



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Ph.D. DISSERTATION

METHODOLOGIES FOR RELIABLE
CLOCK NETWORKS FOR HIGH-SPEED
AND LOW-POWER DIGITAL SYSTEMS

저전력 고성능 디지털 시스템을 위한
고신뢰도의 클럭 네트워크 설계 방법론

BY

Hyungjung Seo

August 2015

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

METHODOLOGIES FOR RELIABLE
CLOCK NETWORKS FOR HIGH-SPEED
AND LOW-POWER DIGITAL SYSTEMS

저전력 고성능 디지털 시스템을 위한
고신뢰도의 클럭 네트워크 설계 방법론

BY

Hyungjung Seo

August 2015

DEPARTMENT OF ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

METHODOLOGIES FOR RELIABLE CLOCK
NETWORKS FOR HIGH-SPEED AND
LOW-POWER DIGITAL SYSTEMS

저전력 고성능 디지털 시스템을 위한
고신뢰도의 클럭 네트워크 설계 방법론

지도교수 김 태 환

이 논문을 공학박사 학위논문으로 제출함

2015 년 5 월

서울대학교 대학원

전기 컴퓨터 공학부

서 형 중

서 형 중의 공학박사 학위논문을 인준함

2015 년 6 월

위 원 장 _____
부위원장 _____
위 원 _____
위 원 _____
위 원 _____

Abstract

As the process variation is dominating to cause the clock timing variation among chips to be much large, conventional clock tree based clock network is not able to guarantee the timing constraint of a digital system. To overcome the limitations of traditional clock design techniques, various techniques have been studied. This dissertation addresses three techniques that have been widely used for designing robust clock network and proposes developed methods.

First, it is widely accepted that post-silicon tunable (PST) clock buffers can effectively resolve the clock timing violation. Since PST buffers, which can reset the clock delay to flip-flops after the chip is manufactured, impose a non-trivial implementation area and control circuitry, it is very important to minimally allocate PST buffers while satisfying the chip yield constraint. In this dissertation, we (1) develop a graph-based chip yield computation technique which can update yields very efficiently and accurately for incremental PST buffer allocation, based on which we (2) propose a systematic (bottom-up and top-down with refinement) PST buffer allocation algorithm that is able to fully explore the design space of PST buffer allocation.

Second, clock skew scheduling is one of the essential steps that must be carefully performed during the design process. This dissertation addresses the clock skew optimization problem integrated with the consideration of the inter-dependent relation between the setup and hold skews, and clk-to-Q delay of flip-flops, so that the time margin is more accurately and reliably set aside over that of the previous methods, which have never taken the integrated problem into account. Precisely, based on an accurate flexible model of setup skew,

hold skew, and clk-to-Q delay, we propose a stepwise clock skew scheduling technique in which at each iteration, the worst slack of setup and hold skews is systematically and incrementally relaxed to maximally extend the time margin.

Lastly, clock tree with cross links and clock spine have an intermediate characteristics for skew tolerance and power consumption, compared to clock tree and clock mesh which are two extreme structures of clock network. Unlike the clock tree with links between clock nodes, which is a sort of an incremental modification of the structure of clock tree, clock spine network is a completely separated structure from the structures of tree and mesh. Consequently, it is necessary and essential to develop a synthesis algorithm for clock spines, which will be compatible to the existing synthesis algorithms of clock trees and clock meshes. To this end, this dissertation first addresses the problem of automating the synthesis of clock-gated clock spines with the objective of minimizing total clock power while meeting the clock skew and slew constraints. The key idea of our proposed synthesis algorithm is to identify and group the flip-flops with tight correlation of clock-gating operations together to form a spine while accurately predicting and maintaining clock skew and slew variations through the buffer insertion and stub allocation.

In summary, this dissertation presents clock tuning techniques with consideration of post-silicon tuning, flexible flip-flop timing model, and clock-gated clock spine synthesis algorithm.

Keywords: VLSI&CAD, post-silicon tuning, static timing analysis, flexible flip-flop timing model, clock spine synthesis

Student Number: 2011-30237

Contents

Abstract	i
Chapter 1 INTRODUCTION	1
1.1 Clock Distribution Network	1
1.2 Process Variation	2
1.3 Flexible Flip-flop Timing Model	3
1.4 Clock Spine	3
1.5 Contributions of This Dissertation	6
Chapter 2 POST-SILICON TUNABLE CLOCK BUFFER AL- LOCATION BASED ON FAST CHIP YIELD COM- PUTATION	8
2.1 Introduction	8
2.2 Systematic Exploration of PST Buffer Allocation	10
2.2.1 Observations	10
2.2.2 Problem Definition	15
2.2.3 Allocation Algorithm	16
2.3 Fast Timing Yield Computation	17
2.3.1 Preliminaries	17

2.3.2	Incremental Yield Computation	22
2.4	Experimental Result	24
2.5	PST Buffer Configuration Techniques	31
2.6	Summary	32
Chapter 3	POST-SILICON TUNING BASED ON FLEXIBLE	
	FLIP-FLOP TIMING	34
3.1	Introduction	34
3.2	Preliminary and Definitions	40
3.2.1	Flexible Flip-Flop Timing Model	40
3.2.2	Definitions	40
3.3	Motivational Examples	42
3.4	Clock Skew Scheduling for Slack Relaxation Based on Flexible Flip-Flop Timing	46
3.4.1	Overall Flow	46
3.4.2	Finding Local Clock Skew Schedule	48
3.5	Experimental Results	51
3.6	Summary	57
Chapter 4	SYNTHESIS FOR POWER-AWARE CLOCK SPINES	61
4.1	Introduction	61
4.2	Preliminaries and Motivation	64
4.2.1	Clock Spine	64
4.2.2	Activity Patterns	67
4.2.3	Power Computation	67
4.3	Algorithm for Clock Spine Synthesis	68
4.3.1	Problem Definition	68
4.3.2	Power-Aware Sink Clustering	70

4.3.3	Spine Relaxation	77
4.3.4	Spine Buffer Allocation	80
4.3.5	Top-Level Tree Construction	86
4.4	Experimental Results	86
4.5	Summary	91
Chapter 5	CONCLUSION	95
5.1	Chapter 2	95
5.2	Chapter 3	95
5.3	Chapter 4	96
Bibliography		97
초록		106

List of Figures

Figure 1.1	Two extreme structures of clock networks in (a) and (b) and intermediate structures in (c) and (d) with terms of the clock skew variability and power dissipation.	5
Figure 2.1	The structure of PST buffer. The delay from <i>Input</i> to <i>Output</i> is adjusted by the delay control register which selectively switches the transistors according to the information from the TAP point. [11]	9
Figure 2.2	The relation between the number of PST buffers allocated and the resulting total PST area.	10
Figure 2.3	Classification of PST buffer allocations: (a) <i>overlapping</i> PST buffer allocation; (b) <i>redundant</i> PST buffer allocation; (c) <i>non-overlapping</i> and <i>non-redundant</i> PST buffer allocation of (a) and (b); (d) a possibly better overlapping PST allocation than (c).	13

Figure 2.4	An example illustrating the bottom-up exploration of PST buffer allocation. A refinement step after the last iteration follows, which iteratively checks if some of the PST buffers can be removed while meeting the yield constraint.	19
Figure 2.5	A portion of circuit with PST buffers to adjust the time to trigger flip-flops f_i and f_j	20
Figure 2.6	Constraint graph representing the timing relation between flip-flops: (a) a portion of circuit instance, I , with the allocation of PST buffers b_1 , b_2 , and b_3 that are used to adjust clock arrival times to flip-flops f_1 , f_2 , and f_3 , respectively; (b) constraint graph $G(I)$	21
Figure 2.7	Enumeration of all loops in an instance of PST buffer allocation and generation of reuse relation between loops and merging candidates: (a) an instance, I , of PST buffer allocation in a clock tree. Six PST buffers $b_1, \dots, b_4, b_{5,6}, b_{7,8}$ are allocated; (b) constraint graph $G(I)$ and PST buffer merging candidates MG_1 , MG_2 , and MG_3 in $G(I)$; (c) list L of loops in $G(I)$; (d) reuse table $R[-, -]$ to represent the T_l reuse relation between the loops and merging candidates.	26
Figure 2.8	Generation of a sequence for <i>max</i> operations. Reusable partial max results for trials of merging candidates are: (a) initial result table R ; (b) $\max\{T_{l_5}, T_{l_6}\}$ followed by $\max\{T_{l_1}, T_{l_2}\}$, $\max\{T_{l_3}, T_{l_4}\}$, and $\max\{T_{l_7}, T_{l_8}\}$; (c) $\max\{T_{(l_3, l_4)}, T_{l_5}, T_{l_6}\}$ ($= T_{(l_3, l_4, l_5, l_6)}$); (d) $\max\{T_{(l_1, l_2)}, T_{(l_3, l_4, l_5, l_6)}\}$ ($= T_{(l_1, \dots, l_6)}$); (e) $\max\{T_{(l_1, \dots, l_6)}, T_{(l_7, l_8)}\}$ ($= T_{(l_1, \dots, l_8)}$).	28

Figure 2.9	Comparing the exploration space and the number PST buffers allocated by exhaustive enumeration of all instances and our PST-alloc.	33
Figure 3.1	Description of setup skew t_S , hold skew t_H , and clk-to-Q delay t_Q for a flip-flop with input D and output Q [29]. The values of t_S and t_H of flip-flop f_i determine the value of t_Q of the flip-flop, which in turn affects the values of t_S and t_H of a flip-flop that will be driven by flip-flop f_i	35
Figure 3.2	Relation among setup skew, hold skew, and clk-to-Q delay [29].	36
Figure 3.3	<i>Setup curve</i> : the change of clk-to-Q delay t_Q as the setup skew t_S changes, in which the hold skew t_H is fixed to $-8ps$. <i>Hold curve</i> : the change of clk-to-Q delay t_Q as the hold skew t_H changes, in which the setup skew t_S is fixed to $37ps$	37
Figure 3.4	Three curves showing the trade-off between the setup skew t_S and hold skew t_H for clk-to-Q delay $t_Q = 73, 77, \text{ and } 92ps$. As the clk-to-Q delay is shortened, the flexibility of trading the setup skew with hold skew is reduced.	38
Figure 3.5	Comparison of the conventional STAs which use nonflexible flip-flop timing and the enhanced STAs ([33, 35, 34, 36]) using flexible setup and hold times, but not clk-to-Q delay.	43

Figure 3.6	Conventional STAs [37, 38] which <i>fully use flexible flip-flop timing</i> in which the inter-dependent relation between clk-to-Q delay, setup skew, and hold skew are utilized. Initially, clk-to-Q delay of a flip-flop is set ($t_Q(i) = 20$ in this example). Then, setup skew $t_S(j) = 100 - 20 - 70 = 10$ and hold skew $t_H(j) = 20 + 40 = 60$, from which $t_Q(j)$ ($= 60$) is computed. In turn, $t_S(i) = 100 - 60 - 30 = 10$ and $t_H(i) = 60 + 20 = 80$, from which $t_Q(i)$ is computed to 60. This large value of $t_Q(i)$ causes $t_S(j) = 100 - 60 - 70 = -30$, which is far less than 10 ($= t_{S_0}$). Thus, the STAs report a time failure at f_j	44
Figure 3.7	Clock skew scheduling integrated with flexible flip-flop timing. Tuning the clock arrival time at f_i to -5 causes to update the setup and hold skews, and clk-to-Q delays of flip-flops in the circuit, resulting in all nonnegative value of setup and hold slacks, indicating no time violation in the circuit.	44
Figure 3.8	A partial circuit structure with three flip-flops f_i , f_j , and f_k , in which CSS-FT wants to improve the worst slack time at f_j by rescheduling clock arrival time x_i at f_i or x_j at f_j while satisfying the three constraints in Step 2.	48
Figure 3.9	Illustration of curves, derived by Case 1 in Step 2 of CSS-FT, of the setup and hold skews ($t_S(i)$, $t_H(i)$, $t_S(j)$, $t_H(j)$) at flip-flops f_i and f_j in Fig. 3.8 with respect to x_i	49
Figure 3.10	Illustration of curves, derived by Case 2 in Step 2 of CSS-FT, of the setup and hold skews ($t_S(j)$, $t_H(j)$, $t_S(k)$, $t_H(k)$) at flip-flops f_j and f_k in Fig. 3.8 with respect to x_j	50

Figure 3.11	(a) Curve showing the change of the setup skew lower bound (t_{S_0}) as the hold skew changes. (t_{S_0} for a value of hold skew is used for calculating the setup slack in Eq (3.3)). (b) Curve showing the change of the hold skew lower bound (t_{H_0}) as the setup skew changes. (t_{H_0} for a value of setup skew is used for calculating the hold slack in Eq (3.5)).	51
Figure 3.12	The distribution of the numbers of setup and hold slacks of all flip-flops for designs s1423 and s5378 before (blue curve) and after (orange curve) the application of CSS-FT to the initial timing analysis results produced by the flexible flip-flop timing based STA in [37]. T_{clk} is set to $T_{clk0} \times 0.95$ in Table 3.1.	55
Figure 3.13	The distribution of time violations before and after the application of CSS-FT to the timing results of s1423 and s5378 produced by the STA in [37]. The blue dots indicate flip-flops.	58
Figure 3.14	The curves showing the trade-off between the minimum clock period and the area overhead required for clock skew rescheduling used by CSS-FT where the data are extracted and averaged over the results of ISCAS89 circuits.	59
Figure 3.15	The distributions of clk-to-Q delays of all flip-flops for designs s13207 and s38584 before (yellow points) and after (green points) the application of CSS-FT to the initial timing analysis results produced by the flexible flip-flop timing based STA in [37]. T_{clk} is set to T_{clk0} in Table 3.1. All the clk-to-Q delays of s13207 are decreased and only 10 clk-to-Q delays are slightly increased in the results of s38584.	60

Figure 4.1	Clock mesh synthesis given by <i>Synopsys IC Compiler</i> [53].	63
Figure 4.2	Comparison among clock spine and other two clock structures. Clock is transitted from clock source(red triangle) to every clock sinks. Compared to clock tree which has a single clock path for each clock source to sink, clock mesh and clock spine structure have multiple clock paths. Buffers in clock tree only drive their own isolated subtree but those in clock mesh and clock spine structure drive mesh grid or spine together.	65
Figure 4.3	An example illustrating activity patterns and clock gating in clock tree and clock spine networks. (a) The generation of activity pattern by the bottom-up process, in a clock tree, from the activity patterns of sinks. (b) The clock tree gating at node n_{10} in (a). (c) A clock spine network with two spines being gated separately.	66
Figure 4.4	Comparison of the structure and energy consumption for three instances of clock spine network. (a) A clock spine network with $ R = 3$. (b) A clock spine network different from that in (a) and $ R = 3$, but less energy consumption. (d) A clock spine network different from that in (a), but $ R = 5$ and less energy consumption.	69
Figure 4.5	Four steps of the proposed power-aware clock spine synthesis algorithm.	71
Figure 4.6	Spine allocation and placement for covering four sinks f_1, f_2, f_3, f_4 . (a) Horizontal spine, (b) Vertical spine.	72
Figure 4.7	Two possible spines updated by attaching sink f_5 . (a) Before attachment. (b) Attachment to the left spine. (c) Attachment to right spine.	73

Figure 4.8 An example illustrating the step-by-step procedure of the *inner-loop* of sink clustering in CSPINE. (a) A spine network produced from the prior *outer-loop*. (b) Generation of a *dummy* spine for each sink. (c) From the first four iterations, f_4 , f_5 , and f_6 are respectively attached to the dummy spines of f_2 , f_1 , and f_3 to form non-dummy spines while f_7 still maintains a dummy spine. (d) From the next three iterations, f_8 , f_9 , and f_{10} are attached to non-dummy spines. (e) The final clock spine network. 76

Figure 4.9 Example illustrating the concept of *isolated sink*. (a) A spine with four sinks with activation patterns. (b) The sinks to be enabled in power mode m_1 . (c) The sinks to be enabled in power mode m_2 . f_2 is an isolated sink in m_2 of the spine in (a). The spine alone has no control to mitigate the clock skew related to f_2 . (d) The sinks to be enabled in power mode m_3 . 79

Figure 4.10 Example showing the spine structure relaxation for two spines to improve the clock skew variability by removing isolated sinks. (a) Before relaxation. (b) After relaxation by *temporally* connecting them. t_g is a transmission gate. 81

Figure 4.11	<p>Example illustrating the application of dynamic programming for solving the problem of buffer allocation. (a) The specification of a spine with the unit-length capacitances of spine and stub, load capacitance of sinks, buffer library \mathcal{L} and spine position candidates. (b) Allocation of buffer b_{1x} to position 6, covering sub-spine $r[4, 8]$. (c) Allocation of buffer b_{2x} to position 6, covering a bigger sub-spine than that in (c). (d) An optimal allocation for the specification in (a). (e) An optimal allocation for the specification in (a) with more fine spine positions, producing a reduced cost of buffer allocation. (f) Decomposition of problem into two subproblems.</p>	84
Figure 4.12	<p>Comparison of clock spine and mesh structures of s13207. In (a) black dots, red dots, black lines, and green lines represent sinks, buffers, spines, and connection between spines. In (b) sky blue lines represent stubs.</p>	92
Figure 4.13	<p>Comparison of clock tree, clock mesh, and clock spine structures of circuit 04 for all power modes. Green lines, blue lines, and red lines represent clock tree, clock mesh, and clock spine networks, respectively.</p>	93

List of Tables

Table 1.1	The characteristics of clock spine.	6
Table 2.1	Comparison the accuracy of yield (Y) computation by Monte-Carlo simulation and PST-alloc. ($Y(\text{PST-alloc}) - Y(\text{MC})$) . . .	29
Table 2.2	Comparison of results by [12] and our PST-alloc.	30
Table 2.3	Comparison of the number of PST buffers by [12] and our PST-alloc (Top-down).	31
Table 3.1	Comparison of the numbers of setup and hold time violations, for various values of clock period, before and after the application of our CSS-FT starting from as input the timing analysis results produced by the flexible flip-flop timing based STA in [37]. Clock skew tunable range of each flip-flop is set to $[-30ps, 30ps]$	53
Table 3.2	Minimum clock period before and after the application of our CSS-FT. Clock skew tunable range of each flip-flop is set to $[-30ps, 30ps]$	54
Table 3.3	Running time and overhead of our CSS-FT.	56

Table 4.1	Comparison of the clock skew variability (μ, σ), wirelength and buffer area (WL, BA), and power consumption (PWR) of the clock trees with clock gating by [59], the clock mesh by [58] and a subsequent application of clock gating, and clock spines by CSPINE.	88
Table 4.2	Comparison of the clock skew variability (μ, σ), wirelength and buffer area (WL, BA), and power consumption (PWR) of the clock trees with clock gating by [59] when its slew rate constraints are changed.	90
Table 4.3	Comparison of the clock skew variability (μ, σ), wirelength and buffer area (WL, BA), and power consumption (PWR) of the clock spines by CSPINE.	91
Table 4.4	The analyzed characteristics of clock spine.	92

Chapter 1

INTRODUCTION

1.1 Clock Distribution Network

In a synchronous digital system, clock distribution network delivers a clock signal to clock sinks (i.e., flip-flops and latches). The signal should be arrived at all sinks at the same time. The objectives for clock network design is making the difference among clock latencies called (*global*) *clock skew* as zero, and it is very difficult task. Due to the effect of the increased variation caused by the CMOS process technology scaling down to sub-100nm, controlling clock skew became more challenging problem in electronic design automation (EDA).

In addition to the problem of increasing the performance of clock distribution networks, the power consumption of clock network should be carefully considered. According to the works of [1, 2], clock distribution network accounts for up to 40% of the total power. Increased requirements for designing robust clock network makes the power problem worse.

Traditional clock tree has been used for a long time for its simple structure.

However, increased variations produce uncertainties and the clock tree could not handle the problem at all. Clock mesh is proposed as an alternatives of clock tree. The structure gives multiple clock paths to clock sinks. Signals transitted to sinks are overlapped, thus the variation is decreased. However, multiple clock paths accompanies large cost, various researches to minimize the overhead have been studied.

1.2 Process Variation

The effect of process variation is one of the most important factors to be considered in the clock tree synthesis. As CMOS scaling down to sub-100nm, clock skew is regarded as an unpredictable value. Process variations to channel length/width, oxide thickness, threshold voltage, and wire width/thickness affect the delay variation of interconnect and it produces maximally 25% clock skew variation [3].

Traditionally, the worst-case timing analysis is used to consider the delay variation caused by the process variation. However, as the delay variation increases, the timing margin given by designer based on the analysis occupies a significant portion of clock timing, causing to degrade circuit performance. To cope with the worst-case timing analysis, the statistical static timing analysis (SSTA) has been developed.

While some researchers (e.g., [4, 5, 6]) focus on SSTA to analyze the effect of process variation, other researchers have tried to control and reduce design margin due to the process variation. One of the noticeable solutions is using Post-Silicon Tunable (PST) clock buffers [7]. A PST buffer is used to adjust clock signal delay after manufacturing. The details of PST buffer are going to be introduced in the Chapter 2.

1.3 Flexible Flip-flop Timing Model

Static Timing Analysis (STA) analyzes the timing paths of sequential circuit based on flip-flop timing parameters, which are composed of setup time and hold time, and clk-to-Q delay (delay from the clock signal to the output Q of flip-flop). The traditional STAs specify setup and hold time as lower bounds of setup and hold skews, to be satisfied to assure stable data propagation passing through flip-flops. Most STAs have used a particular pair of setup and hold times that produces a clk-to-Q delay as the lower bounds to facilitate the easy computation of timing analysis, even though there are many pairs of setup and hold time values for a clk-to-Q delay. To accurately analyze the timing of circuits, full understanding and exploitation of the inter-dependent relation of setup time, hold time, and clk-to-Q delay are required. In Chapter 3, the concept of flexible flip-flop timing parameters is introduced and clock skew optimization algorithm that exploits the new timing model is proposed.

1.4 Clock Spine

The role of clock distribution network is to deliver a clock signal from clock source to the clock sinks in a synchronous digital system. Due to the high complexity in clock distribution networks under the various variations in design and process parameters, the latencies of the clock paths to sinks become large in difference as the delay variation increases. The maximum imbalance among the clock latencies is referred to as *(global) clock skew* and it causes harmful effects on high performance digital systems. In general, the increase of clock skew degrades the clock speed of circuits, eventually failing circuit functions. Thus, it is very important for the circuit designers to reduce the clock skew or maintain the clock skew in a certain bound in synthesizing clock networks.

With the aim of mitigating the effect of the increased variation caused by the CMOS process technology scaling down to sub-100nm on the clock skew, designers and researchers have analyzed the structures of existing clock networks and attempted to explore alternative structures. Clock tree has been the most widely used clock network for its simplicity of the structure. Fig. 1.1(a) shows an example of the structure of clock tree where the yellow squares and blue triangles represent sinks and clock buffers, respectively. Since every clock sink receives the clock signal from the clock source through exactly one clock path, the delay variation on the clock wire and buffers in the clock path directly induces the variation of the clock arrival time at the sink, resulting in a high variability of clock skew. One extreme structure as opposed to that of clock tree to mitigate the variability is clock mesh, whose example of structure is shown in Fig. 1.1(b), since it allows multiple clock paths to a sink through the mesh grid [8].

