. Fig. 3.6 illustrates several curves with respect to the setup slack and hold slack as in [KL14]. On each curve, the clock-to-q delay is a constant, and the curve closer to the axes has a larger delay. Before the delay becomes very large and soon the flip-flop enters the metastable region, all these curves are valid and the flip-flop can work with each setup/hold slack combination on them. The traditional setup/hold time delay model, however, only assumes that the flip-flop works with curves with a delay no larger than the characterized delay, e.g., 110% of the stable delay. All the other curves on the left of the 110% curve are simply ignored.

When characterizing the traditional setup time, the hold slack is set to a very large value to exclude its influence. The setup slack is then decreased gradually, until the delay of the flip-flop increases to 110%. The hold time is characterized similarly. Therefore, the setup and hold time point used for static timing analysis is shown as point A in Fig. 3.6b. When point A is used in STA, the shaded area under the curve is also included. This area is located below the curve, so that the delay in the area is larger than 110% delay. However, the flip-flop delay used in STA is still assumed as the characterized 110% delay, posing a risk that the circuit still does not work even though all timing constraints are met.

The method in [SFD+06, SDT+07] excludes the delay curves lower than the 110% delay curve in timing analysis, so that the clock period cannot be improved. For example, assume flip-flop j in Fig. 3.4 works with a small setup slack, so that the clock period for this flip-flop stage can be lowered. This small setup slack may incur an increase of the clock-to-q delay of flip-flop j. This delay increase, however, may be absorbed by the stage between flip-flops j and k, if this stage has a small combinational delay, so that the lowered clock period works with the whole circuit. This scenario is similar to the case that flip-flop j works at the point B in Fig. 3.6a and flip-flop k still works at point C.

To consider the three dimensional interdependency between clock-to-q delay, setup

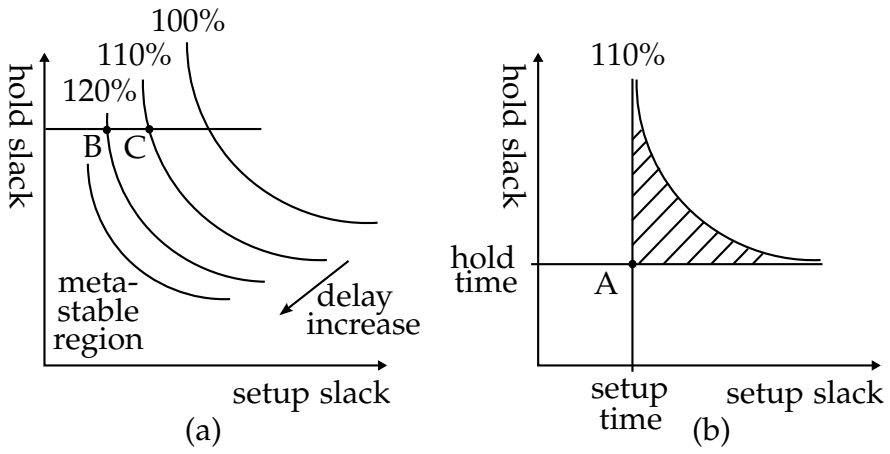

Figure 3.6: Delay curves of a flip-flop. (a) Curves of setup/hold slack combinations with respect to different constant clock-to-q delays. (b) Characterization point of setup time and hold time in traditional STA.

slack and hold slack, the method in [JB05] uses a quadratic programming model to calculate the optimal clock period directly, but it is limited due to the scalability of this high-order programming method. To simplify the three dimensional model, the method in [CLS12] approximates the relation between clock-to-q delay and setup/hold slacks using an analytic function and calculates the minimum clock period of a circuit by iterations. This method, however, cannot guarantee to converge in the given number of iterations. In addition, the method in [KL14] approximates the three dimensional delay surface using linear planes, but in calculating the minimum clock period this method splits the problem into two dimensional problems, so that it cannot guarantee an optimal solution. Furthermore, the method in [YTJ15] proposes a very efficient algorithm to capture timing violations in a circuit, but it only considers the relation between clock-to-q delay and setup slack.

The above methods either cannot solve the STA problem considering the interdependency between clock-to-q delay, setup slack and hold slack, or cannot guarantee the quality of the solution. Accordingly, a holistic method is developed in Section 5 to calculate the minimum clock period of a circuit by modeling the three dimensional delay.

3.2.5 The Confines of Traditional Timing Paradigms

In Section 3.2.4, flexible timing characteristics of flip-flops are used to balance timing slacks in circuits after manufacturing. In the traditional timing paradigm, flip-flops synchronize logic computation, so that logic waves cannot be propagated through them. Therefore, the timing performance of circuits is restricted by the barrier of flip-flops. However, if a flip-flop between two stages is removed, delay imbalances of two stages are automatically exploited without incurring the risk of extreme limit performance of flip-flops. Furthermore, the effect of process variations can be alleviated further by eliminating the inherent clock-to-q delays of flip-flops. The challenge of this technique is to maintain the correct function of the original circuit after some flip-flops are removed. In this section, the function of sequential components in the traditional timing paradigm is explained. Afterwards, an analysis of the confines of this paradigm is provided.

In digital circuit designs, clock frequency determines the timing performance of circuits. In the traditional timing paradigm, sequential components, e.g., edge-triggered flip-flops, synchronize signal propagations between pairs of flip-flops. Consequently, these propagations are blocked at flip-flops until a clock edge arrives. At an active clock edge, the data at the inputs of flip-flops are transferred to their outputs to drive the logic at the next stage. Therefore, combinational logic blocks are isolated by flip-flop stages. This fully synchronous style can reduce design efforts significantly, since only timing constraints local to pairs of flip-flops need to be met.

Within the traditional timing paradigm, many methods have been proposed to improve circuit performance. They are gate sizing, retiming, clock skew scheduling and wave-pipelining, which are introduced in Section 2.3.

Sequential components are assumed to synchronize signal propagation in these methods, where no signal propagation is allowed to pass through sequential components except at the clock edges. This synchronization with sequential components achieves many benefits such as reducing design efforts. However, it limits circuit performance in two regards. Firstly, sequential components have inherent clock-to-q delays and impose setup time constraints. The former becomes a part of combinational paths driven by the corresponding flip-flops and the latter deprives a further part of the timing budget for the critical paths. Secondly, delay imbalances

between flip-flop stages cannot be exploited since signal propagations are blocked at flip-flops instead of being allowed to propagate through flip-flops. Although clock skew scheduling can relieve this problem to some degree, it still suffers the inherent clock-to-q delays and setup time constraints of flip-flops. The third method above, wave-pipelining, allows signals to pass through sequential stages without flip-flops, however, this technique is not compatible with the traditional timing paradigm.

The limitation of timing performance of circuits in traditional timing paradigm can be illustrated using Fig. 3.7. Fig. 3.7(a) shows a part of a circuit where the combinational path between F2 and F3 is critical with a path delay equal to 17. Assume that the clock-to-q delay, the setup time and the hold time of a flip-flop are 3, 1, and 1, respectively. The minimum clock period of this circuit is thus equal to 21.

To reduce the clock period, logic gates with smaller delays can be selected from the library to accelerate signal propagations on the critical paths of the circuit, at the cost of additional area overhead, leading to the circuit shown in Fig. 3.7(b), where the logic gates that are not on the critical path still have their original delays for the sake of saving area. After sizing, the minimum clock period of this circuit is reduced to 16 units. To reduce the clock period further, retiming can be deployed to move F3 to the left of the XOR gate as shown in Fig. 3.7(c), leading to a minimum clock period equal to 11.

The circuit in Fig. 3.7(c) has reached the limit of timing performance in the traditional timing model, and no other method except a logic redesign can reduce the clock period further. However, this strict timing constraint can still be relaxed by removing F6 from the circuit, leading to the circuit in Fig. 3.7(d). If the signal from F2 can reach the sink flip-flops F3 and F4 after the next rising clock edge and before the rising edge two periods later, data can still be latched by F3 and F4 correctly. Since the inverter before F4 can also be sized further, the largest path delay is 16, which imposes a lower bound for the clock period as (16+1)/2=8.5, 22.7% lower than retiming.

Since F6 can be removed from the circuit without affecting its function in fact, it makes no contribution to the logic function or timing performance in Fig. 3.7(c). However, the flip-flop F5 in Fig. 3.7(c) cannot be removed, because the signal from F1 should also arrive at F4 later than one clock period. Without F5, the signal from F1 arrives at F4 even before the next rising clock edge, a loss of logic synchronization

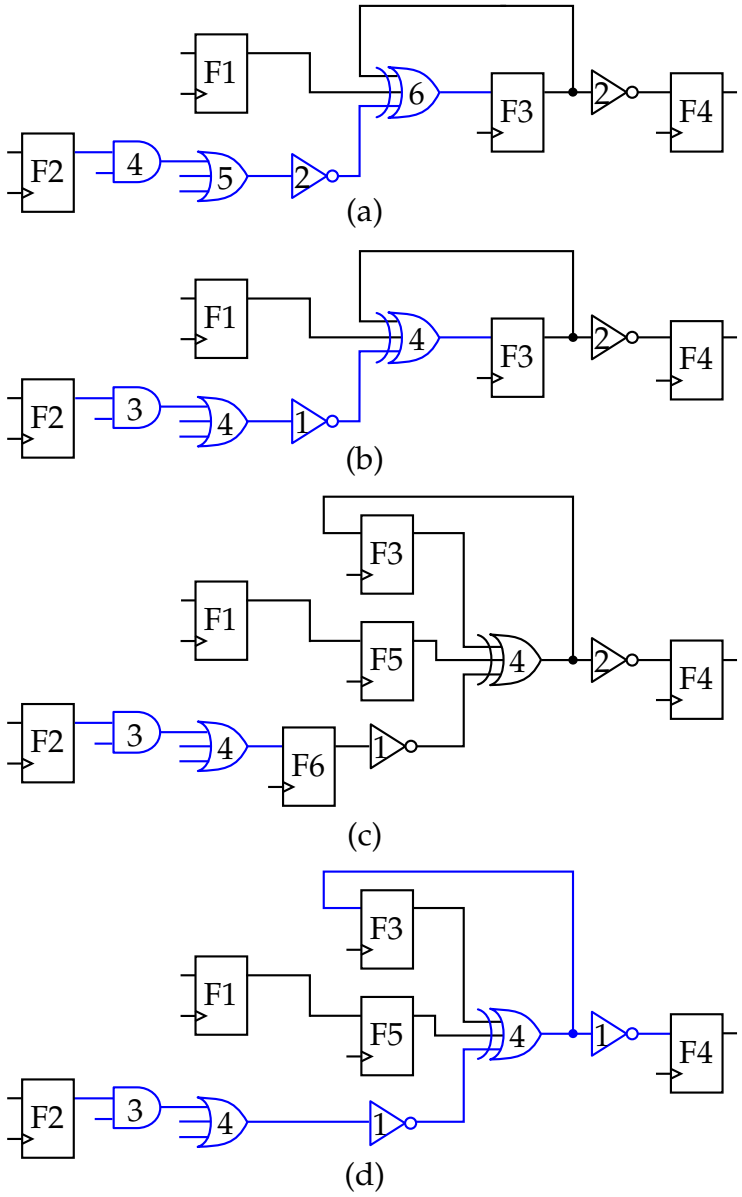

Figure 3.7: Timing optimization methods. Delays of logic gates are shown on the gates. The clock-to-q delay (t_{cq}), setup time (t_{su}) and hold time (t_h) of a flip-flop are 3, 1 and 1, respectively. (a) Original circuit. (b) Sized circuit. (c) Circuit after retiming. (d) Circuit after optimization allowing two waves on logic paths from F2 to F4.

compared with the circuit in Fig. 3.7(a). Comparing Fig. 3.7(b) and Fig. 3.7(d), we can see that F3 in Fig. 3.7(b) simply blocks the fast path from F1 to F4 to avoid loss of logic synchronization or timing violations at F4, but it degrades the circuit performance by delaying the signal from F2 to F4 too.

The concept to allow logic signals to span several sequential stages without a flip-flop separating them is called *wave-pipelining* [BCKL98]. Previously, this technique has only been explored in the context of circuit design, where the numbers of waves on logic paths should be defined and their synchronization should be maintained by designers during the design phase. Since logic design and timing cannot be handled separately as in traditional synchronous designs, wave-pipelining becomes incompatible with the traditional fully synchronous design paradigm and prevents its adoption in industrial designs.

The timing performance in the traditional timing paradigm is limited by the barrier of flip-flops. Accordingly, a novel timing model is established in Section 6 to break the confines of the traditional timing paradigm. In the new timing paradigm, signals, specifically those along critical paths, are allowed to propagate through sequential stages without flip-flop while the functionality of circuits is maintained.

3.3 Summary

As transistor feature sizes continue to be scaled down, process variations are becoming more and more pronounced. To deal with these variations, various methods have been proposed. The traditional corner-based method is pessimistic and requires an exponential number of corners with the increasing of process variations. SSTA models process variations as random variables with known probability distributions. Thus, delay distributions are propagated to determine the probability distribution of circuit performance. However, this method is too complicated with realistic distributions. Another direction to deal with process variations is post-silicon tuning, where post-silicon devices are inserted in the circuits at the design phase and the chips with timing failures are tuned with these devices after manufacturing to improve yield or profit. This method can achieve a better performance by balancing the timing slacks at the cost of tuning circuitry. If the flexible setup

and hold time relation of flip-flops can be exploited, timing imbalances between different flip-flop stages can be achieved automatically. Furthermore, flip-flops in the traditional timing paradigm actually degrade the circuit performance by introducing clock-to-q delay and setup time constraints. By removing them from circuits appropriately, the timing performance of circuits can be improved even beyond the limit of the traditional sequential design.

Chapter 4

Post-Silicon Tuning to Mitigate Process Variations

At submicron manufacturing technology nodes, process variations become increasingly pronounced and affect circuit performances significantly. To counter these variations, a large timing margin is reserved to maintain yield, leading to an unaffordable overdesign. Most of these margins, however, are wasted after manufacturing, because process variations cause only some chips to be really slow, while other chips can easily work at the designated clock frequency. To reduce this pessimism, we can reserve less timing margin and tune failed chips after manufacturing with clock buffers to make them meet timing specifications. With this post-silicon clock tuning, timing budgets of critical paths can be balanced with those of neighboring paths in each chip individually to counter the effect of process variations. Consequently, chips with timing failures can be rescued and the yield can thus be improved.

There are two challenges when using the post-silicon tuning. Firstly, tunable buffers should be inserted into the circuit during the design phase to balance the trade-off between the profit/yield and area overhead of these buffers. Secondly, after manufacturing, chips with timing failures will be rescued by tuning buffers according to the delay test.

In this section, a method to determine where to insert post-silicon tuning buffers during the design phase to improve the overall profit with clock binning is proposed in Section 4.1. The proposed efficient delay test framework (EffiTest2) to solve the post-silicon testing problem will be described in Section 4.2. Results for both of these techniques will be presented in Section 4.3.

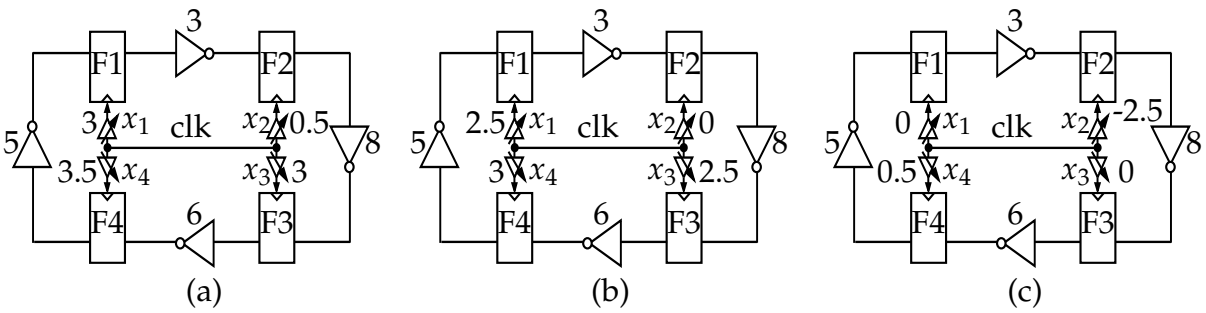

Figure 4.1: Performance improvement using post-silicon tuning buffers. Minimum achievable clock period is 5.5. Tuning values in (a) and (b) are constrained in $[0, 4]$. Setup time and hold time are assumed as 0 for simplicity. (a) Tuning configuration without reduction. (b) Reduced tuning configuration. (c) Reduced tuning configuration with negative tuning values.

4.1 Post-Silicon Tunable Buffer Insertion at the Design Phase

In this section, a method to determine buffer locations by iterative learning is proposed. In each iteration the buffers that are important to the yield/profit of the circuit are captured. Afterwards, the identified buffer locations are refined and buffer ranges are compressed to reduce area cost. Firstly an overview of timing constraints for circuit with post-silicon tuning buffers in Section 4.1.1 is demonstrated. The buffer insertion problem is formulated in Section 4.1.2. The proposed method to determine buffer locations is explained in detail in Section 4.1.3–4.1.7.

4.1.1 Timing Constraints with Post-Silicon Tunable Buffers

If a circuit have post-silicon tunable buffers, the delays of clock paths to flip-flops attached with these buffers can be adjusted after manufacturing for each chip individually. The concept of this tuning can be explained using the example in Fig. 4.1a, where four flip-flops are connected into a loop by combinational paths. Without post-silicon clock tuning, the minimum clock period of this circuit is 8. If clock

edges can be moved by adjusting the delays of these tuning buffers, the minimum clock period can be reduced to 5.5. For example, the buffer value x_2 shifts the launching clock edge at F2 0.5 units later and the buffer value x_3 shifts the capturing clock edge at F3 3 units later. Therefore, with a clock period of 5.5, the combinational path between F2 and F3 now has 5.5-0.5+3=8 time units to finish signal propagation. This shifting of the clock edge reduces the timing budget of the path between F3 and F4 by 3 units, but this path still works with the clock period 5.5 without any timing violation because the buffer value x_4 moves the clock edge at F4 further later.

For an individual chip, this post-silicon clock tuning is similar to the concept of useful clock skews [Fis90]. The difference is that the tuning values are specific to each individual chip with timing failures after manufacturing, so that the effects of process variations can be handled specifically for each chip. If the skew schedule problem in [Fis90] is formulated with process variations, the skew to a flip-flop should still be identical in all manufactured chips, so that there is no chance to tune the chips with respect to the individual effect of process variations after manufacturing.

Four tunable buffers are used in Fig. 4.1a. To reduce the number of buffers, all the delays of the buffers can be reduced by 0.5 time units and the circuit still works with the clock period 5.5, as shown in Fig. 4.1b. Furthermore, we can reduce the number of buffers even to two, if we can move the clock edge at F2 2.5 time units earlier, so that the timing slack of the path between F1 and F2 can be shifted to the path between F2 and F3 directly, as in Fig. 4.1c. This negative delay can be implemented by shortening the original clock path in advance to introduce a negative delay in reference to the predefined arrival time of clock signals. With negative clock delays allowed, timing budgets can be balanced in both the clockwise direction and the counterclockwise direction, so that the number of required buffers can be lowered to reduce area and post-silicon configuration cost.

Fig. 4.2 explains the timing constraints with clock tuning buffers. In this figure, two flip-flops with post-silicon tunable buffers are connected by a combinational circuit. Assume that the clock signal switches at reference time 0. Then the clock events at flip-flops i and j happen at time x_i and x_j, respectively. To satisfy the setup time and

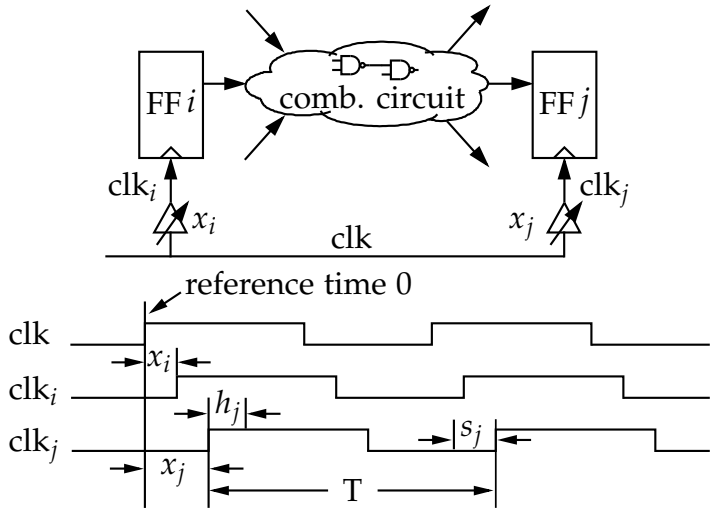

Figure 4.2: Timing of circuits with tuning buffers.

hold time constraints, the following inequations must be met.

$$x_i + \overline{d}_{ij} \leq x_j + T - s_j \tag{4.1}$$

$$x_i + \underline{d}_{ij} \geq x_j + h_j \tag{4.2}$$

where x_i and x_j are delay values of tuning buffers, $\overline{d}_{ij}$ ($\underline{d}_{ij}$) is the maximum (minimum) delay of the combinational circuit between flip-flops i and j, s_j (h_j) is the setup (hold) time of flip-flop j, and T is the clock period. Here the clock buffers introduce two delay variables into the constraints (4.1) and (4.2). Without them, the two inequations fall back to the normal timing constraints of digital circuits.

Due to extra area overhead, the configurable delay of a clock buffer usually has a limited range. Assume that the lower bound of the tuning values of buffer i is r_i and the upper bound is $r_i + \tau_i$, where τ_i is the size of the buffer. The delay value of buffer i can thus be constrained by a range window as

$$r_i \leq x_i \leq r_i + \tau_i. \tag{4.3}$$

Unlike [TZC05], we model the range window of the tuning values as asymmetrical with respect to 0 to achieve a maximal flexibility. Furthermore, x_i may take only discrete values due to implementation of the tunable buffers.

To determine the buffer locations during the design phase, the path delays, setup times and hold times should be considered as random variables modeled using data

Table 4.1: Notations

r_i, τ_i	Lower bound and size of a buffer range
N_b	Upper bound of the number of buffers
N_p	Number of performance bins
$T_{m,l}$, $T_{m,u}$	Lower and upper bounds of the mth bin
p_m	Average profit of a chip in the mth bin
y_m	Percentage of chips in the mth bin
$\mathcal{P}$	Overall profit
x_i^k, x_j^k	Tuning values for the kth sample
$\overline{d}_{ij}^k$, $\underline{d}_{ij}^k$	Sampled delays
s_j^k, h_j^k	Sampled setup time and hold time
T^k	Clock period of the kth sample
c_i	0-1 variable indicating whether the ith flip-flop has a buffer
b_m^k	0-1 variable indicating whether the kth sample is assigned to the mth bin
g_m^k	Auxiliary 0-1 variable to express b_m^k
N_s	Number of samples in the low-discrepancy sequence
N_f	Number of samples in the low-discrepancy sequence after prefiltering
N_t	Number of samples in one batch processed together
$\mathcal{B}, \mathcal{B}'$	Buffer sets saving the allocation candidates

provided by foundries. With process variations considered, the tuning delays x_i and x_j also become statistical, because the clock buffers are subject to process variations too. These variations can be decomposed and merged with the random variables representing $\overline{d}_{ij}$, $\underline{d}_{ij}$, s_j and h_j, e.g., using the canonical form in [VRK+04]. The task of buffer allocation is thus to determine the locations of buffers that can make as many chips as possible meet the given timing specification after manufacturing, using only the statistical timing information available during the design phase. The buffer insertion problem is formulated in the following section.

4.1.2 Problem Formulation of Buffer Insertion

To apply the post-silicon tuning technique, buffers are inserted into the circuit after logic synthesis is finished and before physical design is started. Since buffers incur area overhead, and require additional test to configure them, the number of buffers in a design should be constrained. In addition, the ranges of the buffers should be reduced as much as possible. Furthermore, in high-performance designs such as CPUs, chips are tested after manufacturing and assigned into bins of different performance grades, and the price of a chip from a bin of high speed is higher than that from a low-speed bin. In this scenario, improving the overall profit of all bins is more important than improving the yield of the circuit with respect to a single clock period.

The important notations that appear in this section are listed in Table 4.1, and the problem of buffer allocation is formulated as follows.

Input:

- Circuit structure and statistical path delays;
- Buffer specification, including the maximum allowed size τ_i of buffers defined in (4.3) and the number of discrete steps in the tunable delay range;
- The maximum number of buffers allowed in the circuit N_b;
- The number of performance bins N_p. For the mth bin, an upper bound $T_{m,u}$ and a lower bound $T_{m,l}$ are defined by the designer. After manufacturing, a chip with a clock period T assigned to the mth bin should meet $T_{m,l} < T \leq T_{m,u}$. For a chip in the mth bin, the average profit is given as p_m. For convenience, we order the bins from high performance to low performance, so that $T_{m,u} = T_{m+1,l}$.

Output:

- A set of flip-flops at which tuning buffers should be inserted on the their clock paths;
- The sizes of the buffers inserted into the circuit. These sizes must be no larger than the given maximum size τ_i.

Constraints:

- For any pair of flip-flops i and j with combinational paths between them, the constraints (4.1)–(4.3) hold;
- The number of buffers inserted in the circuit must not exceed N_b.

Objectives:

- Maximize the overall profit
$$\mathcal{P} = \sum_{m=1}^{N_p} p_m y_m \tag{4.4}$$
where y_m is the percentage of the chips that are assigned into the mth bin after manufacturing;
- Reduce the sizes of the inserted buffers while maintaining the overall profit $\mathcal{P}$.

To define the bins, the first bin has the highest clock frequency, and it has no lower bound for the clock period T, so that $T_{1,l}$ can be set to any positive value given by designers. After manufacturing, if a chip cannot be assigned to any of those bins, i.e., $T > T_{N_p,u}$, this chip is considered as a part of yield loss. The definition (4.4) can be expanded. Consider the case that only one bin is used, this problem is equivalent to the yield improvement problem with respect to a single clock period.

Since the relation between profit and the number of buffers is very complicated, we do not include the number of tuning buffers in the problem formulation above as a part of the optimization objective. With our formulation, designers can generate several combinations of buffer number and profit, and select the most appropriate setting according to their own cost model. If the number of buffers are required to be moved from a constraint into the optimization objective (4.4), the proposed method can still work with only a slight modification.

There are two challenges in solving the optimization problem above. The first challenge comes from the random variables in (4.1) and (4.2), because only statistical timing information are available during the design phase when the buffers are allocated. The profit in (4.4) is thus defined similar to an expected value, which is slightly larger than the actual profit after manufacturing because statistical delays and timing properties cannot be measured exactly [ZLS+18]. The second challenge is that the variables x_i and x_j in (4.1) and (4.2) may take only discrete values in the range window defined by (4.3). For example, the de-skew buffer in [TRND+00]

can be configured to only 20 discrete delays. In this case, integer linear programming (ILP) becomes almost the only method available to deal with the constraint set defined by (4.1)–(4.3) after the random variables are fixed by sampling.

To combat these challenges, the learning-based statistical sampling is deployed. The basic idea is that we use a set of representative samples and model the numbers of samples in the different performance bins directly. We then determine buffer locations by maximizing the overall profit calculated from the yield values of these bins and the profit per chip for each bin. By sampling the random variables directly we can transform the statistical optimization problem into an ILP problem. Therefore, the relation between the statistical variables and the profit of the circuit can be established directly. With this relation, we can then capture buffer locations that are sensitive to yield/profit. The proposed method is in fact learning-based when dealing with a large number of samples in the problem space. Learning-based methods have been applied in the design automation field extensively, e.g., for statistical path selection considering large process variations [XSZV09], for sensor placement in dynamic noise management systems [WZXS13], and for parametric yield estimation for analog/mixed signal circuits [GYSH14].

The overall flow of the proposed method is illustrated in Fig. 4.3. In this flow, we first generate a low-discrepancy sample sequence (Sobol sequence) and filter out the samples that are not affected by any buffers. Thereafter, we try to capture buffer locations and refine them iteratively. The ranges of buffers are compressed and the number of buffers is reduced by grouping in the end to reduce area cost. This flow will be explained in detail in the following sections.

4.1.3 Sampling-based ILP Modeling between Statistical Delays and Profit

If a large number of N_s samples can be generated from the joint distribution of all the random variables in the optimization problem, they can actually emulate the chips after manufacturing. If tunable buffers are attached to critical flip-flops, we can introduce intentional clock skews customized for each sample, or emulated chip, individually, to make the failing samples work again, or to move low-performance samples into high-performance bins by tuning the buffers. For each emulated chip,

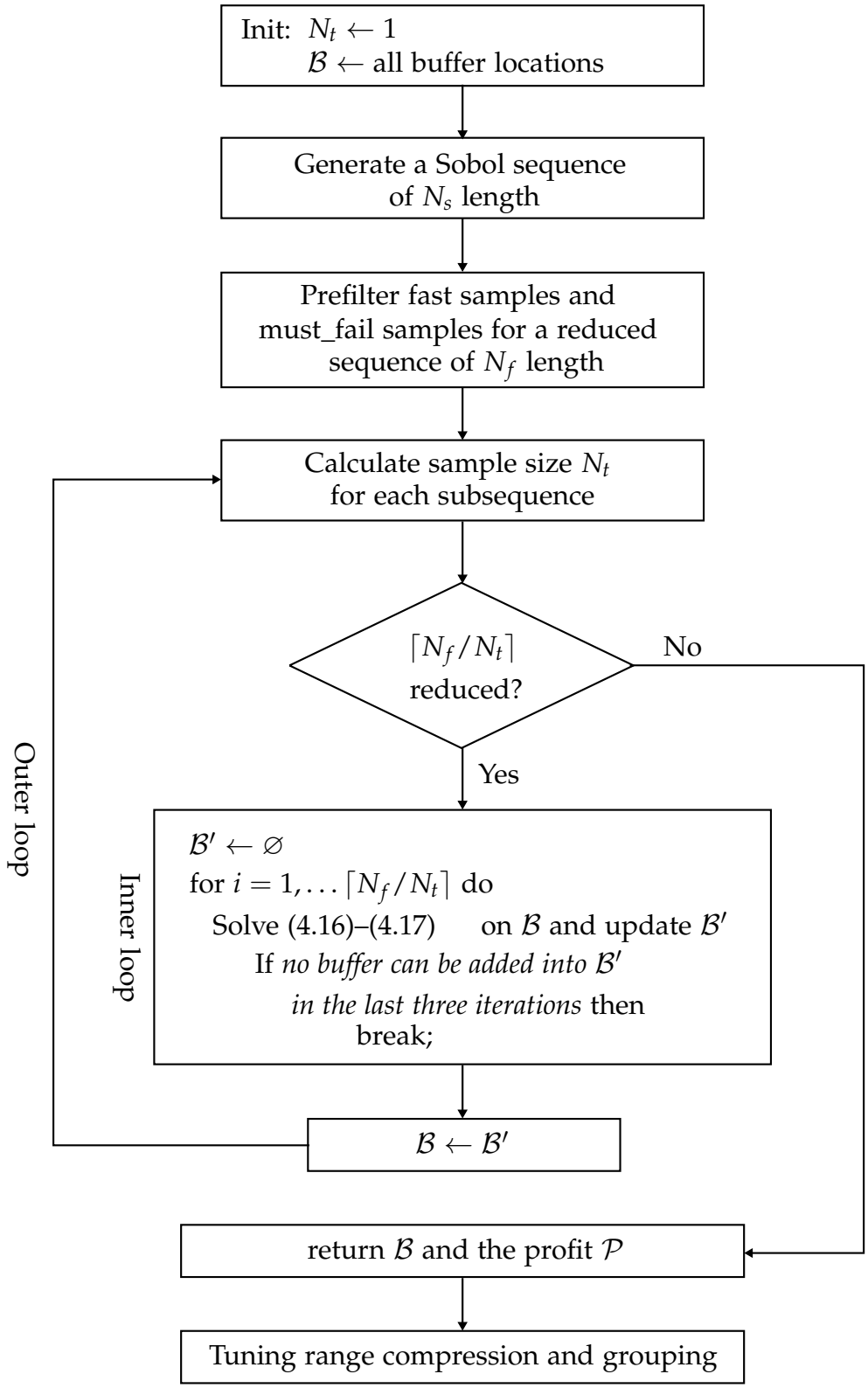

Figure 4.3: Prefiltering and iterative buffer allocation flow.

the statistical variables in the constraints become fixed values, so that the performance improvement can be evaluated. In this way, the relation between buffer locations and the profit can be established directly and an ILP solver can be used to determine the optimal buffer allocation.

For the kth sample from the N_s samples, the constraints (4.1)–(4.3) become

$$x_i^k + \overline{d}_{ij}^k \leq x_j^k + T^k - s_j^k \tag{4.5}$$

$$x_i^k + \underline{d}_{ij}^k \geq x_j^k + h_j^k \tag{4.6}$$

$$r_i \leq x_i^k \leq r_i + \tau_i \tag{4.7}$$

where $\overline{d}_{ij}^k$, $\underline{d}_{ij}^k$, s_j^k and h_j^k are the kth sample values of random variables $\overline{d}_{ij}$, $\underline{d}_{ij}$, s_j and h_j; x_i^k and x_j^k are intentional clock skews for this specific sample introduced by configuring the corresponding tuning buffers after manufacturing to improve the performance, in other words, to reduce the minimum clock period T^k. Note in (4.7) r_i and τ_i are not indexed by k, because if a buffer is inserted on the clock path to a flip-flop, it appears in all the chips after manufacturing, and its range in all chips is also the same.

To indicate whether there is a buffer inserted on the clock path to the ith flip-flop, we assign a binary variable c_i to it. If there is no buffer inserted, c_i is set to 0; otherwise, c_i is set to 1. Because a post-silicon clock skew can be added only when a buffer appears, the skew or the tuning value of the buffer at the ith flip-flop can be written as

$$x_i^k = \begin{cases} 0 & \text{if } c_i = 0, \\ \text{any valve} \in [r_i, r_i + \tau_i] & \text{when } c_i = 1. \end{cases} \tag{4.8}$$

According to the definition of c_i, we need only to force x_i^k to be 0 to disable the potential clock tuning when c_i is equal to 0. The constraint (4.8) can thus be transformed to

$$x_i^k \leq c_i \Gamma \tag{4.9}$$

$$-x_i^k \leq c_i \Gamma \tag{4.10}$$

where Γ is very large constant. If c_i is set to 0, x_i^k must be set to 0 to meet (4.9) and (4.10). If c_i is set to 1, these two constraints have no effect because Γ is a

predetermined constant larger than any possible value of x_i^k or $-x_i^k$. In this case, x_i^k is actually constrained by (4.7).

With c_i defined to indicate the appearance of a buffer at the ith flip-flop, we can constrain the number of buffers in the circuit easily as

$$\sum_i c_i \leq N_b \tag{4.11}$$

where the sum on the left adds the c_i variables for all flip-flops in the circuit together, and N_b is the given upper bound of the number of buffers allowed in the circuit.

To check the minimum clock frequency of an emulated chip that can work with, we need to compare the minimum clock period T^k of the kth sample with the upper and lower bounds of the performance bins. If T^k falls into the mth bin by satisfying $T_{m,l} < T^k \leq T_{m,u}$, the number of the chips in this bin is increased by one. Instead of comparing T^k with the bounds of the bins directly, we take advantage of the fact that the yield values of the circuit in different bins are a part of the optimization objective defined in (4.4) and the price of a chip in a high performance bin is higher than that in a low performance bin. We define the 0-1 variables $g_m^k, \quad m = 1, \ldots, N_p$ to represent whether the minimum clock period T^k of the kth sample is smaller than the upper bound of the mth bin. Therefore, g_m^k can be constrained as

$$g_m^k = 1 \Longleftrightarrow T^k \leq T_{m,u}, m = 1, 2, \ldots, N_p. \tag{4.12}$$

We then use g_m^k to define another 0-1 variable b_m^k which indicates whether the kth sample falls into the mth bin meeting $T_{m,l} < T^k \leq T_{m,u}$, as

$$b_m^k = \begin{cases} g_m^k & m = 1, \\ g_m^k - g_{m-1}^k & m = 2, \ldots, N_p. \end{cases} \tag{4.13}$$

The constraint (4.12) can be transformed into

$$T^k - T_{m,u} \leq (1 - g_m^k)\Gamma, m = 1, 2, \ldots, N_p \tag{4.14}$$

where Γ is very large positive constant.

