Habilitationsschrift

A Perspective of Timing in the Nanometer Era

vorgelegt dem Fakultätsrat der Fakultät für Elektrotechnik und Informationstechnik – Technische Universität München

von

Dr.-Ing. Bing Li

Vorsitz des Fachmentorats:		Prof. Dr.-Ing. Ulf Schlichtmann
Fachmentoren:	1.	Prof. Dr.-Ing. Norbert Wehn
	2.	Prof. Dr. Krishnendu Chakrabarty

Acknowledgments

Since 2014, I have been working on my Habilitation, which leads to this final thesis. During this period, Professor Ulf Schlichtmann gave me all the help that is possible, not only in research but also on a much more general scale. Without his insightful advice and reliable support this thesis could not happen.

For my Habilitation, Professor Norbert Wehn, Professor Doris Schmitt-Landsiedel, and Professor Krishnendu Chakrabarty kindly supported me as Fachmentoren. I am very grateful to them for their generous help.

During the Habilitation period, I also had chances to work with other researchers. Professor Tsung-Yi Ho and I worked together to establish the research team on microfluidic biochips in the EDA institute at TUM. Supported by his expertise and with countless discussions with him, this team has grown quickly and successfully. I have also taken much time from Professor Krishnendu Chakrabarty on various occasions. It was really enjoyment to talk with him with ideas flowing from discussions. Meanwhile, several professors have kindly collaborated with me. Some of them are Professor Yiyu Shi, Professor Bei Yu, Professor David Z. Pan, Professor Jiang Hu, Professor Paul Pop, Professor Bhargab B. Bhattacharya, Professor Robert Wille, Professor Hailong Yao, Professor Masanori Hashimoto, Professor Iris Hui-Ru Jiang and Professor Mark Po-Hung Lin. I appreciate their help and support very much.

Inside the EDA institute, I collaborated closely with Dr. Tsun-Ming Tseng, Li Zhang, Chunfeng Liu and Ying Zhu. I thank them sincerely for their essential contribution to my Habilitation. In addition, I would like to thank Dr. Helmut Gräb and Dr. Daniel Müller-Gritschneder for their long-term support. I also thank Susanne Werner, Hans Ranke and Tobias Baur, whom I bothered very often in these years. The other colleagues in the EDA institute have created a nice atmosphere for research and it was a pleasure to work with them.

Munich, January 2018
Bing Li

Contents

1 Introduction

Integrated Circuits (IC) have become an essential part of human activities. They are the key drivers of many innovations and economic progress, thanks to the rapid advances in design and manufacturing technology of the IC industry in the past several decades [Int]. From mobile phones to supercomputers and from automotive electronics to artificial intelligence, these small chips are connecting people, automating their everyday life, and opening new views to the world with their supreme computing power.

Microscopically viewed, IC chips are composed of tiny transistors, each of which has a size of about 10 nanometers. Within an area of 1 cm^2, more than 2.5×10^9 transistors can be manufactured [App]. Logic gates are constructed from these transistors and connected to implement various computing functions. To coordinate computing activities of logic gates, a regular clock signal is distributed inside a chip to initiate the computing activities synchronously. To guarantee that all logic blocks finish their computation in time, they need to meet special timing constraints in relation to the period of the clock signal. Accordingly, the performance of a digital circuit is defined by its clock frequency, representing how many times new data can be processed by the circuit in a second.

For decades, the IC industry followed the Dennard scaling concept [DGY$^+$74]. Roughly speaking, every new process generation would result in about 30% reduction in propagation delay, and thus enable about 30% higher clock frequencies. From the introduction of the first microprocessor in 1971, the Intel 4004 running at 740kHz, to the 2004 introduction of the Intel Pentium 4 in 90nm technology, running at 3.4 GHz, the clock frequency increased about 4.600 times, or roughly 29% annually.

Unfortunately, this major driving force of the IC industry has essentially come to a stop. Although feature sizes are still continuously being reduced to smaller geometries to integrate an increasing number of transistors for complex integrated circuits, voltage scaling has stopped due to power limitations. Since leakage power concerns prevent a further downscaling of threshold voltages V_{th}, dynamic power consumption has become a big challenge, especially in designs with a high clock frequency, thus hindering a further increase of clock frequency.

As the feature size of IC manufacturing technology is scaled down into deep submicron region, further undesirable side-effects have also appeared. A major concern has been the process variations in manufacturing as it has become more difficult to accurately control important physical parameters such as gate length, oxide thickness, line edge roughness, or doping profiles at ever-smaller dimensions [Nas01]. Consequently, the variations of key parameters also increased, even with the σ values reaching around 15% of the nominal values. Some of these variations may have a low correlation [SBC97], so that they could not be addressed by the traditional best-/worst-case timing analysis and signoff strategies, which verify the ability of a circuit to work at a given clock frequency with the best cases and the worst cases derived from the manufacturing process. Because of these developments, the pressure to squeeze more performance out of process technologies by innovative design approaches and methodologies has increased significantly.

Similar to process variations, another effect that has become practically relevant since about 10 years is aging of transistors as well as the ensuing reliability issues. When currents flow through transistors, stress is accumulated on them. Gradually, chips may become slower compared with the fresh state right after manufacturing. Aging covers several effects impacting devices in nanometer manufacturing nodes, most prominently Hot Carrier Injection (HCI), Negative Bias Temperature Instability (NBTI), Electromigration (EM), Time-Dependent Dielectric Breakdown (TDDB), as well as Positive BTI (PBTI), which is also increasingly becoming a consideration. For example, NBTI and HCI result in an increase of V_{th} or decrease of I_{on}, respectively, causing a loss of performance, up to 20% over time. These aging effects have been around for a long time, and their increasing relative magnitude in today's manufacturing technologies cannot be ignored anymore; Otherwise, the exceedingly large guardband reserved for aging in the traditional design flow would cancel any further advances in manufacturing technologies and essentially put the IC industry into a stalling state.

Since the industry embraced manufacturing technologies with the feature size below 14/10 nm, it has become obvious that timing analysis is no longer a solved problem. The challenges to enable a new timing paradigm and to realize innovative solutions to manage design complexity and to break performance stagnancy in modern ICs require a thorough examination of the existing design philosophy and work flows. In this thesis, accordingly, the challenges of timing in the nanometer era are analyzed. Furthermore, a new perspective of timing will be discussed to open new doors for optimizing timing performance of digital circuits. In addition, it provides a new technique to enhance the security of netlists against counterfeiting by invalidating the traditional single-period clocking scheme.

The rest of this thesis is organized as follows. In Chapter 2, the state of the art of techniques dealing with timing challenges is reviewed. These include analysis methods such as statistical timing analysis (SSTA) and aging evaluation, as well as active tuning methods such as post-silicon clock skew tuning. In Chapters 3–5, details of a post-silicon clock tuning technique are discussed to demonstrate the demands of research efforts from statistical analysis to post-silicon test due to the evolution of the timing paradigm. In Chapter 6, a new perspective to view the traditional timing definition will be provided. This new view may potentially pave new ways for optimizing timing performance of digital circuits, and it also provides a new dimension in the hardware security domain to counter netlist counterfeiting. Thereafter, a discussion of applying the relatively mature timing paradigm of digital circuits to the development of an emerging technology, flow-based microfluidic biochips, will be summarized in Chapter 7. Finally, conclusions are drawn in Chapter 8.

2 State of the Art of Timing

2.1 Digital Circuits and Timing Constraints

Digital circuits contain two types of gates. Logic gates, e.g., AND, OR and NOT gates, implement the function of the circuit. Sequential gates, e.g., flip-flops and latches, on the contrary, do not make any contribution to the logic function directly. Instead, they are used to synchronize intermediate steps of the computation. An example of a digital circuit is shown in Figure 2.1.

A sequential gate, henceforth with flip-flop as example, is only activated at a given moment to store the data from its input. Except this exact moment, the stored data and the output of the flip-flop do not change, no matter what happens at its input. Consequently, flip-flops sitting on combinational paths in a sequential circuit partition the circuit into separate logic/combinational functional blocks. The data at the outputs of a combinational block are latched into the flip-flops connected to them at a given moment. Thereafter, any changes at the outputs of the combinational block and thus the inputs of the flip-flops do not affect the data stored inside the flip-flops, until the next latching moment comes. Since the outputs of flip-flops are connected to the inputs of combinational blocks, the starting time of logic computation can thus be synchronized.

To coordinate the activities of logic blocks, a clock signal is generated and distributed to all flip-flops. The flip-flops are activated at a clock edge, e.g., the rising clock edge henceforth. Since the clock signal reaches all flip-flops at the same time, the latching activities of flip-flops trigger signal transitions at the inputs of logic blocks simultaneously. Because at each rising clock edge new computations are initiated, the

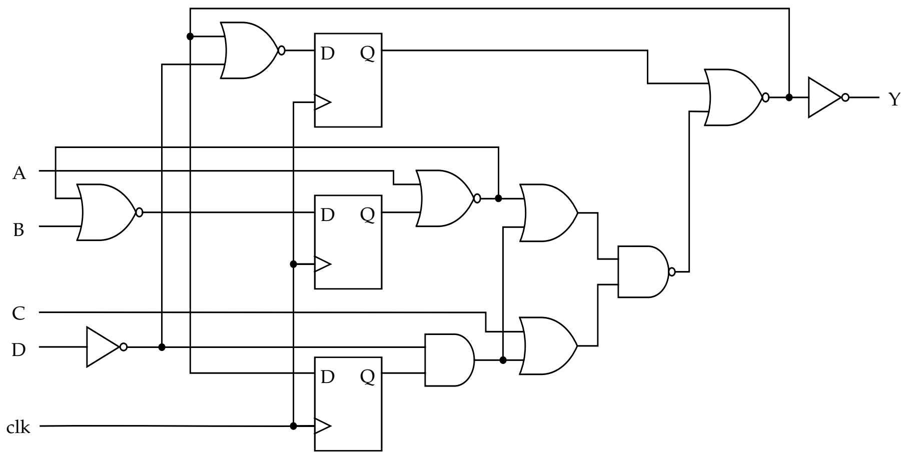


Figure 2.1: Structure of s27 from the ISCAS89 [BBK89] benchmark set.

performance of a circuit is thus often represented by its clock frequency, indicating how many input data can be processed in a second by the circuit.

Since the logic blocks in a digital circuit are activated by the clock signal periodically, the logic computation inside of a logic block must finish its execution before the data at its outputs are latched into the flip-flops of the next stage. The time difference between two consecutive activations of a logic block is the clock period T.

A flip-flop also needs some time to store the data correctly, the data at its input must be stable within a small time window before the rising clock edge, i.e., setup time t_{su} before the rising clock edge. Similarly, the data must stay stable hold time t_h after the rising clock edge. As illustrated in Figure 2.2, the data must be stable in a

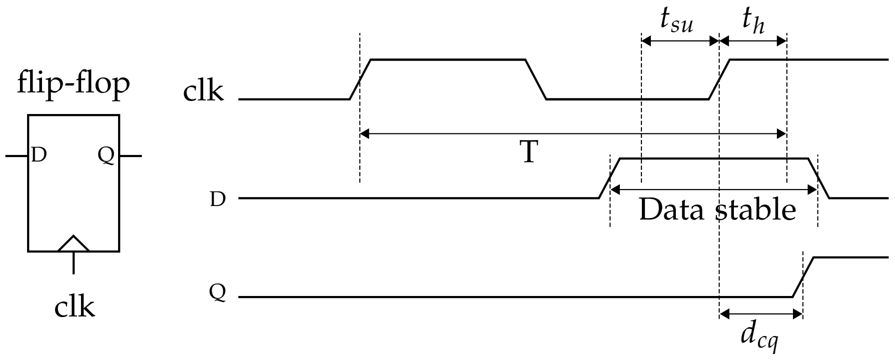


Figure 2.2: Setup time (t_{su}), hold time (t_h) and clock-to-q delay (d_{cq}) of a flip-flop.

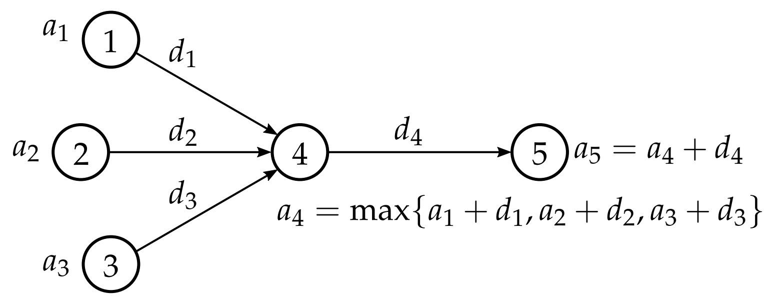


Figure 2.3: *max* and *sum* operations. a_1–a_5 are arrival times and d_1–d_4 represent delays of logic gates and interconnects.

window surrounding the rising clock edge to guarantee a correct latching behavior of the flip-flop. Since the latest time a data signal arrives at a flip-flop is determined by the longest combinational path in the logic blocks, and the earliest change is caused by the shortest combinational path in the logic blocks, the timing constraints considering all logic blocks in a circuit can be written as

$$\max_{p\in P}\{d_{cq} + d_p + t_{su}\} \leq T \tag{2.1}$$

$$\min_{p\in P}\{d_{cq} + d_p\} \geq t_h \tag{2.2}$$

where p is a path with delay d_p from the set P representing all the paths in the combinational blocks. d_{cq} is the delay of the flip-flop. In reality, flip-flops may have different setup times and hold times depending on their structures and dimensions. In (2.1) and (2.2), these properties are assumed as the same for all flip-flops to simplify the discussion.

To verify the timing constraints (2.1) and (2.2), the delays of combinational paths need to be calculated. However, it is timing-consuming and unnecessary to enumerate all combinational paths. Instead, the delay information can be merged by propagating only the maximum or minimum delay information forward in the circuit. To trace the maximum delay of combinational paths to verify the setup time constraint (2.1), two atomic operations, *max* and *sum*, are applied recursively in timing analysis. This concept is illustrated in Figure 2.3, where the arrival times of signals at nodes are denoted as a_1–a_5 and the delays of logic gates and interconnects are denoted as d_1–d_4. When a combinational component is passed, its delay is added to the arrival time. When multiple arrival times converge at a node, the maximum of them is calculated and propagated further. Since logic functions are not considered, the task of timing analysis becomes merely to efficiently traverse a graph representing the circuit structure and delays.

To improve circuit performance, the easiest way is to boost its clock frequency, or, equivalently, shorten the clock period. In 2004 the International Technology Roadmap for Semiconductors (ITRS) has predicted that, driven by advances in manufacturing technology, the clock frequency would increase by 21% every year, jumping from about 2.9 GHz in 2002 to 33.4 GHz in 2015. This optimistic estimation has been, however, adjusted in 2013, predicting that the clock frequency may reach 5.9 GHz in 2015, leading to only 4% annual growth rate. This slowdown of clock frequency increase is caused by several factors, including process variations, power consumption, noise, and reliability and aging issues.

2.2 Timing with Process Variations

In a digital circuit, the latest time a signal becomes stable is after it travels through the combinational path with the largest delay, called the longest path henceforth. The delay of this path is equal to the sum of all the delays of logic gates and interconnects on the path. These delays are, however, not fixed values in reality due to the imperfection in manufacturing processes. In fact, it is not possible to deliver logic gates or interconnects with the exact design parameters, e.g., physical dimensions, from the manufacturing process. Instead, the parameters of logic gates and interconnects vary in different chips of the same design, a phenomenon called process variations [Nas01], so that some of them have large delays but others may have small delays.

Process variations have been existing since the beginning of the semiconductor industry. They did not significantly affect the design flow in the past because the variations were relatively small, so that they could be dealt with by allocating performance margins easily. In advanced technology nodes, these variations have become

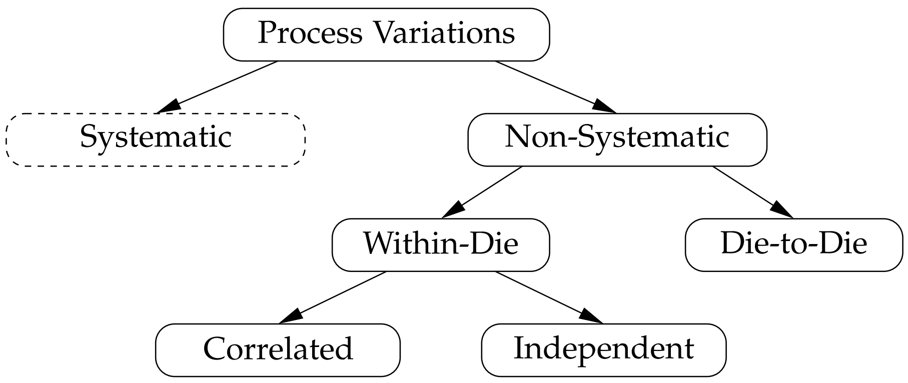


Figure 2.4: Variation Classification [BCSS08].

relatively large, e.g., reaching a one-sigma variation of 15.7% in 70nm manufacturing node [Nas01], so that they cannot easily be compensated anymore by allocating excessive performance margins. In addition, local variations have become more relevant compared to global variations, making the performances of devices exhibit a low correlation. Therefore the traditional worst-case-based design methodology cannot properly handle them anymore, because in the worst-case-based methodology the cases that all devices become slow or fast are checked, which may be too pessimistic in view of local variations.

Process variations cannot be determined before real manufacturing happens and in different manufactured chips their final effects are different. During the design phase, timing analysis is still required because designers need to evaluate how the manufactured chips perform after experiencing process variations. Consequently, the performances of logic gates and interconnects need to be modeled statistically as random variables according to the manufacturing data from foundries, leading to a boom of research on statistical timing analysis (SSTA) [BCSS08].

Process variations can be classified into different categories, as shown in Figure 2.4. Systematic variations, e.g., the randomness in interconnect metal thickness, are caused in part by design characteristics such as the density of interconnects on a metal layer, so that they can be extracted and modeled according to the design information even before manufacturing. The variations of this type can be directly incorporated into post-OPC (Optical Proximity Correction) timing analysis to improve the accuracy [YCS05]. Non-systematic variations, however, cannot be determined before manufacturing, since they are the results of the inaccuracy in process control during manufacturing. Consequently, they need to be modeled as random variables when the circuit is designed. Such variations include those in doping density and layout-independent metal thickness. Non-systematic variations can be split further into die-to-die variations (interdie variations) and within-die variations (intradie variations) [SBC97]. Die-to-die variations take effect equally on all devices and interconnects on a die. For example, the chips at the center of a wafer are usually faster than chips located closer to the boundary of a wafer. Within-die variations have different effects on devices and interconnects on a die, leading to deviation between device parameters inside a chip after manufacturing. If the within-die variations of parameters exhibit no correlation, they become a purely random effect, such as the random distortion caused by lens during photolithography and the purely random variations in doping.

In timing analysis, the delays of the longest and the shortest paths should be calculated to verify (2.1) and (2.2). In the case that all gate delays are fixed values, this is not a challenging task. When process variations are considered, however, the computational complexity increases extraordinarily, because the *max* and *sum* operations become statistical. Moreover, the results of these operations should maintain the correlation to the other random variables, so that the same formulas of the *max* and *sum* operations can be applied recursively. For example, the result a_4 of *max* and *sum* operations at node 4 in Figure 2.3 should be represented in the same form as d_4 so that the *sum* operation can be applied to calculate a_5 similarly.

Several models have been proposed to express the correlation between process parameters. In the quadtree model [ABZ+03b, ABZ03a], different layers are used to represent the random components shared by devices in areas of different scales. This quadtree model, however, does not model the local correlation uniformly. To overcome this problem, the correlation model in [CS03] partitions the die area into a uniform grid with n cells, and a random variable is assigned for each grid cell. To simplify statistical computations in timing analysis, these variables are expressed as linear combinations of independent random variables decomposed by methods such as principal component analysis (PCA) [Jol02]. This correlation model can deal with any correlation between process parameters rather accurately, and it is extended in [CZV+08] to partition the die area using hexagonal grid cells.

The correlation models discussed above are all first-order and only sufficient to deal with the dependency between Gaussian random variables. To include dependency information of a higher order, methods like independent component analysis [HKO01] have been applied, as in [SS06, SS08].

With random variables representing process parameters expressed as linear combinations of independent random variables, the delay of a gate can also be expressed similarly, such as in the canonical delay model in [VRK+04]. In statistical timing analysis, while the sum of two statistical delay variables in the canonical form can be computed relatively easily, the computation of the maximum of them is, however, very challenging. In this computation, not only the mean and variance of the maximum should be calculated, but also the correlation between the maximum and the other delay variables should be maintained to apply the computations of *sum* and *max* in the same form recursively. In [VRK+04], the maximum of two canonical expressions is approximated in the same form using the concept of tightness probability, which requires time-consuming computations compared with the maximum computation of two constants in traditional worst-/best-case timing analysis.

The discussion on statistical timing analysis has exposed the difference between statistical timing analysis (SSTA) and traditional static timing analysis (STA). In STA, the maximum of two arrival times is only a simple comparison of two numbers. But in SSTA, it involves a lot more computation not only for the mean and the variance but also for maintaining the correlation between random variables during arrival time propagation. Furthermore, the linear timing analysis methods above assume that gate delays are approximated as Gaussian random variables. This limitation has been relaxed in [ZSL+05, ZCH+05, FLZ07] by representing timing properties as quadratic functions of independent Gaussian random variables. Moreover, gate delays can also been expressed as linear combinations of non-Gaussian variables, as in the method in [SS06, SS08], while the method in [CZNV05] proposes a general framework to incorporate linear and nonlinear combinations of Gaussian and non-

Gaussian random variables.

The methods above are all block-based since at each component only the maximum or minimum of the variables is propagated. Path-based methods have also been explored for statistical timing analysis, e.g., in [ABZ+02, OB04]. These methods first identify a set of combinational paths from a circuit and analyze their performances individually. Afterwards, the performance of the whole circuit is computed from the performances of these paths. Path-based methods can only process a given set of combinational paths, which, however, are still not easy to identify accurately from the circuit [LLCP08]. Based on path-based statistical timing analysis, common paths have been studied in [ZF02, Vis07] and noises have been considered in [ENH09, ESN+10] to reduce timing pessimism.

2.3 Hierarchical Statistical Timing Analysis

To counter the immense computational effort required for timing analysis considering process variations, hierarchical statistical timing analysis has been investigated to accelerate system-level timing analysis.

Hierarchical timing analysis splits timing analysis into two steps. At first, timing properties inside submodules are extracted individually. Afterwards, the timing models of all the submodules in a circuit are combined together to perform timing analysis of the whole design.

A timing model contains interface timing information of a module, which is usually the delay information of: 1) direct paths from inputs to outputs, type A in Figure 2.5; 2) paths from inputs that can reach a flip-flop or a latch inside the module, type B; 3) paths from an internal flip-flop or a latch to an output, type C; 4) paths between flip-flops or latches, type D. The statistical path delays of type D carry correlation information between modules and are required in hierarchical statistical timing analysis for correlation reconstruction, as to be discussed later in this section.

For static timing analysis without consideration of process variations, timing model extraction has been explored in [DMS+02] to extract black-box models, which only

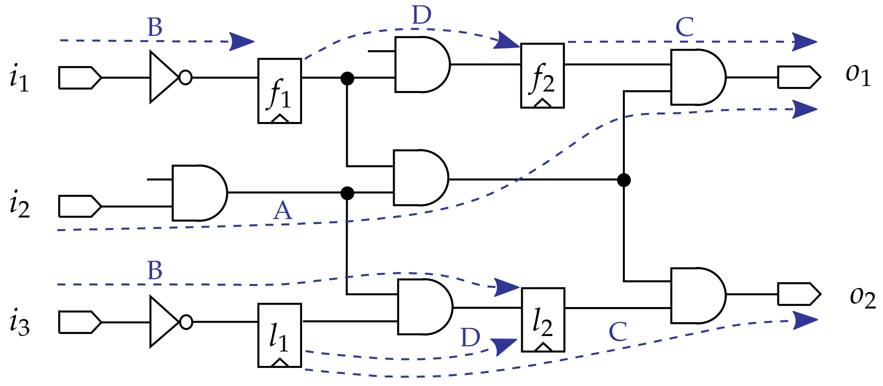


Figure 2.5: Timing paths in timing model extraction.

contain interface timing information. The methods in [KM97, MKB02, ZZH+06, LKS+08] apply graph transformation operations to merge nodes and edges representing the timing information of the original circuit, leading to gray-box timing models because timing information inside the modules is exposed. For sequential circuits, Interface Logic Model (ILM) in [DMS+02] keeps all the combinational paths between input ports to the first-level flip-flops and from the last-level flip-flops to the output ports. The Extracted Timing Model (ETM) [DMS+02], however, collapses all such paths to reduce the size of timing models, sacrificing the flexibility of keeping the original parasitics information associated with the original logic gates and interconnects. For latch-based circuits, all latches can be retained in the timing model as in [MKB02], or the depth of transparency from the input ports to internal latches and from the internal latches to the output ports are assumed as given [VPMS97], so that the number of latches potentially traveled transparently can be determined.

When process variations are considered, timing model extraction methods for combinational circuits and sequential circuits with flip-flops in [KM97, MKB02, DMS+02, ZZH+06, LKS+08] can be applied similarly, but with the maximum and sum computations replaced by the corresponding statistical versions. For sequential circuits with latches, the depth of transparency becomes statistical [LCS10, LCS12]. Therefore, timing models should be extracted with respect to the probabilities of transparency depths [LCS09a, LCXS13], which are affected by the statistical path delays in the original circuit.

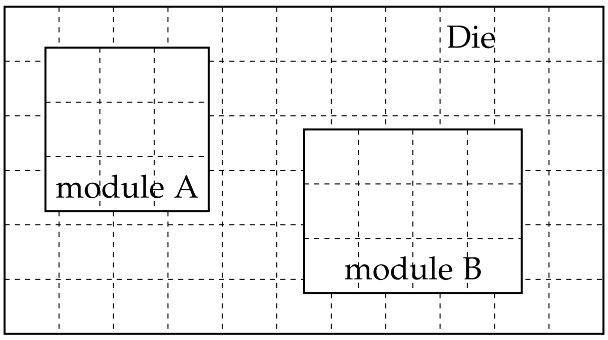


Figure 2.6: Grid partition of the chip in hierarchical statistical timing analysis.

When integrating extracted statistical timing models into timing analysis of the whole circuit, a special challenge should be met when process variations are considered. At the top level, the die area occupied by each module is partitioned to model the correlation between variations when generating timing models, such as using the uniform grid in [CS03]. When the extracted models are placed together, the grids inside them may not be aligned, as illustrated in Figure 2.6. Therefore, the relation between these grids should be established. In [GVTG08, GVTG09], this problem is addressed by linear transformation between the coefficients of independent random variables. But there might be no solution from this method if the transformation matrix is not selected properly. This method is improved in [LCS+09b, LCXS13] by using the coefficient matrices of submodules and the top grid directly.

2.4 Timing with Setup-Hold Time Interdependency

Process variations may affect path delays in a sequential circuit statistically. If the delay of a path in a manufactured chip exceeds the clock period minus the setup time of a flip-flop, a timing violation is assumed to happen. In reality, however, the data may still be latched into the flip-flop correctly even in view of a violation of the setup time constraint (2.1), but the clock-to-q delay may increase significantly, leading to a delay increase of the combinational paths of the next stage. If those paths are short, the increased delay does not cause timing violation in the next stage. As shown in Figure 2.7(a), a flip-flop can work with different setup-hold time combinations with

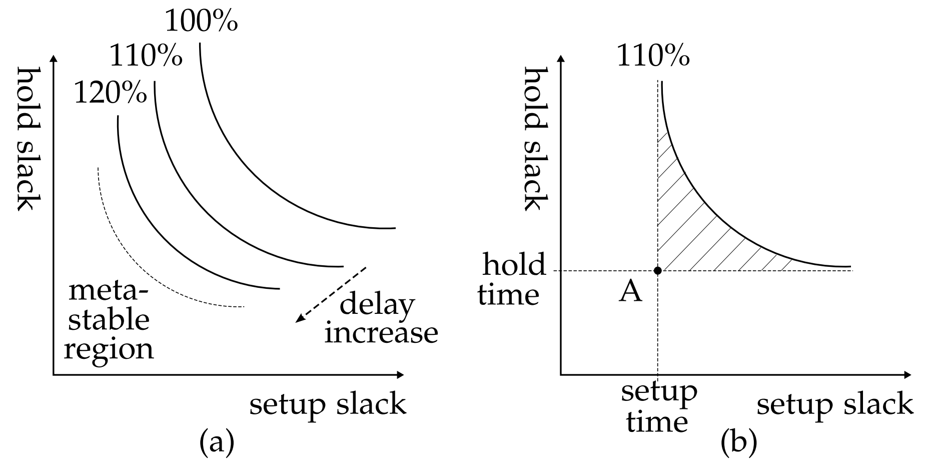


Figure 2.7: Delay curves of a flip-flop. (a) Curves of setup/hold slack combinations with respect to different constant clock-to-q delays. Setup and hold slacks are the time differences between the signal switches and the clock edge. (b) Characterization point of setup time and hold time in traditional STA.

different clock-to-q delays. In the traditional definition, the setup time and hold time are simply approximated as the point A in Figure 2.7(b), losing the flexibility of the flip-flop completely.

The setup-hold interdependency has be investigated in [SFD$^+$06, SDT$^+$07] to exploit the compensation between setup time and hold time combinations with respect to a given clock-to-q delay. In addition, a method based on Euler-Newton tracing is introduced in [SR07, SR08] to characterize the delay curves of flip-flops. These methods, however, do not consider the relation between clock-to-q and setup/hold times, leading to a limited performance improvement. To remove this limitation the method in [JB05] uses a quadratic programming model to calculate the optimal clock period directly, but it is incapable of solving the high-order programming problem for large circuits. To simplify the three dimensional model, the method in [CLS12] applies an analytic function and calculates the minimum clock period by iterations. In addition, the method in [KL14] approximates the three dimensional delay surface using linear planes. In calculating the minimum clock period of a circuit, this method, however, splits the problem into two-dimensional problems. Furthermore, the method in [YTJ15] proposes an efficient algorithm to capture timing violations in a circuit very efficiently, but only the relation between clock-to-q delay and setup slack is considered in this method. Most recently, the method in [ZLS16b] models the delay surface using piecewise polygons and performs the timing analysis using integer linear programming together with reduction techniques. In addition, process variations are considered together with setup-hold interdependency for flip-flop models in [HAP08], but how to incorporate the generated statistical models in statistical timing analysis is still open.

2.5 Aging

With the diminishing dimensions of devices, a new challenge that has been around for a long time, aging of ICs, has also started to attract increasing attention since about 10 years. The term *aging* covers a number of effects impacting devices in nanometer manufacturing nodes, most prominently Hot Carrier Injection (HCI), Negative Bias Temperature Instability (NBTI), Electromigration (EM), Time Dependent Dielectric Breakdown (TDDB), and Positive BTI (PBTI).

With aging the performance of devices deteriorates by up to 20% over time. The analysis of aging effects is challenging, since the amount of aging depends on a number of factors, among which are V_{dd}, temperature, frequency and also the amount of switching or the specific voltage level a transistor experiences. Accordingly, both the specific structure of a circuit and its real usage contribute to the results of aging. Aging analysis is further complicated by the fact that NBTI degradation partly recedes once a stress condition is removed, known as *recovery effect*.

Aging analysis traditionally has focused on the potential effect of aging on individual transistors. As a result of such transistor analysis, an overall safety margin was added into the timing signoff to guarantee the correct functionality of the chips. This coarse-grained approach is increasingly less feasibly because, on the one hand, aging has become more pronounced, and on the other hand, the timing budget is becoming increasingly tight, thus not being able to tolerate a large timing margin anymore. Consequently, it has become desirable to analyze the aging of a given circuit specifically to provide more fine-grained information.

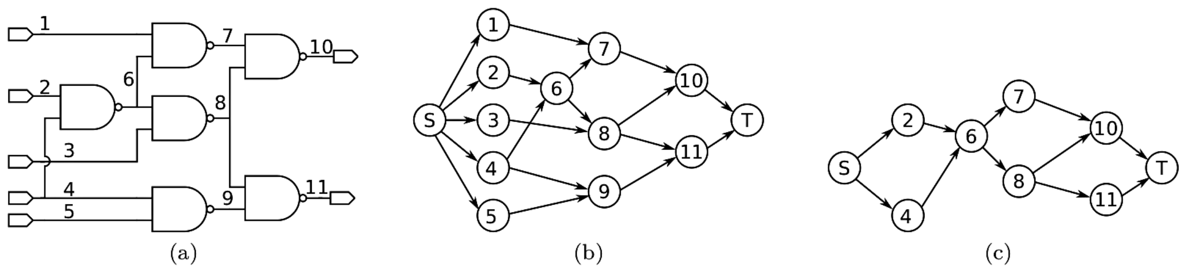


Figure 2.8: Aging graph pruning [LBS10] of the ISCAS85 circuit c17 depicted in (a). Timing graph of this circuit and the reduced graph are shown in (b) and (c), respectively.

To calibrate aging effects, transistor-level simulation is required. Though traditional transistor aging models are accurate, they are too slow for analyzing large ICs. To solve this problem, research has been undertaken to analyze HCI and NBTI and the factors influencing them, and to develop timing models and algorithms for aging analysis on gate level, leading to a speedup of aging analysis by orders of magnitude [BM09, CWT11, KKS06, KKS07, PKK+06, KBW+14, AKGH16, KME+16]. The AgeGate model [LGS09, LBS10, LBS12] is probably still the state of the art in gate-level aging analysis today. This aging model can handle both HCI and NBTI. It is independent of the current use profile, defined by V_{dd} and temperature T over the lifetime of the IC. It also incorporates the aging effect of each transistor in a gate individually and the aging of output slope, both of which are required for accurate timing analysis. The approach of AgeGate builds on the canonical delay model [VRK+04] so that it can be integrated into standard industrial STA-based timing signoff flows smoothly [KS15].

The AgeGate model was extended later to take module-level aging analysis into account, speeding up aging analysis further, while maintaining a good accuracy [LBS14]. This approach extracts a timing graph of a module from the original gate-level netlist. Thereafter, the graph is pruned significantly. For example, any path that can never become critical under aging and process variations is not considered in the analysis. The concept of this pruning is illustrated in Figure 2.8. Empirical results have confirmed that the number of timing paths can be reduced by up to four orders of magnitude or even more for aging analysis. The reduced timing graphs of modules can not only be used to accelerate aging analysis, but also to identify critical paths to be monitored online, hence enabling a systematic design approach by combining periodic monitoring of a circuit during its operation and online tuning.

Similar to statistical timing analysis, the characterization of aging still aims to capture more detailed delay information of logic gates for timing analysis in the design flow. During the design phase, aging effect still have to be considered statistically since both process variations and aging affect manufactured chips individually and produce different critical paths in different chips after manufacturing. To counter the aging effects actively, post-silicon tuning techniques can also be applied to adjust the timing properties of individual chips. Techniques in this category include body bias tuning [KSB06, GLL+15], voltage control [And05, KCL+17, KLS+15] as well as clock tuning [NSG+06, TZC05, LN14].

2.6 Circuit Tuning

While process variations must be modeled as random variables during the design phase, their effects become fixed in individual chips after manufacturing. These chips have different performance because process variations affect them differently. Similarly, aging is also a temporal effect of individual chips. To counter these effects actively, manufactured chips may be tuned accordingly to change the clock skews or body bias with respect to the fixed effects of process variations and usage profiles.

In a sequential circuit, the clock signal reaches all flip-flops at the same moment. With this strict assumption, the performance of the circuit is determined by the longest path in the circuit, no matter how fast the other paths are. To balance the performance between different paths, clock edges can be tuned toward fast paths so that slow paths receive more timing budget to finish their signal propagation to improve the overall yield [LCS11, LS15].

There are various structures of post-silicon clock tuning [TRND+00,TKMH04,MFDN05, NSG+06]. For example, the delay buffer in [NSG+06] is illustrated in Figure 2.9,

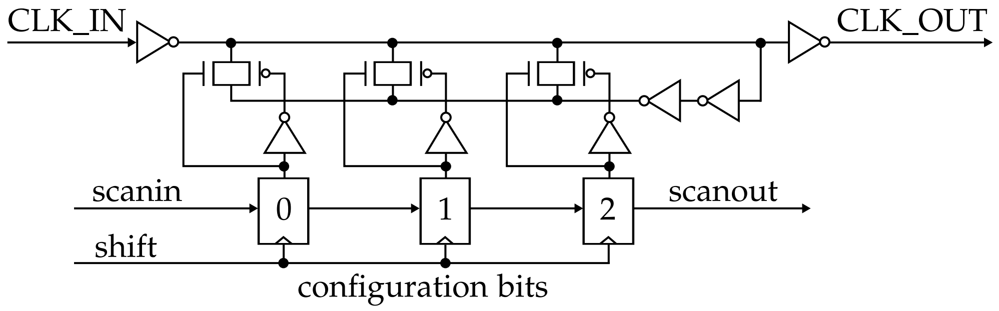


Figure 2.9: Post-silicon delay tunable buffer in [NSG+06].

where the delay between the clock input CLK_IN and the output CLK_OUT is controlled by the values of three registers configured through the test access port (TAP).

To perform post-silicon clock tuning, delay buffers should be inserted into the circuit during the design phase. Since these buffers occupy die area, a tradeoff should be made to balance the yield improvement and the enlarged area. When deciding how many buffers to insert into a circuit, the method in [TBCS04] presents a technique based on clock scheduling to balance the skews resulting from process variations. In [KK17] this problem is solved using a graph-based algorithm. In [TZC05] several algorithms are proposed to insert buffers into the clock tree to guarantee a given yield and minimize the total area taken by these tunable buffers. Furthermore, the method in [ZLS16c, ZLL+18] solves this problem with a sampling-based technique to recognize a limited number of locations to insert tunable buffers for yield improvement. The computational complexity of this method is reduced using machine learning in [YZLS17]. In [KS07], this problem is solved together with gate sizing. In [NK09], the placement of tunable buffers is explored and a considerable yield improvement has been observed.

After manufacturing, the tunable buffers should be configured to adjust the clock skews in the manufactured chips. This configuration requires that delay information should be captured for these chips. In [LN14], this post-silicon configuration is performed by searching a configuration tree together with graph pruning and buffer grouping. In [NK08,TGB09] path delays are measured individually in manufactured chips and delay buffers are tuned accordingly. Since this individual measurement is not efficient, statistical prediction and aligned delay test have been introduced in [ZLS16a].

To minimize design and operating margins, post-silicon tuning can be applied for online adjustment to counter aging effects, where each chip adapts its operating conditions such as supply voltage and body bias dynamically. This adaptation can deal with dynamic environmental fluctuation and aging as well as process variations. For adaptive voltage scaling, two strategies are widely used. The first one is based on error detection and recovery, such as proposed in [DRS+06, YYX11, LN12]. The second one is based on error prediction and prevention, e.g., canary flip-flop [SK07,FHMO12], slack monitor [BCH+15] and timing error predictive flip-flop (TEP-FF) [IMK+13]. In these methods, sensors are used to detect/predict timing errors, and supply voltages or clock skews are adjusted according to the sensor outputs. For adaptive circuit optimization, a stochastic error rate estimation method is introduced in [IMK+13, IMHO15], by modeling adaptive clock control under dynamic delay variation as a continuous-time Markov process.

2.7 Summary

Statistical timing analysis and post-silicon tuning are still incremental steps in the framework of traditional sequential design, where logic computation is finished within one clock period. Consequently, only the timing constraints (2.1) and (2.2) need to be satisfied, either by constraining the delays of logic paths or by tuning clock skews to allow a smaller clock period T.

When clock tuning components are considered, statistical timing analysis becomes more complex, because clock skews are determined with respect to the fixed effects of process variations after manufacturing. This statistical analysis will be discussed in Chapter 3. To determine the effective locations to insert tunable buffers, yield optimization with a limited set of buffers needs to be addressed, as to be discussed in Chapter 4. The configuration of post-silicon tunable buffers relies on circuit delays after manufacturing. In Chapter 5, post-silicon test and buffer configuration will be explained in detail. Chapters 3–5 have been published as [LS15, ZLL+18, ZLS+].

3 Statistical Timing Analysis and Criticality Computation for Circuits with Post-Silicon Clock Tuning Elements

Statistical Timing Analysis and Criticality Computation for Circuits With Post-Silicon Clock Tuning Elements

Bing Li and Ulf Schlichtmann, *Member, IEEE*

***Abstract*—Post-silicon clock tuning elements are widely used in high-performance designs to mitigate the effects of process variations and aging. Located on clock paths to flip-flops, these tuning elements can be configured through the scan chain so that clock skews to these flip-flops can be adjusted after manufacturing. Owing to the delay compensation across consecutive register stages enabled by the clock tuning elements, higher yield and enhanced robustness can be achieved. These benefits are, nonetheless, attained by increasing die area due to the inserted clock tuning elements. For balancing performance improvement and area cost, an efficient timing analysis algorithm is needed to evaluate the performance of such a circuit. So far this evaluation is only possible by Monte Carlo simulation which is very time-consuming. In this paper, we propose an alternative method using graph transformation, which computes a parametric minimum clock period and is more than 10^4 times faster than Monte Carlo simulation while maintaining a good accuracy. This method also identifies the gates that are critical to circuit performance, so that a fast analysis-optimization flow becomes possible.**

***Index Terms*—Criticality computation, post-silicon clock tuning, statistical timing analysis, yield.**

I. Introduction

PROCESS variations have become relatively larger in recent technology nodes. This trend makes the traditional worst-case timing analysis too pessimistic, leading to expensive overdesign and depriving designers of the valuable information of performance and yield. Modeling timing characteristics of a circuit more accurately, statistical static timing analysis (SSTA) has gained much attention in the research community in recent years [2]. This method represents process variations with random variables directly and computes the complete performance-yield curve, either in a parametric form or described by statistical properties, e.g., moments of different orders. Consequently, the yield of the circuit at any given clock period can be evaluated easily.