Though the clock mesh has a high variation tolerance, its power consumption is a big obstacle. On the other hand, clock tree with cross links, shown in Fig. 1.1(c), offers reasonable solution that compromises power consumption with clock skew variation by inserting cross links to proper points in the clock tree [9]. Since the placement locations in the clock tree where links are added greatly influence the degree of clock skew variation, most existing works have looked for finding a minimal number of clock locations at which the links (dotted lines in Fig. 1.1(c)) are inserted to satisfy the constraint of clock skew variation. Besides clock tree with links, clock spine, shown in Fig. 1.1(d), is another compromising solution [10], but the structure is completely different from clock tree with links. The clock spine network contains a set of vertical and horizontal clock wires (heavy lines in Fig. 1.1(d)), which we call *spines* and every sink is attached to the spine in the closest distance. Since each spine

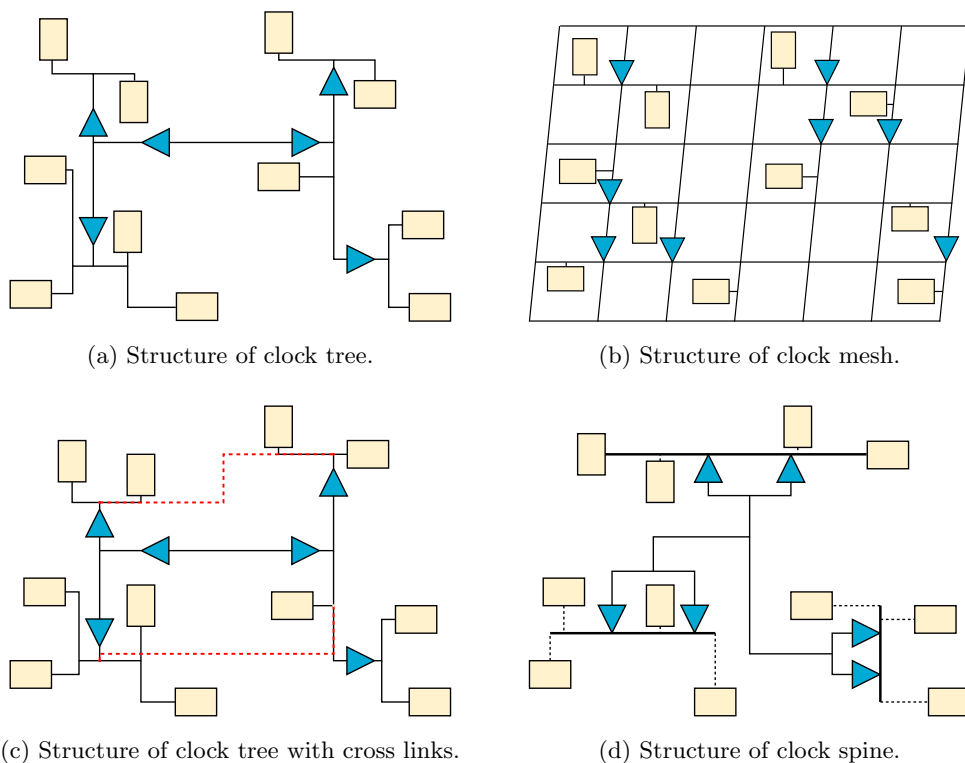


Figure 1.1: Two extreme structures of clock networks in (a) and (b) and intermediate structures in (c) and (d) with terms of the clock skew variability and power dissipation.

will have more than one clock buffer to drive the sinks attached in spine, (blue triangles in Fig. 1.1(d)), each sink can have multiple clock signal paths from the clock source. Unlike the clock tree with cross links, no work has investigated a systematic exploration of clock spine structures as yet. In Chapter 4, we develop a methodology to automate the exploration and synthesis of clock spine networks and fill the blanks in the Table. 1.1.

Table 1.1: The characteristics of clock spine.

	Clock tree	Cross links	Clock spine	Clock mesh
Performance	Low	Medium		High
Cost	Low	Medium	?	High
Analysis	Simple	Less simple		Complex

1.5 Contributions of This Dissertation

In this dissertation, each chapter presents algorithms and optimization techniques for solving the problems of clock network which relate to clock skew. At first, PST buffer allocation algorithm are developed in Chapter 2. To predict the efficiency of PST buffer in the design time, a graph-based yield computation technique is implemented in the algorithm. In Chapter 3, clock skew optimization with flexible flip-flop timing model is developed. Flexible timing model is integrated in the clock skew scheduling to resolve the timing violation. Finally, clock spine structure that is able to fix the skew problems in the early stage of design flow is presented in Chapter 4. The contributions of this dissertation are summarized as follows:

- Chapter 2 proposes a comprehensive graph-based PST buffer allocation algorithm. The algorithm includes (1) *a graph-based chip yield computation* technique for *incremental PST buffer allocations* and *a systematic PST buffer allocation algorithm* that is able to explore *more extended design space* of PST buffer allocation to minimize the number of allocated PST buffers.
- Chapter 3 analyzes flexible flip-flop timing that describes the relation of setup time, hold time, and clk-to-Q delay. In addition, *a new problem of*

clock skew optimization integrated with flexible flip-flop timing is addressed and a *step-by-step localized slack time relaxation* is developed for a solution to the problem.

- Chapter4 addresses the problem of power-aware clock spine synthesis. In the chapter, four synthesis steps, in which the key task is *to identify and group the flip-flops with tight correlation of clock-gating operations together to form a spine* while *accurately predicting and maintaining clock skew and slew variations through the buffer insertion and stub allocation*

Chapter 2

POST-SILICON TUNABLE CLOCK BUFFER ALLOCATION BASED ON FAST CHIP YIELD COMPUTATION

2.1 Introduction

The structure of a PST buffer is shown in Fig. 2.1 in which the delay is adjusted by the capacitors, each of which is controlled i.e., switch-on or switch-off by the control logic after manufacturing. The delay adjustment will be done through the TAP port. Since at the post-silicon stage, all process variations already affect the circuit, PST buffers can help fix the timing violations in the circuit, resulting in enhancing the chip yield. Tsai *et al.* [12] proposed a pre-silicon clock scheduling and a post-silicon PST buffer allocation in which they identified the ‘timing critical’ flip-flops and allocated PST buffers at the frond of the flip-flops. The determination of timing critical flip-flops is based on a certain threshold value given by designer. Subsequently, Tsai *et al.* [13] addressed two

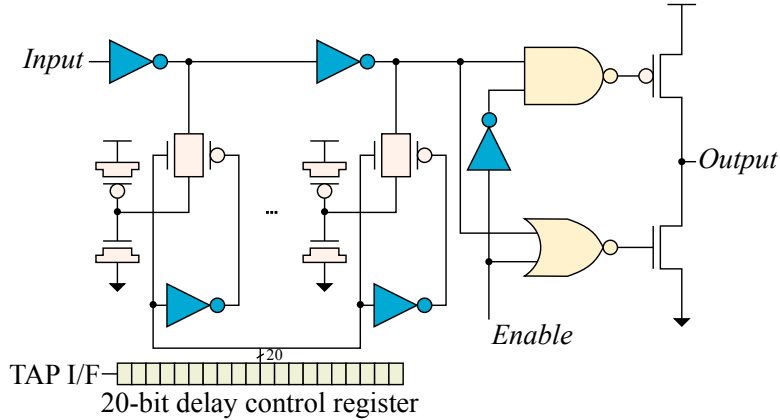


Figure 2.1: The structure of PST buffer. The delay from *Input* to *Output* is adjusted by the delay control register which selectively switches the transistors according to the information from the TAP point. [11]

problems: one is the problem is to minimize the total tunable delay range of PST buffers to be allocated under the target yield constraint, and the other problem to minimize the number of PST buffers to be allocated under the target yield constraint. The PST buffer allocation of their algorithm was performed according to the slack values of flip-flops, and the yields are calculated by the Monte-Carlo simulation. On the other hand, Khandelwal and Srivastava [14] combined gate sizing into PST buffer placement algorithm for a fixed number of PST buffers produced by the algorithm in [13]. The integrated algorithm used Monte-Carlo simulation to compute the yields. Nagaraj and Kundu [15] also studied PST buffer placement problem for a given number of PST buffers.

There are two critical limitations of the previous PST allocation methods [12, 13, 14, 15], which are *the use of slow Monte-Carlo simulation in computing the chip yield* and *a limited design space exploration of PST buffer allocation*. This chapter overcomes the two limitations in the PST buffer allocation. Precisely, (1) *we develop a graph-based chip yield computation technique which is*

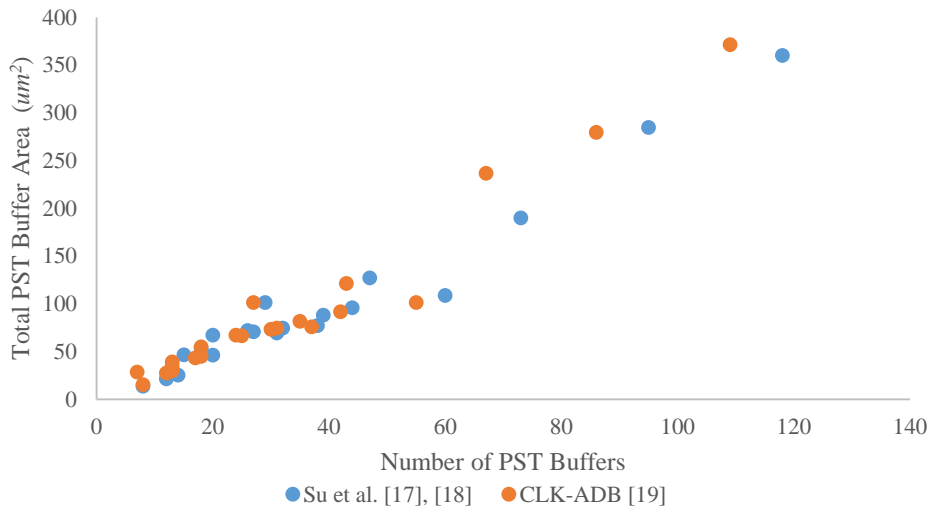


Figure 2.2: The relation between the number of PST buffers allocated and the resulting total PST area.

able to compute yields very efficiently and accurately for incremental PST buffer allocations, and based on the technique, (2) we propose a systematic (bottom-up and top-down) PST buffer allocation algorithm that is able to fully explore the design space of allocation to minimize the number of PST buffers. We have confirmed through experiments that our proposed algorithm drastically increasing chip yield while allocating limited number of PST buffers.

2.2 Systematic Exploration of PST Buffer Allocation

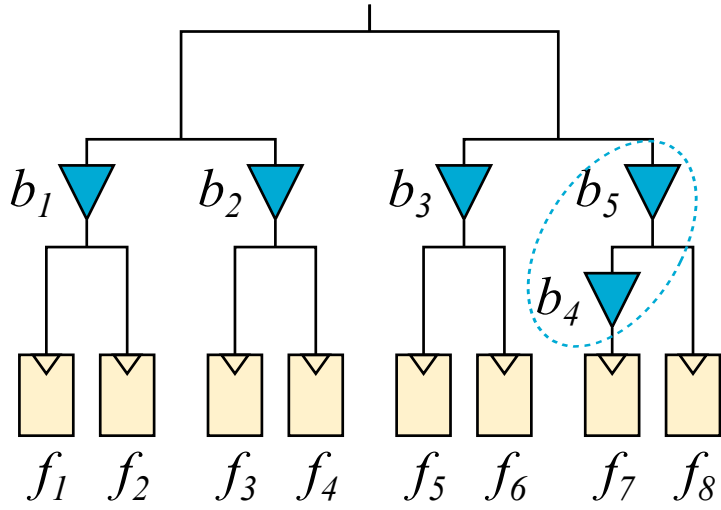
2.2.1 Observations

Due to the large area overhead of PST buffers, the use of PST buffers should be constrained. The area overhead of PST buffers is composed of two parts. One is the tunable range of PST buffers, the other is the number of PST buffers. For simplicity, we assume that all the PST buffers have the same the tunable

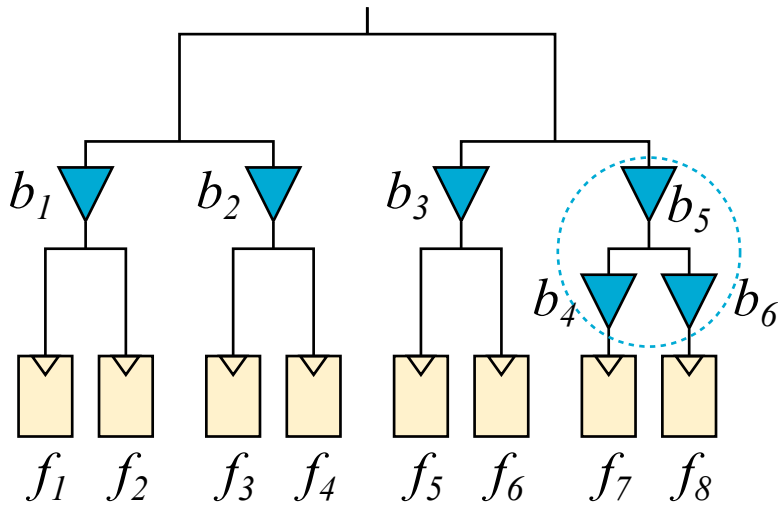
range. On the other hand, one of the previous research works has clarified the relationship between the area overhead and the number of PST buffers. The work by Kim *et al.* [16] provides dot curves in Fig. 2.2 that show the relation between the number of PST buffers and the total PST buffers area, produced by applying the PST buffer allocation algorithms proposed in the previous works [17, 18, 19]. The curves show that the total PST buffer area is highly correlated with the number of PST buffers allocated, implying that minimizing the number of PST buffers to be allocated consistently leads to the least total area of PST buffers.

A simplest way to find a minimal allocation of PST buffers is to enumerate all the combinations of assigning PST buffers to the nodes in the clock tree, run the Monte-Carlo simulation or SSTA to each of the allocations to estimate the timing yield, and select the allocation which uses the least number of PST buffers while meeting the yield constraint. However, this leads to an extraordinarily long computation time, which is unacceptable in practice. Thus, it is needed to employ a clever mechanism of exploring the space of PST buffer allocations. Some analyses on the three cases of PST buffer allocation illustrated in Fig. 2.3 may incite how an ideal exploration of PST buffer allocations could be looked like.

- *Converting an overlapping allocation into a non-overlapping one:* The clock tree in Fig. 2.3(a) has five PST buffers b_1 , b_2 , and b_3 , b_4 , and b_5 by which the clock arrival times to flip-flops f_1 and f_2 , f_3 and f_4 , and f_5 and f_6 are respectively tuned by b_1 , b_2 , and b_3 . However, the time to f_8 is tuned by b_5 and the time to f_7 is tuned by both f_4 and f_5 , which is then translated into an independent delay control to f_7 and f_8 . That is, the PST buffer allocation instance in Fig. 2.3(a) can be translated into that in Fig. 2.3(c), still using five PST buffers. This equivalence, in terms of the number of PST buffers allocated



(a) Overlapping.



(b) Redundant.

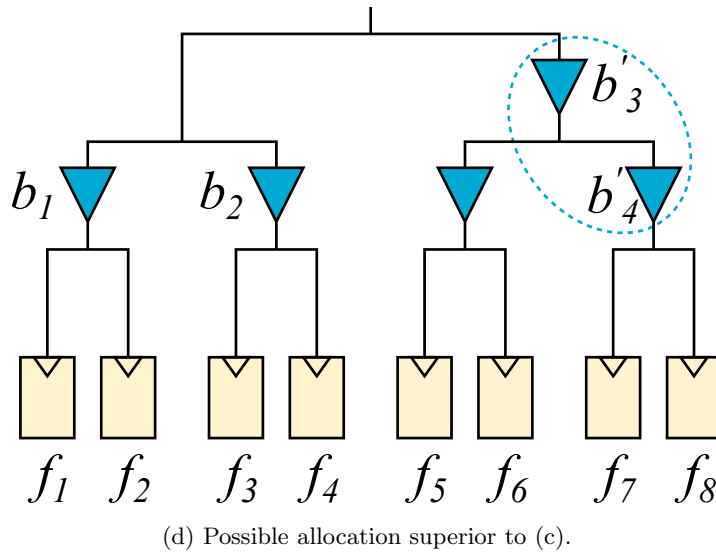
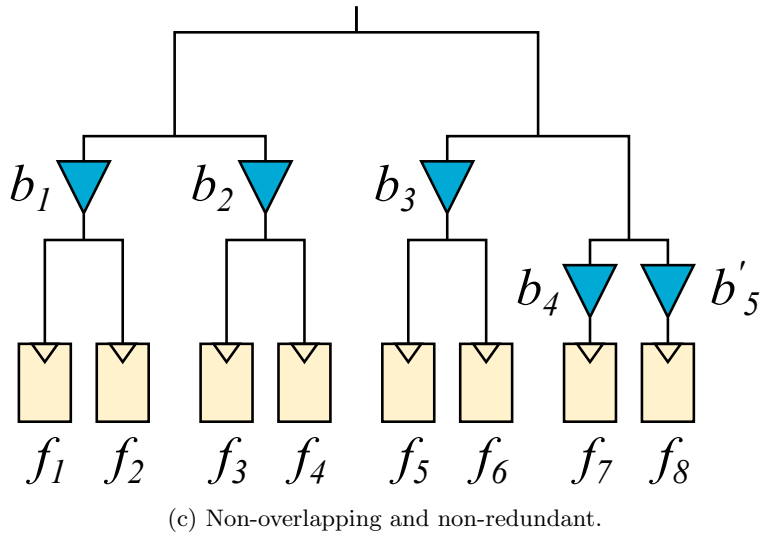


Figure 2.3: Classification of PST buffer allocations: (a) *overlapping* PST buffer allocation; (b) *redundant* PST buffer allocation; (c) *non-overlapping* and *non-redundant* PST buffer allocation of (a) and (b); (d) a possibly better overlapping PST allocation than (c).

and the delay control capability, of the two PST buffer allocation instances in Figs. 2.3(a) and (c) implies that we are able to safely exclude one of them from our consideration of allocation space exploration.

Definition 2.1: (PST depth) *PST depth, $\rho(p_i)$, of a signal path from clock source to flip-flop f_i of an PST allocation instance I in a clock tree is defined to the number of PST buffers on p_i and $\rho(I) = \max\{\rho(p_i)|p_i \in P(I)\}$ where $P(I)$ is the set of all clock signal paths in I .*

Definition 2.2: (Overlapping PST allocation) *An instance, I , of PST buffer allocation in a clock tree \mathcal{T} is said to be an overlapping PST allocation if $\rho(I) \geq 2$.*

For example, for the instance of PST buffer allocation in Fig. 2.3(a), all $\rho(p_i) = 1, i = 1, \dots, 8$ except $\rho(p_7) = 2$. Thus, $\rho(I) = 2$ and it is an overlapping PST allocation, whereas for the instance of PST buffer allocation in Fig. 2.3(c), its $\rho(I) = 1$ and it is not an overlapping PST allocation.

- *Converting a redundant allocation into a non-redundant one:* The clock tree in Fig. 2.3(b) has six PST buffers b_1, \dots, b_6 , in which b_5 controls the clock arrival times to both of f_7 and f_8 . Since the times to f_7 and f_8 also controlled independently by b_4 and b_6 , respectively, the allocation of b_5 is redundant. Thus, the PST buffer allocation in Fig. 2.3(b) can be translated to that in Fig. 2.3(c) by removing b_6 , suggesting that the PST buffer allocation instances with redundant PST buffer can be disregarded in the allocation space exploration.

Definition 2.3: (Redundant PST allocation) *An instance, I , of PST buffer allocation in a clock tree \mathcal{T} is said to be a redundant PST allocation if it is an overlapping PST allocation and there exists a PST buffer in the allocation such that its removal from I does not cause to degrade the yield.*

From the two examples of (overlapping and redundant) PST buffer allocation instances, it is obvious that the allocations that are meaningful to explore are

those that include non-overlapping and non-redundant ones like that shown in Fig. 2.3(c) whose PST depth $\rho(I) \leq 1$.

2.2.2 Problem Definition

The PST buffer allocation problem can be formally described as:

Problem 2.1 (PST buffer allocation) *For a given buffered clock tree \mathcal{T} , statistical timing distributions between sinks (i.e., flip-flops) on \mathcal{T} , and timing yield constraint Υ , minimize the number of PST buffers to be allocated to replace some of the normal buffers in \mathcal{T} so that the yield constraint Υ is met.*

Based on the observations, we narrow down the exploration space of Problem 2.1 by proposing to solve Problem 2.2:

Problem 2.2 (PST buffer allocation with $\rho(I) = 1$) *Solve Problem 2.1 with the additional constraint that the allocation instance, I , should satisfy PST depth $\rho(I) = 1$ ¹*

Note that since an optimal allocation for Problem 2.1 does not always mean an optimal allocation for Problem 2.2, solving Problem 2.2 may lead to a sub-optimal allocation for Problem 2.1. For example, the allocation in Fig. 2.3(d) could be an optimal solution for Problem 2.1 while the allocation in Fig. 2.3(b) could be an optimal solution for Problem 2.2. However, the merits of tackling Problem 2.2 is that (1) the allocation exploration space can be greatly reduced by effectively uncovering (many unnecessary) overlapping and redundant allocations; (2) the constraint of allocations with $\rho(I) = 1$ leads to a graph-theoretic formalization (e.g., Lemma 2.1) that facilitates the ease of statistical timing yield computation.

¹ $\rho(I) = 0$ refers to the allocation instance with only one PST buffer, allocating at the root of clock tree, thus ($\rho(I) = 1$). But, the PST buffer can be safely removed to be $\rho(I) = 0$.

2.2.3 Allocation Algorithm

Our proposed exploration of PST buffer allocations are performed in two directions: bottom-up and top-down, followed by a refinement. Fig. 2.4 illustrates how the bottom-up exploration is carried out: Initially, we allocate a distinct PST buffer at the front of each sink as shown in Fig. 2.4(a). Thus, timing yield for this initial allocation will be the highest. Suppose yield constraint $\Upsilon = 93\%$. Then, we compute the yield for each of the reallocations resulting from merging the PST buffers in the siblings into one. For example, in Fig. 2.4(a) merging PST buffers b_1 and b_2 into $b_{1,2}$, merging b_3 and b_4 into $b_{3,4}$, merging b_5 and b_6 into $b_{5,6}$, and merging b_7 and b_8 into $b_{7,8}$, produce yields of 98%, 92.5%, 97%, and 96%, respectively. In the second iteration, we select the one with the highest yield, which is $b_{1,2}$ among the merging candidates as indicated in Fig. 2.4(b). Then, the yields of the remaining merging candidates are recomputed. In the third iteration, according to the yields computed, $b_{5,6}$ is selected as shown in Fig. 2.4(c). We repeat this process as long as there is a merging candidate whose yield is no less than Υ . Figs. 2.4(d) show the mergings in the last iterations, in which all of merging into $b_{7,8}$ and merging into $b_{5,6,7,8}$ violate yield constraint Υ . Thus, our bottom-up exploration allocates 5 PST buffers as shown in Figs. 2.4(d). In the subsequent *refinement step*, we attempt to remove each PST buffer and compute the resulting yield. Then, we eliminate the PST buffer which has the highest yield while the yield constraint is still satisfied. We repeat this removal until the yield constraint does not met.

Our top-down exploration is exactly towards the reverse direction of the bottom-up, replicating a PST buffer in parent node and moving them down to its children nodes. Thus, the final PST allocation we choose is the one with the less number of PST buffers between the two allocation results produced by the

bottom-up and top-down explorations.