The constraints (4.13) and (4.14) can be explained as follows. If T^k is no larger than the upper bound of the mth bin $T_{m,u}$, the left side of (4.14) is negative, so that g_m^k can be either 0 or 1; otherwise, g_m^k must be 0. Since the objective of the optimization

problem is to increase the numbers of chips in high-performance bins as much as possible, the solver will assign all $g_m^k, g_{m+1}^k, \ldots, g_{N_p}^k$ to 1 if $T^k \leq T_{m,u}$, because the bins are arranged in the high performance to low performance order so that T^k is also smaller than $T_{m+1,u}, \ldots, T_{N_p,u}$. Therefore, the constraint (4.13) only keeps the b_m^k for the fastest bin to which the sample can be assigned to be 1, and for the slower bins it is set to 0. Consequently, b_m^k represents whether the chip is assigned to the mth bin.

With b_m^k we can calculate the numbers of emulated chips in all bins easily, and the yield or the percentage y_m for the mth bin can be expressed as

$$y_m = \sum_{k=1}^{N_s} b_m^k \Big/ N_s \tag{4.15}$$

where N_s is the total number of samples.

With the constraints defined above, the problem to optimize the overall profit can be expressed as

$$\text{maximize} \quad \sum_{m=1}^{N_p} p_m y_m \tag{4.16}$$

$$\begin{aligned} \text{s.t.} \quad & (4.5)\text{–}(4.7), (4.9)\text{–}(4.11), (4.13)\text{–}(4.15), \\ & \text{w.r.t. all flip-flops pair indexed by } (i,j), \\ & \text{and } k = 1, \ldots, N_s. \end{aligned} \tag{4.17}$$

In this formulation, we use a given number of samples to emulate chips after manufacturing and model the bin assignment process. We then use an ILP solver to maximize the profit for the simulated chips to determine the locations of buffers. Since the relation between the locations of buffers and the yield assignment is established in this formulation, we can determine the locations of buffers directly by solving the optimization problem above. In previous methods [TZC05, KS08], the relation between buffer locations and yield is not analyzed directly. Instead, this relation is considered as a separate evaluation problem in these methods, and the yield values for different combinations of buffer locations are calculated using Monte Carlo simulation, and only used as a metric to determine the next decision points in the path search or cutting plane methods. Consequently, Monte Carlo simulation have to be executed many times, leading to a large runtime.

If there are a large number of emulated samples N_s in the integer linear optimization problem (4.16)–(4.17), the profit can be modeled accurately and the results of c_i show the indication of the optimal locations to insert tuning buffers to maximize the profit. However, a large N_s may increase the number of constraints in (4.17) to a point where the size of the ILP problem exceeds the capacity of all existing ILP solvers. To deal with this scalability problem, two techniques are applied: 1) the number of emulated samples N_s is reduced by using a low-discrepancy sample sequence instead of a purely random sampling sequence; 2) the problem (4.16)–(4.17) is split into subsets and use them to learn the locations of buffers iteratively. After each iteration, the candidates of buffer locations can be refined.

4.1.4 Reducing the Number of Emulation Samples Using a Low-discrepancy Sequence

To guarantee the quality of the resulting buffer locations, a large number of samples are required to emulate the chips after manufacturing. If the number of samples are not large enough, the selections of buffers cannot represent the locations that are most important for profit or yield improvement. Consider the extreme case where we use only two samples, which have different probabilities to appear in the manufactured chips. In the formulation (4.16)–(4.17), however, we do not differentiate these two samples with respect to their probabilities so that the two samples have the same influence on the selection of buffers. Consequently, the formulation loses accuracy because the calculated optimal profit deviates from the real profit.

In traditional Monte Carlo simulation methods, this discrepancy problem is solved by using a large number of samples, say 10,000. Since the samples are generated according to the joint distribution of the variables, the number of points falling into a part of the sampling space corresponds to the probability of that region. The effect of probability can thus be handled by (4.16)–(4.17) implicitly, because samples from regions with large probabilities in the problem space appear more often than samples from other regions. The second method to solve this discrepancy problem is to use the probability of representative samples as further coefficients of the yield values in the objective (5.11) directly. But it is not clear how many samples should be generated to guarantee the quality of the result.

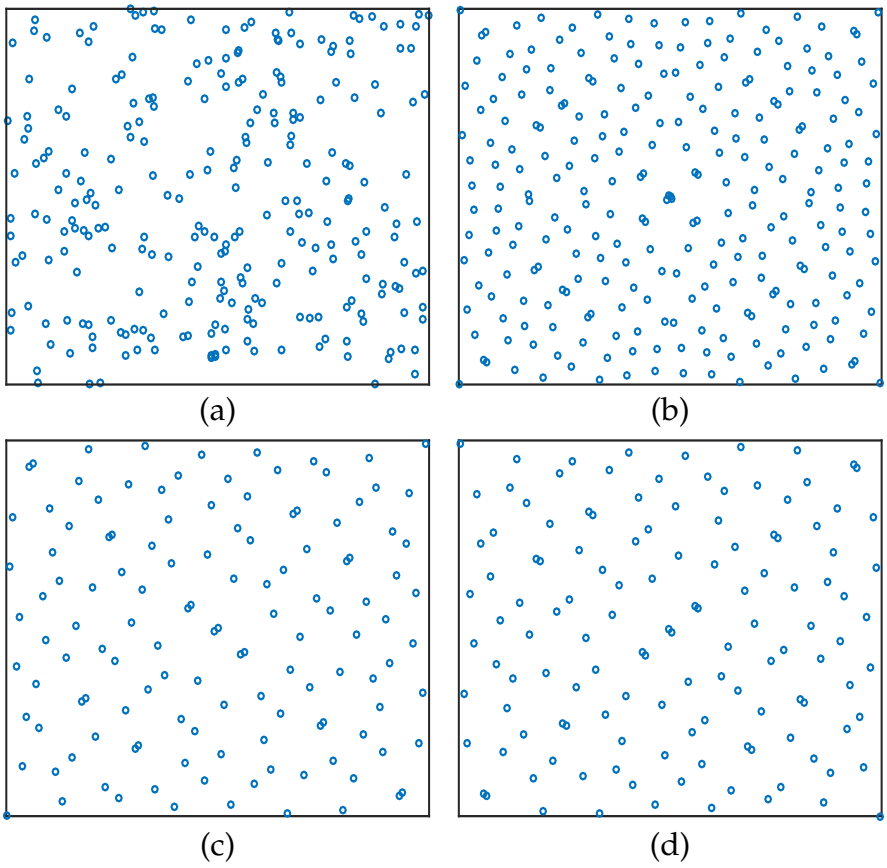

Figure 4.4: Purely random sequence and Sobol low-discrepancy sequence. (a) 256 random samples of two uniform variables. (b) 256 samples of a Sobol sequence for two uniform variables. (c) The first 128 samples from the Sobol sequence in (b). (d) The next 128 samples from the Sobol sequence in (b).

The third method to solve the problem of a large sampling number is to use a low-discrepancy sequence such as studied in [SR07]. In such a sequence, the number of samples in a given part of the sampling space is proportional to the probability of that region. The advantage of such a sequence is that this quasi-random sequence ensures the low discrepancy even with a small number of samples, so that it is widely used in quasi-Monte Carlo methods to reduce runtime. In statistical timing analysis, this method also demonstrates a strong advantage, e.g., more than 20 times acceleration has been achieved in [VCBS11]. In the proposed method, the Sobol sequence is used in [Sob67] to reduce the number of samples N_s. The effect of this sequence can be demonstrated using the examples in Fig. 4.4, where Fig. 4.4a shows a purely random number sequence of 256 samples for two uniform-distributed variables. Fig. 4.4b demonstrate that the Sobol sequence with the same number of samples spreads more evenly in the space. The original Sobol sequence follows uniform distribution, and it can be transformed to other distributions easily using methods such as the Box-Muller transform [BM58]. In the proposed method, 1000 samples in the Sobol sequence are used, which are one tenth of the usually used 10,000 samples of random variables in statistical static timing analysis [BCSS08]. In practice, test cases can converge even earlier with fewer than 1000 samples.

4.1.5 Buffer Allocation with Prefiltering and Iterative Learning

For the N_s emulated chips, some might work at the given clock frequency without tuning buffers; others cannot work even all flip-flops attached with tuning buffers. In both scenarios, tuning buffers are not important in improving the overall profit. Therefore, we exclude these samples from the ILP formulation (4.16)–(4.17) to reduce the number of variables and constraints.

If we want to filter out the samples which are fast enough, we need only to set all values of tuning buffers, x_i^k and x_j^k in (4.5) and (4.6) to 0, and calculate the clock period T_{min}^k for this sample as $T_{min}^k = \max_{i,j}\{\bar{d}_{ij}^k + s_j^k\}$. If T_{min}^k is smaller than the upper bound of the fastest bin, this sample is fast enough and no tuning is required. The constraint (4.6) is checked similarly. If all these constraints can be met without tuning buffers, this sample is excluded to reduce the computation cost.

To filter out the samples that are too slow to be assigned to a bin even when all

flip-flops have tuning buffers, we evaluate each path delay in a sample by verifying whether it is possible to tune this path to meet the upper bound of the slowest bin without considering the other paths. In the constraint (4.5), the sum of the path delay $\bar{d}_{ij}^k + s_j^k$ and $x_i^k - x_j^k$ should be no larger than T^k. We set buffer values x_i^k and x_j^k to the smallest and the largest values that are possible according to buffer specifications, respectively, and check whether the resulting clock period T^k is smaller than the upper bound of the slowest bin. If this still does not hold, there is no chance that this emulated chip can be assigned to one of the bins and the corresponding sample is considered as a part of yield loss and not included in the profit optimization problem. We repeat this prefiltering checking using (4.6) to exclude samples that do not work in any case due to unavoidable hold time violations.

After we prefilter the samples that are not important for profit/yield improvement, the remaining samples are used to determine buffer locations by solving the optimization problem (4.16)–(4.17). The number of these remaining samples is denoted as N_f. For a large circuit, the number of remaining variables and constraints in this ILP problem may still be too large to be dealt with by a modern solver. To reduce the scale of the ILP problem further, we split the ILP problem (4.16)–(4.17) into subproblems and determine the buffer locations with an iterative flow based on: 1) a subsequence of a Sobol sequence still exhibits a good low discrepancy as shown in Fig. 4.4c-d; 2) in a circuit only a small number of buffers can be inserted due to area cost. The iterative flow is illustrated in Fig. 4.3.

The first fact above shows that we may solve the ILP problem (4.16)–(4.17) with only a part of the Sobol sequence, indicating that we can capture the buffer locations only using a subset of samples. Therefore, we partition the whole Sobol sequence into several parts so that each part contains N_t samples which are processed together in one ILP problem (4.16)–(4.17). We call the samples processed in one ILP problem a batch. In our implementation, the number of samples N_t in one batch is determined by evaluating the numbers of variables and constraints and the capacity of the ILP solver. Since variables in an ILP problem define the dimension of the problems space, they carry more complexity into the ILP problem than constraints. Therefore, we consider the complexity of a variable to be five times that of a constraint, and the total number of the equivalent constraints should be smaller than a constant, 2×10^6 for Gurobi [Gur13] used in our experiments.

Though the samples in subsequences generally have lower discrepancy compared with a purely random sequence, there are still some slight patterns with higher discrepancy in these subsequences because of the small number of samples in one subsequence, as shown in Fig. 4.4c-d. Consequently, a subsequence with a limited number of samples may not capture all the buffer locations. We solve this problem by combining the buffer locations captured by different subsequences into a buffer set $\mathcal{B}'$. Once we finish solving (4.16)–(4.17) with all sample batches, the buffer locations in $\mathcal{B}'$ are the possible locations to insert buffers, as shown in the inner loop in Fig. 4.3. In this loop, we also relax the number of buffers from N_b to βN_b in the constraint (4.11) ($\beta = 1.5$ in our experiments) to increase the coverage of potential buffer locations captured by the subsequences. We will use a group technique to reduce the number of buffers back to N_b after all location candidates are captured. The inner iterative flow stops if no new buffer is added into the buffer set $\mathcal{B}'$ in the past three iterations.

After processing all sample batches in the inner loop, we execute the iterative buffer allocation flow as the outer loop in Fig. 4.3. In these iterations, only the buffer candidates in $\mathcal{B}$ need to be modeled with variables c_i as in (4.8) and only the delays of paths connected to these buffer candidates need to be sampled as (4.5)–(4.7). Consequently, more samples can be processed in one iteration so that the number of batches $\lceil N_f / N_t \rceil$ can be reduced. With these outer iterations, buffer locations are gradually refined and the outer loop finishes if the number of batches cannot be decreased.

4.1.6 Reducing Buffer Area by Tuning Concentration and Grouping

After solving the optimization flow in Fig. 4.3, the locations of inserted buffers for profit improvement are determined. However, the sizes of the buffers are not dealt with. In this section, a method is introduced to concentrate tuning values toward each other and to group buffers thereafter.

The concept of area reduction can be illustrated using Fig. 4.5. After executing the iterative buffer allocation in Fig. 4.3, the tuning values of a buffer in all samples may be scattered in a wide range such as in Fig. 4.5a, because the solver only minimizes the number of buffers, but does not consider the relation between the tuning values

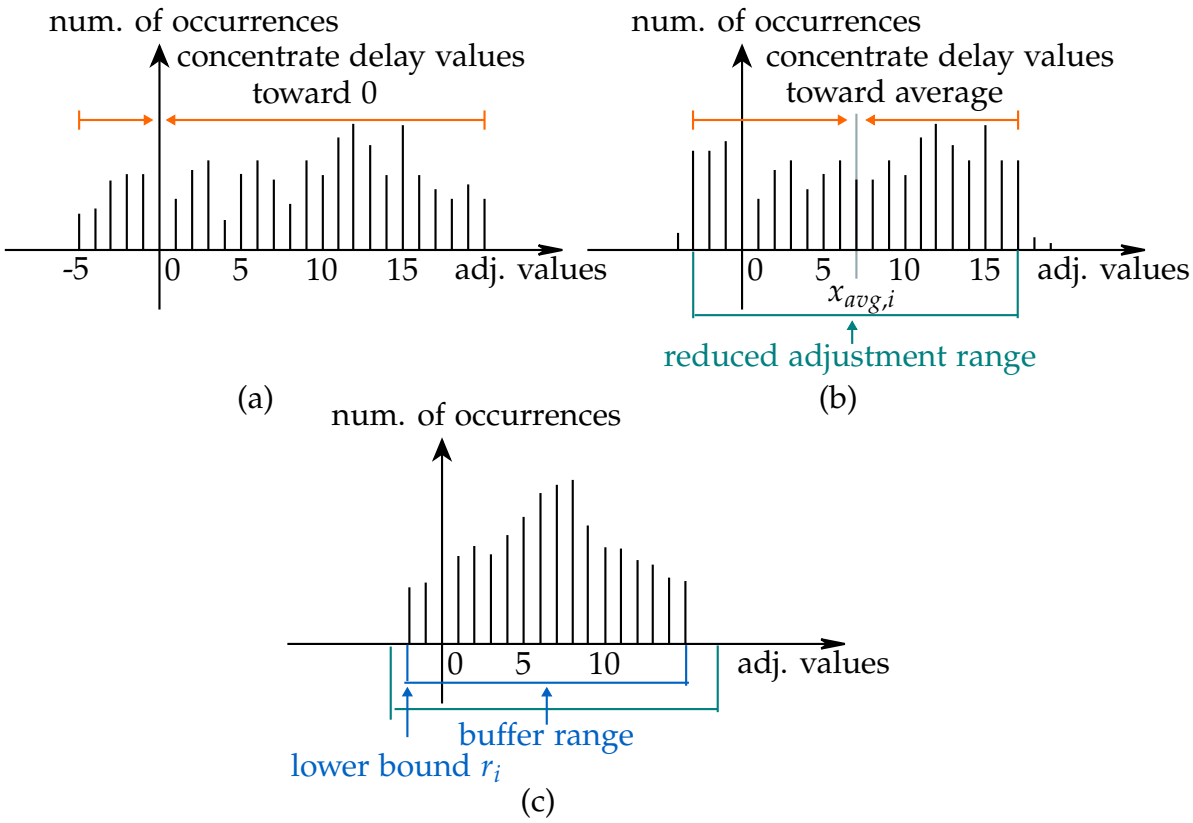

Figure 4.5: Concentrating tuning values of a buffer in all samples. The x-axis represents the adjusted delays of the buffer in all samples, and the y-axis the number of occurrences of the discrete delay values. (a) Scattered tuning values. (b) Tuning values concentrated toward zero. (c) Reduced buffer range after concentrating tuning values toward the average.

of different samples, so that it only returns one of the many feasible tuning combinations. If we can concentrate the tuning values toward each other, the real ranges of the buffers which cover all the tuning values appearing in the samples can be reduced. In addition, the concentrated tuning values may exhibit a high correlation by forming similar trends of tuning values as in Fig. 4.5c. This resemblance can thus be used to group buffers.

To push the scattered tuning values into a narrower range, we minimize their absolute values in the optimization, as illustrated in Fig. 4.5a. In this way, the solver tries to return the buffer values around 0 as much as possible using only the buffer candidates in $\mathcal{B}$ and guaranteeing the profit $\mathcal{P}$ calculated by executing the flow in Fig. 4.3. This process is formulated as follows.

$$\text{minimize} \quad \sum_{i \in I_{\mathcal{B}}, k} |x_i^k| \tag{4.18}$$

$$\begin{aligned} \text{s.t.} \quad & (4.5)\text{–}(4.7), (4.9)\text{–}(4.11), (4.13)\text{–}(4.15), \\ & \text{w.r.t. all flip-flops pair indexed by } (i, j), \\ & \text{and } k = 1, \ldots N_f, \text{ and} \end{aligned} \tag{4.19}$$

$$\sum_{m=1}^{N_p} p_m y_m \geq \mathcal{P} \tag{4.20}$$

where $I_{\mathcal{B}}$ is the index set of all buffer locations in $\mathcal{B}$. The objective function (4.18) can be transformed into a linear form easily as explained in [CBD11].

The difference between the optimization problem (4.18)–(4.20) and the optimization problem (4.16)–(4.17) includes: 1) the objective becomes the sum of the absolute values of all tuning values; 2) the buffer candidates are narrowed as the buffer set $\mathcal{B}$ returned by the flow in Fig. 4.3; 3) the profit becomes a constraint to guarantee the tuning range concentration does not affect the profit. By solving the problem (4.18)–(4.20), all tuning values are pushed toward zero as illustrated in Fig. 4.5b, so that the buffer ranges become more compact.

The second technique to reduce area cost is to group buffers that have similar tuning patterns into one buffer. For example, if two buffers have very similar tuning values in all samples, only one buffer needs to be inserted in the circuit and the delayed

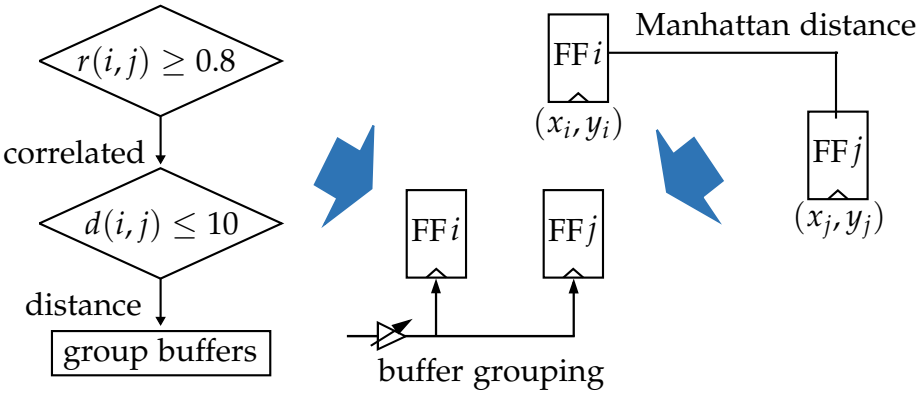

Figure 4.6: Buffer grouping according to tuning correlation and distance. Correlation threshold $r(i,j)$ is set to 0.8. Distance threshold $d(i,j)$ between buffers is set to ten times of the minimum distance between flip-flops.

clock signal is connected to two flip-flops. To make the patterns in buffer tuning more obvious, we first calculate the weighted average of all tuning values of a buffer after solving (4.18)–(4.20). Afterwards, the buffer tuning values are pushed further toward this average. This process makes the number of different tuning values smaller, so that it is easier for two buffers to have similar tuning patterns. The result of this step is that buffer tuning values may form a peak at the tuning average as illustrated in Fig. 4.5c. This step is very similar to the problem formulation in (4.18)–(4.20), except that the optimization objective is replaced by

$$\text{minimize} \quad \sum_{i \in I_B, k} |x_i^k - x_{avg,i}| \tag{4.21}$$

where $x_{avg,i}$ is the weighted average of all tuning values calculated from the result of solving (4.18)–(4.20).

After tuning values are concentrated, we try to cover all the tuning values using the smallest range window. The upper bound of the size of this range window is predefined as τ_i in (4.3). As shown in Fig. 4.5c, the range window slides along the x-axis. Since the y-axis represents the numbers of the corresponding tuning value occurrences in all samples, the total number of buffer tunings covered by the window is the sum of the tuning occurrences in the window. For yield improvement, we select the range window that covers the largest number of tunings, indicating that these tuning values are feasible in post-silicon configuration. The other values that fall out of the window are discarded. With this step, both the buffer size τ_i and

the lower bound r_i in (4.3) are determined.

In the last step of buffer insertion, we group buffers with similar tuning values to reduce the number of buffers inserted into the circuit. Buffers in the same group are implemented by only one physical buffer and the tuning values are shared by all the flip-flops connected to the buffer. The concept of grouping is illustrated in Fig. 4.6.

In grouping buffers, we first calculate the correlation coefficients of tuning values of buffer pairs. If the mutual correlation coefficients between several buffers are all above the threshold $r(i, j)$ and their distance is smaller than $d(i, j)$, they are grouped together and implemented with only one physical buffer. In practice, designers can also constrain the total number of buffers in the circuit as N_b. If the number of buffers after grouping still exceeds the specified number, the buffers with the fewest tunings are removed until the number of buffers meets the specification.

4.1.7 Acceleration Techniques

To improve the efficiency of the proposed method, we sample statistical delays between flip-flops directly instead of sampling delays of combinational gates. For example, the delays in (4.1) and (4.2) are calculated using a statistical timing engine only once. We then generate a Sobol sequence from these statistical delays directly, instead of executing a static timing analysis algorithm for each sample.

In addition, we filter connections between flip-flops according to their statistical distributions. If the 3σ delay of a path is still small enough not to affect the circuit performance, this path is not included when creating the constraints (4.1)–(4.2). For example, in the constraint (4.1) we first set x_i to the largest value and x_j to the smallest value in the range windows, respectively, and $\overline{d}_{ij}$ and s_j to their 3σ values. If this extreme setting still allows this path to work with a clock period in the fastest bin, this path is simply discarded from the problem formulation. Similarly, we also filter hold time constraints (4.2) according to the -3σ values of path delays.

4.2 Post-Silicon Tunable Buffer Configuration after Manufacturing

Post-silicon tunable buffers are inserted in chips during the design phase. After manufacturing, path delays in chips become deterministic. For a chip with timing failures, the delays of combinational paths related to tunable buffers should be evaluated. Thereafter, the configuration values of tunable buffers are determined by finding a feasible solution with respect to the given clock period T. The most challenging task of applying this post-silicon tuning technique is delay evaluation of combinational paths after manufacturing. These delays should be estimated relatively accurately to configure buffers properly. But the cost of this delay test on all failed chips must remain low; otherwise, the benefit of using tunable buffers to improve yield may be offset by the ensuing test cost.

In the previous methods, path delays are measured straightforwardly using frequency stepping. In this technique, a path is tested with a given clock period. If the sink flip-flop of this path can latch data correctly, the setup time constraint at the sink flip-flop is met, so that an upper bound of the path delay is found. Thereafter, a smaller clock period is applied until data cannot be latched correctly anymore to find a lower bound of the path delay. With a binary search of different frequency steps, the path delay can be approximated by narrowing the range defined by the lower and upper bounds.

In using frequency stepping in this test scenario, the number of iterations (frequency steps) might be large if many paths are tested. Though there are some techniques that can be used to combine tests of several paths to reduce the number of iterations, no method has considered the fact that the tunable buffers in the circuit can be used to align path delays, so that a clock period can sweep the delay ranges of several paths at the same time. For example, if delays of tunable buffers in Fig. 4.1c could be set to values as shown, the delays of the combinational paths are well balanced and can thus be tested concurrently.

In this section, the proposed method EffiTest2 based on the previous method in [ZLS16b] is introduced to reduce the total number of frequency stepping iterations in testing path delays with two techniques: statistical prediction and delay alignment

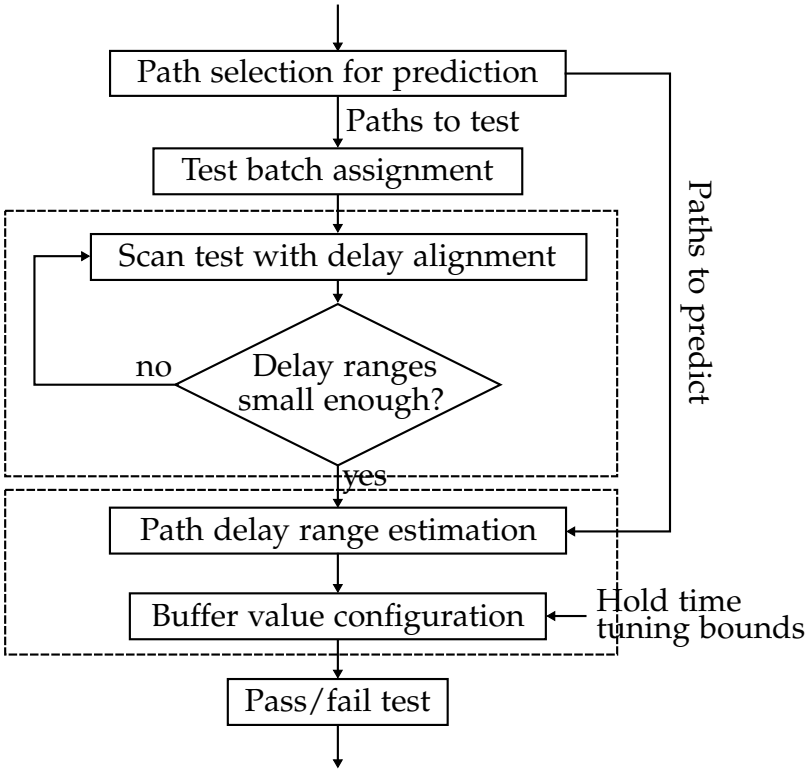

Figure 4.7: Delay test and buffer configuration flow in EffiTest2

during test. With the tested and estimated delays, tunable buffers in chips with timing failures are then configured to maximize the chance that manufactured chips work with the given clock period T. In the test scenario, we assume that the locations of buffers have been determined, using a method in Section 4.1. The flow of the proposed method is summarized in Fig. 4.7. It includes four major steps: path selection in Section 4.2.1, test batch assignment in Section 4.2.2, frequency stepping with delay alignment in Section 4.2.3, buffer configuration in Section 4.2.4, and hold time constraints in Section 4.2.5. The important notations used in the following are listed in Table 4.2.

4.2.1 Path Selection and Statistical Delay Prediction

To tune manufactured chips with timing failures, the maximum delays between flip-flop pairs need to satisfy setup time constraints, and the minimum delays hold time constraints, as defined in (4.1)–(4.2). Fig. 4.8 illustrates an example, where only setup time constraints are considered. In this example, nodes represent flip-flops,

Table 4.2: Notations

$\mathbf{P}$	All critical combinational paths between pairs of flip-flops with at least one tunable buffer
$\mathbf{D}^p$	The statistical delays of combinational paths in $\mathbf{P}$
$\mathbf{D}^m$	The statistical maximum delays between pairs of flip-flops with at least one tunable buffer
$\mathbf{P}_t$	The combinational paths that are tested using frequency stepping
$\mathbf{D}_t^p$	The statistical delays of combinational paths in $\mathbf{P}_t$
$\mathbf{D}_t^m$	The selected maximum delays that can predict $\mathbf{D}^m \setminus \mathbf{D}_t^m$ within a given accuracy
$\mathbf{P}_c$	The chosen combinational paths for maximum delays in $\mathbf{D}_t^m$
$\mathbf{D}_c$	The statistical delays of combinational paths in $\mathbf{P}_c$

solid edges represent combinational paths and dashed edges represent maximum path delays between flip-flops. If a path between a pair of flip-flops is critical, the clock edge to the flip-flop in the middle can be tuned to the other side to give the critical path more timing budget, provided that the timing constraints between the other pair of flip-flops are not violated.

To determine the configuration of tunable buffers attached to the flip-flops with respect to the setup time constraint (4.1), the maximum delays between flip-flops need to be evaluated. Since process variations affect combinational paths in manufactured chips differently, many paths between a pair of flip-flops may be critical after manufacturing. Due to test cost, it is impractical to test all combinational paths that can potentially become critical with frequency stepping directly, as assumed in [TBCS04, LN14, NK08, TGB09]. Instead, statistical delay prediction can be deployed to estimate the maximum delays between flip-flops using the data of representative combinational paths.

Statistical delay prediction requires a strong correlation between path delays to guarantee a high accuracy. Since a high correlation indicates that two delays vary similarly in manufactured chips, the measurement of one delay after manufacturing also contains information about the other. In high-performance designs, logic gates

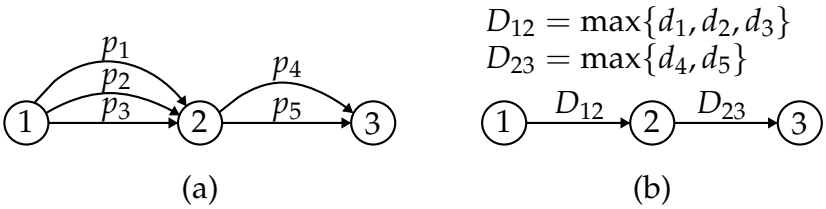

Figure 4.8: Test scenario with maximum path delays, where nodes represent flip-flops with tunable buffers. Multiple combinational paths, p_1–p_5 with delays d_1–d_5, respectively, exist between flip-flops with tunable buffers in (a). Post-silicon skew configuration by tunable buffers is determined by the maximum delays of all paths as simplified in (b).

on a critical path usually are not spread out all over the chip. Therefore, critical paths converging at or leaving from flip-flops with buffers tend to form physical clusters on the chip. This physical proximity results in a high correlation between path delays [BCSS08], which can be exploited to reduce the number of paths to be tested. For example, a conditional statistical prediction technique [JW07] has been used in [LS09] to predict the timing performance of a circuit from the measurements of on-chip test structures.

Consider a pair of flip-flops to at least one of which a tunable buffer is attached. Under process variations, usually many combinational paths between this pair of flip-flops have a probability to dominate the rest of them. We denote all these paths in the circuit as a set $\mathbf{P}$ and their statistical delays as $\mathbf{D}^p$. The task of delay measurement is to extract sufficient information about delays by testing only a small subset of paths $\mathbf{P}_t \subset \mathbf{P}$. Assume the statistical delays of $\mathbf{P}_t$ are denoted as $\mathbf{D}_t^p$. The statistical prediction from $\mathbf{D}_t^p$ to $\mathbf{D}^p \backslash \mathbf{D}_t^p$ is a well-known problem and has been studied extensively, e.g., in [FYCT13].

In post-silicon tuning, the individual delays of paths in $\mathbf{D}^p$, however, are not needed, and only the maximum delays between each pair of flip-flops should be determined as illustrated in Fig. 4.8(b). When process variations are considered, path delays can be represented as random variables. The maximum of a set of path delays between pairs of flip-flops can be calculated using Monte Carlo simulation or statistical timing analysis. Assume the statistical maximum delays are collected in a set $\mathbf{D}^m$, which contains one statistical maximum delay for every pair of flip-flops to at least one of which a tunable buffer is attached. We then need to establish the relation between

the delays $\mathbf{D}_t^p$ of selected combinational paths $\mathbf{P}_t \subset \mathbf{P}$ to $\mathbf{D}^m$, i.e., $\mathbf{D}_t^p \rightarrow \mathbf{D}^m$.

To identify $\mathbf{P}_t$ from $\mathbf{P}$, we need to consider all the combinational paths between each pair of flip-flops with at least one tunable buffer. This leads to a huge amount of paths to be examined. To solve this problem, a second level of statistical prediction is introduced. Instead of predicting the maximum delays $\mathbf{D}^m$, we use the measured delays $\mathbf{D}_t^p$ to predict a subset $\mathbf{D}_t^m \subset \mathbf{D}^m$, provided that the predicted values of $\mathbf{D}_t^m$ can also provide sufficient information for the other maximum delays $\mathbf{D}^m \backslash \mathbf{D}_t^m$, thus establishing a chained relation $\mathbf{D}_t^p \rightarrow \mathbf{D}_t^m \rightarrow \mathbf{D}^m$. Since $\mathbf{D}_t^m$ is a subset of $\mathbf{D}^m$, to identify it from $\mathbf{D}^m$ is the same statistical prediction problem as discussed in [FYCT13]. Therefore, we only need to focus on the task to find a set of combinational paths to predict $\mathbf{D}_t^m$. The information of $\mathbf{D}_t^m$ is derived from the delays of the combinational paths from which $\mathbf{D}_t^m$ is computed. This characteristic allows us to search only the combinational paths between those pairs of flip-flops corresponding to $\mathbf{D}_t^m$, thus reducing the effort of path enumeration significantly.

The relation between the delay sets discussed above is illustrated in Fig. 4.9. In the following, we firstly identify the representative maximum delays $\mathbf{D}_t^m$ from $\mathbf{D}^m$ to reduce the path search scope. Thereafter, the combinational paths between the pairs of flip-flops whose maximum delays are $\mathbf{D}_t^m$ are examined to select the combinational paths $\mathbf{D}_t^p$ for delay test.

To identify the representative maximum delays $\mathbf{D}_t^m$ from $\mathbf{D}^m$, an algorithm based on SVD-QRcp is used that has been used previously such as in [FYCT13, YW99, XD10, FYCT15]. Since the identified $\mathbf{D}_t^m$ from $\mathbf{D}^m$ should provide sufficient prediction accuracy, before the algorithm to determine the $\mathbf{D}_t^m$ from $\mathbf{D}^m$ is introduced, the prediction accuracy is explained as follows.

Assume in a general case that N statistical variables $\mathbf{D}_t$ that are selected to measure their values $\mathbf{d}_t$ in a chip directly, and another variable d_k whose value should be predicted with $\mathbf{d}_t$. Assume also that these delays follow Gaussian distributions, which are widely used in statistical timing analysis [BCSS08]. Under this assumption, these delays can be written together as $\mathbf{D} = \begin{bmatrix} d_k \\ \mathbf{D}_t \end{bmatrix} \sim \boldsymbol{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu}$ is the mean value vector of $\mathbf{D}$, $\boldsymbol{\Sigma}$ is the covariance matrix of $\mathbf{D}$, $d_k \sim N(\mu_k, \sigma_k)$ and $\mathbf{D}_t \sim \boldsymbol{N}(\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$.

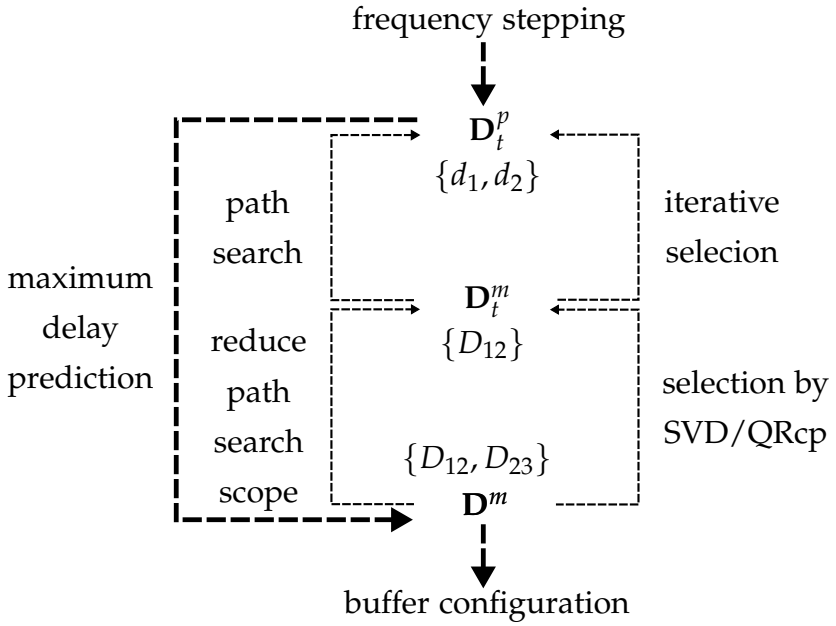

Figure 4.9: Relation between delay sets, using the test scenario in Fig. 4.8 as example. The thin dashed lines represent the relation between the delay sets for identifying representative combinational paths. The thick dashed lines illustrate the real test and prediction procedure.