According to the assumption of the distributions of process variations, the method used to model gate delays, and the statistical operations in timing propagation, statistical timing algorithms can be roughly classified into several groups. First-order methods [3]–[5] use the canonical linear form [5] to represent gate delays and arrival times so that the recursive computations in timing analysis can be simplified but at the expense of accuracy. To improve modeling and propagation accuracy, quadratic methods are proposed in [6]–[10], using second-order polynomials to approximate gate delays and arrival times. Moreover, other methods, such as [11]–[13], can handle non-Gaussian delays and arrival times during timing propagation.

The research on statistical timing analysis focuses mainly on delay representation and arrival time propagation in combinational circuits and the resulting methods are implicitly applicable to circuits with flip-flops. In high-performance circuits, flip-flops with post-silicon clock tuning elements [14], [15] have also been deployed to counter process variations and improve circuit robustness. The tunable or programmable elements are inserted into the clock network to flip-flops that are relevant to critical paths. After manufacturing, the delay values of these elements are adjusted through the test access port (TAP) to assign critical paths more timing budget by shifting the clock edges toward the stages with smaller delays. By allowing delay compensation across consecutive register stages, chips that might have failed to meet the timing specification can be revitalized. Therefore, with clock tuning elements the circuit can achieve a higher yield than without them.

Several methods have already been proposed for statistical timing analysis and optimization of circuits with clock tuning elements. In [16], a clock scheduling method is developed and clock tuning elements are selectively inserted to balance the skews due to the process variations. Further, in [17] algorithms are proposed to minimize the total area of these clock tuning elements or to minimize the number of them in the circuit. In these methods, the yield of the circuit is computed using Monte Carlo simulation which consumes much runtime. In [18], the yield loss due to process variations and the total cost of clock tuning elements are formulated together for gate sizing. The resulting optimization problem is solved using a stochastic cutting-plane method with an STA scheme based on Monte Carlo simulation. This method still converges slowly due to the long runtime of yield evaluation. Moreover, the placement of clock tuning elements is investigated in [19] and a considerable benefit is observed when the clock tree is designed using the proposed tuning system.

The methods discussed above are applied as presilicon optimization or post-silicon adjustment before shipping the chips to customers. Recently, further advances have been made

Manuscript received June 20, 2014; revised December 11, 2014; accepted April 15, 2015. Date of publication May 12, 2015; date of current version October 16, 2015. This work was supported in part by the German Research Foundation through the Transregional Collaborative Research Centre, Invasive Computing (SFB/TR 89). A preliminary version of this paper was published in the proceeding of IEEE/ACM International Conference on Computer-Aided Design, 2011 [1]. This paper was recommended by Associate Editor P. Gupta.

The authors are with the Institute for Electronic Design Automation, Technische Universität München, Munich 80333, Germany (e-mail: b.li@tum.de; ulf.schlichtmann@tum.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2015.2432143

to apply the clock tuning elements on-line to improve the life-time performance [20]–[23]. The method in [21] adjusts the clock skews during runtime according to the occurrence of timing errors to achieve much better performance in timing-speculative circuits. The method in [23] explores the insertion of clock tuning elements and in-system configuration to reduce performance degradation due to aging. In addition, Lak and Nicolici [22] proposed an efficient post-silicon tuning method for each individual chip by searching a configuration tree combined with graph pruning. Moreover, the method in [20] applies clock tuning elements to compensate dynamic delay variations induced by temperature.

The research on statistical timing analysis and optimization has shown the advantage of using post-silicon clock tuning elements in high-performance designs. However, two issues still have not been addressed. The first is the need for a fast statistical timing analysis method, with which the runtime of the methods above can be reduced. Currently, these methods use Monte Carlo simulation to compute the yield of the circuit considering process variations and are thus time-consuming.

The second issue is how to identify a set of critical gates for optimization of such a circuit containing post-silicon clock tuning elements. In statistical timing analysis the probability of a gate affecting the circuit performance is called criticality and many methods have been proposed to describe and compute the criticalities of gates efficiently. In [5], the concept of criticality is first explored without considering correlation. In [24], the sensitivities of gate and path delays to the circuit delay are computed. In [25], the criticalities are computed using a cutset-based method combining with a binary tree partition. Furthermore, in [26] a fast criticality computation method is proposed with incremental yield gradients. Additionally, in [27] a clustering-based pruning is proposed to speed up the computation and improve the accuracy of criticalities. For ranking critical gates tiered criticalities are calculated to provide an order of statistical delays in [28], and for computing criticalities incrementally the reversible statistical max/min operation is investigated in [29]. These methods, though accurate and fast, do not consider post-silicon clock tuning elements, which allow the compensation of path delays across register stages but make the criticality computation more complicated.

In this paper, we propose a fast algorithm to evaluate the circuit performance in the presence of post-silicon clock tuning elements, so that yields of the circuit at different given clock periods can be calculated easily. Additionally, we investigate the criticalities of gate delays in the context that timing compensation is allowed across register boundaries. The registers considered in this paper are all edge-triggered flip-flops. The main contributions of this paper are as follows.

1) The proposed method computes a parametric minimum clock period for the circuit with post-silicon clock tuning elements. The statistical properties of this minimum clock period, such as mean and variance, are directly available so that the yield of the circuit at any given clock period can be evaluated very fast. Since the computed circuit performance is in a parametric form, it can be easily integrated into other optimization methods that are built upon statistical timing analysis.
2) The proposed method is much faster, more than 10^4 times, than Monte Carlo simulation, by handling the path delay compensation across registers with a loop evaluation algorithm based on graph transformation.
3) The criticalities of gate delays considering post-silicon clock tuning elements are defined and computed for circuit optimization. The proposed method can capture the critical gates within very short runtime, therefore, enabling a fast analysis-sizing cycle.

The rest of this paper is organized as follows. In Section II, we give an overview of the timing constraints considering delay tuning elements in circuits with edge-triggered flip-flops. These difference constraints are represented by using a constraint graph in the formulation. In Section III, the basic idea of calculating statistical minimum clock period from a constraint graph is defined and graph transformations are applied to extract it. Based on this result, the sequential criticality across flip-flop stages is defined to capture critical gates considering delay tuning elements. In Section IV, several implementation techniques are explained to accelerate the proposed algorithm. We discuss experimental results in Section V and conclude this paper in Section VI.

II. Background and Problem Formulation

In this section, we describe the timing constraints of digital circuits with post-silicon clock tuning elements. These tuning elements can be configured after manufacturing to change the clock skews to flip-flops, according to path delays affected by process variations. Environmental variations are not considered in this method. In our formulation, all registers are edge-triggered flip-flops. Though transparent latches can also take advantage of these tuning elements, problem formulation becomes more complex in this case, because a timing constraint between a pair of latches contains more variables, so that the method discussed in this paper becomes inapplicable.

A. Timing Constraints for Circuits With Post-Silicon Clock Tuning Elements

In high-performance digital designs, clock tuning elements are directly inserted into the clock paths to flip-flops. The intentional clock skews from these tuning elements are adjusted after manufacturing to counter process variations and aging effects [21]–[23]. These clock tuning elements may have various implementations and characteristics. The method in [30] produces tuning elements with precise adjustable delays shorter than 30 ps by way of voltage-controlled driver strength. The implementation in [15] uses a delay line and is capable of generating delays with 1 ps resolution. The de-skew buffers in [14] are built with CMOS inverters and arrays of passive loads, with 170 ps delay range and 8.5 ps step size. In [31], the delay element is made with a controlled contention circuit to provide a delay range around 140 ps with eight steps. The delays of these tuning elements can be adjusted via TAP after manufacturing [14].

Fig. 1 illustrates an example of two flip-flops with clock tuning elements, where the clock signals clk_i and clk_j are not aligned anymore after passing the tuning elements, so that the timing budget allowed for the combinational circuit between flip-flops i and j can be regulated by the configurable delays x_i and x_j. 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. Therefore, a signal change at the output of flip-flop i starts propagation at time x_i and reaches flip-flop j at the latest at time $x_i + \overline{d}_{ij}$, where $\overline{d}_{ij}$ is the maximum delay of the combinational circuit between i and j. To guarantee the setup time constraint of j, this signal at the input of j should

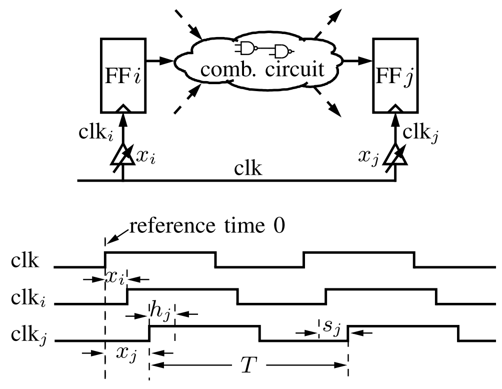


Fig. 1. Flip-flops with tuning elements. Clock signals clk$_i$ and clk$_j$ reach flip-flops unaligned due to the configurable delays x_i and x_j. s_j is the setup time of flip-flop j and h_j is the hold time of j. T is the clock period.

be stable s_j time before the rising clock edge of j, where s_j is the setup time of j. Therefore, the timing constraint for the combinational circuit can be written as

$$x_i + \overline{d}_{ij} \leq x_j + T - s_j \tag{1}$$

where T is the clock period. Let, $\overline{w}_{ij} = \overline{d}_{ij} + s_j$. Then (1) is equivalent to

$$x_j - x_i \geq \overline{w}_{ij} - T. \tag{2}$$

Similar to setup time constraints, hold time constraints should also be included to guarantee that the signal propagated from the clock edge of flip-flop i does not affect the latching function of flip-flop j in the same cycle. Therefore, the tunable delays should satisfy the following constraint:

$$x_i + \underline{d}_{ij} \geq x_j + h_j \tag{3}$$

where $\underline{d}_{ij}$ is the minimum combinational delay between i and j; h_j is the hold time of j. Let, $\underline{w}_{ij} = h_j - \underline{d}_{ij}$. Then, we can write (3) as

$$x_i - x_j \geq \underline{w}_{ij}. \tag{4}$$

In addition to the constraints (2) and (4), the maximum configurable delay or the tuning range is also limited due to area and power consumption [14], [15], [30], [31]. For a tuning element with delay x_i, its range is constrained as

$$0 \leq x_i \leq r_i \tag{5}$$

where r_i is a constant representing the largest delay that the tuning element can add to the clock signal.

To guarantee the proper function of a circuit with clock tuning elements, the constraints (2), (4), and (5) are created for each pair of flip-flops between which there is a combinational path. Compared with the timing constraints of digital circuits without clock tuning elements, the constraints (2) and (4) contain additional variables x_i and x_j which establish the relation between the delays across register stages. In the above timing constraints, $\overline{d}_{ij}$ and $\underline{d}_{ij}$ can be calculated using the traditional breadth- or depth-first propagation algorithms with statistical max and sum operations. However, in each constraint (2) or (4), the configurable delays x_i and x_j of the tuning elements can only be determined after manufacturing. In traditional static timing analysis, there are no such post-silicon tunable parameters. Therefore, by leaving out x_i and x_j, the constraint (2) simply defines a lower bound for the clock period T, so that the minimum clock period of the circuit can be computed easily by calculating the maximum of all the lower bounds. But this method does not work any longer in the presence of the configurable delays x_i and x_j. In addition, $\overline{w}_{ij}$ and $\underline{w}_{ij}$ are random variables, thus excluding the direct application of linear programming solvers to find the minimum clock period constrained by (2), (4), and (5), as in the classic clock skew optimization problem [32]. In addition, Fishburn [33] provided a method to deal with discrete delay settings by adapting Bellman–Ford algorithm. This method works well with deterministic delays, but is still unable to solve the differential constraints when path delays become random variables due to process variations.

B. Graph Representation of Difference Constraints

To establish a fast statistical timing algorithm for circuits with post-silicon clock tuning elements, we represent the constraints (2), (4), and (5) using a directed graph and calculate the statistical minimum clock period by graph transformation. In each of the above constraints, there are not more than two variables x_i or x_j. Therefore, all these constraints together form a difference constraint problem [34], from which a constraint graph can be constructed. The constraint graph contains a node for each flip-flop, corresponding to a variable x_i or x_j. If a constraint (2) exists for flip-flops i and j, meaning that there is at least one combinational path from i to j, a setup edge is created from node i to j in the graph, with the weight $\overline{w}_{ij} - T$. Similarly, for the hold time constraint corresponding to (4) a hold edge is created from node j to i with the weight $\underline{w}_{ij}$. To incorporate (5) into the constraint graph, a root node is created and shared by all the range constraints. The constraint itself can be split into $x_i \geq 0$ and $-x_i \geq -r_i$. For the former a range edge is created from the root node to node i with the weight 0; for the latter a range edge from node i to the root node with the weight $-r_i$.

In Fig. 2(b), the constraint graph of s27 from ISCAS89 benchmarks is illustrated as example. The nodes 1–3 represent the three flip-flops in Fig. 2(a). Edges between these nodes represent timing constraints (2) and (4), where only the weights of edges created from (2) contain $-T$. Node 0 is the shared root node. Edges to and from the root node correspond to the range constraints (5). In this example, we assume every flip-flop has a clock tuning element to show the basic idea. In reality, only those flip-flops that are relevant to critical paths are assigned as tuning elements during circuit optimization [17]. In constructing the constraint graph, if a flip-flop has no tuning element, the corresponding variable x_i or x_j in (2) and (4) is fixed to 0 since no delay adjustment is possible. Consequently, the constraints (2) and (4) degrade into range constraints with only one variable, or additional yield constraints if both variables are set to 0.

The edge weights in the constraint graph contain random variables $\overline{w}_{ij}$ and $\underline{w}_{ij}$ defined in (2) and (4), respectively, so that linear programming cannot be applied directly as in [32]. Another method to compute the minimum clock period is derived from the equivalence of difference constraints and non-positive loops in the constraint graph [34]. For a given clock period T, this equivalence specifies that a set of valid values for x_i and x_j that meets (2), (4), and (5) exists if and only if the sum of the edge weights from any loop in the constraint graph is not greater than 0. For example, the loop formed by the two setup edges and one hold edge across loop $2 \rightarrow 3 \rightarrow 1 \rightarrow 2$

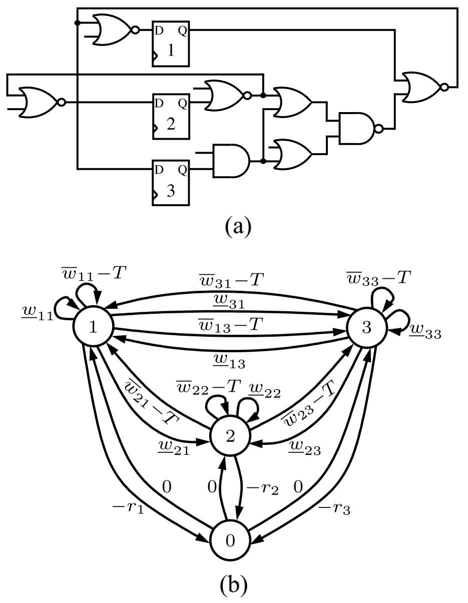


Fig. 2. Construction of constraint graph. Nodes 1–3 correspond to the flip-flops with clock tuning elements. Edges between these nodes represent timing constraints (2) and (4). Node 0 is the shared root node. Edges to and from the root node represent the range constraints (5). (a) s27 from ISCAS89 benchmarks without IO ports. (b) Constraint graph of s27.

in Fig. 2(b) should meet

$$(\overline{w}_{23} - T) + (\overline{w}_{31} - T) + \underline{w}_{21} \leq 0. \tag{6}$$

In the constraint graph, if all loops meet the conditions similar to (6), the given clock period T can be definitely achieved by configuring the clock tuning elements. For a brief explanation, assume that all loops are nonpositive. Under this condition, the Bellman–Ford algorithm [34] can always find the largest distances from the root node to all the other nodes in the graph. These maximum distances together form a valid configuration to all the tunable elements. For example, for a setup time constraint described by (2), we create an edge from node i to j with the weight $\overline{w}_{ij} - T$ in constructing the constraint graph. The maximum distance from the root node to j calculated by the Bellman–Ford algorithm is no smaller than the distance from the root node to i plus the edge weight; otherwise the distance to j is not largest from the root node. That is to say, the maximum distances can naturally meet the constraint (2). This reasoning is valid for all the setup time constraints (2) and the hold time constraints (4). In addition, the range constraints (5) are guaranteed similarly by the range edges constructed from $x_i \geq 0$ and $-x_i \geq -r_i$ described earlier. Therefore, all the maximum distances together form a valid solution for the tunable delays.

Inversely, if there is a loop across which the sum of all edge weights is positive, the given clock period T is infeasible. For example, if (6) is violated, we can deduce from (2) and (4) a contradiction as

$$0 = (x_3 - x_2) + (x_1 - x_3) + (x_2 - x_1) \geq \tag{7}$$

$$(\overline{w}_{23} - T) + (\overline{w}_{31} - T) + \left(\underline{w}_{21}\right) > 0. \tag{8}$$

In this case, the Bellman–Ford algorithm does not converge after sufficient iterations. Therefore, a valid solution for the system of difference constraints (2), (4), and (5) requires that the loops in the constraint graph must be nonpositive. Here, we have only explained the basic idea about the equivalence between the nonpositive loop condition and the existence of a solution for the difference constraint system. A detailed proof can be found in [34].

The above discussion shows that the nonpositive loop condition can be used to verify whether a given clock period T is feasible. This concept has been applied in [35] for clock schedule optimization, in [36] to determine the skew range in static timing analysis, in [37] for clock skew synthesis, and in [38] for statistical timing verification of circuits using level-sensitive latches. In the following sections, we will explain a fast method based on parametric graph transformation and pruning techniques to calculate the statistical minimum clock period without enumerating all the loops in the graph.

III. Statistical Timing Analysis and Criticality Computation

In this section, we first explain the concept of computing the statistical minimum clock period from the constraint graph of a circuit using parametric graph transformations in Sections III-A and III-B. The basic idea of using these transformations was first introduced in [39] for statistical timing analysis of latch-controlled circuits. The challenges in applying these transformations to large graphs will be addressed by several techniques in Section IV. More importantly, we define the sequential criticality considering timing compensation between sequential stages in Section III-C. This concept is a new layer of criticality above the criticality definitions in [5], [25], and [27].

In the following discussion, we assume that each flip-flop has an individual clock tuning element, for simplification. The case that tuning elements are shared by multiple flip-flops can be modeled easily using the cluster method in [23].

A. *Defining T_m Using Constraint Graph*

In Section II, we have discussed that the system of difference constraints formed by (2), (4), and (5) is equivalent to the condition that the constraint graph has no positive loops. In case of static timing analysis, this condition can be verified using the Bellman–Ford algorithm for a given clock period. When process variations are considered, however, timing analysis becomes more complex because delays are represented by random variables. In the proposed method, we compute the statistical minimum clock period T_m for such a circuit by graph transformation while keeping the clock period T as an unknown variable. The resulting T_m is a random variable from which the yield at any given clock period can be calculated easily.

For convenience, we write the edge weights in the constraint graph into a general form $w_{ij} - k_{ij}T$. For setup edges specified by (2), $k_{ij} = 1$ and $w_{ij} = \overline{w}_{ij}$; for hold edges specified by (4), $k_{ij} = 0$ and $w_{ij} = \underline{w}_{ji}$; for range edges (5), $k_{ij} = 0$ and $w_{ij} = 0$ or $w_{ij} = -r_i$. For hold edges, we have switched the indexes so that the edge with a weight in the general form always has the direction from i to j. Assuming that the clock period is still unknown, we can express the nonpositive loop condition exemplified by (6) using edge weights in the general form as

$$w_l = \sum_{i,j}\left(w_{ij} - k_{ij}T\right) = \sum_{i,j} w_{ij} - \sum_{i,j} k_{ij}T \leq 0 \tag{9}$$

where l is the index of the loop, w_l is the weight of the loop, and the sum computation is applied over all edges on the loop.

According to the definition of $w_{ij} - k_{ij}T$, k_{ij} is equal to 0 or 1. For loop l, if the sum of the coefficients $\sum_{i,j} k_{ij}$ is zero, there is no setup edge on the loop. In this case, (9) is a condition which only affects the yield of the circuit due to hold time constraints, but it has no effect on the minimum clock period. If $\sum_{i,j} k_{ij} > 0$, the loop l contains setup edges and the constraint (9) specifies a lower bound for the clock period T as

$$T_l = \sum_{i,j} w_{ij} \Big/ \sum_{i,j} k_{ij} \le T \tag{10}$$

where T_l is called loop constraint in the following discussion.

The constraint (10) from a loop creates a lower bound for the feasible clock period. If all loops in the constraint graph are considered, the minimum clock period T_m for the circuit can be computed as

$$T_m = \max_{l \in L} T_l \tag{11}$$

where L is the set of all loops in the constraint graph except those that only include hold or range edges. The loops of the latter type meet the condition $\sum_{i,j} k_{ij} = 0$ and are denoted by the set $\bar{L}$. Thereafter, the constraints (9) from these loops can be merged as

$$\max_{l \in \bar{L}} \left\{ \sum_{i,j} w_{ij} \right\} \le 0. \tag{12}$$

Because this variable only affects the yield of the circuit and its computation is similar to (11), we will focus only on the discussion of computing the minimum clock period T_m in the following.

To compute the minimum clock period T_m from all loops using (11) and the constraint in (12) directly requires that T_l from every loop should be extracted. Obviously it is impractical to enumerate all these loops due to their prohibitive number in a large graph. In the following, we will discuss three basic graph transformation operations to unroll loops gradually and extract the loop constraints in the form of T_l at the same time.

B. Computing T_m Using Graph Transformation

Instead of enumerating all loops in the constraint graph, we compute T_m in (11) using an iterative method based on graph transformation to capture the loop constraints T_l in (10). Three basic graph transformation operations are used in the proposed method: self-loop removal, serial merge, and parallel merge. Here, a self-loop is formed by only one edge starting and ending at the same node, while a general loop may contain a chain of edges. Assume that the weight of the edge that forms a self-loop at node i is $w_{ii} - k_{ii}T$. The nonpositive loop constraint explained in the preceding section can be written as

$$w_{ii} - k_{ii}T \le 0 \tag{13}$$

and thus be transformed to

$$w_{ii}/k_{ii} \le T, \quad k_{ii} > 0 \tag{14}$$

$$w_{ii} \le 0, \quad k_{ii} = 0. \tag{15}$$

For example, the self-loop with the weight $\overline{w}_{11} - T$ at node 1 in Fig. 2(b) requires that $T \ge \overline{w}_{11}$ while the self-loop with the weight $\underline{w}_{11}$ only constrains the yield. The self-loops are removed from the constraint graph right away once they appear and the corresponding constraints are merged in (11) and (12), respectively.

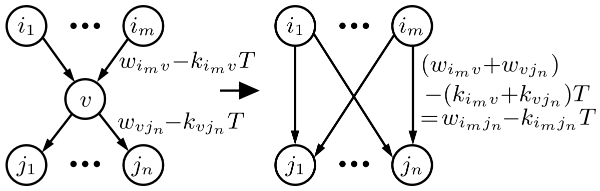


Fig. 3. Serial merge operation. Direct edges are created after node v is removed. New edge weights are calculated as the sums of the former weights.

The self-loops are removed and need not to be considered again in capturing the constraints from other loops, because the extracted lower bounds in the form of (14) or (15) guarantee that the weight of the edge in a self-loop is not positive, so that the weights of other loops including this edge cannot increase compared with the case that the self-loop is not included. For example, in Fig. 2(b) there is a loop formed by the setup edge from node 3 to 1, then the self-loop with weight $\overline{w}_{11} - T$, then the hold edge from 1 to 2 and the setup edge from 2 to 3. If the clock period T is not less than $\overline{w}_{11}$, the timing constraint is guaranteed implicitly if the constraint from the loop without the edge forming the self-loop can be met.

After removing self-loops, a typical structure in the constraint graph is illustrated on the left side of Fig. 3. The serial merge operation removes node v from this structure and creates direct edges between each predecessor and each successor of node v. The weight of a new edge is equal to the sum of the weights of the edges from which the new edge is constructed. Therefore, the constraint from any loop that passes through node v is not affected. The new weight is also in the general form $w_{ij} - k_{ij}T$, so that the serial merge operation can be applied iteratively.

In the serial merge operation, a predecessor node and a successor node may be the same. For example, if we apply the serial merge operation to node 1 in Fig. 2(b) after all original self-loops are removed, a new self-loop is constructed at node 2 due to the setup edge from node 2 to 1 and the hold edge from 1 to 2, with edge weight $\overline{w}_{21} - T + \underline{w}_{21}$. This self-loop is immediately removed from the graph during the graph transformation and the timing constraint $T \ge \overline{w}_{21} + \underline{w}_{21}$ is captured and merged to T_m using (11). Because the serial merge operation creates direct edges between predecessor and successor nodes iteratively, the newly created and then removed self-loops actually contain the sum of the edge weights from the original constraint graph. In other words, the original loops in the graph are collapsed by graph transformation and their loop constraints are captured by self-loops eventually.

After each serial merge operation the number of nodes in the constraint graph is reduced by 1, but many new edges may be created in the graph, because in the worst case $m \times n$ new edges could be created for the node v with m predecessors and n successors. Usually, this is far larger than the number of the removed $m + n$ edges, thus, causing the edge number in the graph to increase very quickly during the transformation. Actually applying the serial merge operation repeatedly to capture all loop constraints without further pruning is equivalent to enumerating all the loops in the graph directly, which is very time-consuming for a large constraint graph. To solve this problem, we will apply various pruning techniques to be discussed in Section IV to reduce the number of edges in each iteration.

Besides the serial merge operation, we apply the parallel merge operation to reduce the number of edges further.

Algorithm 1: Computing the Minimum Clock Period T_m From the Constraint Graph Using Graph Transformation

```
L1  G: the constraint graph created from (2), (4) and (5);
L2  $v$, $v_i$, $v_j$: nodes involved in graph transformation operations;
L3  N: the number of nodes in the original constraint graph.
L4  foreach node $v$ in the constraint graph G do
L5      remove_self_loops ($v$);
L6      update_$T_m$ ();
L7  end
L8  for $k$=1 to $N$ do
L9      prune_edges (G);
L10     $v$=select_node (G);
L11     serial_merge ($v$);
L12     foreach predecessor node $v_i$ of $v$ do
L13         remove_self_loops ($v_i$);
L14         update_$T_m$ ();
L15         foreach successor node $v_j$ of $v$ do
L16             if there exist parallel edges connecting nodes $v_i$ and
                $v_j$ then
L17                 parallel_merge ($v_i$, $v_j$);
L18             end
L19         end
L20     end
L21 end
```

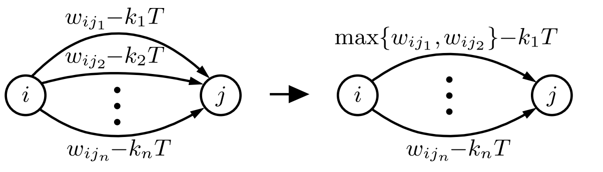


Fig. 4. Parallel merge operation. Edges connecting the same nodes and having the same coefficients of T are merged. The new edge weight is computed as the maximum of the former edge weights. In this case, $-k_1$ is equal to $-k_2$.

In the constraint graph, if there are multiple edges between two nodes, these edges are called parallel edges. For example, between nodes 1 and 3 in Fig. 2(b) there are two sets of parallel edges. Additionally, parallel edges may also be created by the serial merge operation. For example, if node 1 in Fig. 2(b) is removed by the serial merge operation, new parallel edges appear between nodes 2 and 3. These new edges may also have the same coefficient k_{ij} of T, so that they can be merged to reduce the number of edges, by the operation called parallel merge and illustrated in Fig. 4. If the coefficients $-k_1$ and $-k_2$ of T in the weights of the two parallel edges in Fig. 4 are equal, the first two edges can be merged into one edge, whose weight is computed as the maximum of two former edge weights. If these two coefficients are not equal, we cannot merge the two parallel edges because T is kept as an unknown variable to catch its lower bounds constrained by loops so that we cannot compare the weights of the two edges directly. Similar to the serial merge operation, parallel merge operation does not affect the constraints from the weights of loops that pass through i and j, because the maximum weight of the loops through the merged edges is maintained by the new edge.

Applying the graph operations discussed above iteratively, we can capture all loop constraints and compute the minimum clock period T_m by Algorithm 1. At the beginning, the algorithm removes all original self-loops in L4–L7 to reduce the number of edges. In each of the following iterations, a node is removed using the serial merge operation denoted by serial_merge (v) where v is the node selected by the function select_node (G) from the constraint graph G. The function remove_self_loops (v) removes edges that form self-loops at node v. These self-loops may exist in the original constraint graph or are results from merging the edges on the loops by iterative serial merge operations. For each of these self-loops the constraint in the form of (10) is computed and T_m is updated by the function update_T_m () using (11). After each serial operation, only checking the nodes that are the predecessors of the removed node at L12 is enough, since a self-loop can only be formed when the predecessor and successor of the removed node are the same. After each serial merge operation, parallel merge operations are applied to compress edges in the graph further. The algorithm needs to run N iterations, where N is the number of the nodes in the original constraint graph. The constraints from all the loops are captured when eventually all of the nodes are deleted.

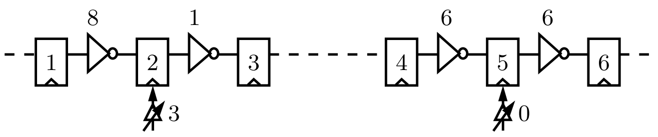


Fig. 5. Critical paths in the presence of clock tuning elements whose tuning ranges are up to 3. The paths with the delays 6 instead of the path with the delay 8 are critical.

Algorithm 1 only shows the basic concept of applying graph transformation operations. In spite of the self-loop removal and parallel merge operations, the number of edges in the constraint graph may still increase very fast. To solve this problem, we apply pruning techniques denoted by the function prune_edges (G) and discuss them as well as the function select_node (G) used to select the next candidate for the serial merge operation in Section IV.

C. Computing Criticalities Considering Clock Tuning Elements

For circuit optimization the timing analysis tool should report a set of gates that are critical to the circuit performance. The probability that a gate delay affects the circuit performance is called criticality [5], [25], [27]. Because clock tuning elements allow the path delays to compensate each other across flip-flops, critical paths in such circuits may span more than one stage. An example of these critical paths is shown in Fig. 5, where the inverters represent combinational paths with delays above them. The ranges of the clock tuning elements are 3. In this example, we use deterministic delays to show the basic idea. In real circuits, process variations expand the delays to statistical distributions, which we discuss right after explaining this example.

In Fig. 5, if the clock tuning elements are not considered, the critical path is between flip-flops 1 and 2. However, the clock tuning element at flip-flop 2 allows a minimum clock period of 5 at this stage if the intentional skew is configured to 3. On the contrary, the minimum clock period constrained by the paths between flip-flops 4 and 6 is still 6, because the change of the clock skew to flip-flop 5 invariably increases the clock period constrained by one of these two combinational paths. Therefore, the paths between flip-flops 4 and 6, though not having the largest combinational delay in the circuit, become the critical paths.

To capture the critical paths in the presence of clock tuning elements, we first define the criticalities for loops and edges in the constraint graph. Thereafter, the concept in [25] and [26] is extended to include this information into the computation of criticalities for combinational gates. Note here, other criticality computation methods such as [27] can also be extended by incorporating the loop constraints similarly.

According to (11), the circuit performance is constrained by the loop constraints T_l from all the loops in the constraint graph. Because edge weights are random variables in real circuits, any loop has a probability to dominate the circuit performance. The probability that a loop l is critical is thus defined as

$$c_l = \text{prob}\{T_l \geq T_m\} \tag{16}$$

where T_m is defined in (11). Because T_m is the maximum of the loop constraints, the definition of c_l in (16) is the probability that the loop constraint T_l is not smaller than any constraints from other loops. The larger c_l is the more the loop l affects the circuit performance. Therefore, c_l is called loop criticality for loop l and a loop with a large c_l is a critical loop. The edges on a critical loop are candidates for optimization.

In the constraint graph an edge may be on multiple loops. If any of these loops dominates the circuit performance, the edge is critical. Therefore, for an edge e representing the combinational delay between a pair of flip-flops, the sequential criticality is defined as

$$c_e = \text{prob}\left\{\bigvee_{l\in L_e}(T_l \geq T_m)\right\} \tag{17}$$

$$= \text{prob}\left\{\neg\left(\bigwedge_{l\in L_e}(T_l < T_m)\right)\right\} \tag{18}$$

$$= \text{prob}\left\{\neg\left(\max_{l\in L_e}\{T_l\} < T_m\right)\right\} \tag{19}$$

$$= \text{prob}\left\{\max_{l\in L_e}\{T_l\} \geq T_m\right\} \tag{20}$$

where L_e is the set of loops across e and $\max_{l\in L_e}\{T_l\}$ is the maximum of the constraints from all these loops. $\wedge$ means logic and, $\vee$ logic or, and $\neg$ logic not.

An edge in the constraint graph, if not connected to the root node, corresponds to a combinational delay between a pair of flip-flops in the circuit. In the following, we will discuss the definition of criticalities for combinational gate delays considering the effect of clock tuning elements. The discussion will focus on maximum combinational delays constrained by setup time constraints (2). The criticalities corresponding to hold time constraints (4) can be computed similarly.

The sequential criticality above indicates whether the maximum delay of combinational paths between a pair of flip-flops is critical. In statistical timing analysis, it is often required to calculate the probability that a gate is on a critical path for circuit optimization, as in [5], [25], and [27]. In the following, we explain how the sequential criticality is incorporated into the definition of criticalities for gate delays, using the cut-set concept in [25] and [26] as an example. Note that, the proposed sequential criticality is an additional layer above the existing definitions of criticality, and it can be combined with any of the methods in [5], [25], and [27].

The cutset concept in [25] and [26] is introduced to define the criticality of a gate in a combinational circuit. This concept is illustrated in Fig. 6, where the paths in the combinational circuit between flip-flops i and j can be partitioned

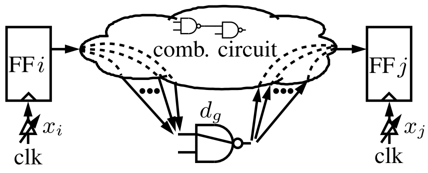


Fig. 6. Path partition for computing criticalities of combinational delays [25], [26]. The gate delay is critical if the paths across it dominate the other paths.

into two sets: P_g and $P_{\bar{g}}$. P_g contains the paths going through the gate g; $P_{\bar{g}}$ contains all the other paths between i and j. The gate delay is critical if the path with the maximum delay passes it, that is to say, the critical path belongs to P_g. The maximum path delay d_{P_g} from P_g can be computed by

$$d_{P_g} = d_{ig} + d_g + d_{gj} \tag{21}$$

where d_{ig} is the maximum delay from i to the input of g, and d_{gj} is the maximum delay from the output of g to j. d_g is the gate delay. Therefore, the probability that the gate delay is critical can be defined as

$$c_g^{\text{no\_tuning}} = \text{prob}\{d_{P_g} \geq d_{P_{\bar{g}}}\} \tag{22}$$

$$= \text{prob}\{d_{P_g} \geq \max\{d_{P_g}, d_{P_{\bar{g}}}\}\} \tag{23}$$

$$= \text{prob}\{d_{P_g} \geq d_{ij}\} \tag{24}$$

where $d_{P_{\bar{g}}}$ is the maximum path delay from $P_{\bar{g}}$; d_{ij} is the maximum delay of all paths between i and j. Note that, (22)–(24) are valid only under assumption of an exact statistical maximum function [26].

In the circuit, the gate g may be on the combinational paths between many pairs of flip-flops, corresponding to a set of edges, written as E_g, in the constraint graph. For example, in Fig. 2(a), the NOR gate connected directly to the output of flip-flop 2 is on the combinational paths from flip-flop 2 to flip-flops 1, 3, and itself, respectively. If any of these paths is critical, the gate is a critical gate. By combining (17)–(20) and (24), the criticality of a gate delay in the presence of clock tuning elements is defined as

$$c_g = \text{prob}\left\{\bigvee_{e\in E_g}\left(\max_{l\in L_e}\{T_l\} \geq T_m \ \wedge\ d_{P_g} \geq d_{ij}\right)\right\} \tag{25}$$

$$= \text{prob}\left\{\bigvee_{e\in E_g}\left(0 \geq \max\left\{T_m - \max_{l\in L_e}\{T_l\}, d_{ij} - d_{P_g}\right\}\right)\right\} \tag{26}$$

where edge e is between nodes i and j in the constraint graph, and L_e is the set of loops containing edge e in the graph.

Let, $\mathcal{C}_e = \max\{T_m - \max_{l\in L_e}\{T_l\}, d_{ij} - d_{P_g}\}$. Then (26) can be rewritten as

$$c_g = \text{prob}\left\{\bigvee_{e\in E_g}(0 \geq \mathcal{C}_e)\right\} \tag{27}$$

$$= \text{prob}\left\{\neg\left(\bigwedge_{e\in E_g}(\mathcal{C}_e > 0)\right)\right\} \tag{28}$$

$$= \text{prob}\left\{\neg\left(\min_{e\in E_g}\{\mathcal{C}_e\} > 0\right)\right\} \tag{29}$$

$$= \text{prob}\left\{\min_{e\in E_g}\{\mathcal{C}_e\} \leq 0\right\}. \tag{30}$$

To compute the criticalities, several variables in (17)–(20) and (25)–(30) should be known. The minimum clock period T_m can be computed using Algorithm 1. The path delays d_{ij} and d_{P_g} can be computed using an SSTA engine as in [25] and [26]. The computation of $\max_{l \in L_e}\{T_l\}$ for an edge e in the constraint graph needs to trace the loops containing the edge e and is explained in the following by extending Algorithm 1.

In Algorithm 1, the function remove_self_loops () deletes self-loops and updates T_m with the newly computed T_l using (11). These loops are either in the original graph, or created from multiple edges by the serial and parallel merge operations. In Algorithm 1, we only keep the weights of the edges during the iterations. Consequently, we have no information to identify the original edges from which a new lower bound T_l is generated, so that we cannot update $\max_{l \in L_e}\{T_l\}$ for individual edges. To solve this problem, we maintain an edge tracing list for each new edge to trace the edges from which the new edge is created. When two consecutive edges are replaced by a new edge during the serial merge operation, the edge tracing lists maintained for the two replaced edges are combined together to construct the new edge list. Because the edges in the graph may have different edge tracing lists, the parallel merge operation in Fig. 4 cannot be applied directly even if the edge weights have the same coefficients of T, because, we need to keep separate edge records so that we can trace back to the original edges when new self-loop constraints are extracted. This limitation increases the number of edges during the graph transformation and leads to a long runtime. In Section IV, we will explain how to adapt the parallel merge operation to handle edge tracing lists. In the iterations, each time when a self-loop is formed, the loop is removed and the loop constraint T_l defined in (10) is updated into the random variable holding $\max_{l \in L_e}\{T_l\}$ for each edge in the edge tracing list. After the iterations are finished, all loop constraints are extracted and the loop constraint $\max_{l \in L_e}\{T_l\}$ for each original edge is computed, so that the criticality for gate delays defined in (30) can be calculated.

The basic concept of criticality computation is summarized in Algorithm 2. The main structure of this algorithm is similar to that of Algorithm 1 because they both capture the nonpositive constraints from loops. The difference is that Algorithm 1 updates the extracted constraints into T_m by the function update_T_m () using (11), but in L8–L10 and in L21–L25 Algorithm 2 updates the constraints into the random variables holding $\max_{l \in L_e}\{T_l\}$ for individual edges that contribute to the weight of the removed self-loop, using the function update_loop_constraint (). After the iterations L12–L32 are finished, we have the loop constraints $\max_{l \in L_e}\{T_l\}$ for all the edges. Thereafter, the criticalities for gate delays are calculated using (30) in L33–L35, where d_{ij} and d_{P_g} are calculated for the combinational paths corresponding to edge e. In this process, we only need to consider the edges with the sequential criticality c_e larger than 0, because other edges are dominated by these edges and thus do not affect the minimum clock period.