2.3 Fast Timing Yield Computation

2.3.1 Preliminaries

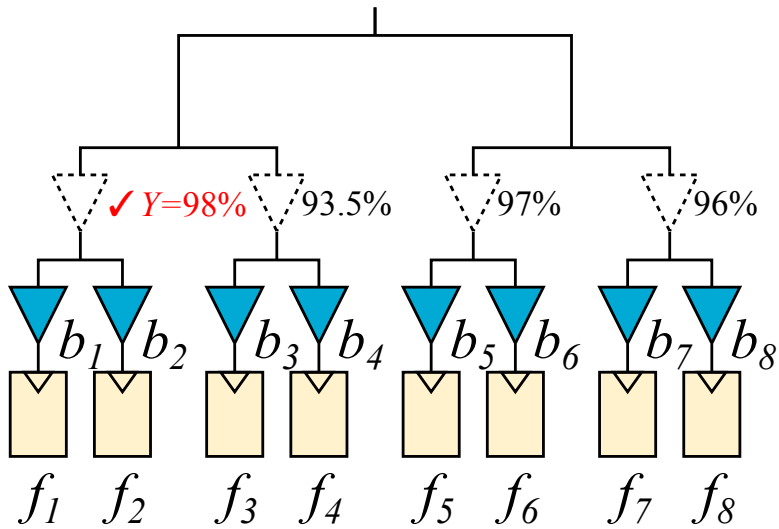
A portion of input circuit with PST buffers for timing analysis is depicted in Fig. 2.5 where the two PST buffers adjust the arrival times of clock signal to flip-flops in the post-silicon stage so that the (setup and hold) time constraints imposed in the circuits should be met. For simplicity, here, only the setup time constraint is reviewed. (The hold time constraint can be similarly explained.) If we assume that the clock signals arrive at f_i and f_j in Fig. 2.5 in the reference time of zero, only the delay values of PST buffers b_i and b_j determine the clock arrival times. To guarantee the setup time constraint at f_j from f_i , the clock period T of a circuit must satisfying the left inequality in the following expression:

$$x_i + D_{ij} \leq T + x_j - s_j \iff x_j - x_i \geq w_{ij} - T \quad (2.1)$$

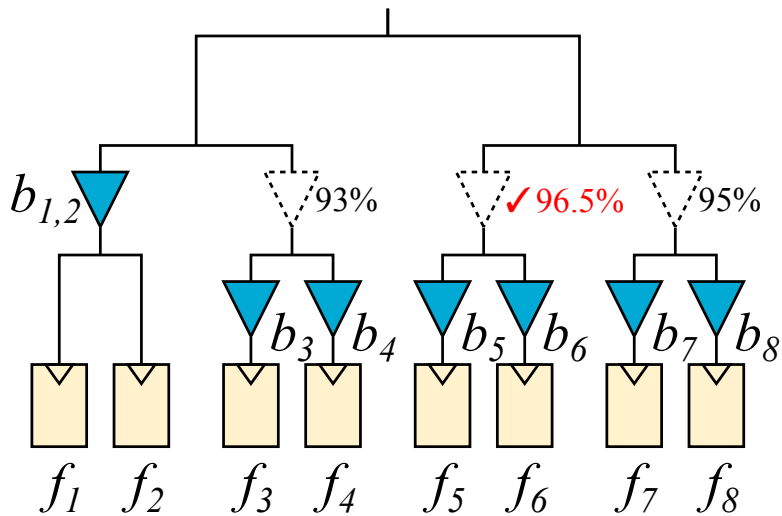
where x_i and x_j are the delay of PST buffers b_i and b_j , D_{ij} represents the maximum delay from f_i to f_j , T is the clock period, and s_j is the setup time of f_j . The inequality on the left side can be expressed into the one on the right side one by replacing $D_{ij} + s_j$ with w_{ij} . The delay range r_i of a PST buffer, b_i , will be determined by circuit designer and can be expressed as:

$$-r_i \leq x_i \leq r_i \quad (2.2)$$

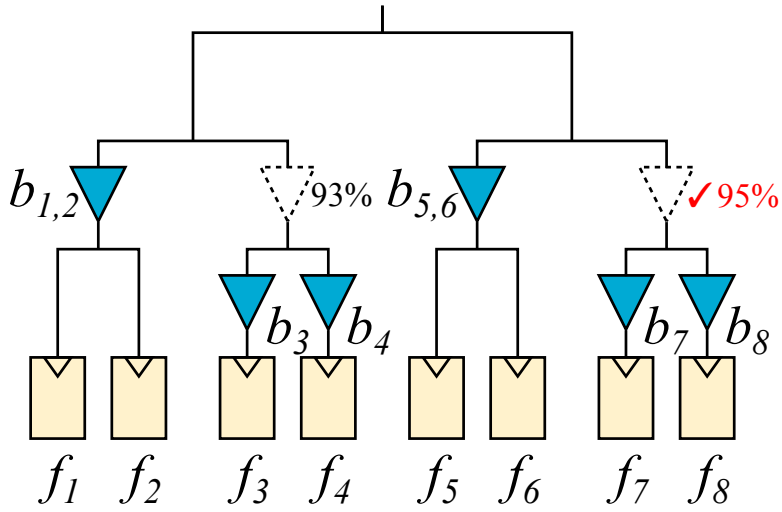
Since process variations affect D_{ij} and s_j values, these values are represented as statistical random variables. A circuit after manufacturing is considered working if there are values of x_i and x_j that satisfy *Eq.(2.1)* and *Eq.(2.2)*. Then, the delay of PST buffers b_i and b_j are adjusted to x_i and x_j . To compute the



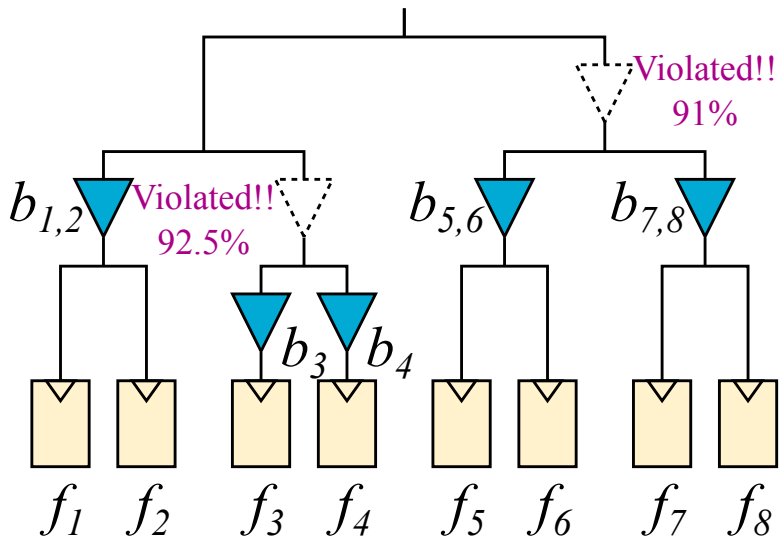
(a) First iteration: an initial PST buffer allocation followed by computing yield for each of mergings of sibling PST buffers.



(b) Second iteration: replacing PST buffers b_1 and b_2 in (a) with one PST buffer $b_{1,2}$ followed by recomputation of yields of remaining merging candidates.



(c) Third iteration: replacing b_5 and b_6 in (b) with $b_{5,6}$ followed by recomputation of yields of remaining merging candidates.



(d) Last iteration: all yields of the merging candidates is less than $\Upsilon = 93$.

Figure 2.4: An example illustrating the bottom-up exploration of PST buffer allocation. A refinement step after the last iteration follows, which iteratively checks if some of the PST buffers can be removed while meeting the yield constraint.

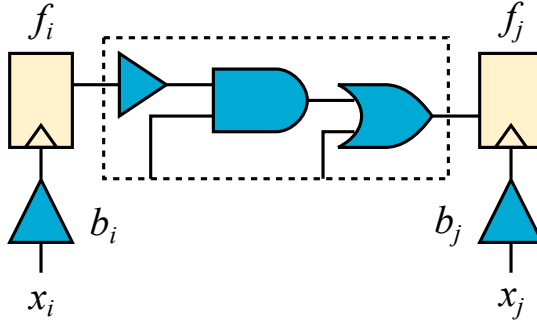
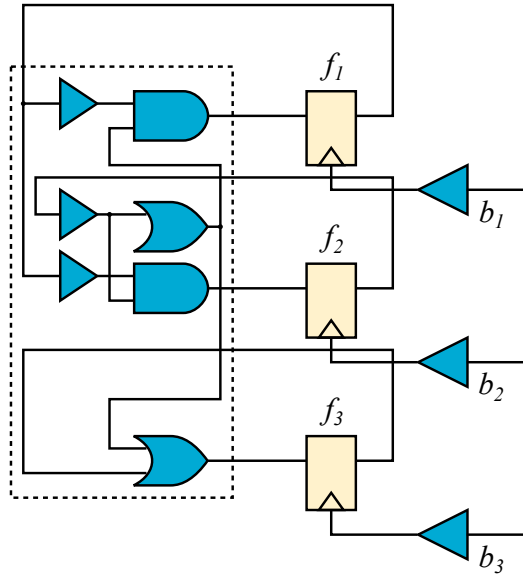


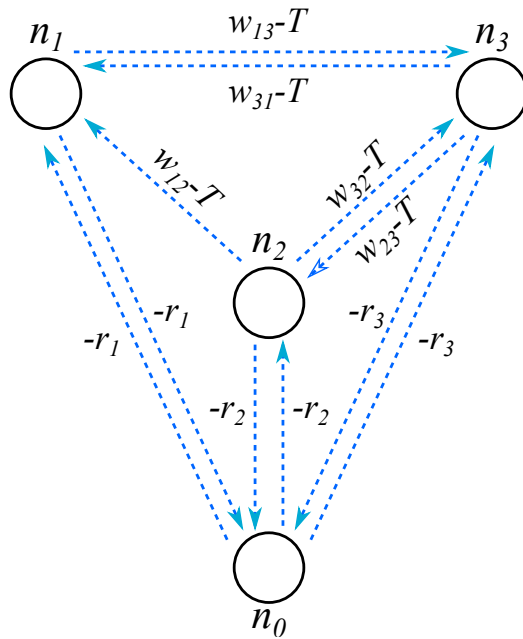
Figure 2.5: A portion of circuit with PST buffers to adjust the time to trigger flip-flops f_i and f_j .

timing yield of circuit which refers to the percentage of chips that work for the clock period T , the previous work [20] builds a form of constraint graph [21] first. For example, Fig. 2.6(b) shows the constraint graph corresponding to the timing relations among three flip-flops f_1 , f_2 , and f_3 in Fig. 2.6(a). Nodes n_i , $i = 1, 2, 3$, in the constraint graph corresponds to PST buffers b_i , $i = 1, 2, 3$, and the weight on the arc from n_i to n_j , $i, j = 1, 2, 3$ and $i \neq j$ is the right term in $x_j - x_i \geq w_{ij} - T$ in Eq.(2.1). Node n_0 is a dummy node to be used to incorporate the inequality in Eq.(2.2). The weights on the arcs from n_0 to n_i and from n_i to n_0 , $i = 1, 2, 3$, are the right terms in $x_i \geq -r_i$ and $-x_i \geq -r_i$ ($= x_i \leq r_i$), respectively. Then, the constraints that there are value assignments of x_i , $i = 1, 2, \dots$, that satisfy the Eq.(2.1) and Eq.(2.2) is equivalent to the constraint that all the loops in the corresponding constraint graph are nonpositive. (The detailed proof of the equivalence can be found in [21].) To solve Problem 2.2 efficiently, we extend the idea of representing the constraints in Eq.(2.1) and Eq.(2.2) into a constraint graph for *any* PST allocation instance I with $\rho(I) = 1$.

Lemma 2.1 *Any PST allocation instance I with $\rho(I) = 1$ can be represented into a constraint graph $G(I)$ such that the following holds: there is a delay*



(a)



(b)

Figure 2.6: Constraint graph representing the timing relation between flip-flops: (a) a portion of circuit instance, I , with the allocation of PST buffers b_1 , b_2 , and b_3 that are used to adjust clock arrival times to flip-flops f_1 , f_2 , and f_3 , respectively; (b) constraint graph $G(I)$.

assignment to the PST buffers in I that satisfies Eq.(2.1) and Eq.(2.2) if and only if all the loops in $G(I)$ are nonpositive. (We leave the proof out due to the space limitation.)

For convenience as does in [20], all arc weights in the constraint graph are replaced by the form of $w_{ij} - k_{ij}T$ where for arcs connected to the dummy node $w_{ij} = -r_i$ and $k_{ij} = 0$, and for others $k_{ij} = 1$. Finding a minimal clock period (T_{min}) that meets the inequalities of Eq.(2.1) and Eq.(2.2) is equivalent to (Step 1) finding the value of T_l for every loop $l \in L$ in the constraint graph such that

$$W_l = \sum_{i,j} (w_{ij} - k_{ij}T) \leq 0 \iff T \geq \sum_{i,j} w_{ij} / \sum_{i,j} k_{ij} = T_l \quad (2.3)$$

where L is the set of loops and W_l is the weight of loop $l \in L$ in the constraint graph and (Step 2) compute T_{min} :

$$T_{min} = \max_{l \in L} T_l. \quad (2.4)$$

Then, yield Y of the circuit can be derived as:

$$Y = \text{prob}\{T_{min} \leq T\}. \quad (2.5)$$

2.3.2 Incremental Yield Computation

The most time consuming parts in computing yield for an instance, I , of PST buffer allocation is (1) the derivation of the distribution of random variable T_l in Eq.(2.3) for every loop l in $G(I)$ and (2) the *max* operation on all the random variables T_{l_i} , $i = 1, \dots, |L|$ in Eq.(2.4) to produce the distribution of T_{min} , where L is the set of loops in $G(I)$. However, in the framework of our bottom-up and top-down explorations of PST buffer allocations with PST depth $\rho(\cdot) = 1$, the yield computation time can be greatly reduced. In the following, we

describe, using an illustrative example, our fast yield computation technique for the bottom-up allocation exploration case. (The yield computation strategy for the top-down case is almost identical to that of the bottom-up case.) Consider the allocation instance, I , in Fig. 2.7(a). Now, we want to find a merging which produces the highest value of Y in Eq.(2.5) among the merging candidates.

(1) Derivation of T_l in Eq.(2.3): Fig. 2.7(b) shows the constraint graph $G(I)$ of the allocation instance I in Fig. 2.7(a). There exists three merging candidates MG_1 , MG_2 , and MG_3 in $G(I)$ that correspond to merging b_1 and b_2 , merging b_3 and b_4 , and merging $b_{5,6}$ and $b_{7,8}$ in I , respectively. In addition, all the loops in $G(I)$ are listed in Fig. 2.7(c). Then, for trial of each merging candidate MG_j , we partition the set of loops L in $G(I)$ in two subsets L_{reuse} and L_{recomp} such that

- $L_{reuse} = \{l_i | l_i \in L \text{ and no arcs between the nodes in } MG_j \text{ are in } l_i\}$.
- $L_{recomp} = \{l_i | l_i \in L \text{ and at least one arc between the nodes in } MG_j \text{ are in } l_i\}$.

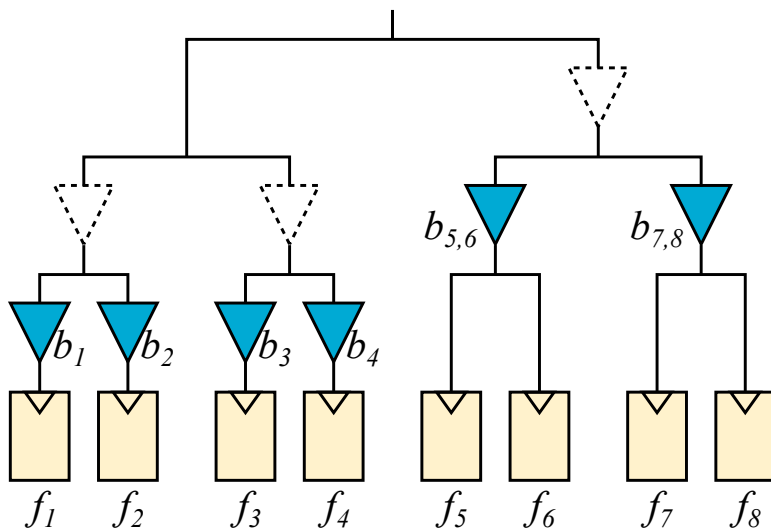
Since T_{l_i} in Eq.(2.3) of every loop $l_i \in L_{reuse}$ has already been computed in the previous iteration and trial of merging candidate MG_j does not change l_i , the value of T_{l_i} can be reused in computing T_{min} in Eq.(2.4) for the incremental reallocation by MG_j . Thus, *only for each $l_i \in L - L_{reuse}(= L_{recomp})$, T_{l_i} needs to be recomputed for the incremental reallocation by MG_j .* Table $R[-, -]$ in Fig. 2.7(d), which we call **Reuse table**, contains the reuse-ability information. For example, $R[3, 1] = 0$ means that T_{l_3} of loop l_3 can be reused for the incremental reallocation by MG_1 , while $R[4, 2] = 1$ means that T_{l_4} of loop l_4 should be recomputed for the incremental reallocation by MG_2 .

(2) Derivation of T_{min} in Eq.(2.4): Since T_{min} derived from random variables $T_{l_1}, \dots, T_{l_{|L|}}$ entails repeated application of *max* operations, if we can keep some

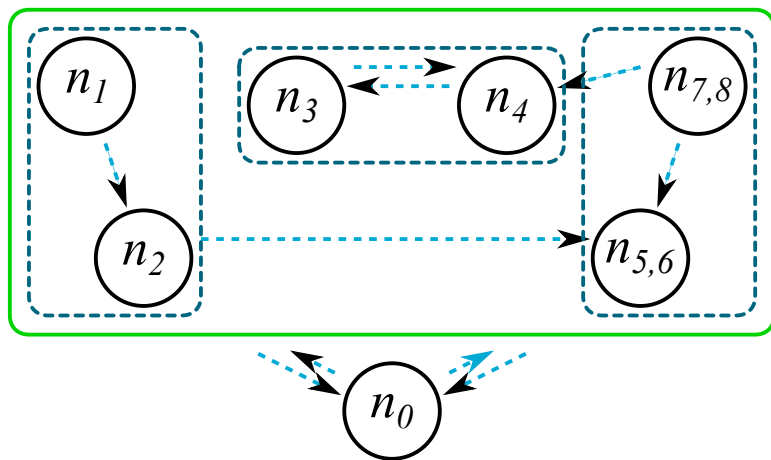
of the results by *max* operations, they can be reused in the trials of merging candidates. To maximize the reuse, we exploit the information in the reuse table R . For example, in the reuse table in Fig. 2.8(a), *since loops l_5 and l_6 has never been changed for any of the trial of merging candidates MG_1 , MG_2 , and MG_3 , the result of $\max\{T_{l_5}, T_{l_6}\}$ is stored to be reused for computing T_{min} in Eq.(2.4) of the incremental reallocation by any merging candidate.* Once $\max\{T_{l_5}, T_{l_6}\}$ is computed and stored, the reuse table is updated by merging rows of l_5 , and l_6 into a row (l_5, l_6) and setting $R[(l_5, l_6), j] = OR(R[l_5, MG_j], R[l_6, MG_j])$, $j = 1, 2, 3$, as shown in Fig. 2.8(b). In the next iteration, select the two loops whose bitwise-OR operation to the reuse table leads to the smallest number of 1's. *The less the number of 1's by the bitwise-OR of two rows l_i and l_j in R is, the more it is likely that trials of merging candidates can reuse the result of $T_{(l_i, l_j)}$ ($= \max\{T_{l_i}, T_{l_j}\}$).* As a results, loops l_1 and l_2 , loops l_3 and l_4 , and then loops l_7 and l_8 are selected and performed $\max\{T_{l_i}, T_{l_j}\}$ operations as shown in the first row of table in Fig. 2.8(b). Then, in the next iterations, bitwise-OR operations to the pairs of rows in the reuse table produce the sequence of max operations: $\max\{T_{(l_3, l_4)}, T_{(l_5, l_6)}\}$ in Fig. 2.8(c), $\max\{T_{(l_1, l_2)}, T_{(l_3, l_4, l_5, l_6)}\}$ in Fig. 2.8(d), and $\max\{T_{(l_1, l_2, l_3, l_4, l_5, l_6)}, T_{(l_7, l_8)}\}$ in Fig. 2.8(e).

2.4 Experimental Result

Our proposed PST buffer allocation algorithm called PST-alloc is implemented in Python3 and C++ and tested using 2.67 GHz linux machine. We used IS-CAS89 [22] and ITC99 [23] benchmark circuits for experiments. The devices in the benchmark circuits are mapped to Nangate 45nm technology library [24]. We use SSTA engine developed in [5] and supplemented in [20]. We compare the results produced by PST-alloc with that by the previous work in [12] as well



(a) An instance, I , of PST buffer allocation.



(b) Constraint graph $G(I)$ and merging candidates in $G(I)$.

Loop₁	$n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow n_0$
Loop₂	$n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow n_{5,6} \rightarrow n_0$
Loop₃	$n_0 \rightarrow n_2 \rightarrow n_{5,6} \rightarrow n_0$
Loop₄	$n_0 \rightarrow n_3 \rightarrow n_4 \rightarrow n_0$
Loop₅	$n_0 \rightarrow n_4 \rightarrow n_3 \rightarrow n_0$
Loop₆	$n_0 \rightarrow n_{7,8} \rightarrow n_4 \rightarrow n_0$
Loop₇	$n_0 \rightarrow n_{7,8} \rightarrow n_4 \rightarrow n_3 \rightarrow n_0$
Loop₈	$n_0 \rightarrow n_{7,8} \rightarrow n_{5,6} \rightarrow n_0$

(c) Loops in $G(I)$.

	MG_1	MG_2	MG_3
Loop₁	1	0	0
Loop₂	1	0	0
Loop₃	0	1	0
Loop₄	0	1	0
Loop₅	0	0	0
Loop₆	0	0	0
Loop₇	0	1	0
Loop₈	0	0	1

(d) Reuse table R for $G(I)$.

Figure 2.7: Enumeration of all loops in an instance of PST buffer allocation and generation of reuse relation between loops and merging candidates: (a) an instance, I , of PST buffer allocation in a clock tree. Six PST buffers $b_1, \dots, b_4, b_{5,6}, b_{7,8}$ are allocated; (b) constraint graph $G(I)$ and PST buffer merging candidates MG_1 , MG_2 , and MG_3 in $G(I)$; (c) list L of loops in $G(I)$; (d) reuse table $R[-, -]$ to represent the T_l reuse relation between the loops and merging candidates.

as Monte-Carlo simulation in terms of run time the number of PST buffers, and the accuracy of yield computation.

- *Checking the accuracy of yield computation:* To assess the accuracy of our SSTA engine, we compared our SSTA results to compute yields during the exploration of PST allocations in PST-alloc with the results by Monte-Carlo simulations with 10,000 iterations. Table 2.1 summarizes the error rate between the values by our SSTA and by Monte-Carlo simulation for two circuits s382 and s5378 in which $MG[i]$, $i = 1, \dots, 10$, in the first column of the table represents the allocation instance produced at the i -th iteration transforming from the initial PST allocation instance INIT. In summary, the error rate of yield is below 0.2% in the mean value and 2% in the standard deviation. It also should be noted that the speed of our SSTA is nearly zero while that of Monte-Carlo simulation is over 10,000 times slower.

- *Comparing PST-alloc with previous work:* We compare our results with that produced by the PST buffer allocation algorithm, which we called Tsai's, in [12]. Tsai's repeatedly find flip-flops with the smallest slack and place PST buffers at the front of the flip-flops. Although the slacks on flip-flops are computed by SSTA, the PST buffer allocations are set by a certain threshold given by designer. Furthermore, the yield computation is done by the slow Monte-Carlo simulation. (For accurate yield computation, all yield results are replaced by the result produced by [20].) Table 2.2 summarizes the results. The comparison shows that PST-alloc (Bottom-up) produces the higher yield than that by Tsai's while the same number of PST buffers are allocated. Similarly, PST-alloc (Top-down) increases chip yield. Due to the different directions of exploration and converging points, results of PST-alloc (Bottom-up) and PST-alloc (Top-down) for circuits s382, s15850, and s38584 produce different results. However, the number of PST buffers are different, the yield results of both approaches are

<i>loop</i>	<i>MG</i> ₁	<i>MG</i> ₂	<i>MG</i> ₃
<i>l</i> ₁	1	0	0
<i>l</i> ₂	1	0	0
<i>l</i> ₃	0	1	0
<i>l</i> ₄	0	1	0
<i>l</i> ₅	0	0	0
<i>l</i> ₆	0	0	0
<i>l</i> ₇	0	1	0
<i>l</i> ₈	0	0	1

(a) Initial reuse table *R*

<i>loop</i>	<i>MG</i> ₁	<i>MG</i> ₂	<i>MG</i> ₃
(<i>l</i> ₁ , <i>l</i> ₂)	1	0	0
(<i>l</i> ₃ , <i>l</i> ₄)	0	1	0
(<i>l</i> ₅ , <i>l</i> ₆)	0	0	0
(<i>l</i> ₇ , <i>l</i> ₈)	0	1	1

(b) *max* for *l*₅ and *l*₆, *max* for *l*₁ and *l*₂, *max* for *l*₃, and *l*₄, *max* for *l*₇ and *l*₈

<i>loop</i>	<i>MG</i> ₁	<i>MG</i> ₂	<i>MG</i> ₃
(<i>l</i> ₁ , <i>l</i> ₂)	1	0	0
((<i>l</i> ₃ , <i>l</i> ₄), (<i>l</i> ₅ , <i>l</i> ₆))	0	1	0
(<i>l</i> ₇ , <i>l</i> ₈)	0	1	1

(c) *max* for (*l*₃, *l*₄) and (*l*₅, *l*₆)

<i>loop</i>	<i>MG</i> ₁	<i>MG</i> ₂	<i>MG</i> ₃
((<i>l</i> ₁ , <i>l</i> ₂), (<i>l</i> ₃ , <i>l</i> ₄), (<i>l</i> ₅ , <i>l</i> ₆))	1	1	0
(<i>l</i> ₇ , <i>l</i> ₈)	0	1	1

(d) *max* for (*l*₁, *l*₂) and (*l*₃, \dots , *l*₆)

<i>loop</i>	<i>MG</i> ₁	<i>MG</i> ₂	<i>MG</i> ₃
((<i>l</i> ₁ , <i>l</i> ₂), (<i>l</i> ₃ , <i>l</i> ₄), (<i>l</i> ₅ , <i>l</i> ₆), (<i>l</i> ₇ , <i>l</i> ₈))	1	1	0

(e) *max* for (*l*₁, \dots , *l*₆) and (*l*₇, *l*₈)

Figure 2.8: Generation of a sequence for *max* operations. Reusable partial max results for trials of merging candidates are: (a) initial result table *R*; (b) *max*{*T*_{*l*₅}, *T*_{*l*₆}} followed by *max*{*T*_{*l*₁}, *T*_{*l*₂}}, *max*{*T*_{*l*₃}, *T*_{*l*₄}}, and *max*{*T*_{*l*₇}, *T*_{*l*₈}}; (c) *max*{*T*_(*l*₃, *l*₄), *T*_{*l*₅}, *T*_{*l*₆}} (= *T*_(*l*₃, *l*₄, *l*₅, *l*₆)); (d) *max*{*T*_(*l*₁, *l*₂), *T*_(*l*₃, *l*₄, *l*₅, *l*₆)} (= *T*_(*l*₁, \dots , *l*₆)); (e) *max*{*T*_(*l*₁, \dots , *l*₆), *T*_(*l*₇, *l*₈)} (= *T*_{*l*₁, \dots , *l*₈}).