Accordingly, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can be expressed as $\boldsymbol{\mu} = \begin{bmatrix} \mu_k \\ \boldsymbol{\mu}_t \end{bmatrix}$, and $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_k & \boldsymbol{\Sigma}_{k,t} \\ \boldsymbol{\Sigma}_{t,k} & \boldsymbol{\Sigma}_t \end{bmatrix}$, where $\boldsymbol{\Sigma}_{k,t} = \boldsymbol{\Sigma}_{t,k}^T$ is the covariance matrix between d_k and $\mathbf{D}_t$.

With the measured values $\mathbf{d}_t$ of $\mathbf{D}_t$, the mean value μ_k' and the variance $\sigma_k'^2$ of d_k under the condition $\mathbf{D}_t = \mathbf{d}_t$ can be expressed as follows [JW07].

$$\mu_k' = \mu_k + \boldsymbol{\Sigma}_{k,t}\boldsymbol{\Sigma}_t^{-1}(\boldsymbol{d}_t - \boldsymbol{\mu}_t) \tag{4.22}$$

$$\sigma_k'^2 = \sigma_k^2 - \boldsymbol{\Sigma}_{k,t}\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}_{t,k}. \tag{4.23}$$

After delay prediction, d_k is still a random variable because there are purely random process variations that reduce the correlation between delays. However, the variance of the predicted delays becomes smaller due to the second product term in (4.23), indicating that the real path delay d_k in a chip is confined into a small range with a nonnegligible probability. This range reduction results from the fact that the measurement results of $\mathbf{d}_t$ provide the information of the shared random components between d_k and $\mathbf{D}_t$ to reduce the variability of d_k. Therefore, it may be unnecessary

to measure the exact delay of d_k for buffer configuration after delay prediction if the correlation between d_k and $\mathbf{D}_t$ is high. On the other hand, a small correlation allows the delay d_k to vary freely, leading to a relatively large variance even after statistical prediction. Since the standard deviation σ' represents how wide the distribution of the predicted value of d_k spreads, we use it as an indicator of the prediction accuracy. If σ' is lower than a given threshold σ_{th}, the predicted value is considered as having a sufficient accuracy.

To identify a set of maximum delays $\mathbf{D}_t^m$ from $\mathbf{D}^m$ with sufficient prediction accuracy, an algorithm based on SVD-QRcp is proposed. Assume that a delay from $\mathbf{D}^m$ is written as a linear combination of M random components $\mathbf{S} = [s_1, s_2, \ldots, s_M]^T$, such as in the canonical form in [VRK+04]. The delay $\mathbf{D}^m$ can then be expressed as $\mathbf{D}^m = \mathbf{CS}$, where $\mathbf{C}$ is the coefficient matrix. The SVD-QRcp algorithm first performs Singular Value Decomposition (SVD) to decompose $\mathbf{C}$ as

$$\mathbf{C} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \tag{4.24}$$

where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices and $\mathbf{\Lambda}$ is a diagonal matrix with singular values in a descending order.

The large singular values in $\mathbf{\Lambda}$ reveal the importance of delays that carry orthogonal statistical information. To select the delays $\mathbf{D}_t^m$ to predict $\mathbf{D}^m \backslash \mathbf{D}_t^m$ the correspondence between the singular values and the delays in $\mathbf{D}^m$ needs to be established using the permutation matrix in the QRcp (QR with column pivoting) decomposition. Assume n delays should be selected from $\mathbf{D}^m$. Then the first n columns of $\mathbf{U}$, written as $\mathbf{U}_{[1:n]}$ are decomposed as

$$\mathbf{U}_{[1:n]}^T = \mathbf{QR}\mathbf{\Pi}^T \tag{4.25}$$

where $\mathbf{\Pi}^T$ is a permutation matrix to identify the n most important random variables from $\mathbf{D}^m$.

To illustrate the decomposition process above, we use an example with three delays in $\mathbf{D}^m$, each of which is expressed as a linear combination of three random components. The SVD and QRcp are performed using the routines from the LAPACK [ABB+99] and GSL [G+] libraries. The coefficient matrix $\mathbf{C}$ of $\mathbf{D}^m$ and the

matrices after decomposition are shown in the following.

$$\underset{\mathbf{C}}{\begin{bmatrix}10 & 6 & 1\\ 13 & 4 & 2\\ 7 & 5 & 1\end{bmatrix}} = \underset{\mathbf{U}}{\begin{bmatrix}-0.59 & 0.43 & -0.68\\ -0.69 & -0.72 & 0.13\\ -0.43 & 0.55 & 0.72\end{bmatrix}} \times \underset{\mathbf{\Lambda}}{\begin{bmatrix}19.83 & 0 & 0\\ 0 & 2.76 & 0\\ 0 & 0 & 0.31\end{bmatrix}} \times \underset{\mathbf{V}^T}{\begin{bmatrix}-0.90 & -0.40 & -0.18\\ -0.42 & 0.90 & 0.10\\ -0.12 & -0.16 & 0.98\end{bmatrix}}$$

$$\mathbf{U}^T_{[1:2]} = \underset{\mathbf{Q}}{\begin{bmatrix}-0.69 & -0.72\\ -0.72 & 0.69\end{bmatrix}} \times \underset{\mathbf{R}}{\begin{bmatrix}0.99 & 0.09 & -0.10\\ 0 & 0.72 & 0.69\end{bmatrix}} \times \underset{\mathbf{\Pi}^T}{\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}}$$

In the example above, two delays are selected to predict the third one, so that only the first two columns of $\mathbf{U}$, written as $\mathbf{U}_{[1:2]}$, are used in the QRcp decomposition. In the permutation matrix $\mathbf{\Pi}^T$, the first column shows that we need to select the second delay because the only 1 in this column appears in the second row. Similarly, the second column of $\mathbf{\Pi}^T$ shows that we need to select the first delay.

The decomposition process above requires that we state the number of delays n to be included in $\mathbf{D}^m_t$. This number needs to be determined so that the concept of prediction accuracy is maintained. To achieve this, we check whether the standard deviation σ' of a delay from $\mathbf{D}^m \backslash \mathbf{D}^m_t$ exceeds a given threshold σ_{th}. If it exceeds σ_{th}, we increase the number of delays from $\mathbf{D}^m$ to be selected and rerun the QRcp decomposition. Since all delays in $\mathbf{D}^m$ contain a purely random component from process variations [BCSS08, VRK+04], σ' cannot be reduced to zero. Instead, it must be larger than the standard deviation of the corresponding purely random component. In EffiTest2, we enumerate all the delays in $\mathbf{D}^m$ to identify the maximum σ_{max} of the standard deviations of all the purely random components and use $\sigma_{th} = 2\sigma_{max}$ as the threshold of the prediction accuracy. Therefore, the iterations should always converge because the given threshold σ_{th} is larger than σ_{max}, which is the accuracy when all delays are measured directly. The selection process of $\mathbf{D}^m_t$ from $\mathbf{D}^m$ is summarized in Algorithm 2.

Algorithm 2: Select $\mathbf{D}_t^m$ from $\mathbf{D}^m$ to reduce path search scope

Input : Coefficient matrix $\mathbf{C}$ of maximum delays $\mathbf{D}^m$ from SSTA
Output: Representative maximum delays $\mathbf{D}_t^m \subseteq \mathbf{D}^m$

```
1  U ← Decompose C using SVD (4.24);
2  for i ← 1 to |D^m| do
3      U_[1:i] ← First i columns of U;
4      Π^T ← Decompose U_[1:i]^T using QRcp (4.25);
5      Select D_t^m from D^m using Π^T;
6      foreach d_k ∈ D^m \ D_t^m do
7          Compute σ_k'^2 using (4.23);
8          if σ_k' > σ_th then
9              goto L2;
10         end
11     end
12     break;
13 end
15 return D_t^m
```

The maximum delays $\mathbf{D}_t^m$ returned by Algorithm 2 are actually used to narrow the search scope of combinational paths for delay test. In manufactured chips, only the delays of these combinational paths can be measured with frequency stepping directly [Pat03, LPR+03]. After $\mathbf{D}_t^m$ is identified from $\mathbf{D}^m$, we scan the circuit to find the starting and ending flip-flops corresponding to $\mathbf{D}_t^m$. For example, in Fig. 4.9 the maximum delay D_{12} is identified from the set $\{D_{12}, D_{23}\}$ using the SVD-QRcp method described above. This maximum delay indicates that the combinational paths between the flip-flops 1 and 2 in Fig. 4.8 are candidates for delay test. To reduce test cost, only the minimum number of paths from them should be tested using frequency stepping for post-silicon configuration. For example, the delays of paths p_1 and p_2 may already provide sufficient accuracy in predicting the maximum delays $\{D_{12}, D_{23}\}$, while more paths in addition to them may not improve the prediction accuracy further, because the purely random components in the maximum

delays then dominate the predicted values.

The number of combinational paths between a pair of flip-flops is usually very large, so that the path candidates related to $\mathbf{D}_t^m$ need to be reduced further. Since the combinational paths related to the same pair of flip-flops are generally located close to each other on the die, their delays exhibit a high correlation due to proximity. Therefore, we need to consider only a small subset of paths between each pair of flip-flops. A static critical path identification is used in our method to extract five combinational paths for each maximum delay in $\mathbf{D}_t^m$ by forward and backward arrival time propagation. The extracted combinational paths are denoted as a set $\mathbf{P}_c$. The delays of these paths are denoted as a set $\mathbf{D}_c$.

The final step for path selection is to choose $\mathbf{P}_t$ from $\mathbf{P}_c$. The objective is that the measured values of the selected paths $\mathbf{P}_t$ should be able to predict the delays of $\mathbf{D}^m$ with a sufficient accuracy. A new challenge in this step is that the set of delays $\mathbf{D}_c$ is not a subset of $\mathbf{D}^m$, so that the SVD-QRcp method cannot be used to identify the paths $\mathbf{P}_t$. To solve this problem, all the path delays in $\mathbf{D}_c$ are enumerate and select the delay that can reduce the maximum of the variances of the predicted values of $\mathbf{D}^m$ the most. The selection step stops when the maximum of the standard deviations σ_k' is smaller than the threshold σ_{th} as used in Algorithm 2.

The procedure of selecting combinational paths for delay test is summarized in Algorithm 3. In L1–L7 the combinational paths related to $\mathbf{D}_t^m$ are saved in $\mathbf{P}_c$ and their statistical delays in $\mathbf{D}_c$. To select representative paths from $\mathbf{P}_c$, the loop L9–L31 adds one delay d_{next} from $\mathbf{D}_c$ into $\mathbf{D}_t^p$ in each iteration. The newly selected delay d_{next} is the one from $\mathbf{D}_c$ that, together with the already selected delays in $\mathbf{D}_t^p$, predicts the maximum delays $\mathbf{D}^m$ with the best accuracy. To identify this delay, each delay in $\mathbf{D}_c \backslash \mathbf{D}_t^p$ is evaluated in L12–L25 as d_c, where d_c and the current $\mathbf{D}_t^p$ are combined as $\mathbf{D}_t^{p\prime}$ to evaluate the accuracy of predicting the maximum delays $\mathbf{D}^m$ in the loop L15–L20 using (4.23). The prediction accuracy is indicated as the maximum variance σ_{max}^2 of predicted values of $\mathbf{D}^m$, and the delay d_c that can produce the smallest σ_{max}^2 is selected and added into $\mathbf{D}_t^p$. Meanwhile, the accuracy indicator σ_{max}^2 is assigned to σ_{next}^2. When σ_{next} becomes lower than the threshold σ_{th}, the selection procedure finishes and the current $\mathbf{P}_t$ is returned as the paths to be tested using frequency stepping.

Algorithm 3: Path selection to predict maximum delays $\mathbf{D}^m$

Input : Maximum delays $\mathbf{D}^m$ from SSTA
Delay set $\mathbf{D}_t^m$ from Algorithm 2
Output: Selected paths $\mathbf{P}_t$ for delay test

$\mathbf{P}_c \leftarrow \emptyset$;
foreach $d_k \in \mathbf{D}_t^m$ **do**
  $\{\text{ff}_{src}, \text{ff}_{dst}\} \leftarrow$ Find flip-flops corresponding to d_k;
  $\mathbf{P}_s \leftarrow$ Trace five most critical paths $\text{ff}_{src} \rightarrow \text{ff}_{dst}$;
  $\mathbf{P}_c \leftarrow \mathbf{P}_c \cup \mathbf{P}_s$;
end
$\mathbf{D}_c \leftarrow$ Delays of $\mathbf{P}_c$;
$\mathbf{D}_t^p \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** $|\mathbf{D}_c|$ **do**
  $\sigma_{next}^2 \leftarrow \infty$;
  $d_{next} \leftarrow null$;
  foreach $d_c \in \mathbf{D}_c \backslash \mathbf{D}_t^p$ **do**
    $\mathbf{D}_t^{p\prime} \leftarrow \{d_c\} \cup \mathbf{D}_t^p$;
    $\sigma_{max}^2 \leftarrow 0$;
    foreach $d_k \in \mathbf{D}^m$ **do**
      Compute $\sigma_k'^2$ from $\mathbf{D}_t^{p\prime}$ to $\mathbf{D}^m$ using (4.23);
      if $\sigma_k'^2 > \sigma_{max}^2$ **then**
        $\sigma_{max}^2 \leftarrow \sigma_k'^2$;
      end
    end
    if $\sigma_{max}^2 < \sigma_{next}^2$ **then**
      $\sigma_{next}^2 \leftarrow \sigma_{max}^2$;
      $d_{next} \leftarrow d_c$;
    end
  end
  $\mathbf{D}_t^p \leftarrow \{d_{next}\} \cup \mathbf{D}_t^p$;
  if $\sigma_{next} \leq \sigma_{th}$ **then**
    $\mathbf{P}_t \leftarrow$ Paths corresponding to $\mathbf{D}_t^p$;
    break;
  end
end
return $\mathbf{P}_t$

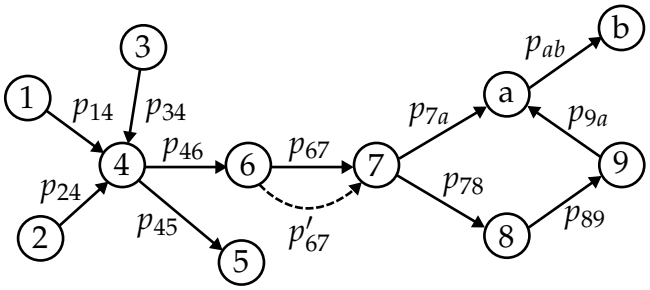

Figure 4.10: Test scenario with multiple combinational paths. The nodes represent flip-flops. The solid edges represent combinational paths whose delays need to be tested using frequency stepping. The dashed edges represent an additional path that can also be tested without increasing the number of test batches.

4.2.2 Path Test Multiplexing

To predict the maximum delays $\mathbf{D}^m$ between flip-flops, the delays $\mathbf{D}_t^p$ of representative combinational paths $\mathbf{P}_t$ need to be tested. In frequency stepping, the delay of a path is compared with the period of the test clock signal by checking whether the sink flip-flop of the path latches data correctly. A violation of the setup time constraint indicates the maximum delay is larger than the test clock period. Since the data latching state of a flip-flop can only indicate whether there is a timing violation, only the delay of one path converging to it can be tested in one clock cycle. In addition, paths leaving from a flip-flop cannot be tested in parallel, because the values of the flip-flops need to be set to trigger specific paths. In practice, the constraints may be relaxed because some paths can share parts of test patterns. In the following discussion, we will only assume the strictest case without allowing this sharing of test vectors to simplify the description of the proposed test multiplexing, which can be adapted easily to deal with the relaxed test scenarios.

Consider the test scenario shown in Fig. 4.10, where the nodes represent flip-flops and the edges represent combinational paths. During delay test, the paths p_{14}, p_{24}, and p_{34} cannot be processed in parallel, because they converge at the same flip-flop. Similarly paths p_{45} and p_{46} cannot be tested at the same time due to the shared source flip-flop. On the contrary, paths that can be tested in parallel can be arranged into the same group. For example, paths p_{14}, p_{46}, p_{67}, p_{7a}, and p_{ab} can be tested

with the same clock period together. These paths are called a *batch* in the following discussion. In real test scenarios, there might be cases that some paths in a test batch cannot be activated by ATPG vectors at the same time. These paths can be set as mutually exclusive and arranged into different test batches. The proposed method does not consider the logic inconsistencies that might arise while activating/propagating faults. However, we can use existing methods, e.g., [MT05] and [PRU95], to obtain testing compatibility, i.e., subsets of paths which can be tested simultaneously for a given set of paths that need to be tested. Accordingly, testing compatibility of the representative combinational paths $\mathbf{P}_t$ after path selection in Algorithm 3 can be derived with these methods. The compatibility of $\mathbf{P}_t$ can be incorporated into path test multiplexing to generate test batches in which paths can be tested simultaneously.

Since the delays of paths in a test batch can be measured in parallel, naturally we should arrange paths to be tested into as few batches as possible to reduce the overall number of frequency stepping iterations. To identify the minimum number of test batches, we formulate this path arrangement task into an Integer Linear Programming (ILP) problem.

Assume there are N_t ($|P_t| = N_t$) paths $p_1, p_2, \ldots, p_{N_t}$ to be tested. For the path p_j, we assign a 0-1 variable $b_{i,j}, i = 1, 2, \ldots, N_t$ to indicate whether p_j is assigned into the ith batch. For all the selected paths, the variables can be written into an $N_t \times N_t$ submatrix, as shown in the first N_t columns of the following matrix,

$$\left(\begin{array}{cccc|ccc} b_{1,1} & b_{1,2} & \ldots & b_{1,N_t} & b_{1,N_t+1} & \ldots & b_{1,N_t+N_a} \\ b_{2,1} & b_{2,2} & \ldots & b_{2,N_t} & b_{2,N_t+1} & \ldots & b_{2,N_t+N_a} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{N_t,1} & b_{N_t,2} & \ldots & b_{N_t,N_t} & b_{N_t,N_t+1} & \ldots & b_{N_t,N_t+N_a} \end{array}\right) \tag{4.26}$$

where the columns correspond to the paths to be tested, and the rows correspond to test batches.

Because a path delay needs to be measured only once, the sum of the variables in a column in (4.26) should be equal to one, written as

$$\sum_{i=1}^{N_t} b_{i,j} = 1, \quad 1 \leq j \leq N_t. \tag{4.27}$$

To prevent paths from converging at or leaving from the same flip-flop to be arranged in the same batch, we add the following constraints for each flip-flop,

$$\sum_{p_j \in I_F} b_{i,j} \leq 1, \quad \sum_{p_j \in O_F} b_{i,j} \leq 1, \quad 1 \leq i \leq N_t \tag{4.28}$$

where I_F is the set of paths converging at the flip-flop and O_F the set of paths leaving the flip-flop.

To reduce the number of batches, the number of rows containing at least one 1 value in (4.26) should be minimized. For the ith row corresponding to the ith test batch, we assign a 0-1 variable B_i to indicate whether this test batch is occupied, so that

$$b_{i,j} \leq B_i, \quad 1 \leq j \leq N_t, \; 1 \leq i \leq N_t. \tag{4.29}$$

By minimizing $\sum_{i=1}^{N_t} B_i$, we can minimize the number of test batches that are really occupied by test paths.

In test iterations, the delays of the paths in the same test batch are always swept by the same test clock. If the delays of these paths differ significantly, the test clock can only capture the delay information of a part of them, while the other paths are swept by changing the period of the clock signal in other test iterations. Consequently, the number of iterations may have to be increased. To improve test efficiency, we arrange the paths with comparable delays into the same test batch according to their statistical delay information.

Since comparable delays in a test batch mean that large delays tend to be assigned in the same batch and small ones in other ones, we simplify the delay balancing problem in path arrangement by pushing paths with large delays into into the same test batch as much as possible. For each test batch, we assign a variable $W_i, i = 1, 2, \ldots, N_t$ to represent the sum of the delays of the paths in the ith batch. Therefore, W_i can be defined as

$$W_i = \sum_{j=1}^{N_t} b_{i,j} \mu_j, \quad 1 \leq i \leq N_t \tag{4.30}$$

where μ_j is the mean value of the jth path. If the jth path is assigned into the ith batch, its delay contributes to W_i. Afterwards, we maximize the weighted sum of $\sum_{i=1}^{N_t} \varepsilon_i W_i$, where ε_i are constants and $\varepsilon_i > \varepsilon_{i+1}$. With the weights ε_i in the descending order, the paths with large delays tend to be assigned to the first test batches to improve the efficiency of frequency stepping.

After test batches are formed, there might still be some unoccupied slots in a test batch because paths might not be distributed evenly at flip-flops with buffers. For example, the test scenario in Fig. 4.10 requires at least three test batches because there are three edges converging at node 4. Therefore, these test batches can cover not only the edge p_{67} but also p'_{67}. Because the batches of paths should be tested anyway, we add additional paths to these empty test slots to gather more delay information.

Additional paths are added according to the prediction accuracy of their corresponding maximum delays. As discussed in Section 4.2.1, the predicted standard deviation is used as an indicator of the prediction accuracy. Since a large standard deviation σ'_k calculated by (4.23) represents that the corresponding maximum delay cannot be estimated with enough accuracy, we first identify those maximum delays whose predicted standard deviations are larger than a given percentage of their original standard deviations, 10% in our framework. Thereafter, for each of these delays, we find a combinational path from the corresponding source flip-flop to the sink flip-flop to reduce the predicted variance of the delay. These newly identified combinational paths are written as a set $\mathbf{P}_a$ with delays $\mathbf{D}_a$ and $|\mathbf{D}_a| = N_a$. To incorporate these paths into the test batches, we assign 0-1 variables $b_{i,j}, i = 1, 2, \ldots, N_t, j = N_t + 1, N_t + 2, \ldots, N_t + N_a$ as shown in the extended submatrix (4.26). Since it is preferred, but not mandatorily required, to add these new paths into the test batches, their appearance in the test batches can be constrained as

$$\sum_{i=1}^{N_t} b_{i,j} \leq 1, \quad N_t \leq j \leq N_t + N_a. \tag{4.31}$$

Consequently, a new path is included into one of the test batches when the sum above is equal to 1. To incorporate the new paths into test batches as many as possible, we maximize the objective $\sum_{1 \leq i \leq N_t, N_t+1 \leq j \leq N_t+N_a} b_{i,j}$. With the new columns in (4.26), (4.29) and (4.30) should be revised to incorporate the extended indexes as

$$b_{i,j} \leq B_i, \quad 1 \leq i \leq N_t, \; 1 \leq j \leq N_t + N_a \tag{4.32}$$

$$W_i = \sum_{j=1}^{N_t+N_a} b_{i,j}\mu_j, \quad 1 \leq i \leq N_t. \tag{4.33}$$

Considering the three objectives discussed above, we formulate the path assignment

task into an ILP problem as

$$\text{Minimize} \quad \alpha \sum_{i=1}^{N_t} B_i - \beta \sum_{i=1}^{N_t} \varepsilon_i W_i - \gamma \sum_{\substack{1 \le i \le N_t \\ N_t+1 \le j \le N_t+N_a}} b_{i,j} \tag{4.34}$$

$$\text{Subject to} \quad \text{(4.27)–(4.28) and (4.31)–(4.33)} \tag{4.35}$$

where α, β and γ are constants with $\alpha \gg \beta \gg \gamma$ to guarantee the minimum number of test batches are generated. After solving this problem, only the rows with at least a one in (4.26) are kept as test batches, denoted as $\mathbf{B}$.

4.2.3 Test with Delay Alignment by Tuning Buffers

After path batches are identified, they should be tested using frequency stepping to evaluate the path delays. In this section, how the delays of paths in a single batch are measured is discussed. Note this is the only step in the proposed framework that is executed by expensive testers able to generate various clock signals with a high accuracy.

In frequency stepping, a clock period is applied to the chip under test and the paths in a test batch are triggered by test vectors. If the setup time constraint (4.1) at a flip-flop is violated, the data at this flip-flop cannot be latched correctly. This error shows that $D_{ij} + x_i - x_j$ is larger than T so that T is its lower bound. On the other hand, if the clock period is large enough so that there is no timing violation, the constraint (4.1) is met and T is an upper bound of $D_{ij} + x_i - x_j$. By applying different clock periods in a binary search style, D_{ij} can be approximated with a given accuracy.

Consider the case shown in Fig. 4.11(a), where a delay has given upper and lower bounds. These bounds are initialized with $\mu \pm 3\sigma$, where μ and σ are the mean value and the standard deviation of the delay calculated by statistical timing analysis. When the delay is tested with a given clock period T in an iteration, either a new upper bound or a new lower bound of it is generated. Consequently, the corresponding delay range is partitioned into two parts by T and the real delay value falls into one of them. To partition the delay range efficiently, it is preferable that T is aligned to the center of the range. Otherwise, T might not partition the delay range evenly, but instead slices it in small steps, leading to many test iterations to estimate the delay, as illustrated in Fig. 4.11(b).

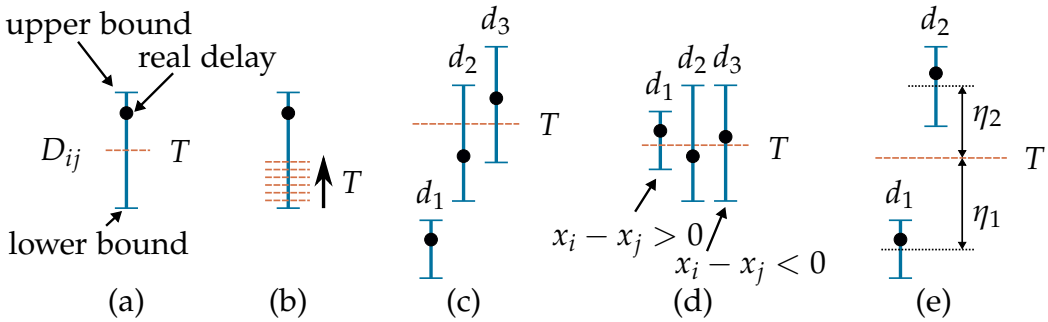

Figure 4.11: Frequency stepping and delay range alignment.

When several path delays in one test batch are considered as in Fig. 4.11(c), it is not always possible to partition all the delay ranges evenly with one clock period. However, we can still find a clock period T that partitions several delay ranges at the same time, so that the ranges of these delays can be reduced in one test iteration.

To use a clock period T to partition multiple delay ranges, there must be some overlap between the delay ranges, such as d_2 and d_3 in Fig. 4.11(c). According to (4.1), the actual constraint that is tested using T is $D_{ij} + x_i - x_j$. Since the tunable buffers are already deployed in the circuit and their values x_i and x_j can be adjusted through the scan chain, we change the value of $x_i - x_j$ to align the delay ranges, as illustrated in Fig. 4.11(d). Consequently, a clock period can partition more delay ranges so that the delays can be measured more efficiently compared with the case in Fig. 4.11(c). In EffiTest2, at-speed scan test is deployed for delay tests. At-speed scan test has been applied in [NK08], [TGB09] and investigated thoroughly in [PZCB+10]. In this method, scan chains are loaded with test vectors and two clock pulses are applied at the functional frequency. Because the configuration bits of buffers can be scanned into the chip under test together with the test vectors, the proposed technique requires no change to the existing test platform.

In real circuits, the buffer values x_i and x_j can only be adjusted in a limited range as specified by (4.3). In addition, these buffer values may affect more than one path delay. For example, in Fig. 4.10 the buffer value of node 4 affects all the paths converging at or leaving from it. To test the path delays efficiently, we need to find a proper set of buffer values to align the ranges of path delays as much as possible.

Assume that the upper and lower bounds of D_{ij} between nodes i and j are u_{ij} and l_{ij}, respectively. When the buffers at the source and sink nodes of the path are

considered, the lower bounds and the upper bounds are shifted by $x_i - x_j$ as defined in (4.1). Therefore, the distance η_{ij} between a given T and the center of the shifted range of the path delay D_{ij} can be expressed as

$$\eta_{ij} = |T - ((u_{ij} + l_{ij})/2 + x_i - x_j)|. \tag{4.36}$$

If we minimize the sum of η_{ij} from all delay ranges, the resulting T will approximate the centers of delay ranges as much as possible, while the buffer values x_i and x_j are also determined.

Minimizing the sum of η_{ij} directly, however, cannot handle the special case in Fig. 4.11(e) where the two delay ranges still do not overlap even after the buffer values have been adjusted to the limit. In this case, the sum of distances $\eta_1 + \eta_2$ is independent of where T is placed between the centers of the two ranges. To solve this problem, we sort the centers of delay ranges determined in the previous test iteration. Thereafter, we assign the weight k_0 to the range whose center is in the middle of the sorted list, and reduce the weights of other ranges by k_d successively. In the proposed method, we set $k_0 \gg k_d$, so that the ranges at the middle of the sorted list have slightly higher priorities. With this weight assignment, the weights of the two ranges in Fig. 4.11(e) are different so that the next test clock period T should align at the center of the range with the larger weight.

The optimization problem to determine the clock period T and the corresponding set of buffer values x_i and x_j to align delay ranges can thus be expressed as

$$\text{Minimize} \quad \sum_{i,j} k_{ij}\eta_{ij} \tag{4.37}$$

$$\text{Subject to} \quad \forall \text{ path } p_{ij} \text{ in the test batch}$$

$$T - ((u_{ij} + l_{ij})/2 + x_i - x_j) \leq \mathcal{M} z_{ij}^p \tag{4.38}$$

$$(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) - \eta_{ij} \leq \mathcal{M}(1 - z_{ij}^p) \tag{4.39}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) + \eta_{ij} \leq \mathcal{M}(1 - z_{ij}^p) \tag{4.40}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) \leq \mathcal{M} z_{ij}^n \tag{4.41}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) - \eta_{ij} \leq \mathcal{M}(1 - z_{ij}^n) \tag{4.42}$$

$$(T - ((u_{ij} + l_{ij})/2 + x_i - x_j) + \eta_{ij} \leq \mathcal{M}(1 - z_{ij}^n) \tag{4.43}$$

$$r_i \leq x_i \leq r_i + \tau_i,\ r_j \leq x_j \leq r_j + \tau_j \tag{4.44}$$

where (4.38)–(4.43) are linear constraints transformed from (4.36) and $\mathcal{M}$ is a very large positive constant [CBD11]; z_{ij}^p and z_{ij}^n are two 0-1 variables corresponding to the two cases that $T - ((u_{ij} + l_{ij})/2 + x_i - x_j)$ are no less than zero and no greater than zero, respectively. (4.44) defines the ranges of buffer values as in (4.3).

After the clock frequency and the corresponding buffer values are determined by solving the ILP problem (4.37)–(4.44), the paths in the current batch are tested. According to the test result, either the upper bounds or the lower bounds of their delays are updated. If the distance between the range bounds u_{ij} and l_{ij} of a path is smaller than a threshold ϵ, which is set to a constant times of the maximum of the mean values of the path delays, 0.005 in our framework, the path is removed from the current batch. The test iterations finish when all paths in the batch have been removed. The pseudocode of the test process is shown in Algorithm 4. The testing process of one test batch only requires the calculation of buffer configuration and one clock frequency. The test patterns are determined once and no adaptive test generation based on the measurements from the tester is needed.

4.2.4 Buffer Configuration with Delay Estimation

To make chips with timing failures after manufacturing work with a given clock frequency, buffers can be configured according to results of the delay test and prediction. Unlike delay alignment using existing tuning buffers to reduce the number of test iterations above, this step really configures tuning buffers so that the corresponding chips with timing failures can operate at the designated clock frequency. After a path in P_t has been tested by frequency stepping, its delay is confined to a narrow range with a lower bound and an upper bound. For another delay d_k that is not measured directly but is to be estimated, (4.22) and (4.23) are used to calculate the mean value μ_k' and the standard deviation σ_k'. According to (4.22) and (4.23), σ_k' is determined exclusively by the covariance matrix, but μ_k' is affected by $\boldsymbol{d}_t$, which are the delays measured by frequency stepping. When calculating μ_k', we use the upper bounds of $\boldsymbol{d}_t$ so that the estimated delays are conservative. Since the variances of estimated delays are often non-zero, which indicate that purely random variations still affect path delays, we assign a lower bound and an upper bound $\mu_k' - 3\sigma_k'$ and $\mu_k' + 3\sigma_k'$ for an estimated delay, so that all path delays are constrained similarly for

Algorithm 4: Test procedure with frequency stepping

Input: B: the queue of test batches

```
1  foreach B_k ∈ B do
2      while B_k contains an edge do
3          T ← solve (4.37)–(4.44);
4          test_with_frequency_stepping(B_k, T);
5          foreach p_ij in B_k do
6              if passed(p_ij) then
7                  u_ij=T − x_i + x_j;
8              else
9                  l_ij=T − x_i + x_j;
10             end
11             if u_ij − l_ij < ϵ then
12                 remove_edge(p_ij, B_k);
13             end
14         end
15     end
16 end
```

the following buffer configuration.

In the delay range defined by the lower and upper bounds, a real delay may take any value. However, due to test resolution and delay estimation, the exact location of this delay in the range is unknown. To deal with uncertainty, a conservative method to configure the buffers is to assume the upper bounds of the ranges to be path delays, so that the chip always works with the resulting buffer configuration. This method, however, may incorrectly evaluate some chips as nonfunctional due to pessimistic delay overestimation. To alleviate this problem, we try to find a buffer configuration for a chip while assuming the delays are as close to their corresponding upper bounds as possible. By minimizing the distance of the assumed delays from their corresponding upper bounds when determining the buffer configuration, the chance that the chip works after configuration becomes large, so that the final pass/fail test will accept most post-silicon configured chips as functional. Because the variances of

predicted maximum delays differ from each other, the distance to the upper bounds are also scaled by the standard deviations of the predicted delays.

The optimization problem to find a buffer configuration while minimizing the distance ξ of the assumed delays from the corresponding upper bounds is described as follows.

$$\text{Minimize} \quad \xi \tag{4.45}$$

$$\text{Subject to} \quad \forall \text{ path } p_{ij}$$

$$T_d \geq D'_{ij} + x_i - x_j \tag{4.46}$$

$$l_{ij} \leq D'_{ij} \leq u_{ij},\ \xi \geq (u_{ij} - D'_{ij})/\sigma'_{ij} \tag{4.47}$$

$$r_i \leq x_i \leq r_i + \tau_i,\ r_j \leq x_j \leq r_j + \tau_j \tag{4.48}$$

where D'_{ij} is the assumed delay value of a path during buffer configuration; σ'_{ij} is the standard deviation of the corresponding predicted delay; T_d is the designated clock period for the design; (4.46) and (4.48) are derived from (4.1) and (4.3), respectively. By solving the optimization problem (4.45)–(4.48), a set of buffer configuration values x_i and x_j can be found.

4.2.5 Tuning Bounds Due to Hold Time Constraints

In the discussion above, we do not consider hold time constraints. However, tuning buffers may affect hold time constraints significantly if they are configured improperly. For example, in Fig. 4.2, if x_j is much larger than x_i, the constraint (4.2) may be violated.

As shown in (4.2), hold time constraints are affected by $x_i - x_j$ instead of individual values of x_i and x_j. In the proposed method, we do not test against hold time violations after configuring buffers. Instead, we set a lower bound λ_{ij} for $x_i - x_j$ by sampling the statistical distribution of d_{ij} in (4.2) so that a given yield can be maintained.

Consider the case that d_{ij} in (4.2) is sampled M times for all short paths and its value in the kth sample is $d_{ij,k}$. For the kth sample, we use a 0-1 variable y_k to represent that the lower bound λ_{ij} meet

$$\lambda_{ij} - d_{ij,k} \geq \mathcal{M}(y_k - 1), \quad \text{for all short paths } p_{ij} \tag{4.49}$$

where $\mathcal{M}$ is a very large constant. The yield of the circuit with respect to hold time can thus be constrained as

$$\sum y_i / M \geq Y, \quad i = 1, 2, \ldots M \tag{4.50}$$

where Y is a given yield for hold time constraints, set to 0.99 in our method. To allow buffers to have the largest freedom in value configuration, we minimize the sum of all the lower bounds $\sum_{i,j} \lambda_{ij}$. After λ_{ij} are determined, the buffer configuration values can be constrained to avoid hold time violation, as shown below

$$x_i - x_j \geq \lambda_{ij}. \tag{4.51}$$

This constraint is added into the optimization problems in Section 4.2.3 and Section 4.2.4 to incorporate hold time constraints to determine buffer values x_i and x_j.