Algorithm 2: Computing Criticalities of Gate Delays by Edge Tracing

```
L1  G: the constraint graph created from (2), (4) and (5);
L2  $\nu$, $\nu_i$, $\nu_j$: nodes involved in graph transformation operations;
L3  $\epsilon$, $\epsilon_i$, $\epsilon_j$: edges involved in criticality computation;
L4  N: the number of nodes in the original constraint graph.
L5  Calculate $T_m$ using Algorithm 1;
L6  foreach node $\nu$ in the constraint graph G do
L7      remove_self_loops ($\nu$);
L8      foreach removed edge $\epsilon$ do
L9          update_loop_constraint ($\epsilon$);
L10     end
L11 end
L12 for $k$=1 to N do
L13     prune_edges (G, $T_m$);
L14     $\nu$=select_node (G);
L15     serial_merge ($\nu$);
L16     foreach new edge $\epsilon$ created from the successive edges $\epsilon_i$ and
        $\epsilon_j$ by serial merge operation do
L17         update_tracing_lists ($\epsilon$, $\epsilon_i$, $\epsilon_j$);
L18     end
L19     foreach predecessor $\nu_i$ of $\nu$ do
L20         remove_self_loops ($\nu_i$);
L21         foreach removed edge $\epsilon$ in a self-loop do
L22             foreach edge $\epsilon_i$ in the tracing list of $\epsilon$ do
L23                 update_loop_constraint ($\epsilon_i$);
L24             end
L25         end
L26         foreach successor node $\nu_j$ of $\nu$ do
L27             if there exist parallel edges connecting nodes $\nu_i$ and
                $\nu_j$ then
L28                 parallel_merge ($\nu_i$, $\nu_j$);
L29             end
L30         end
L31     end
L32 end
L33 foreach edge $\epsilon$ with the sequential criticality $c_\epsilon > 0$ do
L34     compute_combinational_gate_criticality ();
L35 end
```

IV. Acceleration Techniques and Discussion

In computing the minimum clock period and criticalities, the edges in the constraint graph are transformed as shown in the main iterations of Algorithms 1 and 2. By connecting the predecessors and successors of a node directly, the serial merge operations in the iterations actually unroll all loops to capture the nonpositive constraints. Without further improvements, these algorithms may be very time-consuming due to the large number of loops in the constraint graph. In this section, we discuss implementation techniques to enhance Algorithms 1 and 2 for better performance.

A. Eliminating Edges Using Ranges of Clock Tuning Elements

The first technique, we use is to remove the edges that are dominated by other edges in the constraint graph. These edges have small weights so that any configuration of the clock tuning elements does not make them critical. This idea has been used in [22] for post-silicon configuration. In this paper, we extend it to handle statistical delays and apply it during iterations to process edges formed by the serial merge operations. These new edges represent the concatenated edges on the paths in the original constraint graph, so that they are potential candidates to be pruned due to the delay compensation across multiple flip-flop stages.

Consider the edge between flip-flops 2 and 3 in Fig. 5, where the ranges of the tuning elements are 3. In any delay configuration of these tuning elements, the path connecting these flip-flops does not affect the minimum clock period, so that we can remove it before starting the graph transformation. Now, we consider another example in Fig. 2(b). If the weight $\overline{w}_{23} - T$ of the setup edge from node 2 to 3 is smaller than $-r_2$, the setup time constraint from this edge can never be violated, because the largest possible configurable clock skew to flip-flop 2 is $-r_2$ and the smallest one to flip-flop 3 is 0. In the constraint graph, the edge from node 2 to the root node with the weight $-r_2$ is created according to the former range constraint, and the edge from the root node to node 3 with the weight 0 is created according to the latter range constraint. If $\overline{w}_{23} - T < -r_2$ holds as discussed above, the setup edge is always dominated by the concatenated edges from node 2 to the root node with the weight $-r_2$ and from the root node to node 3 with the weight 0. Therefore, the setup edge actually makes no contribution to the minimum clock period. Consequently, we can remove this edge safely without losing any accuracy in statistical timing analysis. In the general case, an edge with the weight $w_{ij} - k_{ij}T$ can be removed if it meets

$$w_{ij} - k_{ij}T < -r_i \tag{31}$$

where r_i is the largest configurable delay of the clock tuning element attached to the source node i of the edge.

In the condition (31), w_{ij}, k_{ij}, and r_i are already known. Because a feasible clock period T should always be not less than the minimum clock period T_m, that is, $T \geq T_m$, we check the condition (31) using

$$w_{ij} - k_{ij}T_m < -r_i \tag{32}$$

which is a sufficient condition of (31). In the iterations of Algorithm 1, T_m increases gradually when new loop constraints are merged into it using (11). Therefore, the edge elimination technique is more effective in pruning edges in the later iterations of Algorithm 1 and in the computation of criticalities in Algorithm 2. Since both w_{ij} and T_m are random variables, the condition (32) can only hold with a probability as

$$\text{prob}\{w_{ij} - k_{ij}T_m < -r_i\}. \tag{33}$$

If this pruning probability for an edge approximates 1, for example, if it is larger than 0.98 as in our experiments, we remove the edge from the constraint graph to reduce runtime. This technique is implemented in the function prune_edges() in Algorithms 1 and 2.

B. Pruning Edges by Parallel Dominance

In the serial merge operation in Fig. 3, direct edges are created between the predecessors and successors of the removed node. After many iterations, a large number of parallel edges may appear. But the parallel merge operation in Fig. 4 can only handle parallel edges with the same coefficients of T. To reduce the number of edges in the constraint graph further, we compare the weights of parallel edges and remove the edge whose weight is dominated by the other edges.

Consider two edges with the weights $w_1 - k_1T$ and $w_2 - k_2T$ where $k_1 \neq k_2$. If the weight of the second edge is dominated by the weight of the first edge, the second edge is removed from the constraint graph. The removal condition can be expressed as

$$w_1 - k_1T > w_2 - k_2T. \tag{34}$$

In case of $k_2 - k_1 > 0$, we can transform (34) into

$$T > (w_2 - w_1)/(k_2 - k_1). \tag{35}$$

Similar to the computation in (31)–(33), we substitute T_m for T to create a sufficient condition of (35) as

$$T_m > (w_2 - w_1)/(k_2 - k_1). \tag{36}$$

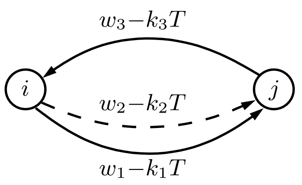


Fig. 7. Parallel pruning. The edge with the weight $w_2 - k_2T$ can be removed if it is statistically dominated by the edge with the weight $w_1 - k_1T$. This removal shall not affect the accuracy of T_m and the lower bounds T_l in (17)–(30) of the edges that are on the loops passing the dominated edge to guarantee correct criticalities.

Because edge weights are random variables, the comparison can only be performed statistically. Therefore, we compute the probability of parallel dominance as

$$\text{prob}\{T_m > (w_2 - w_1)/(k_2 - k_1)\}. \tag{37}$$

If this probability is close to 1, the second edge is dominated by the first edge so that it can be removed from the constraint graph.

The parallel pruning technique removes edges whose delays are statistically dominated by others. In the following, we explain this technique with more details. In the example illustrated in Fig. 7, two parallel edges form loops with the edge having the weight $w_3 - k_3T$ representing the paths from node j to i. Assume that the edge with the weight $w_2 - k_2T$ is dominated by the edge with the weight $w_1 - k_1T$ and removed from the graph. Furthermore, assume that in the current iteration in Algorithm 1 the lower bound of T created by the extracted loop constraints is T_c, which is the current value of T_m before the loop constraints from the two loops in Fig. 7 are processed. Let, $T_1 = (w_1 + w_3)/(k_1 + k_3)$ and $T_2 = (w_2 + w_3)/(k_2 + k_3)$. If the only constraint from the dominating edge is extracted, the new constraint for T can be written as

$$T \geq \max\{T_c, T_1\} = T'. \tag{38}$$

Thereafter, if the constraint from the dominated edge would be included, the constraint for T becomes

$$T \geq \max\{T', T_2\}. \tag{39}$$

This augmented constraint should be equivalent to (38) so that the second edge can be removed safely. A sufficient condition for this requirement is $T' > T_2$, which we will deduce from the dominance condition (34) and the condition $k_2 - k_1 > 0$ for (36). From (38), we have

$$T' \geq T_1 = (w_1 + w_3)/(k_1 + k_3) \Leftrightarrow k_1T' + k_3T' \geq w_1 + w_3. \tag{40}$$

If the edge pruning technique is applied in Algorithm 1 to calculate the minimum clock period, the new loop constraints are updated into T_m using (11) gradually. At the moment of edge pruning, the variable T_m used in the condition (36) is actually T_c in (38), so that we can write the pruning condition in (36) as

$$T' \geq T_c > (w_2 - w_1)/(k_2 - k_1) \Leftrightarrow k_2T' - k_1T' > w_2 - w_1. \tag{41}$$

Adding both sides of (40) and (41), respectively, we have

$$k_2T' + k_3T' > w_2 + w_3 \Leftrightarrow T' > (w_2 + w_3)/(k_2 + k_3) = T_2. \tag{42}$$

Comparing (39) and (42) we know that (36) is a sufficient condition for the safe removal of the dominated edge.

According to (40)–(42), we can apply the pruning technique in Algorithm 1 to calculate T_m. When computing criticalities of gate delays from (17) to (30) using Algorithm 2, the minimum clock period T_m has been calculated by Algorithm 1 in L5 of Algorithm 2. Therefore, we cannot assume that T_m used in (36) is equal to T_c. Instead, we use (25) to explain the pruning technique in criticality computation similar to (38)–(42). The condition $\max_{l\in L_e}\{T_l\} \geq T_m$ in (25) can be rewritten as

$$\max\left\{\max_{l\in L_e\setminus l_1,l_2}\{T_l\} - T_m, T_1 - T_m, T_2 - T_m\right\} \geq 0 \quad (43)$$

where T_1 and T_2 are the loop constraints from the loops formed by the two parallel edges and the third edge in Fig. 7, respectively. Let, $d_c = \max\{\max_{l\in L_e\setminus l_1,l_2}\{T_l\} - T_m, T_1 - T_m\}$, and we have

$$d_c \geq T_1 - T_m = (w_1 + w_3)/(k_1 + k_3) - T_m \Leftrightarrow \quad (44)$$

$$(k_1 + k_3)d_c \geq (w_1 + w_3) - (k_1 + k_3)T_m. \quad (45)$$

From (36), we can deduce

$$(k_2 - k_1)T_m > w_2 - w_1. \quad (46)$$

Adding both sides of (45) and (46) we have

$$(k_1 + k_3)d_c > (w_2 + w_3) - (k_2 + k_3)T_m. \quad (47)$$

Because T_m is the maximum of all loop constraints, it is no smaller than $\max_{l\in L_e\setminus l_1,l_2}\{T_l\}$ and T_1, so that we can deduce that d_c is not greater than 0. Therefore, (47) can be written as

$$0 > (w_2 + w_3) - (k_2 + k_3)T_m \Leftrightarrow \quad (48)$$

$$T_m > (w_2 + w_3)/(k_2 + k_3) = T_2. \quad (49)$$

Thus, we can state that with the condition (36) the loop containing the dominated edge with the weight $w_2 - k_2T$ in Fig. 7 does not affect the minimum clock period and also does not contribute to the criticalities of gate delays.

The parallel pruning technique above is implemented in the function prune_edges () in Algorithms 1 and 2. During the iterations, the serial merge operations create new edges representing paths in the original graph. Therefore, the new edge weights are balanced between consecutive stages and exhibit a tendency of being dominated by other parallel edges, so that they can be handled effectively by the parallel pruning technique. This can also be explained by the fact that in most cases a large combinational path delay in the circuit needs not to be compensated by flip-flop stages far away.

The pruning technique is applied in each iteration of Algorithm 2 to trim edges. The overall result is that only the critical part of the graph is unrolled during the iterations so that the runtime can be reduced. Moreover, the pruning technique also reduces redundant loops created by the serial merge operation, as exemplified in Fig. 8 where the edge weights are denoted as shown for simplification. In this graph, if node 1 is removed first by the serial merge operation, an edge with the weight $d_{21} + d_{13}$ is created from node 2 to node 3 in the graph on the right. If node 2 is removed thereafter, this edge is merged with the edge from 4 to 2 and a new edge with the weight $d_{41} + d_{12} + d_{21} + d_{13}$ will be created. The new edge is dominated by the edge from node 4 to 3 with the weight $d_{41} + d_{13}$ because $d_{12} + d_{21}$ is not greater than 0 and this constraint has been captured by the self-loop at node 2. This example shows that although the serial merge operation does not miss any loop in the graph, it may create redundant edges.

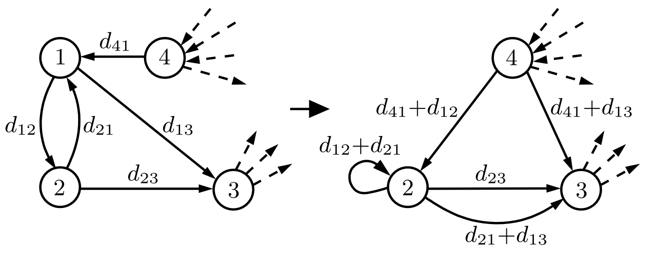


Fig. 8. Redundant edge created by serial merge operation. The edges between nodes 1 and 2 lead to a redundant edge from node 2 to 3 after node 1 is removed by the serial merge operation.

These edges are then pruned under the condition of parallel dominance discussed above, so that the number of the edges in the constraint graph does not increase unnecessarily.

C. Merging Tracing Lists in Computing Criticalities

In computing criticalities of gate delays using Algorithm 2, for each newly created edge we maintain an edge list to trace edges from which the new edge is created. Each time when a self-loop is removed, the loop constraint of each edge in the tracing list is updated in L21–L25 of Algorithm 2. The serial merge operation creates direct edges between predecessors and successors of the removed node. For such a new edge the traced edges in the lists of the two replaced edges are merged into the new tracing list, unless a self-loop is formed and the loop constraints for the traced edges are updated. The operation of merging edge tracing lists is implemented in the function update_tracing_lists () in L16–L18.

For parallel edges, we apply the pruning technique discussed above to reduce the number of edges in the function prune_edges (). Specifically, if the coefficients of T in the edge weights of parallel edges are equal, we can compress them into one edge directly using the parallel merge operation parallel_merge () in L26–L30 of Algorithm 2. In this case, the two parallel edges with the weights $w_{ij_1} - k_1T$ and $w_{ij_2} - k_2T$ can be merged into a new edge. The new edge weight is calculated as $\max\{w_{ij_1}, w_{ij_2}\} - k_1T$ because $k_1 = k_2$. When computing the criticalities, however, the parallel edges may have different edge tracing lists and the new edge weight cannot be used for the traced edges from any of the lists, because $\max\{w_{ij_1}, w_{ij_2}\}$ is different from w_{ij_1} and w_{ij_2} which are calculated across different loops. To solve this problem, we maintain a random variable for each traced edge. In the above case, suppose we have an edge e in the tracing list of the first merged edge. After the parallel operation, weight difference $\max\{w_{ij_1}, w_{ij_2}\} - w_{ij_1}$ is added to the tracing variable of e. This variable represents the difference between the weight of the new edge and the weight of the edge replaced by the parallel merge operation. When a self-loop is formed later, the loop constraint T_l for edge e can be recovered by the edge weight of the self-loop minus the accumulated weight difference for computing the criticalities of gate delays.

D. Node Order During Graph Transformation

In Algorithms 1 and 2, the next node for the serial merge operation is selected by the function select_node (). The order of the selected nodes may affect the performance significantly. If a node with m predecessors and n successors is removed by the serial merge operation, $m \times n$ new edges may be created. An extreme case is the root node, which has edges to and from all the other nodes. If the root node is removed using the serial merge operation, between any two nodes in the graph two new edges are constructed. In a graph with many nodes, it is impractical to process so many edges by further graph transformation.

In a complex graph, although an optimal node processing order that guarantees the minimal number of edges during graph transformation may exist, to find this optimal order is very difficult and, even if it is possible, consumes much runtime. In the proposed method, we select the next node for the serial merge operation heuristically. In each iteration, we select the node with the smallest node connection, which is defined as $m_i \times n_i$ for node i, where m_i is the number of predecessors of i and n_i the number of successors of i. With this heuristic method, the nodes that might lead to many new edges after their removal, for example, the root node, are processed later. The pruning techniques applied to the edges in the earlier iterations may reduce the number of the edges in the graph so that the overall runtime can be reduced. In implementation, an ordered list of all the nodes in the graph is maintained. Each time when a node is removed by the serial merge operation, the node connections of the predecessors and successors of the removed node are updated for refreshing the node order. Therefore, the next node for transformation is always at the head of the ordered list so that the runtime in node selection can be reduced.

E. Discussion

In Section III, the timing constraints (2), (4), and (5) are represented in a constraint graph. The existence of a solution for the constraint set is equivalent to the condition that all loops in the graph have no positive accumulated weights. To capture the timing constraints from loops, we apply the graph transformation operations explained in Section III iteratively. The feasibility of this method can be explained by using a sample in Monte Carlo simulation. For such a sample all the methods discussed above are still valid so that we can use them to calculate a minimum clock period for this sample. With all samples together, we have the distribution curve of the minimum clock period. From the statistical view, we can still apply the proposed method with the same node order and the same merge operations to calculate the minimum clock period. The only difference is that we should substitute statistical max and sum computations for the static counterparts. This is the same reasoning for statistical timing analysis of combinational circuits, where we use the same propagation algorithm but statistical computations.

The equivalence between the difference constraints and the graph representation is valid only when the ranges defined in (5) take continuous values. If the ranges of clock tuning elements are discrete as in many implementations, integer linear programming should be used to calculate the exact minimum clock period for each sample in Monte Carlo simulation. However, integer linear programming works differently with each sample of Monte Carlo simulation, for example, the different searching directions in finding the optimal value. Therefore, we cannot establish a closed-form formulation, and thus, cannot find a general method to perform fast statistical timing analysis in this case.

Instead of computing the exact minimum clock period T_d directly, the result T_m by assuming that all tuning elements have continuous ranges is a lower bound of the clock period of the case with discrete ranges, since all discrete configuration values are also feasible in the continuous configuration space. Assume that the interval of the discrete ranges of the clock tuning elements is θ. Then $T_m + \theta$ is an upper bound for the minimum clock period in the discrete case, because for each solution of the continuous case we can move the delay of each tuning element to the nearest lower integer value to find a feasible discrete configuration that leads to the minimum clock period not larger than $T_m + \theta$. According to this discussion, we can conclude these two bounds as $T_m \leq T_d \leq T_m + \theta$. In reality, the interval of the ranges may be very small compared with the clock period, so that a good approximation accuracy can still be expected.

V. Experimental Results

The discussed algorithms were implemented in C++ and tested using a 2.67 GHz CPU. We used five large circuits, s5378 to s38584, from the ISCAS89 benchmarks and five other large circuits, mem_ctrl to des_perf, from TAU 2013 variation-aware timing analysis contest [40] for our experiments. Information about these circuits is shown in Table I, where n_s is the number of flip-flops and n_g the number of logic gates. The largest circuit in the experiments has more than 8K flip-flops and 86K logic gates. For experiments, we assumed that each flip-flop has a clock tuning element to test the efficiency of the proposed method with large constraint graphs. In practice, these tuning elements are selectively inserted considering circuit performance and area cost at the same time, e.g., using the methods in [16] and [17]. The ranges of the clock tuning elements were set to 1/8 of the clock periods from the original circuits, roughly the range used in [14]. The logic gates in the circuits were mapped to a library from an industry partner. 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, respectively [41]. The gate delays were generated using the method proposed in [3], in which spatial correlation from global and local variations is decomposed by principal component analysis. The resulting timing model is in a linear form of independent random variables. We used the method in [5] to compute the sum and maximum/minimum of random variables.

To verify the accuracy of the proposed method in computing the minimum clock period, we ran Monte Carlo simulation with 10 000 samples. For each sample, the minimum clock period constrained by (2), (4), and (5) was computed using a linear programming solver [42]. The distribution formed by all the performance samples was compared with T_m computed by the proposed method. The results are shown in Table I, where $\mathcal{E}_\mu$ is the relative error of the mean of the minimum clock period T_m, defined as $|\mu_{\text{SSTA}} - \mu_{\text{MC}}|/\mu_{\text{MC}}$, where μ_{SSTA} and μ_{MC} are the means of the minimum clock period computed by the proposed method and by Monte Carlo simulation, respectively. Similar to $\mathcal{E}_\mu$, $\mathcal{E}_\sigma$ shows the accuracy of the standard deviation of the clock period. $\mathcal{E}_{T_{2\sigma}}$ shows the relative yield error at the 2σ clock period from Monte Carlo simulation. From $\mathcal{E}_\mu$, $\mathcal{E}_\sigma$, and $\mathcal{E}_{T_{2\sigma}}$, we can see that the results of the proposed method have good accuracy and the predicted yields have not more than 0.5% error.

The major advantage of the proposed method is its efficiency. The runtimes of the proposed method working on different benchmark circuits are shown in Table I with t_p in seconds, and the runtimes of Monte Carlo simulation are shown as t_m in hours. The speedup ratios of the proposed method compared with Monte Carlo simulation are shown in the r_t column. From this comparison, we can conclude that the proposed

TABLE I
RESULTS OF STATISTICAL TIMING ANALYSIS AND CRITICALITY COMPUTATION

Circuit			SSTA accuracy			Runtime			Criticality>0.3			Criticality>0.1			Runtime		
	n_s	n_g	$\mathcal{E}_\mu(\%)$	$\mathcal{E}_\sigma(\%)$	$\mathcal{E}_{T_{2\sigma}}(\%)$	$t_p(s)$	$t_m(h)$	r_t	n_c	n_m	$\mathcal{E}_c$	n_c	n_m	$\mathcal{E}_c$	$t_p(s)$	$t_m(h)$	r_t
s5378	179	2779	0.55	1.18	0.05	0.10	0.39	14685	34	0	-	38	0	-	0.15	0.70	16851
s9234	211	5597	0.13	0.42	0.10	0.17	0.99	21579	78	4	0.09	167	2	0.09	0.29	2.59	31733
s13207	638	7951	0.12	0.23	0.04	0.16	1.75	39448	31	0	-	129	26	0.14	0.34	7.37	77800
s15850	534	9772	0.62	1.04	0.16	0.43	2.94	24782	74	0	-	273	12	0.10	0.90	12.83	51407
s38584	1426	19253	0.86	0.95	0.16	0.53	4.80	32832	74	0	-	110	1	0.04	1.29	47.18	132132
mem_ctrl	1065	10327	0.12	0.39	0.26	0.91	5.49	21700	37	2	0.06	44	0	-	3.20	59.02	66359
usb_funct	1746	14381	1.39	0.89	0.03	0.66	5.21	28575	56	4	0.05	94	0	-	1.90	67.67	128352
ac97_ctrl	2199	9208	1.19	0.35	0.21	0.60	5.45	32935	0	0	-	70	0	-	2.02	92.89	165359
pci _bridge32	3321	12494	0.14	1.11	0.36	3.28	13.96	15296	33	0	-	74	4	0.03	8.79	350.75	143734
des_perf	8808	86020	0.75	0.13	0.15	4.81	55.48	41489	-	-	-	-	-	-	45.49	-	-

n_s: # of flip-flops; n_g: # of gates; $\mathcal{E}_\mu$: difference of mean in percentage; $\mathcal{E}_\sigma$: difference of standard deviation in percentage; $\mathcal{E}_{T_{2\sigma}}$: difference of yield at 2σ clock period; t_p: runtime of the proposed method in seconds; t_m: runtime of Monte Carlo simulation in hours; r_t: ratio of runtimes; n_c: # of gates with criticality above threshold; n_m: # of gates not captured by the proposed method; $\mathcal{E}_c$: maximum difference of the criticalities of the uncaptured gates.

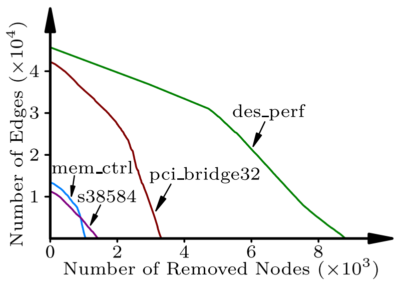
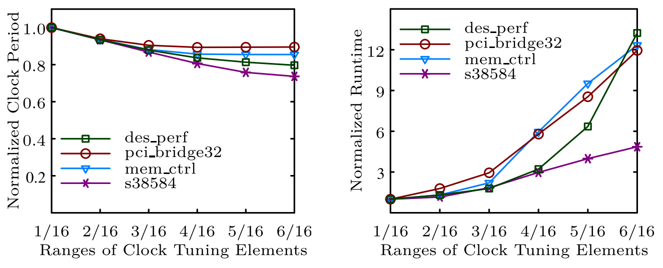
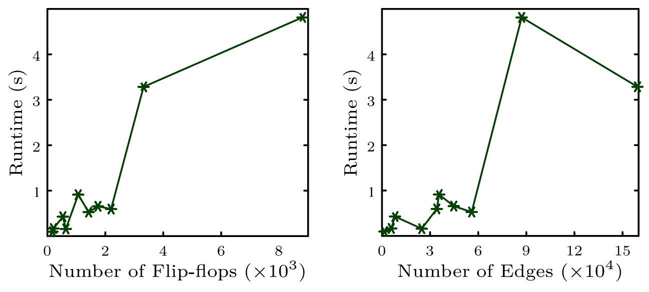


Fig. 9. Edge numbers during node removal. The numbers of edges exhibit monotonic decrease due to pruning and merging techniques.



(a) (b)

Fig. 11. Performance and runtime in relation to ranges of clock tuning elements. Circuit performance increases as the ranges are increased, but is bounded by the loops formed by paths across flip-flops exclusively. Runtime increases because the pruning techniques (33) and (37) become less effective with a decreased T_m. (a) Performance and (b) SSTA runtime versus ranges of clock tuning elements.



(a) (b)

Fig. 10. Runtime trend in relation to circuit size. Runtime versus number of (a) nodes and (b) edges.

method is at least four orders of magnitude faster than Monte Carlo simulation. Since Monte Carlo simulation is the only existing method available for statistical timing analysis of circuits with clock tuning elements, this comparison demonstrates the advantage of the proposed method and its applicability to accelerate methods that depend on the results of statistical timing analysis, for example, in circuit optimization.

The proposed method unrolls the loops in the constraint graph using the serial merge operation, and parallel edges are pruned and merged to reduce the number of edges in the graph. In the worst case, the complexity of the algorithm is exponential in the numbers of nodes and edges in the constraint graph. The efficiency of the proposed method results from the techniques in Section IV, where edges are pruned during graph transformations. The effect of these heuristic techniques depends on the circuit structure and gate delays, so that the computational complexity cannot be presented in an accurate mathematical form. To demonstrate the efficiency of the proposed method, we show the trends of the edge numbers in the constraint graphs of several test cases in Fig. 9. In all these cases, the numbers of edges actually decrease monotonically, since only the edges that affect the minimum clock period are kept in the graphs due to edge pruning, thus, explaining the much shorter runtime compared with Monte Carlo simulation. To show the complexity trend of the proposed method, we illustrate the runtimes of processing circuits with different sizes regarding the numbers of nodes and edges in the constraint graphs in Fig. 10(a) and (b), respectively. These diagrams show that the runtime of the proposed method increases with the circuit size, but still remains in the acceptable range. The complexity has a similar trend if different global and local variations are considered, because gate delays are usually represented in the same form, e.g., linear or quadratic polynomial of independent random variables.

Using clock tuning elements the performance of a circuit can be improved, as explained in Section II. However, the performance improvement is bounded because after the ranges of clock tuning elements reach a threshold the circuit performance is determined by the maximum average edge delay across loops exclusively. This maximum average edge delay does not change as the ranges of the clock tuning elements are enlarged, so that the circuit performance cannot be improved by further time borrowing. To show the relation between circuit performance and ranges of clock tuning elements, we tested the circuit performances of some benchmark circuits by setting the ranges of tuning elements r_i in (5) from $T_n/16$ to $6T_n/16$ with $T_n/16$ as interval, where T_n is the clock period without considering the clock tuning elements. The trends of the mean values of the minimum clock periods are shown in Fig. 11(a). From this diagram, we can see that the clock

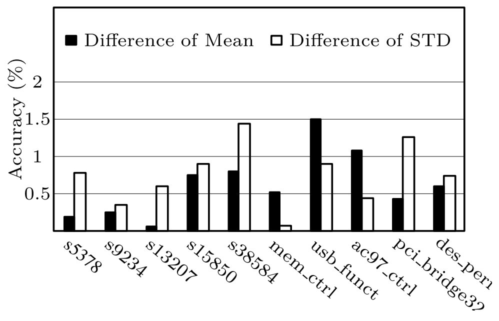


Fig. 12. Approximation accuracy of discrete Monte Carlo simulation. The results from SSTA for the circuits having clock tuning elements with continuous ranges are lower bounds for the discrete cases.

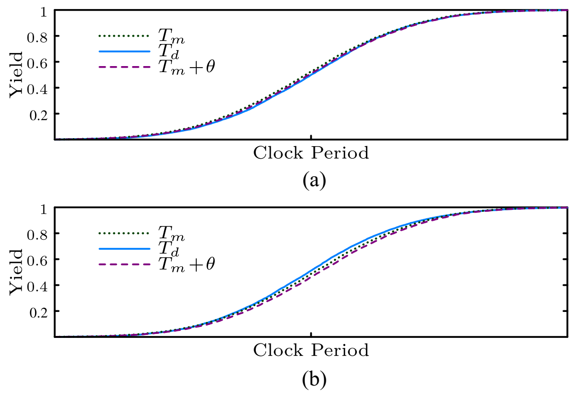


Fig. 13. Approximation and bounds of minimum clock periods of circuits with discrete clock tuning ranges. The upper and lower bounds have good accuracy of approximation, but are not bounds exactly due to the approximation in statistical computations. Bounds of (a) s38584 and (b) des_perf with discrete clock tuning ranges.

period of s38584 decreases as the ranges of clock tuning elements are enlarged. But the clock period of pci_bridge32 nearly has no change when the ranges of clock tuning elements are larger than $3T_n/16$, because in this case the constraints from loops across flip-flops dominate the circuit performance. In Fig. 11(b), the trends of the runtimes of the proposed method with respect to different ranges of clock tuning elements are also shown. It is obvious that the runtimes increase as the ranges of clock tuning elements are enlarged, because the pruning techniques (33) and (37) become less effective with a decreased T_m. However, even with this increase, the absolute runtime of the proposed method is still small. For example, the analysis of the largest case des_perf finished within 50 s but the corresponding Monte Carlo simulation did not produce the result even in days. Therefore, the proposed method can be used to evaluate the relation between the minimum clock period and the ranges of clock tuning elements efficiently, so that designers have the chance to evaluate trade-offs between performance and die size of the tuning elements.

To verify the proposed criticality computation, we sampled the constraint graph in each iteration of Monte Carlo simulation. Then, we calculated the distances between nodes in the constraint graph using the Bellman–Ford algorithm. After this, each edge was checked whether the loop across it determines the minimum clock period computed by linear programming. The criticalities from Monte Carlo simulation and the proposed method are compared and the results are shown in Table I. Owing to the approximation in the statistical computations of SSTA engines, the criticalities cannot be calculated accurately. As pointed out in [25], the skewness of the distribution, which is not considered by many SSTA engines, may cause large inaccuracy in criticality computation. Especially when the delays of critical paths are compared with the minimum clock period of the circuit, the criticality is very sensitive to the inaccuracy of the statistical approximations. Nevertheless, because the purpose to compute criticalities is to select the gates for optimization, we compare the sets of critical gates selected by Monte Carlo simulation and the proposed method. In Table I, the columns >0.3 and >0.1 show the numbers of gate delays with criticalities larger than 0.3 and 0.1, respectively. The comparison of criticalities for des_perf was not fulfilled in the experiment due to the unaffordable runtime in Monte Carlo simulation. In Table I, n_c is the number of gates identified by Monte Carlo simulation, and n_m is the number of gates which are not captured by the proposed method. $\mathcal{E}_c$ is the maximum difference between the criticalities of gates which are not captured by the proposed method. For example, for s9234, in the 78 gates with criticalities larger than 0.3 four gates are not captured. In these four missing gates, the maximum criticality difference compared with the criticalities computed by Monte Carlo simulation is 0.09. From this comparison, we can conclude that the proposed method can capture most of the critical gates, but it may still miss some due to the approximation in statistical computations, though the difference between the criticalities is not large. The runtime comparison for computing criticalities is shown in the last three columns of Table I. The acceleration ratio of the proposed method to Monte Carlo simulation is still remarkable, particularly for the large benchmark circuits.

In Section IV, we have discussed the upper and lower bounds of the minimum clock period T_d for circuits containing clock tuning elements with discrete ranges. If we assume the ranges are continuous and apply the proposed method, the resulting minimum clock period T_m is a lower bound of T_d. In addition, if we increase T_m by θ which is the discrete interval of the ranges of the clock tuning elements, we then create an upper bound for T_d. In Fig. 12, we show the differences of means and standard deviations of T_m and T_d. Here, the discrete adjustable range has eight steps. These differences shown in the y-axis in percentage demonstrate a reasonable approximation of T_m to T_d generally. In Fig. 13(a) and (b), the cumulative distribution functions of T_m as the lower bound, T_d as the result of Monte Carlo simulation, and $T_m+\theta$ as the upper bound for circuits s38584 and des_perf are shown, respectively. In these two comparisons, both bounds are lower bounds for s38584 and upper bounds for des_perf, due to the approximation in the max and sum computations in statistical timing analysis. But these bounds all exhibit a reasonable approximation accuracy.

VI. Conclusion

In this paper, we propose a fast method to compute the minimum clock periods for circuits with post-silicon clock tuning elements. The delays of these elements can be adjusted for each individual chip after manufacturing to achieve the maximum performance. The proposed method applies serial merge operations to unroll the loops in the constraint graph so that nonpositive loop constraints can be captured by self-loops. Parallel merge operations and pruning techniques are applied to trim edges during iterations to reduce runtime. Criticalities of logic gates are also calculated by tracing the edges on critical loops. Experimental results confirm that the propose method is faster than Monte Carlo simulation by several orders of magnitude while still maintaining good accuracy.

References

[1] 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 Design*, San Jose, CA, USA, 2011, pp. 111–117.

[2] 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.*, vol. 27, no. 4, pp. 589–607, Apr. 2008.

[3] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2003, pp. 621–625.

[4] K. Kang, B. C. Paul, and K. Roy, "Statistical timing analysis using levelized covariance propagation," in *Proc. Design Autom. Test Europe Conf.*, vol. 2. Munich, Germany, 2005, pp. 764–769.

[5] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *Proc. Design Autom. Conf.*, San Diego, CA, USA, 2004, pp. 331–336.

[6] V. Khandelwal and A. Srivastava, "A general framework for accurate statistical timing analysis considering correlations," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2005, pp. 89–94.

[7] Y. Zhan *et al.*, "Correlation-aware statistical timing analysis with non-Gaussian delay distributions," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2005, pp. 77–82.

[8] L. Zhang, W. Chen, Y. Hu, J. A. Gubner, and C. C.-P. Chen, "Correlation-preserved non-Gaussian statistical timing analysis with quadratic timing model," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2005, pp. 83–88.

[9] Z. Feng, P. Li, and Y. Zhan, "Fast second-order statistical static timing analysis using parameter dimension reduction," in *Proc. Design Autom. Conf.*, San Diego, CA, USA, 2007, pp. 244–249.

[10] S. Bhardwaj, S. Vrudhula, and A. Goel, "A unified approach for full chip statistical timing and leakage analysis of nanoscale circuits considering intradie process variations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1812–1825, Oct. 2008.

[11] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah, "Parameterized block-based statistical timing analysis with non-Gaussian parameters, nonlinear delay functions," in *Proc. Design Autom. Conf.*, Anaheim, CA, USA, 2005, pp. 71–76.

[12] J. Singh and S. Sapatnekar, "Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2006, pp. 155–160.

[13] A. Lange *et al.*, "Probabilistic standard cell modeling considering non-Gaussian parameters and correlations," in *Proc. Design Autom. Test Europe Conf.*, Dresden, Germany, 2014, pp. 1–4.

[14] S. Tam *et al.*, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1545–1552, Nov. 2000.

[15] 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.*, San Francisco, CA, USA, 2005, pp. 292–293.

[16] J.-L. Tsai, D. Baik, C. C.-P. Chen, and K. K. Saluja, "A yield improvement methodology using pre- and post-silicon statistical clock scheduling," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2004, pp. 611–618.

[17] J.-L. Tsai, L. Zhang, and C. C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2005, pp. 575–581.

[18] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation," in *Proc. Int. Symp. Phys. Design*, Austin, TX, USA, 2007, pp. 11–18.

[19] 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. Test Europe Conf.*, Nice, France, 2009, pp. 292–295.

[20] A. Chakraborty *et al.*, "Dynamic thermal clock skew compensation using tunable delay buffers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 6, pp. 639–649, Jun. 2008.

[21] R. Ye, F. Yuan, and Q. Xu, "Online clock skew tuning for timing speculation," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2011, pp. 442–447.

[22] Z. Lak and N. Nicolici, "A novel algorithmic approach to aid post-silicon delay measurement and clock tuning," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1074–1084, May 2014.

[23] 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.*, vol. 31, no. 12, pp. 1845–1856, Dec. 2012.

[24] X. Li, J. Le, M. Celik, and L. Pileggi, "Defining statistical sensitivity for timing optimization of logic circuits with large-scale process and environmental variations," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2005, pp. 844–851.

[25] J. Xiong, V. Zolotov, N. Venkateswaran, and C. Visweswariah, "Criticality computation in parameterized statistical timing," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2006, pp. 63–68.

[26] J. Xiong, V. Zolotov, and C. Visweswariah, "Incremental criticality and yield gradients," in *Proc. Design Autom. Test Europe Conf.*, Munich, Germany, 2008, pp. 1130–1135.

[27] H. Mogal, H. Qian, S. Sapatnekar, and K. Bazargan, "Clustering based pruning for statistical criticality computation under process variations," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2007, pp. 340–343.

[28] J. Xiong, C. Visweswariah, and V. Zolotov, "Statistical ordering of correlated timing quantities and its application for path ranking," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2009, pp. 122–125.

[29] D. Sinha, C. Visweswariah, N. Venkateswaran, J. Xiong, and V. Zolotov, "Reversible statistical max/min operation: Concept and applications to timing," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 1067–1073.

[30] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi, "Post-fabrication clock-timing adjustment using genetic algorithms," *IEEE J. Solid-State Circuits*, vol. 39, no. 4, pp. 643–650, Apr. 2004.

[31] S. Naffziger *et al.*, "The implementation of a 2-core, multi-threaded Itanium family processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 197–209, Jan. 2006.

[32] J. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[33] J. P. Fishburn, "Solving a system of difference constraints with variables restricted to a finite set," *Inf. Process. Lett.*, vol. 82, no. 3, pp. 143–144, 2002.

[34] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1990.

[35] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Graph algorithms for clock schedule optimization," in *Proc. Int. Conf. Comput.-Aided Design*, Santa Clara, CA, USA, 1992, pp. 132–136.

[36] R. Deokar and S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proc. Int. Symp. Circuits Syst.*, London, U.K., 1994, pp. 407–410.

[37] S.-H. Huang and Y.-T. Nieh, "Synthesis of nonzero clock skew circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 6, pp. 961–976, Jun. 2006.

[38] R. Chen and H. Zhou, "Statistical timing verification for transparently latched circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 9, pp. 1847–1855, Sep. 2006.

[39] 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 Design*, San Jose, CA, USA, 2010, pp. 524–531.

[40] (2013). *TAU 2013 Contest: Variation Aware Timing Analysis*. [Online]. Available: https://sites.google.com/site/taucontest2013/

[41] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. Custom Integr. Circuits Conf.*, San Diego, CA, USA, 2001, pp. 223–228.

[42] *Gurobi Optimizer Reference Manual*, Gurobi Optim. Inc., Houston, TX, USA, 2013. [Online]. Available: http://www.gurobi.com

Bing Li received the bachelor's and master's degrees in communication and information engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2000 and 2003, respectively, and the Dr.Ing. degree in electrical engineering from Technische Universität München (TUM), Munich, Germany, in 2010.

He is currently a Researcher with the Institute for Electronic Design Automation, TUM. His current research interests include timing and power analysis and emerging systems.

Ulf Schlichtmann (S'88–M'90) received the Dipl.Ing. and Dr.Ing. degrees in electrical engineering and information technology from Technische Universität München (TUM), Munich, Germany, in 1990 and 1995, respectively.

He was with Siemens AG, Munich, and Infineon Technologies AG, Munich, from 1994 to 2003, where he held various technical and management positions in design automation, design libraries, IP reuse, and product development. He has been with TUM as a Professor and the Head of the Institute for Electronic Design Automation, since 2003. He served as the Dean of the Department of Electrical Engineering and Information Technology, TUM, from 2008 to 2011. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems.

4 Design-Phase Buffer Allocation for Post-Silicon Clock Binning by Iterative Learning

Design-Phase Buffer Allocation for Post-Silicon Clock Binning by Iterative Learning

Grace Li Zhang, Bing Li, Jinglan Liu, Yiyu Shi, *Senior Member, IEEE*, and Ulf Schlichtmann, *Member, IEEE*

***Abstract*—At submicrometer manufacturing technology nodes, process variations affect circuit performance significantly. To counter these variations, engineers are reserving more timing margin 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 meet given timing specifications. 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, critical paths can be balanced with neighboring paths in each chip specifically to counter the effect of process variations. Consequently, chips with timing failures can be rescued and the yield can thus be improved. This is specially useful in high-performance designs, e.g., high-end CPUs, where clock binning makes chips with higher performance much more profitable. In this paper, we propose a method to determine where to insert post-silicon tuning buffers during the design phase to improve the overall profit with clock binning. This method learns the buffer locations with a Sobol sequence iteratively and reduces the buffer ranges afterward with tuning concentration and buffer grouping. Experimental results demonstrate that the proposed method can achieve a profit improvement of about 14% on average and up to 26%, with only a small number of tuning buffers inserted into the circuit.**

***Index Terms*—Clock binning, iterative learning, post-silicon tuning, process variations, yield.**

I. Introduction

AT ADVANCED technology nodes, process variations have become relatively larger, and thus caused expensive overdesign due to timing margins reserved during the design phase. To meet the challenges imposed by process variations, previous methods model process variations as random variables and incorporate them into timing analysis directly, leading to a boom of research on statistical static timing analysis (SSTA) in the last decade [2]. With the knowledge of the distributions of process variations, SSTA methods produce a performance-yield curve with which designers have a chance to make a tradeoff between different design goals. To alleviate the effect of process variations, many researchers have also worked on circuit structure level to introduce special devices and mechanisms. For instance, the Razor method [3] boosts circuit performance up to the limit where timing errors occur during circuit operation. Another technique to counter process variations is to use post-silicon tuning devices to adapt chips individually according to the effect of process variations after manufacturing.