Table 2.1: Comparison the accuracy of yield (Y) computation by Monte-Carlo simulation and PST-alloc. ($Y(\text{PST-alloc}) - Y(\text{MC})$)

Alloc.	s382		s5378	
	μ	σ	μ	σ
INIT	0.052%	0.669%	0.051%	1.163%
MG[1]	0.029%	0.151%	0.130%	0.375%
MG[2]	0.044%	0.575%	0.008%	0.958%
MG[3]	0.117%	0.214%	0.080%	0.297%
MG[4]	0.049%	0.085%	0.059%	0.555%
MG[5]	0.152%	0.399%	0.095%	0.394%
MG[6]	0.063%	0.609%	0.065%	0.391%
MG[7]	0.012%	1.712%	0.035%	0.231%
MG[8]	0.031%	1.638%	0.001%	0.451%
MG[9]	0.015%	0.205%	0.053%	0.339%
MG[10]	0.059%	0.158%	0.086%	1.200%

the same. Due to the timing problems which are solved by PST buffers in PST-alloc (Bottom-up) dominate that by additional PST buffers in PST-alloc (Top-down), many PST buffers in design of PST-alloc (Top-down) are useless. Table 2.3 shows the number of PST buffers when [12] produces the similar level of chip yield to that of PST-alloc (Top-down). PST-alloc uses 26% less number of PST buffers than that by the previous works while maintaining the similar yield constraint. Note that the run time of PST-alloc (Bottom-up) is quite large, but much faster than that of Tsai's.

- *Comparing PST-alloc with exhaustive enumeration:* Figs. 2.9(a) and (b) show the comparison of the size of design space explored by the exhaustive search

Table 2.2: Comparison of results by [12] and our PST-alloc.

Circuit	# FF	T	Tsai's [12]	
		(ps)	Yield	# PST
s386	20	503.193	0.97	4
s382	30	574.799	0.927	6
s1423	74	1354.41	0.942	9
s5378	247	798.285	0.09	6
s13207	482	836.713	0.905	6
s15850	235	860.383	0.452	19
s38584	1458	1396.93	0.829	33
Circuit	PST-alloc (Bottom Up)			
	# iter.	Time (s)	Yield	# PSTs
s386	17	1.85	0.97	4
s382	26	14	0.995	5
s1423	103	1354.41	0.886	9
s5378	242	6128.26	0.903	6
s13207	477	1044.76	0.977	6
s15850	229	53.14	0.918	7
s38584	1444	6137.75	0.871	15
Circuit	PST-alloc (Top Down)			
	# iter.	Time (s)	Yield	# PSTs
s386	3	0.01	0.97	4
s382	5	0.01	0.995	6
s1423	8	3.66	0.886	9
s5378	5	9.36	0.903	6
s13207	5	5.81	0.977	6
s15850	18	1.73	0.918	19
s38584	32	1116.46	0.871	33

Table 2.3: Comparison of the number of PST buffers by [12] and our PST-alloc (Top-down).

Circuit	T	Tsai's [12]		PST-alloc	
	(ps)	Yield	# PST	Yield	# PSTs
s386	503.193	0.97	4	0.970	4
s382	574.799	0.994	7	0.995	5
s1423	1354.41	0.942	9	0.886	9
s5378	798.285	0.902	40	0.903	6
s13207	836.713	0.905	6	0.977	6
s15850	860.383	0.917	31	0.918	7
s38584	1396.93	0.829	14	0.871	15

and our PST-alloc and the numbers of PST buffers allocated. We can see that the design space explored by PST-alloc is much smaller than the exhaustive one, but the numbers of PST buffers allocated by PST-alloc is closer to the optimal ones.

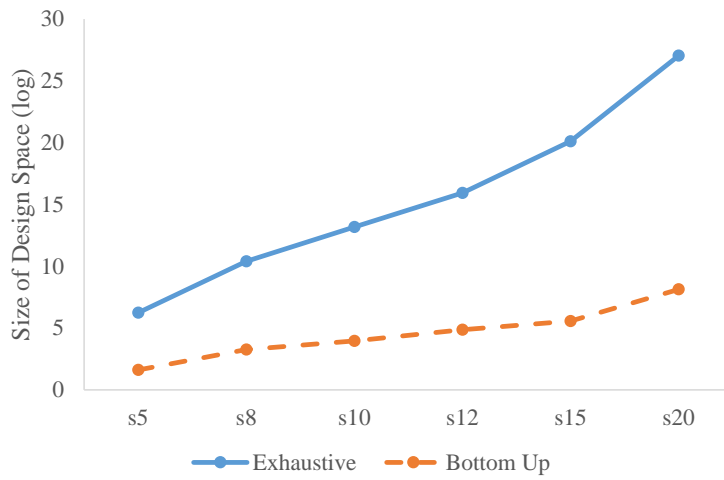
2.5 PST Buffer Configuration Techniques

This chapter proposed a PST buffer allocation algorithm to increase yield of the circuit. However, the timing problem of the circuit is able to be solved when the delay values of all PST buffers are configured after manufacturing process. Previous works have been tried to solve the problem by using measurements produced at post-silicon time. In this section, techniques which defined the PST buffer delay configuration problem and proposed an algorithm to solve it. [25] introduced Genetic Algorithm-based clock adjustment method to increase chip yield. The work assumed that all flip-flops have their own PST buffers. Other

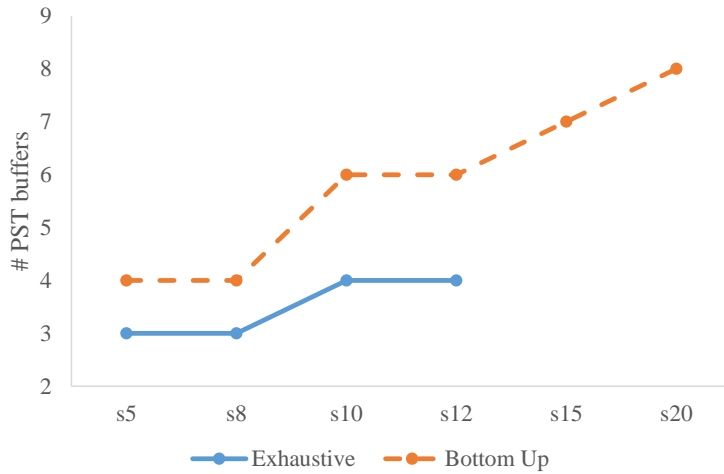
researchers proposed a technique to reduce the effect of process variation on the same environment [26]. The work found an unique tuning setting for each die to maximize performance. The problem of above two works is that the algorithms find solution for a single die, so they are quite expensive. In test process, a large number of chips should be tested and the number of applied tests is also big. AutoRex [27] pointed out scalability issues of the previous works and proposed the algorithm which finds a tuning setting based on information of a batch of chips. The algorithm used Satisfiability Modulo Theory (SMT) solvers to process large data and generate results. In addition, Lak and Nicolici [28] extended the algorithm of Tadesse *et al.* to consider hold time violation.

2.6 Summary

In this chapter, we proposed a comprehensive graph-based algorithm for PST buffer allocation to overcome the two critical limitations of the prior works, which are (1) the use of slow Monte-Carlo simulation in computing the chip yield and (2) a limited design space exploration of PST buffer allocation. Precisely, we develop *a graph-based chip yield computation* technique which is able to compute yields *very efficiently and accurately* for *incremental PST buffer allocations*, and based on the technique, we developed *a systematic (bottom-up and top-down) PST buffer allocation algorithm* that is able to *fully explore the design space* of allocation to minimize the number of PST buffers. The speed of proposed algorithm is much faster than the prior algorithms and it significantly improves yield while the same number of PST buffers are allocated.



(a) The design space explored by PST-alloc is much smaller than the exhaustive one



(b) The numbers of PST buffers allocated by PST-alloc is closer to the optimal ones

Figure 2.9: Comparing the exploration space and the number PST buffers allocated by exhaustive enumeration of all instances and our PST-alloc.

Chapter 3

POST-SILICON TUNING BASED ON FLEXIBLE FLIP-FLOP TIMING

This chapter presents a concept of flexible flip-flop timing model and clock skew optimization technique which integrates flexible flip-flop timing model in STA.

3.1 Introduction

Definitions of *setup skew*, *hold skew*, and *clk-to-Q delay* are shown in Fig. 3.1. *Setup skew* (t_S) is the time interval ending at the clock's active edge during which the input data is stable and *hold skew* (t_H) is the time interval starting at the clock's active edge during which the input data is stable. *Clk-to-Q delay* (t_Q) is the time lapse between the clock's active edge of flip-flop and the time to register the input data to the output of the flip-flop. See Fig. 3.1 for the illustration.¹

¹Note that the value of clk-to-Q delay in a flip-flop is determined by the setup and hold skews on that flip-flop. The delay in turn affects the values of setup and hold skews of the

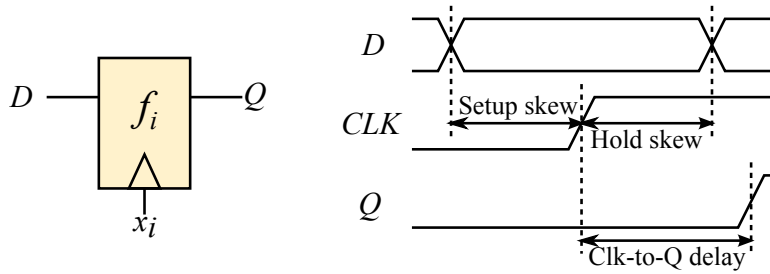


Figure 3.1: Description of setup skew t_S , hold skew t_H , and clk-to-Q delay t_Q for a flip-flop with input D and output Q [29]. The values of t_S and t_H of flip-flop f_i determine the value of t_Q of the flip-flop, which in turn affects the values of t_S and t_H of a flip-flop that will be driven by flip-flop f_i .

Static Timing Analysis (STA) checks setup and hold time constraints for all flip-flops. *Setup time* is the minimum amount of time required before the clock's active edge by which the input data must be stable for it to be latched correctly, whereas *hold time* is the minimum amount of time required after the clock's active edge during which the data must be stable. If setup and hold skews of a flip-flop are larger than setup and hold times, the flip-flop is able to pass a correct data. While traditional STA have regarded setup time, hold time, and clk-to-Q delay of flip-flops as static values, recent works pointed out the interdependent relations among them.

The interdependent relationship is shown in Fig. 3.2. Clk-to-Q delay is changed as setup skew or hold skew is changed. HSPICE simulation results for the changes of the values of setup skew, hold skew, and clk-to-Q delay are shown in Figs. 3.3 and 3.4. ² *Setup curve* (the right-upper one) in Fig. 3.3 shows how the clk-to-Q delay decreases as the setup skew increases while the hold skew

flip-flops that are driven by the output of the flip-flop.

²The inter-dependent relation between the values of setup skew, hold skew, and clk-to-Q delay produced by HSPICE simulation with 45 nm Nangate Open Cell Library [24]. The short (long) setup or hold skews cause long (short) clk-to-Q delay and there are multiple pairs of setup and hold skews that can generate the same value of clk-to-Q delay.

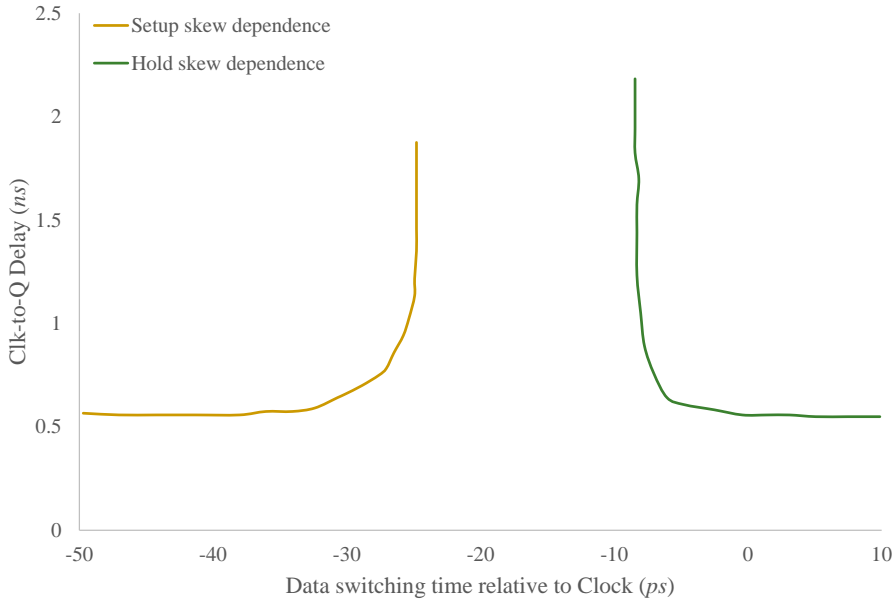


Figure 3.2: Relation among setup skew, hold skew, and clk-to-Q delay [29].

$= -8ps$. It is seen that the shortest clk-to-Q delay to be achievable is $89.2ps$ at the expense of the setup skew of around $110ps$. If the setup skew is decreased to $65.9ps$, the clk-to-Q delay is expected to increase by 10%. Similarly, *hold curve* (the left-lower one) in Fig. 3.3 shows how the clk-to-Q delay decreases as the hold skew increases while the setup skew = $37ps$. The shortest clk-to-Q delay is $67.1ps$ with hold skew of around $20ps$. Further, if the hold skew decreases from $20ps$ to $8.7ps$, the clk-to-Q delay is expected to decrease by 10%. The two curves in Fig. 3.3 clearly show the trade-off between the setup/hold skews and clk-to-Q delay. On the other side, the three curves in Fig. 3.4 shows the trade-off between the setup and hold skews when the clk-to-Q delay is set to 73, 77, and $92ps$, in which the numbers at the end points of the curves indicate the least values of setup and hold skews achievable for the corresponding clk-to-Q delays. We can see that setup or hold time to generate a particular value of

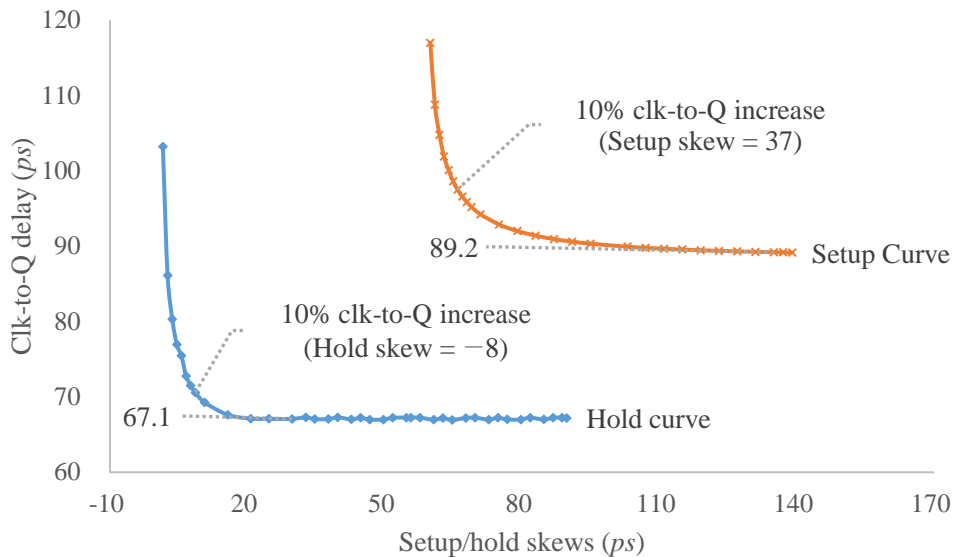


Figure 3.3: *Setup curve*: the change of clk-to-Q delay t_Q as the setup skew t_S changes, in which the hold skew t_H is fixed to $-8ps$. *Hold curve*: the change of clk-to-Q delay t_Q as the hold skew t_H changes, in which the setup skew t_S is fixed to $37ps$.

clk-to-Q delay is depended on the value of the opposite skews.³ In short, the curves in Figs. 3.3 and 3.4 imply that STAs should take into account the inter-dependent relation between setup skew, hold skew, and clk-to-Q delay in their flip-flop timing model, if we really want to avoid unnecessary late-stage ECO (engineering change order) or high cost design change before timing sign-off due to inaccurate analysis of the timing behavior of circuits.

Several works analyzed the inter-dependent relation of setup time, hold time and, clk-to-Q delay, and used the time relation in their STAs [30, 31, 32, 33, 34, 35, 36, 37]. Rao and Howick [30] firstly clarified the fact that the setup and hold times have inter-dependent relation. Then, Srivastava and Roychowdhury [31, 32] developed a methodology of fast characterization of the time relation.

³For generating $70ps$ of clk-to-Q delay, the setup time is set to $8ps$ when the hold skew is $-8ps$.

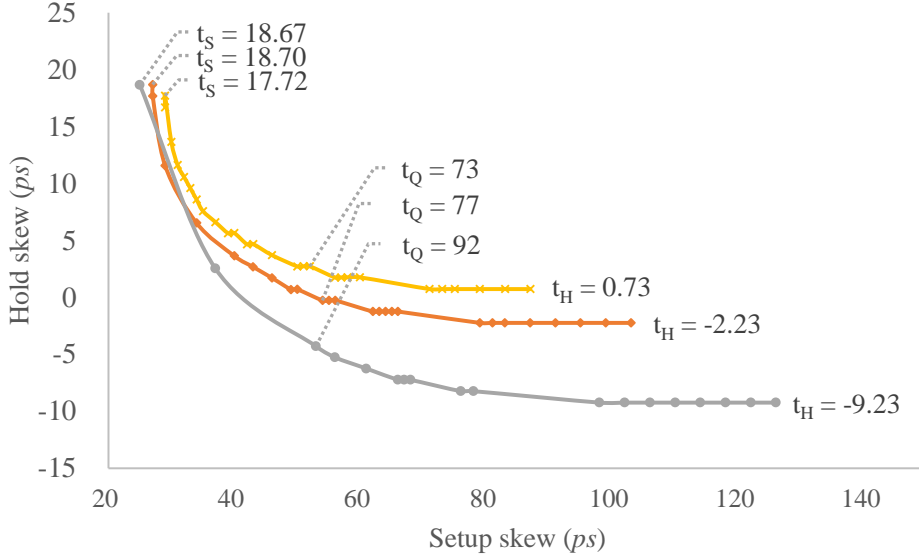


Figure 3.4: Three curves showing the trade-off between the setup skew t_S and hold skew t_H for clk-to-Q delay $t_Q = 73, 77,$ and $92ps$. As the clk-to-Q delay is shortened, the flexibility of trading the setup skew with hold skew is reduced.

They employed Euler-Newton curve tracing method to characterize the inter-dependent setup and hold time contours. Salman et al. [33, 35] enhanced the accuracy of STA by reflecting the inter-dependent relation of setup and hold times, and clk-to-Q delay all together, which is commonly called ‘flexible’ timing model in the literature. In their STA, they selected several representative pairs of setup and hold times and stored them in cell library. Then, accurate timing analysis was performed by alternating the mapping of cells of different times. Salman and Friedman [34] attempted to utilize the inter-dependent time relation in tolerating delay variations, so that delay uncertainty be reduced. Hatami et al. [36] showed by Monte Carlo simulation that using the flexible timing model can significantly improve the time accuracy in statistical STA. Later, Chen and Schlichtmann [37] expressed clk-to-Q delay as an analytic function of setup and hold skews, and proposed an iterative timing analysis

method, in which clk-to-Q delay of a flip-flop is iteratively computed, starting from a valid clk-to-Q delay value and reports a time violation on a flip-flop if the clk-to-Q delay value of the flip-flop does not converge. One major limitation of the method is that if the initial delay value is improperly set, it may lead to a false negative result. Recently, Kahng and Lee [38] proposed a sequential linear programming method to further improve the accuracy of STA over that in [37]. However, still the linearization of flexible timing model in their method causes non-trivial fitting errors.

The key feature of this chapter is that unlike all the previous works, in which they have addressed the problem of improving the accuracy of STA by employing the flexible timing model into STA, *our work integrates the task of clock skew optimization into the flexible timing model in STA*. Thus, for circuits in which existing STAs with flexible timing model report a time violation or a very tight time slack, our proposed technique can be applied to look for a possible clock schedule (objective-1) to resolve the time violation or (objective-2) to produce a more relaxed time margin. To achieve objectives-1 and 2, a light ECO such as detouring (or snaking) clock wire or resizing clock buffers to adjust clock skew is required (e.g., [39]). However, the reconfigurable metals and spare cells available for ECO are limited, it is important to use the amount of tuning resources minimally. From this viewpoint, our work is very helpful since it constrains the range of clock skew tuning so that the tuning should be as minimal as possible. To sum up, for the situation where there seems no hope to resolve timing violation in the postsilicon stage whatever we apply any conventional timing optimization technique, the timing optimization based on flexible flip-flop timing model could be a last resort.

3.2 Preliminary and Definitions

3.2.1 Flexible Flip-Flop Timing Model

One noticeable expression which describes the relation of setup and hold skews, and clk-to-Q delay in Figs. 3.3 and 3.4 has been given by [37]:

$$t_Q = a_0 + \frac{a_1}{t_S - t_{S_0}} + \frac{a_2}{t_H - t_{H_0}} \quad (3.1)$$

where t_Q , t_S , and t_H are clk-to-Q delay, setup skew, and hold skew, respectively. t_{S_0} and t_{H_0} are the lower bounds of t_S and t_H , respectively.

We use the expression in Eq.(3.1) as a flexible flip-flop timing model in our clock skew optimization technique.⁴ Note that the effect of load and data/clock slew on clk-to-Q can also be included in the constants a_0 , a_1 , and a_2 in the expression.

3.2.2 Definitions

For two flip-flops f_i and f_j , let D_{max} and D_{min} denote the maximum and minimum delays of the combinational logic from f_i to f_j , respectively, and T_{clk} denote the clock period. Further, let $t_Q(i)$, $t_S(i)$, and $t_H(i)$ denote clk-to-Q delay, setup skew, and hold skew at f_i , respectively. Let x_i and x_j be the clock signal arrival times to f_i and f_j , respectively.

- *Setup skew* ($t_S(i, j)$) : it is the time interval ending at the clock's active edge of flip-flop f_j during which the input data of f_j coming from flip-flop f_i is stable and can be expressed as:

$$t_S(i, j) = (x_j - x_i + T_{clk}) - (t_Q(i) + D_{max}). \quad (3.2)$$

⁴Our proposed clock skew scheduling technique does not depend on particular forms of flexible timing model and can accept any timing model.

- *Setup slack* ($slk_S(i, j)$) : it is the extra setup skew available beyond the minimally required setup skew t_{S_0} at flip-flop f_j with respect to the data input coming from flip-flop f_i , i.e.,

$$slk_S(i, j) = t_S(i, j) - t_{S_0}. \quad (3.3)$$

Thus, $slk_S(-, j) \geq 0$ if and only if no setup time violation occurs at f_j .

- *Hold skew* ($t_H(i, j)$) : it is the time interval starting at the clock's active edge of flip-flop f_j during which its input data coming from flip-flop f_i is stable and can be expressed as:

$$t_H(i, j) = (x_i - x_j) + t_Q(i) + D_{min}. \quad (3.4)$$

- *Hold slack* ($slk_H(i, j)$) : it is the extra hold skew beyond the minimally required hold skew t_{H_0} at flip-flop f_j with respect to the data input coming flip-flop f_i , i.e.,

$$slk_H(i, j) = t_H(i, j) - t_{H_0}. \quad (3.5)$$

No hold time violation occurs if and only if $slk_H(-, j) \geq 0$.