4.3 Experimental Results

The proposed framework was implemented in C++ and tested using a 3.20 GHz CPU. We demonstrate the results with four circuits, s9234 to s38584, from the ISCAS89 benchmark set and four circuits, mem_ctrl to pci_bridge32, from the TAU13 variation-aware timing analysis contest. We set the maximum allowed buffer ranges to 1/8 of the original clock period and all tuning delays with 20 discrete steps [TRND+00]. The logic gates in the circuits were sized and mapped using a 45 nm library. The standard deviations of transistor length, oxide thickness and threshold voltage were set to 15.7%, 5.3% and 4.4% of the nominal values [Nas01]. We used Gurobi [Gur13] to solve the optimization problems in the proposed method.

4.3.1 Results of Post-Silicon Tunable Buffer Insertion at the Design Phase

The results with circuits from the ISCAS89 benchmark and the TAU13 benchmark set are demonstrated. The number of flip-flops and the number of logic gates in these circuits are shown in the columns n_s and n_g in Table 4.3. Three bins were used

Table 4.3: Results of buffer allocation for post-silicon binning

Circuit			Buffer		With Buffer Allocation						Runtime
	n_s	n_g	n_b	s_b	$\mathcal{Y}_{b1}$	$\mathcal{Y}_{b2}$	$\mathcal{Y}_{b3}$	$\mathcal{Y}_{\mu_T+\sigma_T}$	$\mathcal{Y}_{inc}(\%)$	$\mathcal{P}_{inc}(\%)$	$t_c(s)$
s9234	211	5597	2	18.00	52.31%	18.80%	13.72%	84.83%	0.70%	3.37%	20.83
s13207	638	7951	6	13.83	63.40%	13.55%	11.03%	87.98%	3.85%	18.47%	34.93
s15850	534	9772	5	7.20	67.93%	14.01%	10.16%	92.10%	7.97%	26.18%	56.81
s38584	1426	19253	14	12.52	63.79%	16.33%	10.71%	90.83%	6.70%	20.62%	71.03
mem_ctrl	1065	10327	10	13.06	58.41%	17.49%	12.93%	88.83%	4.70%	12.76%	164.62
usb_funct	1746	14381	17	14.71	54.61%	17.58%	14.03%	86.22%	2.09%	6.67%	147.88
ac97_ctrl	2199	9208	21	13.08	57.96%	16.45%	12.85%	87.26%	3.13%	11.39%	115.93
pci _bridge32	3321	12494	33	8.08	60.02%	16.84%	12.00%	88.86%	4.73%	14.87%	1816.81
			Average	12.56	59.80%	16.38%	12.18%	88.36%	4.23%	14.29%	
		Yield without buffers			50.00%	19.15%	14.98%	84.13%			

in the experiments to improve the overall profit. The boundaries between these bins were set to μ_T, $\mu_T + 0.5\sigma_T$ and $\mu_T + \sigma_T$, where μ_T and σ_T are the mean value and the standard deviation of the clock period of the original circuit without clock buffers. Chips with clock period larger than $\mu_T + \sigma_T$ were considered as yield loss. With this setting, the original yield values of these three bins without tuning buffers are 50%, 19.15%, and 14.98%, respectively. In all these test cases, the numbers of allocated buffers N_b were constrained as lower than 1% of the numbers of flip-flops in the circuits, as shown in the n_b column. After allocating post-silicon tuning buffers using the proposed method, Monte Carlo simulation are run with these circuits to verify the yield improvement. In the simulation, 10 000 samples were generated. For each sample its minimum clock period was calculated using an ILP solver due to the appearance of tuning buffers, and assigned the sample to one of the performance bins. The yield value of a circuit in a bin is the number of samples in that bin divided by 10 000. The samples in the experiments are conceptually different from the samples discussed in Section 4.1.3, because they were only used to emulate post-silicon measurements. For each sample, whether a chip can be assigned into a bin was verified by solving the classical skew scheduling problem in [Fis90]. In reality, the delays and timing properties cannot be measured exactly from the manufactured chips, so that the actual yield is slightly smaller than the reported yield, as discussed in [ZLS+18]. This yield, however, still serves as a good indicator to determine buffer locations.

The yield values of the three bins are shown in the columns $\mathcal{Y}_{b1}$, $\mathcal{Y}_{b2}$ and $\mathcal{Y}_{b3}$ in Table 4.3, respectively. Compared with the yield values without clock buffers, we can see that the yield in the first bin is increased significantly but the yield values of the other two bins are smaller, because with tuning buffers chips have a better chance to be tuned to a higher performance. Adding the yield values of the three bins together, we can calculate the yield of a circuit with respect to $\mu_T + \sigma_T$, shown in the $\mathcal{Y}_{\mu_T+\sigma_T}$ column. Compared with the original yield 84.13%, the yield increase is shown in the column $\mathcal{Y}_{inc}$, with an average 4.23%.

With these yield values in the three bins, we can calculate the profit using (4.4). In the experiments, we set the profit per chip of the three bins to 6, 2, and 1, respectively. The overall profit increase is shown in the column $\mathcal{P}_{inc}$, with an average 14.29%. If we compare the column $\mathcal{P}_{inc}$ and the column $\mathcal{Y}_{inc}$, we can see that the

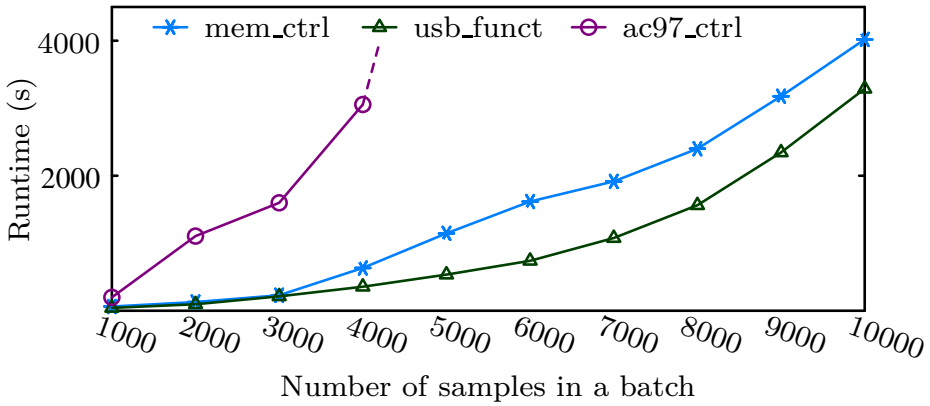

Figure 4.12: Scalability trend of the proposed method with a fixed number of samples in each batch.

improvement of profit is much more significant than the overall yield improvement due to the introduced tuning buffers and clock binning. To achieve this profit improvement, the number of buffers in the circuit is still less than 1% of the number of flip-flops. If we assume that a buffer takes 10 times area of a flip-flop and flip-flops take 5% of the die area, the area cost of these buffers is about 0.5% of the die area. Therefore, we can expect a good overall revenue improvement, even when we consider the potential cost of post-silicon configuration. A concrete evaluation of this cost will be our future work.

In the proposed method, we also reduced the buffer sizes by concentrating tuning values. The average buffer sizes in the benchmark circuits are shown in the column s_b. Compared with the maximum allowed size 20, the buffer sizes have been reduced effectively by the proposed method while maintaining a good profit improvement. The execution time of the proposed method is shown in the last column of Table 4.3. The largest execution time of the proposed method is 1816.81 seconds, which is already acceptable because the proposed method is executed offline only for a few times.

Since the runtime of solving an ILP problem depends on the structure of constraints as well as their relations, it is difficult to analyze the scalability of the proposed method theoretically. Instead, we tested this method by fixing the number of samples in each batch to solve the buffer insertion problem with respect to a given clock period $\mu_T + \sigma_T$ as used in Table 4.3. The relation between the number of samples in

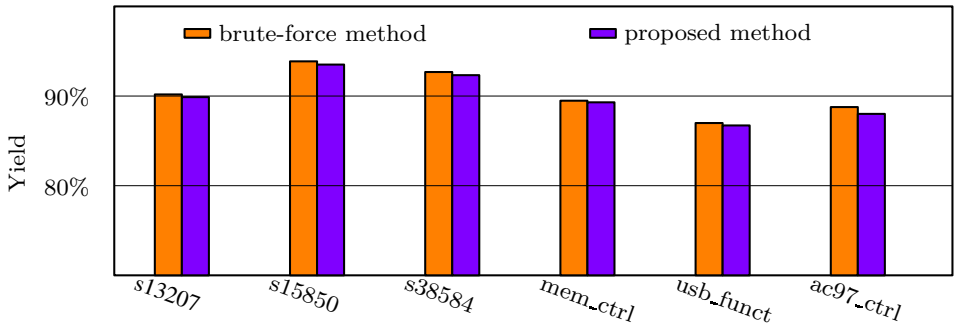

Figure 4.13: Yield comparison between the proposed method and the brute-force method with 10 000 samples.

a batch and the runtime is illustrated in Fig. 4.12. *pci_bridge32* did not finish due to memory limitation, so that it was not included in this evaluation. According to these results, the runtime increases exponentially with respect to the number of samples, especially with large circuits. In the proposed method, the number of samples in each batch is limited to N_t as discussed in Section 4.1.5. This limitation might lead to a yield degradation because the optimization problem is split into several small problems. To verify the quality of the results produced by the proposed method, we compared them with the yield results of a brute-force method processing 10 000 samples as a whole, as shown in Fig. 4.13. With this comparison, it can be observed that the yield degradation of the proposed method is negligible, because the proposed work flow in Fig. 4.3 first tries to capture all the buffer locations that have a potential to affect the yield. Afterwards, only these locations are considered in further iterations so that a batch can contain more samples, still leading to a good yield result. The runtime of the brute-force method, however, is much larger than the proposed method, as shown in Fig. 4.14.

In the profit definition (4.4), if we use only one bin, the problem formulation becomes the problem to improve the yield with respect to a single clock period. In our experiments, we tested this single-bin setting using μ_T, $\mu_T + 0.5\sigma_T$, and $\mu_T + \sigma_T$ as the upper bounds of the single bins, respectively. The results of yield improvement are shown in Fig. 4.15. In all these test cases, the yield values have been improved effectively, up to 18.19% for the circuit s15850 in the μ_T bin. In these test cases, the yield improvement is consistently better for bins with higher performance, because in these bins the original yield values without tuning buffers are lower so that there

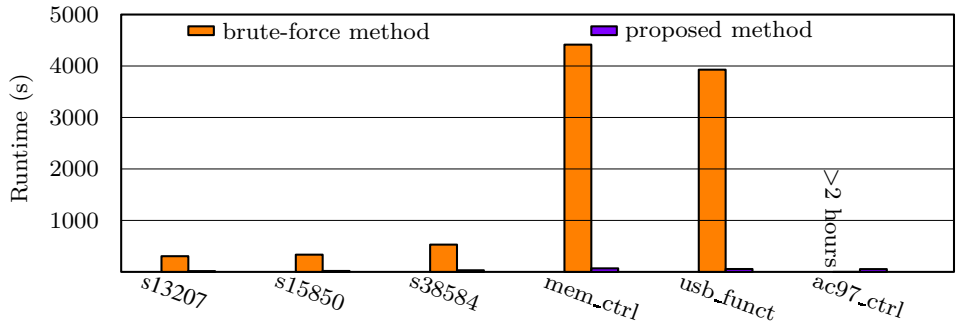

Figure 4.14: Runtime comparison between the proposed method and the brute-force method with 10 000 samples.

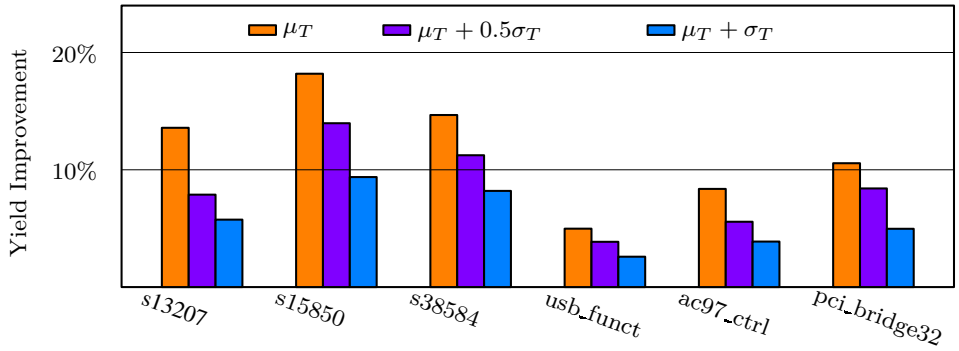

Figure 4.15: Yield improvement with clock tuning buffers with respect to μ_T, $\mu_T + 0.5\sigma_T$ and $\mu_T + \sigma_T$, compared with the yield values without tuning buffers.

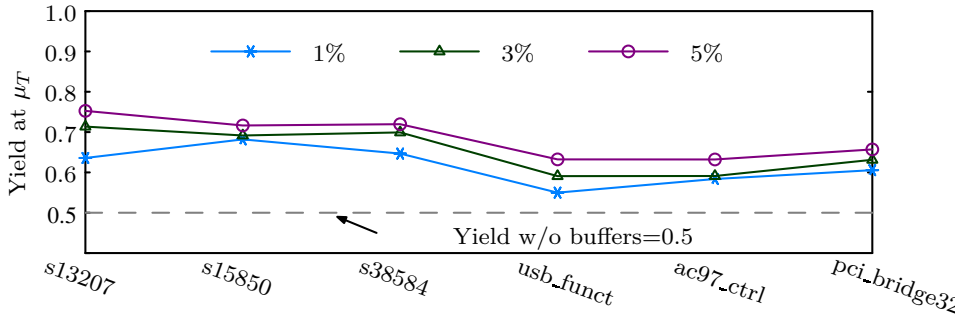

Figure 4.16: Yield increase with respect to different numbers of tuning buffers with the target clock period set to μ_T.

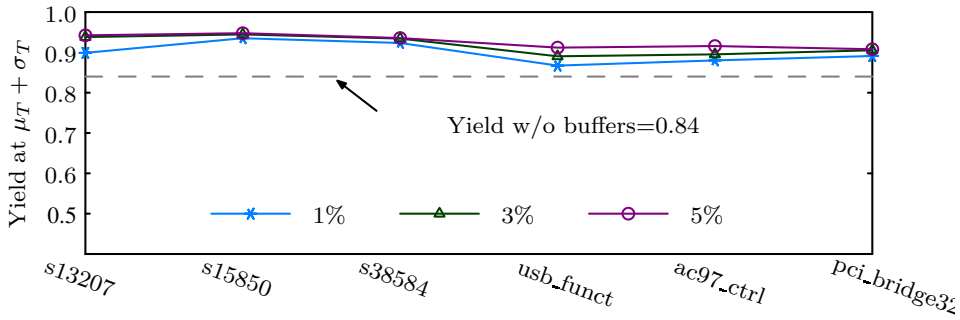

Figure 4.17: Yield increase with respect to different numbers of tuning buffers with the target clock period set to $\mu_T + \sigma_T$.

is a large potential for the tuning buffers to take effect.

In the experiments, we constrained the number of buffers to be smaller than 1% of the number of the flip-flops. If this number can be increased, we can expect an increase of yield because there are more chances to tune the chips after manufacturing. To show the effect of more tuning buffers, we tested the numbers of buffers equal to 1%, 3%, and 5% of the number of flip-flops. For each of these buffer numbers, we calculated the yield values with respect to the single clock periods μ_T and $\mu_T + \sigma_T$, respectively. The results are shown in Fig. 4.16 and Fig. 4.17. According to these experiments, we can see that the yield generally increases when the number of buffers inserted into the circuit increases. Similar to the trend of the yield improvement with respect to different clock periods in Fig. 4.15, the yield improvement with respect to μ_T in Fig. 4.16 is more obvious compared with the yield improvement with respect to $\mu_T + \sigma_T$ in Fig. 4.17. For the former, the average improvement of the 5% setting to the 1% setting is 6.78%, but for the latter this improvement is

Table 4.4: Runtime comparison w/o and w/ acceleration techniques

Circuit	s15850	s38584	ac97_ctrl	pci_bridge32
Without acceleration (s)	3411.29	8435.14	15967.7	$> 8h$
With acceleration (s)	56.81	71.03	115.9	1816.811

only 2.75%. Consequently, we can conclude that post-silicon buffers are more useful in high-performance designs, specially with clock binning, where the potential for profit/yield improvement is large.

To reduce the execution time of the proposed method, several acceleration techniques were introduced. With the Sobol sequence, the inner loop of the iterative flow in Fig. 4.3 converged with the test cases *usb_funct* and *pci_bridge32*, while the other cases used up all the samples. To demonstrate the efficiency of the acceleration techniques, we disable all of them and show the execution time in Table 4.4. According to this comparison, it is obvious that the proposed acceleration techniques can shorten the execution time effectively.

The buffer insertion problem is also addressed in [TZC05] with a direct statistical model. For comparison, we show the results from their paper and the results of our method applied to the same set of circuits in Table 4.5. The N_1 column shows the number of buffers in [TZC05], and the N_2 column that of our method. Note their method is designed for a clock network with a tree structure and they do not group buffers as we do. Consequently, there is a large difference between the numbers of buffers. The columns Y_1 and Y_2 show the yield values from their method and our method with the same clock period setting. In this comparison, the proposed method outperforms the method in [TZC05] with a higher yield, while the number of clock tuning buffers is much smaller. Furthermore, we have implemented the method in [LN14] and the yield comparison is shown in Fig. 4.18. In this comparison, the numbers of inserted buffers are equal, so that we can conclude that the proposed method outperforms the method in [LN14] consistently.

In the last step of the proposed method, we group buffers according to the correlation between tuning values. This correlation information is a natural result of the sampling-based method. In [LN14], a grouping algorithm is also proposed according to circuit structure and distances between flip-flops. We compare the results

Table 4.5: Yield comparison with [TZC05]

Circuit	N_1	Y_1	N_2	Y_2
s9234	8	96.94%	2	98.57%
s13207	18	98.95%	6	99.40%
s15850	21	99.24%	5	99.96%
s38584	162	98.17%	14	99.70%

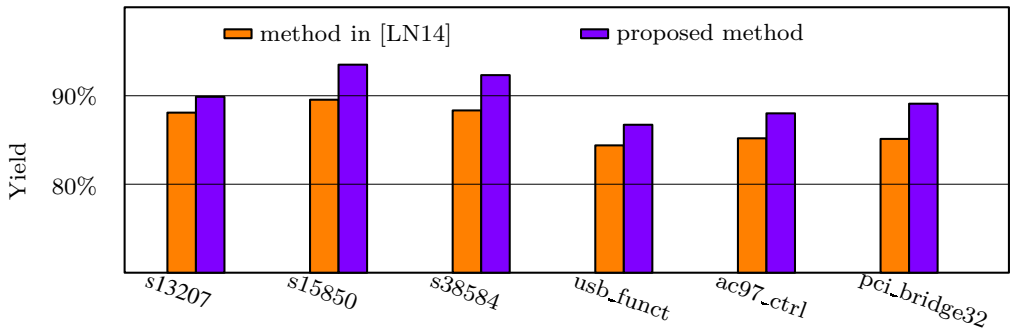

Figure 4.18: Yield comparison with the buffer insertion method in [LN14].

of our correlation-based grouping method with theirs and the results are shown in Table 4.6, where Y_1 is the yield with the grouping algorithm in [LN14] and Y_2 is the yield with the proposed correlation-based grouping. For comparison, we have changed the numbers of buffers in the proposed method so that they are equal to the ones in [LN14]. From this comparison, we can see that our method produces a better yield, because we have the correlation information from emulated samples.

The method proposed in [ZLS16a] uses the same concept in this dissertation, but it captures the locations of buffers by processing emulated samples once at a time. Therefore, the relation between tuning values in different samples is not incorpo-

Table 4.6: Yield comparison of different grouping algorithms

Circuit	s15850	s38584	ac97_ctrl	pci_bridge32
Y_1	84.57%	85.43%	84.67%	84.16%
Y_2	93.51%	92.33%	88.01%	89.10%

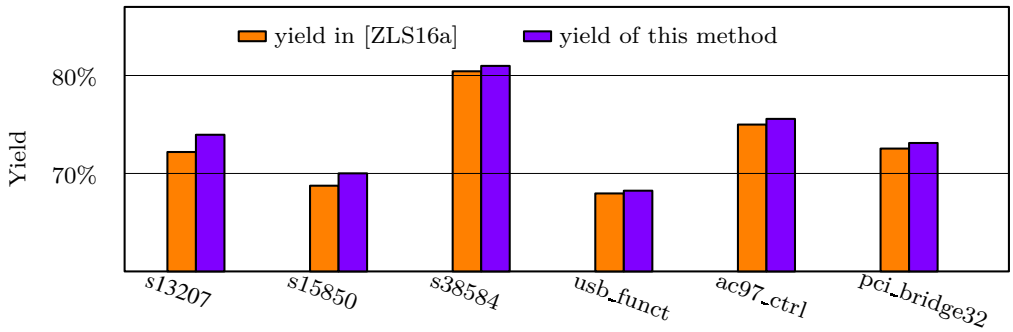

Figure 4.19: Yield improvement with the same setting compared with [ZLS16a].

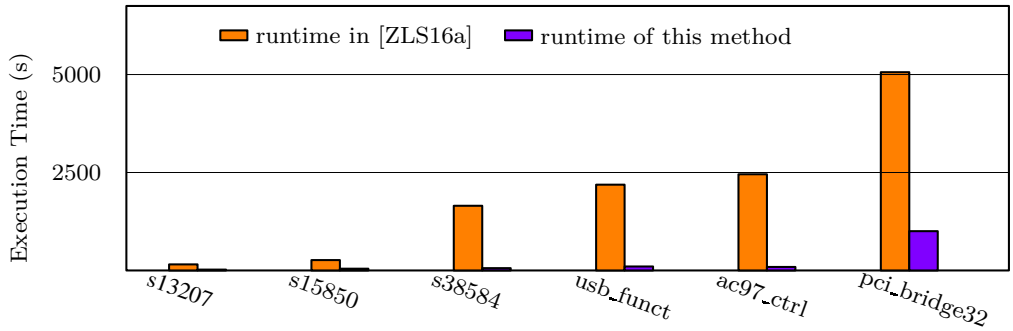

Figure 4.20: Execution time compared with [ZLS16a].

rated. In addition, the method in [ZLS16a] uses a purely random sequence so that the number of samples is still large. To verify the improvement of the proposed method, we mapped the circuits used in [ZLS16a] to the same library and tested the yield improvement with respect to μ_T. The results are shown in Fig. 4.19, where we can see that the proposed method produces a better yield improvement than [ZLS16a] with the same number of buffers. Furthermore, we show the execution time of these methods in Fig. 4.20. It is clear that the extended method in this dissertation is more efficient than [ZLS16a].

4.3.2 Results of Post-Silicon Tunable Buffer Configuration after Manufacturing

The results with circuits from the ISCAS89 benchmark and the TAU13 benchmark are demonstrated. The number of flip-flops and the number of logic gates in these circuits are shown in the columns n_s and n_g in Table 4.7. The column $|\mathbf{D}^m|$ shows the numbers of maximum delays between flip-flops whose delays need to be evaluated for post-silicon buffer configuration. If a flip-flop is attached a tunable buffer, the maximum delays from all its fanin flip-flops to it and from it to all its fanout flip-flops are added into $\mathbf{D}^m$ as discussed in Section 4.2.1. Consequently, these numbers may still be large, specially in the test cases mem_ctrl and pci_bridge32, despite only a small number of buffers (n_b) are inserted into the circuits. The column $|\mathbf{D}_t^m|$ shows the numbers of maximum delays $\mathbf{D}_t^m$ after applying SVD-QRcp decomposition in Algorithm 2. These maximum delays are identified from $\mathbf{D}^m$ and used to narrow the search scope of real combinational paths. Due to statistical prediction, the size of $\mathbf{D}_t^m$ is much smaller than that of $\mathbf{D}^m$, so that the number of paths that are really tested can be reduced effectively. The numbers of real combinational paths to be tested are shown in the column $|\mathbf{P}_t|$. These paths are identified by iterative selection in Algorithm 3. The tested delays of these paths are used to predicted $\mathbf{D}^m$. The efficiency of this delay prediction can be demonstrated by the comparison between $|\mathbf{P}_t|$ and $|\mathbf{D}^m|$ clearly, where the numbers of test paths are only about 2%–3% of the numbers of the maximum delays in most cases. The paths in $\mathbf{P}_t$ are grouped in test batches as described in Section 4.2.2, and the numbers of these batches are shown in the column $|\mathbf{B}|$. Since multiple combinational paths are multiplexed during delay

Table 4.7: Test Results With Delay Alignment and Statistical Prediction

Circuit										Frequency Stepping				Runtime		
	n_s	n_g	n_b	$\lvert\mathbf{D}^m\rvert$	$\lvert\mathbf{D}_t^m\rvert$	$\lvert\mathbf{P}_t\rvert$	$\lvert\mathbf{B}\rvert$	n_a	n_v	n'_a	n'_v	$r_a(\%)$	$r_v(\%)$	$T_p(s)$	$T_t(s)$	$T_s(s)$
s9234	211	5597	2	87	6	7	5	46.33	6.62	871.31	10.02	94.68	33.91	5.93	0.06	0.00
s13207	638	7951	6	456	12	12	2	51.25	4.27	4550.88	9.98	98.87	57.20	16.34	0.10	0.00
s15850	534	9772	5	546	10	11	4	47.05	4.28	5452.90	9.99	99.14	57.17	50.46	0.12	0.01
s38584	1426	19253	14	437	14	14	3	61.42	4.39	4366.07	9.99	98.59	56.09	89.94	0.15	0.03
mem_ctrl	1065	10327	10	2210	15	36	4	105.48	2.93	22038.12	9.97	99.52	70.62	299.63	0.69	0.06
usb_funct	1746	14381	17	1402	13	28	2	87.98	3.14	13975.14	9.97	99.37	68.48	143.13	0.36	0.04
ac97_ctrl	2199	9208	21	1714	13	20	2	58.78	2.94	16879.47	9.85	99.65	70.16	146.53	0.24	0.02
pci _bridge32	3321	12494	33	4501	22	58	6	181.60	3.13	44784.95	9.95	99.59	68.53	1712.57	0.92	0.79

test, these numbers can be much smaller than those of $\mathbf{P}_t$.

In the experiments, we tested 10 000 simulated chips by sampling statistical delays from statistical timing analysis. The column $\boldsymbol{n_a}$ shows the average number of frequency stepping iterations for each chip using EffiTest2 , and the column $\boldsymbol{n_v}$ shows the average number of iterations per path, where $n_v = n_a/|\mathbf{P}_t|$. For comparison, we implemented the method applying frequency stepping to each path individually, as assumed in [TBCS04, LN14, NK08, TGB09]. The column $\boldsymbol{n'_a}$ in Table 4.7 shows the average number of test iterations for each chip. These large numbers confirm that the straightforward frequency stepping method is impractical for large circuits. Furthermore, the column $\boldsymbol{n'_v}$ shows the average number of frequency stepping iterations per path where $n'_v = n'_a/|\mathbf{D}^m|$. The columns $\boldsymbol{r_a(\%)}$ and $\boldsymbol{r_v(\%)}$ show the reduction ratios of the test iterations per chip and the test iterations per path achieved using EffiTest2, where $r_a = (n'_a - n_a)/n'_a * 100$ and $r_v = (n'_v - n_v)/n'_v * 100$. Combining statistical prediction and aligned delay test, EfiiTest2 can reduce the overall test effort by more than 94% (94.68%∼99.65%). In addition, the ratios of test iterations per path $r_v(\%)$ demonstrate that test reduction can reach from 33.91% to 70.62%. This reduction comes only from test multiplexing and aligned delay test, while the statistical prediction technique does not affect this ratio. Both comparisons confirm that the proposed framework reduces test effort significantly.

The runtimes of the proposed method are shown in the last three columns in Table 4.7, where $\boldsymbol{T_p}$ is the runtime for path identification, batch assignment and hold time bound computation before delay test starts. Because these steps are performed offline, the runtime is already acceptable. The column $\boldsymbol{T_t(s)}$ shows the average runtime when computing the clock period T and the buffer configuration values for all test batches of a chip. Since this computation can be performed in parallel while path batches are tested, the runtime is also acceptable compared with the execution time of scan test. The last column $\boldsymbol{T_s(s)}$ shows the runtime to determine the final buffer values using the method in Section 4.2.4. This step is not performed on high-end testers so that the efficiency is good enough.

In the proposed framework, the results of aligned delay test produce lower and upper bounds for delays. This inaccuracy cannot be avoided due to the nature of delay test and it affects the yields of the circuits after buffer configuration. In addition, the technique of statistical prediction also introduces configuration inaccuracy in the

Table 4.8: Yield Comparison

Circuit	T_1			T_2		
	$y_i(\%)$	$y_t(\%)$	$y_r(\%)$	$y_i(\%)$	$y_t(\%)$	$y_r(\%)$
s9234	52.77	52.51	0.26	85.01	84.99	0.02
s13207	63.58	61.98	1.60	89.88	89.81	0.07
s15850	68.19	67.08	1.11	93.51	92.63	0.88
s38584	64.67	63.01	1.66	92.33	91.56	0.77
mem_ctrl	59.08	58.35	0.73	89.30	88.95	0.35
usb_funct	54.98	53.69	1.29	86.72	85.85	0.87
ac97_ctrl	58.37	57.84	0.53	88.01	87.81	0.20
pci _bridge32	60.56	58.87	1.69	89.10	88.08	1.02
Yield w/o tuning	50.00			84.13		

estimated delays. Consequently, it is expected that the yield values of the circuits should drop from the ideal yield values with delays assumed being measured exactly. We tested several cases with two clock periods T_1 and T_2 and the results are shown in Table 4.8. The yield in this table was calculated by checking the setup and hold time constraints between pairs of flip-flops for the 10000 simulated chips after buffer configuration. If the timing constraints were satisfied for a simulated chip that failed initially without buffer configuration, we assumed the chip after buffer configuration was rescued, so that the yield was improved by 0.01%. For T_1 and T_2 the original yield values without buffers were 50% and 84.13%, corresponding to the cases of setting the target clock period to mean and mean plus standard deviation of the clock period calculated by SSTA, respectively. The column y_i shows the yield values with a perfect delay measurement; the column y_t shows the yield values with delays measured by the proposed method; and the column y_r shows the yield drops due to the inaccuracy in the tested delays, where $y_r = y_i - y_t$. In these results, we can see that the yield drops are around 1-2%, where the improved yield values are still far better than those without buffers.

Since the results of the statistical prediction technique in Section 4.2.1 depend on the correlations between path delays, we manually increased the standard deviations of all delays by 10%. These increased variations are added to the purely random part of the delays so that the correlations between delays are decreased accordingly. Figure 4.21 shows the yield results of three cases with respect to the clock period T_2

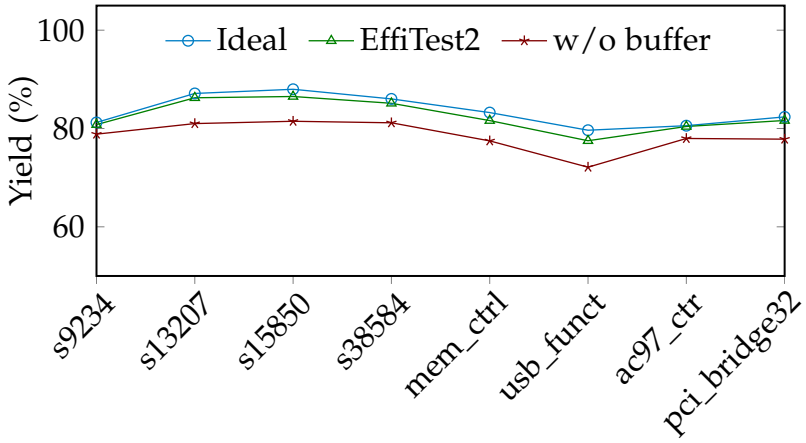

Figure 4.21: Yield with enlarged random variation.

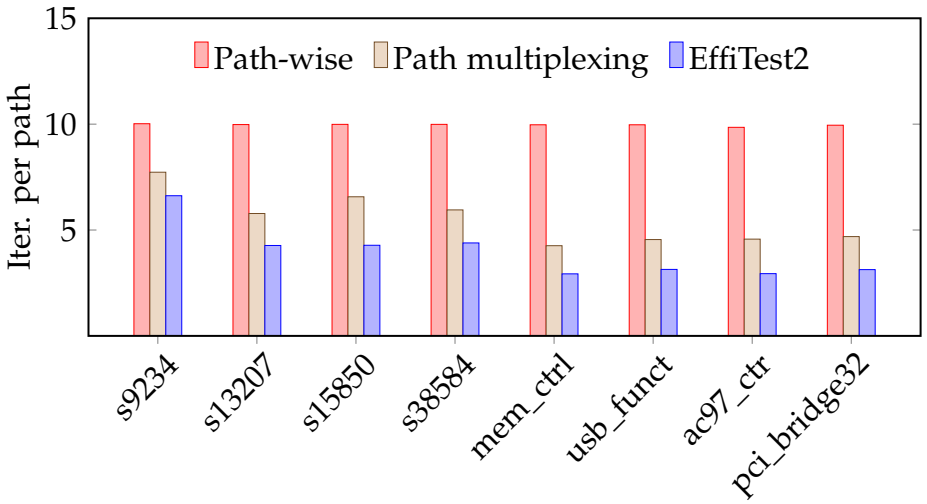

Figure 4.22: Test comparison without statistical prediction.

in Table 4.8: 1) ideal yield with buffers configured with presumed accurate delays; 2) with buffers configured using tested and predicted delays in ; 3) no buffers in the circuits. Compared with the results in Table 4.8, the yield values in Fig. 4.21 are lower due to the increased random variation. The first two cases, however, demonstrate clearly that the yield results were still improved impressively using tunable buffers when compared with the cases without them. When testing and configuring the buffer values with , the yield values dropped slightly from the ideal cases in Fig. 4.21, because of the expected inaccuracy in delay test and prediction. These yield values, however, still followed the ideal cases closely, confirming the strength of EffiTest2.

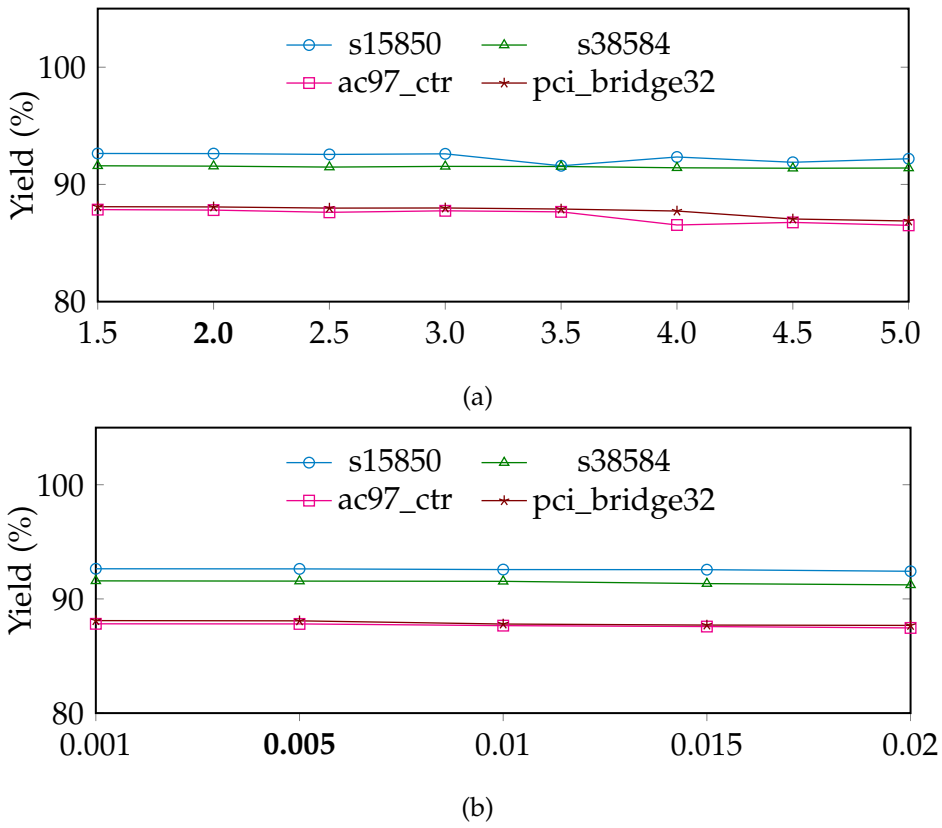

Figure 4.23: Prediction threshold and its effect on yield in Algorithm 2 and Algorithm 3 (a), and test threshold over yield in Algorithm 4 (b).