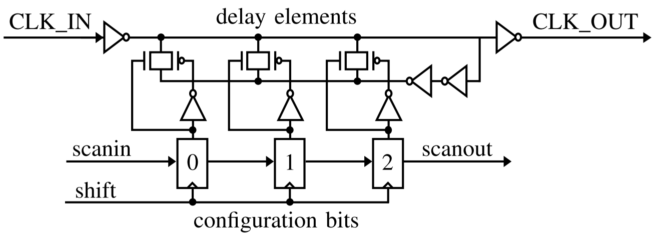


Fig. 1. Post-silicon tuning buffer in [4] with three configuration bits.

A widely used post-silicon tuning technique is clock tuning with delay buffers. For example, the structure of the delay buffer used in [4] is illustrated in Fig. 1. The delay of this buffer can be changed by setting the configuration bits in the three registers. In high-performance designs, tuning buffers like this are inserted during the design phase. After manufacturing, the delay values of these buffers are configured to allot critical paths more timing budget by shifting clock edges toward the stages with smaller combinational delays. These critical paths might be different in individual chips due to process variations, so that only post-silicon tuning can counterbalance them efficiently. By balancing delay budgets across consecutive register stages, chips that might have failed to meet timing specifications can be revitalized, leading to an increased yield at the expense of additional area taken by these buffers. This post-silicon tuning technique works seamlessly with other optimization techniques, e.g., gate/wire sizing and timing-driven placement, since it mainly deals with delay imbalance introduced in manufacturing by process variations instead of during the design phase.

Manuscript received July 28, 2016; revised December 19, 2016; accepted April 26, 2017. Date of publication May 9, 2017; date of current version January 19, 2018. This work was supported in part by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre "Invasive Computing" (SFB/TR 89). A preliminary version of this paper was published in the Proceedings of the Design, Automation and Test in Europe Conference, 2016 [1]. The major improvement of this paper over [1] is to process multiple samples with an iterative learning procedure using a low-discrepancy sample sequence (Sobol sequence). This paper was recommended by Associate Editor P. Gupta.

G. L. Zhang, B. Li, and U. Schlichtmann are with the Institute for Electronic Design Automation, Technical University of Munich, 80333 Munich, Germany (e-mail: grace-li.zhang@tum.de; b.li@tum.de; ulf.schlichtmann@tum.de).

J. Liu and Y. Shi are with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: jliu16@nd.edu; yshi4@nd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2702632

Post-silicon clock tuning buffers have various implementations and characteristics. The tuning buffer proposed in [5] provides precise adjustable delays of less than 30 ps by voltage-controlled driver strength. The design in [6] uses a delay line to generate delays with 1-ps resolution. The de-skew buffer in [7] consists of CMOS inverters and arrays of passive loads and is capable of creating a 170-ps tunable delay range in 8.5-ps steps. The controlled contention design [4] provides a 140-ps delay range with eight steps. After manufacturing, these delays can be adjusted through the test access port to tune individual chips.

In recent years, several methods have been proposed for statistical timing analysis and optimization of circuits with post-silicon clock tuning buffers. In [8], a clock scheduling method is developed and clock tuning buffers are selectively inserted to balance the skews due to process variations. In [9], algorithms are proposed to insert buffers into the clock tree to guarantee a given yield, while either the number of buffers or the total area of all buffers is minimized. The optimization problem is solved by evaluating the yield gradient with simultaneous perturbation and Monte Carlo simulation. In [10], the yield loss due to process variations and the total cost of clock tuning buffers are formulated together for gate sizing. The resulting optimization problem is solved using a stochastic cutting-plane method with an STA scheme based on Monte Carlo simulation. In [11], the placement of clock tuning buffers is investigated and a considerable benefit is observed when the clock tree is designed using the proposed tuning system. In addition, the work in [12] proposes an efficient post-silicon tuning method by searching a configuration tree combined with graph pruning, and an insertion algorithm to group buffers into clusters. The yield of a circuit with clock tuning buffers can be evaluated efficiently using the method in [13], and post-silicon testing methods for such circuits have been discussed in [14] and [15].

The methods above are applied as presilicon optimization or post-silicon adjustment before shipping the manufactured chips to customers. Several other methods have exploited these tuning buffers to improve circuit performance and reliability online, i.e., while the circuit is running. The method in [16] adjusts clock skews when the circuit is running according to timing errors to achieve a better performance in timing-speculative circuits. The method in [17] explores the insertion of clock tuning buffers and in-system configuration to reduce performance degradation due to aging. In addition, the method in [18] applies clock tuning buffers to compensate dynamic delay variations induced by temperature.

In order to take advantage of post-silicon tuning, these buffers should be inserted into the circuit during the design phase. Since they take die area and require special treatment during physical design, the number of tuning buffers in the circuit should be small to provide a good yield/profit improvement. This is essentially a statistical optimization problem when process variations are considered. Previous methods [9], [10] 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.

In this paper, we propose a method to determine buffer locations by iterative learning. In each iteration we try to capture the buffers that are important to the yield/profit of the circuit. Afterward, we refine the identified buffer locations and compress buffer ranges to reduce area cost. The contributions of the proposed method are as follows.

1) Instead of searching along a few paths in the problem space to find a set of buffer locations, we use representative sample points to identify the buffers that are important to the yield/profit directly. Using a low-discrepancy sample sequence, the proposed method can identify the proper buffer locations efficiently.
2) We introduce a new way to model yield in representative samples to convert a statistical optimization problem into an ILP problem, so that heuristic statistical optimization can be avoided.
3) We model the overall profit optimization problem instead of the yield at a given clock period. Consequently, the produced method can determine the buffer locations with respect to multiple clock bins in high-performance designs. When only one bin is used, this method is equivalent to the yield improvement problem with respect to a single clock period.
4) The proposed sampling-based method produces tuning values in the representative chip samples. With these values, buffers can be grouped according to their tuning correlation to reduce area cost further.
5) Compared with other methods, the proposed method is much faster, thanks to several acceleration techniques, even when the intermediate sample batches are not parallelized

The rest of this paper is organized as follows. We give an overview of timing constraints for circuits with post-silicon clock tuning buffers in Section II and formulate the buffer allocation problem in Section III. We explain the proposed method in detail in Section IV. Experimental results are shown in Section V. The conclusion and future work are given in Section VI.

II. Timing Constraints With Clock Buffers

In a circuit with post-silicon tuning buffers, the delays of clock paths to flip-flops can be adjusted after manufacturing for each chip individually. The concept of this tuning can be explained using the example in Fig. 2(a), 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

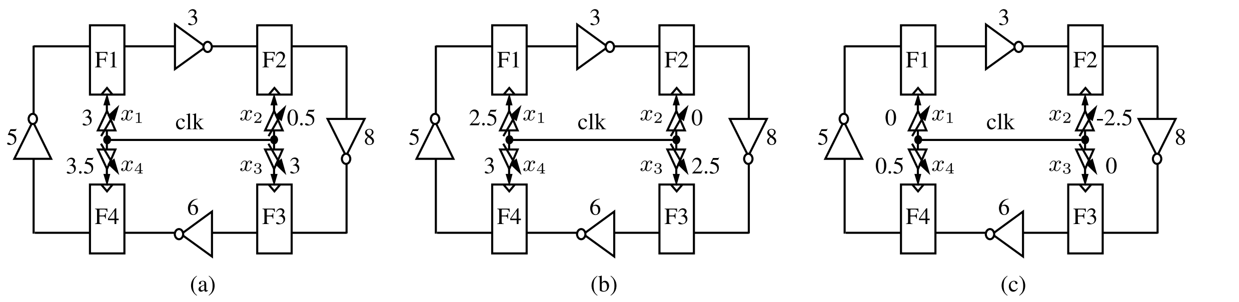


Fig. 2. Performance improvement using post-silicon tuning buffers. Minimum achievable clock period is 5.5. Tuning values in (a) and (b) are constrained in [0, 4]. Setup time and hold time are assumed as 0 for simplicity. (a) Tuning configuration without reduction. (b) Reduced tuning configuration. (c) Reduced tuning configuration with negative tuning values.

value x_3 shifts the launching 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 because the buffer value x_4 moves the clock edge at $F4$ further later.

The timing imbalance between combinational paths as in Fig. 2(a) potentially appears when process variations become large in advanced technology nodes. For an individual chip, this post-silicon clock tuning is similar to the concept of useful clock skews [19]. The difference is that the tuning values are specific to each individual chip after manufacturing, so that the effects of process variations can be dealt with specifically for each chip. If the skew schedule problem in [19] 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.

In Fig. 2(a), four tuning buffers are used. However, 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. This way, the number of buffers can be reduced by one, as shown in Fig. 2(b). 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. 2(c). 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 clockwise direction and counterclockwise direction, so that the number of required buffers can be lowered to reduce area and post-silicon configuration cost. The task of buffer allocation during the design phase is thus to identify the smallest set of buffers with which chips after manufacturing can be tuned to a higher performance.

The timing constraints with clock tuning buffers can be explained using Fig. 3, where two flip-flops with 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.

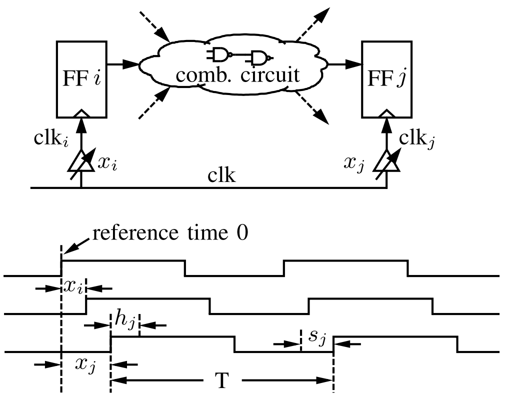


Fig. 3. Timing of circuits with tuning buffers.

To meet the setup time and hold time constraints, the following inequations must be satisfied:

$$x_i + \overline{d}_{ij} \leq x_j + T - s_j \tag{1}$$

$$x_i + \underline{d}_{ij} \geq x_j + h_j \tag{2}$$

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 (1) and (2). Without them, the two inequations fall back to the normal timing constraints of digital circuits.

Owing to area constraints, 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

$$r_i \leq x_i \leq r_i + \tau_i. \tag{3}$$

Unlike [9], 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 limitations.

To guarantee the proper function of a circuit with clock tuning buffers, the constraints (1)–(3) are created for each pair of flip-flops. For a chip after manufacturing, the variables $\overline{d}_{ij}$, $\underline{d}_{ij}$, s_j and h_j in (1) and (2) become fixed values. These delays and timing properties in manufactured chips can be measured using frequency stepping [20], such as in [14], [15], and [21]. A detailed discussion of this technique can be found, e.g., in [21]. After the delays and timing properties are measured, the values of x_i and x_j that make a chip work with a given clock period T can be found easily using the Bellman–Ford algorithm [22] or using linear programming.

In this paper, we only focus on determining buffer locations during the design phase. Consequently, the path delays, setup times and hold times should be considered as random variables modeled using data 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 [23]. For convenience, we assume that a tuning delay can be configured to a fixed value in the following discussion. 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.

III. Problem Formulation

In applying the post-silicon tuning technique, we need to insert the buffers after logic synthesis is finished and before physical design is started. Since buffers take precious die area, and require additional test to configure them, the number of buffers in a design should be limited. 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, it is more important to improve the overall profit of all bins than to improve the yield of the circuit with respect to a single clock period.

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

Input:

1) circuit structure and statistical path delays;
2) buffer specification, including the maximum allowed size τ_i of buffers defined in (3) and the number of discrete steps in the tunable delay range;
3) the maximum number of buffers allowed in the circuit N_b;
4) 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 $T_{m,u} = T_{m+1,l}$.

TABLE I
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

Output:

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

Constraints:

1) for any pair of flip-flops i and j with combinational paths between them, the constraints (1)–(3) hold;
2) the number of buffers inserted in the circuit must not exceed N_b.

Objectives:

1) maximize the overall profit

$$\mathcal{P} = \sum_{m=1}^{N_p} p_m y_m \tag{4}$$

where y_m is the percentage of the chips that are assigned into the mth bin after manufacturing;
2) reduce the sizes of the inserted buffers while maintaining the overall profit $\mathcal{P}$.

In the definition of bins, the first bin has the highest performance, and it has no lower bound for the clock period T, so that $T_{1,l}$ can be set to any value no larger than zero. 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) is very general. If only one bin is used, this problem falls back to the yield improvement problem with respect to a single clock period.

In the problem formulation above, we do not include the number of tuning buffers as a part of the optimization objective, because the relation between profit and the number of buffers is very complex. 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 necessary, however, the number of buffers can also be moved from a constraint into the optimization objective (4), and the proposed method can still work with only a slight modification.

The predominant challenge in solving the optimization problem above comes from the random variables in (1) and (2), because only statistical timing information are available during the design phase when the buffers are allocated. The profit in (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 [21]. Another challenge is that the variables x_i and x_j in (1) and (2) may take only discrete values in the range window defined by (3). For example, the de-skew buffer in [7] 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 (1)–(3) after the random variables are fixed by sampling.

To deal with the 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 [24], for sensor placement in dynamic noise management systems [25], and for parametric yield estimation for analog/mixed signal circuits [26]. In the following section, we will introduce an efficient iterative learning-based method to capture buffer locations for yield improvement.

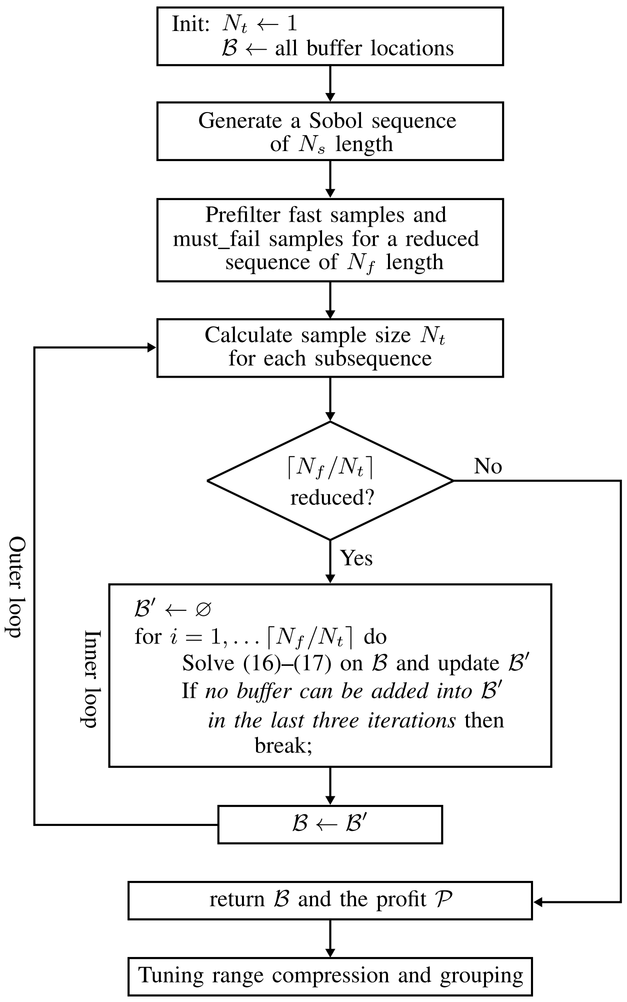
IV. Buffer Allocation Using Representative Sampling

The buffer allocation problem is essentially a statistical optimization problem. In the linear constraints in (1)–(3) the path delays, setup times and hold times are correlated random variables. Instead of using path search or the cutting plane method as in previous methods, we solve this problem using statistical sampling. 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 flow of the proposed method is illustrated in Fig. 4. 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.

A. Sampling-Based ILP Modeling Between Statistical Delays and Profit

Consider the case that we generate N_s samples from the joint distribution of all the random variables in the optimization problem. If N_s is large enough, these samples can actually emulate the chips after manufacturing. If we have tuning buffers at the clock paths to some 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. For each emulated chip we can now evaluate how performance can be improved because the statistical variables in the constraints become fixed in the samples. This way, we can establish the relation between buffer locations and the profit, and use an ILP solver to determine the optimal buffer allocation.



Init: $N_t \leftarrow 1$
$\mathcal{B} \leftarrow$ all buffer locations
Generate a Sobol sequence of N_s length
Prefilter fast samples and must_fail samples for a reduced sequence of N_f length
Calculate sample size N_t for each subsequence
$\lceil N_f/N_t \rceil$ reduced?
No
Yes
Outer loop
Inner loop
$\mathcal{B}' \leftarrow \varnothing$
for $i = 1, \dots \lceil N_f/N_t \rceil$ do
Solve (16)–(17) on $\mathcal{B}$ and update $\mathcal{B}'$
If *no buffer can be added into* $\mathcal{B}'$ *in the last three iterations* then break;
$\mathcal{B} \leftarrow \mathcal{B}'$
return $\mathcal{B}$ and the profit $\mathcal{P}$
Tuning range compression and grouping

Fig. 4. Prefiltering and iterative buffer allocation flow.

For the kth sample from the N_s samples, the constraints (1)–(3) become

$$x_i^k + \overline{d}_{ij}^k \le x_j^k + T^k - s_j^k \tag{5}$$

$$x_i^k + \underline{d}_{ij}^k \ge x_j^k + h_j^k \tag{6}$$

$$r_i \le x_i^k \le r_i + \tau_i \tag{7}$$

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 (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

$$x_i^k = \begin{cases} 0 & \text{if } c_i = 0 \\ \text{any value} \in [r_i, r_i + \tau_i] & \text{when } c_i = 1. \end{cases} \tag{8}$$

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 (8) can thus be transformed to

$$x_i^k \leq c_i \Gamma \tag{9}$$
$$-x_i^k \leq c_i \Gamma \tag{10}$$

where Γ is very large constant. If c_i is set to 0, x_i^k must be set to 0 to meet (9) and (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 (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

$$\sum_i c_i \leq N_b \tag{11}$$

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 evaluate the performance of an emulated chip, 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 meeting $T_{m,l} < T^k \leq T_{m,u}$, 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) 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, $m = 1, \ldots, N_p$ 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

$$g_m^k = 1 \iff T^k \leq T_{m,u}, m = 1, 2, \ldots, N_p. \tag{12}$$

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

$$b_m^k = \begin{cases} g_m^k & m = 1 \\ g_m^k - g_{m-1}^k & m = 2, \ldots, N_p. \end{cases} \tag{13}$$

The constraint (12) can be transformed into

$$T^k - T_{m,u} \leq \left(1 - g_m^k\right)\Gamma, m = 1, 2, \ldots, N_p \tag{14}$$

where Γ is very large positive constant.

The constraints (13) and (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 (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 (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

$$y_m = \sum_{k=1}^{N_s} b_m^k \bigg/ N_s \tag{15}$$

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

$$\begin{aligned} \text{maximize} \quad & \sum_{m=1}^{N_p} p_m y_m & (16) \\ \text{s.t.} \quad & (5)\text{–}(7),\ (9)\text{–}(11),\ \text{and } (13)\text{–}(15) \\ & \text{with respect to all flip-flops pair indexed by } (i,j) \\ & \text{and } k = 1, \ldots, N_s. & (17) \end{aligned}$$

The basic idea of this formulation is that 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 in this simulated scenario to determine which flip-flops should have 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 [9], [10], the relation between buffer locations and yield is not analyzed directly. Instead, these methods consider this relation as a separate evaluation problem, 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, resulting in a large runtime.

If the number of emulated samples N_s in the integer linear optimization problem (16), (17) is large enough, the profit can be modeled accurately and the values of c_i in the solution indicate the optimal locations to insert tuning buffers for the maximum profit. However, a large N_s may increase the number of constraints in (17) to the degree that the size of the ILP problem exceeds the capacity of all existing ILP solvers. To deal with this scalability problem, we apply two techniques.

1) We reduce the number of emulated samples N_s by using a low-discrepancy sample sequence instead of a purely random sampling sequence.
2) We split the problem (16), (17) into subsets and use them to learn the locations of buffers iteratively.

After each iteration, the candidates of buffer locations can be refined.

B. Reducing the Number of Emulation Samples Using Low-Discrepancy Sequence

The sampling-based concept above requires a large number of samples to guarantee the quality of the resulting buffer locations. Consider the extreme case where we use only two samples, which have different probabilities to appear in the manufactured chips. In the formulation (16), (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. 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 (16) and (17) implicitly, because samples from regions with large probabilities in the problem space appear more often than samples from other regions. Another way to solve this discrepancy problem is to use the probability of representative samples as further coefficients of the yield values in the objective (16) directly. But it is not clear how many samples should be generated to guarantee the quality of the result.

The third method to solve the problem of a large sampling number is to use a low-discrepancy sequence such as studied in [27]. 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 [28]. In this paper, we use the Sobol sequence in [29] to reduce the number of samples N_s. The effect of this sequence can be demonstrated using the examples in Fig. 5, where Fig. 5(a) shows a purely random number sequence of 256 samples for two uniform-distributed variables. Fig. 5(b) 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 [30]. In our method, we use 1000 samples in the Sobol sequence, which are one tenth of the usually used 10 000 samples of random variables in SSTA [2]. In practice, test cases can converge even earlier with fewer than 1000 samples.

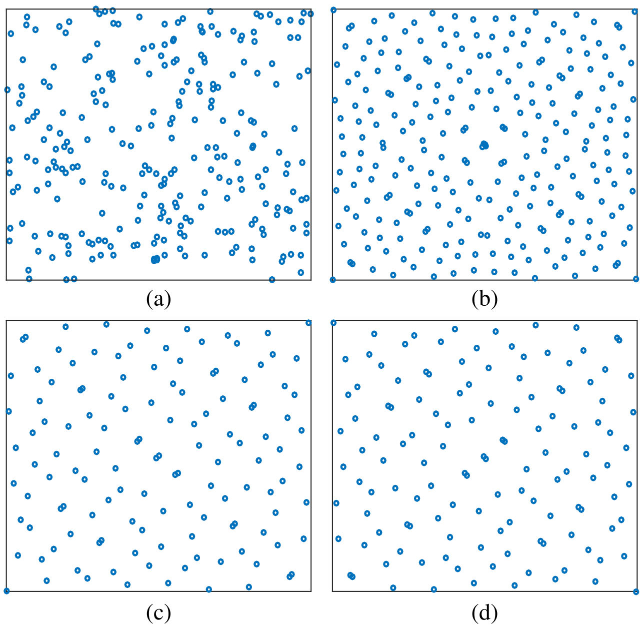


Fig. 5. Purely random sequence and Sobol low-discrepancy sequence. Two hundred fifty-six random samples of (a) two uniform variables and (b) Sobol sequence for two uniform variables. (c) First 128 samples from the Sobol sequence in (b). (d) Next 128 samples from the Sobol sequence in (b).

C. Buffer Allocation With Prefiltering and Iterative Learning

In the N_s samples, some might be fast enough to be assigned into the fastest bin without tuning; others might be too slow to be tuned into the slowest bin, even with all flop-flops connected with tuning buffers. In both scenarios, tuning buffers play no role in improving the overall profit. Therefore, we exclude these samples from the ILP formulation (16), (17) to reduce the number of variables and constraints.

To filter out the samples of the first type, we need only to set all values of tuning buffers, x_i^k and x_j^k in (5) and (6) to 0, and calculate the clock period $T^k_{\min}$ for this sample as $T^k_{\min} = \max_{i,j}\{\overline{d}^k_{ij}+s^k_j\}$. If $T^k_{\min}$ is smaller than the upper bound of the fastest bin, this sample is fast enough and no tuning is required. The constraint (6) is checked similarly. If all these constraints can be met without tuning buffers, this sample is excluded.

To filter out the samples that are too slow to be assigned to a bin even with extensive buffer tuning, 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 (5), the sum of the path delay $\overline{d}^k_{ij}+s^k_j$ 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 sample can be assigned to one of the bins and the corresponding sample is not included in the profit optimization problem. We repeat this prefiltering checking using (6) to exclude samples that do not work in any case due to unavoidable hold time violations.

After prefiltering, the remaining samples are used to determine buffer locations by solving the optimization problem (16), (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 (16), (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. 5(c) and (d) and 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.

The first fact above shows that we may solve the ILP problem (16), (17) with only a part of the Sobol sequence, meaning 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 (16), (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 [31] 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 in these subsequences because of the small number of samples in one subsequence, as shown in Fig. 5(c) and (d). Consequently, a subsequence with a limited number of samples may not capture all the buffer locations. We alleviate this problem by combining the buffer locations captured by different subsequences into a buffer set $\mathcal{B}'$. Once we finish solving (16) and (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. In this loop, we also relax the number of buffers from N_b to βN_b in the constraint (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. In these iterations, only the buffer candidates in $\mathcal{B}$ need to be modeled with variables c_i as in (8) and only the delays of paths connected to these buffer candidates need to be sampled as (5)–(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.

D. Reducing Buffer Area by Tuning Concentration and Grouping

The iterative optimization flow in Fig. 4 only determines the locations to insert buffers for profit improvement after manufacturing. But the sizes of the buffers are not addressed. In this section, we introduce a method to concentrate tuning values toward each other and to group buffers thereafter.

The concept of area reduction can be explained using Fig. 6. After executing the iterative buffer allocation in Fig. 4, the tuning values of a buffer in all samples may be scattered in a wide range such as in Fig. 6(a), because the solver only minimizes the number of buffers, but does not consider the relation between the tuning values 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. 6(c). 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. 6(a). 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. This process is formulated as follows:

$$\text{minimize} \quad \sum_{i \in I_{\mathcal{B}},k} |x_i^k| \tag{18}$$

$$\begin{aligned} \text{s.t.} \quad & (5)\text{–}(7),\ (9)\text{–}(11),\ \text{and } (13)\text{–}(15) \\ & \text{with respect to all flip-flops pair indexed by } (i,j) \\ & \text{and } k = 1, \ldots N_f, \text{ and} \end{aligned} \tag{19}$$

$$\sum_{m=1}^{N_p} p_m y_m \geq \mathcal{P} \tag{20}$$

where $I_{\mathcal{B}}$ is the index set of all buffer locations in $\mathcal{B}$. The objective function (18) can be transformed into a linear form easily as explained in [32].

The difference between the optimization problem (18)–(20) and the optimization problem (16), (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; and 3) the profit becomes a constraint to guarantee the tuning range concentration does not affect the profit. By solving the problem (18)–(20), all tuning values are pushed toward zero as illustrated in Fig. 6(b), so that the buffer ranges become more compact.

Another 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 built in the circuit and the delayed

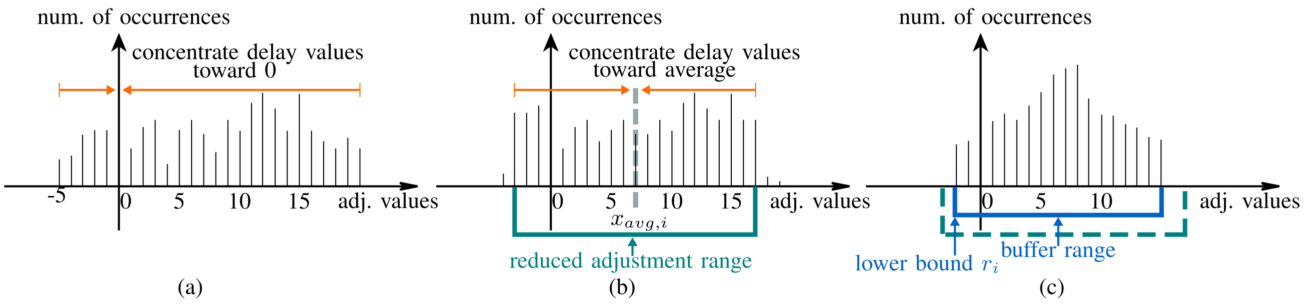


Fig. 6. Concentrating tuning values of a buffer in all samples. The x-axis represents the adjusted delays of the buffer in all samples, and the y-axis the number of occurrences of the discrete delay values. (a) Scattered tuning values. (b) Tuning values concentrated toward zero. (c) Reduced buffer range after concentrating tuning values toward the average.

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 (18)–(20). Afterward, 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. 6(c). This step is very similar to the problem formulation in (18)–(20), except that the optimization objective is replaced by

$$\text{minimize} \quad \sum_{i\in I_{\mathcal{B}},k} \left| x_i^k - x_{\text{avg},i} \right| \tag{21}$$

where $x_{\text{avg},i}$ is the weighted average of all tuning values calculated from the result of solving (18)–(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 (3). As shown in Fig. 6(c), 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, meaning 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 buffer size τ_i and lower bound r_i in (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. 7.

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.

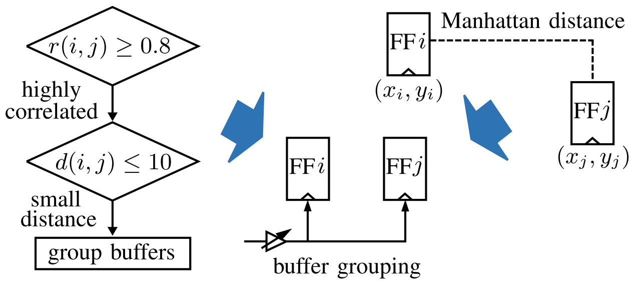


Fig. 7. Buffer grouping according to tuning correlation and distance. Correlation threshold $r(i, j)$ is set to 0.8. Distance threshold $d(i, j)$ between buffers is set to ten times of the minimum distance between flip-flops.

E. 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 (1) and (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 (1) and (2). For example, in the constraint (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 (2) according to the -3σ values of path delays.

V. Experimental Results

The proposed method was implemented in C++ and tested using a 3.20-GHz CPU with one thread. We demonstrate the results using circuits from the ISCAS89 benchmark set and from the TAU 2013 variation-aware timing analysis contest.

TABLE II
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%			

The number of flip-flops and the number of logic gates are shown in the columns n_s and n_g in Table II, respectively.

The benchmark circuits in our experiments were sized using a 45-nm library. We assumed that the maximum allowed buffer ranges were 1/8 of the original clock period and tuning delays of the buffers were discrete with 20 steps, as in [7]. The standard deviations of transistor length, transistor width and oxide thickness were set to 15.7%, 11.1%, and 5.3% of the nominal values, respectively. We used Gurobi [31] to solve the optimization problems in the proposed method.

We used three bins 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, we ran Monte Carlo simulation with these circuits to verify the yield improvement. In the simulation, we generated 10 000 samples. For each sample we calculated its minimum clock period 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 our experiments are conceptually different from the samples discussed in Section IV, because they were only used to emulate post-silicon measurements. For each sample, we verify whether a chip can be assigned into a bin by solving the classical skew scheduling problem in [19]. 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 [21]. 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 II, 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 $\mathcal{Y}_{\text{inc}}$, with an average 4.23%.

With these yield values in the three bins, we can calculate the profit using (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 $\mathcal{P}_{\text{inc}}$, with an average 14.29%. If we compare the column $\mathcal{P}_{\text{inc}}$ and the column $\mathcal{Y}_{\text{inc}}$, we can see that the 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 ten 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 II. The largest execution time of the proposed method is 1816.81 s, 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 II. The relation between the number of samples in a batch and the runtime is illustrated in Fig. 8. *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

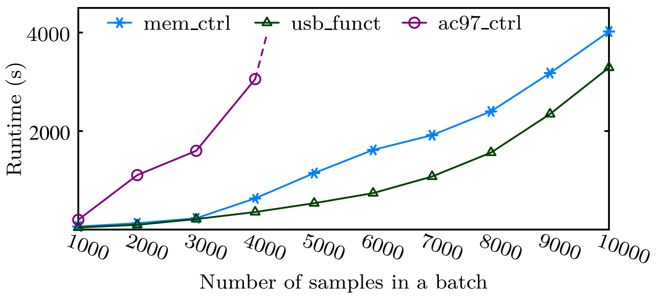


Fig. 8. Scalability trend of the proposed method with a fixed number of samples in each batch.

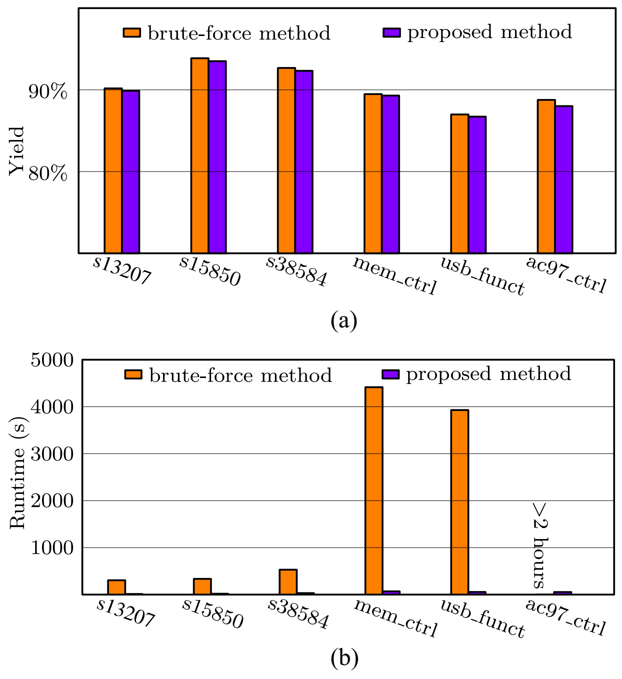


Fig. 9. Yield and runtime comparison between the proposed method and the brute-force method with 10 000 samples. (a) Yield comparison. (b) Runtime comparison.

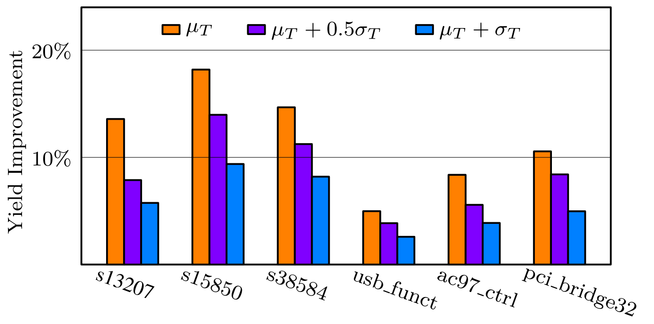


Fig. 10. Yield improvement with clock tuning buffers with respect to μ_T, $\mu_T + 0.5\sigma_T$, and $\mu_T + \sigma_T$, compared with the yield values without tuning buffers.

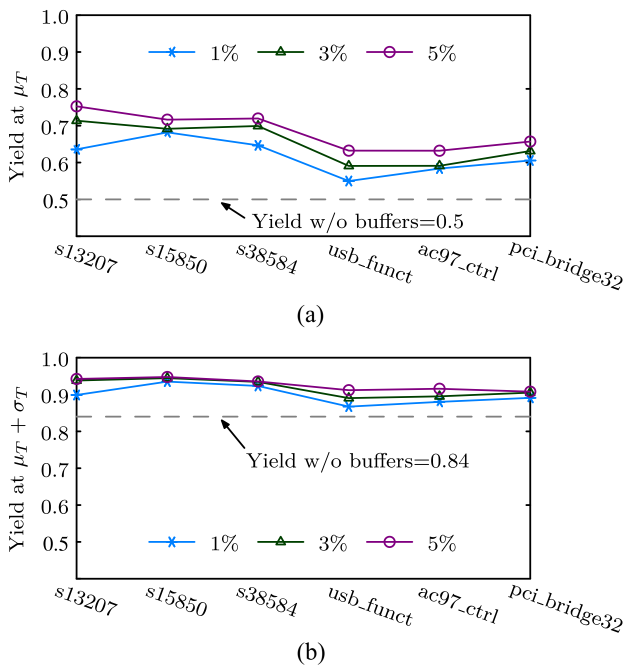


Fig. 11. Yield increase with respect to different numbers of tuning buffers. Target clock period is set to (a) μ_T and (b) $\mu_T + \sigma_T$.

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 IV-C. 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. 9(a). 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 first tries to capture all the buffer locations that have a potential to affect the yield. Afterward, 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. 9(b).

In the profit definition (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. 10. 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 is a large potential for the tuning buffers to take effect.

In our 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

TABLE III
RUNTIME COMPARISON WITHOUT AND WITH 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

TABLE IV
YIELD COMPARISON WITH [9]

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%

and $\mu_T + \sigma_T$, respectively. The results are shown in Fig. 11. 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. 10, the yield improvement with respect to μ_T in Fig. 11(a) is more obvious compared with the yield improvement with respect to $\mu_T + \sigma_T$ in Fig. 11(b). For the former, the average improvement of the 5% setting to the 1% setting is 6.78%, but for the latter this improvement is 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, we introduced several acceleration techniques. With the Sobol sequence, the inner loop of the iterative flow in Fig. 4 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 III. 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 [9] 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 IV. The N_1 column shows the number of buffers in [9], 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 [9] with a higher yield, while the number of clock tuning buffers is much smaller. Furthermore, we have implemented the method in [12] and the yield comparison is shown in Fig. 12. In this comparison, the numbers of inserted buffers are equal, so that we can conclude that the proposed method outperforms the method in [12] 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 [12], a grouping algorithm is also proposed according to circuit structure and distances between flip-flops. We compare the results of our correlation-based grouping method with theirs and the results are shown in Table V, where Y_1 is the yield with the grouping algorithm in [12] 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 [12]. 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 [1] uses the same concept in this paper, 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 incorporated.

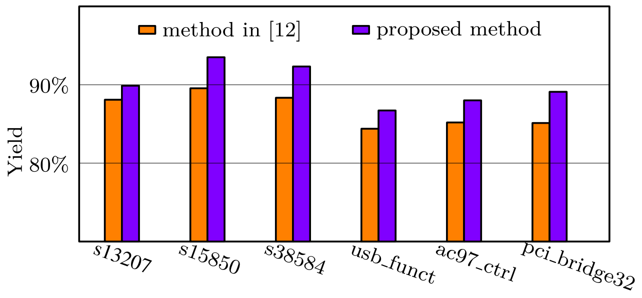


Fig. 12. Yield comparison with the buffer insertion method in [12].

TABLE V
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%

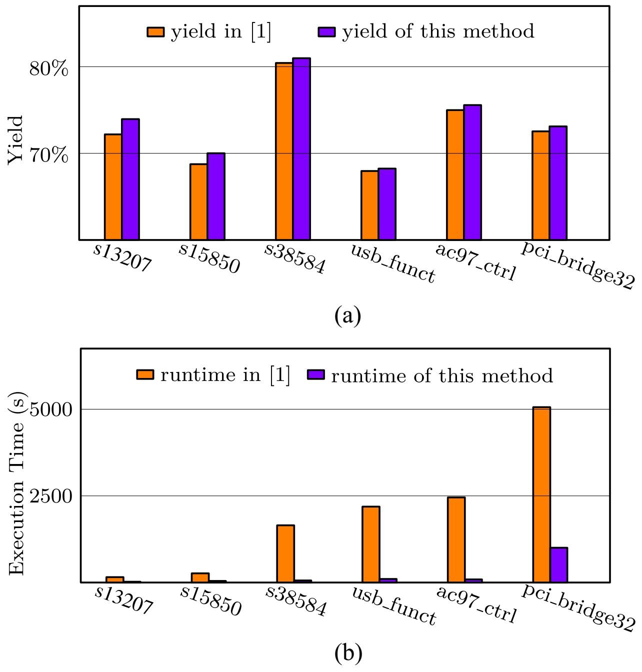


Fig. 13. Comparison with [1]. (a) Yield improvement with the same setting. (b) Comparison of execution time of both methods.

In addition, the method in [1] 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 [1] to the same library and tested the yield improvement with respect to μ_T. The results are shown in Fig. 13(a), where we can see that the proposed method produces a better yield improvement than [1] with the same number of buffers. Furthermore, we show the execution time of these methods in Fig. 13(b). It is clear that the extended method in this paper is more efficient than [1].

VI. Conclusion

In this paper, we propose a sampling-based method 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. Future tasks of this paper include post-silicon testing and configuration of delays buffers to achieve the given clock period or profit. The major challenge is to make a good tradeoff between test cost and profit improvement.

References

[1] G. L. Zhang, B. Li, and U. Schlichtmann, "Sampling-based buffer insertion for post-silicon yield improvement under process variability," in *Proc. Design Autom. Test Europe Conf.*, Dresden, Germany, 2016, pp. 1457–1460.

[2] 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.*, vol. 27, no. 4, pp. 589–607, Apr. 2008.

[3] D. Ernst *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. Int. Symp. Microarchitect.*, San Diego, CA, USA, 2003, pp. 7–18.

[4] S. Naffziger *et al.*, "The implementation of a 2-core, multi-threaded Itanium family processor," *IEEE J. Solid-State Circuits*, vol. 41, no. 1, pp. 197–209, Jan. 2006.

[5] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi, "Post-fabrication clock-timing adjustment using genetic algorithms," *IEEE J. Solid-State Circuits*, vol. 39, no. 4, pp. 643–650, Apr. 2004.

[6] 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.*, 2005, pp. 292–293.