- *Worst slack* : it is the minimum value of the setup and hold slacks for all flip-flops in the circuit:

$$slk_{worst} = \min\{slk_S(i, j), slk_H(i, j), \forall i, j\}. \quad (3.6)$$

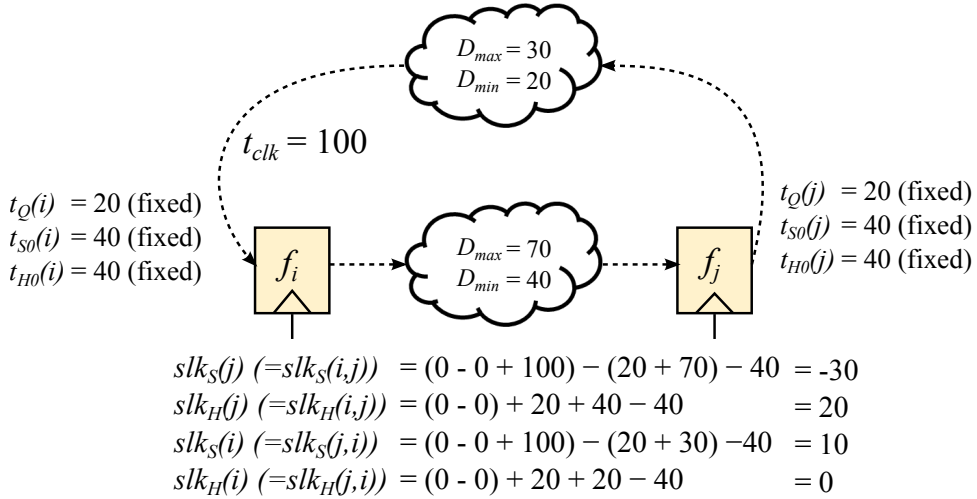
For brevity, if flip-flop f_i is the only one whose output directly drives the input of flip-flop f_j , we use short notations $t_S(j)$, $t_H(j)$, $slk_S(j)$, and $slk_H(j)$ to represent $t_S(i, j)$, $t_H(i, j)$, $slk_S(i, j)$, and $slk_H(i, j)$, respectively.

Note that the main role of STAs is to check whether the input circuit contains no setup and hold time violations or not. In reality, it is not a simple

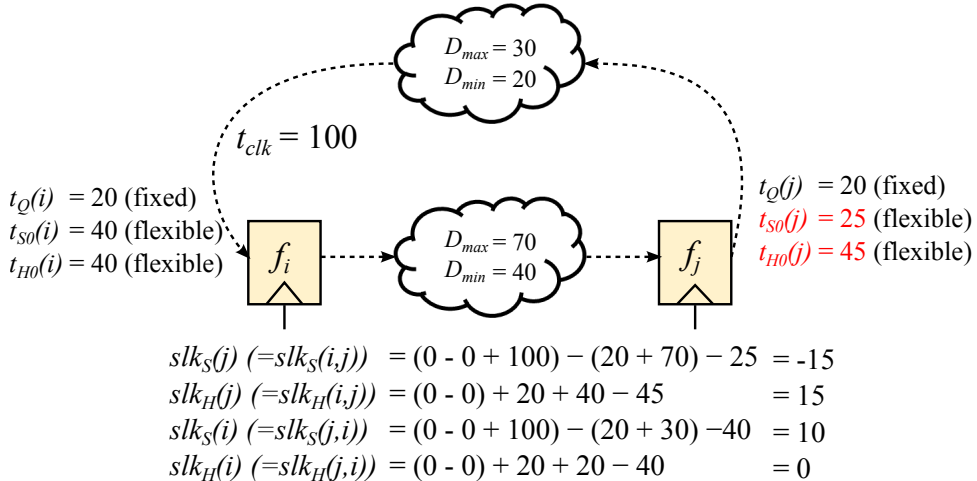
matter to accurately compute the setup and hold slacks of all flip-flops in the circuit. The conventional STAs have been progressed in a way to improve the accuracy of the computation of the setup and hold slacks. On the other hand, this work progresses towards another direction based on the improved STAs: the objective of this work is to maximally relax the worst slack by using clock skew scheduling while integrating a flexible flip-flop timing model to maintain the accuracy of the setup and hold slack computation. Relaxing the worst slack is very important in that (i) it could resolve the setup or hold time violations, if exist, by a light clock skew tuning, thereby avoiding enormous effort to remove few time violations using (relatively unplanned) ECO knobs such as trial-and-error of gate sizing and threshold voltage swap etc. and (ii) it enables the circuit to be highly tolerant to the delay variation, which is very likely to be exposed in nano-scale high-speed designs.

3.3 Motivational Examples

The section describes how the previous STA approaches have been progressed, using small examples in Figs. 3.5 and 3.6 and where our work is positioned in Fig. 3.7. Fig. 3.5(a) depicts a circuit consisting of two flip-flops f_i and f_j and combinational logics between them with the specification of their maximum and minimum delays. The conventional non-flexible flip-flop timing model uses fixed values as their setup and hold time constraints to be satisfied for all flip-flops. For example, the timing model in Fig. 3.5(a) assumes $t_Q = 20$ for all flip-flops. Then, a pair of the values of t_S and t_H is chosen from the curve in Fig. 3.4 for $t_Q = 20$. Here, $t_S = t_H = 40$ are selected. Then, the conventional STAs set $t_{S_0} = t_S (= 40)$ and $t_{H_0} = t_H (= 40)$ and use them in slack computation in *Eqs.*(3.3) and (3.5). In Fig. 3.5(a), it is shown that the setup slack at f_j has negative slack ($= -30$), reporting time violation. However, it should be noted



(a) Conventional STAs which use *non-flexible flip-flop timing*. Close to averaged values of setup times and hold times are set to t_{S_0} ($= 40$) and t_{H_0} ($= 40$), respectively. Since $slk_S(j) = -30 < 0$, The STAs will report the setup time violation at f_j , but the decision uncertainty is high.



(b) Conventional STAs which *partially use flexible flip-flop timing* in which the inter-dependent relation between setup time and hold time for a fixed clk-to-Q delay are utilized. The STAs examines alternative valid pairs of setup and hold times that generate the same value ($= 20$) of t_Q to increase the accuracy of STAs. By changing t_{S_0} and t_{H_0} in f_j to 25 and 45, the setup slack at j is improved from -30 to -15 , and the decision uncertainty is reduced compared to that in (a).

Figure 3.5: Comparison of the conventional STAs which use nonflexible flip-flop timing and the enhanced STAs ([33, 35, 34, 36]) using flexible setup and hold times, but not clk-to-Q delay.

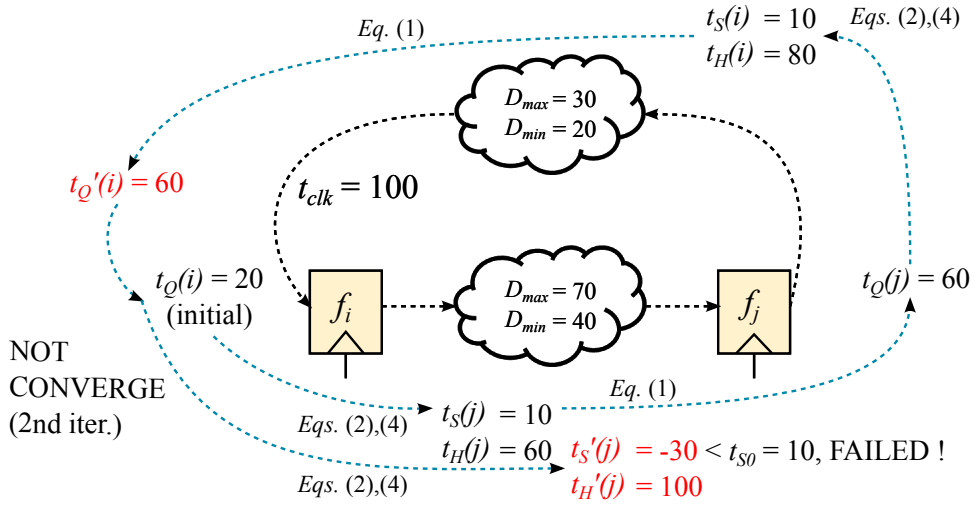


Figure 3.6: Conventional STAs [37, 38] which *fully use flexible flip-flop timing* in which the inter-dependent relation between clk-to-Q delay, setup skew, and hold skew are utilized. Initially, clk-to-Q delay of a flip-flop is set ($t_Q(i) = 20$ in this example). Then, setup skew $t_S(j) = 100 - 20 - 70 = 10$ and hold skew $t_H(j) = 20 + 40 = 60$, from which $t_Q(j)$ ($= 60$) is computed. In turn, $t_S(i) = 100 - 60 - 30 = 10$ and $t_H(i) = 60 + 20 = 80$, from which $t_Q(i)$ is computed to 60. This large value of $t_Q(i)$ causes $t_S(j) = 100 - 60 - 70 = -30$, which is far less than 10 ($= t_{S0}$). Thus, the STAs report a time failure at f_j .

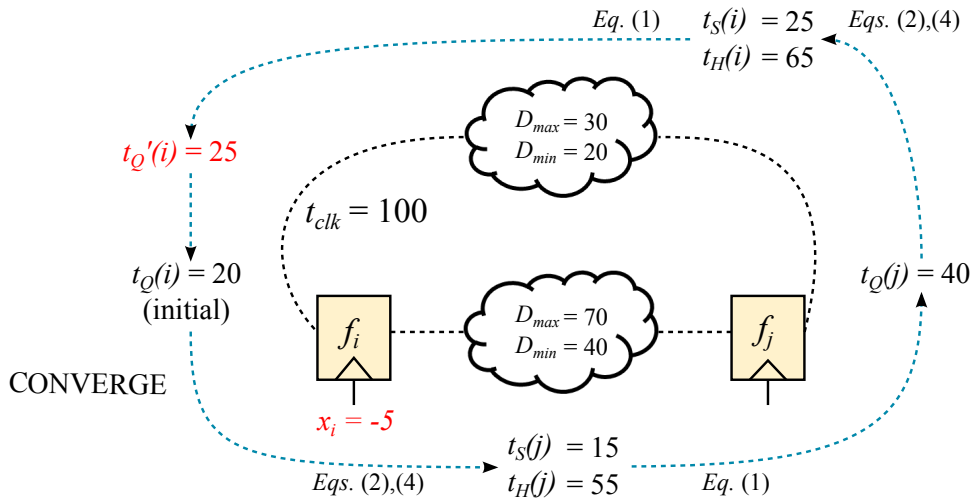


Figure 3.7: Clock skew scheduling integrated with flexible flip-flop timing. Tuning the clock arrival time at f_i to -5 causes to update the setup and hold skews, and clk-to-Q delays of flip-flops in the circuit, resulting in all nonnegative value of setup and hold slacks, indicating no time violation in the circuit.

that the existence of negative slack does not always mean a time violation in this non-flexible timing model since t_{S_0} and t_{H_0} to be satisfied were set to high values. To overcome the inaccuracy of STAs in Fig. 3.5(a), The STAs in [33, 35, 34, 36] utilized multiple pairs of t_{S_0} and t_{H_0} values for the given value of t_Q as indicated in Fig. 3.5(b), in which the values of t_{S_0} and t_{H_0} at f_j can be 25, and 45 as well as 40 and 40, respectively. Thus, the value of setup slack $slk_S(j)$ is computed to -15 , which is more accurate value than -30 in Fig. 3.5(a). The limitation of this STAs is that it does not consider the multiple values of t_Q , i.e., consider only the multiple pairs of the values of t_S and t_H .

Fig. 3.6 shows an iterative computation of setup and hold slacks used in [37], which further improves the accuracy of STAs used in Fig. 3.5(b). The setup and hold skew/slack computation in Fig. 3.6 is performed in the following way: flip-flop f_i is chosen and its $t_Q(i)$ is initially set to a number (here, 20). Then, according to *Eqs.*(3.2) and (3.4), the setup and hold skews at flip-flop f_j are computed, from which its $t_Q(j)$ is obtained by *Eq.*(3.1). From the $t_Q(j)$ value, the setup and hold skews of f_i are computed again by *Eqs.*(3.2) and (3.4), from which $t_Q(i)$ (red color) is obtained. Since the updated value of $t_Q(i)$ is now used again to compute the setup and hold skews (red color) of f_j . Since the setup slack $slk_S(j)$ ($= t_S(j) - t_{S_0} = -30 - 10 = -40 < 0$), the corresponding STAs report a setup time violation at f_j . Like this way, the STAs iteratively update setup and hold skews of flip-flops in the circuit until there is no further update (converging) or a time violation occurs.

Note that all the STAs used in Figs. 3.5(a) and (b), and Fig. 3.6 assumed that the clock signal arrival times (x_i and x_j) to f_i and f_j are all fixed⁵ However, if the clock arrival times can be tuned, the setup and hold slacks of the circuit

⁵Here, we assume $x_1, x_2, \dots = 0$ for all flip-flops f_1, f_2, \dots in the input circuit for simplifying the presentation.

may be improved, so that time violations previously existed (e.g., Fig. 3.6) can be removed or the circuit can be more tolerant to time variations afterwards. For example, in Fig. 3.7, x_i is set to -5 , which leads to ultimately $t_Q(i) = 25$ of f_i , which converges on the initial value of $t_Q(i)$.

3.4 Clock Skew Scheduling for Slack Relaxation Based on Flexible Flip-Flop Timing

3.4.1 Overall Flow

Proposed technique can be applied to an analyzed result produced by any STA and tries to enlarge the timing margins at flip-flops so that the worst slack is maximally relaxed. Our iterative technique called CSS-FT (clock skew scheduling utilizing flexible timing) consists of three steps and the core part(Step 2) can be utilized freely.

- **Step 1** (*Sorting flip-flops*): CSS-FT arranges the flip-flops in the circuit in a non-decreasing order according to the value of the worst slack of the setup and hold skews at every flip-flop. Let the sorted flip-flop list be $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$ and the corresponding values of the slack time be $\mathcal{S} = \{slk_{S/H}(1), slk_{S/H}(2), \dots, slk_{S/H}(n)\}$.
- **Step 2** (*Locally relaxing slack times*): for each f_j in \mathcal{F} , CSS-FT increases the value of $slk_{S/H}(j)$ by rescheduling the clock arrival time x_j at f_j or the clock arrival time(s) at the preceding flip-flops that directly drive f_j . Let f_i be a preceding flip-flop of f_j and f_k be a succeeding flip-flop that is directly driven by f_j . (See the partial circuit in Fig. 3.8.) The procedure of finding the value of x_i or x_j that maximizes the increase of the value of $slk_{S/H}(j)$ will be described in detail in Sec.3.4.2. Our procedure of relaxing local slack times should satisfy the following constraints:

1. $t'_{S/H}(j) > t_{S/H}(j)$ where $t'_{S/H}(j)$ and $t_{S/H}(j) (= \min\{t_S(j), t_H(j)\})$ are

the worst setup and hold skews at f_j after and before the reschedule of x_i or x_j , respectively.

2. $t'_{S/H}(j) \leq \min\{t'_{H/S}(j), t'_S(i), t'_H(i)\}$ when clock arrival time of flip-flop f_i is changed, and $t'_{S/H}(j) \leq \min\{t'_{H/S}(j), t'_S(k), t'_H(k)\}$ when clock arrival time of flip-flop f_j is changed.
3. $\gamma_1 \leq x_i, x_j \leq \gamma_2$ where γ_1 and γ_2 are the lower and upper bounds of the amount of (decrement/increment) adjustment of clock arrival times set by designer.

Constraint-1 ensures that the reschedule of x_i or x_j should improve the worst slack at f_j . *Constraint-2* guarantees that the resulting target slack is not able to exceed other related slacks. *Constraint-3* constrains the decrement/increment range of resetting x_i and x_j to be in a small interval. The values of γ_1 and γ_2 will be determined by designers through the analysis of layout congestion.

This local slack time relaxation is performed one at a time from the first flip-flop to the last in the sorted list \mathcal{F} . If the slack time relaxation for a flip-flop is successfully made, i.e., satisfying all of the three constraints, CSS-FT moves on the next iteration. If unsuccessful or all iterations are completed, CSS-FT checks if there is at least one flip-flop whose slack has been successfully relaxed. If it does, CSS-FT moves on Step 3. Otherwise, it stops the execution.

• **Step 3** (*Globally relaxing slack times*): this step spreads out the gain (i.e., the locally relaxed slack times) obtained in Step 2 over the entire flip-flops in the circuit. We apply the technique in [37] which repeatedly propagates the setup and hold skews until all clk-to-Q delays are converge. Then, since the setup and hold skews of flip-flops are globally refreshed, CSS-FT goes to Step 1 to look for a further improvement of slack times in the next iteration.

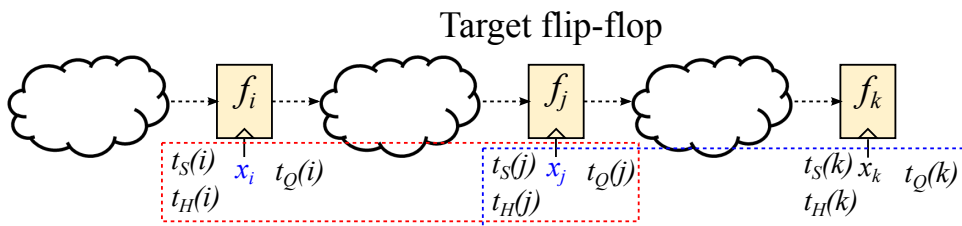


Figure 3.8: A partial circuit structure with three flip-flops f_i , f_j , and f_k , in which CSS-FT wants to improve the worst slack time at f_j by rescheduling clock arrival time x_i at f_i or x_j at f_j while satisfying the three constraints in Step 2.

3.4.2 Finding Local Clock Skew Schedule

The worst slack time at f_j in Fig. 3.8 can be improved by rescheduling the value of x_i or x_j . CSS-FT in Step 2 determines the rescheduling value by analyzing the changes of the setup and hold skews, and clk-to-Q delays of the driving flip-flop f_i and driven flip-flop f_k of f_j .

- **Case 1** (*scheduling x_i of driving flip-flop f_i of target f_j*): as shown in Eqs.(3.2) and (3.4), x_i is included in the equations of setup and hold skews of f_i and f_j . Thus, by abstracting all the parameters except x_i into constants, $t_S(-, i)$, $t_H(-, i)$, $t_S(i, j)$, and $t_H(i, j)$, which are respectively specified as $t_S(i)$, $t_H(i)$, $t_S(j)$, and $t_H(j)$ in Fig. 3.8, can be expressed as simplified equations with respect to x_i :

$$t_S(j) = -x_i - t_Q(i) + C_1 \quad (3.7)$$

$$t_H(j) = x_i + t_Q(i) + C_2 \quad (3.8)$$

$$t_S(i) = x_i + C_3 \quad (3.9)$$

$$t_H(i) = -x_i + C_4 \quad (3.10)$$

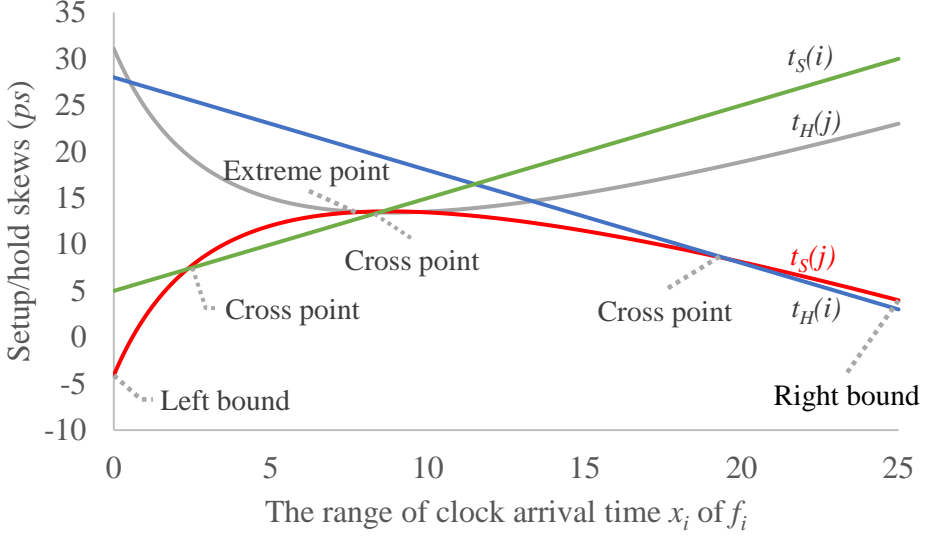


Figure 3.9: Illustration of curves, derived by Case 1 in Step 2 of CSS-FT, of the setup and hold skews ($t_S(i)$, $t_H(i)$, $t_S(j)$, $t_H(j)$) at flip-flops f_i and f_j in Fig. 3.8 with respect to x_i .

where C_1 , C_2 , C_3 , and C_4 are the abstracted constants. Precisely, $C_1 = x_j + T_{clk} - D_{max}(i, j)$, $C_2 = -x_j + D_{min}(i, j)$, $C_3 = -x_h + T_{clk} - t_Q(h) - D_{max}(-, i)$, and $C_4 = x_h + t_Q(h) + D_{min}(-, i)$, assuming flip-flop f_h drives f_i .

Then, substituting $t_Q(i)$ in $t_S(j)$ and $t_H(j)$ with the right term in Eq.(3.1) reduces to:

$$t_S(j) = -x_i - \left(a_0 + \frac{a_1}{(x_i + C_3) - t_{S_0}} + \frac{a_2}{(-x_i + C_4) - t_{H_0}} \right) + C_1 \quad (3.11)$$

$$t_H(j) = x_i + \left(a_0 + \frac{a_1}{(x_i + C_3) - t_{S_0}} + \frac{a_2}{(-x_i + C_4) - t_{H_0}} \right) + C_2 \quad (3.12)$$

Fig. 3.9 illustrates the four curves of $t_S(j)$, $t_H(j)$, $t_S(i)$, and $t_H(i)$ with respect to x_i , assuming that $\gamma_1 = 0$ and $\gamma_2 = 25$. For example, to increase the value

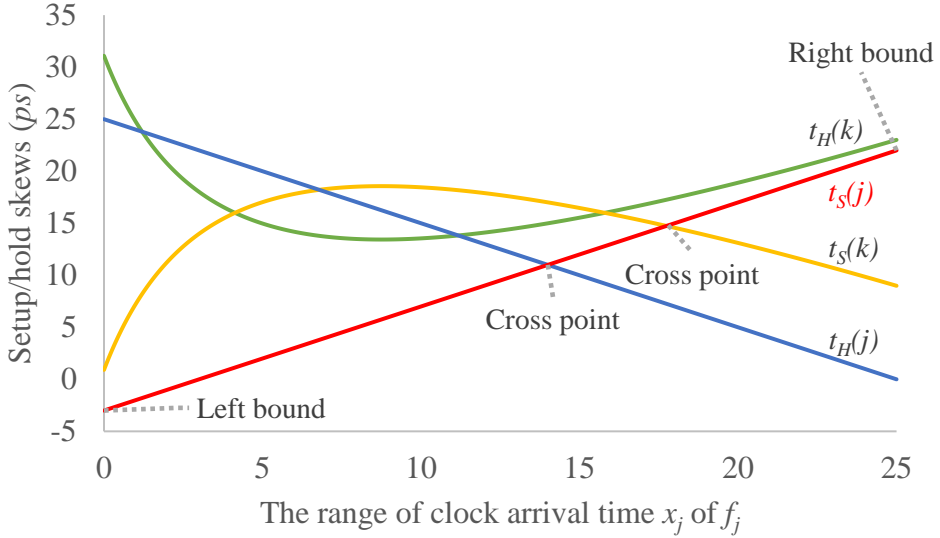


Figure 3.10: Illustration of curves, derived by Case 2 in Step 2 of CSS-FT, of the setup and hold skews ($t_S(j)$, $t_H(j)$, $t_S(k)$, $t_H(k)$) at flip-flops f_j and f_k in Fig. 3.8 with respect to x_j .

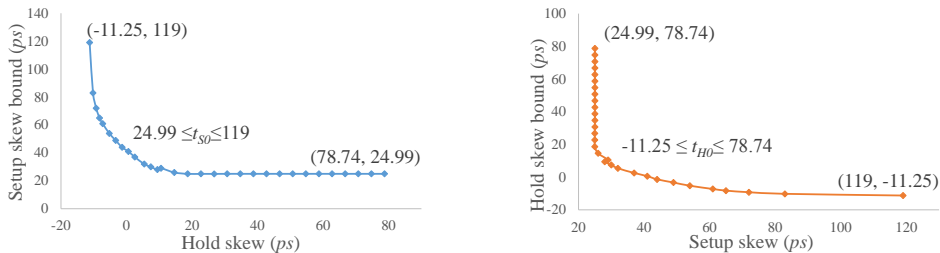
of $t_S(j)$, the candidate values of x_i to be examined are the ones in the cross points of the curve of $t_S(j)$ and the other curves, and the extreme points, and the bound values (γ_1 and γ_2). Among the candidate values of x_i , CSS-FT selects the one which maximizes the value of $\min\{t_S(j), t_H(j)\}$ while meeting the three constraints in Step 2.