To verify the effectiveness of aligned delay ranges described in Section 4.2.2 and Section 4.2.3, we applied them directly to reduce test iterations without statistical prediction. Figure 4.22 shows the comparison of the numbers of test iterations per path in three cases: 1) path-wise frequency stepping, where around ten iterations were needed for each path; 2) test multiplexing without delay alignment using buffers; 3) multiplexing with delay alignment using buffers in EffiTest2. The second case used the method in Section 4.2.2 and Section 4.2.3, but all the buffers values were set to zero during test. Comparing the results of the first case and the second case, we can see that test multiplexing is a powerful technique to reduce test iterations. When the technique of delay alignment is applied, test iterations can be reduced further, as demonstrated by the third case. These results confirm that even without taking advantage of the correlations between path delays, the proposed method can still reduce test cost significantly.

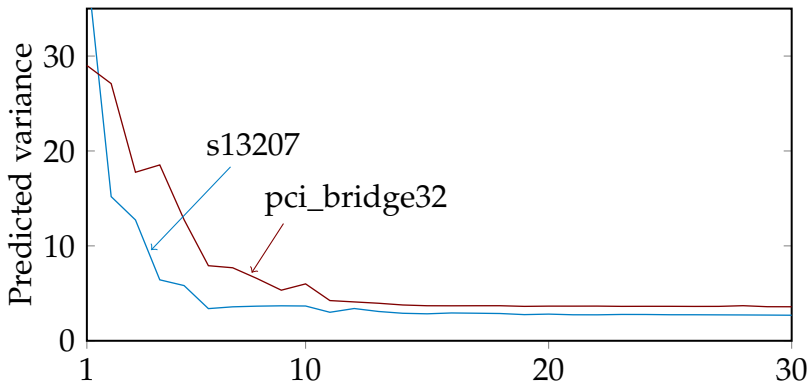

Figure 4.24: Effect of the number of selected variables over prediction accuracy.

In the statistic prediction technique described in Algorithm 2 and 3, the iterations stop when the predicted maximum variances reach a threshold σ_{th}. In delay prediction, the variance of a predicted variable cannot be lower than that of its purely random component. To experiment with different thresholds σ_{th} used in Algorithm 2 and 3, we first find the maximum of these purely random components of the predicted delays and set the threshold as constant times of them. Figure 4.23(a) shows the effect of these threshold values. As the threshold value reaches $3.5 \times \sigma_{max}$, the yield values of the circuits start to drop, because of the large range of the predicted delays. In EffiTest2, this constant was set to 2.0. Similarly, in the test procedure, the binary search of frequency stepping quits when an accuracy is reached. We have also tested different threshold values of ϵ in Algorithm 4 and the results are shown in Figure 4.23(b), where the x axis shows the constant times of the maximum of the mean values of the delays. As this number reaches 0.01, slight accuracy loss starts to appear. This number was set to 0.005 in EffiTest2 to maintain the test quality.

In Algorithm 2 and 3 we also increased the number of variables in $\mathbf{D}_t^m$ gradually in the loops, instead of using a binary search to reduce execution time. Figure 4.24 shows the trend of accuracy improvement with the two cases s13207 and pci_bridge32. For these two circuits, the accuracy does not improve notably after the number of variables becomes relatively large. Using the threshold setting discussed in Section 4.2.1, this number was set to 12 for s13207 and 22 for pci_bridge32 in the experiments. Furthermore, it can be observed that the curves for these two circuits do not decrease monotonously, so that a binary search may not return the best result. For example, 15 instead of 12 variables for s13207, and 24 instead of 22

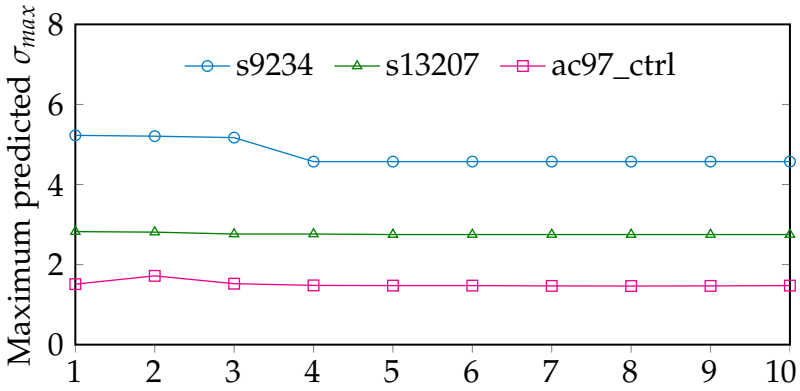

Figure 4.25: Comparisons of predicted standard deviations using different numbers of combinational paths. Lower values indicate better prediction accuracy.

variables for pci_bridge32, should be selected if a binary search would be used for these two cases.

To demonstrate the prediction accuracy using a given number of combinational paths for each maximum delay in $\mathbf{D}_t^m$ as discussed in Section 4.2.1, we compared the accuracy of all delays in $\mathbf{D}_m$ when the number of selected combinational paths is varied to from 1 to 10 in Algorithm 3. For a circuit, the threshold of the prediction accuracy σ_{th} for path selection in Algorithm 3 is set with respect to the standard deviations of purely random components of path delays described in Section 4.2.1. Accordingly, the predicted maximum standard deviations σ_{max} in $\mathbf{D}^m$ are different in the tested circuits. With more paths selected for testing, σ_{max} decreases due to the correlation information, however, with an increase of test cost. Fig. 4.25 shows the trend of maximum standard deviations σ_{max} of predicted values of $\mathbf{D}_m$ with respect to the number of selected paths. With the increase of the selected number, σ_{max} decreases, meaning the prediction accuracy is improved. When the number of selected paths is larger than 5, the accuracy does not change noticeably. Therefore, we set this number to 5 in to maintain the accuracy. The other circuits that are not included in Fig. 4.25 show less trend changes in the predicted standard deviations in terms of the number of combinational paths.

In EffiTest2, we do not test short paths using frequency stepping so that test iterations can be reduced. Instead, we set adjustment ranges as described in Section 4.2.5 to reduce hold time violations. These constraints are controlled by the threshold Y

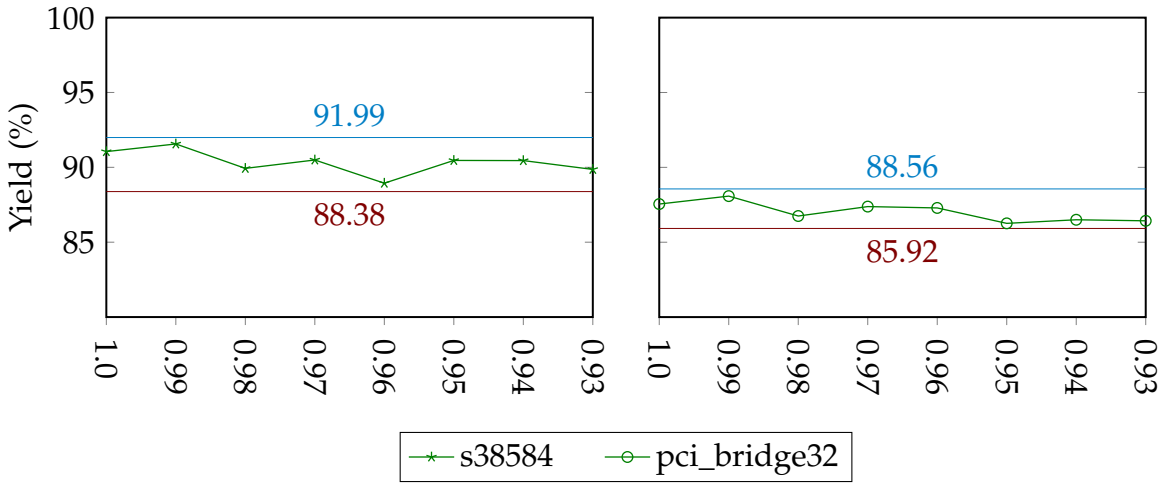

Figure 4.26: Hold time threshold effect over yield.

in (4.50), whose effect over the yield values of s38584 and pci_bridge32 are shown in Fig. 4.26. In each of these two figures, the two straight lines and the corresponding numbers show upper bound and lower bound of the yield values, where the former is computed by ignoring all hold time violations and the latter is computed by not adding the constraints in (4.50)–(4.51). When the threshold Y decreases to 0.98, the yield values of the circuits start to drop. Therefore, we set Y to 0.99 in EffiTest2.

4.4 Summary

In this chapter, a complete framework is proposed to consider the yield improvement with post-silicon tuning and the ensuing cost due to additional area and test holistically. Firstly, a sampling-based method is proposed to determine locations and ranges of post-silicon tuning buffers in a circuit to improve the overall profit with clock binning. By establishing the relation between buffer locations and the yield with an ILP model directly, the proposed method can learn the buffer locations for yield improvement effectively. With acceleration techniques such as a low discrepancy sequence, the proposed method takes much less time than previous methods. Experimental results confirm that the profit of the circuit after manufacturing can be improved significantly with a small number of buffers.

To reduce the test cost in configuring tunable buffers, an efficient framework is

proposed. By providing customized clock schemes to manufactured chips, timing failures may be alleviated by intentional clock skews with respect to the effect of process variations. The proposed framework combines statistical prediction and aligned delay test with path multiplexing to reduce test cost during post-silicon configuration. Consequently, the number of test iterations can be reduced by more than 94%, while the improved yield of the circuit is well maintained. The effectiveness of these techniques has been confirmed by experimental results using ISCAS89 and TAU13 benchmark circuits.

Chapter 5

A Holistic Timing Analysis Framework Considering Setup/Hold Time Interdependency

In static timing analysis, clock-to-q delays of flip-flops are considered as constants. Setup times and hold times are characterized separately and also used as constants. The characterized delays, setup times and hold times, are applied in timing analysis independently to verify the performance of circuits. However, clock-to-q delays of flip-flops depend on both setup and hold times in reality. Instead of being constants, these delays change with respect to different setup/hold time combinations. Consequently, the simple abstraction of setup/hold times and constant clock-to-q delays introduces inaccuracy in timing analysis. In this chapter, a holistic timing analysis framework considering setup/hold time interdependency is proposed.

To consider the flexibility of flip-flop delays, two problems have to be solved. 1) The three dimensional surface of the flip-flop in Fig. 3.5 should be modeled. In this step, only the necessary delay information that can be used to calculate the minimum clock period should be retained. Other delay information should be omitted to reduce simulation time; 2) A timing analysis algorithm using the piecewise delay model to calculate an accurate minimum clock period for a circuit. This algorithm should take the delay compensation across flip-flop stages into account and calculate the minimum clock period in a reasonable time.

In Section 5.1, firstly the three dimensional surface of the flip-flop in Fig. 3.5 is modeled using linear planes. The proposed algorithm to use the piecewise model to calculate an accurate minimum clock period is introduced in Section 5.2.

5.1 Adaptive Piecewise Polygonization of a Three-dimensional Delay Surface

To transform the delay surface in Fig. 3.5 into a form that can be used by static timing analysis, we partition it into small regions and approximate each one using a linear plane in the region, or a polygon. This piecewise approximation has the flexibility enabling the tradeoff between runtime and accuracy. The more polygons into which the surface is partitioned, the more accurate this approximation is, but the more time is required to generate these polygons and the slower the timing analysis algorithm becomes.

The linearization of a three dimensional surface is a problem studied widely and many methods have been proposed [Cla88, VHB87], using techniques such as adaptive sampling. But these methods are very general and do not take advantage of the special shape of the three dimensional delay surface as shown in Fig. 3.5. In this section, an adaptive method is proposed to approximate the delay surface using a set of polygons. This piecewise delay model can be used by the timing analysis algorithm in Section 5.2 directly.

The linearization of the delay surface includes three steps. First, we identify the boundary of the surface projected to the setup/hold slack dimensions using triangles. When the setup slack or the hold slack is very small, the flip-flop enters the metastable region, which should be excluded from the valid working space of the flip-flop. Second, we partition the delay surface inside the boundary with rectangles. We then check the approximation accuracy of each rectangle and split it further if the approximated delay is too far away from the real delay on the surface. Third, we merge the rectangles produced in the second step so that the number of polygons can be reduced, which improves the efficiency of timing analysis using this piecewise model.

5.1.1 Approximating the Surface Boundary Using Triangles

When the setup slack or the hold slack becomes too small, the flip-flop may enter the metastable region. The boundary between the working region of a flip-flop and

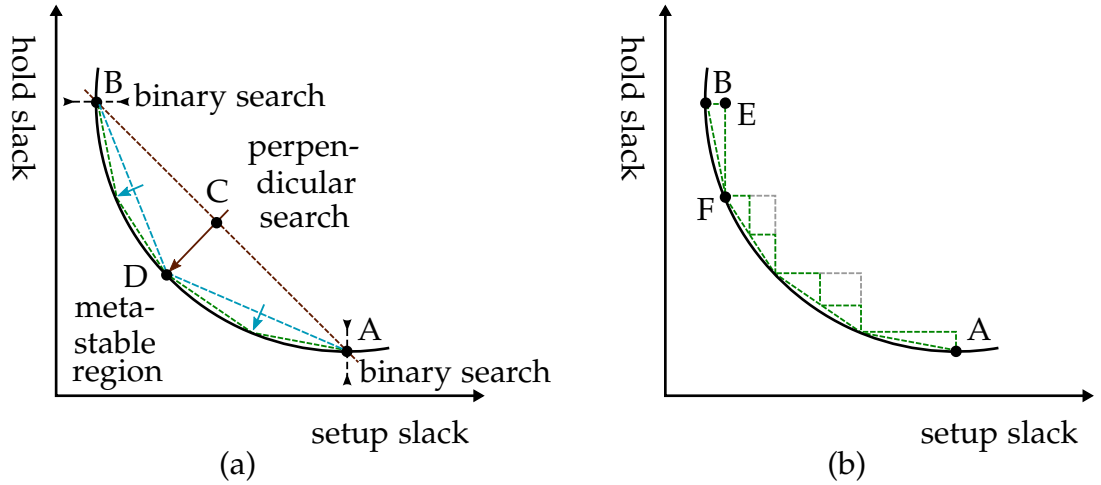

Figure 5.1: Approximation of the boundary of the delay surface. (a) Approximation with linear segments. (b) Delay surface approximation with triangles.

its metastable region has a shape similar to the curve in Fig. 5.1a when projected into the setup and hold dimensions.

In the proposed method, the boundary of the delay surface is approximated by a chain of linear segments whose ending points are on the boundary curve, as shown in Fig. 5.1a. As we do not know the shape of the curve, we generate these linear segments to approximate it. We start from a single linear segment shown as the right-most segment connecting points A and B where the setup slack and hold slack are set to large values, respectively.

In this setting, the setup slack at point A does not affect the delay of the flip-flop, so that a binary search of different hold time slacks can capture the delay value at A quickly. In each search, we run SPICE simulation at the target point. If the clock-to-q delay is larger than a given threshold or the SPICE simulator does not return a valid delay, this point is considered outside the feasible region of the clock-to-q delay. Otherwise, we find a valid point in the feasible region. After the binary search is finished, the valid point with the smallest hold slack is identified as point A in Fig. 5.1a. Similar to this process, we assign the flip-flop a very large hold time slack and execute a binary search along the setup slack to identify point B.

The linear segment connecting A and B (A↔B) still cannot be used as an approximation of the surface boundary, because the point at the middle of this segment may be far away from the real boundary. To check the distance between this ap-

proximation point (C) and the real boundary point (D), we apply a binary search in the direction perpendicular to the segment A↔B toward D using SPICE simulation. Afterwards, we compare the distance between C and D. If it is larger than a given threshold (k_{th}), we split the segment A↔B and create two linear segments A↔D and B↔D. In this way, the distance between the linear segments and the real boundary is reduced. This refining process is repeated further until the entire boundary curve is approximated within the accuracy requirement.

With the linear segments following the boundary closely, we then create triangles to approximate the three dimensional delay surface in the area close to this boundary. First we create triangles using these linear segments as their hypotenuse, as shown in Fig. 5.1b. Each of these triangles defines a valid region in which the delay surface is approximated with a linear plane defined by the corner points of the triangle. For example, in the triangle formed by the points B, E, and F, the clock-to-q delays at B and F are already known from SPICE simulation during constructing the linear segments. We then run SPICE simulation at point E. With the coordinates of the three corner points B, E, and F as well as the corresponding clock-to-q delays, the linear plane in the three dimensional space can be constructed easily.

To verify the accuracy of the triangular approximation in the three dimensional space, we select a point inside the triangle and run SPICE simulation again. We then compare the approximated delay by the linear plane and the simulated delay. Since the real delay surface is convex, the point that has an equal distance to the three ending points is chosen for the verification. In a right triangle, this point is at the middle of its hypotenuse.

In the comparison above, if the difference between the approximated value and the real delay value is larger than a threshold d_{th}, we split the triangle into two as illustrated in Fig. 5.1b. Consequently, we create more linear segments along the delay boundary to increase the approximation accuracy. Note in generating the linear segments along the boundary above, we check the distance of the approximate point to the boundary in the setup/hold slack dimensions using the threshold k_{th}. In verifying the approximation accuracy of the delay, we compare the delays directly so that the linear segments approximating the boundary might be split further as shown in Fig. 5.1b.

5.1.2 Approximating the Delay Surface Using Rectangular Polygons

The triangles found above only cover the areas of the delay surface close to the boundary. In order to approximate the whole surface, more polygons are required. Instead of using a lot of polygons directly, we take advantage of the fact that the delay surface is a constant surface when both the setup slack and the hold slack are large. In other words, if the signal switching at the input of a flip-flop is always far way from the active clock edge, the flip-flop always works in a stable region with a constant delay. To approximate this area, we need only one linear plane.

To identify the constant linear plane, we start a search from a stable point (G) at which the setup slack and hold slack are very large, in the direction in which setup and hold slacks decrease at the same time, as illustrated in Fig. 5.2a. At each step, we run SPICE simulation to find the real delay on the surface. Once we reach the first point at which the simulated delay increases, we use the last stable point (H) as the the corner of the stable linear plane.

The ending point H in Fig. 5.2a is hardly the optimal point from which the largest stable region is covered, because there are many such points at which the clock-to-q delay deviates from the stable value on the delay surface. However this non-optimal point does not affect the approximation accuracy or efficiency, because the uncovered region will be split and merged as follows.

To cover the areas between the triangles close to the boundary of the delay surface and the newly identified stable region, we create rectangles and split them with respect to the approximation accuracy. As illustrated in Fig. 5.2b, we start from the edges of the triangles and expand to the right and to the top of the area. Consequently, relatively large rectangles are created. Note there is some overlapping between these rectangles, e.g., the ones covering the corner point H. In the proposed ILP formulation, this overlapping is allowed and the solver chooses one of the points on the overlapping polygons as the working point of the flip-flop. In fact, a working point from any polygon works because they are all delay approximations meeting the specified accuracy.

The corners of a rectangle in Fig. 5.2b are located on the real delay surface directly. Therefore, the largest approximation error likely happens at the center of the rectangle. We run SPICE simulation at each center of the rectangle and compare the real

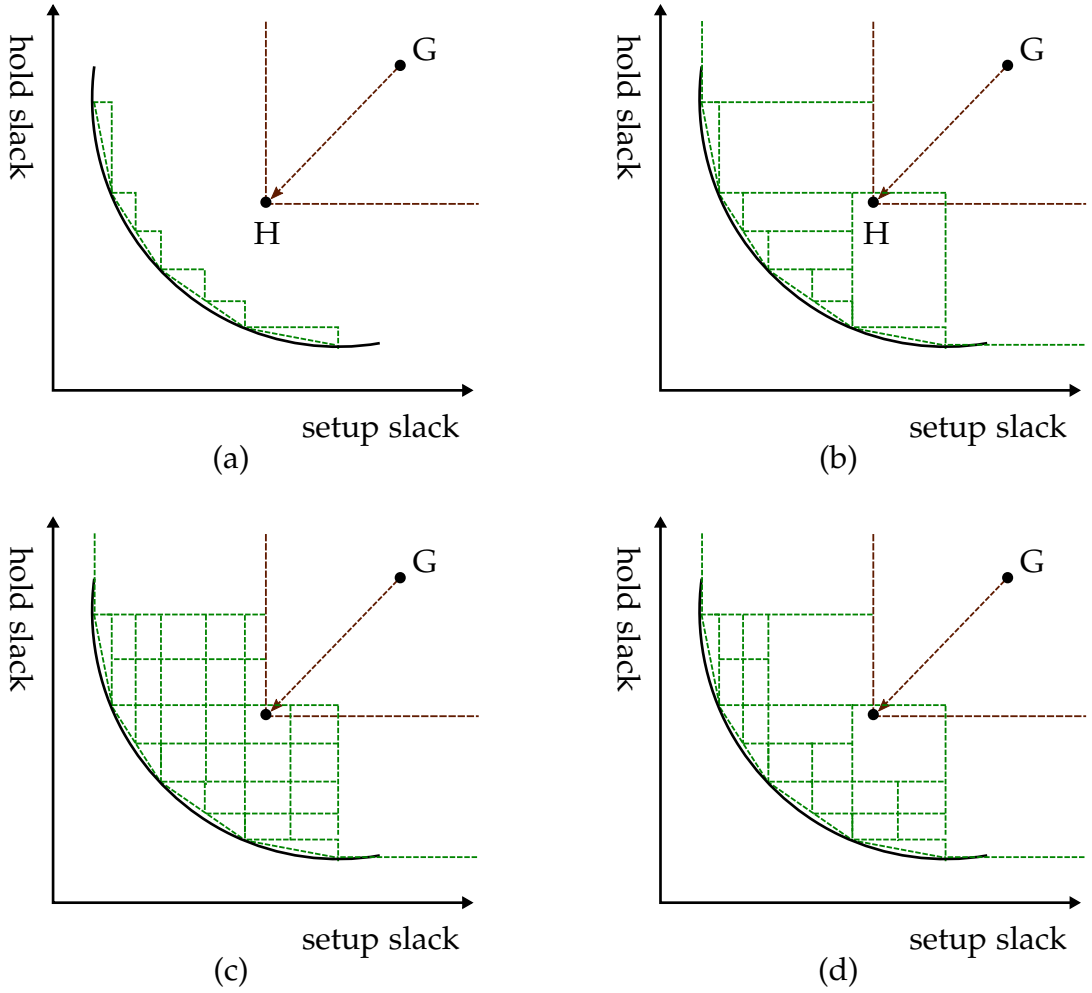

Figure 5.2: Rectangle construction, split and merge. (a) Identify the stable plane by search from G to H. (b) Construct rectangle between the triangles and the boundary of the stable region. (c) Split the rectangles to increase modeling accuracy. (d) Merge rectangles to reduce modeling complexity.

delay with the approximated delay on the plane. If the difference is larger than the threshold d_{th}, the rectangle is split further, as shown in Fig. 5.2c.

In forming the rectangles, we simply expand from the triangles. This simplification and the following split may produce more rectangles than necessary. Therefore, we try to merge neighboring rectangles in the last step, because a smaller number of rectangles means fewer constraints in the following ILP formulation. To merge rectangles, we search from each rectangle to the upper and right directions, because the delay in these directions changes relatively slowly. We combine each pair of neighboring rectangles into one rectangle, if the approximation value at the center of the new rectangle is in the range d_{th} from the real delay on the delay surface. The result of the rectangle merging is illustrated in Fig. 5.2d.

5.2 Piecewise ILP Model for Calculating the Minimum Clock Period

After the delay surface is approximated using polygons, we need to calculate the minimum clock period of a circuit. Compared with traditional STA, the challenge of using this piecewise model is to determine on which polygon a flip-flop works. In this section, a method based on ILP formulation to calculate the minimum clock period is proposed.

Assume in total there are n_p polygons approximating the delay surface. Since in timing analysis a flip-flop can only work with one setup/hold slack combination, only one of these polygons should be selected for the flip-flop. Therefore, we define a 0-1 variable z_i^k for the kth polygon in the piecewise delay model for flip-flop i. If the working point of the flip-flop falls into the kth polygon, $z_i^k = 1$; otherwise, $z_i^k = 0$. To allow the solver to choose one and only one polygon, we specify the following constraint

$$\sum_{k=1,\ldots,n_p} z_i^k = 1. \tag{5.1}$$

If the projection of the kth polygon to the setup/hold slack dimensions is a rectangle, as illustrated in Fig. 5.3a, the clock-to-q delay of flip-flop i in this region can be

expressed as

$$d^k_{cq,i} = f(s^k_i, h^k_i) = c^k \cdot z^k_i + c^k_s \cdot s^k_i + c^k_h \cdot h^k_i \tag{5.2}$$

$$s^k_l \cdot z^k_i \leq s^k_i \leq s^k_u \cdot z^k_i, \quad h^k_l \cdot z^k_i \leq h^k_i \leq h^k_u \cdot z^k_i \tag{5.3}$$

where s^k and h^k are the setup slack and the hold slack, respectively, and (5.2) defines the linear plane in the three dimensional delay space.

As described in Section 5.1, the real delays of the flip-flop at the corner points of the rectangle are known from SPICE simulation. Therefore, we can deduce a linear plane which passes the delay points corresponding to the corners of the rectangle. Such a plane can be characterized with only three points, and to be conservative we select three out of the four corner points with the largest delays to create the plane. Consequently, the constant coefficients c^k, c^k_s, c^k_h in (5.2) can be determined. Since this polygon is valid only in the rectangular region as illustrated in Fig. 5.3a, the lower and upper bounds of the setup slack are known from the characterization process as s^k_l and s^k_u. Similarly, the lower and upper bounds of the hold slack are known as h^k_l and h^k_u. In (5.3) the lower and upper bounds are all multiplied by z^k_i to enable or disable this polygon. If this polygon is selected so that $z^k_i = 1$, the constraints (5.2) and (5.3) describe a set of linear constraints. If this polygon is not selected with $z^k_i = 0$, the ranges of the setup slack and the hold slack are all forced to 0, so that both s^k_i and h^k_i are forced to 0. In this case, the delay $f(s^k_i, h^k_i)$ is also equal to 0, because the constant coefficient c^k is also multiplied by z^k_i.

For a triangular region, the corresponding polygon can be defined similar to (5.2) and (5.3). To prevent a setup/hold slack combination from falling into the area lower than the hypotenuse of the triangle, we add another constraint as

$$h^k_i \geq c^k_t + c^k_{t,s} \cdot s^k_i \tag{5.4}$$

where c^k_t and $c^k_{t,s}$ are characterized constants for the triangle. The concept of this additional constraint can be explained using the example in Fig. 5.3a. The newly added constraint (5.4) only allows the slack combination to fall into the region above the hypotenuse. Together with (5.3), this new constraint defines exactly the triangular region.

With the constraints (5.2)–(5.4) defined, the setup slack s_i, hold slack h_i and the

clock-to-q delay d_{cq}^{i} at flip-flop i can be written as

$$s_i = \sum_{k=1,\ldots,n_p} s_i^k, \quad h_i = \sum_{k=1,\ldots,n_p} h_i^k \tag{5.5}$$

$$d_{dq}^{i} = f(s_i, h_i) = \sum_{k=1,\ldots,n_p} f(s_i^k, h_i^k) \tag{5.6}$$

The constraints (5.5) and (5.6) are valid because the solver can only select one polygon constrained by (5.1). Consequently, the slacks and clock-to-q delays s_i^k, h_i^k and $d_{cq,i}^k$ can take nonzero values only in one region. Therefore, the sums in (5.5) and (5.6) are equal to the slacks and clock-to-q delay at the flip-flop.

In static timing analysis, timing constraints are defined with setup time, hold time and constant clock-to-q delays. For flip-flops i and j, the timing constraints can be written as

$$d_{cq,i} + \overline{d}_{ij} + t_{su,j} \leq T \tag{5.7}$$

$$d_{cq,i} + \underline{d}_{ij} \geq t_{h,j} \tag{5.8}$$

where $d_{cq,i}$ is the delay of flip-flop i, $\overline{d}_{ij}$ ($\underline{d}_{ij}$) is the maximum (minimum) delay of the combinational circuit between flip-flops i and j, $t_{su,j}$ ($t_{h,j}$) is the setup (hold) time of flip-flop j, and T is the clock period.

In setup constraints of pairs of flip-flops (5.7), setup time should be added to the path delay to verify the clock period. This setup time, however, is a virtual metric to guarantee that the clock-to-q delay is no larger than a given value, e.g., 110% of the stable delay. Since we incorporate the increase of clock-to-q delay in the delay model, this setup time is not needed in the constraint anymore. For the same reason, hold time is also removed from the constraint (5.8). Consequently, the timing constraints between flip-flops i and j can be written as

$$s_j \leq T - f(s_i, h_i) - \overline{d}_{ij} \tag{5.9}$$

$$h_j \leq f_(s_i, h_i) + \underline{d}_{ij}. \tag{5.10}$$

In the constraints (5.9) and (5.10), the feasible working region of the flip-flop is already included in the regions of the polygons approximating the delay surface.

With the constraints above, the ILP problem to calculate the minimum clock period

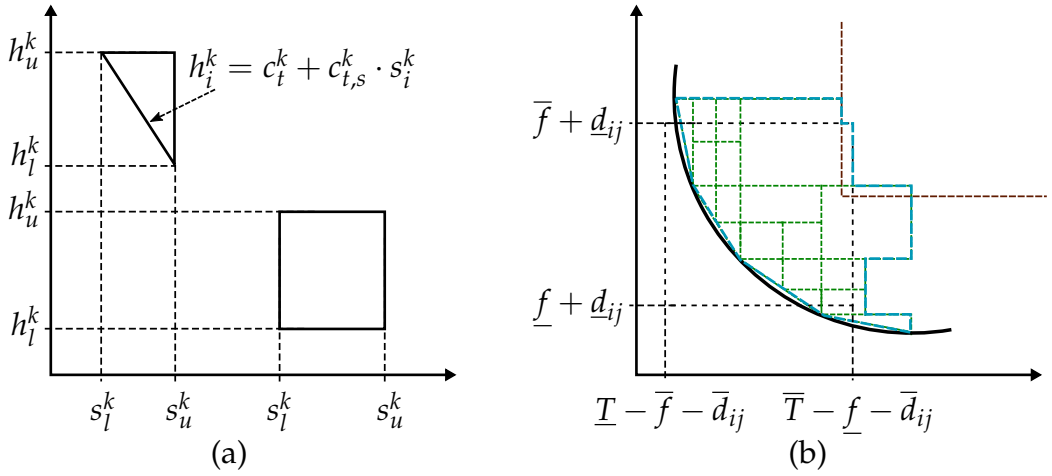

Figure 5.3: Slack ranges and polygon trimming. (a) Triangular and rectangular ranges of setup and hold slacks. (b) Trimming polygons using absolute upper and lower bounds of slacks.

can be expressed as

$$\text{minimize} \quad T \tag{5.11}$$

$$\text{subject to} \quad (5.1) - (5.10),\ \forall \text{ flip-flop pair} \tag{5.12}$$

Note that we relax the constraints for the variables s_j and h_j in (5.9) and (5.10) from equation to inequation to simplify the formulation. Since the values of s_j and h_j returned by the solver are always no larger than the real slacks defined by the right side of (5.9) and (5.10), the calculated minimum value of T is always a feasible solution.

Since the interdependency between clock-to-q, setup slack and hold slack allows timing compensation across flip-flop stages, the clock period can be lowered compared with the results of the traditional STA. However, this improvement incurs a large runtime in solving the ILP problem (5.11)–(5.12).

To reduce the computational complexity, we trim the polygons in the delay model of flip-flops. The basic idea is that we find absolute lower and upper bounds of the setup slack s_i and the hold slack h_i. The polygons completely falling outside the bounding box can be removed from the delay model.

In characterizing the flip-flop delay surface, we know that the delay $f(s_i, h_i)$ are bounded in a range $[\underline{f}, \overline{f}]$, where $\underline{f}$ is the stable delay of the flip-flop when the setup

slack and the hold slack are very large, and $\overline{f}$ is the delay beyond which we consider that the flip-flop enters metastability. For the clock period T, we also specify its range as $[\underline{T}, \overline{T}]$, where $\underline{T}$ ($\overline{T}$) is the lower (upper) bound of the clock period calculated by setting all setup slacks to the smallest (largest) values from delay characterization, and all clock-to-q delays to the lower (upper) bound $\underline{f}$ ($\overline{f}$). Consequently, we can specify the range of the setup slack of flip-flop j with a combinational path from flip-flop i as $[\underline{T} - \overline{f} - \overline{d}_{ij}, \overline{T} - \underline{f} - \overline{d}_{ij}]$. For the hold slack at flip-flop j, the range can be calculated similarly as $[\underline{f} + \underline{d}_{ij}, \overline{f} + \underline{d}_{ij}]$. Thereafter, we check all the polygons in the delay model and delete those that are completely outside the bounding box, as illustrated in Fig. 5.3b, where the thick dashed line shows the boundary of the possible region for the flip-flop.

After trimming polygons, we remove combinational paths and flip-flops that do not need to be included in the ILP formulation. If the source and sink flip-flops of a path always work in the stable region, this path is removed because it does not affect the minimum clock period. If all the paths connected to a flip-flop are removed, the flip-flop is also removed from the ILP formulation to reduce the computational complexity.

5.3 Experimental Results

The proposed framework was implemented in C++ and tested using a 2.67 GHz CPU. We demonstrate the results with circuits from the TAU13 benchmark set. These circuits were synthesized and optimized using a 45nm library and thereafter balanced by clock skews. The number of flip-flops and the number of logic gates in these circuits are shown in the columns n_s and n_g in Table 5.1, respectively. The interdependency of clock-to-q delay, setup slack and hold slack was characterized with HSPICE. The ILP solver for the optimization problem was Gurobi [Gur13].

In characterizing the three dimensional surface as discussed in Section 5.1, the piecewise model was constructed in 9 minutes and in total 401 points were simulated using HSPICE. The final piecewise model contains 64 polygons. We also simulated the delay surface within the region in which setup and hold slacks are smaller than 100ps with 1ps resolution. The total simulation time of this surface is 2.8 hours, confirming the efficiency of the proposed characterization method.

Table 5.1: Experimental Results of The Piecewise Linear Model and The ILP-based STA

Circuit			Trimming		Comparison						Runtime
	n_s	n_g	n_t	g_t	$t_s(\%)$	$t'_s(\%)$	v^s_p	v^s_f	v^h_p	v^h_f	$T(s)$
systemcdes	339	3617	160	53	1.30	1.27	875	67	167	10	2
wb_dma	550	3780	338	59	2.42	2.36	3031	255	177	11	4
aes_core	1015	26638	413	54	1.04	1.02	9351	246	201	13	11
tv80	1044	8499	433	54	1.12	1.10	6199	108	101	11	5
mem_ctrl	2043	9833	635	60	0.91	0.88	1087	89	152	25	24
usb_funct	2262	19234	763	45	0.99	0.94	6123	201	69	11	4
ac97_ctrl	2525	11482	1328	62	1.92	1.87	5580	799	178	12	9
pci _bridge32	3673	16918	1620	55	1.22	1.19	3918	187	191	10	17

The results applying the piecewise delay model and the ILP-based timing analysis are shown in Table 5.1. The number of flip-flops after trimming is shown in the column n_t. The number of polygons in the delay model was reduced from 64 to the average number shown in the column g_t. These results demonstrate that the trimming technique in our algorithm can reduce the number of flip-flops in the ILP model and the number of polygons effectively, resulting in a much smaller problem space.

To demonstrate the improvement in clock period considering the interdependency of clock-to-q delay, setup slack and hold slack, we compare the clock period calculated by our method with the result from traditional STA. The column t_s in Table 5.1 shows the relative clock period reduction from STA, in which the setup time is defined as the input slack when the clock-to-q delay is degraded to the 110% of the stable delay. The column t'_s shows the clock period reduction from the result of STA with the setup time defined with respect to the point at which the clock-to-q delay just starts to increase. This setting produces a smaller clock-to-q delay but a larger setup time. In both scenarios, our method achieved up to 2.42% and 2.36% of improvement in the clock period. This reduction of clock period exclusively results from the more accurate modeling and evaluation of the interdependency of clock-to-q delay, setup slack and hold slack, and no additional resource is required. Therefore, the proposed method is very useful in late stages of design flow, where

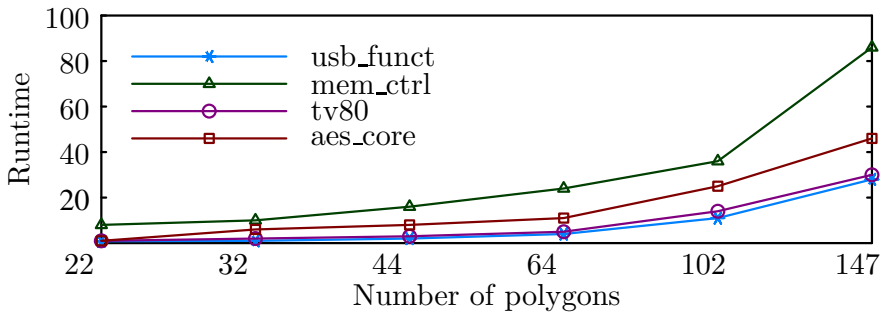

Figure 5.4: Runtime with different polygon numbers.

design iteration is normally not preferred.