[7] S. Tam *et al.*, "Clock generation and distribution for the first IA-64 microprocessor," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1545–1552, Nov. 2000.

[8] J.-L. Tsai, D. Baik, C. C.-P. Chen, and K. K. Saluja, "A yield improvement methodology using pre- and post-silicon statistical clock scheduling," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2004, pp. 611–618.

[9] J.-L. Tsai, L. Zhang, and C. C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2005, pp. 575–581.

[10] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and postsilicon tunability allocation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 4, pp. 610–620, Apr. 2008.

[11] 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. Test Europe Conf.*, 2009, pp. 292–295.

[12] Z. Lak and N. Nicolici, "A novel algorithmic approach to aid post-silicon delay measurement and clock tuning," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1074–1084, May 2014.

[13] 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.*, vol. 34, no. 11, pp. 1784–1797, Nov. 2015.

[14] 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.*, 2008, pp. 1–8.

[15] D. Tadesse, J. Grodstein, and R. I. Bahar, "AutoRex: An automated post-silicon clock tuning tool," in *Proc. Int. Test Conf.*, Austin, TX, USA, 2009, pp. 1–10.

[16] R. Ye, F. Yuan, and Q. Xu, "Online clock skew tuning for timing speculation," in *Proc. Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2011, pp. 442–447.

[17] 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.*, vol. 31, no. 12, pp. 1845–1856, Dec. 2012.

[18] A. Chakraborty *et al.*, "Dynamic thermal clock skew compensation using tunable delay buffers," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 6, pp. 639–649, Jun. 2008.

[19] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[20] K. S. Kim, S. Mitra, and P. G. Ryan, "Delay defect characteristics and testing strategies," *IEEE Design Test Comput.*, vol. 20, no. 5, pp. 8–16, Sep./Oct. 2003.

[21] 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.*, Austin, TX, USA, 2016, pp. 1–6.

[22] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 1990.

[23] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *Proc. Design Autom. Conf.*, 2004, pp. 331–336.

[24] J. Xiong, Y. Shi, V. Zolotov, and C. Visweswariah, "Statistical multilayer process space coverage for at-speed test," in *Proc. Design Autom. Conf.*, San Francisco, CA, USA, 2009, pp. 340–345.

[25] 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 Design*, San Jose, CA, USA, 2013, pp. 437–443.

[26] F. Gong, Y. Shi, H. Yu, and L. He, "Variability-aware parametric yield estimation for analog/mixed-signal circuits: Concepts, algorithms, and challenges," *IEEE Design Test*, vol. 31, no. 4, pp. 6–15, Aug. 2014.

[27] 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. Qual. Electron. Design*, San Jose, CA, USA, 2007, pp. 685–692.

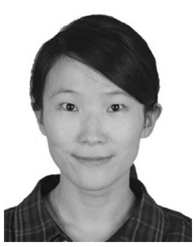
[28] V. Veetil, K. Chopra, D. Blaauw, and D. Sylvester, "Fast statistical static timing analysis using smart Monte Carlo techniques," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 6, pp. 852–865, Jun. 2011.

[29] I. M. Sobol, "The distribution of points in a cube and the approximate evaluation of integrals," *USSR Comput. Math. Math. Phys.*, vol. 7, no. 4, pp. 86–112, 1967.

[30] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Stat.*, vol. 29, no. 2, pp. 610–611, 1958.

[31] Gurobi Optimization, Inc. (2013). *Gurobi Optimizer Reference Manual*. [Online]. Available: http://www.gurobi.com

[32] D. Chen, R. G. Batson, and Y. Dang, *Applied Integer Programming: Modeling and Solution*. Hoboken, NJ, USA: Wiley, 2011.



Grace Li Zhang received the master's degree from the School of Microelectronics, Xidian University, Xi'an, China, in 2014. She is currently pursuing the Ph.D. degree with the Institute for Electronic Design Automation, Technical University of Munich, Munich, Germany.

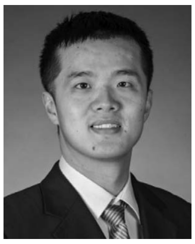
Her current research interests include high-performance and lower-power design, and emerging systems.

Bing Li received the bachelor's and master's degrees in communication and information engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2000 and 2003, respectively, and the Dr.-Ing. degree in electrical engineering from the Technical University of Munich (TUM), Munich, Germany, in 2010.

He is currently a Researcher with the Institute for Electronic Design Automation, TUM. His current research interests include high-performance and lower-power design, and emerging systems.

Jinglan Liu received the B.E. degree in communication engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2014. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA.

Her current research interests include low-power system design and machine learning applications on interdisciplinary fields.



Yiyu Shi (M'09–SM'14) received the B.S. (Hons.) degree in electronic engineering from Tsinghua University, Beijing, China, in 2005, and the M.S. and Ph.D. degrees in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 2007 and 2009, respectively.

He is currently an Associate Professor with the Departments of Computer Science and Engineering and Electrical Engineering, University of Notre Dame, Notre Dame, IN, USA. His current research interests include 3-D integrated circuits and machine learning on chips.

Dr. Shi was a recipient of several best paper nominations in top conferences, the IBM Invention Achievement Award in 2009, the Japan Society for the Promotion of Science Faculty Invitation Fellowship, the Humboldt Research Fellowship for Experienced Researchers, the IEEE St. Louis Section Outstanding Educator Award, Academy of Science (St. Louis) Innovation Award, the Missouri S&T Faculty Excellence Award, the National Science Foundation CAREER Award, the IEEE Region 5 Outstanding Individual Achievement Award, and the Air Force Summer Faculty Fellowship.

Ulf Schlichtmann (S'88–M'90) received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering and information technology from the Technical University of Munich (TUM), Munich, Germany, in 1990 and 1995, respectively.

He was with Siemens AG, Munich, and Infineon Technologies AG, Munich, from 1994 to 2003, where he held various technical and management positions in design automation, design libraries, IP reuse, and product development. He has been a Professor and the Head of the Institute for Electronic Design Automation, TUM, since 2003, where he served as the Dean of the Department of Electrical and Computer Engineering, from 2008 to 2011. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems.

5 Efficient Delay Test and Prediction for Post-Silicon Clock Skew Configuration under Process Variations

EffiTest2: Efficient Delay Test and Prediction for Post-Silicon Clock Skew Configuration under Process Variations

Grace Li Zhang, Bing Li, Yiyu Shi *Senior Member, IEEE*, Jiang Hu *Fellow, IEEE* and Ulf Schlichtmann *Member, IEEE*

Abstract—**At nanometer manufacturing technology nodes, process variations affect circuit performance significantly. This trend leads to a large timing margin and thus overdesign in the traditional worst-case circuit design flow. To combat this pessimism, post-silicon clock tuning buffers can be deployed to balance timing slacks of consecutive combinational paths in individual chips by tuning clock skews after manufacturing. A challenge of this method is that path delays of each chip with timing failures should be measured to gather the information for clock skew configuration. However, current methods for delay measurement rely on path-wise frequency stepping, which requires much time from expensive testers. In this paper, we propose an efficient delay test framework (EffiTest2) to solve the post-silicon testing problem by testing only representative paths with delay alignment using the already-existing tunable buffers in the circuit. Experimental results demonstrate that EffiTest2 can reduce the number of frequency stepping iterations by more than 94% with only a slight yield loss.**

Index Terms—**Process Variations, Post-Silicon Tuning, Clock Skew, Yield, Path Selection, Delay Test, Statistical Prediction**

I. Introduction

Modern IC design faces tremendous challenges to achieve performance goals while maintaining a profitable yield. For example, at advanced technology nodes, increasing process variations together with aging effects require a very large timing margin, thus causing expensive overdesign. To combat such challenges, process variations may be modeled directly in timing analysis, leading to a boom of research on statistical static timing analysis (SSTA) in the last decade [2]–[11]. With the information of distributions of process variations, SSTA methods produce a performance-yield curve with which designers have a chance to make a tradeoff between different design goals. This method can effectively reduce the timing margin compared with traditional worst-case design, but it still does not counter process variations actively. To alleviate the effect of process variations, many researchers have also worked on circuit level to introduce special devices and mechanisms. For instance, the Razor method [12]–[15] boosts circuit performance until timing errors occur, and recently the performance capacity of flip-flops has been exploited to the extreme limit by considering the interdependency between setup and hold time [16]–[21].

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre “Invasive Computing” (SFB/TR 89) as well as NSF (CCF-1525749, CNS-1618824) and SRC (2016-TS-2688).

A preliminary version of this paper was published in the Proceedings of Design Automation Conference (DAC), 2016 [1].

Grace Li Zhang, Bing Li, and Ulf Schlichtmann are with the Chair of Electronic Design Automation, Technical University of Munich (TUM), Munich 80333, Germany (e-mail: grace-li.zhang@tum.de; b.li@tum.de; ulf.schlichtmann@tum.de).

Yiyu Shi is with the Department of Computer Science and Engineering, University of Notre Dame (e-mail: yshi4@nd.edu).

Jiang Hu is with the Department of Electrical and Computer Engineering, Texas A&M University (e-mail: jianghu@tamu.edu).

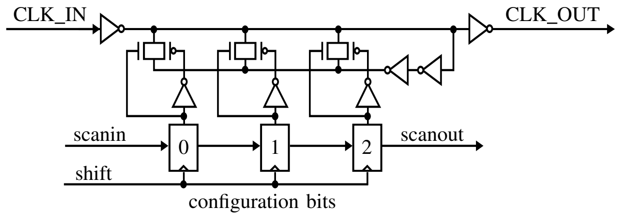


Fig. 1: Post-silicon delay tunable buffer in [25].

Another direction to deal with process variations is to tune chips after manufacturing. 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 with various structures [22]–[25]. An example of industrial applications of this technique is demonstrated in [25]. The structure of the delay buffer (clock vernier device) in this work is illustrated in Fig. 1, 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 create different clock skews to these flip-flops. The objective of this tuning is to allot critical paths more timing slack by shifting clock edges toward stages with smaller combinational delays, so that a manufactured chip can work at the designated frequency.

To apply the post-silicon tuning technique, tunable delay buffers must be inserted into the circuit during the design phase. In recent years, several methods have been proposed to determine buffer locations and evaluate the potential resulting yield improvement. In [26] a clock scheduling method is developed and tunable buffers are selectively inserted to balance the skews resulting from process variations. In [27] the buffer allocation problem is solved with a graph-based algorithm under useful clock skew scheduling. In [28] 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. This problem is investigated further in [29], [30] with a sampling-based method to recognize a limited number of locations to insert tunable buffers for yield improvement. In [31] yield loss due to process variations and the total cost of tunable buffers are formulated together for gate sizing. In [32], 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 [33], [34].

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 [35] an efficient post-silicon tuning method is proposed to search a configuration tree together with graph pruning and buffer grouping. The methods in [36], [37] 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 [38], [39]. Moreover, the method in [40] applies tunable buffers to compensate dynamic delay uncertainty induced by temperature variations.

In applying post-silicon clock tuning, a major challenge is that delays of combinational paths need to be measured specifically for each chip after manufacturing. However, so far this measurement is still performed by applying frequency stepping to individual paths [35]–[37], which requires much time from expensive testers.

In this paper, we investigate the post-silicon test problem and propose an efficient framework (EffiTest2) to improve test efficiency using statistical prediction and delay alignment. Our contributions are as follows.

1) A path selection approach combining SVD/QRcp decomposition and iterative accuracy evaluation is proposed to choose the paths for post-silicon test. The delays of these paths are used to predict the maximum delays between flip-flops with tunable buffers.

2) Multiple paths are tested in parallel in our framework. The delays of paths in a test batch are aligned statistically during path assignment. Nonrepresentative paths with large random variations are added into test batches to reduce inaccuracy in delay estimation. During delay test, we also adjust the already-existing tunable buffers to align the real delays of paths adaptively so that a frequency step can capture delay information of multiple paths.

3) Since predicted path delays are still in the form of small ranges instead of exact numbers, configuration values of tunable buffers are determined with respect to the upper bounds of these ranges, so that potential timing violations due to the inaccuracy in delay test and prediction can be reduced.

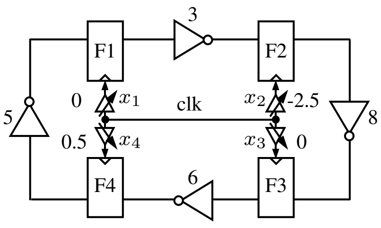


Fig. 2: Post-silicon clock tuning reduces the minimum clock period from 8 to 5.5. Setup time and propagation delay of flip-flops are assumed as 0.

4) Hold time constraints are incorporated by imposing range constraints on configuration values of tunable buffers. These additional constraints limit yield loss due to hold time constraints by a given threshold to avoid further test iterations on short paths.

The rest of this paper is organized as follows. In Section II we give an overview of timing constraints for circuits with post-silicon tunable buffers. We explain the proposed method in detail in Section III. Experimental results are shown in Section IV. Conclusions are drawn in Section V.

II. Background of Post-silicon Clock Tuning

In a circuit with post-silicon tunable buffers, the propagation delays of clock paths to flip-flops with tunable buffers can be adjusted after manufacturing for each chip individually. The concept of this technique can be explained using the example in Fig. 2, where four flip-flops are connected into a loop by combinational paths represented by inverters. The numbers next to the inverters denote delays of the corresponding combinational paths. During the design phase, these combinational delays should be considered as statistical due to process variations. But after manufacturing, the delays in a single chip become fixed values, enabling a concrete clock skew tuning to counter the effect of process variations.

In Fig. 2, if all the delays of the tunable buffers are set to 0, this example is equivalent to the case without post-silicon tuning. The minimum clock period is thus equal to 8 when setup time and propagation delays of the flip-flops are assumed as 0. On the other hand, if clock edges can be moved by adjusting the delays of the tunable 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 2.5 units earlier. Therefore, with a clock period of 5.5, the combinational path between F2 and F3 now has 5.5+2.5=8 time units to finish signal propagation. This shifting of the clock edge reduces the timing budget of the path between F1 and F2 to 5.5-2.5=3 units after post-silicon tuning, which is still sufficient for this path without violating any timing constraint. Note that the buffer delays are defined with respect to a reference clock signal, so that they can have negative values.

This clock tuning concept is similar to assigning useful skews [41] to improve circuit performance. The difference, however, is that the skew scheduling is executed individually for each chip with timing failures after manufacturing. Since critical paths in these chips may differ due to process variations, customized timing schemes generated in response to the

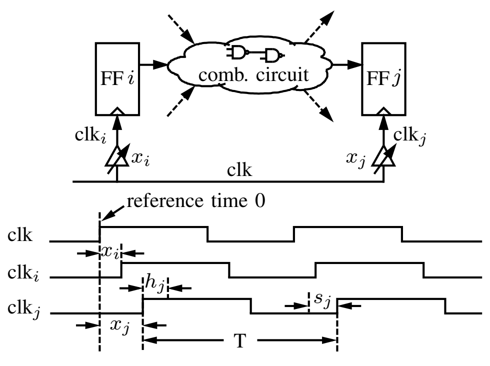


Fig. 3: Timing with tunable buffers.

effect of process variations can rescue these chips by balancing slacks across consecutive register stages.

Timing constraints with clock tunable buffers can be explained using Fig. 3, where two flip-flops with such buffers are connected by a combinational circuit. Assume that the clock signal switches at reference time 0. The clock events at flip-flops i and j happen at time x_i and x_j, respectively. To meet the setup time and hold time constraints, the following constraints must be satisfied

$$x_i + \overline{d}_{ij} \le x_j + T - s_j \iff T \ge D_{ij} + x_i - x_j \quad (1)$$
$$x_i + \underline{d}_{ij} \ge x_j + h_j \iff x_i - x_j \ge d_{ij} \quad (2)$$

where x_i and x_j are delay values of tunable 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, T is the clock period, $D_{ij} = \overline{d}_{ij} + s_j$, and $d_{ij} = h_j - \underline{d}_{ij}$. The two constraints above indicate that the clock tuning values x_i and x_j should be determined according to D_{ij} and d_{ij}. In delay test, the latter two values are actually measured instead of the path delays $\overline{d}_{ij}$ and $\underline{d}_{ij}$. In the following, we will also refer to D_{ij} and d_{ij} as maximum delay and minimum delay for simplicity.

Owing to area cost, the configurable delay of a clock buffer usually has a limited range. For buffer i, this range is specified as

$$r_i \le x_i \le r_i + \tau_i \quad (3)$$

where r_i and τ_i are constants determined by methods such as [28]. In the range (3), x_i may only take discrete values according to the implementation of buffers.

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 meeting the constraints (1)–(3) 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 previous methods [26], [35]–[37], 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.

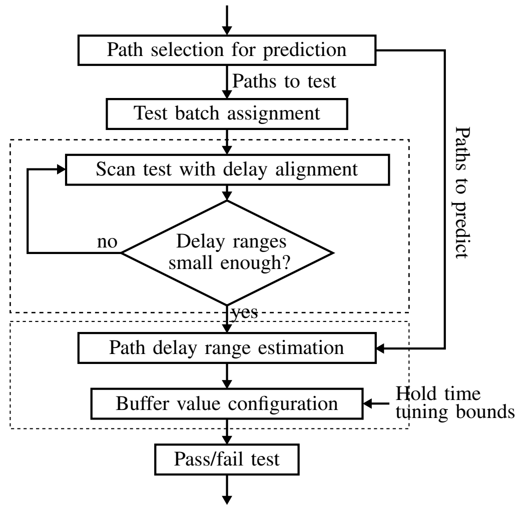

Path selection for prediction
Paths to test
Test batch assignment
Scan test with delay alignment
no
Delay ranges small enough?
yes
Path delay range estimation
Buffer value configuration
Hold time tuning bounds
Pass/fail test
Paths to predict

Fig. 4: Delay test and buffer configuration flow in EffiTest2.

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. 2 could be set to values as shown, the delays of the combinational paths are well balanced and can thus be tested concurrently. To reduce test cost further, the correlation information between path delays provided by statistical timing analysis techniques [11] can also be used. Consequently, only a set of representative paths need to be tested while the maximum delays between flip-flops with tunable buffers are estimated from the test results.

III. Statistical Prediction and Aligned Delay Test for Buffer Configuration

In post-silicon delay test, previous frequency stepping methods test all paths connected to flip-flops with post-silicon tunable buffers individually. According to the test results, the configuration bits of the tunable buffers are adjusted to make the chips work with the required clock period. This exhaustive path-wise test strategy is very expensive because it requires a lot time from testers. Practically, however, not all paths delays need to be evaluated exactly. Instead, they may only need to be estimated with a given accuracy that is sufficient for post-silicon configuration.

In this section, we introduce our method EffiTest2 to reduce the total number of frequency stepping iterations in testing path delays with two techniques: statistical prediction and delay alignment during test. With the tested and estimated delays, tunable buffers in chips with timing failures are then

TABLE I: 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
$\mathbf{D}^p_t$	The statistical delays of combinational paths in $\mathbf{P}_t$
$\mathbf{D}^m_t$	The selected maximum delays that can predict $\mathbf{D}^m \backslash \mathbf{D}^m_t$ within a given accuracy
$\mathbf{P}_c$	The chosen combinational paths for maximum delays in $\mathbf{D}^m_t$
$\mathbf{D}_c$	The statistical delays of combinational paths in $\mathbf{P}_c$

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 such as [28], [30]. The flow of the proposed method is summarized in Fig. 4. It includes four major steps: path selection in Section III-A, test batch assignment in Section III-B, frequency stepping with delay alignment in Section III-C, buffer configuration in Section III-D, and hold time constraints in Section III-E. The important notations used in the following are listed in Table I.

The statistical delay prediction in Section III-A assumes that path delays follow Gaussian distribution. In cases such as ultra-low voltage designs, this assumption may be invalid. In this scenario, the accuracy evaluation (6)–(7) needs to be adapted accordingly while the overall flow can still be deployed.

A. Path Selection and Statistical Delay Prediction

When tuning manufactured chips to resolve timing failures, the maximum delays between flip-flop pairs need to meet setup time constraints, and the minimum delays hold time constraints, as defined in (1)–(2). Fig. 5 illustrates an example, considering only setup time constraints. In this example, nodes represent flip-flops, 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 slack, 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 (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 [26], [35]–[37]. Instead, statistical delay prediction can be deployed to estimate the maximum delays between flip-flops using the data of representative combinational paths.

1) Concept of path selection for delay prediction: Statistical delay prediction relies on the correlation between path delays to maintain a high accuracy. Since a high correlation indicates that two delays vary similarly in manufactured chips, the measurement of one delay after manufacturing also

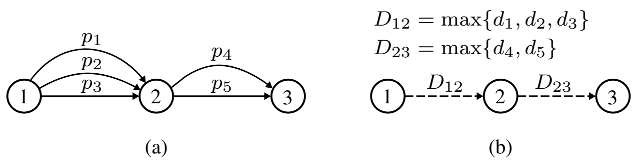


Fig. 5: Test scenario with maximum path delays, where nodes represent flip-flops with tunable buffers. Multiple combinational paths, p_1–p_5 with delays d_1–d_5, respectively, exist between flip-flops with tunable buffers in (a). Post-silicon skew configuration by tunable buffers is determined by the maximum delays of all paths as simplified in (b).

discloses information about the other. In high-performance designs, logic gates 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 [11], which can be exploited to reduce the number of paths to be tested. For example, a conditional statistical prediction technique [42] has been used in [43] 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 $\mathbf{D}^p_t$. The statistical prediction from $\mathbf{D}^p_t$ to $\mathbf{D}^p \backslash \mathbf{D}^p_t$ is a well-known problem and has been studied extensively, e.g., in [44].

In post-silicon tuning, the individual delays of paths in $\mathbf{D}^p$, however, are not required. Instead, only the maximum delays between each pair of flip-flops should be determined as illustrated in Fig. 5(b). When process variations are considered, path delays are represented as random variables. The maximum of a set of path delays 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 $\mathbf{D}^p_t$ of selected combinational paths $\mathbf{P}_t \subset \mathbf{P}$ to $\mathbf{D}^m$, i.e., $\mathbf{D}^p_t \rightarrow \mathbf{D}^m$.

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, we introduce a second level of statistical prediction. Instead of predicting the maximum delays $\mathbf{D}^m$, we use the measured delays $\mathbf{D}^p_t$ to predict a subset $\mathbf{D}^m_t \subset \mathbf{D}^m$, provided that the predicted values of $\mathbf{D}^m_t$ can also provide sufficient information for the other maximum delays $\mathbf{D}^m \backslash \mathbf{D}^m_t$, thus establishing a chained relation $\mathbf{D}^p_t \rightarrow \mathbf{D}^m_t \rightarrow \mathbf{D}^m$. Since $\mathbf{D}^m_t$ is a subset of $\mathbf{D}^m$, to identify it from $\mathbf{D}^m$ is the same statistical prediction problem as discussed in [44]. Therefore, we only need to focus on the task to find a set of combinational paths to predict

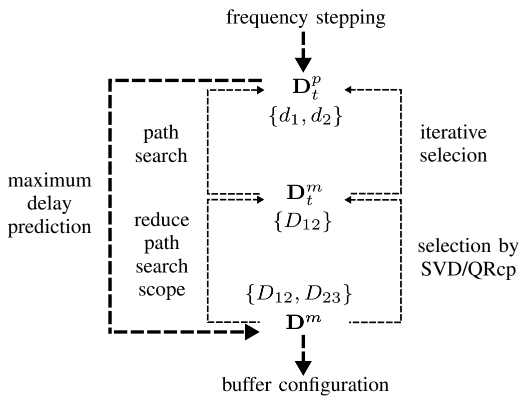


Fig. 6: Relation between delay sets, using the test scenario in Fig. 5 as example. The thin dashed lines represent the relation between the delay sets for identifying representative combinational paths. The thick dashed lines illustrate the real test and prediction procedure.

$\mathbf{D}_t^m$. 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. 6. We 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. The details of these steps are described as follows.

2) Determining a set of maximum delays $\mathbf{D}_t^m$ from $\mathbf{D}^m$ using SVD-QRcp: In the delay prediction concept shown in Fig. 6, the first step is to determine a subset of variables $\mathbf{D}_t^m$ from $\mathbf{D}^m$, so that the values of $\mathbf{D}_t^m$ can predict the values of $\mathbf{D}^m$. This problem has been studied previously such as in [44]–[47] using an algorithm based on SVD-QRcp. Assume that a delay from $\mathbf{D}^m$ is written as a linear combination of M random components $\mathbf{S} = [s_1, s_2, \dots, s_M]^T$, such as in the canonical form in [3]. 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

$$\mathbf{C} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T \tag{4}$$

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

$$\mathbf{U}_{[1:n]}^T = \mathbf{QR\Pi}^T \tag{5}$$

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 [48] and GSL [49] libraries. The coefficient matrix $\mathbf{C}$ of $\mathbf{D}^m$ and the matrices after decomposition are shown in the following.

$$\overset{\mathbf{C}}{\begin{bmatrix} 10 & 6 & 1 \\ 13 & 4 & 2 \\ 7 & 5 & 1 \end{bmatrix}} = \overset{\mathbf{U}}{\begin{bmatrix} -0.59 & 0.43 & -0.68 \\ -0.69 & -0.72 & 0.13 \\ -0.43 & 0.55 & 0.72 \end{bmatrix}} \times \overset{\mathbf{\Lambda}}{\begin{bmatrix} 19.83 & 0 & 0 \\ 0 & 2.76 & 0 \\ 0 & 0 & 0.31 \end{bmatrix}}$$

$$\times \overset{\mathbf{V}^T}{\begin{bmatrix} -0.90 & -0.40 & -0.18 \\ -0.42 & 0.90 & 0.10 \\ -0.12 & -0.16 & 0.98 \end{bmatrix}}$$

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

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 $\mathbf{D}_t^m$. This number needs to be decided so that the prediction accuracy is maintained. 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 [11]. Under this assumption, these delays can be written together as $\mathbf{D} = \begin{bmatrix} d_k \\ \mathbf{D}_t \end{bmatrix} \sim \boldsymbol{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$, where $\boldsymbol{\mu}$ is the mean value vector of $\mathbf{D}$, $\mathbf{\Sigma}$ is the covariance matrix of $\mathbf{D}$, $d_k \sim N(\mu_k, \sigma_k)$ and $\mathbf{D}_t \sim \boldsymbol{N}(\boldsymbol{\mu}_t, \mathbf{\Sigma}_t)$. Accordingly, $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ can be expressed as $\boldsymbol{\mu} = \begin{bmatrix} \mu_k \\ \boldsymbol{\mu}_t \end{bmatrix}$, and $\mathbf{\Sigma} = \begin{bmatrix} \sigma_k & \mathbf{\Sigma}_{k,t} \\ \mathbf{\Sigma}_{t,k} & \mathbf{\Sigma}_t \end{bmatrix}$, where $\mathbf{\Sigma}_{k,t} = \mathbf{\Sigma}_{t,k}^T$ 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 [42].

$$\mu_k' = \mu_k + \mathbf{\Sigma}_{k,t}\mathbf{\Sigma}_t^{-1}(\boldsymbol{d}_t - \boldsymbol{\mu}_t) \tag{6}$$

$$\sigma_k'^2 = \sigma_k^2 - \mathbf{\Sigma}_{k,t}\mathbf{\Sigma}_t^{-1}\mathbf{\Sigma}_{t,k}. \tag{7}$$

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 (7), 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

```
Algorithm 1: 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 $\mathbf{D}_t^m \subseteq \mathbf{D}^m$
L1   $\mathbf{U} \leftarrow$ Decompose $\mathbf{C}$ using SVD (4);
L2   for $i \leftarrow 1$ to $|\mathbf{D}^m|$ do
L3       $\mathbf{U}_{[1:i]} \leftarrow$ First $i$ columns of $\mathbf{U}$;
L4       $\mathbf{\Pi}^T \leftarrow$ Decompose $\mathbf{U}_{[1:i]}^T$ using QRcp (5);
L5       Select $\mathbf{D}_t^m$ from $\mathbf{D}^m$ using $\mathbf{\Pi}^T$;
L6       foreach $d_k \in \mathbf{D}^m \backslash \mathbf{D}_t^m$ do
L7           Compute $\sigma_k'^2$ using (7);
L8           if $\sigma_k' > \sigma_{th}$ then
L9               goto L2;
L10          end
L11      end
L12      break;
L13  end
L15  return $\mathbf{D}_t^m$
```

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.

In identifying $\mathbf{D}_t^m$ from $\mathbf{D}^m$, if the standard deviation σ' of a delay from $\mathbf{D}^m \backslash \mathbf{D}_t^m$ 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 [3], [11], 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 $\mathbf{D}_t^m$ from $\mathbf{D}^m$ is summarized in Algorithm 1.

3) Identifying representative paths using iterative selection: The maximum delays $\mathbf{D}_t^m$ returned by Algorithm 1 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 [50], [51]. 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. 6 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. 5 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.

```
Algorithm 2: 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 1
     Output: Selected paths $\mathbf{P}_t$ for delay test
L1   $\mathbf{P}_c \leftarrow \emptyset$;
L2   foreach $d_k \in \mathbf{D}_t^m$ do
L3       $\{\text{ff}_{src}, \text{ff}_{dst}\} \leftarrow$ Find flip-flops corresponding to $d_k$;
L4       $\mathbf{P}_s \leftarrow$ Trace five most critical paths $\text{ff}_{src} \rightarrow \text{ff}_{dst}$;
L5       $\mathbf{P}_c \leftarrow \mathbf{P}_c \cup \mathbf{P}_s$ ;
L6   end
L7   $\mathbf{D}_c \leftarrow$ Delays of $\mathbf{P}_c$;
L8   $\mathbf{D}_t^p \leftarrow \emptyset$;
L9   for $i \leftarrow 1$ to $|\mathbf{D}_c|$ do
L10      $\sigma_{next}^2 \leftarrow \infty$;
L11      $d_{next} \leftarrow null$;
L12      foreach $d_c \in \mathbf{D}_c \backslash \mathbf{D}_t^p$ do
L13          $\mathbf{D}_t^{p\prime} \leftarrow \{d_c\} \cup \mathbf{D}_t^p$;
L14          $\sigma_{max}^2 \leftarrow 0$;
L15          foreach $d_k \in \mathbf{D}^m$ do
L16              Compute $\sigma_k'^2$ from $\mathbf{D}_t^{p\prime}$ to $\mathbf{D}^m$ using (7);
L17              if $\sigma_k'^2 > \sigma_{max}^2$ then
L18                  $\sigma_{max}^2 \leftarrow \sigma_k'^2$;
L19              end
L20          end
L21          if $\sigma_{max}^2 < \sigma_{next}^2$ then
L22              $\sigma_{next}^2 \leftarrow \sigma_{max}^2$;
L23              $d_{next} \leftarrow d_c$;
L24          end
L25      end
L26      $\mathbf{D}_t^p \leftarrow \{d_{next}\} \cup \mathbf{D}_t^p$;
L27      if $\sigma_{next} \leq \sigma_{th}$ then
L28          $\mathbf{P}_t \leftarrow$ Paths corresponding to $\mathbf{D}_t^p$;
L29          break;
L30      end
L31  end
L33  return $\mathbf{P}_t$
```

Because the number of combinational paths between a pair of flip-flops is usually very large, 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, we enumerate all the path delays in $\mathbf{D}_c$ 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 1.

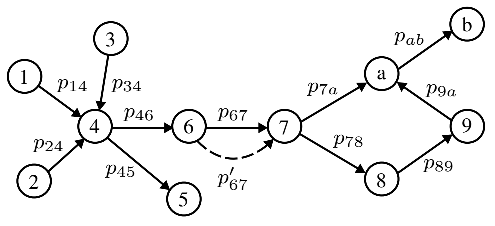


Fig. 7: Test scenario with multiple combinational paths. The nodes represent flip-flops. The solid edges represent combinational paths whose delays need to be tested using frequency stepping. The dashed edges represent an additional path that can also be tested without increasing the number of test batches.

The procedure of selecting combinational paths for delay test is summarized in Algorithm 2. 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 (7). 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.

B. 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 exceeds 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. 7, 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., [52] and [53], 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 2 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,

$$\left(\begin{array}{cccc|ccc} b_{1,1} & b_{1,2} & \ldots & b_{1,N_t} & b_{1,N_t+1} & \ldots & b_{1,N_t+N_a} \\ b_{2,1} & b_{2,2} & \ldots & b_{2,N_t} & b_{2,N_t+1} & \ldots & b_{2,N_t+N_a} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{N_t,1} & b_{N_t,2} & \ldots & b_{N_t,N_t} & b_{N_t,N_t+1} & \ldots & b_{N_t,N_t+N_a} \end{array}\right) \tag{8}$$

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 (8) should be equal to one, written as

$$\sum_{i=1}^{N_t} b_{i,j} = 1, \quad 1 \le j \le N_t. \tag{9}$$

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,

$$\sum_{p_j \in I_F} b_{i,j} \le 1, \quad \sum_{p_j \in O_F} b_{i,j} \le 1, \quad 1 \le i \le N_t \tag{10}$$

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 (8) 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

$$b_{i,j} \le B_i, \quad 1 \le j \le N_t,\ 1 \le i \le N_t. \tag{11}$$

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

$$W_i = \sum_{j=1}^{N_t} b_{i,j}\mu_j, \quad 1 \le i \le N_t \tag{12}$$

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. 7 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 III-A2, the predicted standard deviation is used as an indicator of the prediction accuracy. Since a large standard deviation σ'_k calculated by (7) 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 (8). 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

$$\sum_{i=1}^{N_t} b_{i,j} \le 1, \quad N_t \le j \le N_t + N_a. \tag{13}$$

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 $\sum_{1\le i\le N_t, N_t+1\le j\le N_t+N_a} b_{i,j}$. With the new columns in (8), (11) and (12) should be revised to incorporate the extended indexes as

$$b_{i,j} \le B_i, \quad 1 \le i \le N_t,\ 1 \le j \le N_t + N_a \tag{14}$$

$$W_i = \sum_{j=1}^{N_t+N_a} b_{i,j}\mu_j, \quad 1 \le i \le N_t. \tag{15}$$

Considering the three objectives discussed above, we formulate the path assignment task into an ILP problem as

$$\text{Minimize} \quad \alpha\sum_{i=1}^{N_t} B_i - \beta\sum_{i=1}^{N_t}\varepsilon_i W_i - \gamma \sum_{\substack{1\le i\le N_t \\ N_t+1\le j\le N_t+N_a}} b_{i,j} \tag{16}$$

$$\text{Subject to} \quad \text{(9)–(10) and (13)–(15)} \tag{17}$$

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 (8) are kept as test batches, denoted as $\mathbf{B}$.

C. Test with Delay Alignment by Tuning Buffers

After path batches are identified, they should be tested using frequency stepping to determine the path delays. In this section, we discuss how the delays of paths in a single batch are measured. 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 sensitized by test vectors. If the setup time constraint (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 (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, the value of D_{ij} can be approximated with a given accuracy.

Consider the case shown in Fig. 8(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. 8(b).

When several path delays in one test batch are considered as in Fig. 8(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

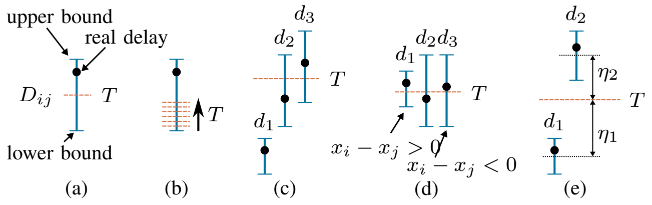


Fig. 8: Frequency stepping and delay range alignment.

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. 8(c). According to (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. 8(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. 8(c). In EffiTest2, at-speed scan test is deployed for delay tests. At-speed scan test has been applied in [36], [37] and investigated thoroughly in [54]. 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 (3). In addition, these buffer values may affect more than one path delay. For example, in Fig. 7 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 (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

$$\eta_{ij} = |T - ((u_{ij} + l_{ij})/2 + x_i - x_j)|. \tag{18}$$

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. 8(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. 8(e) are different so that the next test clock period T should align at the center of the range with the larger weight.

Algorithm 3: Test procedure with frequency stepping

```
Input: B: the queue of test batches
foreach $B_k \in$ B do
    while $B_k$ contains an edge do
        $T \leftarrow$ solve (19)–(26);
        test_with_frequency_stepping($B_k$, $T$);
        foreach $p_{ij}$ in $B_k$ do
            if passed($p_{ij}$) then
                $u_{ij}$=$T - x_i + x_j$;
            else
                $l_{ij}$=$T - x_i + x_j$;
            end
            if $u_{ij} - l_{ij} < \epsilon$ then
                remove_edge($p_{ij}$, $B_k$);
            end
        end
    end
end
```

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

$$\text{Minimize} \quad \sum_{i,j} k_{ij}\eta_{ij} \tag{19}$$

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

$$T - ((u_{ij} + l_{ij})/2 + x_i - x_j) \leq \mathcal{M} z^p_{ij} \tag{20}$$

$$(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) - \eta_{ij} \leq \mathcal{M}(1 - z^p_{ij}) \tag{21}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) + \eta_{ij} \leq \mathcal{M}(1 - z^p_{ij}) \tag{22}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) \leq \mathcal{M} z^n_{ij} \tag{23}$$

$$-(T - ((u_{ij} + l_{ij})/2 + x_i - x_j)) - \eta_{ij} \leq \mathcal{M}(1 - z^n_{ij}) \tag{24}$$

$$(T - ((u_{ij} + l_{ij})/2 + x_i - x_j) + \eta_{ij} \leq \mathcal{M}(1 - z^n_{ij}) \tag{25}$$

$$r_i \leq x_i \leq r_i + \tau_i, \; r_j \leq x_j \leq r_j + \tau_j \tag{26}$$

where (20)–(25) are linear constraints transformed from (18) and $\mathcal{M}$ is a very large positive constant [55]; z^p_{ij} and z^n_{ij} 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. (26) defines the ranges of buffer values as in (3).

After the clock frequency and the corresponding buffer values are determined by solving the ILP problem (19)–(26), 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 3. 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.

D. Buffer Configuration with Delay Estimation

To rescue chips with timing failures after manufacturing, 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 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 range with a lower bound and an upper bound. For another delay d_k that is not measured directly but is to be estimated, (6) and (7) are used to calculate the mean value μ'_k and the standard deviation σ'_k. According to (6) and (7), 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 the following buffer configuration.

A real delay may take any value in the range defined by the lower and upper bounds, but the exact location of this delay in the range is unknown due to test resolution and delay estimation. To tackle 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 report some chips as nonfunctional due to pessimistic delay overestimation. To solve 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.

$$\text{Minimize} \quad \xi \tag{27}$$

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

$$T_d \geq D'_{ij} + x_i - x_j \tag{28}$$

$$l_{ij} \leq D'_{ij} \leq u_{ij},\ \xi \geq (u_{ij} - D'_{ij})/\sigma'_{ij} \tag{29}$$

$$r_i \leq x_i \leq r_i + \tau_i,\ r_j \leq x_j \leq r_j + \tau_j \tag{30}$$

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; (28) and (30) are derived from (1) and (3), respectively. By solving the optimization problem (27)–(30), a set of buffer configuration values x_i and x_j can be found.

E. 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. 3, if x_j is much larger than x_i, the constraint (2) may be violated.

As shown in (2), hold time constraints are affected by $x_i - x_j$ instead of individual values of x_i and x_j. In our 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 (2) so that a given yield can be maintained.

Consider the case that d_{ij} in (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

$$\lambda_{ij} - d_{ij,k} \geq \mathcal{M}(y_k - 1), \quad \text{for all short paths } p_{ij} \tag{31}$$

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

$$\sum y_i/M \geq Y, \quad i = 1, 2, \ldots M \tag{32}$$

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

$$x_i - x_j \geq \lambda_{ij}. \tag{33}$$

This constraint is added into the optimization problems in Section III-C and Section III-D to incorporate hold time constraints to determine buffer values x_i and x_j.

IV. 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. Details of these circuits are shown in Table II, where $\boldsymbol{n_s}$ denotes the number of flip-flops and $\boldsymbol{n_g}$ the number of logic gates. The numbers of inserted tunable buffers are shown in the column $\boldsymbol{n_b}$, which were less than 1% of the numbers of flip-flops in the benchmark circuits. The locations of these buffers were determined using [30]. We set the maximum allowed buffer ranges to 1/8 of the original clock period and all tuning delays with 20 discrete steps [22]. 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 [56]. The correlation of variations between logic gates is defined using the curve in [57]. The ILP solver for the optimization problems was Gurobi [58].

In Table II the column $|\mathbf{D}^{\boldsymbol{m}}|$ 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

TABLE II: Test Results With Delay Alignment and Statistical Prediction

Circuit				EffiTest2						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

fanin flip-flops to it and from it to all its fanout flip-flops are added into $\mathbf{D}^m$ as discussed in Section III-A. 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 1. 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 2. 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 III-B, and the numbers of these batches are shown in the column $|\mathbf{B}|$. Since multiple combinational paths are multiplexed during delay 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 [26], [35]–[37]. The column $\boldsymbol{n_a'}$ in Table II 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%$\sim$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 EffiTest2 framework reduces test effort significantly.

The runtimes of the proposed method are shown in the last three columns in Table II, 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 III-D. 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 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 III. 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 III-A 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 9 shows the yield results of three cases with respect to the clock period T_2 in

TABLE III: 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		

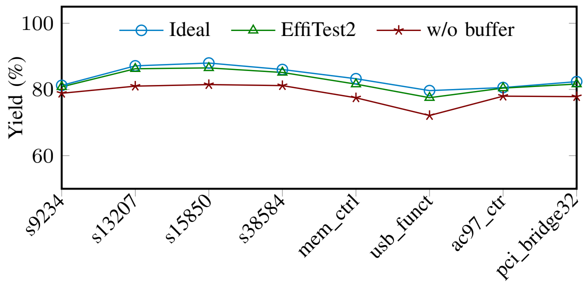


Fig. 9: Yield with enlarged random variation.