- **Case 2** (*scheduling x_j of target flip-flop f_j*): by checking Eqs.(3.2) and (3.4), rescheduling x_j in Fig. 3.8 may change the values of $t_S(j)$, $t_H(j)$, $t_S(k)$, and $t_H(k)$. With the analysis similar to that in Case 1, the curves for $t_S(j)$, $t_H(j)$, $t_S(k)$, and $t_H(k)$ with respect to x_j can be obtained. Fig. 3.10 illustrates the curves, from which CSS-FT checks the cross points, extreme points, and bound values of x_j as Case 1 does.

The time complexity of CSS-FT is bounded by $O(K(n \log n + nT_1 + T_2))$ where $n \log n$ is the time to sort n flip-flops in Step 1, T_1 is the time to extract all

candidate values of x_i and x_j by a numerical analysis method in Step 2, and T_2 is the time taken by the application of the conventional STA in Step 3. K is the number of iterations. Note that CSS-FT is designed to incrementally relax the worst slack as well as maximize the amount of total slack by localizing the worst slack relaxation in Step 2 at the expense of the cost of clock skew rescheduling. Thus, designers are able to control the value of K while monitoring the trade-off between the amount of slack relation and the scheduling (area) overhead.⁶

3.5 Experimental Results



(a) Minimally required setup skews (t_{S_0}) wrt. hold skews.

(b) Minimally required hold skews (t_{H_0}) wrt. setup skews.

Figure 3.11: (a) Curve showing the change of the setup skew lower bound (t_{S_0}) as the hold skew changes. (t_{S_0} for a value of hold skew is used for calculating the setup slack in Eq (3.3)). (b) Curve showing the change of the hold skew lower bound (t_{H_0}) as the setup skew changes. (t_{H_0} for a value of setup skew is used for calculating the hold slack in Eq (3.5)).

Our flexible flip-flop timing based clock skew scheduling technique CSS-FT is implemented in C++ and Python3, and run on Linux machine with 8 cores of 3.50GHz Intel i7 CPU and 16GB memory. First, we extracted an extensive data set of the setup skew, hold skew, and clk-to-Q delay of a flip-flop by using HSPICE simulation with 45nm NanGate Open Cell library [24], and characterized the inter-dependent relation between them by fitting the

⁶Supporting results are provided in the experiments.

data (using Matlab [40]) into the equation (in Eq 3.1) proposed by the flexible timing based STA in [37]. We also extracted curves that show the values of the minimum setup skew as the hold skew varies and the values of the minimum hold skew as the setup skew varies. The curves are depicted in Figs. 3.11(a) and (b), from which we derived accurate values of t_{S_0} (in y -axis in (a)) for various values of hold skew (in x -axis in (a)) and t_{H_0} (in y -axis in (b)) for various values of setup skew (in x -axis in (b)) for each flip-flop to check the existence of time violation.

We tested our CSS-FT on a set of benchmark circuits in ISCAS89 (s-series in Table 3.1) [22], ITC99 (b22 in Table 3.1) [23] and an intra-prediction circuit (h.264 in Table 3.1) of H.264. The testcases were synthesized with *Synopsys IC Compiler* with 45nm Nangate Open Cell Library [24]. We extracted the time information on the connections between flip-flops and the location of the flip-flops. In addition, the initial schedule of clock signal arrival times (i.e., clock skew schedule) was obtained by using *Mosek* [41].

Table 3.1 shows a comparison of the numbers of setup and hold time violations before and after the application of our clock skew scheduling based on flexible timing model to the timing analysis results produced by the STA in [37]. The third, fourth, and fifth columns show a comparison of the numbers of setup and hold time violations when the clock period for each design is set to the value in the corresponding row of column T_{clk0} . We determined the T_{clk0} value for each design to be the minimum clock period that allows no setup and hold time violation *after* the application of CSS-FT. From the comparison of the numbers of violations in the fourth and fifth columns, we can see that CSS-FT has relaxed the worst slack significantly. The next three groups of three columns show the comparisons of the numbers of setup and hold time violations when the clock periods are set to $T_{clk0} \times 0.95$, $T_{clk0} \times 0.90$, and $T_{clk0} \times 0.85$. The com-

Table 3.1: Comparison of the numbers of setup and hold time violations, for various values of clock period, before and after the application of our CSS-FT starting from as input the timing analysis results produced by the flexible flip-flop timing based STA in [37]. Clock skew tunable range of each flip-flop is set to $[-30ps, 30ps]$.

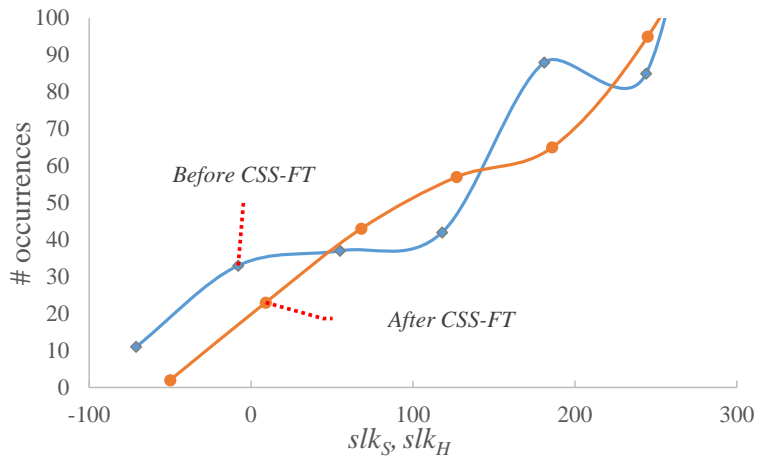
Circuit	#FFs	T_{clk} (ps)	# Violations		T_{clk} (ps)	# Violations	
			before	after		before	after
s382	30	430ps (2.33GHz)	0	0	410	26	20
s386	20	450ps (2.22GHz)	0	0	420	24	17
s1423	111	1210ps (826MHz)	19	0	1150	43	23
s5378	247	610ps (1.64GHz)	305	0	580	390	57
s13207	482	460ps (2.17GHz)	244	0	440	363	111
s15850	235	560ps (1.79GHz)	0	0	530	254	254
s38417	1698	930ps (1.08GHz)	0	0	880	2150	1945
s38584	1458	1040ps (962MHz)	0	0	990	1213	1209
b22	638	1340ps (746MHz)	26	0	1280	8415	8188
h264	6665	2600ps (385MHz)	8574	0	2470	16863	15068
Avg.ratio.		(1.00)	1	0	(0.95)	1	0.8328
Circuit	#FFs	T_{clk} (ps)	# Violations		T_{clk} (ps)	# Violations	
			before	after		before	after
s382	30	390	29	27	370	33	32
s386	20	400	28	27	380	31	30
s1423	111	1090	82	63	1030	121	109
s5378	247	550	506	351	520	611	470
s13207	482	420	410	362	390	513	447
s15850	235	500	292	286	480	305	304
s38417	1698	840	3354	3207	790	4899	4725
s38584	1458	940	1398	1389	890	1577	1576
b22	638	1210	13086	12977	1140	14459	14375
h264	6665	2340	22947	20473	2210	32397	29771
Avg.ratio.		(0.90)	1	0.8755	(0.85)	1	0.92

Table 3.2: Minimum clock period before and after the application of our CSS-FT. Clock skew tunable range of each flip-flop is set to $[-30ps, 30ps]$.

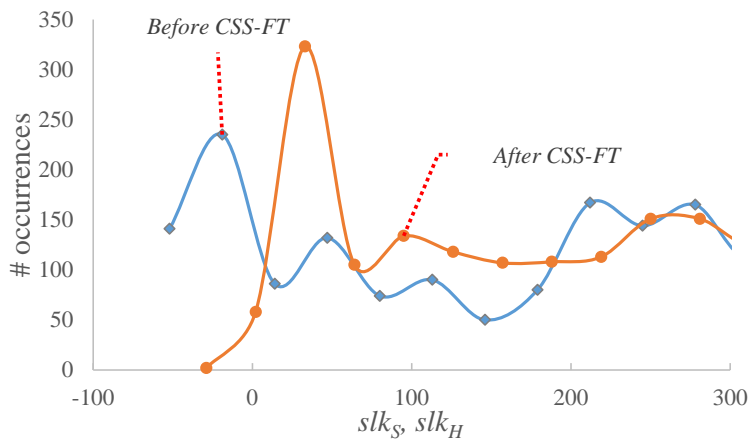
Circuit	# FFs	T_{clk} (ps)	
		<i>before</i>	<i>after</i>
s382	30	428.56	423.61
s386	20	440.53	440.53
s1423	111	1221.86	1200.86
s5378	247	632.02	609.02
s13207	482	468.1	459.96
s15850	235	553.69	553.69
s38417	1698	928.77	924.8
s38584	1458	1037.42	1037.42
b22	638	1344.42	1337.63
h264	6665	2629.28	2599.27

parisons show that as the clock period constraint is close to T_{clk0} , the number of time violations produced by the application of CSS-FT is relatively much less than that of the initial circuit before the application. In summary, CSS-FT relaxes the worst slack of circuits, so that the minimum clock period of the input circuits with no time violation is reduced to the clock period (T_{clk0}), which is 4.2% shorter on average. To put it in another way, the clock speed of the tested designs in Table 3.1 is improved from 369MHz~2.23GHz to 385MHz~2.33GHz. It also reduces the total numbers of setup and hold time violations by 27.7%, 9.5%, and 6.5% on average when the clock periods are set to 95%, 90%, and 85% of the value of T_{clk0} , respectively. This trend is well understood by examining the distribution graphs of setup and hold slacks before and after the application of CSS-FT shown in Fig. 3.12. Figs. 3.12(a) and (b) show the distributions of the numbers of occurrences of the setup and hold slacks of all flip-flops with respect to the change of the values of their setup and hold slacks for designs s1423 and s5378 when T_{clk} is set to $T_{clk0} \times 0.95$ in Table 3.1, respectively.

Table. 3.2 shows the reduced minimum clock period after CSS-FT is applied



(a) s1423.



(b) s5378.

Figure 3.12: The distribution of the numbers of setup and hold slacks of all flip-flops for designs s1423 and s5378 before (blue curve) and after (orange curve) the application of CSS-FT to the initial timing analysis results produced by the flexible flip-flop timing based STA in [37]. T_{clk} is set to $T_{clk0} \times 0.95$ in Table 3.1.

Table 3.3: Running time and overhead of our CSS-FT.

Circuit	#FFs	Run time (s) [37] + <i>Ours</i>	Overhead (μm^2)
s382	30	0.001	0.798
s386	20	0.001	0.798
s1423	111	0.001	18.354
s5378	247	0.001	123.69
s13207	482	0.001	138.852
s15850	235	0.001	< 0.798
s38417	1698	0.06	122.892
s38584	1458	0.01	14.364
b22	638	0.04	87.78
h264	6665	13.99	5208.546

to the circuits. The third and fourth columns show a comparison of the minimum clock period before and after CSS-FT is applied. The fifth and sixth columns are the maximum clock frequency. We can see that the minimum clock period is decreased up to $30.01ps$ that is critical to designer.

Table 3.3 show the run time of CSS-FT where [37]+*Ours* indicates the total time consumed by CSS-FT in which the STA in [37] is used in the global time refresh step (i.e., Step 3) of CSS-FT, and the buffer area overhead used to reschedule the clock arrival times. For example, for design with about 6660 flip-flops, CSS-FT takes 14 seconds and uses area of $5200\mu m^2$, which amounts to only 0.0052% of $1cm \times 1cm$ die area.

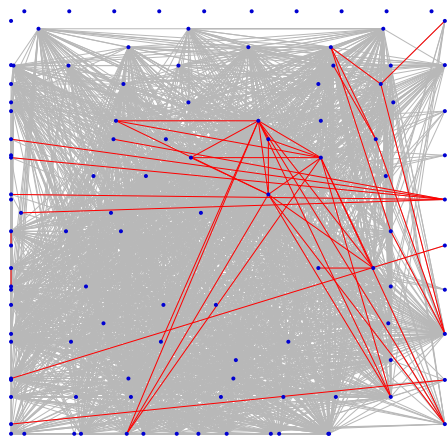
Figs. 3.13(a) and 3.13(b) show the distribution of setup and hold time violations of s1423 with $T_{clk} = 1150ps$ (i.e., 870MHz clock speed) before and after the application of CSS-FT to the result produced by the STA in [37], respectively. In addition, Figs. 3.13(c) and 3.13(d) show the distribution of setup and hold time violations of s5378 with $T_{clk} = 580ps$ (i.e., 1.72GHz clock speed) before and after the application of CSS-FT, respectively. Finally, Fig. 3.14 show the trade-off between the minimal clock period and the area overhead required

to reschedule clock times produced by CSS-FT where the data are extracted and averaged from the test results of ISCAS89 circuits. It is shown that by controlling the range of clock tuning interval $[-\gamma, \gamma]$, CSS-FT is able to reschedule clock skew with minimal area overhead while meeting clock period constraint.

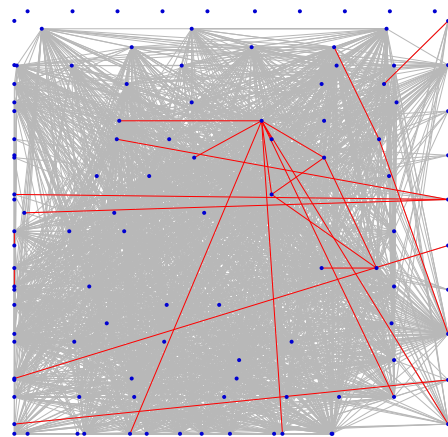
Fig. 3.15 shows the distributions of clk-to-Q delay of all flip-flops before and after CSS-FT is applied. CSS-FT also decreases clk-to-Q delays of most of flip-flops. In the flexible flip-flop timing model curve of Fig. 3.1, the sensitivity of clk-to-Q delay is decreased when the value is decreased. Therefore, the application of CSS-FT is able to decrease the problem of process variation.

3.6 Summary

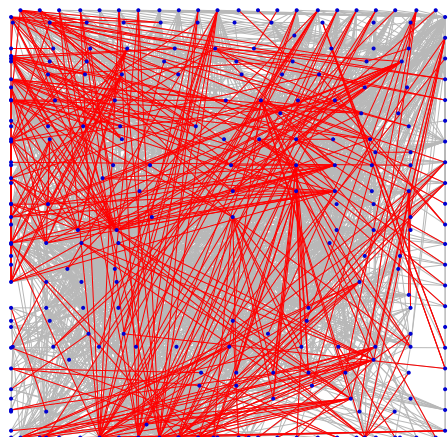
Recent works have consistently claimed and validated the importance of incorporating the flexible flip-flop timing, which describes the inter-dependent relation of setup skew, hold skew, and clk-to-Q delay of flip-flops, into static timing analysis (STA) tools in order to analyze the circuit timing more reliably and accurately. In the light of the importance, this work *addressed a new problem of clock skew optimization integrated with flexible flip-flop timing to complement the conventional flexible timing based STAs. We showed that a step-by-step localized slack time relaxation was possible by devising a careful clock skew tuning technique that exploited the inter-dependent time relation and able to incrementally and effectively increase the time margins of circuits.*



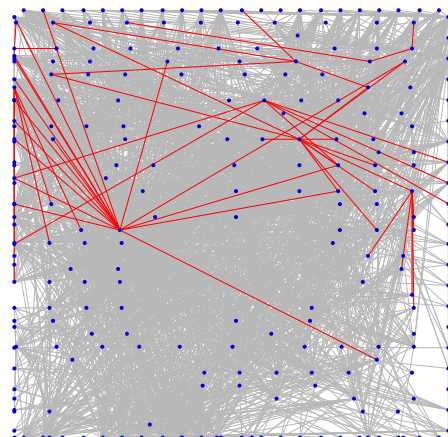
(a) The distribution of setup and hold time violations (red lines) of s1423 with $T_{clk} = 1150ps$ (870MHz) *before* the application of CSS-FT for the result produced by the STA in [37].



(b) The distribution of setup and hold time violations (red lines) of s1423 with $T_{clk} = 1150ps$ (870MHz) *after* the application of CSS-FT for the result produced by the STA in [37].



(c) The distribution of setup and hold time violations (red lines) of s5378 with $T_{clk} = 580ps$ (1.72GHz) *before* the application of CSS-FT for the result produced by the STA in [37].



(d) The distribution of setup and hold time violations (red lines) of s5378 with $T_{clk} = 580ps$ (1.72GHz) *after* the application of CSS-FT for the result produced by the STA in [37].

Figure 3.13: The distribution of time violations before and after the application of CSS-FT to the timing results of s1423 and s5378 produced by the STA in [37]. The blue dots indicate flip-flops.

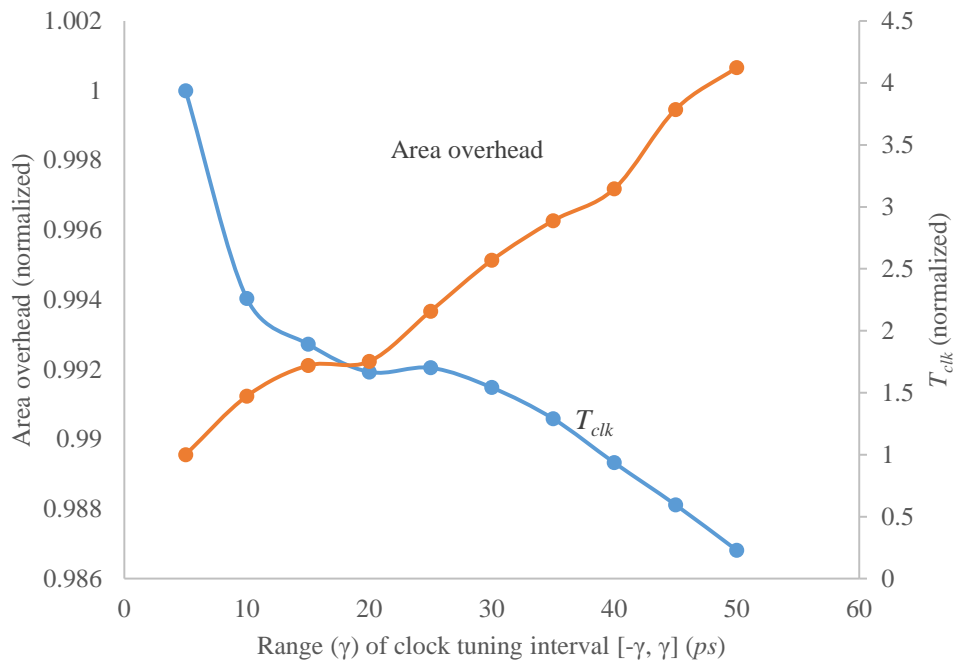
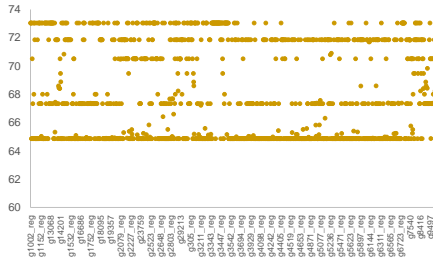
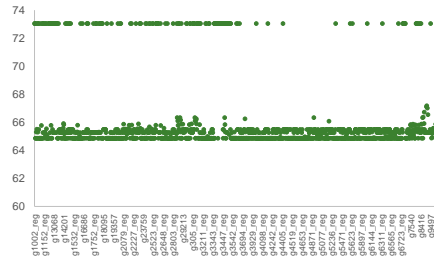


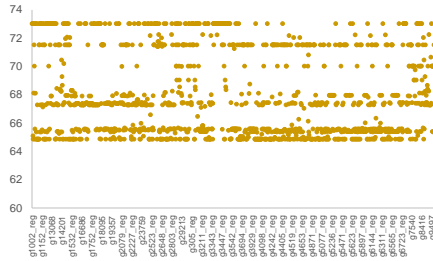
Figure 3.14: The curves showing the trade-off between the minimum clock period and the area overhead required for clock skew rescheduling used by CSS-FT where the data are extracted and averaged over the results of ISCAS89 circuits.



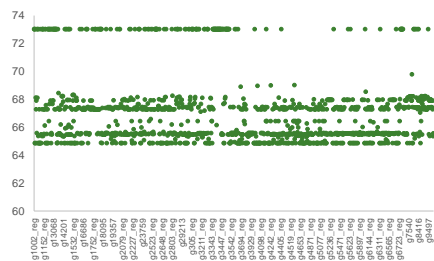
(a) s13207 *before*.



(b) s13207 *after*.



(c) s38584 *before*.



(d) s38584 *after*.

Figure 3.15: The distributions of clk-to-Q delays of all flip-flops for designs s13207 and s38584 before (yellow points) and after (green points) the application of CSS-FT to the initial timing analysis results produced by the flexible flip-flop timing based STA in [37]. T_{clk} is set to T_{clk0} in Table 3.1. All the clk-to-Q delays of s13207 are decreased and only 10 clk-to-Q delays are slightly increased in the results of s38584.

Chapter 4

SYNTHESIS FOR POWER-AWARE CLOCK SPINES

This chapter presents a methodology to automate the exploration and synthesis of clock spine networks.

4.1 Introduction

Minimizing the amount of power consumed by the clock network is an important concern in the synthesis of clock structures. Clock gating, which prunes the clock network to disable portions of circuit by preventing the flip-flops in them from switching states, is a popular technique in synchronous circuits for reducing dynamic power dissipation. Several researchers have attempted to apply the clock gating technique *before* synthesizing clock networks. Some representative works include [42] in the register-transfer-level (RTL), [43] in the architecture level, and [44] in the logic level. The objective of the clock gating at these levels

is minimizing the number of switching activities at the sinks. One drawback of clock gating at the stages is that since the placement information of the sinks is unavailable, the clock gating results often yield unnecessary detours and wire snaking afterward [45]. To overcome the limitation, many work attempted to integrate the clock gating into the framework of clock network synthesis. Tellez *et al.* [46] built a clock tree topology by using the placement information and pre-defined activity patterns of the clock sinks in a way to minimize a weighted sum of the total wirelength of clock tree and (estimated) total number of activities of the sinks. The activity pattern of a sink used in the work refers to a sequence of the per-cycle enable/disable (i.e., 1/0) status of the sink. They captured the activity patterns of sinks from the results of high-level synthesis. Once a clock tree topology is obtained, they properly insert clock gating cells to the clock tree. Chen, Kang, and Sarafzadeh [47] elaborated the procedure of inserting clock gating cells used in [46]. They recursively, from the sinks of clock tree toward the root, performed a bitwise-OR operation for the activity patterns of every pair of child nodes in the clock tree. The computed activity pattern at the corresponding parent clock node represents the sequence of enable/disable status of the portion of circuit driven by the clock signal from the parent node. They used the activity patterns to determine if clock gating cells are inserted at the corresponding clock nodes. Oh and Pedram [48, 49] utilized an instruction stream to extract a probabilistic information of activity patterns on clock nodes, based on which they calculated the switched downstream capacitance of each node to relatively estimate the amount of power consumed by the portion of clock tree driven by the node. Chao and Mak [50] improved the previous works in [48, 49] by constructing low-power gated clock trees with zero clock skew. Later, Lu, Chow, and Sham [51] proposed fast algorithms for low-power gated clock tree with clock skew and slew constraints.

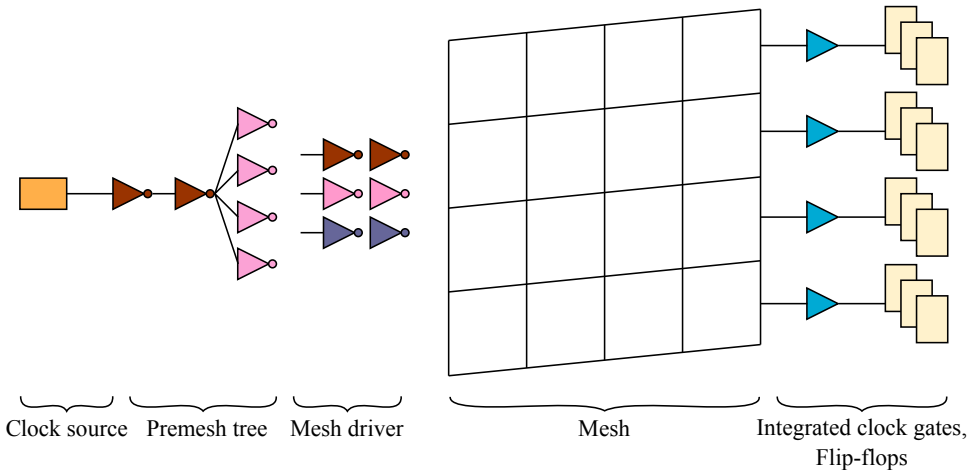


Figure 4.1: Clock mesh synthesis given by *Synopsys IC Compiler* [53].