In traditional STA, the three dimensional clock-to-q delay model is not used. Therefore, STA cannot recover from timing violations. Consider the scenario that the target clock period is the one calculated by the proposed method. With the target clock period given, the column v_p^s in Table 5.1 shows the number of paths with setup violation in STA, and the column v_f^s shows the number of flip-flops with timing violation. This comparison demonstrates that the proposed method has a significant advantage in removing timing violations. Furthermore, the numbers of paths and flip-flops with hold violation in STA are shown in the columns v_p^h and v_f^h, respectively. This comparison confirms again the advantage of the proposed method where there is no timing violation in all these cases.

The runtime of the proposed method is shown in the last column in Table 5.1. The largest runtime is 24 seconds for the circuit mem_ctrl with 2043 flip-flops. This runtime is larger than that of a block-based STA algorithm. However, considering that the application scenario of the proposed method is at late stages of design flow, this runtime is acceptable. In the proposed method, the number of polygons in the clock-to-q model affects the runtime of the ILP-based timing analysis. Figure 5.4 shows the runtime trend of the proposed method with respect to the number of polygons in the delay model. In all these cases, there is no noticeable difference in the calculated clock periods. Generally the runtime increases proportionally to the number of polygons.

To compare the proposed method with previous methods, we first calculate a minimum clock period using our method and use it as the target clock period. Thereafter, we identify the timing violations in the results from the methods in [KL14]

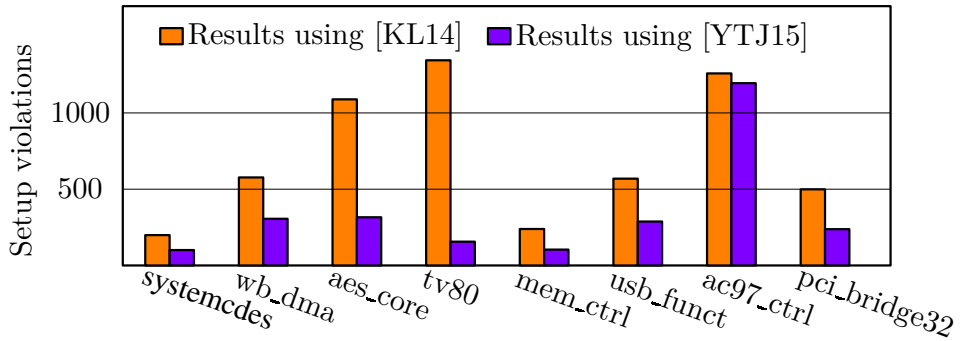

Figure 5.5: Comparisons of setup violations of flip-flops with [KL14] and [YTJ15]. The proposed method has no violation.

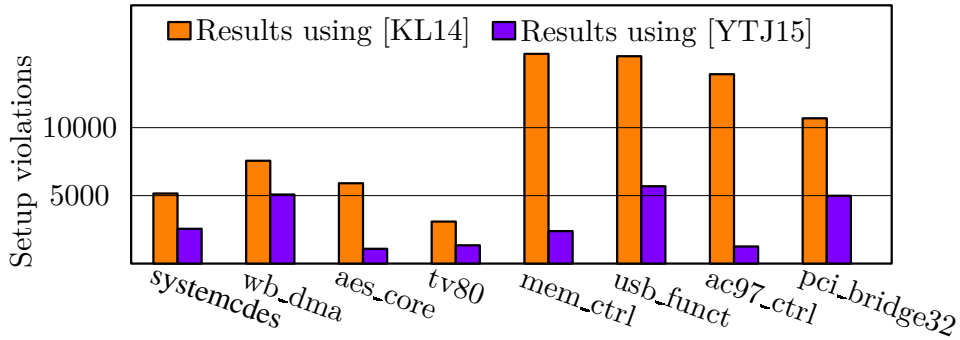

Figure 5.6: Comparisons of setup time violations of paths with [KL14] and [YTJ15]. The proposed method has no violation.

and [YTJ15]. Figure 5.5 and Fig. 5.6 show the comparisons of setup violations of flip-flops and paths, respectively. In [YTJ15], only the dependency between setup slack and clock-to-q delay is considered, and it does not exploit the interdependency of clock-to-q delay, setup slack and hold slack together. In the method [KL14], hold slack is fixed when maximizing the shared setup slack. Consequently, these limitations lead to many timing violations in their results. Besides setup violations, the number of hold violations from the method [YTJ15] are shown in Fig. 5.7. This comparison confirms again the effectiveness of the proposed method.

5.4 Summary

In this chapter, a holistic method is proposed to evaluate the timing performance of a circuit considering the interdependency of clock-to-q delay, setup slack and hold slack. Because this interdependency allows timing compensation across flip-

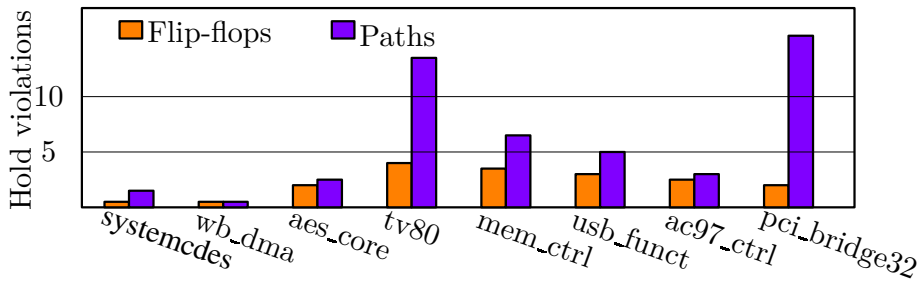

Figure 5.7: Hold time violations at flip-flops and on paths from [YTJ15]. The proposed method has no violation.

flop stages, the clock period of a circuit can be reduced. This is especially useful in late-stage designs where timing ECO is expensive. The proposed method models the clock-to-q delay surface using a piecewise model, reducing the modeling details by extracting only the necessary delay information useful to timing analysis. Thereafter, the minimum clock period of the circuit is evaluated using an ILP-based formulation, which for the first time provides a holistic solution considering the interdependency to improve circuit performance.

Chapter 6

Timing Optimization by Synchronizing Logic Waves with Delay Units

In digital circuit designs, sequential components such as flip-flops are used to synchronize signal propagations. Logic computations are aligned at and thus isolated by flip-flop stages. Although this fully synchronous style can reduce design efforts significantly, it may affect circuit performance negatively, because sequential components can only introduce delays into signal propagations instead of accelerating them. In this chapter, a new timing model, VirtualSync, is proposed in which signals, specially those along critical paths, are allowed to propagate through several sequential stages without flip-flops. Timing constraints are still satisfied at the boundary of the optimized circuit to maintain a consistent interface with existing designs.

To establish VirtualSync, sequential components and combinational logic gates are considered as delay units. Combinational logic gates add linear delays of the same amount to short and long paths. However, sequential components have non-linear delay effects, providing different delay effects to fast and slow signal propagations.

With the new timing model, sequential components are allocated only at necessary locations in the circuit to synchronize signal propagations, while the functionality of circuits is maintained. The absence of flip-flops at some sequential stages allows a virtual synchronization to provide identical functionality as in the original circuit. Consequently, the original clock-to-q delays and setup requirements along the critical paths can be removed to achieve a better circuit performance even beyond the limit of traditional sequential design.

To incorporate the above concepts into the VirtualSync framework, all flip-flops are removed and then the necessary locations are identified to block fast signals

using combinational gates and sequential components, e.g., buffers, flip-flops, and latches. The advantage of this formulation is that it is possible to insert the minimum number of delay units into the circuit to achieve the theoretical minimum clock period.

The problem formulation of VirtualSync is described as follows:

Given: the netlist of a digital circuit; the delay information of the circuit; the target clock period T.

Output: a circuit with adjusted number and locations of sequential components; logic gates with new sizes; inserted delay units, e.g., buffers.

Objectives: the circuit should maintain the same function viewed from the sequential components at the boundary of the optimized circuit; the target timing specification should be met; the area of the optimized circuit should be reduced.

6.1 The New Timing Model

6.1.1 Delay Units

In the VirtualSync framework, we first remove all sequential components, flip-flops, from the circuit under optimization. Consequently, logic synchronization may be lost because signals across fast paths may arrive at flip-flops in incorrect clock cycles, e.g., earlier than specified, or timing violations may be incurred. In addition, signals along combinational loops should also be blocked to avoid the loss of logic synchronization.

To slow down a signal, three different components can be used as delay units, namely, combinational gates such as buffers, flip-flops, and latches, which exhibit different delay characteristics, as shown in Fig. 6.1, where the term ***input gap*** refers to the difference of arrival times of two signals at a delay unit, and the term ***output gap*** represents the difference between their arrival times after they pass through the unit.

In Fig. 6.1(a), a combinational delay unit adds the same amount of delay to any input signal. Consequently, the arrival time s_v at the output of the combinational

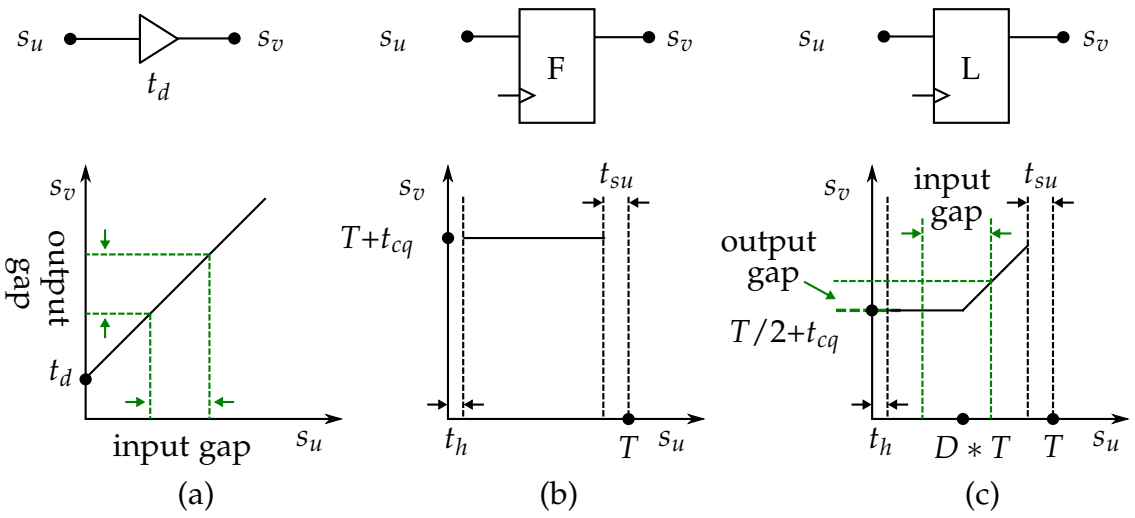

Figure 6.1: Properties of delay units. (a) Linear delaying effect of a combinational delay unit. (b) Constant delaying effect of a flip-flop. (c) Piecewise delaying effect of a latch.

delay unit is linear to the arrival time s_u at the input of the delay unit. Therefore, the absolute gap between arrival times of signals through short and long paths does not change when a combinational delay unit is passed through.

In delaying input signals, a flip-flop, as a sequential delay unit, behaves completely differently from a combinational delay unit, as shown in Fig. 6.1(b). If the arrival time of a signal falls into the time window $[t_h, T - t_{su}]$, where t_h is the hold time and t_{su} is the setup time, the output signal always leaves at the time $T + t_{cq}$, with t_{cq} as the clock-to-q delay of the flip-flop. Therefore, the gap between the arrival times of two signals reaching at the input of a flip-flop is always reduced to zero at the output of the flip-flop. This property is very useful when the delays of short paths and long paths in a circuit differ significantly after all sequential components are removed from the circuit under optimization. For many short paths, it is not possible to pad their delays by adding combinational delay units such as buffers to them, because the combinational delay units on the short paths may also appear on other long paths. The increased delays along long paths might affect circuit performance negatively. Flip-flops thus can be used in this scenario, because short paths receive more delay padding than long paths to align logic waves in the circuit.

As the second type of sequential delay units, level-sensitive latches, have a delay property combining those of combinational delay units and flip-flops, as shown in

Fig. 6.1(c), where $0 < D < 1$ is the duty cycle of the clock signal. Assume that a latch is non-transparent in the first part of clock period and transparent in the second part of the clock period. If two input signals arrive at a latch when it is non-transparent, the output gap is reduced to zero. If both signals arrive at a latch when it is transparent, the gap remains unchanged. However, if the fast signal reaches the latch when it is non-transparent while the slow signal reaches it when it is transparent, the output gap of the two signals is neither zero nor unchanged. Instead, it takes a value between the two extreme cases as illustrated in Fig. 6.1(c). This property allows us to modulate signals with different arrival times more flexibly, specifically those along critical paths where fast signals require more delay padding and slow signals should be not affected.

6.1.2 Relative Timing References

For a circuit under optimization, we consider the flip-flops which locate at the boundary of the circuit as boundary flip-flops. No matter how signals inside the circuit propagate, the function of the whole circuit is still maintained if we can guarantee that for any input pattern at the input boundary flip-flops in the circuit produces the same result at the output boundary flip-flops at the same clock cycle as the original circuit.

Consider a general case in Fig. 6.2, where F1 and F4 are the boundary flip-flops and F2 and F3 are removed in the initial circuit for optimization. At F4, the arrival times are required to meet the setup and hold time constraints, written as

$$s_z + t_{su} \leq T \tag{6.1}$$

$$s'_z \geq t_h \tag{6.2}$$

where s_z and s'_z are the latest and earliest arrival times at z. These two constraints in fact are defined with respect to the rising clock edge at F3, since the clock period T in (6.1) shows that the signal should arrive at F4 within one clock period. Although F2 and F3 are removed from the circuit, the constraints at F4 should still be the same as (6.1)-(6.2) to maintain the compatibility of the timing interface at the boundary flip-flops.

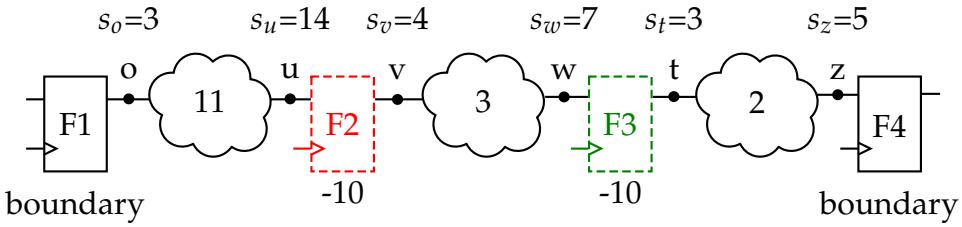

Figure 6.2: Concept of relative timing references. Clock period T=10. Clock-to-q delay t_{cq}=3. Both setup time t_{su} and hold time t_h are equal to 1. F3 is kept in the optimized circuit and F2 is not included.

In the general case in Fig. 6.2, we can also observe that the timing constraint at F3 in the original circuit is also defined with respect to the rising clock edge at F2. This definition can be chained further back until the source flip-flop F1 at the boundary is be reached. We call the locations of these removed flip-flops such as F2 and F3 ***anchor points***. After all sequential components are removed from the circuit under optimization, these anchor points still allow to relate timing information to boundary flip-flops. Every time when a signal passes an anchor point, its arrival time is converted by subtracting T in VirtualSync. When a signal finally arrives at a boundary flip-flop along a combinational path, its arrival time must be converted so many times as the number of flip-flops on the path, so that (6.1)-(6.2) is still valid.

In Fig. 6.2, assume that F2 is removed but F3 is inserted back in the optimized circuit. The arrival time s_u is subtracted by the clock period T=10 to convert it with respect to the time at F1, leading to s_v=4. The arrival time s_w is defined with respect to the previous flip-flop before F3, so that the timing constraints can be checked using (6.1)-(6.2). Since the arrival time before F4 should meet its timing constraints, F3 thus cannot be removed. Otherwise, the arrival time s_t would be equal to 7-10=-3. Accordingly, the arrival time s_z becomes -3+2=-1, definitely violating the hold time constraint in (6.2).

Since F3 is kept in the optimized circuit, it introduces the delay with the property shown in Fig. 6.1(b). The arrival time after this sequential delay unit thus becomes $T + t_{cq}$=13. This signal at t in Fig. 6.2 also passes an anchor point. Therefore, the arrival time s_t is equal to 3, leading to no timing violation at F4. This example demonstrates that the timing constraints at the boundary flip-flops force the usage of the internal sequential delay units. The model to insert these delay units automatically will be explained in the next section.

6.1.3 Synchronizing Logic Waves by Delay Units

With all flip-flops removed from the circuit under optimization, we only need to delay signals that are so fast that they reach boundary flip-flops too early; signals that propagate slowly are already on the critical paths, thus requiring no additional delay. Since it is not straightforward to determine the locations for inserting additional delays, we formulate this task as an ILP problem and solve it later with introduced heuristic steps. The values of variables in the following sections are determined by the solver, unless they are declared as constants explicitly.

The scenario of delay insertion at a circuit node, i.e., a logic gate, is illustrated in Fig. 6.3, where a combinational delay unit ξ_{uv} may be inserted, the original delay of the logic gate may be sized, and a sequential delay unit may be inserted to block fast and slow signals with different delays. Furthermore, the number of flip-flops between w and t in the original circuit is represented by an integer constant λ_{tz}. When $\lambda_{tz} \geq 1$, an anchor point is found at the location. λ_{tz} is used to convert arrival times.

In Fig. 6.3, the delay at the circuit node can be changed by sizing the delay of the logic gate, e.g., the XOR gate in Fig. 6.3. For the case that the required gate delay exceeds the largest permissible value, a combinational delay unit is inserted at the corresponding input. For convenience, we assume the combinational delay unit inserted at the input is implemented with buffers. The relation between the arrival times u and w is thus expressed as

$$s_w \geq s_u + \xi_{uv} * r^u + d_{vw} * r^u \tag{6.3}$$

$$s'_w \leq s'_u + \xi_{uv} * r^l + d_{vw} * r^l \tag{6.4}$$

where s_u, s'_u, s_w and s'_w are the latest and earliest arrival times of node u and w , respectively. ξ_{uv} is the extra delay introduced by an inserted buffer and d_{vw} is the pin-to-pin delay of the logic gate. If ξ_{uv} is reduced to 0 after optimization, no buffer is required in the optimized circuit. The $\leq$ and $\geq$ relaxations of the relation between arrival times guarantee that only the latest and the earliest arrival times from multiple inputs are propagated further. r^u and r^l are two constants to reserve a guard band for process variations, so that $r^u > 1$ and $r^l < 1$.

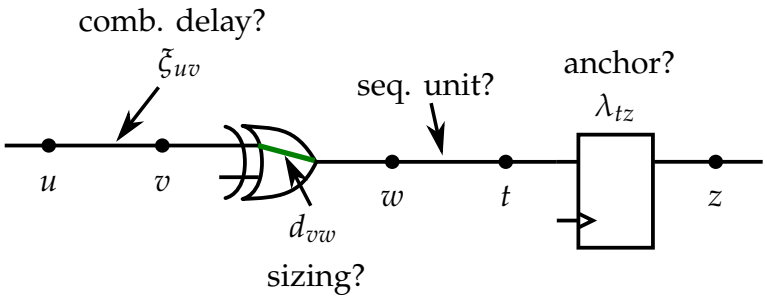

Figure 6.3: Delay insertion model in VirtualSync.

Since arrival times through long and short paths reaching w may have a large difference, we may need to insert sequential delay units to delay the fast signal more than the slow signal. This can be implemented with the sequential units shown in Fig. 6.1, where the gap between the arrival times is reduced after passing a sequential delay unit, either a flip-flop or a latch. To insert a sequential delay unit, three cases need to be examined.

Case 1: No sequential delay unit is inserted between w and t in Fig. 6.3, so that

$$s_t \geq s_w \tag{6.5}$$

$$s'_t \leq s'_w. \tag{6.6}$$

Case2: A flip-flop is inserted between w and t. Assume the flip-flop works at a rising clock edge. As shown in Fig. 6.1(b), a flip-flop only works properly in a region t_h after the rising clock edge and t_{su} before the next rising clock edge. Therefore, we need to bound the arrival times s_w and s'_w into such a region by

$$s_w, s'_w \geq N_{wt} * T + \phi_{wt} + t_h * r^u \tag{6.7}$$

$$s_w, s'_w \leq (N_{wt} + 1) * T + \phi_{wt} - t_{su} * r^u \tag{6.8}$$

where N_{wt} is an integer variable determined by the solver. T is the given clock period. ϕ_{wt} is phase shift of the clock signal. The available values of ϕ_{wt} can be set by designers. If only one clock signal is available, ϕ_{wt} can be set to 0 and T/2 to emulate flip-flops working at rising and falling clock edges.

When the input arrival times fall into the valid region of a flip-flop as constrained by (6.7)–(6.8), the signal always starts to propagate from the next active clock edge,

so that constraints can be written as

$$s_t \geq (N_{wt} + 1)T + \phi_{wt} + t_{cq} * r^u \tag{6.9}$$

$$s'_t \leq (N_{wt} + 1)T + \phi_{wt} + t_{cq} * r^l. \tag{6.10}$$

Case3: A level-sensitive latch is inserted between w and t. To be consistent with the active region of flip-flops, we assume that the latches are transparent when the clock signal is equal to 0. We can then bound the arrival times at w the same as (6.7)–(6.8).

As illustrated in Fig. 6.1(c), the latch is non-transparent in the first part of the region and transparent in the second region. Accordingly, the latest time a signal leaves the latch can be expressed as

$$s_t \geq N_{wt} * T + \phi_{wt} + D * T + t_{cq} * r^u \tag{6.11}$$

$$s_t \geq s_w + t_{dq} * r^u \tag{6.12}$$

where (6.11) corresponds to the case that the latch is non-transparent, so that the signal leaves the latch at the moment the clock switches to 1. D is the duty cycle of the clock signal with $0 < D < 1$. (6.12) corresponds to the case that the latch is transparent, so that only the delay of the latch is added to s_w. t_{dq} is the data-to-q delay of the latch.

The earliest time a signal leaves the latch is, however, imposed by a constraint in the less-than-max form as in [SMO90a],

$$s'_t \leq \max\{N_{wt} * T + \phi_{wt} + D * T + t_{cq} * r^l,\ \ s'_w + t_{dq} * r^l\} \tag{6.13}$$

which cannot be linearized easily. In the VirtualSync framework, the purpose of introducing the sequential delay unit is to delay the short path as much as possible. This effect happens when a signal arrives at a non-transparent latch. Therefore, we impose the arrival times of fast signals to be positioned in the non-transparent region, expressed as

$$N_{wt} * T + \phi_{wt} + t_h * r^u \leq s'_w \leq N_{wt} * T + \phi_{wt} + D * T \tag{6.14}$$

while relaxing (6.13) as

$$s'_t \leq N_{wt} * T + \phi_{wt} + D * T + t_{cq} * r^l. \tag{6.15}$$

When inserting the sequential delay unit, each of the three cases above can happen in the optimized circuit. We use an integer variable to represent the selection and let the solver determine which case happens during the optimization.

The arrival times in the model need to be converted each time when an anchor point is passed. The constant λ_{tz} represents the number of flip-flops at such a point in the original circuit. In Fig. 6.3, the arrival time at z is shifted as

$$s_z = s_t - \lambda_{tz} T. \tag{6.16}$$

Since we allow multiple waves to propagate along a combinational path, we need to guarantee that the signal of the next wave starting from a boundary flip-flop never catches the signal of the previous wave starting from the same flip-flop [BCKL98]. This constraint should be imposed to every node in the circuit. For example, the constraint for node u is written as

$$s_u + t_{stable} \leq s'_u + T \tag{6.17}$$

where t_{stable} is the minimum gap between two consecutive signals.

The introduction of the relative timing references, or the anchor points, in Section 6.1.2 guarantees that the number of clock cycles along any path does not change after optimization. With the timing constraints (6.1)-(6.2) at boundary flip-flops, the correct function of the optimized circuit is always maintained, without requiring any change in other function blocks.

The constraints (6.1)–(6.17) excluding (6.13) needs to be established at each node in the circuit after flip-flops are removed. The appearance of the combinational and sequential delay units needs to be determined by the solver. The delays of logic gates should also be sized. The objective of the optimization is to find a solution to make the circuit work at a given clock period T, while reducing the area cost. Taken all these factors into account, the straightforward ILP formulation may become insolvable. In practice, however, this technique only needs to be applied to isolated circuit parts containing critical paths. In addition, heuristic techniques are introduced to overcome this scalability problem, as explained in the following section.

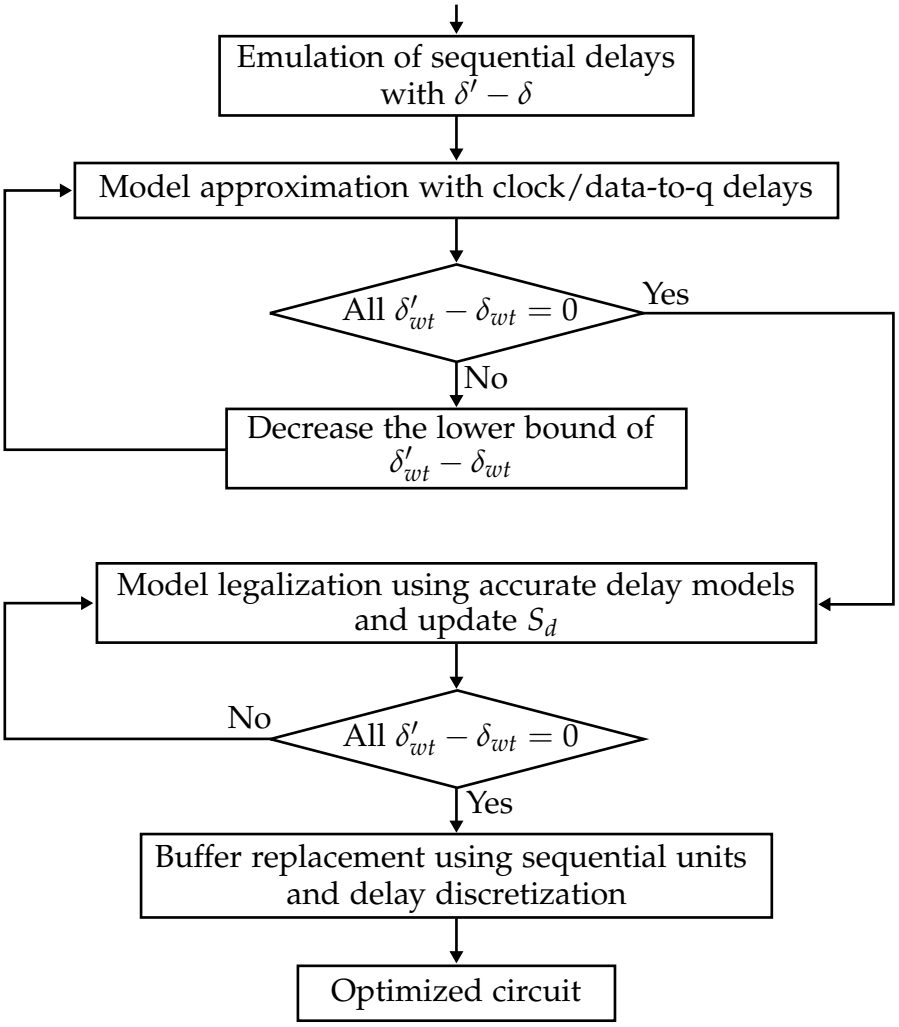

Figure 6.4: VirtualSync flow.

6.2 Iterative Relaxation

In applying the timing model above, a framework is introduced to identify the locations of delay units iteratively. The flow of this framework is shown in Fig. 6.4. Necessary locations of sequential delay units are refined gradually in the models from Section 6.2.1 to Section 6.2.3.

6.2.1 Emulation of Sequential Delay Units

After we remove all the flip-flops from the original circuit, the short paths may have extremely small delays. The gap between these delays and those of long paths is very large, which cannot be reduced with combinational delay units since they introduce the same delays to the fast and slow signals as shown in Fig. 6.1(a). Instead, only sequential delay units are able to reduce this gap so that the fast and slow signals still arrive at boundary flip-flops within the same clock cycle as those of the original circuit.

In the first step of the framework, the locations at which sequential delay units are indispensable are identified. Without these units, the fast and slow signals, such as those within feedback loops, may not be aligned properly into the correct clock cycles, even though unlimited combinational delay units can be inserted into the circuit. In practice, however, it is not easy to identify the exact locations of these units using the exact and complete model in (6.5)–(6.15) directly. To solve this problem, the delay effects of sequential delay units shown in Fig. 6.1(b)–(c) are firstly emulated, in which sequential delay units provide different delay paddings for short and long paths. Therefore, two variables δ_{wt} and δ'_{wt} are used to emulate different delays to be padded to slow and fast signals, respectively. When signals travel from u to z in Fig. 6.3, the relation of arrival times from nodes u to z can be written as

$$s_z \geq s_u + \xi_{uv} * r^u + d_{vw} * r^u + \delta_{wt} - \lambda_{tz} T \tag{6.18}$$

$$s'_z \leq s'_u + \xi_{uv} * r^l + d_{vw} * r^l + \delta'_{wt} - \lambda_{tz} T \tag{6.19}$$

$$\delta_{wt} \leq \delta'_{wt} \tag{6.20}$$

$$s'_u + \delta'_{wt} \leq s_u + \delta_{wt} \tag{6.21}$$

where the variables δ_{wt} and δ'_{wt} emulate delays introduced by sequential delay units. (6.20) specifies that the fast signal should be padded with more delays than the slow signal. (6.21) specifies that the arrival time of the fast signal should not exceed the arrival time of the slow signal after padding.

The optimization problem to find the potential locations of sequential delay units is

thus written as

$$\text{minimize} \quad \alpha \sum_{G}(\delta'_{wt} - \delta_{wt}) + \beta \sum_{G}(\delta'_{wt} + \xi_{uv}) - \gamma \sum_{G} d_{vw} \tag{6.22}$$

$$\text{subject to} \quad \text{(6.17)–(6.21) for each gate in } G \tag{6.23}$$

$$\text{constraints (6.1)–(6.2) for each boundary flip-flop} \tag{6.24}$$

where G is the set of all logic gates in the original circuit. This optimization problem also maximizes the overall delays of logic gates in the circuit, so that not only the inserted delays but also the area of the circuit can be reduced. α, β and γ are constants, set to 100,10,10, to specify the balance between sequential delay units, inserted buffers and logic gates roughly. Solving the optimization problem above identifies nodes with unequal padding delays δ_{wt} and δ'_{wt}, indicating potential locations of sequential delay units, as a set S. These delays may still violate the exact constraints in (6.5)–(6.15), so that they need to be refined further.

6.2.2 Modeling with Clock/Data-to-Q Delays of Sequential Delay Units

The optimization problem (6.22)–(6.24) does not consider the inherent clock-to-q delays of flip-flops and data-to-q delays of latches. Since these delays are introduced only at locations where sequential delay units are inserted, they need to be modeled for all the locations S returned by the previous step. A binary variable x_{wt} is introduced to represent whether a sequential delay unit appears at a location from S, and revise the constraints (6.18)–(6.19) as

$$s_z \geq s_u + \xi_{uv} * r^u + d_{vw} * r^u + x_{wt}\delta_{wt} + x_{wt}t_{cd\to q} * r^u - \lambda_{tz}T \tag{6.25}$$

$$s'_z \leq s'_u + \xi_{uv} * r^l + d_{vw} * r^l + x_{wt}\delta'_{wt} + x_{wt}t_{cd\to q} * r^l - \lambda_{tz}T \tag{6.26}$$

where $t_{cd\to q}$ represents clock-to-q delay or data-to-q delay, approximated with the same value for simplicity, and the delays δ_{wt}, δ'_{wt} and $t_{cd\to q}$ are only valid when x_{wt} is equal to 1. The inclusion of the binary variables x_{wt} is very computation-intensive, so that they can only be dealt with after the potential locations of sequential delay units are reduced to S by solving (6.22)–(6.24). Since x_{wt} is a binary variable, the multiplications $x_{wt}\delta_{wt}$ and $x_{wt}\delta'_{wt}$ can be converted into equivalent linear forms so that the overall formulation is still an ILP problem [CBD11].

Considering the inherent delays of sequential delay units, their locations can be refined further by solving the optimization problem as

$$\text{minimize} \quad \alpha \sum_{G/S} (\delta'_{wt} - \delta_{wt}) + \beta \sum_{G/S} (\delta'_{wt} + \xi_{uv}) - \gamma \sum_{G} d_{vw} \tag{6.27}$$

$$\text{subject to} \quad \text{(6.17)–(6.21) for each gate in } G/S \tag{6.28}$$

$$\text{(6.17), (6.20)–(6.21) and (6.25)–(6.26) for each gate in } S \tag{6.29}$$

$$\text{constraints (6.1)–(6.2) for each boundary flip-flop.} \tag{6.30}$$

In the implementation, the lower bound of $\delta'_{wt} - \delta_{wt}$ is also constrained when x_{wt} is equal to 1 and lower it iteratively, so that the most important locations for inserting sequential delay units are identified first, as illustrated in Fig. 6.4. The iterations terminate when no different δ_{wt} and δ'_{wt} exist, indicating no indispensable sequential delay units are required to align fast and slow signals. The refined locations of sequential delay units from this step are returned as a set S_d.

6.2.3 Model Legalization for Timing of Sequential Delay Units

In this step, the complete model described in Section 6.1.3 is applied to the locations in S_d to generate sequential delay units that are really required in the circuit. The optimization problem is described as

$$\text{minimize} \quad \alpha \sum_{G/S_d} (\delta'_{wt} - \delta_{wt}) + \beta \sum_{G/S_d} (\delta'_{wt} + \xi_{uv}) - \gamma \sum_{G} d_{vw} \tag{6.31}$$

$$\text{subject to} \quad \text{(6.17)–(6.21) for each gate in } G/S_d \tag{6.32}$$

$$\text{(6.5)–(6.12) and (6.14)–(6.17) for each gate in } S_d \tag{6.33}$$

$$\text{constraints (6.1)–(6.2) for each boundary flip-flop.} \tag{6.34}$$

After solving the optimization above, there might still be different δ_{wt} and δ'_{wt} in G/S_d, because the timing legalization of sequential delay units with the complete model in Section 6.1.3 may invalidate some locations in S_d. The accurate sequential delay model is applied to these new locations iteratively, until no different δ_{wt} and δ'_{wt} exists, indicating the remaining timing synchronization can be achieved with buffers and gate sizing directly.

6.2.4 Buffer Replacement with Sequential Units

After solving (6.31)–(6.34), delays d_{vw} of logic gates are discretized according to the library. Buffer delays ξ_{uv} are also determined. If ξ_{uv} is large, several buffers are needed for its implementation. As shown in Fig. 6.1, sequential delay units can introduce a very large delay. For example, a flip-flop can introduce a delay as large as $T + t_{cq} - t_h$, if the incoming signal arrives at the flip-flop right after a clock edge. According to this observation, buffers with large delays are replaced using sequential delay units to reduce area iteratively. In each iteration, the accurate sequential model (6.5)–(6.12) and (6.14)–(6.17) is applied to guarantee these new sequential delay units are valid. The iteration stops when no buffer can be replaced by sequential units. Buffers that cannot be replaced by sequential delay units are implemented directly in the optimized circuit.

6.3 Experimental Results

The proposed method was implemented in C++ and tested using a 3.20 GHz CPU. We demonstrate the results using circuits from the ISCAS89 benchmark set and the TAU 2013 variation-aware timing analysis contest as shown in Table 6.1. The number of flip-flops and the number of logic gates are shown in the columns n_s and n_g, respectively. The benchmark circuits were sized using a 45 nm library. To tolerate process variations, 10% of timing margin was assigned, so that r^u and r^l in previous sections were set to 1.1 and 0.9, respectively. The allowed phase shifts ϕ_{wt} in previous sections are 0, T/4, T/2 and 3T/4. The ILP solver used in the proposed framework was Gurobi [Gur13].