Table III: 1) ideal yield with buffers configured with presumed accurate delays; 2) with buffers configured using tested and predicted delays in EffiTest2; 3) no buffers in the circuits. Compared with the results in Table III, the yield values in Fig. 9 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 EffiTest2, the yield values dropped slightly from the ideal cases in Fig. 9, 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.

To verify the effectiveness of aligned delay ranges described in Section III-B and Section III-C, we applied them directly to reduce test iterations without statistical prediction. Figure 10 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 III-B and Section III-C, 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.

In the statistic prediction technique described in Algorithm 1 and 2, 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 1 and 2, we first find the maximum

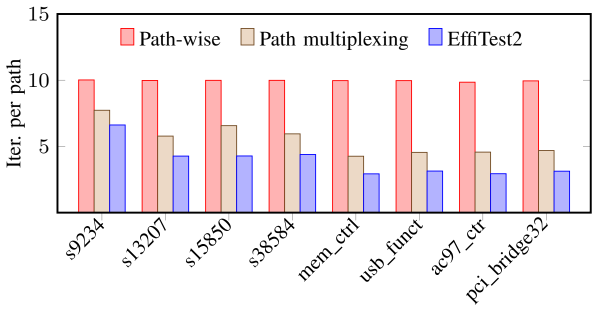


Fig. 10: Test comparison without statistical prediction.

of these purely random components of the predicted delays and set the threshold as constant times of them. Figure 11(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 3 and the results are shown in Figure 11(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 1 and 2 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 12 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 III-A, 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 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 III-A, 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 2. For a circuit, the threshold of the prediction accuracy σ_{th} for path selection in Algorithm 2 is set with respect to the standard deviations of purely random components of path delays described in Section III-A. 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. 13 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 EffiTest2 to maintain the accuracy. The other circuits that are

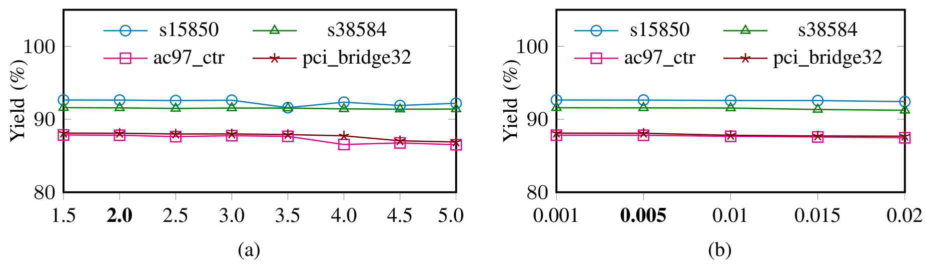


Fig. 11: Prediction threshold and its effect on yield in Algorithm 1 and Algorithm 2 (a), and test threshold over yield in Algorithm 3 (b).

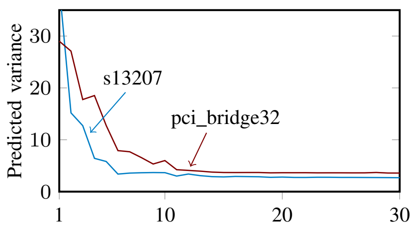


Fig. 12: Effect of the number of selected variables over prediction accuracy.

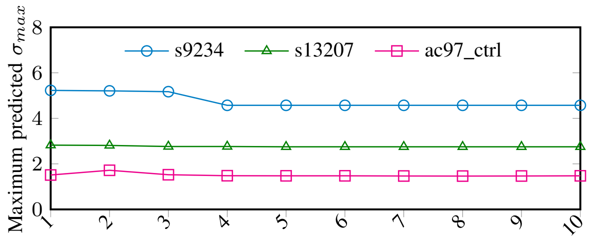


Fig. 13: Comparisons of predicted standard deviations using different numbers of combinational paths. Lower values indicate better prediction accuracy.

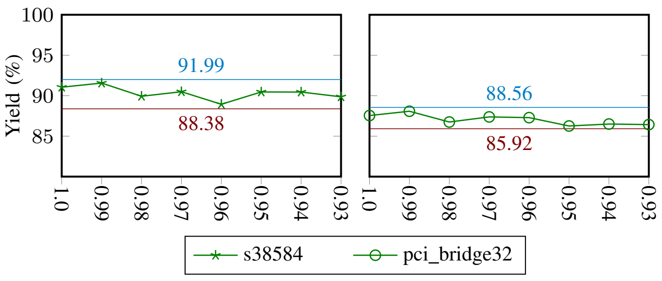


Fig. 14: Hold time threshold effect over yield.

not included in Fig. 13 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 III-E to reduce hold time violations. These constraints are controlled by the threshold Y in (32), whose effect over the yield values of s38584 and pci_bridge32 are shown in Fig. 14. 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 (32)–(33). 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.

V. Conclusions

In this paper we have proposed an efficient framework to reduce test cost in configuring tunable buffers in high-performance designs. 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. Future work will consider techniques combining post-silicon tuning and flexible timing such as in [59].

References

[1] 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.*, 2016, pp. 60:1–60:6.
[2] 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.*, 2003, pp. 621–625.
[3] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis," in *Proc. Design Autom. Conf.*, 2004, pp. 331–336.
[4] L. Zhang, W. Chen, Y. Hu, J. A. Gubner, and C. C. Chen, "Correlation-preserved non-gaussian statistical timing analysis with quadratic timing model," in *Proc. Design Autom. Conf.*, 2005, pp. 83–88.
[5] J. Singh and S. Sapatnekar, "Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis," in *Proc. Design Autom. Conf.*, 2006, pp. 155–160.
[6] A. Goel, S. B. K. Vrudhula, F. Taraporevala, and P. Ghanta, "A methodology for characterization of large macro cells and IP blocks considering process variations," in *Proc. Int. Symp. Quality Electron. Des.*, 2008, pp. 200–206.
[7] A. Goel, S. Vrudhula, F. Taraporevala, and P. Ghanta, "Statistical timing models for large macro cells and IP blocks considering process variations," *IEEE Trans. Semicond. Manuf.*, vol. 22, pp. 3–11, 2009.
[8] 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.*, 2009, pp. 1320–1325.
[9] 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.*, vol. 32, no. 3, pp. 367–380, 2013.
[10] 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.*, 2015, pp. 1587–1590.
[11] 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.*, vol. 27, no. 4, pp. 589–607, Apr. 2008.
[12] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. Int. Symp. Microarch.*, 2003, pp. 7–18.
[13] D. Blaauw, S. Kalaiselvan, K. Lai, W.-H. Ma, S. Pant, C. Tokunaga, S. Das, and D. M. Bull, "Razor II: In situ error detection and correction for pvt and ser tolerance," in *Proc. Int. Solid-State Circuits Conf.*, pp. 400–401.
[14] M. Fojtik, D. Fick, Y. Kim, N. R. Pinckney, D. M. Harris, D. Blaauw, and D. Sylvester, "Bubble razor: Eliminating timing margins in an ARM cortex-m3 processor in 45 nm CMOS using architecturally independent error detection and correction," *J. Solid-State Circuits*, vol. 48, no. 1, pp. 66–81, 2013.
[15] I. Kwon, S. Kim, D. Fick, M. Kim, Y. Chen, and D. Sylvester, "Razor-lite: A light-weight register for error detection by observing virtual supply rails," *IEEE J. Solid-State Circuits*, vol. 49, no. 9, pp. 2054–2066, 2014.
[16] N. Chen, B. Li, and U. Schlichtmann, "Iterative timing analysis based on nonlinear and interdependent flipflop modelling," *IET Circuits, Devices & Systems*, vol. 6, no. 5, pp. 330–337, 2012.
[17] 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.*, vol. 26, no. 6, pp. 1114–1125, 2007.
[18] S. Srivastava and J. S. Roychowdhury, "Interdependent latch setup/hold time characterization via Euler-Newton curve tracing on state-transition equations," in *Proc. Design Autom. Conf.*, 2007, pp. 136–141.
[19] A. B. Kahng and H. Lee, "Timing margin recovery with flexible flip-flop timing model," in *Proc. Int. Symp. Quality Electron. Des.*, 2014, pp. 496–503.

[20] Y. Yang, K. H. Tam, and I. H. Jiang, "Criticality-dependency-aware timing characterization and analysis," in *Proc. Design Autom. Conf.*, 2015, pp. 167:1–167:6.

[21] 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.*, 2016, p. 100.

[22] 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*, vol. 35, no. 11, pp. 1545–1552, Nov. 2000.

[23] E. Takahashi, Y. Kasai, M. Murakawa, and T. Higuchi, "Post-fabrication clock-timing adjustment using genetic algorithms," *IEEE J. Solid-State Circuits*, vol. 39, no. 4, pp. 643–650, Apr. 2004.

[24] 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.*, 2005, pp. 292–293.

[25] 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*, vol. 41, no. 1, pp. 197–209, Jan. 2006.

[26] 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.*, 2004, pp. 611–618.

[27] J. Kim and T. Kim, "Adjustable delay buffer allocation under useful clock skew scheduling," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 4, pp. 641–654, Apr. 2017.

[28] 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.*, 2005, pp. 575–581.

[29] 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.*, 2016, pp. 1457–1460.

[30] 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.*, vol. 37, no. 2, pp. 392–405, 2018.

[31] V. Khandelwal and A. Srivastava, "Variability-driven formulation for simultaneous gate sizing and post-silicon tunability allocation," in *Proc. Int. Symp. Phys. Des.*, 2007, pp. 11–18.

[32] 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.*, 2009, pp. 292–295.

[33] 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.*, 2011, pp. 111–117.

[34] 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.*, vol. 34, no. 11, pp. 1784–1797, 2015.

[35] Z. Lak and N. Nicolici, "A novel algorithmic approach to aid post-silicon delay measurement and clock tuning," *IEEE Trans. Comput.*, vol. 63, no. 5, pp. 1074–1084, May 2014.

[36] 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.*, 2008, pp. 1–8.

[37] D. Tadesse, J. Grodstein, and R. I. Bahar, "AutoRex: An automated post-silicon clock tuning tool," in *Proc. Int. Test Conf.*, 2009, pp. 1–10.

[38] R. Ye, F. Yuan, and Q. Xu, "Online clock skew tuning for timing speculation," in *Proc. Int. Conf. Comput.-Aided Des.*, 2011, pp. 442–447.

[39] 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.*, vol. 31, no. 12, pp. 1845–1856, Dec. 2012.

[40] 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.*, vol. 16, no. 6, pp. 639–649, Jun. 2008.

[41] J. Fishburn, "Clock skew optimization," *IEEE Trans. Comput.*, vol. 39, no. 7, pp. 945–951, Jul. 1990.

[42] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis*. Upper Saddle River: Pearson Prentice Hall, 2007.

[43] 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.*, vol. 28, no. 8, pp. 1201–1212, Aug. 2009.

[44] F. Firouzi, F. Ye, K. Chakrabarty, and M. B. Tahoori, "Representative critical-path selection for aging-induced delay monitoring," in *Proc. Int. Test Conf.*, 2013, pp. 1–10.

[45] J. Yen and L. Wang, "Simplifying fuzzy rule-based models using orthogonal transformation methods," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 29, no. 1, pp. 13–24, 1999.

[46] L. Xie and A. Davoodi, "Representative path selection for post-silicon timing prediction under variability," in *Proc. Design Autom. Conf.*, 2010, pp. 386–391.

[47] 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*, vol. 20, no. 3, pp. 1–39, Jun. 2015.

[48] 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*, 3rd ed. Society for Industrial and Applied Mathematics, 1999.

[49] M. Galassi *et al.*, *GNU Scientific Library Reference Manual*, 3rd ed.

[50] S. Pateras, "Achieving at-speed structural test," *IEEE Des. Test. Comput.*, vol. 20, no. 5, pp. 26–33, 2003.

[51] 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.*, vol. 20, no. 5, pp. 17–25, 2003.

[52] M. Michael and S. Tragoudas, "Function-based compact test pattern generation for path delay faults," *IEEE Trans. VLSI Syst.*, vol. 13, no. 8, pp. 996–1001, Aug. 2005.

[53] 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.*, vol. 14, no. 12, pp. 1505–1515, 1995.

[54] 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.*, 2010, pp. 1–8.

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

[56] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. Custom Integr. Circuits Conf.*, 2001, pp. 223–228.

[57] J. Xiong, V. Zolotov, and L. He, "Robust extraction of spatial correlation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 4, pp. 619–631, 2007.

[58] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2013. [Online]. Available: http://www.gurobi.com

[59] 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.*, 2018.

Grace Li Zhang received the master's degree from the school of microelectronics, Xidian University, Xi'an, China, in 2014. She is currently pursuing the Ph.D. degree with the Chair of Electronic Design Automation, Technical University of Munich (TUM). Her research interests include high-performance and lower-power design, as well as emerging systems.

Bing Li received the bachelor's and master's degrees in communication and information engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2000 and 2003, respectively, and the Dr.-Ing. degree in electrical engineering from Technical University of Munich (TUM), Munich, Germany, in 2010. He is currently a researcher with the Chair of Electronic Design Automation, TUM. His research interests include high-performance and lower-power design, as well as emerging systems.

Yiyu Shi (SM'06) is currently an associate professor in the Departments of Computer Science and Engineering and Electrical Engineering at the University of Notre Dame. He received his B.S. degree (with honors) in Electronic Engineering from Tsinghua University, Beijing, China in 2005, the M.S and Ph.D. degree in Electrical Engineering from the University of California, Los Angeles in 2007 and 2009 respectively. His current research interests include three-dimensional integrated circuits, and machine learning on chips. In recognition of his research, he has received many best paper nominations in top conferences. He was also the recipient of IBM Invention Achievement Award in 2009, Japan Society for the Promotion of Science (JSPS) Faculty Invitation Fellowship, Humboldt Research Fellowship for Experienced Researchers, IEEE St. Louis Section Outstanding Educator Award, Academy of Science (St. Louis) Innovation Award, Missouri S&T Faculty Excellence Award, National Science Foundation CAREER Award, IEEE Region 5 Outstanding Individual Achievement Award, and the Air Force Summer Faculty Fellowship.

Jiang Hu (F'16) received the B.S. degree in optical engineering from Zhejiang University (China) in 1990, the M.S. degree in physics in 1997 and the Ph.D. degree in electrical engineering from the University of Minnesota in 2001. He worked with IBM Microelectronics from January 2001 to June 2002. In 2002, he joined the electrical engineering faculty at Texas A&M University. His research interest is in optimization for large scale computing systems, especially VLSI circuit optimization, adaptive design, hardware security, resource allocation and power management in computing systems.

Dr. Hu received the Best Paper Award at the ACM/IEEE Design Automation Conference in 2001, an IBM Invention Achievement Award in 2003, and the Best Paper Award from the IEEE/ACM International Conference on Computer-Aided Design in 2011. He has served as a technical program committee member for DAC, ICCAD, ISPD, ISQED, ICCD, DATE, ISCAS, ASP-DAC and ISLPED. He is general chair for the 2012 ACM International Symposium on Physical Design, and an associate editor of IEEE transactions on CAD 2011-2016. Currently he is an associate editor for the ACM transactions on Design Automation of Electronic Systems. He received a Humboldt research fellowship in 2012.

Ulf Schlichtmann (S'88–M'90) received the Dipl.-Ing. and Dr.-Ing. degrees in electrical engineering and information technology from Technical University of Munich (TUM), Munich, Germany, in 1990 and 1995, respectively. He was with Siemens AG, Munich, and Infineon Technologies AG, Munich, from 1994 to 2003, where he held various technical and management positions in design automation, design libraries, IP reuse, and product development. He has been a Professor and the Head of the Chair of Electronic Design Automation with TUM, since 2003. He served as the Dean of the Department of Electrical and Computer Engineering, TUM, from 2008 to 2011. His current research interests include computer-aided design of electronic circuits and systems, with an emphasis on designing reliable and robust systems.

6 Timing beyond the Traditional Paradigm

In natural science, the development of a new field becomes stable generally after a fundamental theory or model prevails from the early exploration. For example, the concept of digital circuits is built upon Boolean algebra. To reduce design complexity, clocked circuits have been adopted and timing analysis has become one of the primary supporting pillars of sequential design. Together with the continuous advances in semiconductor manufacturing, a flourishing IC industry has been established successfully.

Based on Boolean algebra and clocked digital circuits, design methods and flows for integrated circuits have been evolving rapidly in the past several decades. Meanwhile, another industry, electronic design automation, has taken a phenomenal role in the advance of the IC industry. Above the concept of clocked digital circuits, techniques such as static timing analysis, pipelining and retiming for circuit optimization, and clock networks for delivering the clock signal to sequential components have been introduced and widely used in the industry. Some of these techniques and their relation to the basic concept of Boolean logic and clocked circuit are illustrated in Figure 6.1.

At the root of this tree structure, Boolean algebra enables a simplification from analog values to digital values, which are the only two extreme ends of a signal switching range, so that the exact values of a signal between these two ends become relatively irrelevant, thus allowing the IC industry to focus on the integration of more devices into a chip instead of tuning each device meticulously. A digital circuit only processes Boolean values, and its structure can be synthesized from high-level

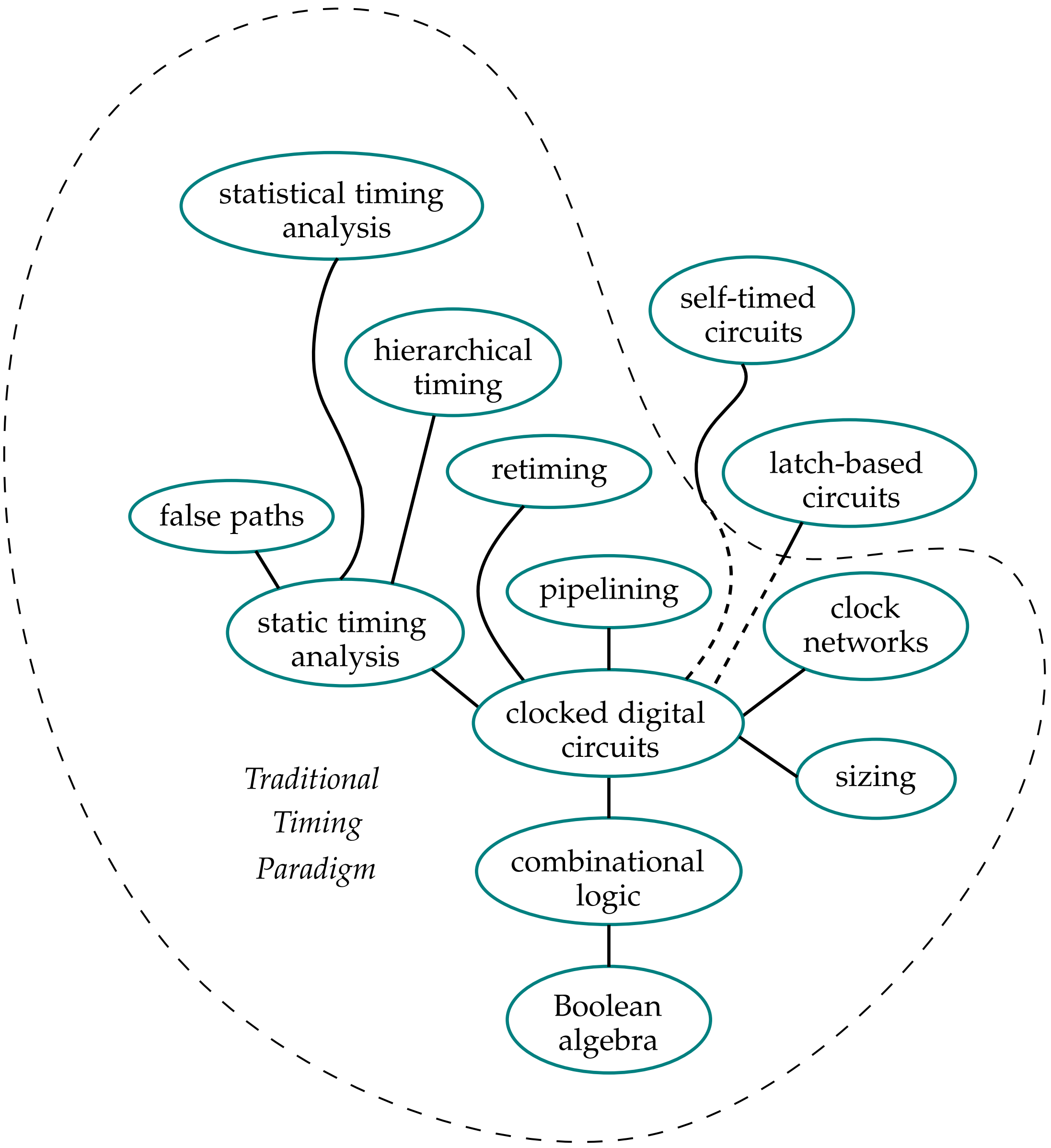


Figure 6.1: Structure of Timing Concepts.

descriptions, truth tables, or Karnaugh maps into connected logic gates to form a netlist. To coordinate the operations of combinational function blocks, a clock signal is used to align all the computation. The minimum period of this clock signal is determined by the largest combinational delay, calculated using static timing analysis. Since a circuit cannot time itself, a regularly repeating clock signal needs to be generated and distributed through a clock network to sequential components such as flip-flops and latches. With the clock signal and the sequential components, it is then possible to allow different function blocks of the circuit to work simultaneously, naturally leading to the concept of pipelining. While sizing trades more performance of logic gates with chip area, retiming and false path identification are two other ways to improve performance at the circuit level, although the former shifts sequential components along combinational paths, while the latter just excludes combinational paths that are timing-critical but never triggered when the circuit operates. To improve the efficiency of timing analysis, timing models can be generated for submodules and the overall performance of a circuit can be evaluated using hierarchical timing analysis. When the semiconductor industry entered the nanometer realm, process variations could not be hidden from designers by corner-based design anymore. Accordingly, statistical timing analysis has been introduced as a further expansion of static timing analysis, by modeling process variations as random variables and incorporating them into timing analysis directly, but with the cost of more analysis time.

At the root, the concept of digital circuits has been serving the industry for several decades. But at the top, the concepts are much newer, and many of them are still the objects of active research. For example, statistical timing analysis and optimization thrived after 2000 for several years. At the top of this concept tree, the number of leaf nodes is much larger, meaning more detailed problems are addressed by academia and industry. However, the number of leaf nodes cannot increase infinitely, since the complexity of designing a circuit may become unmanageable.

The techniques in the traditional timing paradigm are generally confined by the timing constraints (2.1)–(2.2), where a logic computation usually needs to be finished within one clock period. As factors such as power consumption and process variations become more pronounced in recent years, it has becomes very difficult to improve the performance of a single computation unit. Accordingly, the IC industry has switched to incorporate more cores inside a chip to provide more parallel computing power, while leaving the task of performance improvement of applications to software implementation. In highly parallelized scenarios such as data centers, this new design paradigm is valid; but for user scenarios demanding a prompt response from a system, the improvement of computation performance of a single core is still an unmet requirement.

In the development of the timing concepts, additional techniques such as latch-based design and self-timed circuits have also been explored. These methods go beyond the flip-flop-bound single-period constraints (2.1)–(2.2) to allow a flexible timing, but it is still challenging to integrate these techniques into existing IC design flows.

To meet the demand for high-performance circuit design and to cope with the ever-increasing uncertainty in manufacturing, new timing paradigms beyond the traditional single-period clocking scheme need to be examined to redefine timing in the nanometer era. In the following discussion, a new timing design and optimization concept will be introduced. This concept allows timing flexibility across sequential stages while being compatible with the traditional timing paradigm at the interface of computation units. In addition, it can be applied to increase netlist security against counterfeiting by reverse engineering. This new timing concept has been published in [ZLY+18, ZLHS18].

6.1 VirtualSync: Delay-based Timing Synchronization

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. A widely adopted method is sizing, in which gates are sized to improve objectives such as clock frequency and area efficiency, while timing constraints between flip-flops are satisfied. 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 metaheuristic approach is developed to size logic gates that have the greatest impact on power-performance tradeoffs. This method guarantees slack, capacitance and slew constraints throughout the optimization process.

The second method to improve circuit performance in the traditional paradigm is retiming, which moves sequential components, e.g., flip-flops, but still preserves the correct functional behavior of circuits. 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. A retimed circuit can be further improved by introducing intentional clock skews or latches to balance the delays of sequential stages with a finer granularity [LPF99].

Wave-pipelining is the third method to improve circuit performance, where logic waves are allowed to propagate through combinational paths without intermediate sequential components. This method provides a mechanism to make the clock frequency of a circuit independent of the largest path delay, which limits circuit performance in traditional circuit designs [BCKL98]. As early as in [JC93], a linear method to minimize the clock period using wave pipelining is proposed. 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].

The first two methods above can be used separately or jointly to improve circuit performance. However, sequential components are assumed to synchronize signal propagations 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.

In the following, a new timing model, VirtualSync, which breaks the confines of the traditional timing paradigm, will be discussed. In this timing model, both sequential components and combinational logic gates are considered as delay units. Signal propagations in the combinational circuits after removing flip-flops are examined and fast signals are delayed so that they do not catch a slow signal from the previous clock cycle. To delay fast signals, combinational logic gates add linear delays of the same amount to short and long paths, while sequential components provide non-linear delay effects, leading to different delays appended to fast and slow signals. Since a sequential delay unit, flip-flop or latch, is able to delay a fast signal up to about one clock cycle, their appearance in a circuit can effectively reduce the number of delay buffers required for signal synchronization, so that chip area can also be reduced.

The VirtualSync timing model views the circuit structure from a different perspective. In the traditional timing model, flip-flops are allocated in advance, and timing constraints (2.1)–(2.2) between flip-flops are verified afterwards. In the VirtualSync model, all flip-flops in a given digital circuit are removed at first. Consequently, data waves from different clock cycles may mix, because the essential function of flip-flops in the original circuit is to block the propagation of fast signals. In other words, flip-flops in a traditional digital circuit are used to make a circuit slow instead of making it fast. In the second step of the VirtualSync model, flip-flops are inserted back only at necessary locations in the circuit to block fast signals. Therefore, the circuit can be optimized by delaying it with the minimum amount of real delays introduced by combinational gates or virtual delays introduced by sequential components. In this formulation, the original logic gates in a circuit are also sized to increase delays, leading to even a potential area reduction. Within the VirtualSync model, the absence of flip-flops at some sequential stages allows a virtual synchronization to provide identical functionality as in the original circuit. Since the critical paths usually do not require any additional delays, no clock-to-q delays or setup requirements of flip-flops are involved on the critical paths, so that a better circuit performance even beyond the limit of traditional sequential design can be achieved.

In Section 6.1.1, the motivation and the basic idea of the VirtualSync model are explained. The corresponding timing optimization problem is formulated in Section 6.1.2. The VirtualSync model is explained in detail in Sections 6.1.3–6.1.5. Conclusions are drawn in Section 6.1.6.

6.1.1 Background and Motivation of VirtualSync

In traditional digital circuits, sequential components such as flip-flops synchronize signal propagations between pairs of flip-flops using a global clock signal, as shown in Figure 6.2(a). 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

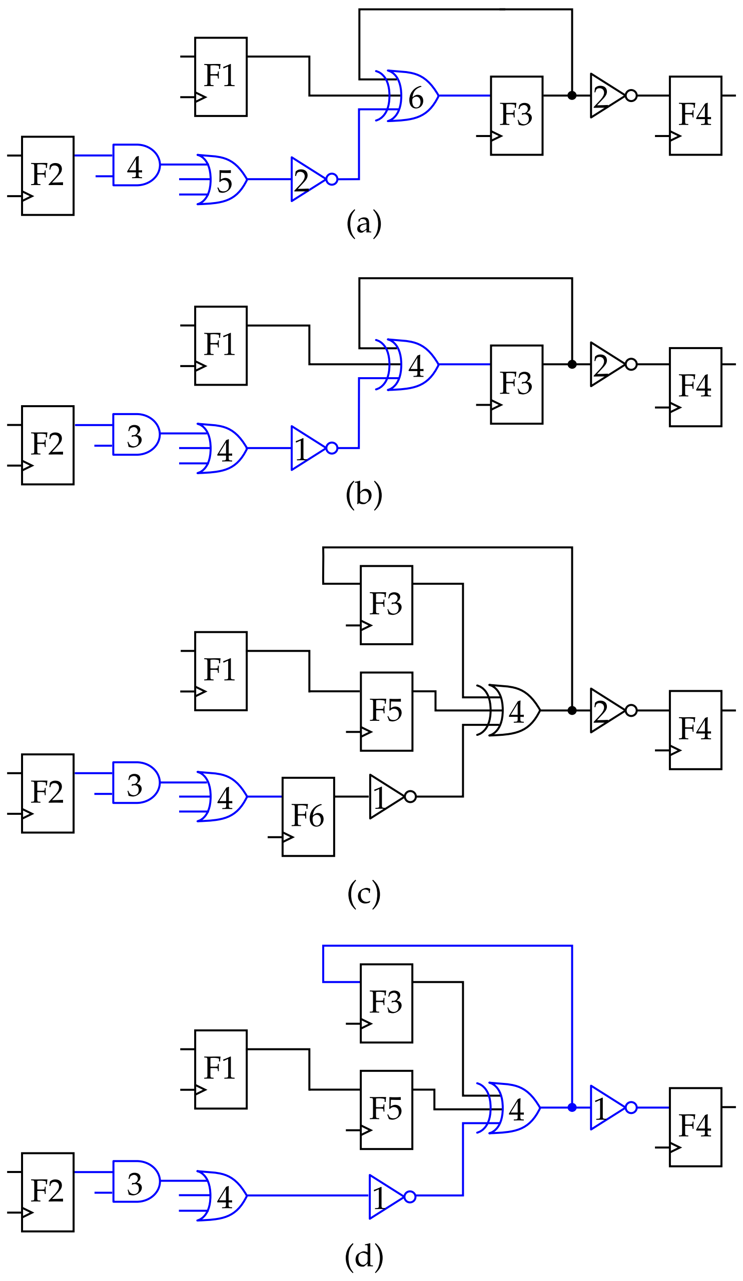


Figure 6.2: Timing optimization methods. Delays of logic gates are shown on the gates. The clock-to-q delay (t_{cq}), setup time (t_{su}) and hold time (t_h) of a flip-flop are 3, 1 and 1, respectively. (a) Original circuit. (b) Sized circuit. (c) Circuit after retiming. (d) Circuit after optimization using VirtualSync.

of additional area overhead, leading to the circuit shown in Figure 6.2(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 Figure 6.2(c), leading to a minimum clock period equal to 11.

The circuit in Figure 6.2(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 Figure 6.2(d). If the signal from 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 Figure 6.2(c). However, the flip-flop F5 in Figure 6.2(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 compared with the circuit in Figure 6.2(a). Comparing Figure 6.2(b) and Figure 6.2(d), it can be seen that F3 in Figure 6.2(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 practical designs. In VirtualSync, a new timing model that allows multiple waves on logic paths is introduced as a technique of timing optimization for circuits in the traditional design style. The resulting circuits still provide correct timing interfaces to sequential components, e.g., flip-flops, at the boundary of the optimized circuits to maintain timing compatibility.

6.1.2 Problem Formulation of VirtualSync

In digital circuits, the essential function of sequential components is to delay signals along fast paths in a circuit. For example, in Figure 6.2(d), F5 must be kept in the circuit to delay the signal propagation from F1 to F4. The sequential components that only sit on the critical path can thus be removed to improve circuit performance, such as F6 in Figure 6.2(d).

In the VirtualSync framework, all flip-flops are first removed and then the necessary locations to block fast signals using combinational gates and sequential components, e.g., buffers, flip-flops, and latches, are identified. 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.

6.1.3 Delay Units

In the VirtualSync framework, all sequential components, flip-flops, are removed 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. For example, in Figure 6.2(d), the combinational loop across the XOR gate must have a sequential component; otherwise a signal loses synchronization after traveling across it many times.

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 Figure 6.3, 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 Figure 6.3(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 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 Figure 6.3(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 is a very useful property because the delays of short paths and long paths in a circuit may differ significantly after all sequential components are removed from the circuit under optimization. For many short paths, it is not possible to increase their delays by adding combinational delay units such as buffers

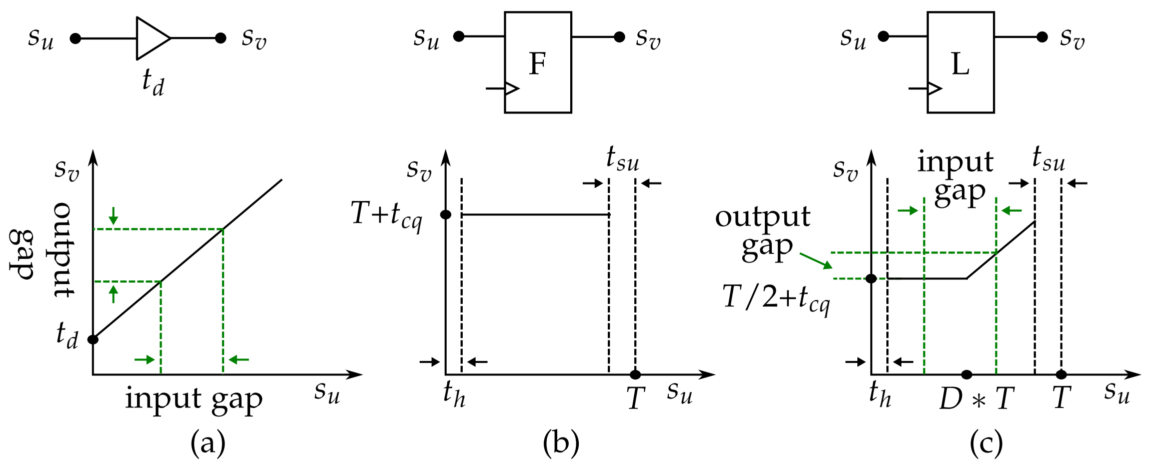


Figure 6.3: Properties of delay units. (a) Linear delaying effect of a combinational delay unit. (b) Constant delaying effect of a flip-flop. (c) Piecewise delaying effect of a latch.

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 are thus of great use 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 Figure 6.3(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 Figure 6.3(c). This property gives us more flexibility to modulate signals with different arrival times, specifically those along critical paths where fast signals require more delay padding and slow signals should be not affected.

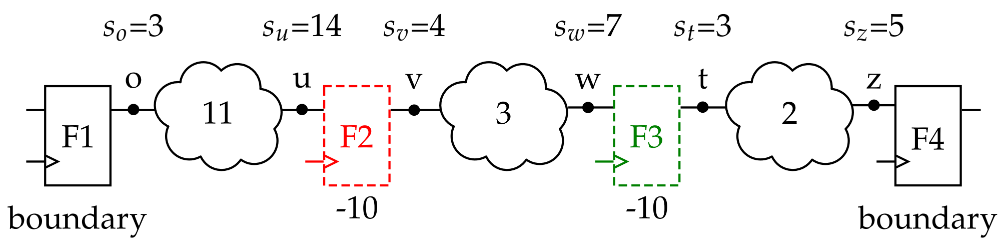


Figure 6.4: Concept of relative timing references. Clock period T=10. Clock-to-q delay t_{cq}=3. Both setup time t_{su} and hold time t_h are equal to 1. F3 is kept in the optimized circuit and F2 is not included.

6.1.4 Relative Timing References

In Figure 6.2(a), if all the logic gates and flip-flop F3 are considered as the the circuit under optimization, F1, F2 and F4 are thus the boundary flip-flops. No matter how signals inside the circuit propagate, the function of the whole circuit is still maintained if it can be guaranteed that for any input pattern at flip-flops F1 and F2 the circuit produces the same result at F4 at the same clock cycle as the original circuit.

Consider a general case in Figure 6.4, 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.

In the general case in Figure 6.4, it can also be observed 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. The locations of these removed flip-flops such as F2 and F3 are called ***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 Figure 6.4, 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 Figure 6.3(b). The arrival time after this sequential delay unit thus becomes $T + t_{cq}$=13. This signal at t in Figure 6.4 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.

6.1.5 Synchronizing Logic Waves by Delay Units

With all flip-flops removed from the circuit under optimization, only signals that are so fast that they reach boundary flip-flops too early need to be delayed; 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, this task is formulated 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 Figure 6.5, 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.

Combinational delay unit and gate sizing

In Figure 6.5, the delay at the circuit node can be changed by sizing the delay of the logic gate, e.g., the XOR gate in Figure 6.5. 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, it is assumed that 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 $r^l < 1$.

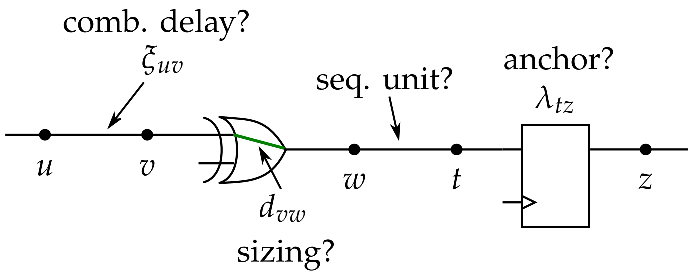


Figure 6.5: Delay insertion model in VirtualSync.

Insertion of sequential delay units

Since arrival times through long and short paths reaching w may have a large difference, it might be required 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 Figure 6.3, 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 Figure 6.5, 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 Figure 6.3(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, the arrival times s_w and s'_w need to be bound 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, it is assumed that the latches are transparent when the clock signal is equal to 0. The arrival times at w can then be bound in the same way as (6.7)–(6.8).

As illustrated in Figure 6.3(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 [SMO90],

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

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, the arrival times of fast signals can be imposed 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. An integer variable is used to represent the selection and let the solver determine which case happens during the optimization.

Reference shifting with respect to anchor points

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 Figure 6.5, the arrival time at z is shifted as

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

Wave non-interference condition

Since multiple waves are allowed to propagate along a combinational path, it needs to be guaranteed 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.

Overall formulation

The introduction of the relative timing references, or the anchor points, in Section 6.1.4 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 practically. Therefore, conservative approximation and relaxation techniques need to be developed.

6.1.6 Summary

The VirtualSync model views the timing optimization problem from a perspective different from the traditional flip-flop-based design flow. Its advantage is that only the minimum amount of delays need to be introduced into a combinational network generated by removing all sequential components from a given design. To find the proper locations to insert the delay units is, however, computation-intensive. Techniques such as circuit partition and iterative refinement with Lagrangian Relaxation may be applied, but to achieve a good tradeoff between quality of the results and runtime of this optimization process is still a demanding task.

6.2 Timing Camouflage 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.

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. In the following, a new perspective to counter counterfeiting based on reverse engineering is proposed. By integrating unconventional timing information, a netlist, even if recognized exactly through reverse engineering, does not function correctly anymore when a conventional timing scheme is assumed.

The advantages of the introduced method include: 1) The camouflaged netlist only works with a given set of timing information, which, however, is difficult to be recognized exactly by reverse engineering even with much additional effort and cost.

2) The camouflaged netlist only contains normal logic gates, so that it is challenging for attackers to isolate and then identify the timing encryption locations. 3) The introduced wave-pipelining false paths obstruct test-based counterfeiting methods further by camouflaging originally testable paths as false paths. 4) The proposed method is fully compatible with other security techniques introduced previously, so that they can be combined seamlessly.

In the following, the concept of timing camouflage will be discussed in detail. In Section 6.2.1, the motivation and the basic idea of the proposed method are explained. In Section 6.2.2, a detailed description of the wave-pipelining technique is provided. In Section 6.2.3, potential attack techniques are analyzed and counter measures to thwart them are proposed. Conclusions are stated in Section 6.2.4.

6.2.1 Motivation and Basic Concept of Timing Camouflage

Digital circuits rely on their structures to define their functions. A netlist is usually sufficient to reproduce a correctly working circuit. To prevent a netlist from being recognized by reverse engineering, techniques from physical level to netlist level can be applied to camouflage the logic. These methods, however, are still restricted to the conventional single-period clocking timing model so that attackers only need to recognize the netlist correctly.

In the conventional single-period clocking timing model, all the paths in a combinational block operate within one clock period. Figure 6.6(a) shows a part of a sequential circuit with three flip-flops F1, F2 and F3. At each sampling clock edge, assumed as the rising clock edge henceforth, the data at the inputs of the flip-flops are latched. To guarantee the correct operation of the flip-flops, the data at the input of a flip-flop must become stable t_{su} time before the rising clock edge, and the data must stay stable t_h time after the rising clock edge.

With the single-period clocking model, designers only need to guarantee that the logic functions of combinational blocks are correct without having to worry about the interaction between different clock stages. Consequently, the netlist carries all

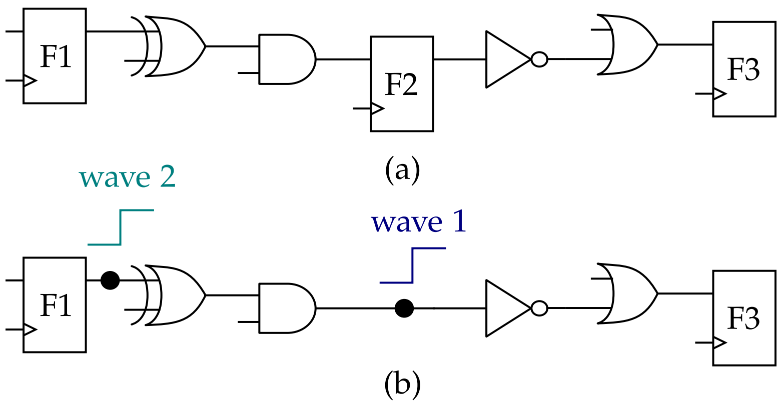


Figure 6.6: Conventional timing and wave-pipelining: (a) Single-period clocking; (b) Pipelining with two data waves.