While there are lots of clock tree synthesis algorithms considering clock gating technique, research on other gated clock structures is rarely announced. To our knowledge, the work by Lu, Mao, and Taskin [52] is the only one which has addressed gated clock network other than clock tree. They proposed a synthesis algorithm for clock mesh that enables clock gating on their local sub-trees attached to the mesh. The proposed algorithm includes flip-flop clustering and placement to form the local sub-trees. Even though the synthesis algorithm deployed gated local sub-trees under the clock mesh to save power, since the clock mesh should be enabled even when exactly one sink is to be activated in a clock cycle, the overall power saving is inherently very limited. *Synopsys IC Compiler* is able to handle integrated clock-gating cells which are generated by RTL or Gate-level design flow (i.e. *Synopsys Power CompilerTM*) and offers clock mesh structure that is similar to the design of [52]’s work. Fig. 4.1 shows the structure of clock meshes which are supported by *IC Compiler*. In the same manner, the structure cannot locate the clock gate over the clock mesh grid, the power reduction is limited. In this chapter, we first propose a synthesis

algorithm for clock spines with the capability of clock gating. The clock spine structure offers a higher tolerance to the delay variation over that of clock trees while it allows an opportunity to reduce more power over that of the clock meshes.

The rest of the chapter is organized as follows. Section 4.2 includes some preliminaries and motivation of the work. Section 4.3 describes our proposed algorithm for the synthesis of gated clock spine networks. Then experimental results to check the effectiveness of our method are provided in Section 4.4. Lastly, we summarize the chapter in Section 4.5.

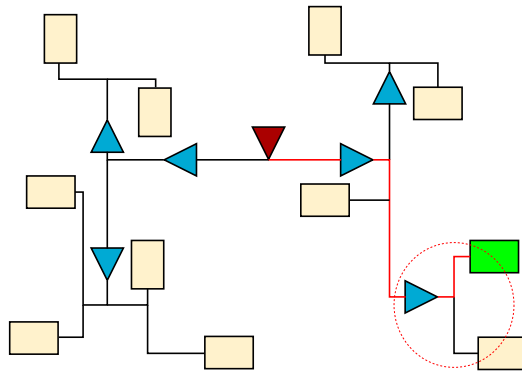
4.2 Preliminaries and Motivation

4.2.1 Clock Spine

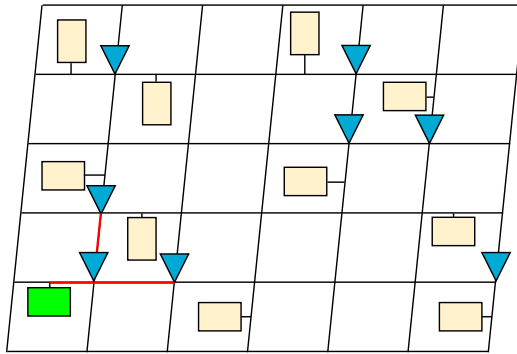
Clock spine structure is composed of a set of vertical and horizontal clock wires. Though it has similarities with other clock structures, its distinguished characteristics separate it from others.

In Fig. 4.2, the differences among clock tree, clock mesh and clock spine structure are presented. Clock sources in clock mesh and clock spine structure are able to give multiple clock paths to clock sinks while that in clock tree only gives single path (clock path with red colored). Since subtrees in clock tree are driven and isolated by a single buffer, the analysis of clock tree is easier than that of clock mesh and clock spine structure. When compared to clock mesh, buffer insertion of clock spine structure is only limited on each local spine, while clock mesh should insert buffers to a global mesh grid. Surely, mesh grid in clock mesh can be removed and spines in clock spine structure can be connected.¹ Eventually, structures of clock mesh and clock spine structure are equalized.

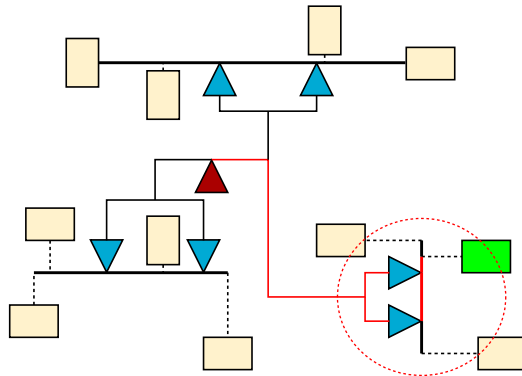
¹The merging technique will be presented in the following sections.



(a) Clock tree.



(b) Clock mesh.



(c) Clock spine.

Figure 4.2: Comparison among clock spine and other two clock structures. Clock is transitted from clock source(red triangle) to every clock sinks. Compared to clock tree which has a single clock path for each clock source to sink, clock mesh and clock spine structure have multiple clock paths. Buffers in clock tree only drive their own isolated subtree but those in clock mesh and clock spine structure drive mesh grid or spine together.

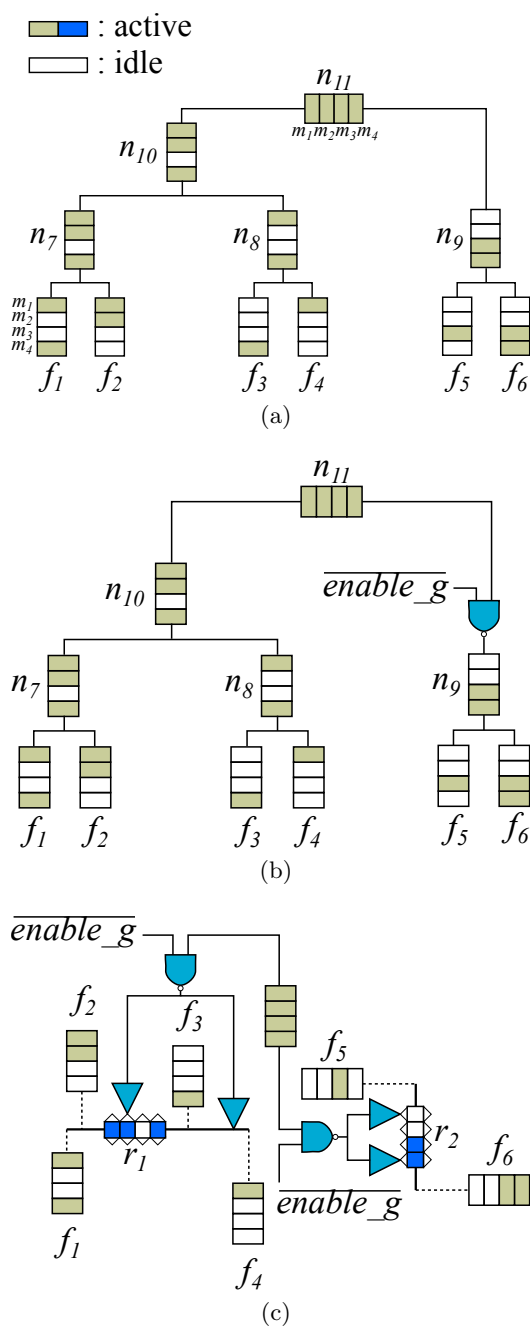


Figure 4.3: An example illustrating activity patterns and clock gating in clock tree and clock spine networks. (a) The generation of activity pattern by the bottom-up process, in a clock tree, from the activity patterns of sinks. (b) The clock tree gating at node n_{10} in (a). (c) A clock spine network with two spines being gated separately.

4.2.2 Activity Patterns

Clock gating is a technique to block the activation of a portion of circuit by disabling the flip-flops (i.e., sinks) in the sub-circuit. Thus, it is essential to know the power modes at which individual sinks require clock signal for launching/capturing data. Such information can be expressed by *activity patterns* [54]. Activity pattern of a sink is a sequence of binary numbers where value-1 in position i means the sink should be enabled at the power mode m_i and value-0 means it does not need to be enabled. For example, Fig. 4.3(a) shows the activity patterns of 6 sinks in the circuit with four power modes m_1 , m_2 , m_3 , and m_4 , in which sink f_5 has activity pattern of [0010], indicating f_5 can be idle in m_1 , m_2 and m_4 , and f_6 has [0011], indicating f_6 can be idle only in m_1 and m_2 . Thus, by inserting gating cell at n_9 , it is possible to let both f_5 and f_6 be idle in m_1 and m_2 , as shown in Fig. 4.3(b). Likewise, the activity pattern of a spine can be extracted by performing bitwise-OR operations for all patterns of the sinks attached to the spine. Fig. 4.3(c) shows an example of generating activity patterns of spines, in which two gating cells are inserted to gate the sinks f_1 , f_2 , f_3 , and f_4 in spine r_1 and to gate the sinks f_5 and f_6 in r_2 .

4.2.3 Power Computation

The amount of power, $P_{m_i}^{tot}$, consumed by the activation of circuit with a gated clock spine network \mathcal{S} at power mode m_i can be expressed as:

$$P_{m_i}^{tot} = \sum_{r_j \in R} P_{m_i}^{Clk}(r_j) + P_{m_i}^{nonClk} \quad (4.1)$$

where R is the set of spines, $P_{m_i}^{Clk}(r_j)$ is the power dissipated at mode m_i by the spine r_j together with the sinks and buffers in r_j , and $P_{m_i}^{nonClk}$ represents the total power dissipated at m_i by the circuit other than the clock resources.

Let D_i be the total time duration in which the circuit is activated in power mode m_i . Then, the total energy, E^{tot} , we want to minimize is the quantity of

$$E^{tot} = \sum_{m_i \in M} P_{m_i}^{tot} \cdot D_i \quad (4.2)$$

where M represents the set of power modes.

Fig. 4.4 shows the computation of power and energy consumption for three instances of clock spine network, assuming that the D_i value for every power mode equals 10. The comparison of the clock spine structures in Figs. 4.4(a) and (b) indicates that the energy consumption can vary according to the grouping of sinks to form spines even though the same number of spines is used to maintain the clock skew. On the other hand, the comparison of the clock spine structures in Figs. 4.4(a) and (c) implies that allocating more spines is likely to dissipate more power, but offers a better control of clock skew. The goal of this work is to explore the structures of clock spine network and find the one with a minimal energy consumption while satisfying the clock skew and slew constraints.

4.3 Algorithm for Clock Spine Synthesis

4.3.1 Problem Definition

The clock spine synthesis problem we want to solve is described as:

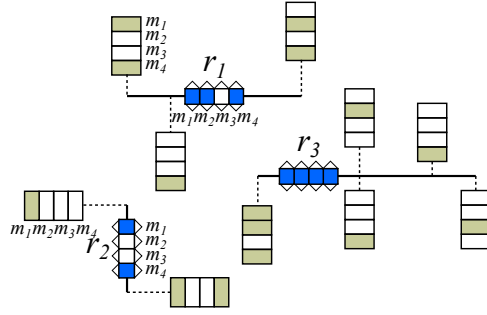
Problem 4.1 (Clock spine synthesis): *Given an activation pattern set A for sinks, a power mode set M with activation duration, and the values of $P_{m_i}^{nonClk}$ (Eq.4.1) in a circuit \mathcal{C} , generate a clock spine structure \mathcal{S} which has the least amount of E^{tot} (Eq.4.2) while satisfying the clock skew and slew constraints.*

An instance of clock spine structure is uniquely defined by specifying:

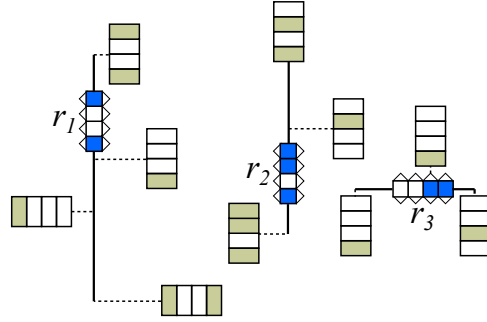
- *spec-1* the number of spines allocated and their activation patterns;²

²Clock gating is applied per-spine basis in this work for simplicity. Clock gating for multiple spines will be considered at the stage when a top-level clock tree is constructed. (See the “spine gating” at Step 4 in Fig. 4.5.)

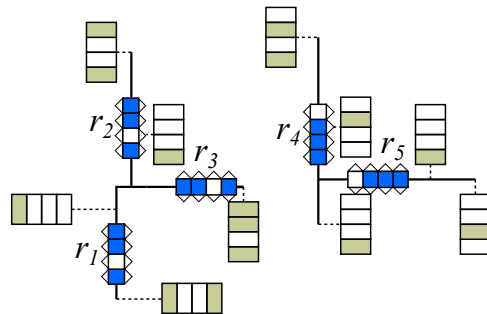
: active
 : idle



(a) A clock spine network with $|R| = 3$. $P^{tot} = 30 + 45 + 71 = 146$, $E^{tot} = 30 \times 0.5 + 45 \times 0.75 + 71 \times 1 = 119.75$.



(b) Another clock spine network with $|R| = 3$. $P^{tot} = 61 + 44.5 + 41.25 = 146.75$, $E^{tot} = 61 \times 0.5 + 44.5 \times 0.75 + 41.25 \times 0.5 = 84.5$.



(c) A clock spine network with $|R| = 5$. $P^{tot} = 30 + 30 + 25 + 43 + 41.25 = 169.25$, $E^{tot} = 0.75 \times (30 + 30 + 25 + 43 + 41.25) = 126.94$.

Figure 4.4: Comparison of the structure and energy consumption for three instances of clock spine network. (a) A clock spine network with $|R| = 3$. (b) A clock spine network different from that in (a) and $|R| = 3$, but less energy consumption. (d) A clock spine network different from that in (a), but $|R| = 5$ and less energy consumption.

- *spec-2* the length and (vertical or horizontal) shape of the spines;
- *spec-3* the location at which the spines are placed;
- *spec-4* the sinks attached to the spines;
- *spec-5* the number of clock buffers that drives the spines;³
- *spec-6* the location at which the spine buffers are placed.

Note that the power consumption of a clock spine instance is directly affected by *spec-1*, *2*, *4*, and *5*, while the clock skew and slew are affected by *spec-4*, *5*, and *6*. Since the specification items are tightly related each other and it is practically very timing consuming to explore all feasible clock spine candidates, we solve the problem of clock spine synthesis by performing four steps. Precisely, our proposed power-aware clock spine synthesis called CSPINE consists of four steps: (Step 1) *Clustering clock sinks*, (Step 2) *Relaxing spines*, (Step 3) *Inserting spine buffers*, and (Step 4) *Constructing a top-level clock tree*. Fig. 4.5 shows the flow of the four steps, which will be described in detail in the next four subsections.

4.3.2 Power-Aware Sink Clustering

The sink clustering in CSPINE consists of two loops, one loop nesting the other. Every iteration of the *outer-loop* produces a clock spine structure \mathcal{S} whose description is given by the data on *spec-1*, *2*, *3*, and *4*. The *outer-loop* iterates until there is no further reduction in E^{tot} in Eq.4.2. Then, before moving on the *inner-loop*, CSPINE computes, for each spine $r_i \in R$ of \mathcal{S} , the mean location, (\bar{x}_i, \bar{y}_i) of the sinks attached to r_i , and nullifies the sink attachment in S . Each of the mean locations is assumed to have a dummy spine with no physical size. The *inner-loop* performs the following three actions.

³Buffer sizing is also considered in this work.

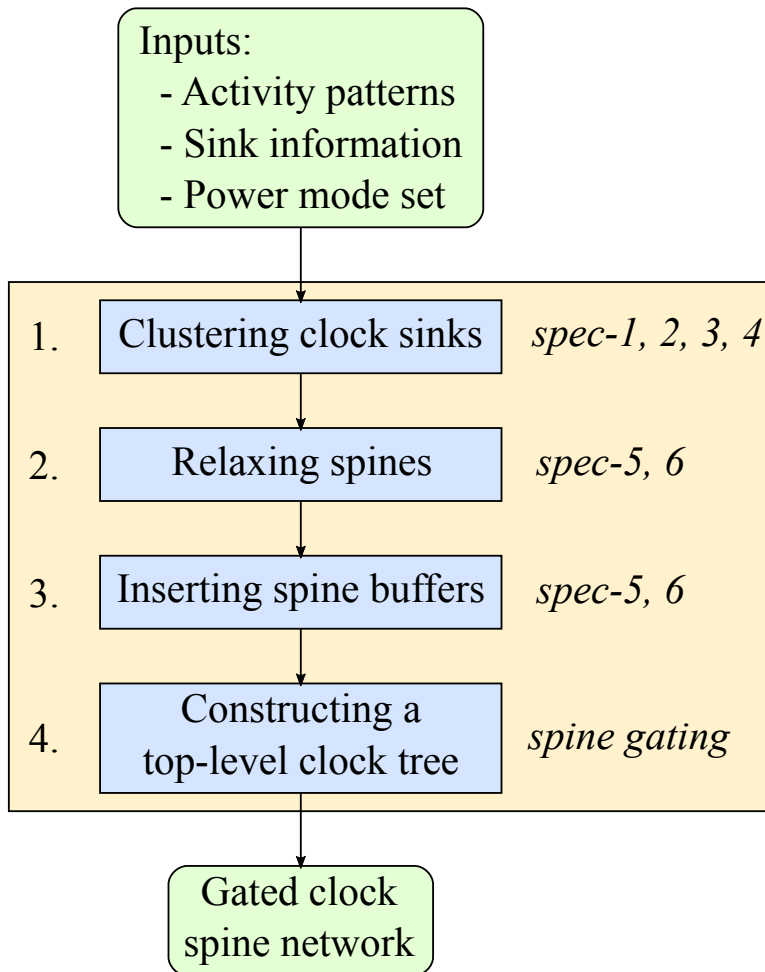


Figure 4.5: Four steps of the proposed power-aware clock spine synthesis algorithm.

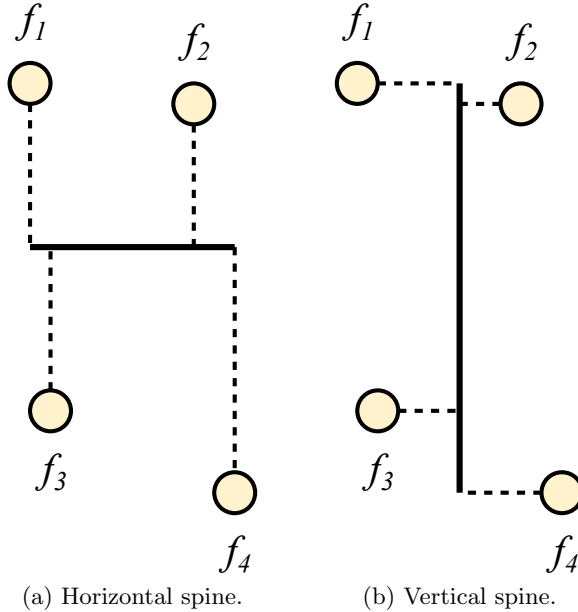


Figure 4.6: Spine allocation and placement for covering four sinks f_1, f_2, f_3, f_4 . (a) Horizontal spine, (b) Vertical spine.

1. The *inner-loop* iteratively assigns the sinks to the existing dummy or non-dummy spines, one sink at an iteration. Whenever a sink is assigned to a spine, the spine is updated to a power-minimal spine. For example, Fig. 4.6 shows the two possible updates of spine when sink f_4 is attached to a spine with sinks f_1, f_2 , and f_3 . It is seen that the right one is superior to the left one in terms of wirelength and power consumption.

2. At each iteration of the *inner-loop*, the selection of spine to which a sink, say f_k , is to be attached is determined by computing the value of $\Delta E^{tot}(f_k, r_j)$, which indicates the increase of total energy in Eq. 4.2 caused by attaching f_k to an existing spine r_j . For example, Fig. 4.7(a) shows the situation in which sink f_5 is to be attached to one of the two spines r_1 and r_2 . Fig. 4.7(b) shows the updated power-minimal spines. CSPINE

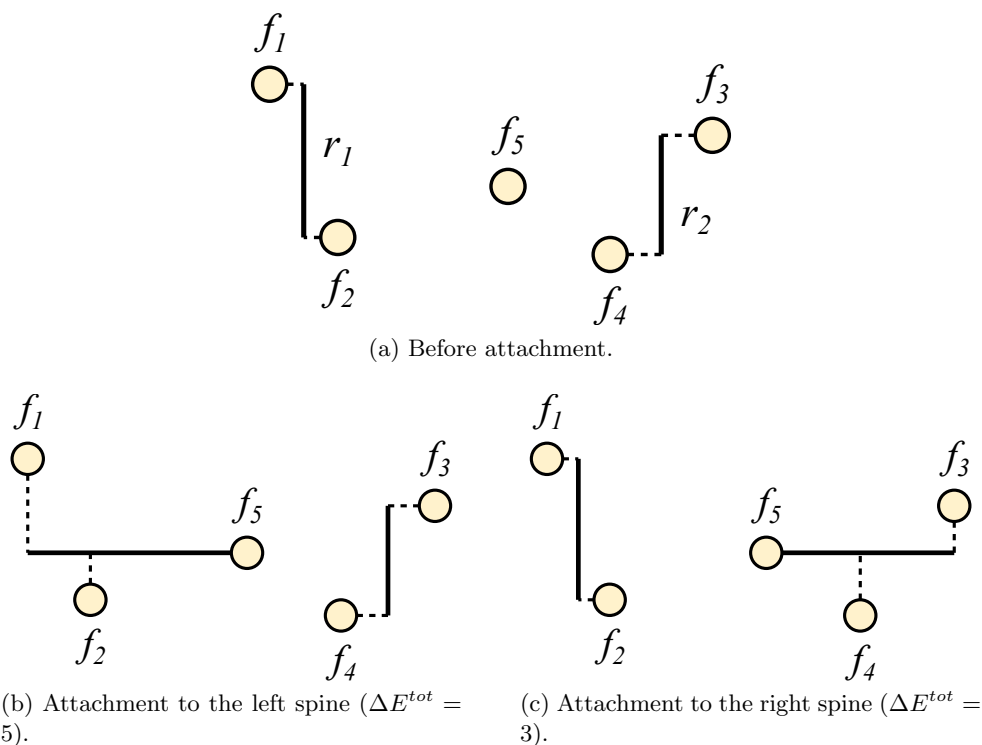
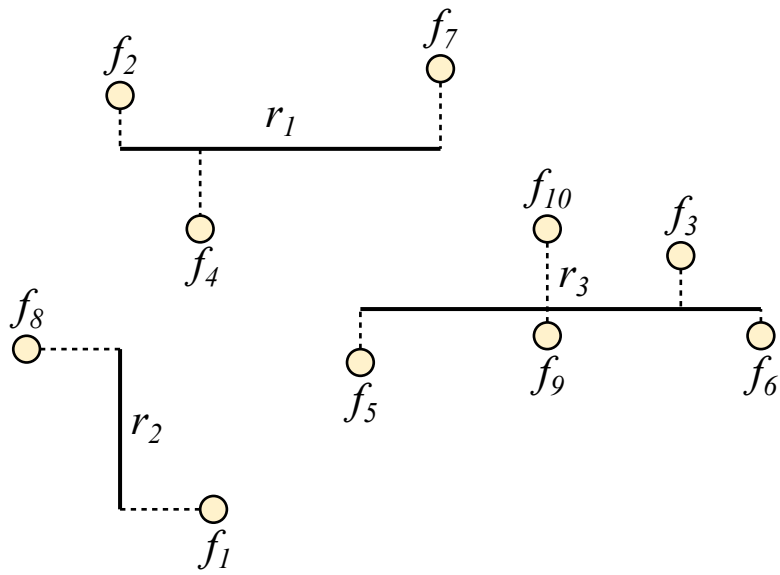


Figure 4.7: Two possible spines updated by attaching sink f_5 . (a) Before attachment. (b) Attachment to the left spine ($\Delta E^{tot} = 5$). (c) Attachment to the right spine ($\Delta E^{tot} = 3$).