For timing optimization, combinational paths whose delays were within 95% range of the largest path delay in the circuit were selected. The source and sink flip-flops of these paths were allowed to be removed, while the other flip-flops in the circuits were considered as boundary flip-flops. All the combinational logic gates that can reach the flip-flops at the sources or sinks of these selected paths through a combinational path in the original circuit are considered as the critical parts of a circuit together. This extraction of the critical parts of a circuit in fact allows wave-pipelining within three sequential stages. In real circuits, the extracted critical parts

Table 6.1: Results of VirtualSync

Circuit			Cri. Part		Opt. Circuit			Comparison		Runtime
	n_s	n_g	n_{cs}	n_{cg}	n_f	n_l	n_b	n_t	n_a	$t(s)$
s5378	179	2779	35	1877	11	14	94	11.5%	2.84%	121.6
s9234	228	5597	91	3981	58	45	91	2.5%	-5.17%	7251.1
s13207	669	7951	191	3483	95	73	52	2.5%	-1.09%	3121.6
s15850	534	9772	71	3847	72	18	26	0%	6.01%	289.97
s38584	1452	19253	126	9498	62	75	46	0.5%	-0.50%	1142.3
systemcdes	190	3266	92	3232	90	81	227	3.5%	2.43%	7310.5
mem_ctrl	1065	10327	136	7500	101	39	140	3.5%	0.97%	3750.1
usb_funct	1746	14381	138	5378	123	37	60	4%	0.21%	1211.7
ac97_ctrl	2199	9208	237	4873	42	172	218	0%	-9.76%	2936.8
pci_bridge	3321	12494	239	9510	188	68	338	3%	0.05%	7418.5

may overlap, thus allowing wave-pipelining with more than three stages. Since the original circuits have been sized to reduce the clock period, the critical parts of the circuits still occupied a large portion of original circuits. As shown in the n_{cs} and n_{cg} columns in Table 6.1, more than 7% of flip-flops and more than 35% of logic gates have been selected for timing optimization.

The column n_f and n_l show the numbers of flip-flops and latches after optimization, respectively. The sums of these numbers are comparable or even smaller than the numbers of flip-flops in the original critical parts of the circuits. The numbers of extra inserted buffers to match arrival times are shown in the column n_b. Compared with the number of original logic gates shown in the column n_{cg}, these numbers show that the cost due to the inserted buffers is still acceptable.

To verify the improvement of circuit performance, we gradually reduced the clock period by 0.5% of the clock period obtained from combining retiming and sizing. The column n_t in Table 6.1 shows the clock period reduction compared with the circuits after retiming&sizing. The maximum and average reduction are 11.5% and 3.1%, respectively, which resulted from the compensations between several flip-flop stages and the removal of clock-to-q delays and setup time requirements on critical paths. For most cases, the minimum clock periods have been pushed even further than those from retiming&sizing, the limit of the traditional sequential design. This

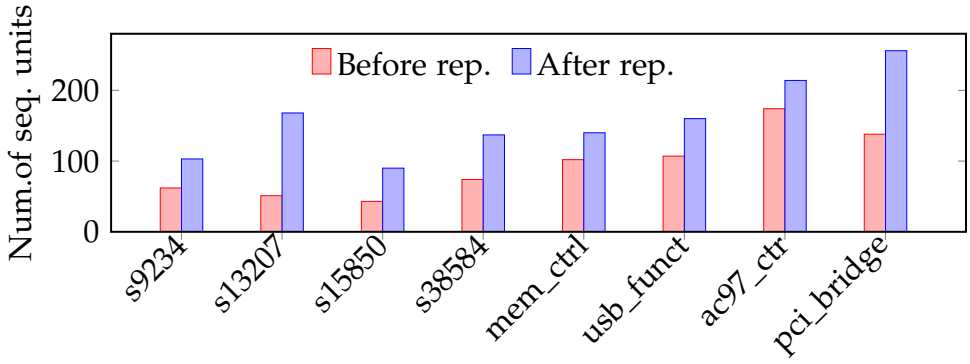

Figure 6.5: Comparison of sequential delay units after buffer replacement.

comparison demonstrates that the proposed method provides a very good potential method to improve circuit performance specially at a late stage of timing closure further, because no circuit redesign is required. The area increase compared with retiming&sizing is shown in column n_a. In the cases with area increase, the overhead is still negligible; in other cases, the area is even smaller because unnecessary flip-flops were removed in the proposed framework, whereas in retiming flip-flops can only be moved instead of being removed. The last column t_r in Table 6.1 shows the runtime of the proposed method. Since the ILP formulation with the complete model in Section 6.1.3 is NP-hard, it is impractical to find a solution with respect to area and clock period. In the experiments, the runtime with iterative relaxations is acceptable at a late stage of timing closure, but still has space for further improvement.

In the proposed framework, sequential delay units are first inserted only at necessary locations to delay signal propagations. Afterwards, more of them are used to replace buffers to reduce area, as described in Section 6.2. Figure 6.5 shows the numbers of sequential delay units before and after buffer replacement, which shows a clear increase of the number of such delay units to replace buffers. Figure 6.6 shows the area comparison after this replacement. In most test cases, the area taken by the sequential delay units and buffers in the optimized circuits is less than 1% of those replaced buffers, demonstrating the efficiency of sequential delay units in delaying fast signals.

The comparison of the area overhead in Table 6.1 is between the method retiming&sizing with its own clock period and the proposed method with a smaller clock

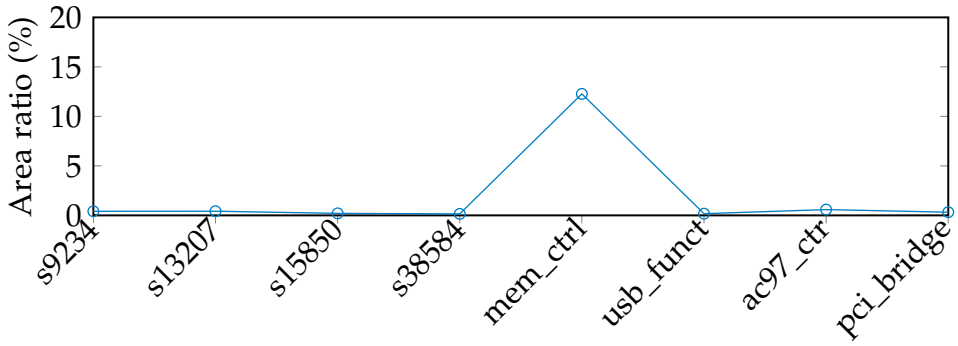

Figure 6.6: Area comparison before and after buffer replacement.

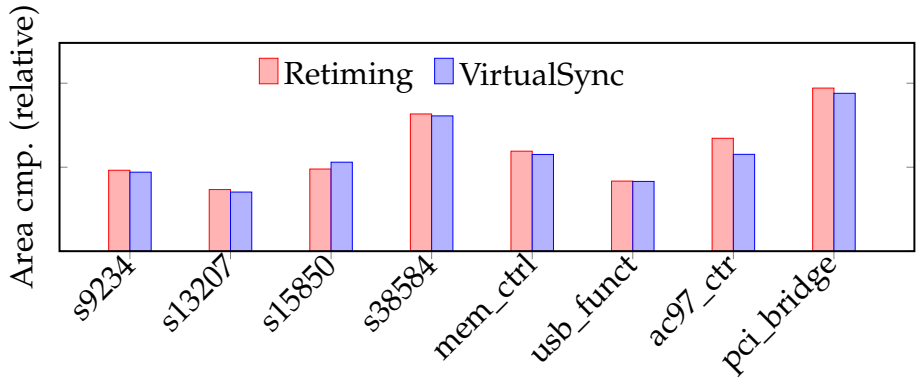

Figure 6.7: Area comparisons with retiming&sizing with the same clock period.

period. To demonstrate the area efficiency of the proposed method, we also compared the proposed method and the timing&sizing with the same clock period from the latter. The results are shown in Fig. 6.7. In most cases, the area overhead with our framework is smaller.

6.4 Summary

In this chapter, a new timing model is proposed, in which sequential components and combinational logic gates are considered as delay units. They provide different delay effects on signal propagations on short and long paths. With this new timing model, a timing optimization framework has been proposed to insert delay units only at necessary locations. With this technique, circuit performance can be improved by up to 11.5% with a negligible increase of area overhead.

Chapter 7

Flexible Timing for Netlist Security

Today's semiconductor business model involves many global vendors from various countries and regions. This distributed supply chain makes integrated circuits vulnerable to attacks and counterfeiting in nearly all phases from design to post-fabrication. Consequently, the research community has invested a great effort to deal with security challenges [GHD+14].

A major IC counterfeiting threat is the production of illegal chips by a third party with a netlist reverse engineered from authentic chips. In reverse engineering, authentic chips are delayered and imaged to identify logic gates, flip-flops, and their connections. Afterwards, the recognized netlist can be processed by a standard IC design flow and manufactured in a foundry, even with a different technology. This reverse engineering flow gives counterfeiters much freedom in reproducing authentic chips, because the recognized netlist carries all necessary design information and counterfeiters can revise and optimize it freely.

Several techniques have been proposed to thwart reverse engineering attacks on authentic chips. Firstly, IC camouflage tries to prevent the netlist from being recognized easily. In [BRPB14] transistors are manipulated with a stealthy doping technique during manufacturing so that they function differently than they appear. The work in [MBPB15, RSSK13, RSK13] mixes real and dummy contacts to camouflage standard cells. The method in [LT15] explores netlist obfuscation by iterative logic fanin cone analysis at circuit level. Moreover, the method in [LSM+16] introduces a quantitative security criterion and proposes camouflaging techniques with a low-overhead cell library and an AND-tree structure. In addition, logic locking inserts additional logic gates, e.g., XOR/XNOR in [RPSK12, RKM08], AND/OR in [DBN+14] and MUX in [PM15], into the netlist to disable its function if the correct key is not applied. This method is expanded in [XS17] to incorporate delay information into the locking mechanism.

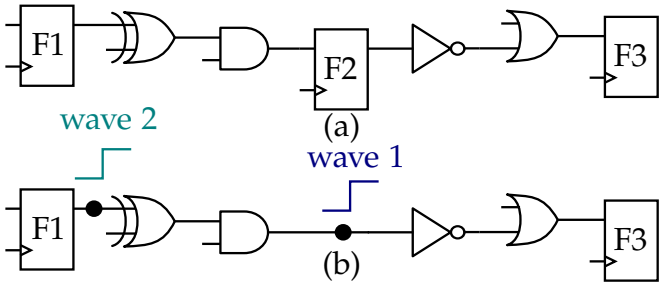

Figure 7.1: Conventional timing and wave-pipelining: (a) Single-period clocking; (b) Pipelining with two data waves.

The methods discussed above all focus on either making the netlist more difficult to be recognized, or making the correct behavior of the circuit dependent on additional input information even after the netlist is recognized. However, they are still restricted to the conventional single-period clock timing model so that attackers only need to recognize the netlist correctly.

In this chapter, a method is proposed to invalidate the assumption that a netlist completely represents the function of a circuit. With the help of wave-pipelining paths, this method forces attackers to capture delay information from manufactured chips, which is a very challenging task because false paths are also introduced. An analysis of counterfeiting of digital circuits is made in Section 7.1. Thereafter, the concept of wave-pipelining deployed in the proposed method is introduced in Section 7.2. This application of wave-pipelining is different compared with that introduced in Section 2.3.4, where the objective of wave-pipelining is to optimize the timing performance of circuits by implementing pipelining in logic without the use of intermediate latches or registers. However, in this chapter, wave-pipelining is applied to preventing the counterfeiting of digital integrated circuits. Afterwards, potential attack techniques are analyzed and counter measures to thwart them are proposed in Section 7.3. The implementation details of constructing wave-pipelining paths and false paths are described in Section 7.4.

7.1 Analysis of Counterfeiting of Digital Circuits

In the conventional single-period clocking timing model, all the paths in a combinational block operate within one clock period. Figure 7.1(a) shows a part of a sequen-

tial circuit with three flip-flops F1, F2 and F3. To guarantee the correct operation, setup and hold time constraints should be satisfied between all pairs of flip-flops and interaction between different clock stages can be ignored. In this single-period clocking model, the netlist carries all logic information and this simplification allows attackers to counterfeit chips relatively easily because they only need to recognize the logic types of gates, flip-flops, and interconnect connections during reverse engineering.

To thwart the attack attempt on a design, a method is proposed to invalidate the conventional timing model in some parts of the circuit. For example, the flip-flop in the middle of Fig. 7.1(a) is removed to construct the circuit in Fig. 7.1(b). On the combinational path from F1 to F3, there are now two data waves without a flip-flop separating them. If the second wave does not catch the first one before it is latched by F3, the correct function of the circuit is still maintained. This technique is called *wave-pipelining (WP)* and has been investigated for circuit optimization [HLCG95, BCKL98, SV09]. When attackers recognize a netlist as in Fig. 7.1(b), they face the challenge to determine whether there should be one or two logic waves. If they assume the former and process the netlist using a standard EDA flow, the circuit loses synchronization because the data at the input of F3 is latched one clock period earlier. If they want to determine whether it is the latter case, additional effort is required to extract the timing information for the combinational path. In this way, the function of the circuit depends on both its structure and the timing of combinational paths.

Although wave-pipelining paths look similar to multiple-cycle paths in digital design, the essential difference is that there is only one wave on a multiple-cycle path at a moment and the circuit still works if a multiple-cycle path is optimized to finish its calculation in one clock period, or if the clock frequency is lowered to make it work in one clock period. Therefore, multiple-cycle paths cannot be used to replace wave-pipelining paths to increase netlist security.

7.2 Wave-Pipelining Paths

A wave-pipelining path such as the one in Fig. 7.1(b) allows two data waves propagating on the path at the same time. Since the second data wave should not catch the

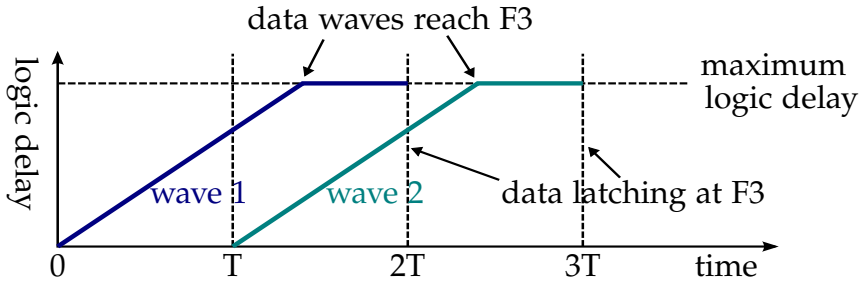

Figure 7.2: Temporal/spatial diagram for wave propagation on a combinational path.

first one, special timing constraints should be specified for this path. The scenario of data wave propagation is illustrated in Fig. 7.2. At first, wave 1 is injected into the path by F1. This data wave propagates along the path continuously and should reach F3 after the first rising clock edge at T and before the second rising clock edge at $2T$. At time $2T$, the first data is latched by F3. The second wave is injected by F1 at the rising clock edge at time T and it starts to propagate along the same path. Since this wave arrives at F3 with a delay larger than T, it does not catch the first wave at any time during the propagation, shown as the vertical gap between the two data waves in Fig. 7.2. Consequently, the two data waves on the path never interfere and F3 always latches the same value as in the original circuit shown in Fig. 7.1(a).

In forming wave-pipelining paths, a flip-flop is removed from the circuit as in the example from Fig. 7.1(a) to 7.1(b). In practice, this operation may lead to many paths with wave pipelining, because any combinational path reaching F2 together with any path starting from F2 forms a new wave-pipelining path. All these wave-pipelining paths should meet two constraints. First, the delay of a path should be larger than the clock period T; otherwise, the data wave is latched at the first rising clock edge instead of the second by F3. Second, the delay of the path should be no larger than $2T$ to guarantee that the data is latched by F3 in time. Assume the set of all these paths is P and the delay of a path $p \in P$ is d_p. The timing constraints for all these paths can be written as

$$d_p \geq T + t_h, \forall p \in P \iff \min_{p \in P}\{d_p - t_h\} \geq T \tag{7.1}$$

$$d_p \leq 2T - t_{su}, \forall p \in P \iff \max_{p \in P}\{d_p + t_{su}\} \leq 2T. \tag{7.2}$$

After removing a flip-flop from the circuit, if all the wave-pipelining paths meet the

two constraints (7.1) and (7.2), the wave-pipelining version of the circuit is functionally equivalent to the original circuit.

If attackers try to duplicate the function of a circuit by reverse engineering, in addition to netlist extraction, they need to find out which paths have wave-pipelining. For this purpose, they need to capture the delay information of logic gates or paths, which is very challenging and increases the cost of counterfeiting significantly. Without this information, the circuit still does not work if all the paths are assumed as working within a single clock period by default, since the logic containing wave-pipelining loses synchronization with other parts of the circuit in this case.

To apply wave-pipelining, some changes in existing design tools should be made to guarantee the timing constraints (7.1) and (7.2). For example, physical design tools need to maintain not only an upper bound of path delays but also a lower bound. In addition, timing analysis algorithms need to be aware of these constraints instead of only checking the worst case against one clock period. Testers are required to consider wave-pipelining paths in post-silicon test. Since only the paths with wave-pipelining in the circuit should be dealt with specially, these additional constraints may be imposed in a post-processing/ECO step in practice.

7.3 Attack Techniques and Counter Measures

In attacking a design with wave-pipelining, if attackers have no knowledge that this technique has been applied, the recognized netlist by reverse engineering does not function correctly. Once attackers become aware of this technique, various methods may be deployed to identify where the wave-pipelining paths are or to circumvent them simply. In the assumed attack model, the available information includes a netlist recognized by reverse engineering and estimated delays of logic gates as well as interconnects with an inaccuracy factor τ. The objective of the attack is to identify on which combinational paths in the netlist wave-pipelining is applied. The potential attack techniques are summarized in Fig. 7.3.

The first attack technique is to measure all gate and interconnect delays while the netlist is recognized by reverse engineering. With all gate and interconnect delays known, path delays can be calculated from the netlist easily. Since the delays of

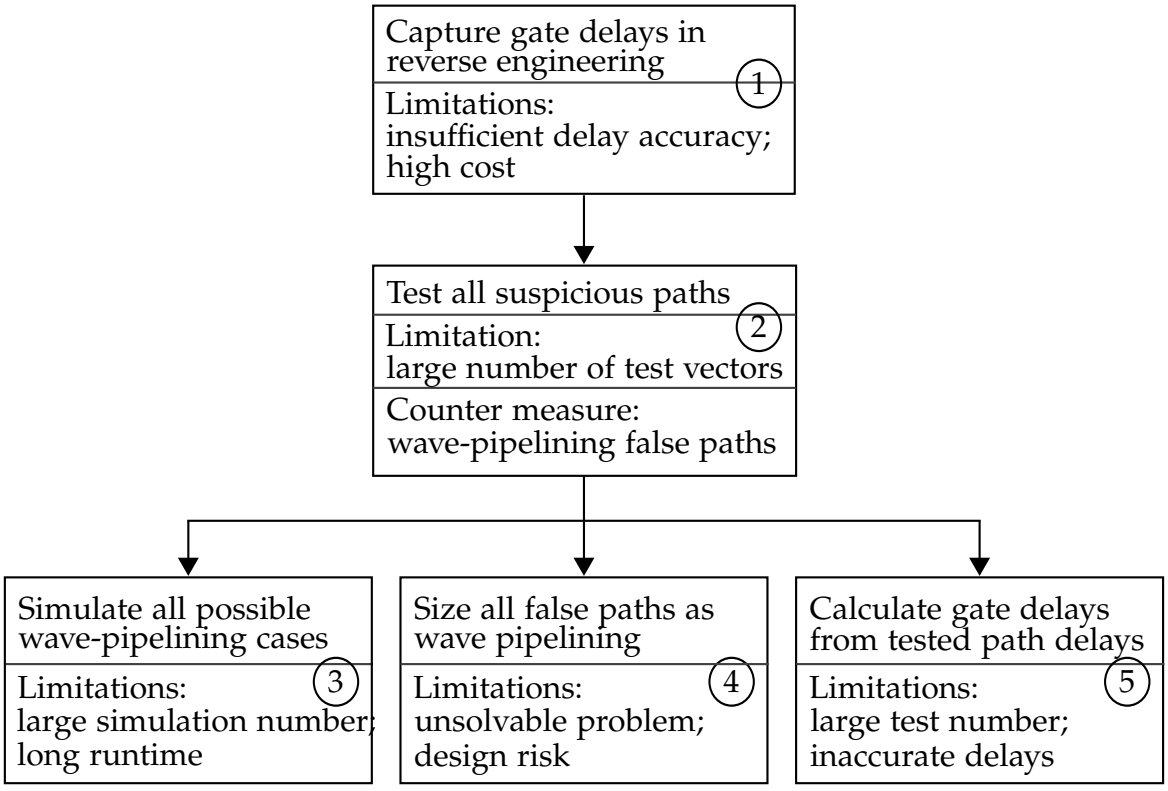

Figure 7.3: Attack techniques to identify or circumvent wave pipelining, where the last three techniques may be combined to reduce the problem space of attack.

wave-pipelining paths are between T and $2T$ as defined in (7.1) and (7.2), these paths can therefore be identified. The challenge of this attack technique is that it is difficult to extract accurate gate and interconnect delays just from reverse engineering due to unknown process parameters, challenges in 3D RC extraction, and switching-window-dependent crosstalk-induced delay variations, etc. Assume that the real delay of a path is d and the delay recognition technique suffers an inaccuracy factor τ ($0 < \tau < 1$). Consequently, this path delay can be any value in the range $[(1-\tau)d, (1+\tau)d]$ when recognized. If the upper bound of a path delay is smaller than T, this path is definitely a single-period clocking path. If the lower bound of a path delay is larger than T, the path is definitely a wave-pipelining path. However, if a path delay covers the clock period T, namely,

$$(1-\tau)d \leq T \leq (1+\tau)d. \tag{7.3}$$

This path can only be considered as suspicious of wave-pipelining but without a

clear differentiation. In the following, we call the range $[(1-\tau)d, (1+\tau)d]$ the *gray region* for a path with delay d. In reality, a well-optimized design contains many critical paths with delays close to the clock period T so that their gray regions cover T easily. When constructing wave-pipelining paths in the proposed method, their delays are guaranteed in the gray region. Consequently, this direct delay recognition in reverse engineering is not sufficient to decipher the information of wave-pipelining.

With the estimated delays, attackers can actually narrow down the number of potential wave-pipelining paths, because paths with delays definitely smaller or larger than T considering the inaccuracy in delay estimation can be screened out. *The second attack technique* is to test the delays of the remaining paths using authentic chips from the market. With the netlist recognized, it is not difficult to determine test vectors to trigger the suspicious paths. Since the only information of interest is whether a path delay is larger than T, only one delay test for each path is sufficient. Without considering the cost to test many paths, this test strategy is in fact able to differentiate wave-pipelining paths from other paths eventually.

To prevent all suspicious paths from being tested, a counter measure is proposed to create unsensitizable paths with wave-pipelining. When wave-pipelining paths are constructed by removing flip-flops, the paths are preferred that, viewed directly with the conventional single-period clocking, are false paths, which cannot be sensitized by any test vectors.

Definition 1. *False Path: A combinational path which cannot be activated in functional mode or test due to controlling signals from other paths [YX10, DYG89]. On the contrary, true paths can be activated in functional mode or test.*

Definition 2. *Wave-Pipelining False Path (WP False Paths): A combinational path with wave pipelining that is a false path when viewed with the conventional single-period clocking.*

Wave-pipelining false paths are true paths with data waves propagating along them when the circuit is running, but they are false paths when the netlist is examined only. An example of wave-pipelining false paths is shown in Fig. 7.4, which is a snippet of the s298 circuit from the ISCAS89 benchmark set. When the flip-flop in the middle is removed, the dashed path becomes a wave-pipelining path and also a false path, if it is considered as working within a single clock period. In this case,

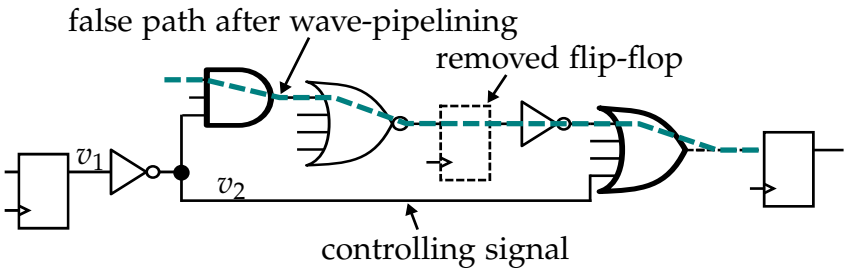

Figure 7.4: Two true paths form a wave-pipelining false path.

a signal switching at the beginning of the dashed path never reaches the final flip-flop. If the signal v_2 has a value '1', which is the controlling signal to an OR gate, it blocks the dashed path at the last OR gate; if the signal v_2 has a value '0', it blocks the dashed path at the AND gate right away. Consequently, the dashed path cannot be triggered for delay test and attackers have no way to differentiate it from all the other false paths in the original circuit, which may contribute up to 75% of all the combinational paths in real circuits [HPA97]. The counter measure of forming wave-pipelining false paths relies on the circuit structure. If there is no such a structure in the original circuit, this technique cannot be applied. In practice, however, most relatively large circuits have such paths, as shown in the experimental results in Section 7.5.

Since the delays of false paths cannot be tested, *the third attack technique*, brute-force logic simulation, could be considered to differentiate the camouflaged false paths from real false paths. In this method, each false path that cannot be excluded by delay screening in the first step is assumed to be a real false path once and a wave-pipelining false path once. Assuming the number of such paths is n, then 2^n simulations of the complete circuit should be performed to check which combination is correct. In theory, this method can eventually find the correct combination of real false paths and wave-pipelining false paths. However, it is still impractical because of the unaffordable simulation time due to the large number of false paths in the original design [HPA97, YX10] and the very long runtime for a full simulation of the complete circuit. If an attacker tries to decipher the circuit by sweeping the operation frequency with parameterized wiring parasitics exhaustively, the data of parasitics must be available, which, however, are difficult to extract directly from the chips sold in the market. In addition, a systematic simulation-learning solution is also required, leading to additional counterfeiting effort and risk induced by the

proposed timing camouflage technique.

The fourth technique to attack wave-pipelining false paths is to consider all false paths in the circuit as wave-pipelining paths and size logic gates so that delays of all these paths meet the constraints (7.1) and (7.2). The concept behind this technique is that false paths are not triggered anyway so that they do not affect the logic of the circuit if their delays are larger than the clock period T. This assumption, however, is too optimistic because false paths sized to have delays larger than T may still affect the normal circuit operation [DYG89]. Another challenge of this attack technique is that it is very difficult to find a solution to size so many false paths without affecting the normal true paths whose delays should be smaller than T, because a gate delay may appear on many paths, either false or active paths.

The fifth technique to identify wave-pipelining paths is to calculate all gate delays in a circuit from path delays measured by at-speed test, such as applied in [VDP14]. Since path delays are linear combinations of gate delays, the measured path delays can be used to calculate gate delays by linear algebra. The challenges of this method are: 1) a large number of combinational paths should be tested in a commercial design; 2) all logic gates should appear on testable paths in a way that the coefficient matrix of linear equations has a rank equal to the number of gate/interconnect delays, even in view of a large percentage of false paths [HPA97,YX10]; 3) inaccuracy in at-speed test of path delays due to environmental factors such as noise and temperature as well as the nature of binary-search of at-speed delay test. Detailed analysis of attack and counter measures based on this technique is still an open question requiring further exploration at this moment. Potential methods such as machine learning might be applied to facilitate the attack, but further counter measures, such as logic and delay camouflage at gate level, may be combined with wave-pipelining paths to form a holistic netlist protection solution.

7.4 Wave-Pipelining Construction

When constructing wave-pipelining paths into a circuit while maintaining its original function, the constructed paths should be guaranteed to meet the timing constraints (7.1) and (7.2). To counter the attack techniques discussed in Section 7.3,

the constructed paths should not be screened out easily by delay test and estimation. Furthermore, the constructed wave-pipelining paths should contain false paths when considered as single-period clocking paths. The wave-pipelining construction problem can thus be formulated as follows.

Inputs: An optimized design; delay information; the given clock period T; the delay recognition inaccuracy factor τ ($0 < \tau < 1$); the required numbers of wave-pipelining true and false paths n_{wpt} and n_{wpf}.

Outputs: A revised design containing at least the given numbers of wave-pipelining true and false paths. The delays of these wave-pipelining paths should meet the gray region requirement (7.3).

Objectives: The original design should be kept unchanged as much as possible; the increased resource usage should be as little as possible.

7.4.1 Work Flow of Wave-Pipelining Construction

The major steps to construct wave-pipelining paths are shown in Fig. 7.5. To construct wave-pipelining false paths, flip-flops are visited in the netlist iteratively. At each flip-flop ff_i, whether there are wave-pipelining false paths formed from single-period true paths on the left and on the right of ff_i is checked. The number of such paths is stored in n_f as shown in L5. Thereafter, wave-pipelining false paths are constructed at this flip-flop with the function construct_WP_paths(ff_i, T, τ) which will be explained later.

As shown in Fig. 7.1(b), a wave-pipelining path requires that the flip-flop at the beginning of the path and the flip-flop at the end of the path are kept in the circuit. These fanin and fanout flip-flops are inserted into the set F_w and all the flip-flops tracked by F_w cannot be considered as candidates to construct wave-pipelining paths.

In the last step of the proposed method, additional wave-pipelining paths are constructed that are still true when viewed with the single-period clocking model. These paths are used to guarantee that attackers must test all single-period clocking or wave-pipelining true paths whose delays are in the gray region. Without these paths, attackers can assume all testable paths are clocked by a single clock period

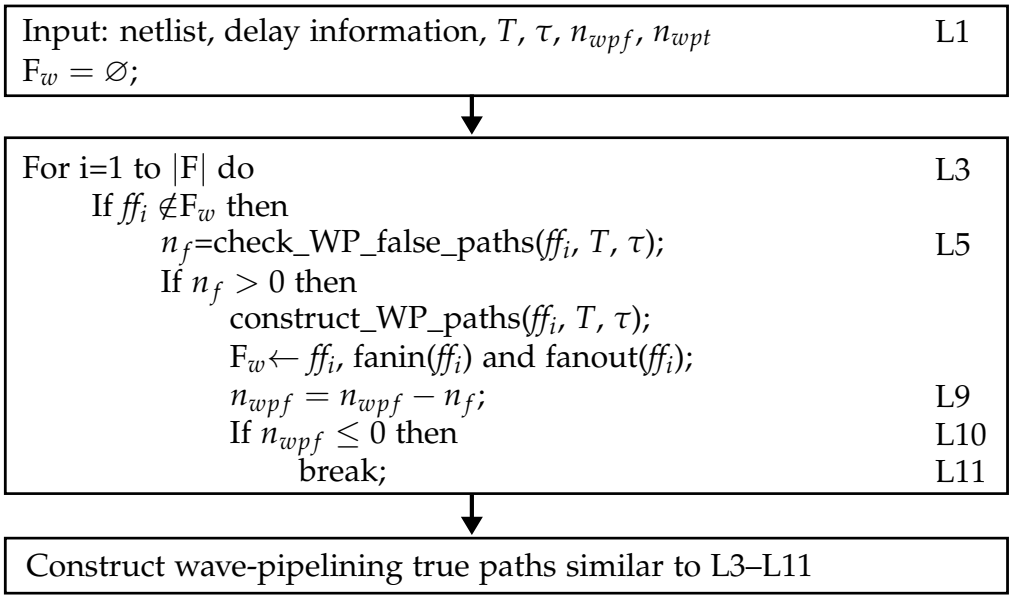

Figure 7.5: Major steps of wave-pipelining construction.

and skip the expensive test procedure. The path construction in this step is nearly the same as L3–L11 in Fig. 7.5. The only differences are that at L5 wave-pipelining true paths should be checked and in L9 and L10 n_{wpt} should be used as the number of such paths to be constructed.

7.4.2 False Path Checking

In the work flow above, whether a path is a false path need to be checked. In the proposed method, the statically unsensitizable paths are considered as false paths [DKM93,Cou10], such as the false path shown in Fig. 7.4. In this example, the path cannot be sensitized because the controlling signal blocks either the AND gate or the last OR gate no matter what its value is.

To verify whether a path is statically unsensitizable, a Boolean variable is assigned to the output of each gate and formulate false path checking as a SAT problem [Cou10]. The logic relations between these variables are established according to functions of logic gates. If a path can be sensitized, all the side inputs of the path must be set to the non-controlling values. For example, the path in Fig. 7.4 requires that the condition $(v_2 \wedge \neg v_2)$ is true, which is, however, always false. In implementing the function check_WP_false_paths(ff_i, T, τ) in Fig. 7.5, 500 paths are selected

randomly that drive the current flip-flop ff_i and the false paths are excluded from them, because the wave-pipelining paths to be constructed should be formed by two single-period clocking true paths. Similarly 500 paths are selected that are driven by ff_i and the false paths are excluded. The selected number is in fact abundant in the circuits as demonstrated by experimental results in Section 7.5. The concept of this path selection is illustrated in Fig. 7.6(a).

7.4.3 Wave-Pipelining Path Construction

At flip-flop ff_i, wave-pipelining paths are needed to be constructed in the circuit with the function construct_WP_paths(ff_i, T, τ) in Fig. 7.5. Unfortunately, the intuitive idea to remove flip-flop ff_i in the middle is not a viable solution, because there usually are many short paths on the left and on the right of ff_i and connecting them directly generates many paths whose delays are too small to meet the lower bound of the path delay constraint (7.1).

To solve this problem, the logic in the circuit is duplicated and the gates are sized so that the delays of all wave-pipelining paths meet (7.1) and (7.2) as illustrated in Fig. 7.6(b). In the duplicated circuit on the right of ff_i, only the flip-flops at which wave-pipelining paths terminate are kept. The other flip-flops stay in the original circuit. Afterwards, the logic gates backwards are deleted to remove those gates that do not drive any flip-flop to reduce resource usage. When duplicating the logic on the left of ff_i, however, all the logic gates are needed to be kept to maintain the correct function of the circuit.

In the duplicated logic in Fig. 7.6(b), flip-flop ff_i are not duplicated. Therefore, all combinational paths in the duplicated logic are wave-pipelining paths and their delays should meet the gray region requirement (7.3) as well as (7.1)–(7.2). To meet these constraints, the gates are sized in the duplicated logic with an ILP formulation. In this formulation, two variables are assigned to a pin of a logic gate to represent the latest and earliest arrival times, respectively. Assume that an input pin of a gate is indexed by i and the variables are written as $\overline{a}_i$ and $\underline{a}_i$. Similarly, assume that the output pin of the gate is indexed by j and the two variables are $\overline{a}_j$ and $\underline{a}_j$. Furthermore, the gate delay from an input pin to the output pin is written as d_{ij}, which is a variable since the corresponding logic gate is sized. With these definitions,

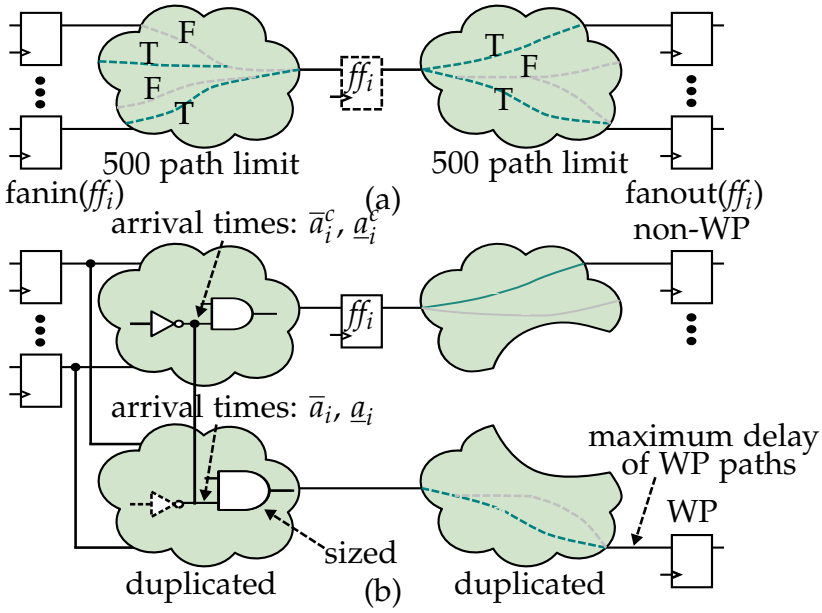

Figure 7.6: WP path construction. (a) The number of paths on each side of ff_i is limited to 500. A WP false path is constructed by two single-period clocking true paths. (b) Logic duplication and gate sizing.

the arrival time constraints from an input pin to the output pin can be written as

$$\overline{a}_j \geq \overline{a}_i + d_{ij} \tag{7.4}$$

$$\underline{a}_j \leq \underline{a}_i + d_{ij}. \tag{7.5}$$

To reduce the number of duplicated gates, we try to connect the input pins of logic gates in the duplicated logic to the original gates as much as possible, as illustrated in Fig. 7.6(b). In the original logic, the latest and the earliest arrival times are constants. Assume that the two arrival times to the original counterpart of an input pin are $\overline{a}_i^c$ and $\underline{a}_i^c$, and a 0-1 variable p_i indicates whether the input pin in the duplicated logic should be driven by the original logic. the constraints (7.4)–(7.5) can be extended as

$$\overline{a}_j \geq \overline{a}_i + d_{ij} - p_i M \tag{7.6}$$

$$\overline{a}_j \geq \overline{a}_i^c + d_{ij} - (1 - p_i) M \tag{7.7}$$

$$\underline{a}_j \leq \underline{a}_i + d_{ij} + p_i M \tag{7.8}$$

$$\underline{a}_j \leq \underline{a}_i^c + d_{ij} + (1 - p_i) M, \tag{7.9}$$

where M is a very large positive constant used to transform the conditional constraints to linear constraints [CBD11]. In either case when the input pin is connected or disconnected in the duplicated logic, only two constraints in (7.6)–(7.9) are valid.