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, it is proposed to invalidate the conventional timing model in some parts of the circuit. For example, the flip-flop in the middle of Figure 6.6(a) can be removed to construct the circuit in Figure 6.6(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 Figure 6.6(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 the circuit in Figure 6.6(b), at each rising clock edge, a new data is injected into the combinational path by flip-flop F1, so that the two waves are always separated by one clock period initially. To guarantee that the second data wave does not flush the first data wave when it is waiting for the next rising clock edge to be latched by F3, the path delay between F1 and F3 should be larger than one clock period. This path delay is, however, not contained in the extracted netlist from conventional reverse engineering. Consequently, the function of the circuit depends on both its structure and the timing of combinational paths.

Though attackers may have access to the standard cell library, e.g., through a third-party IP vendor, it is still very hard to obtain accurate interconnect/RC parasitics by delayering authentic chips, due to unknown process parameters, challenges in 3D RC extraction, and switching-window-dependent crosstalk-induced delay variations, etc. In any case, the more accurate the original timing information should be recognized from delayered chips, the harder and more expensive it becomes. In combination with other obfuscation methods, such as, dopant-level camouflage gate delay [BRPB14, VCPK17] and dummy contact insertion [MBPB15, RSSK13, RSK13], the unconventional timing concept has a potential to open up a new dimension of netlist security.

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.

6.2.2 Wave-Pipelining

A wave-pipelining path such as the one in Figure 6.6(b) allows two data waves propagating on the path at the same time. Since the second data wave should not catch the first one, special timing constraints should be specified for this path. The scenario of data wave propagation is illustrated in Figure 6.7. 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

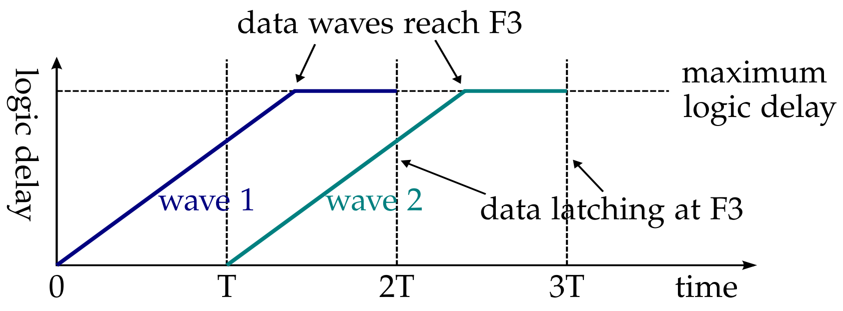


Figure 6.7: Temporal/spatial diagram for wave propagation on a combinational path.

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 Figure 6.7. Consequently, the two data waves on the path never interfere and F3 always latches the same value as in the original circuit shown in Figure 6.6(a).

In forming wave-pipelining paths, a flip-flop is removed from the circuit as in the example from Figure 6.6(a) to 6.6(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

$$d_p \geq T + t_h, \forall p \in P \iff \min_{p \in P}\{d_p - t_h\} \geq T \tag{6.18}$$

$$d_p \leq 2T - t_{su}, \forall p \in P \iff \max_{p \in P}\{d_p + t_{su}\} \leq 2T. \tag{6.19}$$

After removing a flip-flop from the circuit, if all the wave-pipelining paths meet the two constraints (6.18) and (6.19), the wave-pipelining version of the circuit is functionally equivalent to the original circuit.

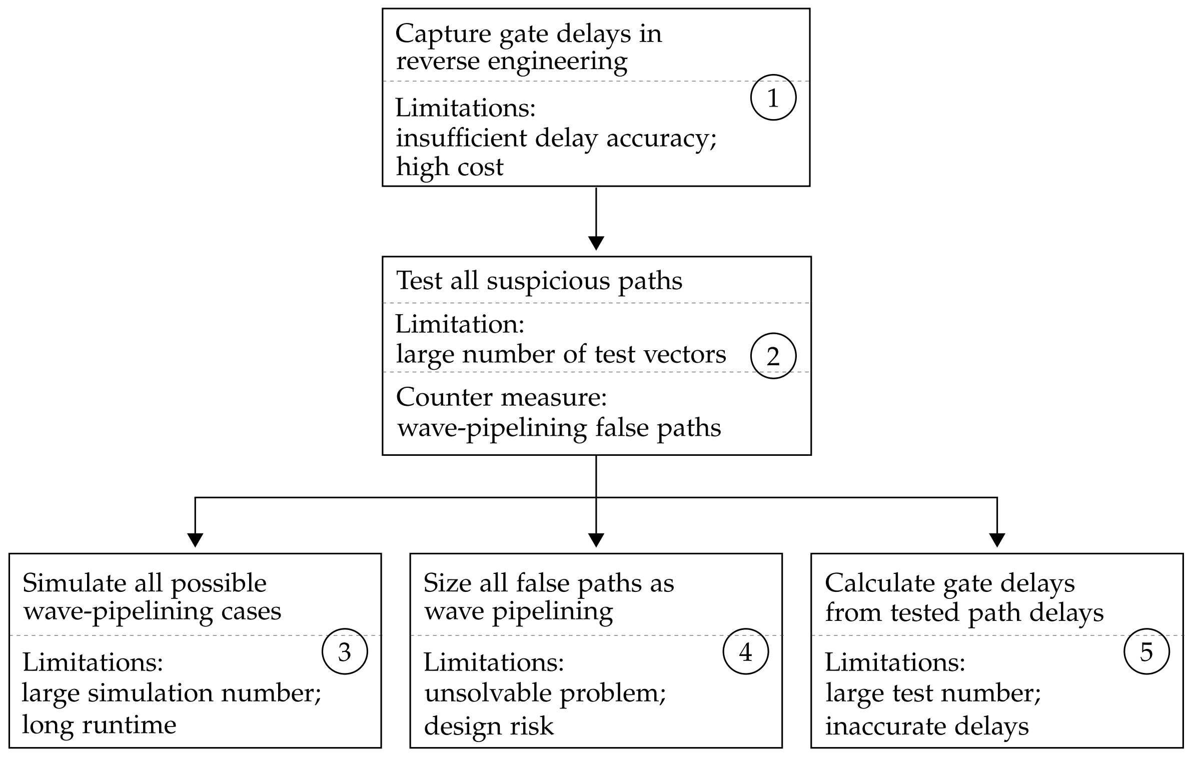

Capture gate delays in reverse engineering
Limitations: insufficient delay accuracy; high cost
1
Test all suspicious paths
Limitation: large number of test vectors
2
Counter measure: wave-pipelining false paths
Simulate all possible wave-pipelining cases
Limitations: large simulation number; long runtime
3
Size all false paths as wave pipelining
Limitations: unsolvable problem; design risk
4
Calculate gate delays from tested path delays
Limitations: large test number; inaccurate delays
5

Figure 6.8: Attack techniques to identify or circumvent wave pipelining, where the last three techniques may be combined to reduce the problem space of attack.

6.2.3 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 Figure 6.8.

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 wave-pipelining paths are between T and $2T$ as defined in (6.18) and (6.19), 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. 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,

$$(1-\tau)d \leq T \leq (1+\tau)d. \tag{6.20}$$

This path can only be considered as suspicious of wave-pipelining but without a clear differentiation. In the following, the range $[(1-\tau)d, (1+\tau)d]$ is called 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, it is also guaranteed that their delays are in the gray region.

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 introduced to create unsensitizable paths with wave-pipelining. When wave-pipelining paths are constructed by removing flip-flops, the paths that, viewed directly with the con-

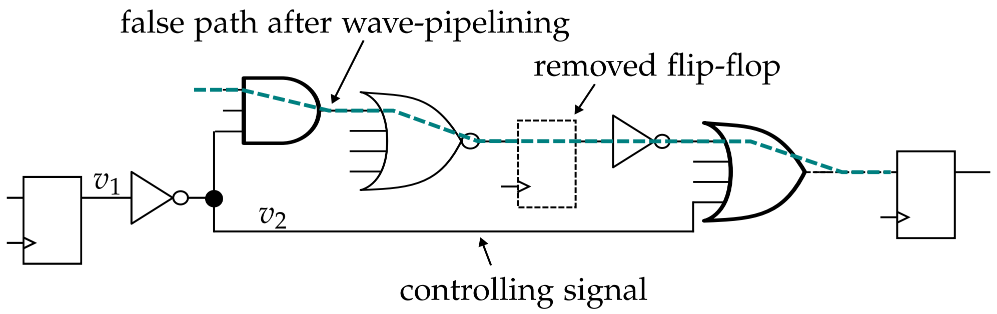


Figure 6.9: Two true paths form a wave-pipelining false path.

ventional single-period clocking, are false paths and cannot be sensitized by any test vectors are preferred.

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 Figure 6.9, 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, 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].

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.

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 (6.18) and (6.19). 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.

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. Potential methods such as machine learning might be applied to facilitate the attack, but further counter measures, e.g., logic and delay camouflage at gate level, may be combined with wave-pipelining paths.

6.2.4 Summary

The new timing camouflage technique above focuses on securing circuit netlists against counterfeiting by invalidating the assumption that the netlist represents the function of a circuit completely [ZLY+18]. Consequently, the difficulty of attack is increased significantly due to additional test cost and the introduced wave-pipelining false paths. This technique potentially opens up a new dimension of circuit security and it is fully compatible with all previous anti-counterfeiting methods.

When constructing wave-pipelining paths into a circuit while maintaining its original function, it should be guaranteed that the constructed paths meet the timing constraints (6.18) and (6.19). To counter potential attack techniques discussed above, 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.

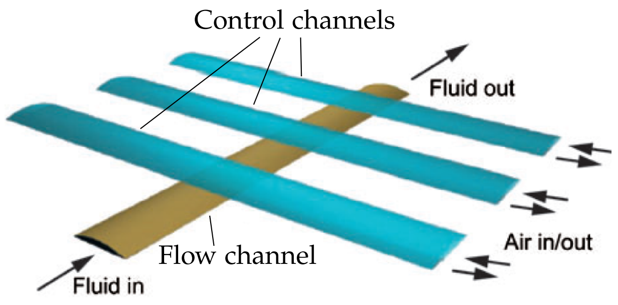
Future work in this direction includes incorporating gate delay camouflage by doping modification [BRPB14, VCPK17] to decouple gate delays from layout further. In addition, clock skew scheduling in [LS15, ZLS16c, ZLS16a, ZLL+18] would also be explored in the same timing dimension to enhance the security of netlists.

7 Sequential Design and Timing for Flow-based Microfluidic Biochips

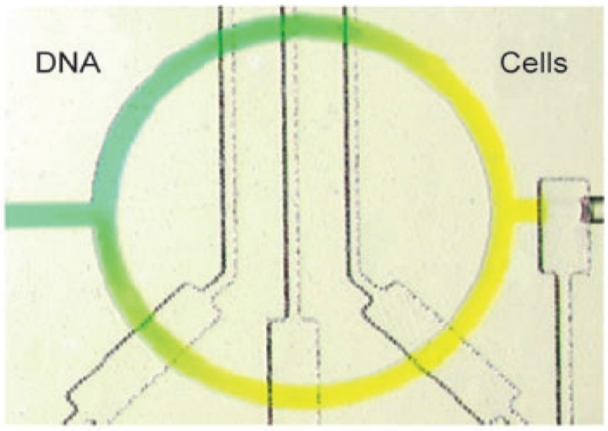
Microfluidic biochips are revolutionizing the traditional biochemical experiment flow with their high execution efficiency and miniaturized fluid manipulation [Per08, VR03, MQ07]. Devices are built in such a chip to execute specific operations, such as mixing and detection. Fluid samples are transported through microchannels between devices to carry out the protocol of a bioassay. All these functions are performed at the nanoliter level and controlled by a microcontroller without human intervention. The efficiency and reliability of such miniaturized and automated chips endow them with a great potential to improve human life significantly, and the research to bridge them with real-world applications is key to their success.

7.1 Background of Flow-based Biochips

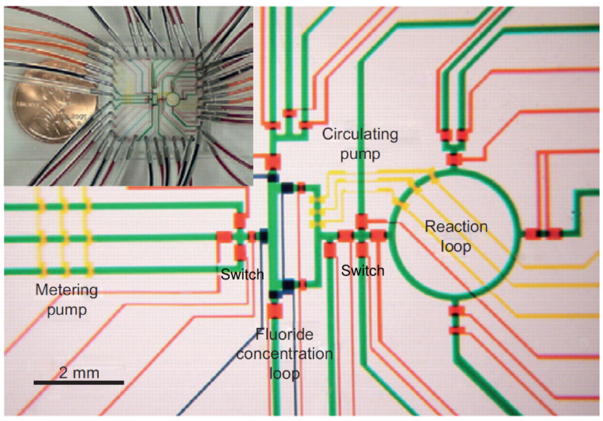
A flow-based microfluidic biochip is constructed from basic components such as microchannels and microvalves, henceforth named as channels and valves for simplicity. Flow channels are used to transport reaction samples and reagents between different locations. Above flow channels, control channels are built to conduct air pressure to intersections of flow channels and control channels to form valves, as illustrated in Figure 7.1(a), where three valves are constructed at the intersections. These channels are built from elastic materials, so that air pressure in a control channel can block the movement of fluid sample by squeezing the flow channel downwards. Conversely, if the pressure in the control channel is released, the fluid sample



(a)



(b)



(c)

Figure 7.1: Components and structure of flow-based biochips. (a) Valves constructed at intersections of flow/control channels [MQ07]. (b) Mixer [MQ07]. (c) A part of a biochip containing a mixer surrounded by a transportation channel network [EiSWd13].

can resume its movement. Since the channel width has been miniaturized down to 50 um [SHP+04] thanks to the advance of manufacturing technology, a huge number of channels and valves can already be integrated into a single biochip to perform large-scale experiments and diagnoses.

With valves as basic controlling components, complex devices can be constructed. For example, mixers can be built using channels and valves to execute mixing operations, which are very common in biochemical applications. The structure of a mixer is shown in Figure 7.1(b), where the three valves at the bottom are actuated alternately by applying and releasing air pressure in the control channels to mix fluid samples and reagents by peristalsis. After the mixing operation is completed, the resulting fluid sample can be stored in a dedicated storage unit temporarily.

In a biochip, devices executing specific operations, e.g., mixing and heating, are connected by channels so that intermediate reaction results can be transported between devices for processing. All these operations and transportation are controlled by a microcontroller, which issues instructions in a given order to actuate valves to move fluid samples and execute operations. Figure 7.1(c) shows a mixer (reaction loop) surrounded by flow channels (green), control channels (yellow and red) and valves (yellow and red blocks). These channels and valves together form a network similar to the road transportation system. If fluid channels should cross, valves are built to form a switch, as shown in Figure 7.1(c). At any moment, only two out of the four valves should be opened to direct fluid transportation; the other two valves must be closed to avoid fluid contamination. Consequently, the role of the valves at the intersection of flow channels is similar to that of the traffic lights in the road transportation system, while the open/closed states of the valves are controlled by a microcontroller according to the protocol of the application. The mixer and the channel network in Figure 7.1(c) are implemented into a biochip of the size comparable to that of a coin as shown at the upper left corner, demonstrating the miniaturized integration of microfluidic biochips.

In a biochip, the open/closed states of valves and the transportation of fluid samples are determined according to the biochemical application executed by the biochip. A biochemical application, or *bioassay* henceforth, is usually described with a *sequencing*

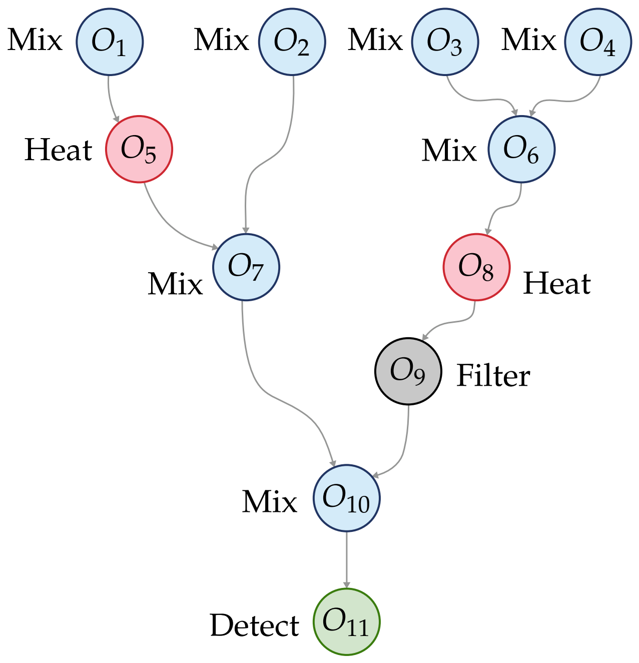


Figure 7.2: Sequencing graph of a bioassay.

graph $\mathcal{G} = (\mathcal{O}, \mathcal{E})$, such as in Figure 7.2, where $\mathcal{O}$ is the set of nodes and $\mathcal{E}$ is the set of edges. A node $O_i \in \mathcal{O}$ in the sequencing graph represents an operation, whose type τ_i and duration u_i are specified by the user. An edge $e_{ij} \in \mathcal{E}$ from O_i to O_j in the sequencing graph specifies that O_i must be executed before O_j and the result of O_i is the input of O_j. If O_i and O_j are executed by different devices, the required fluid transportation must be performed by the channel network between devices.

Biochips have a huge advantage over the traditional manual experiment flow, where operations performed by humans are error-prone and inaccurate. Any inadvertent mistake in this manual process might ruin a complex experiment that may last for several days. In a biochip, the volumes of fluid samples and reagents are controlled accurately and fluid samples are moved to target devices reliably, all of which are managed by a microcontroller exactly following a given protocol. In addition, the miniaturized size of biochips makes them extremely portable, so that a complex lab can be integrated into a single chip and carried conveniently to perform on-the-spot tests to counter acute disease outbreaks, such as the devastating Ebola virus disease a few years ago. Furthermore, reactions with fluid samples and reagents of tiny volumes take less time to complete than those with large volumes in tubes and droppers in the traditional experiment flow, so that biochips are also more responsive in dealing with urgent situations. Moreover, biochips save precious reagents by performing operations at nanoliter level. Such reagents may be exorbitantly expensive. For example, RNase inhibitor, a polyclonal antibody commonly used in reverse transcription polymerase chain reaction (RT-PCR), cost 600 euros per milliliter in December 2014 [Qia14].

Owing to their efficiency and cost-effectiveness, microfluidic biochips are reshaping many fields such as pharmacy, biotechnology and health care. In recent years, genomic bioassay protocols, such as nucleic-acid isolation, DNA purification and DNA sequencing, have been successfully demonstrated with microfluidic biochips. In addition, this technology has attracted a lot of commercial attention, e.g., from Illumina [Ill], a market leader in DNA sequencing. Accordingly, the International Technology Roadmap for Semiconductors (ITRS) 2015 [Int] has recognized the importance of microfluidic devices as having a rapid growth in the next several years.

Synthesis of microfluidic biochips with applications

In a biochip, operations are executed by a given number of devices with time multiplexing, described as a schedule. For example, an execution of the bioassay illustrated in Figure 7.2 is shown in Figure 7.3(a), where two mixers, one heater, one filter, and one detector are available. According to the schedule, the layout of a biochip, including the locations of devices and the transportation channels between them, can be determined to generate a physical design, as shown in Figure 7.3(b), where the devices are connected by a channel network controlled by valves.

The synthesis process above demonstrates that the schedule of operations of a bioassay determines the overall execution time. In addition, the fluid transportation between devices affects the structure of the channel network. Consequently, a holistic design automation flow is required to bridge the low-level components introduced by the microfluidics community with high-level real-world applications. In each step of this design automation flow, various design objectives should be optimized

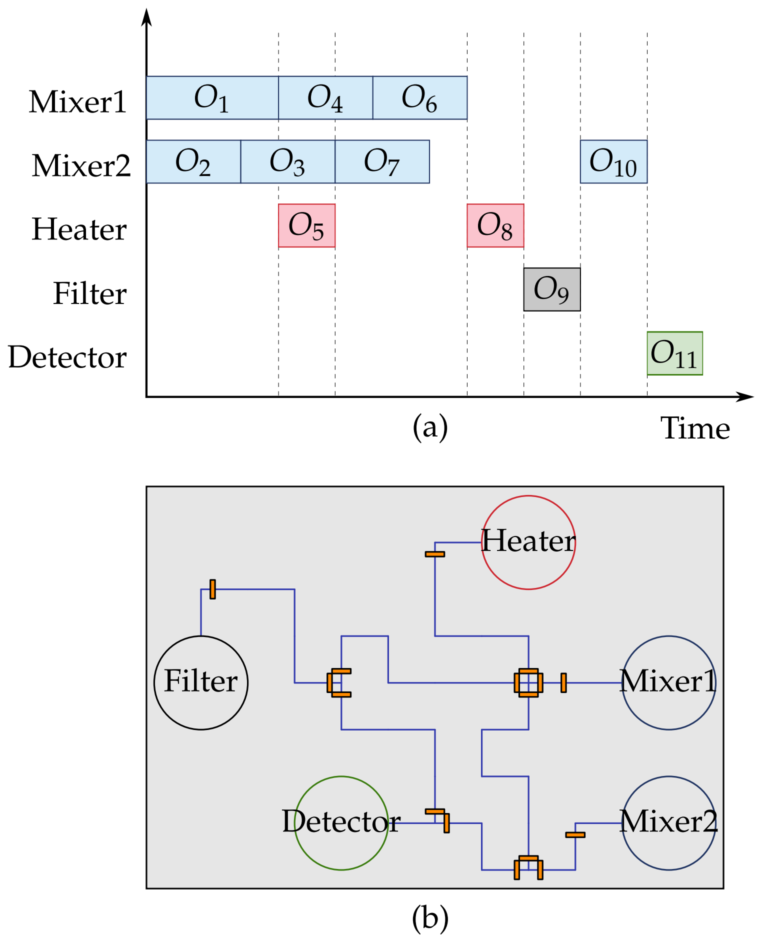


Figure 7.3: Synthesis of microfluidic biochips. (a) Scheduling. (b) Physical design.

to achieve an efficient architecture for the biochip.

The synthesis flow of biochips is similar to the synthesis flow for integrated circuits [Mic94]. Therefore, researchers in the electronic design automation community have started to expand into this area in recent years [CFZ10, PAC15]. However, these research efforts are still in an early stage and many unique characteristics of microfluidic biochips have still not been considered.

Flow-based microfluidic biochips: the electronic view

In the microfluidics community, researchers are focusing on developing new technologies and new structures to build fundamental components and devices, such as valves and pumps [UCT$^+$00, MGJ10]. Prototype microfluidic biochips are also built very often to demonstrate the function and performance of new components and new devices. Another major focus of the microfluidics community is to increase the integration density of basic components. With the advance in MEMS technology, a large number of components such as valves can now be built in a single biochip [AQ12]. Unfortunately, the abundant available resources have mostly been left unexplored, because end users cannot use them without a system layer that presents an interface for user applications, similar to the scenario that an operating system is missing for computer users. On the other hand, the effort of the microfluidics research community has been spread out in exploring even more technologies for microfluidic biochips, leading to a flourishing but fragmented panorama in the research on microfluidics.

The status quo of the microfluidics community is similar to the early period of the semiconductor industry. At that time, researchers were exploring different materials and device structures to build smaller but faster transistors. Thereafter, CMOS-based technology became dominant in this industry, while other technologies are employed only for specific applications. CMOS technology obtained its dominance because of, first of all, its performance. However, a very important factor which assisted this dominance is that the semiconductor industry and the electronic design automation community have found a way to carry out mass production of these devices and shrink the feature size continuously. In the meantime, the computer community has developed a successful computing model to present the available resources to end users and facilitate the development of high-level applications.

Observing the state of the art of microfluidic biochips, researchers from computer science and electrical engineering have started to bring their own computing models into microfluidic biochips. For example, the architecture of a microfluidic biochip from [ATA09] is shown in Figure 7.4. In this architecture, the mixer functions as the computing unit and intermediate results from the mixer are stored in the dedicated

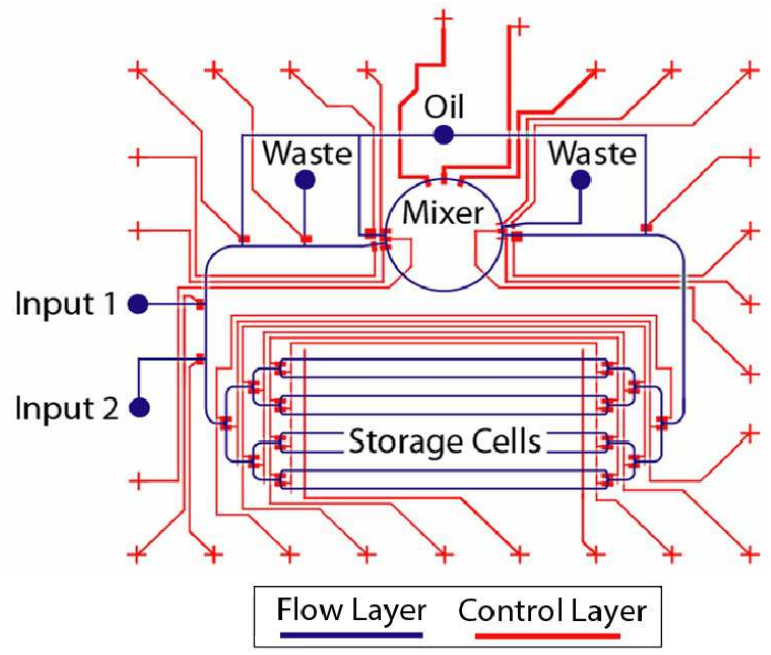


Figure 7.4: Computing-based biochip architecture containing a mixer and a dedicated storage unit with eight cells [ATA09].

storage unit. The cells in the storage unit are built from normal channels. At the ports of this storage unit, valves form multiplexers to direct fluid samples to enter into or leave from specific cells. This architecture emulates the classical von Neumann computer architecture to build a biochemical computing system from basic components. However, this simple emulation forsakes many unique characteristics of flow-based biochips, leading to inefficient execution of bioassays.

Similar to the semiconductor industry, design automation tool chains are also needed to support the development of microfluidic biochips. In recent years, the electronic design automation community has tried to migrate the existing design methodologies for integrated circuits to microfluidic biochip design, covering the phases from high-level synthesis down to physical design. Although this top-down flow has served the semiconductor industry in the past 50 years very successfully, fundamental changes should still be made to deal with specific requirements of biochips and take advantage of their unique features.

Flow-based microfluidic biochips: the unique characteristics

In microfluidic biochips, the inputs to an operation are fluid samples. Unlike electrical signals in integrated circuits, these fluid samples have a physical mass. In executing operations of a bioassay, fluid samples are processed with various operations, such as mixing, heating and detecting in different devices. The results of these operations are often fluid samples of different properties, so that inadvertent contamination between them should be avoided. The intermediate results of these operations should be stored in the chip temporarily in case they are not used immediately. Consequently, the physical mass and the variety of fluid samples become the major differences between biochips and integrated circuits, leading to several unique characteristics in biochip design.

Volume Management: In executing a bioassay, the volumes of fluid samples should be managed. Assume all the devices executing the bioassay in Figure 7.2 have a capacity v. Each of the resulting samples of O_1 and O_2 thus has a volume v. When these two fluid samples reach the device executing O_7, half of their volumes should be disposed of because the device only accepts a volume v. This volume management is not stated explicitly in the sequencing graph, but must be dealt with implicitly according to the volumes of intermediate fluid samples and the capacities of devices.

Storage management: In the schedule in Figure 7.3(a), O_2 completes before O_5 does. The intermediate result of O_2 should be moved out of Mixer2 and stored somewhere temporarily so that the mixer can execute O_3. In the biochip shown in Figure 7.4, this storage function is fulfilled by moving the result of O_2 to the dedicated storage unit through a channel. In synthesizing biochips, if operations are not scheduled properly, many storage requirements may appear, leading to many transportation channels and a large storage unit. In contrast to a dedicated storage unit as shown in Figure 7.4, the storage function can actually be implemented using distributed transportation channels. In fact, a fluid sample can stay anywhere in a channel in the biochip until it is used by the next operation. This is a significant difference between biochips and electronic systems, where intermediate data can only be stored in special memory units, either flip-flops or RAM components. This observation can be confirmed by the storage cells in the dedicated storage unit in Figure 7.4.

These cells are built of normal channels but with valves at each end of a channel to control the store/fetch operations. Instead of forming a monolithic storage unit, these channels and valves can actually be distributed in the chip so that they can be used for storage when required, and for transportation otherwise. Consequently, the efficiency of channels and valves can be improved significantly.

Washing: Unlike electrical signals, fluid samples leave residue in channels after they travel through them. Before such a channel is reused by another fluid sample, it should be washed by neutral fluids such as silicon oil. Washing contaminated channels can be very flexible because several channel segments can be washed simultaneously if they form a connected graph while being isolated from the rest of the biochip that is executing other operations.

Flow-layer and control-layer codesign: In a flow-based biochip, valves are controlled by air pressure through control channels, e.g., the red channels in Figure 7.4. If all the valves are controlled independently, the routing of control channels in a complex design becomes very complicated. To solve this problem, control channels of some valves can be shared if operations can still be executed correctly. This sharing requires a codesign of the flow layer and the control layer to match the actuation patterns of valves.

State-of-the-art research on design automation for flow-based biochips

In recent years, design and optimization methods for flow-based microfluidic biochips have started to appear. For high-level synthesis, the top-down flow in [MPMB12] generates a biochip architecture and minimizes the execution time of the bioassay, while the method in [TYLH13] minimizes valve switching activities during architectural synthesis. For scheduling and binding, the method in [DYHHA13] uses a maximum clique finding formulation to reduce assay execution time. In addition, the concept of general modeling of devices is introduced in [LTL+17] to improve the efficiency of the synthesis process, and special devices such as sieve valves are considered in [LTL+16]. Furthermore, fault-tolerance is considered during synthesis in [HGR+17] using a progressive optimization procedure.

For physical design of biochips, the method in [LLC$^+$14] considers obstacles during routing and solves the problem using a rectilinear Steiner minimum tree, while both routability and assay completion time are considered to achieve an efficient flow-layer design in [SHL16]. The placement of devices and routing of channels in flow-based biochips are dealt with simultaneously in [WRY$^+$16] using a sequence-pair representation, and they are formulated as a SAT problem in [GWY$^+$17] to achieve a close-to-optimal result.

Control logic synthesis is investigated in [MPMH13] to reduce the number of control pins. The method in [HDHC17] minimizes pressure propagation delay in the control layer to reduce the response time of valves and synchronize their actuations. Switching patterns of valves are examined in [WZY$^+$17, WXZ$^+$17] to reduce the largest number of switching activities in the control logic to avoid potential reliability problems. Furthermore, codesign of flow layer and control layer is investigated in [YWR$^+$15] to achieve valid routing results on both layers iteratively, and length-matching is incorporated in routing control channels in [YHC15] as well. Moreover, flow-layer, control-layer and valve switching are considered together in [TLL$^+$16b, TLF$^+$18] to simplify overall valve actuations.

To avoid contamination, washing is implemented in [HHC14b, HHC16] to clean devices and channel segments after they are used. This method still traces path sets and block-based partial washing has not been explored. The latter requires a co-optimization between operation scheduling and washing activities. The volume management problem in biochips has been explored in [ATV$^+$08] and [MRB$^+$14] for the specific bioassay sample preparation, but the optimization of volume management for general bioassays and the interaction of this task with fluid transportation for normal operations have not been taken into account.

To deal with manufacturing defects, fault models and an ATPG-based test strategy for flow-based biochips are proposed in [HHC13, HYHC14]. Design-for-testability and defect diagnosis are further addressed in [HHC14a, HBC15].

On system level, the concept of distributed channel storage in flow-based biochips is explored in [TLSH15, LLY$^+$17]. A more general architecture of biochips from [FM11], where valves instead of dedicated devices are built regularly and connected with short channels to implement Programmable Microfluidic Devices (PMDs) or Fully Programmable Valve Arrays (FPVAs), has been explored in [TLHS15, TLL^{+}16a] to provide better reliability and flexibility in executing bioassays. Though dynamic flow connections can be constructed on these chips relatively easily, channel crossing needs to be avoided [LLH18] and valve control sequences need to be arranged carefully [GKHW18]. A test generation for this new architecture is also proposed in [LLB^{+}17].

7.2 Design Automation for Flow-based Biochips of Large Integration

Similar to the industry of integrated circuits, the research in the microfluidics field has also led to a rapid increase of the integration of devices in a biochip. Already in 2008, a biochip array with 25K valves was accomplished [Per08], and recent advances in manufacturing technologies have achieved a valve density of 1 million per cm^2 [AQ12]. An integration of this scale can potentially enable the long-aspired exhaustive diagnoses in identifying the illness of a patient by testing pathological samples with thousands of reagents simultaneously. This breakthrough would not only reduce the inaccuracy in medical diagnoses, where individual expertise and experience of doctors play an important role, but also change the current guess-then-test model of medical treatment. In addition, such exhaustive diagnoses can be performed in small health-care centers routinely, due to the tremendously miniaturized chip size and lowered cost. With this exhaustive diagnosis model, illnesses can be detected at a very early stage and treatment cost can be reduced significantly as well.

7.2.1 Sequential Design for Flow-based Biochips

The state-of-the-art research on flow-based biochips exhibits some characteristics inherited from the research field electronic design automation for integrated circuits.

Operations in a bioassay are assigned to devices. These devices are distributed inside a given area and connected with channels. The overall objectives of optimization usually include the minimization of the completion time of the bioassay and the reduction of resource usage to achieve a smaller chip area or simpler peripheral circuits.

When dealing with biochips with a large integration, the scalability of existing problem formulations needs to be reexamined. For example, many existing methods use the completion time of a bioassay as one of the major optimization objectives. The durations of operations executed by devices are modeled individually and the overall performance of a biochip in executing a bioassay is determined by the longest chain of operations after scheduling and binding. When the numbers of operations, available devices and channels become very large, existing methods may not be able to reach the expected optimal solution and thus need to return non-optimal results instead. These results may deviate far from the ideal solution. As the scale of the whole system grows further, existing monolithic formulations might face the challenge that even no valid solution could be found.

To deal with the scalability problem, the operations in a sequencing graph can be partitioned into subgroups according to the relation between operations. For example, operations between which much fluid transportation exists should be grouped together. Thereafter, the subgroups are considered as basic modules and mapped to the biochip as a floorplan. After this mapping, each subgroup is optimized individually to improve the execution efficiency of the operations. The basic concept of this partitioning and synthesis is illustrated in Figure 7.5.

The objectives of this partitioning include: 1) operations in the sequencing graph should be partitioned in a way to reduce the communication between function subgroups; 2) the size of each subgroup should be kept moderate to reduce the complexity of synthesis; 3) the floorplan of the biochip should guarantee that subgroups requiring fluid transportation are located in close proximity; 4) the floorplan should consider special devices such as sensors and heaters.

During partitioning, the reliability of the biochip should also be considered. For example, during mixing the three valves in a mixer switch much more often to

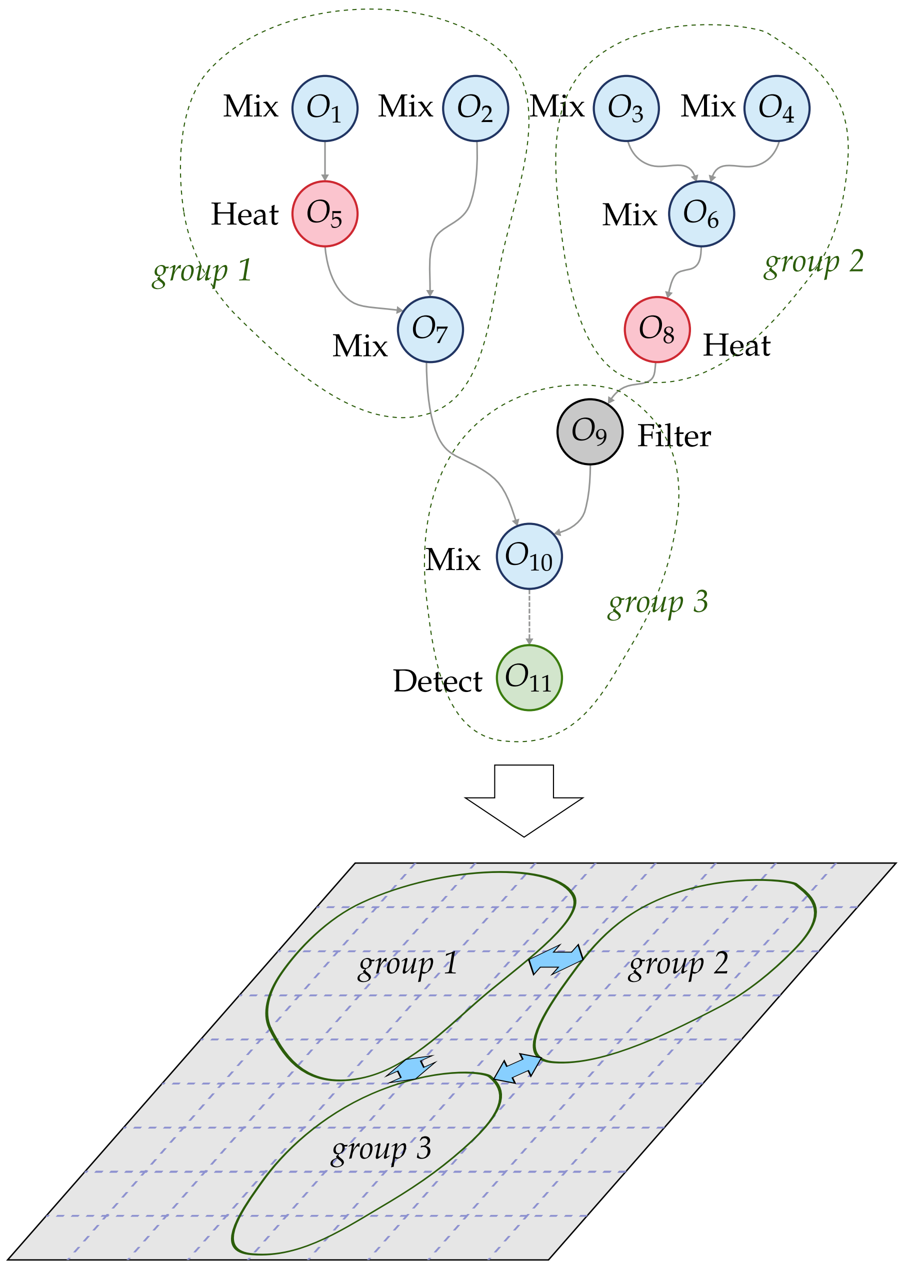


Figure 7.5: Hierarchical/Sequential design of flow-based biochips.

drive the circular flow inside the mixer than the other valves that only control fluid transportation. Consequently, these valves wear out more quickly than the others. To maintain a reliable function of the biochip, operations should be distributed evenly among devices of the same type to lower the worst-case wearout. This reliability issue should be taken into account as early as during partitioning to balance the mixing operations in the subgroups so that the overall reliability of the biochip can be improved.

The partitioning strategy described above is similar to sequential design in digital circuits, where combinational gates are the components that implement the functions, while flip-flops are majorly used as storage components to cache intermediate results. By splitting the whole combinational network of a design with flip-flops, design automation tools can thus optimize submodules individually. These submodules correspond to the subgroups of operations in a bioassay after partitioning. In a biochip, the caching function can be easily implemented by channels [TLSH15, LLY$^+$17]. To allow these submodules to be optimized independently, the function of a submodule should not depend on the order of fluid samples arriving at its input. If some intermediate results arrive earlier, they must stay in the caching channels and wait for the starting time of the submodule.

With this concept, a biochip essentially becomes a sequential system, emulating sequential design of digital circuits. For each submodule in a biochip, existing design and optimization methods can still be applied. The objective of the overall design and optimization thus becomes to guarantee that the results reach the input of submodules in time. In fact, it is not beneficial to produce a part of the intermediate results as early as possible. Instead, the optimal case happens when all the intermediate results arrive at the interface between submodules simultaneously. Therefore, the time of executing operation paths in a subgroup after partitioning should not vary much, similar to the critical path wall in a digital design. With this synchronization at the interface between submodules, the executions of submodules are thus decoupled.

The sequential design style is also flexible when facing process variations, which have become one of the major challenges of digital design in the nanometer era.

As the manufacturing technology for biochips advances, process variations may also come into play potentially, leading to deviation of execution time in different biochips after manufacturing. In addition, the stochastic nature of many biochemical operations also leads to variations in their finishing times. With the caching channels between modules, the execution of submodules can be aligned each time when intermediate results pass the interface between them, so that the correct function of the biochip can still be maintained.

When a biochip is viewed as a sequential design, its throughput can be also be optimized. Besides reducing its completion time, which corresponds to latency for a digital circuit, the volume of valid output produced in a given time unit can also be incorporated into the formulation of synthesis and optimized accordingly. This performance is important for applications such as sample preparation or in special scenarios, e.g., in large biochemical labs or medical centers.

7.2.2 Biochip Architectures for Sequential Design

When a bioassay is partitioned into submodules, each subgroup of operations needs to be optimized individually. In each subgroup, the number of operations and their types may differ. The operations in a subgroup are executed by devices built inside an area on the biochip. These devices are manufactured directly, like a full-custom design in the IC industry. This style, however, sacrifices design flexibility, since a subgroup of operations after partitioning are assigned to a given area in the biochip strictly. To reduce area cost, it may still be desirable to reuse some area of the biochip to execute different subgroups of operations, essentially creating a mixture of styles of hard-wired circuits and reconfigurable logic like FPGAs.

In order to allow devices built in an area of a biochip to process different operation groups, these devices need to be reconfigurable. This reconfiguration can partially be implemented by Programmable Microfluidic Devices (PMDs) or Fully Programmable Valve Arrays (FPVAs) [MQ07, FM11].

A part of the large valve array in [FM11] is shown in Figure 7.6(a) to demonstrate the architecture of FPVA. In this architecture, valves (solid blocks) are arranged in

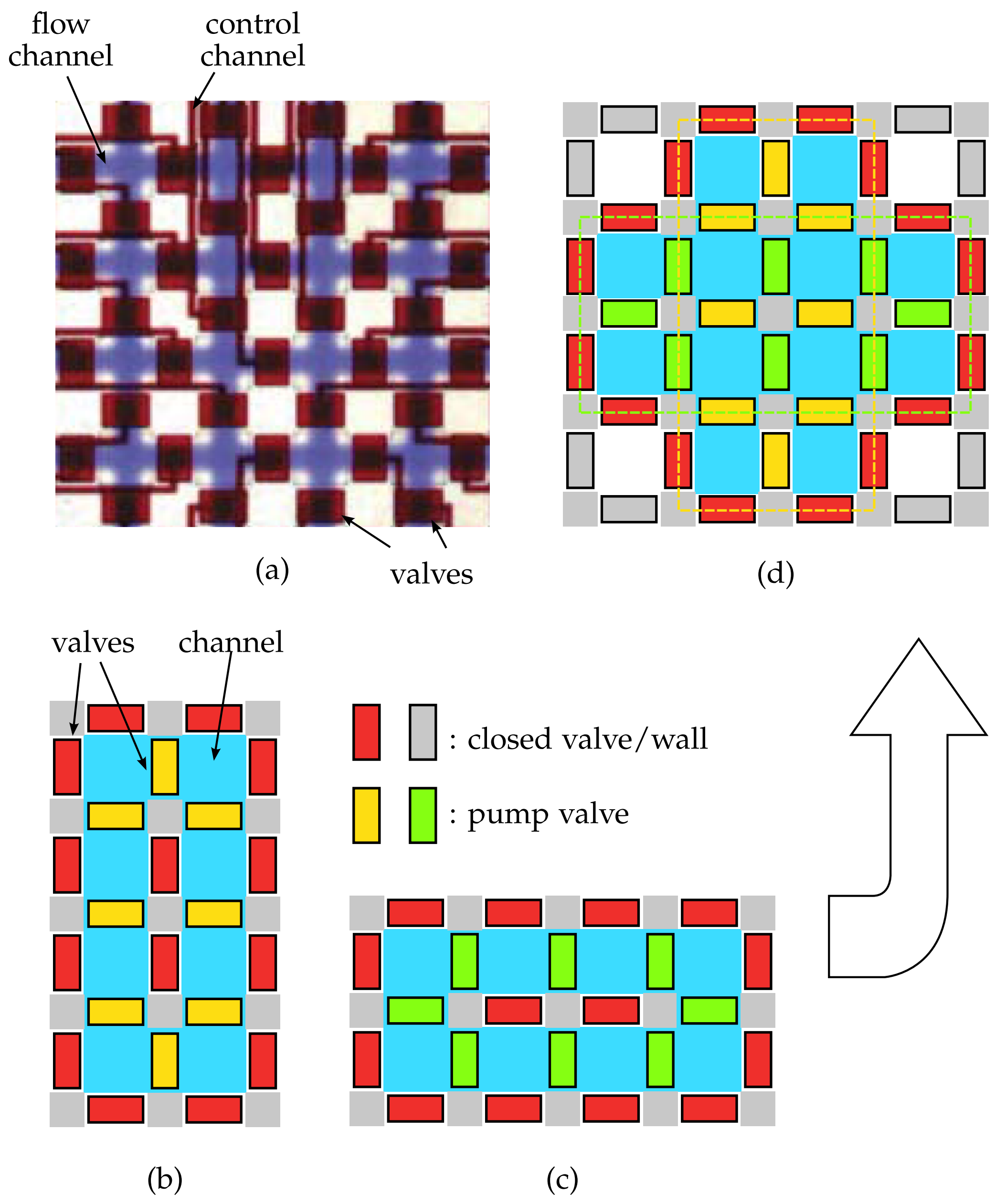


Figure 7.6: Fully programmable valve array (FPVA). (a) Architecture [FM11]. (b)/(c) A 4×2/2×4 dynamic mixer. (d) Dynamic mixers of different orientations sharing the same area.

a regular structure along horizontal and vertical flow channels (light color). These valves are controlled by air pressure sources through control channels (narrow channels). Transportation paths can be formed by opening and closing specific valves on the array, respectively. These channels also function as temporary storage caches.

Besides transportation channels, mixers can also be constructed on the valve array directly, taking advantage of the flexibility and reconfigurability of this architecture. For example, a 4×2 mixer and a 2×4 mixer can be constructed as in Figure 7.6(b) and 7.6(c), respectively. In such a dynamic mixer, the eight valves along the enclosed channel function as peristalsis valves, which switch in a given pattern to drive the fluid samples and reagents inside the channel for mixing, similar to the three valves in Figure 7.1(b). Compared with the traditional mixer in Figure 7.1(b), these dynamic mixers have a different shape and more peristalsis valves, eight in each case, to form a strong circular mixing flow. Moreover, the two mixers in Figure 7.6(b) and 7.6(c) can share the same area on the biochip as shown in Figure 7.6(d), provided that they are not used at the same time.

In short, a given area of the valve array can execute various functions such as mixing and flow transportation, as well as detection if the corresponding sensors are included in the area. This flexibility enables an independency of the partitioned submodules to low-level detail of chip realization, analogous to the case that all logic gates are built from an array of transistors in an integrated circuit.

7.2.3 Timing and Flow-based Biochips

Due to the advance of manufacturing technology, more resources are being integrated into a biochip rapidly. Accordingly, the working efficiency of individual devices may not be the first priority of design and optimization anymore. Instead, to use up the available resources to implement a highly efficient system with a large throughput is becoming another objective, similar to the trend manifested clearly in the IC industry.

To achieve this goal, it is already affordable to trade resources for design simplicity, and sequential design above a general architecture such as valve array may be a direction worth exploration. In this design style, operations in a sequencing graph can be treated as logic blocks. Since their functions, except for special devices such as heaters and filters, need not to be considered, the synthesis process thus becomes timing-driven, in which the original large-scale sequential graph is partitioned so that the finishing times of submodules, determined by the longest paths in the subgroups of operations, are well-balanced.

This timing interpretation allows more techniques from the circuit design domain to be migrated onto biochips. Since the channels at the interface between submodules function as flip-flops and the virtual devices between them can be considered as combinational logic blocks, the concepts such as aging analysis, monitoring and online-tuning can be applied similarly. Process variations can also be dealt with at path level instead logic gate level so that the pessimism in the worst-case design can be reduced.

In summary, the ever-increasing integration of biochips provides a potential to switch from a semi-analog design flow into a digital design flow. To exploit extreme performance from all devices in a biochip may be impractical anymore, and to trade the available abundant resources for throughput, robustness and design simplicity may be a potential way to deal with this ever-increasing integration. To pursue this direction or not still depends on the future development in the microfluidics field and its collaboration with the research field of electronic design automation for integrated circuits.

8 Conclusion

Already for several decades, it has been taken for granted that combinational logic blocks in a circuit perform computation, and sequential components, latches and flip-flops, are used to synchronize intermediate computation results. When process variations became prominent around 2000, this traditional timing model was extended by modeling the delays of logic gates, sequential components and interconnects as random variables. The essential definition of timing was, however, not modified. In timing analysis with process variations, only the statistical arrival times are propagated inside combinational blocks to calculate the latest and the earliest arrival times at flip-flops to verify setup time and hold time constraints. Since the delays and arrival times are random variables, timing constraints can only be met with a given probability. Accordingly, yield optimization of digital circuits has also become a popular research topic. With the advance of manufacturing technology, devices are becoming ever smaller. Consequently, aging effects make circuits slower after being stressed. These effects can also be modeled and incorporated into timing analysis and optimization to improve circuit reliability.

Statistical timing analysis and aging analysis, however, still play within the traditional framework of timing analysis. In static timing analysis, delays of logic gates and interconnects are constants representing the worst-/best-case corners. When process variations and aging are considered, these delays become correlated random variables. The original framework of static timing analysis, however, is still reserved, where statistical versions of sum, maximum and minimum computations are performed during timing propagation.

In the traditional timing framework, the performance of a circuit is normally constrained by the critical paths between flip-flops. Statistical timing optimization considering process variations and aging still respects the boundary of flip-flops generally and does not revise the fundamental single-period clocking scheme. After manufacturing, process variations may lead to chips in which the critical paths are neighbors of combinational paths with small delays. Since these paths are separated by flip-flops, the extra timing budget of the shorter paths cannot be used by the critical paths. Enlightened by this observation, post-silicon clock skew tuning moves clock edges according to the delays of paths after process variations or aging is experienced. This tuning technique essentially allows a better timing balance between neighboring sequential stages, but its effect is limited by the available tuning range of clock skews, the area taken by the tuning components and the cost of post-silicon test for tuning configuration. In addition, flip-flops still separate consecutive combinational paths even after post-silicon tuning. Consequently, their clock-to-q delays still contribute to the delays of critical paths and their setup times reduce clock frequency as well.

Considering the interdependency between setup time, hold time and clock-to-q delay of flip-flops is an interesting approach to relax the clear-cut separation between combinational logic blocks by flip-flops, since the timing budgets between sequential stages are balanced automatically without any cost of tuning. The potential of this technique is, unfortunately, limited, since the overall delay introduced into the circuit by flip-flops becomes even larger when this interdependency takes effect.

As the manufacturing technology advances into 10 nm node or below, variations in manufacturing and fragility of devices call for a new definition of timing. One of the possible expansions of the traditional timing framework is to reexamine the function of sequential components such as flip-flops. Since a flip-flop can only delay a signal instead of accelerating it, its function in the traditional digital design is actually to delay fast signals. Unfortunately, it also slows those signals traveling through critical paths. According to this observation, it might be possible to develop a new framework to delay fast signals at locations that are not on the critical paths. Since flip-flops on the original critical paths could be removed, forming wave-pipelining essentially, the corresponding clock-to-q delays and setup time requirements are also not needed anymore, leading to a further improvement of circuit performance beyond the limit of the traditional timing definition. Since there are no flip-flops between sequential stages of critical paths, this design style also tolerates process variations and aging automatically by allowing a delay compensation between sequential stages after manufacturing.

New timing schemes may also have the potential to benefit non-traditional fields such as hardware security. When flip-flops are removed from some paths in a digital design, the netlist of a circuit does not contain all the design information anymore. Even if an attacker recovers the netlist of a design by reverse-engineering, the locations of wave-pipelining still need to be identified. Without this information the traditional toolchains of digital design can only treat these paths as clocked by a single clock period. Consequently, the synchronization between sequential stages is lost and the counterfeit circuits cannot work correctly.

The IC industry has established a successful model in the past 40 years, where the down-scaling of transistors provides a huge amount of resources to build high-performance computing systems. As the huge number of transistors available to designers become nearly unmanageable, the focus of design methodologies has shifted from exploiting the potential performance of transistors to the extreme limit to trading the performance of individual transistors for design manageability. Consequently, the design style has evolved quickly from the early semi-analog design to digital design. This evolution may be followed by other fields such as microfluidic biochips, where the basic components are becoming smaller and the integration level is increasing rapidly. Gradually, biochips of large integration may also face the challenge of managing a huge amount of resources, and the sequential computing model of digital circuits may indicate a potential direction.

Viewed generally, timing is only one of the many ways to manage computing resources to keep design complexity under control. Essentially, it is similar to design abstraction from transistors to designing with Boolean logic and further up to hierarchical design. In each layer of this abstraction, only the necessary information from the lower level is kept. The overall design, however, may not be strictly optimal due to this information separation and abstraction. In the traditional timing definition, design information is split into different spheres. Since logic functions are not considered in the traditional timing model, analysis and optimization effort can be reduced significantly. As expected, this split sacrifices circuit performance potentially, since critical paths might not always be actuated during computing, but the clock period is still computed with respect to them. With this perspective, more flexible timing schemes might be introduced, where the barriers of sequential components can be managed with more freedom to achieve efficient and robust designs.

Bibliography

[ABZ$^+$02] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Path-based statistical timing analysis considering inter- and intra-die correlations. In *ACM/IEEE Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, pages 16–21, 2002.

[ABZ03a] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 900–907, 2003.

[ABZ$^+$03b] 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.

[App] Apple A10. https://en.wikipedia.org/wiki/Apple_A10.

[AQ12] I. E. Araci and S. R. Quake. Microfluidic very large scale integration (mVLSI) with integrated micromechanical valves. *Lab Chip*, 12:2803–2806, 2012.

[ATA09] N. Amin, W. Thies, and S. P. Amarasinghe. Computer-aided design for microfluidic chips based on multilayer soft lithography. In *Proc. Int. Conf. Comput. Des. (ICCD)*, pages 2–9, 2009.

[ATV$^+$08] A. M. Amin, M. Thottethodi, T. N. Vijaykumar, S. Wereley, and S. C. Jacobson. Automatic volume management for programmable microfluidics. In *Proc. Conf. Programming Language Design and Implementation*, pages 56–67, 2008.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. Int. Symp. Circuits and Syst. (ISCAS)*, pages 1929–1934, 1989.

[BCH$^+$15] A. Benhassain, F. Cacho, V. Huard, M. Saliva, L. Anghel, C. Parthasarathy, A. Jain, and F. Giner. Timing in-situ monitors: Implementation strategy and applications results. In *Proc. Custom Integr. Circuits Conf. (CICC)*, pages 1–4, 2015.

[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.

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

[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.

[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.

[CFZ10] K. Chakrabarty, R. B. Fair, and J. Zeng. Design tools for digital microfluidic biochips: Toward functional diversification and more than moore. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 29(7):1001–1017, 2010.

[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.

[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.

[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.

[CZNV05] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah. Parameterized block-based statistical timing analysis with non-Gaussian parameters, nonlinear delay functions. In *Proc. Design Autom. Conf. (DAC)*, pages 71–76, 2005.

[CZV$^+$08] R. Chen, L. Zhang, C. Visweswariah, J. Xiong, and V. Zolotov. Static timing: Back to our roots. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 310–315, 2008.

[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.

[DGY$^+$74] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc. Design of ion-implanted mosfets with very small physical dimensions. *IEEE J. Solid-State Circuits*, 9(5):256–268, 1974.

[DMS$^+$02] A. Daga, L. Mize, S. Sripada, C. Wolff, and Q. Wu. Automated timing model generation. In *Proc. Design Autom. Conf. (DAC)*, pages 146–151, 2002.

[DRS$^+$06] S. Das, D. Roberts, L. Seokwoo, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. A self-tuning DVS processor using delay-error detection and correction. *IEEE J. Solid-State Circuits*, 41(4):792–804, 2006.

[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.

[DYHHA13] T. A. Dinh, S. Yamashita, T.-Y. Ho, and Y. Hara-Azumi. A clique-based approach to find binding and scheduling result in flow-based microfluidic biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 199–204, 2013.

[EiSWd13] K. S. Elvira, X. C. i Solvas, R. C. R. Wootton, and A. J. deMello. The past, present and potential for microfluidic reactor technology in chemical synthesis. *Nature Chemistry*, (5):905–915, 2013.

[ENH09] T. Enami, S. Ninomiya, and M. Hashimoto. Statistical timing analysis considering spatially and temporally correlated dynamic power supply noise. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 28(4):541–553, 2009.

[ESN$^+$10] T. Enami, K. Shinkai, S. Ninomiya, S. Abe, and M. Hashimoto. Statistical timing analysis considering clock jitter and skew due to power supply noise and process variation. *IEICE Trans. on Fundamentals*, E93-A(12):2399–2408, 2010.

[FHMO12] H. Fuketa, M. Hashimoto, Y. Mitsuyama, and T. Onoye. Adaptive performance compensation with in-situ timing error predictive sensors for subthreshold circuits. *IEEE Trans. VLSI Syst.*, 20(2):333–343, 2012.

[FLZ07] Z. Feng, P. Li, and Y. Zhan. Fast second-order statistical static timing analysis using parameter dimension reduction. In *Proc. Design Autom. Conf. (DAC)*, pages 244–249, 2007.

[FM11] L. M. Fidalgo and S. J. Maerkl. A software-programmable microfluidic device for automated biology. *Lab Chip*, 11:1612–1619, 2011.

[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.

[GKHW18] A. Grimmer, B. Klepic, T.-Y. Ho, and R. Wille. Sound valve-control for programmable microfluidic devices. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, 2018.

[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.

[GVTG08] A. Goel, S. Vrudhula, F. Taraporevala, and P. Ghanta. A methodology for characterization of large macro cells and IP blocks considering process variations. In *Proc. Int. Symp. Quality Electron. Des. (ISQED)*, pages 200–206, 2008.

[GVTG09] A. Goel, S. Vrudhula, F. Taraporevala, and P. Ghanta. Statistical timing models for large macro cells and IP blocks considering process variations. *IEEE Trans. on Semiconductor Manufacturing*, 22(1):3–11, 2009.

[GWY+17] A. Grimmer, Q. Wang, H. Yao, T.-Y. Ho, and R. Wille. Close-to-optimal placement and routing for continuous-flow microfluidic biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 530–535, 2017.

[HAP08] S. Hatami, H. Abrishami, and M. Pedram. Statistical timing analysis of flip-flops considering codependent setup and hold times. In *Proc. Great Lakes Symp. VLSI (GLSVLSI)*, pages 101–106, 2008.

[HBC15] K. Hu, B. B. Bhattacharya, and K. Chakrabarty. Fault diagnosis for flow-based microfluidic biochips. In *Proc. VLSI Test Symp. (VTS)*, pages 1–6, 2015.

[HDHC17] K. Hu, T. A. Dinh, T.-Y. Ho, and K. Chakrabarty. Control-layer routing and control-pin minimization for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 36(1):55–68, 2017.

[HGR$^+$17] W.-L. Huang, A. Gupta, S. Roy, T.-Y. Ho, and P. Pop. Fast architecture-level synthesis of fault-tolerant flow-based microfluidic biochips. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 1671–1676, 2017.

[HHC13] K. Hu, T.-Y. Ho, and K. Chakrabarty. Testing of flow-based microfluidic biochips. In *Proc. VLSI Test Symp. (VTS)*, pages 1–6, 2013.

[HHC14a] K. Hu, T.-Y. Ho, and K. Chakrabarty. Test generation and design-for-testability for flow-based mVLSI microfluidic biochips. In *Proc. VLSI Test Symp. (VTS)*, pages 97–102, 2014.

[HHC14b] K. Hu, T.-Y. Ho, and K. Chakrabarty. Wash optimization for cross-contamination removal in flow-based microfluidic biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 244–249, 2014.

[HHC16] K. Hu, T.-Y. Ho, and K. Chakrabarty. Wash optimization and analysis for cross-contamination removal under physical constraints in flow-based microfluidic biochips. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 35(4):559–572, 2016.

[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.

[HKO01] A. Hyvärinen, J. Karhunen, and E. Oja. *Independent component analysis*. Wiley & Sons, 2001.

[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.

[HYHC14] K. Hu, F. Yu, T.-Y. Ho, and K. Chakrabarty. Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 33(10):1463–1475, 2014.

[Ill] Illumina. http://www.illumina.com/.

[IMHO15] S. Iizuka, Y. Masuda, M. Hashimoto, and T. Onoye. Stochastic timing error rate estimation under process and temporal variations. In *Proc. Int. Test Conf. (ITC)*, pages 1–10, 2015.

[IMK$^+$13] S. Iizuka, M. Mizuno, D. Kuroda, M. Hashimoto, and T. Onoye. Stochastic error rate estimation for adaptive speed control with field delay testing. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 1–10, 2013.

[Int] International Technology Roadmap for Semiconductors. http://www.itrs2.net/.

[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.

[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.

[KM97] N. Kobayashi and S. Malik. Delay abstraction in combinational logic circuits. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 16(10):1205–1212, 1997.

[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.

[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.

[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.

[LLB$^+$17] C. Liu, B. Li, B. B. Bhattacharya, K. Chakrabarty, T.-Y. Ho, and U. Schlichtmann. Testing microfluidic fully programmable valve arrays (FPVAs). In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 91–96, 2017.

[LLC$^+$14] C.-X. Lin, C.-H. Liu, I.-C. Chen, D. T. Lee, and T.-Y. Ho. An efficient bi-criteria flow channel routing algorithm for flow-based microfluidic biochips. In *Proc. Design Autom. Conf. (DAC)*, pages 141:1–141:6, 2014.

[LLCP08] X. Li, J. Le, M. Celik, and L. T. Pileggi. Defining statistical timing sensitivity for logic circuits with large-scale process and environmental variations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(6):1041–1054, 2008.

[LLH18] G.-R. Lai, C.-Y. Lin, and T.-Y. Ho. Pump-aware flow routing algorithm for programmable microfluidic devices. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, 2018.

[LLY$^+$17] C. Liu, B. Li, H. Yao, P. Pop, T.-Y. Ho, and U. Schlichtmann. Transport or store? Synthesizing flow-based microfluidic biochips using distributed channel storage. In *Proc. Design Autom. Conf. (DAC)*, pages 49:1–49:6, 2017.

[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.

[LPF99] X. Liu, M. C. Papaefthymiou, and E. G. Friedman. Maximizing performance by retiming and clock skew scheduling. In *Proc. Design Autom. Conf. (DAC)*, pages 231–236, 1999.

[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.

[LTL$^+$16] M. Li, T.-M. Tseng, B. Li, T.-Y. Ho, and U. Schlichtmann. Sieve-valve-aware synthesis of flow-based microfluidic biochips considering specific biological execution limitations. In *Proc. Design, Autom., and Test Europe Conf. (DATE)*, pages 624–629, 2016.

[LTL$^+$17] M. Li, T.-M. Tseng, B. Li, T.-Y. Ho, and U. Schlichtmann. Component-oriented high-level synthesis for continuous-flow microfluidics considering hybrid-scheduling. In *Proc. Design Autom. Conf. (DAC)*, pages 51:1–51:6, 2017.

[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.

[MGJ10] R. Mathies, W. Grover, and E. Jensen. Multiplexed latching valves for microfluidic devices and processors. In *US Patent 7,766,033*, 2010.

[Mic94] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.

[MKB02] C. W. Moon, H. Kriplani, and K. P. Belkhale. Timing model extraction of hierarchical blocks by graph reduction. In *Proc. Design Autom. Conf. (DAC)*, pages 152–157, 2002.

[MPMB12] W. H. Minhass, P. Pop, J. Madsen, and F. S. Blaga. Architectural synthesis of flow-based microfluidic large-scale integration biochips. In *Proc. Int. Conf. on Compilers, Architecture, and Synthesis for Embed. Sys.*, pages 181–190, 2012.

[MPMH13] W. H. Minhass, P. Pop, J. Madsen, and T.-Y. Ho. Control synthesis for the flow-based microfluidic large-scale integration biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 205–212, 2013.

[MQ07] J. Melin and S. Quake. Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annu. Rev. Biophys. Biomol. Struct.*, 36:213–231, 2007.

[MRB$^+$14] D. Mitra, S. Roy, S. Bhattacharjee, K. Chakrabarty, and B. B. Bhattacharya. On-chip sample preparation for multiple targets using digital microfluidics. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 33(8):1131–1144, 2014.

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

[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.

[OB04] M. Orshansky and A. Bandyopadhyay. Fast statistical timing analysis handling arbitrary delay correlations. In *Proc. Design Autom. Conf. (DAC)*, pages 337–342, 2004.

[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.

[PAC15] P. Pop, I. E. Araci, and K. Chakrabarty. Continuous-flow biochips: Technology, physical-design methods, and testing. *IEEE Design & Test*, 32(6):8–19, 2015.

[Per08] J. M. Perkel. Microfluidics: Bringing new things to life science. *Science*, 322(5903):975–977, 2008.

[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.

[Qia14] Qiagen, Inc. QIAGEN RNase Inhibitor, 2014.

[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.

[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.

[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.

[SHL16] Y.-S. Su, T.-Y. Ho, and D.-T. Lee. A routability-driven flow routing algorithm for programmable microfluidic devices. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 605–610, 2016.

[SHP+04] V. Studer, G. Hang, A. Pandolfi, M. Ortiz, W. F. Anderson, and S. R. Quake. Scaling properties of a low-actuation pressure microfluidic valve. *J. Appl. Phys.*, 95(1):393–398, 2004.

[SK07] T. Sato and Y. Kunitake. A simple flip-flop circuit for typical-case designs for DFM. In *Proc. Int. Symp. Quality Electron. Des. (ISQED)*, pages 539–544, 2007.

[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.

[SMO90] 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.

[SR07] S. Srivastava and J. S. Roychowdhury. Interdependent latch setup/hold time characterization via Euler-Newton curve tracing on state-transition equations. In *Proc. Design Autom. Conf. (DAC)*, pages 136–141, 2007.

[SR08] S. Srivastava and J. Roychowdhury. Independent and interdependent latch setup/hold time characterization via Newton-Raphson solution and Euler curve tracking of state-transition equations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(5):817–830, 2008.

[SS06] J. Singh and S. Sapatnekar. Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis. In *Proc. Design Autom. Conf. (DAC)*, pages 155–160, 2006.

[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, 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.

[TLF$^+$18] T.-M. Tseng, M. Li, D. N. Freitas, T. McAuley, B. Li, T.-Y. Ho, I. E. Araci, and U. Schlichtmann. Columba 2.0: A co-layout synthesis tool for continuous-flow microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2018.

[TLHS15] T.-M. Tseng, B. Li, T.-Y. Ho, and U. Schlichtmann. Reliability-aware synthesis for flow-based microfluidic biochips by dynamic-device mapping. In *Proc. Design Autom. Conf. (DAC)*, pages 141:1–141:6, 2015.

[TLL$^+$16a] T.-M. Tseng, B. Li, M. Li, T.-Y. Ho, and U. Schlichtmann. Reliability-aware synthesis with dynamic device mapping and fluid routing for flow-based microfluidic biochips. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 35(12):1981–1994, 2016.

[TLL$^+$16b] T.-M. Tseng, M. Li, B. Li, T.-Y. Ho, and U. Schlichtmann. Columba: Co-layout synthesis for continuous-flow microfluidic biochips. In *Proc. Design Autom. Conf. (DAC)*, pages 147:1–147:6, 2016.

[TLSH15] T.-M. Tseng, B. Li, U. Schlichtmann, and T.-Y. Ho. Storage and caching: Synthesis of flow-based microfluidic biochips. *IEEE Design & Test*, 32(6):69–75, 2015.

[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.

[TYLH13] K.-H. Tseng, S.-C. You, J.-Y. Liou, and T.-Y. Ho. A top-down synthesis methodology for flow-based microfluidic biochips considering valve-switching minimization. In *Proc. Int. Symp. Phys. Des. (ISPD)*, pages 123–129, 2013.

[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.

[UCT$^+$00] M. A. Unger, H.-P. Chou, T. Thorsen, A. Scherer, and S. R. Quake. Monolithic microfabricated valves and pumps by multilayer soft lithography. *Science*, 288(5463):113–116, 2000.

[VCPK17] A. V. Vinay C. Patil and S. Kundu. Manufacturer turned attacker: Dangers of stealthy trojans via threshold voltage manipulation. In *Proc. North Atlantic Test Workshop (NATW)*, pages 1–6, 2017.

[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.

[Vis07] C. Visweswariah. Within-die variations in timing: From derating to CPPR to statistical methods. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, 2007. Tutorial.

[VPMS97] S. Venkatesh, R. Palermo, M. Mortazavi, and K. A. Sakallah. Timing abstraction of intellectual property blocks. In *Proc. Custom Integr. Circuits Conf. (CICC)*, pages 99–102, 1997.

[VR03] E. Verpoorte and N. F. D. Rooij. Microfluidics meets MEMS. *Proc. IEEE*, 91(6):930–953, 2003.

[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.

[WRY$^+$16] Q. Wang, Y. Ru, H. Yao, T.-Y. Ho, and Y. Cai. Sequence-pair-based placement and routing for flow-based microfluidic biochips. In *Proc.*

Asia and South Pacific Des. Autom. Conf. (ASP-DAC), pages 587–592, 2016.

[WXZ$^+$17] Q. Wang, Y. Xu, S. Zuo, H. Yao, T.-Y. Ho, B. Li, U. Schlichtmann, and Y. Cai. Pressure-aware control layer optimization for flow-based microfluidic biochips. *IEEE Trans. Biomed. Circuits and Systems*, 11(6):1488–1499, 2017.

[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.

[WZY$^+$17] Q. Wang, S. Zuo, H. Yao, T.-Y. Ho, B. Li, U. Schlichtmann, and Y. Cai. Hamming-distance-based valve-switching optimization for control-layer multiplexing in flow-based microfluidic biochips. In *Proc. Asia and South Pacific Des. Autom. Conf. (ASP-DAC)*, pages 524–529, 2017.

[XS17] Y. Xie and A. Srivastava. Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction. In *Proc. Design Autom. Conf. (DAC)*, 2017.

[YCS05] J. Yang, L. Capodieci, and D. Sylvester. Advanced timing analysis based on post-OPC extraction of critical dimensions. In *Proc. Design Autom. Conf. (DAC)*, pages 359–364, 2005.

[YHC15] H. Yao, T.-Y. Ho, and Y. Cai. PACOR: practical control-layer routing flow with length-matching constraint for flow-based microfluidic biochips. In *Proc. Design Autom. Conf. (DAC)*, pages 142:1–142:6, 2015.

[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.

[YWR$^+$15] H. Yao, Q. Wang, Y. Ru, Y. Cai, and T.-Y. Ho. Integrated flow-control codesign methodology for flow-based microfluidic biochips. *IEEE Design & Test*, 32(6):60–68, 2015.

[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.

[ZCH$^+$05] L. Zhang, W. Chen, Y. Hu, J. A. Gubner, and C. C.-P. Chen. Correlation-preserved non-Gaussian statistical timing analysis with quadratic timing model. In *Proc. Design Autom. Conf. (DAC)*, pages 83–88, 2005.

[ZF02] J. Zejda and P. Frain. General framework for removal of clock network pessimism. In *Proc. Int. Conf. Comput.-Aided Des. (ICCAD)*, pages 632–639, 2002.

[ZLHS18] G. L. Zhang, B. Li, M. Hashimoto, and U. Schlichtmann. VirtualSync: Timing optimization by synchronizing 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.

[ZLS$^+$] 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 variations. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* doi: 10.1109/TCAD.2018.2818713.

[ZLS16a] 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.

[ZLS16b] 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.

[ZLS16c] 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.

[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.

[ZSL$^+$05] Y. Zhan, A. J. Strojwas, X. Li, L. T. Pileggi, D. Newmark, and M. Sharma. Correlation-aware statistical timing analysis with non-Gaussian delay distributions. In *Proc. Design Autom. Conf. (DAC)*, pages 77–82, 2005.

[ZZH$^+$06] S. Zhou, Y. Zhu, Y. Hu, R. Graham, M. Hutton, and C.-K. Cheng. Timing model reduction for hierarchical timing analysis. In *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, pages 415–422, 2006.

Publications of the author during the Habilitation period

Journal papers:

1. Grace Li Zhang, Bing Li, Yiyu Shi, Jiang Hu, Ulf Schlichtmann, "EffiTest2: Efficient Delay Test and Prediction for Post-Silicon Clock Skew Configuration under Process Variations", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), accepted
2. Tsun-Ming Tseng, Mengchu Li, Daniel Nestor Freitas, Travis McAuley, Bing Li, Tsung-Yi Ho, Ismail Emre Araci, Ulf Schlichtmann, "Columba 2.0: A Co-Layout Synthesis Tool for Continuous-Flow Microfluidic Biochips", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), accepted
3. Grace Li Zhang, Bing Li, Jinglan Liu, Yiyu Shi, Ulf Schlichtmann, "Design-Phase Buffer Allocation for Post-Silicon Clock Binning by Iterative Learning", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 37(2), pp. 392-405, February 2018
4. Bing Li, Masanori Hashimoto, Ulf Schlichtmann, "From Process Variations to Reliability: A Survey of Timing of Digital Circuits in the Nanometer Era", IPSJ Transactions on System LSI Design Methodology (T-SLDM), February 2018 (invited)
5. Qin Wang, Yue Xu, Shiliang Zuo, Hailong Yao, Tsung-Yi Ho, Bing Li, Ulf Schlichtmann, Yici Cai, "Pressure-Aware Control Layer Optimization for Flow-Based Microfluidic Biochips", IEEE Transactions on Biomedical Circuits and Systems (TBioCAS), 11(6), pp. 1488-1499 December 2017
6. Tsun-Ming Tseng, Bing Li, Ching-Feng Yeh, Hsiang-Chieh Jhan, Zuo-Min Tsai, Mark Po-Hung Lin, Ulf Schlichtmann, "An Efficient Two-Phase ILP-Based Algorithm for Precise CMOS RFIC Layout Generation", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 36(8), pp. 1313-1326, August 2017

7. Tsun-Ming Tseng, Bing Li, Mengchu Li, Tsung-Yi Ho, Ulf Schlichtmann, "Reliability-aware Synthesis with Dynamic Device Mapping and Fluid Routing for Flow-based Microfluidic Biochips", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 35(12), pp. 1981-1994, December 2016
8. Bing Li, Ulf Schlichtmann, "Statistical Timing Analysis and Criticality Computation for Circuits With Post-Silicon Clock Tuning Elements", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), vol. 34(11), pp. 1784-1797, November 2015
9. Tsun-Ming Tseng, Bing Li, Tsung-Yi Ho, Ulf Schlichtmann, "Storage and Caching: Synthesis of Flow-based Microfluidic Biochips", IEEE Design & Test (D&T), 32(6), pp. 69-75, December 2015
10. Tsun-Ming Tseng, Bing Li, Tsung-Yi Ho, Ulf Schlichtmann, "ILP-based Alleviation of Dense Meander Segments with Prioritized Shifting and Progressive Fixing in PCB Routing", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 34(6), pp. 1000-1013, June 2015

Papers in proceedings of conferences and workshops:

11. Grace Li Zhang, Bing Li, Ulf Schlichtmann, "Timing with Virtual Signal Synchronization for Circuit Performance and Netlist Security", IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2018 (special session)
12. Chunfeng Liu, Bing Li, Bhargab B. Bhattacharya, Krishnendu Chakrabarty, Tsung-Yi Ho, Ulf Schlichtmann, "Test Generation for Microfluidic Fully Programmable Valve Arrays (FPVAs) with Heuristic Acceleration", International Conference on IC Design and Technology (ICICDT), June 2018 (invited)
13. Chunfeng Liu, Bing Li, Tsung-Yi Ho, Krishnendu Chakrabarty, Ulf Schlichtmann, "Design-for-Testability for Continuous-Flow Microfluidic Biochips", ACM/IEEE Design Automation Conference (DAC), June 2018

14. Yu-Kai Chuang, Kuan-Jung Chen, Kun-Lin Lin, Shao-Yun Fang, Bing Li, Ulf Schlichtmann, "PlanarONoC: Concurrent Placement and Routing Considering Crossing Minimization for Optical Networks-on-Chip", ACM/IEEE Design Automation Conference (DAC), June 2018
15. Grace Li Zhang, Bing Li, Masanori Hashimoto, Ulf Schlichtmann, "VirtualSync: Timing Optimization by Synchronizing Logic Waves with Sequential and Combinational Components as Delay Units", ACM/IEEE Design Automation Conference (DAC), June 2018
16. Fengxian Jiao, Sheqin Dong, Bei Yu, Bing Li, Ulf Schlichtmann, "Thermal-Aware Placement and Routing for 3D Optical Networks-on-Chips", IEEE International Symposium on Circuits and Systems (ISCAS), May 2018
17. Grace Li Zhang, Bing Li, Bei Yu, David Z. Pan, Ulf Schlichtmann, "TimingCamouflage: Improving Circuit Security against Counterfeiting by Unconventional Timing", Design, Automation and Test in Europe (DATE), March 2018
18. Bing Li and Ulf Schlichtmann, "Reliability-aware Synthesis and Fault Test of Fully Programmable Valve Arrays (FPVAs)", IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), October 2017 (special session)
19. Baris Yigit, Grace Li Zhang, Bing Li, Yiyu Shi, Ulf Schlichtmann, "Application of Machine Learning Methods in Post-Silicon Yield Improvement", IEEE International System on Chip Conference (SOCC), September 2017
20. Mengchu Li, Tsun-Ming Tseng, Bing Li, Tsung-Yi Ho, Ulf Schlichtmann, "Component-Oriented High-Level Synthesis for Continuous-Flow Microfluidics Considering Hybrid-Scheduling", ACM/IEEE Design Automation Conference (DAC), June 2017
21. Chunfeng Liu, Bing Li, Hailong Yao, Paul Pop, Tsung-Yi Ho, Ulf Schlichtmann, "Transport or Store? Synthesizing Flow-based Microfluidic Biochips using Distributed Channel Storage", ACM/IEEE Design Automation Conference (DAC), June 2017

22. Chunfeng Liu, Bing Li, Bhargab B. Bhattacharya, Krishnendu Chakrabarty, Tsung-Yi Ho, Ulf Schlichtmann, "Testing Microfluidic Fully Programmable Valve Arrays (FPVAs)", Design, Automation and Test in Europe (DATE), March 2017

23. Qin Wang, Shiliang Zuo, Hailong Yao, Tsung-Yi Ho, Bing Li, Ulf Schlichtmann, Yici Cai, "Hamming-Distance-Based Valve-Switching Optimization for Control-Layer Multiplexing in Flow-Based Microfluidic Biochips", IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), January 2017

24. Grace Li Zhang, Bing Li, Ulf Schlichtmann, "PieceTimer: A Holistic Timing Analysis Framework Considering Setup/Hold Time Interdependency Using A Piecewise Model", IEEE/ACM International Conference on Computer-Aided Design (ICCAD), November 2016

25. Qin Wang, Zeyan Li, Haena Cheong, Oh-Sun Kwon, Hailong Yao, Tsung-Yi Ho, Kwanwoo Shin, Bing Li, Ulf Schlichtmann, Yici Cai, "Control-Fluidic CoDesign for Paper-Based Digital Microfluidic Biochips", IEEE/ACM International Conference on Computer-Aided Design (ICCAD), November 2016

26. Robert Wille, Bing Li, Ulf Schlichtmann, Rolf Drechsler, "From Biochips to Quantum Circuits: Computer-Aided Design for Emerging Technologies", IEEE/ACM International Conference on Computer-Aided Design (ICCAD), November 2016 (special session)

27. Grace Li Zhang, Bing Li, Ulf Schlichtmann, "EffiTest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers", ACM/IEEE Design Automation Conference (DAC), June 2016 (nominated for Best Paper Award)

28. Tsun-Ming Tseng, Bing Li, Ching-Feng Yeh, Hsiang-Chieh Jhan, Zuo-Min Tsai, Mark Po-Hung Lin, Ulf Schlichtmann, "Novel CMOS RFIC Layout Generation with Concurrent Device Placement and Fixed-Length Microstrip Routing", ACM/IEEE Design Automation Conference (DAC), June 2016

29. Tsun-Ming Tseng, Mengchu Li, Bing Li, Tsung-Yi Ho, Ulf Schlichtmann, "Columba: Co-Layout Synthesis for Continuous-Flow Microfluidic Biochips", ACM/IEEE Design Automation Conference (DAC), June 2016
30. Grace Li Zhang, Bing Li, Ulf Schlichtmann, "Sampling-based Buffer Insertion for Post-Silicon Yield Improvement under Process Variability", Design, Automation and Test in Europe (DATE), March 2016
31. Mengchu Li, Tsun-Ming Tseng, Bing Li, Tsung-Yi Ho, and Ulf Schlichtmann, "Sieve-valve-aware Synthesis of Flow-based Microfluidic Biochips Considering Specific Biological Execution Limitations", Design, Automation and Test in Europe (DATE), March 2016
32. Ulf Schlichtmann, Masanori Hashimoto, Iris Hui-Ru Jiang, Bing Li, "Reliability, Adaptability and Flexibility in Timing: Buy a Life Insurance for Your Circuits", IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC), January 2016 (special session)
33. Tsun-Ming Tseng, Bing Li, Tsung-Yi Ho, Ulf Schlichtmann, "Reliability-aware Synthesis for Flow-based Microfluidic Biochips by Dynamic-device Mapping", ACM/IEEE Design Automation Conference (DAC), June 2015
34. Rohit Kumar, Bing Li, Yiren Shen, Ulf Schlichtmann, Jiang Hu, "Timing Verification for Adaptive Integrated Circuits", Design, Automation and Test in Europe (DATE), March 2015