In the description above, gate delays are not bounded strictly. Instead, they are allowed to exceed the maximum gate delays defined in the library, respectively, so that the path delay constraints (7.1)–(7.2) and the gray region constraint (7.3) can be guaranteed. However, we try to keep the increased gate delays as small as possible, so that they can be absorbed by interconnect delays during physical design. To reduce resource usage and avoid excessive delay padding, the optimization problem can be formulated as

$$\text{minimize} \quad \alpha \sum_{I} d_{ij} - \beta \sum_{I} p_i \tag{7.10}$$

$$\text{subject to} \quad \text{(7.1)–(7.2), (7.3), (7.6)–(7.9),} \tag{7.11}$$

where $\alpha \gg \beta$ and I is the index set of all input pins. After the ILP problem above is solved, the gates that do not drive any other gates in the duplicated logic are removed from the circuit.

7.5 Experimental Results

The proposed method was implemented in C++ and tested using a 3.20 GHz CPU. The results using circuits from the ISCAS89 benchmark set are demonstrated. The number of flip-flops and the number of logic gates are shown in the columns n_s and n_g in Table 7.1, respectively. The benchmark circuits were sized using a 45 nm library. 15% of timing margin is kept to tolerate PVT (Process, Voltage and Temperature) variations and the inaccuracy factor τ of delay estimation in (7.3) is set to 20%. Gurobi [Gur13] is used to solve the optimization problems in the proposed method.

The results of wave-pipelining path construction are shown in Table 7.1. The column n_t shows the number of single-period clocking combinational paths that are true paths in the original circuits and whose delays meet the gray region requirement (7.3). When attackers try to detect the locations of wave-pipelining paths, these true paths need to be tested to determine whether their delays are actually larger or

Table 7.1: Results of Constructing WP Paths

Circuit			WP Cons.					Runtime
	n_s	n_g	n_t	n_{wpt}	n_{wpf}	n_d	n_p	$t_r(s)$
s35932	1728	16065	180039	20	1022	178	80	625.29
s38584	1452	19253	502561	48	431	130	117	3685.88
s38417	1636	22179	298922	82	63	321	65	1711.01
s15850	522	9772	361544	20	838	186	141	3018.06
s13207	669	3716	927424	20	115	152	74	446.17
s9234	228	5597	10922	20	983	148	83	291.45
s5378	179	2779	10143	401	78	139	55	266.022
s4863	104	2342	4140	680	0	184	77	3766.98
s1423	74	657	8506	450	12	75	213	1170.71
s1238	12	508	15	3	4	94	90	2.07

smaller than T. These results show that attackers need to perform many expensive test iterations to attack a chip even if they can estimate gate delays to some degree.

The column $\boldsymbol{n_{wpt}}$ shows the numbers of wave-pipelining true paths whose delays are in the gray region. These paths are used to guarantee that attackers must test all single-period clocking or wave-pipelining true paths whose delays are in the gray region. The column $\mathbf{n_{wpf}}$ shows the numbers of wave-pipelining false paths whose delays are in the gray region. These paths are used to obstruct the attempt that attackers test all paths to determine the wave-pipelining paths. In the experiments, the target numbers of wave-pipelining true and false paths both is set to 10. The construction of wave-pipelining true and false paths shown as in Fig. 7.5 is executed. When wave-pipelining false paths were constructed using the technique illustrated in Fig. 7.6, wave-pipelining true paths were also found in the duplicated circuit snippet. In addition, wave-pipelining false paths were found in the circuit snippet duplicated to construct wave-pipelining true paths. Consequently, the numbers of these paths shown in the columns n_{wpt} and n_{wpf} are larger than 10 for many test cases except s4863 and s1238. In s4863 there is no wave-pipelining false path and in s1238 the numbers of wave-pipelining paths are very small due to the limited circuit size. In all the large test cases, however, wave-pipelining paths have been

Table 7.2: Wave-pipelining False Paths in Test Cases

Circuit	n_f	$\tau = 0.2$	$\tau = 0.1$
s5378	122757	80386	4845
s4863	0	0	0
s1423	2331927	58992	37312
s1238	392	0	0

constructed successfully.

The column n_d in Table 7.1 shows the number of logic gates duplicated in the final circuits. Since wave-pipelining paths are only inserted at limited locations, generally the number of duplicated gates does not increase with respect to circuit size. The column n_p shows the number of delay units equivalent to buffer delays that were inserted to extend wave-pipelining path delays. Since the number of duplicated gates does not increase with respect to circuit size, the area cost for constructing wave-pipelining paths is negligible in relatively large circuits. The last column t_r in Table 7.1 shows the runtime of the proposed method, which is acceptable because wave-pipelining construction is a one-time effort.

In the proposed method, the number of wave-pipelining false paths depends on the original circuit structure. If there is no such a path in a circuit, we cannot use this technique to thwart test-based attack. To verify whether this feature is common for most circuits, we checked the numbers of wave-pipelining false paths in the test cases and the results are shown in Table 7.2, where the column n_f shows the numbers of wave-pipelining false paths without considering path delays. The columns $\tau = 0.2$ and $\tau = 0.1$ show the numbers of such paths with delays meeting the gray region requirement (7.3). Since $\tau = 0.1$ means that the gray region is smaller, the numbers of wave-pipelining paths under this condition decrease compared with the $\tau = 0.2$ cases. For all the other test cases not appearing in Table 7.2, the numbers of such paths corresponding to the three columns are all larger than 100k, meaning that there are plenty of wave-pipelining false paths which can be used to camouflage the timing of these circuits.

In the proposed wave-pipelining construction formulation (7.10)–(7.11), the number of signals is maximized that can be driven by the original circuit as illustrated in

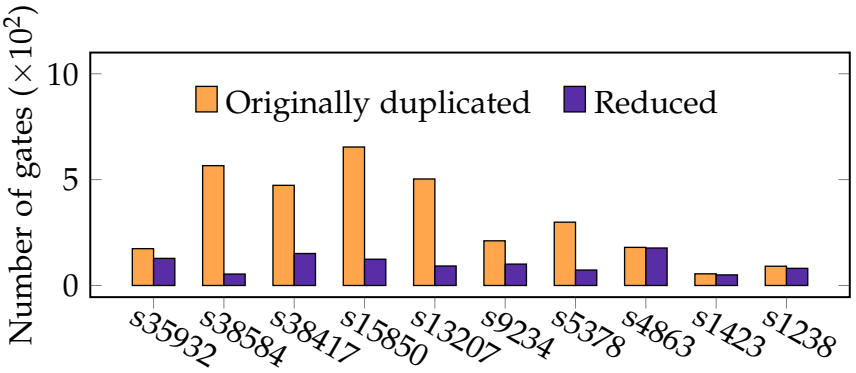

Figure 7.7: Comparison of gate numbers before/after reduction.

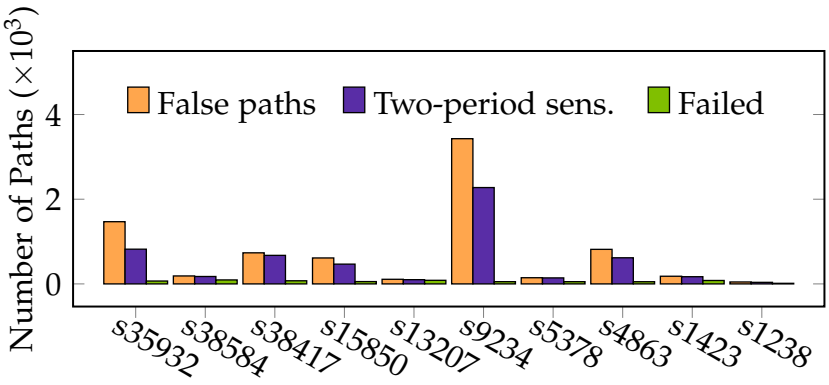

Figure 7.8: Results of false path sizing attack.

Fig. 7.6. Consequently, the number of logic gates in the duplicated circuit can be reduced. Fig. 7.7 compares the numbers of gates in the originally duplicated circuit before the removed flip-flop in Fig. 7.6 and the number of gates after reduction. In all the test cases, the numbers of duplicated gates were reduced significantly.

In the experiments, the gate sizing attack on the netlist as discussed in Section 7.3 is also simulated. The basic idea was that all false paths whose delays were in the gray region were treated as wave-pipelining paths and their delays were sized to meet (7.1)–(7.2). The results of this simulated attack are shown in Fig. 7.8, where the first bar shows the number of false paths we used to simulate the attack. The last bar shows the number of false paths that were not sized successfully. In all these simulation cases, no sizing attack succeeded. As discussed in Section 7.3, false paths may be sensitized if their delays exceed one clock period. The second bar in Fig. 7.8 shows the number of the false paths that can be sensitized when considered

as wave-pipelining paths in the attack. Obviously many of them can be sensitized so that the circuit does not work even if the sizing attack could succeed.

7.6 Summary

In this chapter, a new timing camouflage technique is proposed to secure circuit netlists against counterfeiting. Since a netlist itself does not carry all design information anymore, the difficulty of attack has increased significantly due to additional test cost and the introduced wave-pipelining false paths. The introduced method opens up a new dimension of netlist camouflage at circuit level, and it is fully compatible with other previous counterfeiting methods so that they can be combined together to strengthen netlist security.

Chapter 8

Conclusion

With the increasing process variations at nanometer technology nodes, it is a challenging task to meet timing of sequential circuits before manufacturing and chips might not work with the given clock period after manufacturing. To counter process variations, different kinds of methods from the design phase to the test phase were proposed in this thesis.

As a method to alleviate the effects of process variations, post-silicon tuning technique is widely adopted. This method can adjust the timing properties of chips after manufacturing individually. To apply this technique, two challenges need to be handled. The first challenge is where the tuning components should be inserted to balance the trade-off between area overhead and yield. The second challenge is that chips must be tested after manufacturing to determine the effect of process variations. In this thesis, a complete framework was developed to consider the yield improvement and the cost due to additional area and test holistically. According to the experimental results, the yield for all circuits can be improved significantly, while the number of post-silicon tuning components is less than 1% of the number of flip-flops. Furthermore, the test cost is reduced by more than 94%, thanks to the statistical prediction and delay test with delay alignment.

Facing the challenges from process variations, the concepts in the traditional timing paradigm should be also examined. For example, in STA, setup times, hold times and clock-to-q delays of flip-flops are characterized as constants. This simplification sacrifices circuit performance for the sake of timing simplicity. In reality, clock-to-q delays of flip-flops depend on both setup and hold times. In this thesis, a holistic timing analysis framework considering interdependency of setup and hold time using a piecewise model was proposed. With this model, the minimum clock period of circuits is evaluated using an ILP-based formulation. Experimental results demonstrate the effectiveness of the proposed model in terms of accuracy and runtime

efficiency.

In digital circuits, combinational logic blocks perform computation and sequential components, e.g., flip-flops, are used to synchronize the logic computation. However, the performance of circuits has reached a limit in the traditional timing paradigm. If we look beyond the barriers of flip-flops, the effects of process variations can be alleviated further. For example, if the flip-flop between two stages is removed, the delay imbalance is exploited automatically. To break the confines of the traditional timing paradigm, in this thesis, a new timing model was proposed, where sequential components and combinational logic gates are both considered as delay units. With this model, a timing optimization framework was proposed to allocate sequential components only at necessary locations in the circuits. Compared with the method combining sizing and retiming, the circuit performance can be improved even beyond the traditional limit.

The traditional timing model also increases the security risk for digital circuits. Since the netlist carries all the design information, attackers can reconstruct the original netlist to counterfeit chips with reverse engineering. To invalidate this assumption, in this thesis, a timing camouflage method was proposed to secure digital circuits. This technique potentially opens up a new dimension of circuit security and it is fully compatible with previous anti-counterfeiting methods.

The methods deploying post-silicon tuning and flexible timing properties of flip-flops are continuous refinements of the traditional timing. The improvement in timing performance with these methods is already approaching its limit. Therefore, a new definition of timing is also proposed in this thesis, which provides an opportunity to break the confines of the traditional timing paradigm. To enable such a new paradigm, innovative solutions from logic synthesis to physical design are required to balance the design complexity and the benefits of improvement in timing performance.

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, third edition, 1999.

[ABZ+03] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Statistical delay computation considering spatial correlation. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 271–276, 2003.

[AKGH16] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel. Reliability-aware design to suppress aging. In *Proc. Design Autom. Conf. (DAC)*, pages 12:1–12:6, 2016.

[And05] Y. Ando. Integrated circuits having post-silicon adjustment control. In *US Patent 6,957,163*, 2005.

[BCKL98] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu. Wave-pipelining: A tutorial and research survey. *IEEE Trans. VLSI Syst.*, 6(3):464–474, 1998.

[BCSS08] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(4):589–607, 2008.

[BM58] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29(2):610–611, 1958.

[BM09] A. H. Baba and S. Mitra. Testing for transistor aging. In *Proc. VLSI Test Symp. (VTS)*, pages 215–220, 2009.

[BN00] D. S. Boning and S. Nassif. Models of process variations in device and interconnect. In *Design of High Performance Microprocessor Circuits, chapter 6*. IEEE Press, 2000.

[BRPB14] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson. Stealthy dopant-level hardware trojans: Extended version. *J. Cryptographic Engineering*, 4(1):19–31, 2014.

[CBD11] D. Chen, R. Batson, and Y. Dang. *Applied Integer Programming: Modeling and Solution*. Wiley, 2011.

[CCBC06] B. Cline, K. Chopra, D. Blaauw, and Y. Cao. Analysis and modeling of CD variation for statistical static timing. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 60 – 66, 2006.

[CCW98] C.-P. Chen, C. C. Chu, and D. Wong. Fast and exact simultaneous gate and wire sizing by lagrangian relaxation. In *Proc. Design Autom. Conf. (DAC)*, pages 617–624, 1998.

[CDS+08] A. Chakraborty, K. Duraisami, A. V. Sathanur, P. Sithambaram, L. Benini, A. Macii, E. Macii, and M. Poncino. Dynamic thermal clock skew compensation using tunable delay buffers. *IEEE Trans. VLSI Syst.*, 16(6):639–649, 2008.

[Cla88] J. H. Clark. Tutorial: Computer graphics; image synthesis. chapter A Fast Algorithm for Rendering Parametric Surfaces, pages 88–93. Computer Science Press, Inc., New York, NY, USA, 1988.

[CLS12] N. Chen, B. Li, and U. Schlichtmann. Iterative timing analysis based on nonlinear and interdependent flipflop modelling. *IET Circuits, Devices & Systems*, 6(5):330–337, 2012.

[Cou10] O. Coudert. An efficient algorithm to verify generalized false paths. In *Proc. Design Autom. Conf. (DAC)*, pages 188–193, 2010.

[CS03] H. Chang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 621–625, 2003.

[CS05] H. Chang and S. S. Sapatnekar. Statistical timing analysis under spatial correlations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 24(9):1467–1482, September 2005.

[CWT11] J. Chen, S. Wang, and N. B. M. Tehranipoor. A framework for fast and accurate critical-reliability paths identification. In *IEEE North Atlantic test workshop (NATW)*, 2011.

[DBN+14] S. Dupuis, P.-S. Ba, G. D. Natale, M.-L. Flottes, and B. Rouzeyre. A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans. In *Int. On-Line Testing Symp. (IOLTS)*, pages 49–54, 2014.

[DKM93] S. Devadas, K. Keutzer, and S. Malik. Computation of floating mode delay in combinational circuits: Theory and algorithms. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 12(12):1913–1923, November 1993.

[DYG89] D. H.-C. Du, S. H. Yen, and S. Ghanta. On the general false path problem in timing analysis. In *Proc. Design Autom. Conf. (DAC)*, pages 555–560, 1989.

[Fis90] J. Fishburn. Clock skew optimization. *IEEE Trans. Comput.*, 39(7):945–951, 1990.

[FYCT13] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori. Representative critical-path selection for aging-induced delay monitoring. In *Proc. Int. Test Conf. (ITC)*, pages 1–10, 2013.

[FYCT15] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori. Aging- and variation-aware delay monitoring using representative critical path selection. *ACM Trans. Design Autom. Electr. Syst*, 20(3):1–39, 2015.

[G+] M. Galassi et al. *GNU Scientific Library Reference Manual*. Third edition.

[GHD+14] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris. Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain. *Proc. IEEE*, 102(8):1207–1228, 2014.

[GLL+15] H. Geng, J. Liu, P.-W. Luo, L.-C. Cheng, S. L. Grant, and Y. Shi. Selective body biasing for post-silicon tuning of sub-threshold designs: An adaptive filtering approach. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 34(5):713–725, 2015.

[Gur13] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2013.

[GYSH14] F. Gong, H. Yu, Y. Shi, and L. He. Variability-aware parametric yield estimation for analog/mixed-signal circuits: Concepts, algorithms, and challenges. *IEEE Design & Test*, 31(4):6–15, 2014.

[HKK+12] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov. Sensitivity-guided metaheuristics for accurate discrete gate sizing. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 233–239, 2012.

[HLCG95] H.-Y. Hsieh, W. Liu, R. K. Cavin, and C. T. Gray. Concurrent timing optimization of latch-based digital systems. In *Proc. Int. Conf. Comput. Des. (ICCD)*, pages 680–685, 1995.

[HMB08] A. P. Hurst, A. Mishchenko, and R. K. Brayton. Scalable min-register retiming under timing and initializability constraints. In *Proc. Design Autom. Conf. (DAC)*, pages 534–539, 2008.

[HPA97] K. Heragu, J. H. Patel, and V. D. Agrawal. Fast identification of untestable delay faults using implications. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 642–647, 1997.

[JB05] A. Jain and D. Blaauw. Slack borrowing in flip-flop based sequential circuits. In *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, pages 96–101, 2005.

[JC93] D. A. Joy and M. J. Ciesielski. Clock period minimization with wave pipelining. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 12(4):461–472, 1993.

[Jol02] I. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[JW07] R. A. Johnson and D. W. Wichern. *Applied multivariate statistical analysis*. Pearson Prentice Hall, 2007.

[KBW+14] V. B. Kleeberger, M. Barke, C. Werner, D. Schmitt-Landsiedel, and U. Schlichtmann. A compact model for NBTI degradation and recovery under use-profile variations and its application to aging analysis of

digital integrated circuits. *Microelectronics Reliability*, 54(6–7):1083–1089, 2014.

[KCL+17] J. Kao, C. Chao, C. Lin, N. Katta, K. Yang, and C. Wang. Post-silicon tuning in voltage control of semiconductor integrated circuits. In *US Patent 9,564,896*, 2017.

[KK17] J. Kim and T. Kim. Adjustable delay buffer allocation under useful clock skew scheduling. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 36(4):641–654, 2017.

[KKS06] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. An analytical model for negative bias temperature instability. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 493–496, 2006.

[KKS07] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. NBTI-aware synthesis of digital circuits. In *Proc. Design Autom. Conf. (DAC)*, pages 370–375, 2007.

[KL14] A. B. Kahng and H. Lee. Timing margin recovery with flexible flip-flop timing model. In *Proc. Int. Symp. Quality Electron. Des. (ISQED)*, pages 496–503, 2014.

[KLS+15] R. Kumar, B. Li, Y. Shen, U. Schlichtmann, and J. Hu. Timing verification for adaptive integrated circuits. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1587–1590, 2015.

[KME+16] N. Koppaetzky, M. Metzdorf, R. Eilers, D. Helms, and W. Nebel. RT level timing modeling for aging prediction. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 297–300, 2016.

[KS07] V. Khandelwal and A. Srivastava. Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation. In *Proc. Int. Symp. Phys. Des. (ISPD)*, pages 11–18, 2007.

[KS08] V. Khandelwal and A. Srivastava. Variability-driven formulation for simultaneous gate sizing and postsilicon tunability allocation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(4):610–620, 2008.

[KS15] S. Karapetyan and U. Schlichtmann. Integrating aging aware timing analysis into a commercial STA tool. In *Int. Symp. on VLSI Des., Aut. and Test (VLSI-DAT)*, pages 1–4, 2015.

[KSB06] S. H. Kulkarni, D. Sylvester, and D. Blaauw. A statistical framework for post-silicon tuning through body bias clustering. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 39–46, 2006.

[LBS10] D. Lorenz, M. Barke, and U. Schlichtmann. Aging analysis at gate and macro cell level. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 77–84, 2010.

[LBS12] D. Lorenz, M. Barke, and U. Schlichtmann. Efficiently analyzing the impact of aging effects on large integrated circuits. *Microelectronics Reliability*, 52(8):1546–1552, 2012.

[LBS14] D. Lorenz, M. Barke, and Schlichtmann. Monitoring of aging in integrated circuits by identifying possible critical paths. *Microelectronics Reliability*, 54(6-7):1075–1082, 2014.

[LCS09a] B. Li, N. Chen, and U. Schlichtmann. Timing model extraction for sequential circuits considering process variations. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 336–343, 2009.

[LCS+09b] B. Li, N. Chen, M. Schmidt, W. Schneider, and U. Schlichtmann. On hierarchical statistical static timing analysis. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1320–1325, 2009.

[LCS10] B. Li, N. Chen, and U. Schlichtmann. Fast statistical timing analysis of latch-controlled circuits for arbitrary clock periods. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 524–531, 2010.

[LCS11] B. Li, N. Chen, and U. Schlichtmann. Fast statistical timing analysis for circuits with post-silicon tunable clock buffers. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 111–117, 2011.

[LCS12] B. Li, N. Chen, and U. Schlichtmann. Statistical timing analysis for latch-controlled circuits with reduced iterations and graph transformations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 31(11):1670–1683, 2012.

[LCXS13] B. Li, N. Chen, Y. Xu, and U. Schlichtmann. On timing model extraction and hierarchical statistical timing analysis. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 32(3):367–380, 2013.

[LGS09] D. Lorenz, G. Georgakos, and U. Schlichtmann. Aging analysis of circuit timing considering NBTI and HCI. In *Int. On-Line Testing Symp. (IOLTS)*, pages 3–8, 2009.

[LHS18] B. Li, M. Hashimoto, and U. Schlichtmann. From process variations to reliability: A survey of timing of digital circuits in the nanometer era. *PSJ Transactions on System LSI Design Methodology*, 11:2–15, 2018.

[LKLZ12] L. Li, P. Kang, Y. Lu, and H. Zhou. An efficient algorithm for library-based cell-type selection in high-performance low-power designs. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 226–232, 2012.

[LKS+08] B. Li, C. Knoth, W. Schneider, M. Schmidt, and U. Schlichtmann. Static timing model extraction for combinational circuits. In *Int. Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 156–166, 2008.

[LN12] Z. Lak and N. Nicolici. On using on-chip clock tuning elements to address delay degradation due to circuit aging. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 31(12):1845–1856, 2012.

[LN14] Z. Lak and N. Nicolici. A novel algorithmic approach to aid post-silicon delay measurement and clock tuning. *IEEE Trans. Comput.*, 63(5):1074–1084, 2014.

[LPR+03] X. Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson, and N. Tamarapalli. High-frequency, at-speed scan testing. *IEEE Des. Test. Comput.*, 20(5):17–25, 2003.

[LS09] Q. Liu and S. S. Sapatnekar. A framework for scalable postsilicon statistical delay prediction under process variations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 28(8):1201–1212, 2009.

[LS15] B. Li and U. Schlichtmann. Statistical timing analysis and criticality computation for circuits with post-silicon clock tuning elements. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 34(11):1784–1797, 2015.

[LSM+16] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan. Provably secure camouflaging strategy for IC protection. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 28–35, 2016.

[LT15] Y.-W. Lee and N. A. Touba. Improving logic obfuscation via logic cone analysis. In *Latin-American Test Symp.*, pages 1–6, 2015.

[LZ06] C. Lin and H. Zhou. An efficient retiming algorithm under setup and hold constraints. In *Proc. Design Autom. Conf. (DAC)*, pages 945–950, 2006.

[MBPB15] S. Malik, G. T. Becker, C. Paar, and W. P. Burleson. Development of a layout-level hardware obfuscation tool. In *Comput. Society Ann. Symp. on VLSI*, pages 204–209, 2015.

[MFDN05] P. Mahoney, E. Fetzer, B. Doyle, and S. Naffziger. Clock distribution on a dual-core, multi-threaded Itanium®-family processor. In *Proc. Int. Solid-State Circuits Conf. (ISSCC)*, pages 292–293, 2005.

[MT05] M. Michael and S. Tragoudas. Function-based compact test pattern generation for path delay faults. *IEEE Trans. VLSI Syst.*, 13(8):996–1001, 2005.

[Nas01] S. R. Nassif. Modeling and analysis of manufacturing variations. In *Proc. Custom Integr. Circuits Conf. (CICC)*, pages 223–228, 2001.

[NF96] J. L. Neves and E. G. Friedman. Optimal clock skew scheduling tolerant to process variations. In *Proc. Design Autom. Conf. (DAC)*, pages 623–628, 1996.

[NK08] K. Nagaraj and S. Kundu. An automatic post silicon clock tuning system for improving system performance based on tester measurements. In *Proc. Int. Test Conf. (ITC)*, pages 1–8, 2008.

[NK09] K. Nagaraj and S. Kundu. A study on placement of post silicon clock tuning buffers for mitigating impact of process variation. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 292–295, 2009.

[NSG+06] S. Naffziger, B. Stackhouse, T. Grutkowski, D. Josephson, J. Desai, E. Alon, and M. Horowitz. The implementation of a 2-core,

multi-threaded Itanium family processor. *IEEE J. Solid-State Circuits*, 41(1):197–209, 2006.

[OBH11] M. M. Ozdal, S. Burns, and J. Hu. Gate sizing and device technology selection algorithms for high-performance industrial designs. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 724–731, 2011.

[Pat90] D. Patel. CHARMS: characterization and modeling system for accurate delay prediction of ASIC designs. In *Proc. Custom Integr. Circuits Conf. (CICC)*, pages 9.5/1–9.5/6, 1990.

[Pat03] S. Pateras. Achieving at-speed structural test. *IEEE Des. Test. Comput.*, 20(5):26–33, 2003.

[PKK+06] B. Paul, K. Kang, H. Kufluoglu, M. Alam, and K. Roy. Temporal performance degradation under NBTI: Estimation and design for improved reliability of nanoscale circuits. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 780–785, 2006.

[PM15] S. M. Plaza and I. L. Markov. Solving the third-shift problem in IC piracy with test-aware logic locking. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 34(6):961–971, 2015.

[PRU95] I. Pomeranz, S. Reddy, and P. Uppaluri. NEST: A nonenumerative test generation method for path delay faults in combinational circuits. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 14(12):1505–1515, 1995.

[PZCB+10] P. Pant, J. Zelman, G. Colon-Bonet, J. Flint, and S. Yurash. Lessons from at-speed scan deployment on an Intel® Itanium® microprocessor. In *Proc. Int. Test Conf. (ITC)*, pages 1–8, 2010.

[RKM08] J. A. Roy, F. Koushanfar, and I. L. Markov. EPIC: Ending piracy of integrated circuits. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1069–1074, 2008.

[Roe03] W. Roethig. Library characterization and modeling for 130 nm and 90 nm SoC design. In *Proc. Int. SOC Conf.*, pages 383–386, 2003.

[RPSK12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *Proc. Design Autom. Conf. (DAC)*, pages 83–89, 2012.

[RSK13] J. Rajendran, O. Sinanoglu, and R. Karri. VLSI testing based security metric for IC camouflaging. In *Proc. Int. Test Conf. (ITC)*, pages 1–4, 2013.

[RSSK13] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri. Security analysis of integrated circuit camouflaging. In *Proc. Conf. on Comput. & Commun. Security*, pages 709–720, 2013.

[SBC97] B. E. Stine, D. S. Boning, and J. E. Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *IEEE Trans. Semiconductor Manufacturing*, 10(1):24–41, 1997.

[SDT+07] E. Salman, A. Dasdan, F. Taraporevala, K. Küçükçakar, and E. G. Friedman. Exploiting setup-hold-time interdependence in static timing analysis. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 26(6):1114–1125, 2007.

[Seb77] G. Seber. *Linear Regression Analysis*. John Wiley & Sons, 1977.

[SFD+06] E. Salman, E. G. Friedman, A. Dasdan, F. Taraporevala, and K. Küçükçakar. Pessimism reduction in static timing analysis using interdependent setup and hold times. In *Proc. Int. Symp. Quality Electron. Des. (ISQED)*, pages 159–164, 2006.

[SHJL16] U. Schlichtmann, M. Hashimoto, I. H.-R. Jiang, and B. Li. Reliability, adaptability and flexibility in timing: Buy a life insurance for your circuits. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 705–711, 2016.

[SMB05] D. R. Singh, V. Manohararajah, and S. D. Brown. Incremental retiming for FPGA physical synthesis. In *Proc. Design Autom. Conf. (DAC)*, pages 433–438, 2005.

[SMO90a] K. Sakallah, T. Mudge, and O. Olukotun. checkT_c and minT_c: Timing verification and optimal clocking of synchronous digital circuits. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 552–555, 1990.

[SMO90b] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. In *ACM/IEEE Design Automation Conference (DAC)*, pages 111–117, 1990.

[Sob67] I. M. Sobol. The distribution of points in a cube and the approximate evaluation of integrals. *Comput. Math. Math. Phys.*, 7(4):86–112, 1967.

[SR07] A. Singhee and R. A. Rutenbar. From finance to flip flops: A study of fast quasi-monte carlo methods from computational finance applied to statistical circuit analysis. In *Proc. Int. Symp. Quality Electron. Des. (ISQED)*, pages 685–692, 2007.

[SS08] J. Singh and S. S. Sapatnekar. A scalable statistical static timing analyzer incorporating correlated non-gaussian and gaussian parameter variations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(1):160–173, January 2008.

[SV09] G. Seetharaman and B. Venkataramani. Automation schemes for FPGA implementation of wave-pipelined circuits. *ACM Trans. Reconf. Tech. Sys.*, 2(2):11:1–11:19, 2009.

[TBCS04] J. Tsai, D. Baik, C. C.-P. Chen, and K. K. Saluja. A yield improvement methodology using pre- and post-silicon statistical clock scheduling. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 611–618, 2004.

[TGB09] D. Tadesse, J. Grodstein, and R. I. Bahar. AutoRex: An automated post-silicon clock tuning tool. In *Proc. Int. Test Conf. (ITC)*, pages 1–10, 2009.

[TKMH04] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi. Post-fabrication clock-timing adjustment using genetic algorithms. *IEEE J. Solid-State Circuits*, 39(4):643–650, 2004.

[TRND+00] S. Tam, S. Rusu, U. Nagarji Desai, R. Kim, J. Zhang, and I. Young. Clock generation and distribution for the first IA-64 microprocessor. *IEEE J. Solid-State Circuits*, 35(11):1545–1552, 2000.

[TZC05] J. Tsai, L. Zhang, and C. C.-P. Chen. Statistical timing analysis driven post-silicon-tunable clock-tree synthesis. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 575–581, 2005.

[VCBS11] V. Veetil, K. Chopra, D. Blaauw, and D. Sylvester. Fast statistical static timing analysis using smart Monte Carlo techniques. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 30(6):852–865, 2011.

[VDP14] K. Vaidyanathan, B. P. Das, and L. T. Pileggi. Detecting reliability attacks during split fabrication using test-only BEOL stack. In *Proc. Design Autom. Conf. (DAC)*, pages 156:1–156:6, 2014.

[VHB87] B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Comput. Graph.*, 21(4):103–110, 1987.

[VRK+04] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proc. Design Autom. Conf. (DAC)*, pages 331–336, 2004.

[WZB17] H.-L. Wang, M. Zhang, and P. A. Beerel. Retiming of two-phase latch-based resilient circuits. In *Proc. Design Autom. Conf. (DAC)*, pages 1–6, 2017.

[WZXS13] T. Wang, C. Zhang, J. Xiong, and Y. Shi. Eagle-eye: A near-optimal statistical framework for noise sensor placement. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 437–443, 2013.

[XD10] L. Xie and A. Davoodi. Representative path selection for post-silicon timing prediction under variability. In *Proc. Design Autom. Conf. (DAC)*, pages 386–391, 2010.

[XS17] Y. Xie and A. Srivastava. Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction. In *Proc. Design Autom. Conf. (DAC)*, pages 1–6, 2017.

[XSZV09] J. Xiong, Y. Shi, V. Zolotov, and C. Visweswariah. Statistical multilayer process space coverage for at-speed test. In *Proc. Design Autom. Conf. (DAC)*, pages 340–345, 2009.

[XZH07] J. Xiong, V. Zolotov, and L. He. Robust extraction of spatial correlation. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 26(4):619–631, 2007.

[YTJ15] Y.-M. Yang, K. H. Tam, and I. H.-R. Jiang. Criticality-dependency-aware timing characterization and analysis. In *Proc. Design Autom. Conf. (DAC)*, pages 167:1–167:6, 2015.

[YW99] J. Yen and L. Wang. Simplifying fuzzy rule-based models using orthogonal transformation methods. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 29(1):13–24, 1999.

[YX10] F. Yuan and Q. Xu. On timing-independent false path identification. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 532–535, 2010.

[YYX11] R. Ye, F. Yuan, and Q. Xu. Online clock skew tuning for timing speculation. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 442–447, 2011.

[YZLS17] B. Yigit, G. L. Zhang, B. Li, and U. Schlichtmann. Application of machine learning methods in post-silicon yield improvement. In *Proc. Int. System-on-Chip Conf. (SOCC)*, pages 243–248, 2017.

[ZLHS18] G. L. Zhang, B. Li, M. Hashimoto, and U. Schlichtmann. VirtualSync: Timing optimization by sychronizing logic waves with sequential and combinational components as delay units. In *Proc. Design Autom. Conf. (DAC)*, 2018.

[ZLL+18] G. L. Zhang, B. Li, J. Liu, Y. Shi, and U. Schlichtmann. Design-phase buffer allocation for post-silicon clock binning by iterative learning. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 37(2):392–405, 2018.

[ZLS16a] G. L. Zhang, B. Li, and U. Schlichtmann. Sampling-based buffer insertion for post-silicon yield improvement under process variability. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1457–1460, 2016.

[ZLS16b] G. L. Zhang, B. Li, and U. Schlichtmann. EffiTest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers. In *Proc. Design Autom. Conf. (DAC)*, pages 60:1–60:6, 2016.

[ZLS16c] G. L. Zhang, B. Li, and U. Schlichtmann. PieceTimer: A holistic timing analysis framework considering setup/hold time interdependency using a piecewise model. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 100:1–100:8, 2016.

[ZLS+18] G. L. Zhang, B. Li, Y. Shi, J. Hu, and U. Schlichtmann. EffiTest2: Efficient delay test and prediction for post-silicon clock skew configuration under process varaitions. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2018.

[ZLY+18] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann. TimingCamouflage: Improving circuit security against counterfeiting by unconventional timing. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, 2018.

[ZMT+17] O. Zografos, A. D. Meester, E. Testa, M. Soeken, P. E. Gaillardon, G. D. Micheli, L. Amarù, P. Raghavan, F. Catthoor, and R. Lauwereins. Wave pipelining for majority-based beyond-CMOS technologies. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1306–1311, 2017.