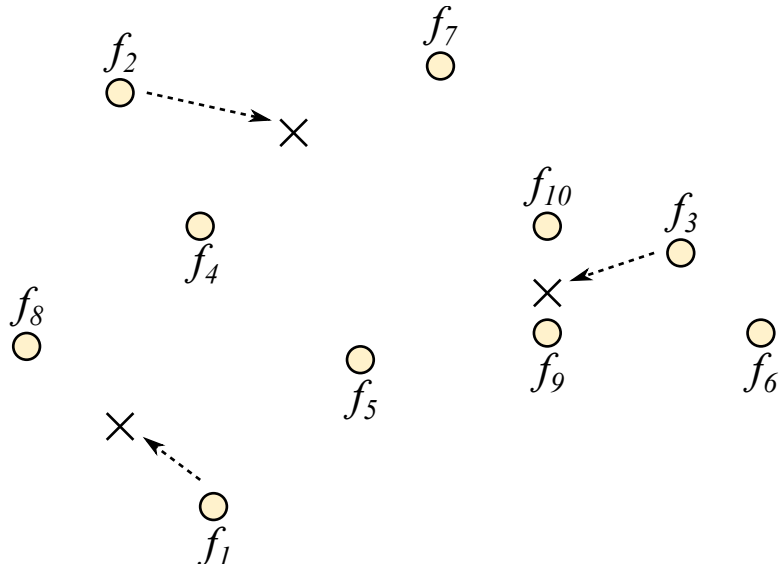
assigns f_5 to r_2 since $\Delta E^{tot}(f_5, r_1) = 3 < 5 (= \Delta E^{tot}(f_5, r_2))$.

3. CSPINE commits the actions 1 and 2 only if the attachment of a sink f_k does not require a power-minimal spine whose the longest length between the sinks exceeds L_{sink} or the spine length exceeds L_{spine} where L_{sink} and L_{spine} are threshold parameters to control the tolerance of clock skew. (The tolerance of clock slew will be maintained mainly by the spine buffer insertion step.) If no spine is able to meet the L_{sink} and L_{spine} bound constraints, the sink f_k itself becomes a new spine.

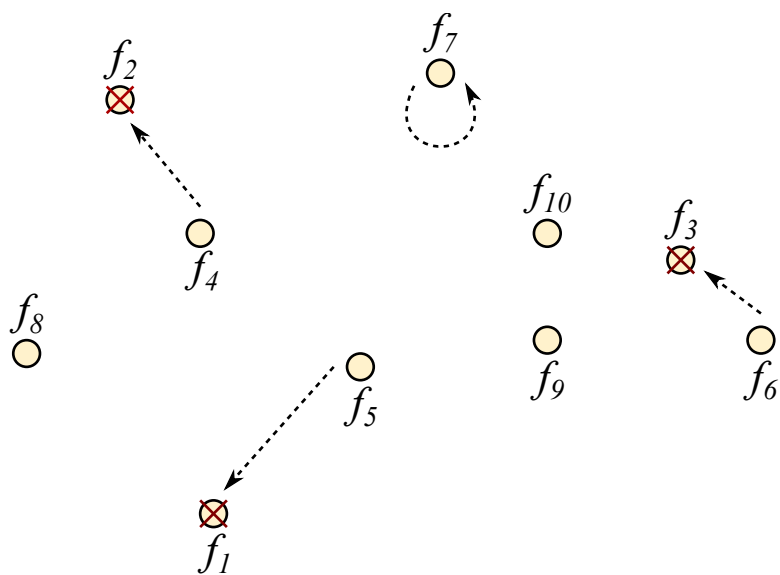
Fig. 4.8 illustrates the step-by-step procedure of sink clustering in CSPINE. Fig. 4.8(a) shows a spine \mathcal{S} produced by the prior iteration of the *outer-loop*,



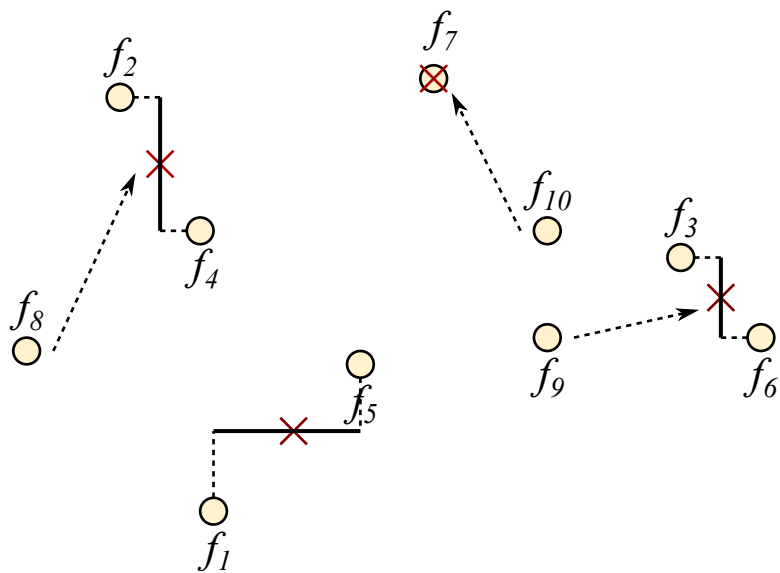
(a) Initial cluster ($E^{tot} = 40$).



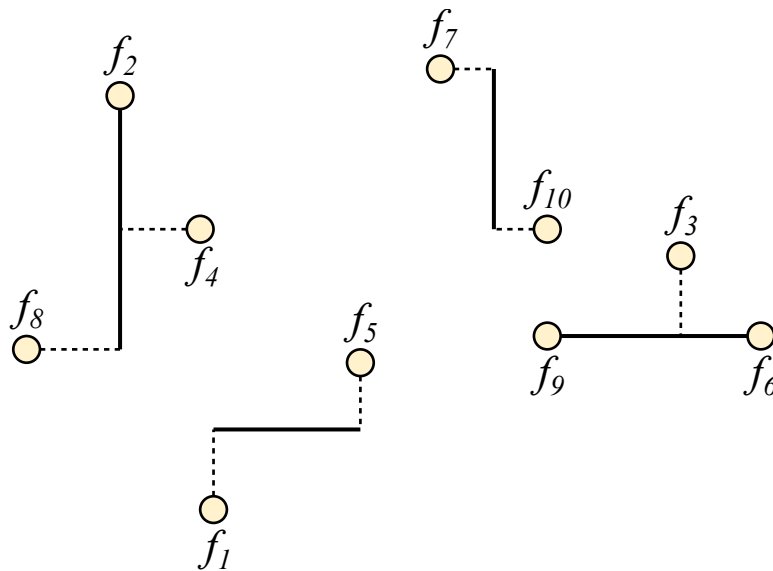
(b) $f_1 \sim f_3$ are attached to dummy spines.



(c) $f_4 \sim f_6$ are attached to non-dummy spines and f_7 becomes a new spine.



(d) $f_8 \sim f_{10}$ are attached to non-dummy spines.



(e) New clock spine structure has been generated ($E^{tot} = 31.8$).

Figure 4.8: An example illustrating the step-by-step procedure of the *inner-loop* of sink clustering in CSPINE. (a) A spine network produced from the prior *outer-loop*. (b) Generation of a *dummy* spine for each sink. (c) From the first four iterations, f_4 , f_5 , and f_6 are respectively attached to the dummy spines of f_2 , f_1 , and f_3 to form non-dummy spines while f_7 still maintains a dummy spine. (d) From the next three iterations, f_8 , f_9 , and f_{10} are attached to non-dummy spines. (e) The final clock spine network.

in which \mathcal{S} consists of three spines r_1 , r_2 and r_3 and total energy $E^{tot} = 40$. Then, Fig. 4.8(b) shows the dummy spines, marked by ‘ \times ’, and the revocation of sink attachment. Figs. 4.8(c) and (d) show the assignment of sinks $f_4 \sim f_{10}$ and its energy costs for the two spine shapes in Fig. 4.6 and 4.7. No spine is able to meet the L_{sink} and L_{spine} constraint for f_7 , it generates a new spine itself. Fig. 4.8(e) shows the clock spine structure produced by the *inner-loop*, in which the total energy is reduced from 40 to 31.8.

Note that the principle of our sink clustering algorithm is identical to the well-known *K-mean clustering* algorithm [55]. The key difference is that ours automatically determines the value of K (i.e., the number of spines) that minimizes the energy cost in *Eq.4.2* while the *K-mean clustering* assumes K to be a fixed number. The application of our sink clustering algorithm produces a clock spine with *spec-1*, *2*, *3*, and *4*. Step 2 will refine the structures of the spines obtained in this step to facilitate the derivation of *spec-5* and *6*, and Step 3 will near-optimally determine *spec-5* and *6* while meeting the clock skew and slew tolerance. Step 4 completes the clock spine synthesis by constructing a top-level clock tree to deliver clock signal to the clock buffers inserted to the spines in Step 3. The pseudo-code of our sink clustering algorithm is described in Algorithm. 1.

4.3.3 Spine Relaxation

In comparison with the clock mesh structure, the clock spine structure is not as strong as the clock mesh in mitigating clock skew variation. This is because every ‘spine’ is sticked to either a vertical or horizontal line and nothing else. This step relaxes the restriction of the spine shape, so that the potential cause of the clock skew violation can be removed.

For example, Fig. 4.9(a) shows a spine that attaches four sinks f_1 , f_2 , f_3 ,

Algorithm 1 *Power-aware sink clustering*

```
1: procedure CLUSTERING( $F$ )
2:   Construct initial spine structure;
3:    $i \leftarrow 0$ ;
4:   while  $i < I$  do ▷ outer-loop
5:     Calculate mean location  $(\bar{x}_i, \bar{y}_i)$ ;
6:     Nullify the sink attachment in  $S$ ;
7:     for  $f_k \in F$  do ▷ inner-loop
8:       Synthesize a power-minimal spine,  $r_i$ , under skew/slew tolerance
       constraint;
9:       if  $r_i \neq \text{Null}$  then
10:        Attach  $f_k$  to  $r_i$ ;
11:         $E^{tot} = E^{tot} + \Delta E^{tot}(f_k, r_i)$ ;
12:       else
13:        Generate a new spine  $r_i$ ;
14:         $E^{tot} = E^{tot} + \Delta E^{tot}(f_k, r_i)$ ;
15:       end if
16:     end for
17:     Save the spine structure of minimum  $E^{tot}$ ;
18:   end while
19:   return  $R$ ; ▷ A spine structure of minimum cost
20: end procedure
```

and f_4 . It is seen that according to the activity patterns of the sinks, the spine will be enabled in power modes m_1 , m_2 , and m_4 . Figs. 4.9(b), (c), and (d) show the sinks enabled in m_1 , m_2 , and m_4 , respectively. In view of the clock skew variability, the spine contributes to mitigating the potential clock skews between the sinks f_1 , f_2 , and f_4 in mode m_1 , as shown in Fig. 4.9(b) and between the sinks f_1 and f_3 in mode m_1 , as shown in Fig. 4.9(d). However, since in mode m_2 , only f_2 will be enabled, as shown in Fig. 4.9(c), the spine alone makes no contribution to reducing the potential clock skew associated with f_2 .

Definition 4.1 (Isolated sinks): *If a sink f_k on a spine r_i is the only one, among the sinks on r_i , to be enabled in a power mode, it is said to be an isolated sink of r_i .*

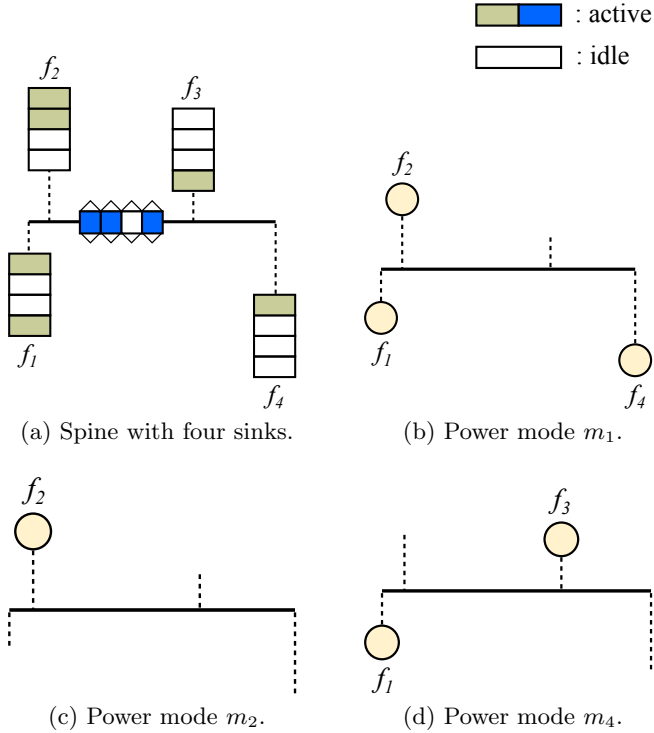


Figure 4.9: Example illustrating the concept of *isolated sink*. (a) A spine with four sinks with activation patterns. (b) The sinks to be enabled in power mode m_1 . (c) The sinks to be enabled in power mode m_2 . f_2 is an isolated sink in m_2 of the spine in (a). The spine alone has no control to mitigate the clock skew related to f_2 . (d) The sinks to be enabled in power mode m_3 .

Definition 4.2 (Isolation counts): $N_{iso_sk}(f_k)$ is defined to the number of distinct power modes in which sink f_k is an isolated sink and $N_{iso_spn}(r_i)$ is defined to the total sum of the values of $N_{iso_sk}(\cdot)$ for all sinks in spine r_i .

The *spine relaxation problem* we want to solve in this step is to relax the restriction of the spine shape so that multiple spines can be connected, if needed, to remove all the isolated sinks from the clock spine network. Our CSPINE applies a greedy approach for a spine network \mathcal{S} obtained from Step 1:

Step 2.1 Extract all isolated sinks for each spine in \mathcal{S} , and compute the values of

$N_{iso_sk}(f_k)$ and $N_{iso_spn}(r_i)$ for all sinks and spines in \mathcal{S} . Pick a spine, r_i , which has the largest value of $N_{iso_spn}(\cdot)$. If there are ties, select the one which has the larger maximum among the $N_{iso_sk}(\cdot)$ values.

Step 2.2 If no spine is extracted in Step 2.1, terminate the process. Otherwise, search a neighbor spine, r_j , that can be connected to the target spine r_i extracted in Step 2.1 with minimal cost of wire, and connect them.

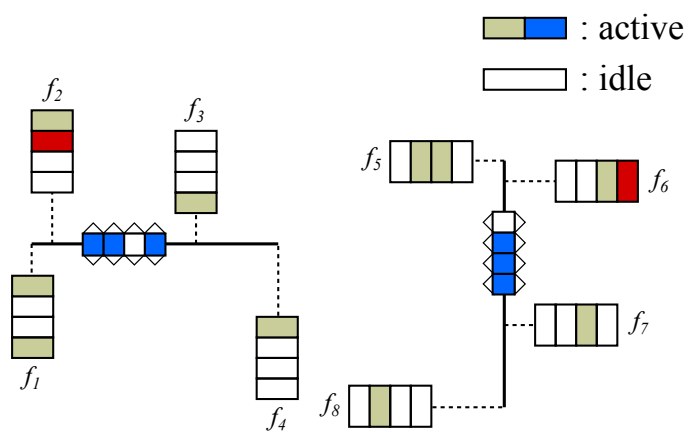
Step 2.3 Update the values of $N_{iso_sk}(\cdot)$ and $N_{iso_spn}(\cdot)$ values in r_i and r_j . Go to Step 2.1.

For example, Fig. 4.10 shows how two spines are *temporally* connected to mitigate the clock skew variation, in which the left and right two spines in Fig. 4.10(a) has isolated sinks f_2 in mode m_2 and f_6 in m_4 , respectively. The blue wire in Fig. 4.10(a) connects the two spines. The transmission gate switches on in modes m_2 and m_4 , which enables the two sinks f_2 and f_6 not to be isolated sinks any longer.

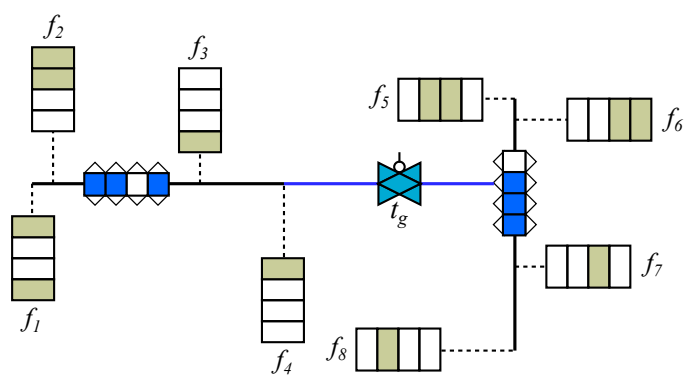
4.3.4 Spine Buffer Allocation

Placing a buffer on a location in a spine can drive a clock signal to a certain range. The range is determined the by driving strength by the buffer and the amount of the capacitance of the driven spine, stubs, and sinks. For a spine with sinks, the driving range of a buffer can be extracted. For example, the red dotted circle in Fig. 4.11(b) indicate the driving range of buffer b_{1x} in the circle; the buffer equally drives 30 strength-units to the left and right, driving 10 of the 30 in the left side to f_3 and its stub and 10 of the 30 in the right side to f_4 and its stub.

The **spine buffer allocation problem** can be stated as: *Given a buffer library \mathcal{L} , a spine r with the capacitances of unit-length spine and stub, and the*



(a) Two spines with isolated sinks.





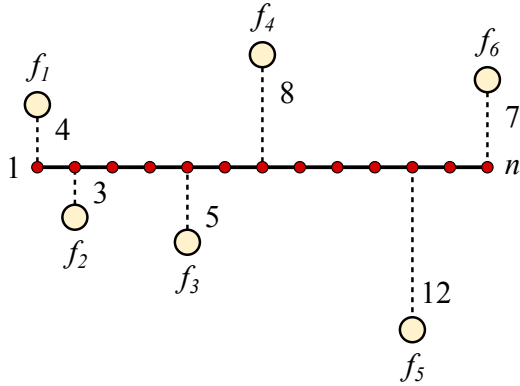
(b) Two spines are connected by transmission gate.

Figure 4.10: Example showing the spine structure relaxation for two spines to improve the clock skew variability by removing isolated sinks. (a) Before relaxation. (b) After relaxation by *temporally* connecting them. t_g is a transmission gate.

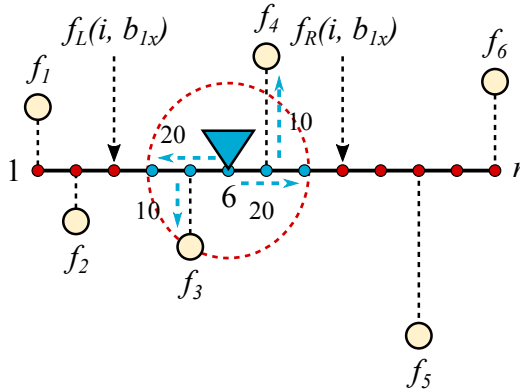
input load capacitances of the sinks on r , determine the locations on r and the types of buffers in \mathcal{L} to be placed on the locations so that all the capacitances of r , the stubs in r , and the inputs to sinks are covered by the allocated buffers while minimizing the total cost of buffers to be used.

Let us assume that the candidate spine positions at which buffers can be placed have already been determined by the designers, and the input capacitances of all sinks are the same, for simplicity. First, by using a spine example

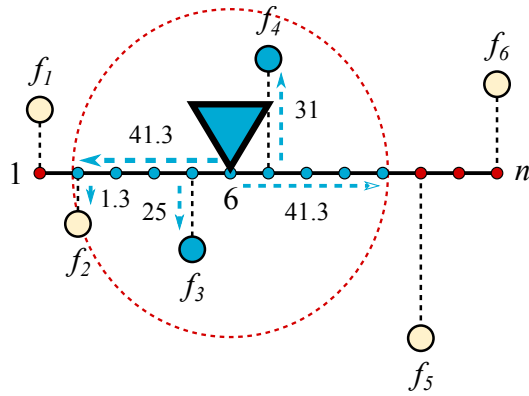
C_{sink} : 15	 : b_{1x} , 60, 6
C_{stub} : 2 /unit	 : b_{2x} , 140, 13
C_{spine} : 2 /unit	
l_{unit} : 5 unit	



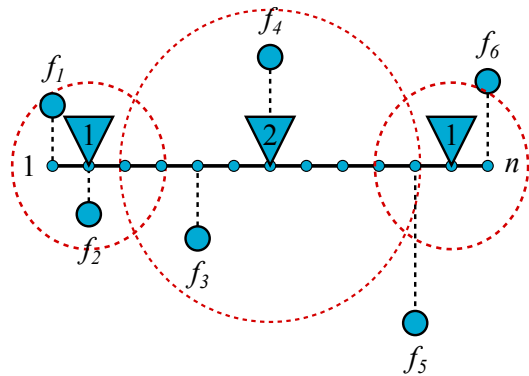
(a) A spine with 6 sinks and its buffer location candidate.



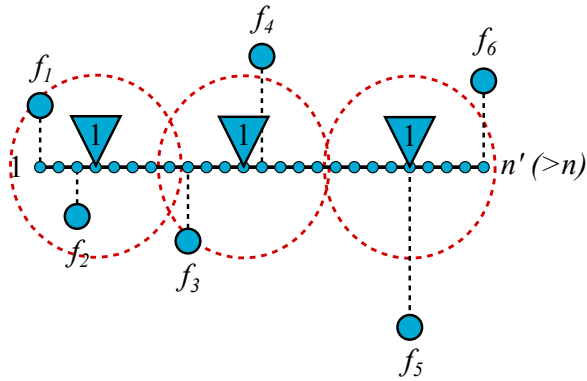
(b) Buffer type b_{1x} is inserted on point 6 and it covers only part of spine.



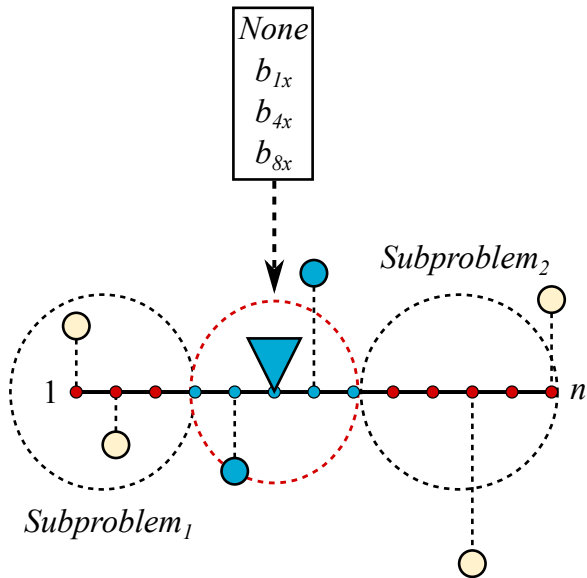
(c) Buffer type b_{2x} is inserted on point 6 and it covers sinks f_3 , f_4 and part of spine.



(d) Buffers are inserted, Total cost is 25.



(e) Buffers are inserted in more discretized spine, Total cost is 18.



(f) Dynamic programming based buffer insertion problem.

Figure 4.11: Example illustrating the application of dynamic programming for solving the problem of buffer allocation. (a) The specification of a spine with the unit-length capacitances of spine and stub, load capacitance of sinks, buffer library \mathcal{L} and spine position candidates. (b) Allocation of buffer b_{1x} to position 6, covering sub-spine $r[4, 8]$. (c) Allocation of buffer b_{2x} to position 6, covering a bigger sub-spine than that in (c). (d) An optimal allocation for the specification in (a). (e) An optimal allocation for the specification in (a) with more fine spine positions, producing a reduced cost of buffer allocation. (f) Decomposition of problem into two subproblems.

shown in Fig. 4.11, we illustrate how the spine buffer allocation can be recursively solved. Besides the spine description, Fig. 4.11(a) summarizes the unit-length capacitances, C_{spine} and C_{stub} , of spine and stub, the input load, C_{sink} , of a sink, and the driving strength of buffers in \mathcal{L} . Fig. 4.11(d) shows an optimal buffer allocation of the spine with capacitance data, a set of candidate positions, and \mathcal{L} in Fig. 4.11(a). Further, Fig. 4.11(e) shows an optimal buffer allocation of the same spine in Fig. 4.11(a), but different set of positions. Let $\{1, \dots, n\}$ be the spine position, from the left or the top, at which buffers can be placed, and $r[i, j]$ be the sub-spine of spine r from positions i to j on r . In addition, let $b_{opt}(i, j)$ be an optimal solution of buffer allocation for $r[i, j]$ and $C(b_{opt}(i, j))$ be its buffer cost. By updating $\mathcal{L}' = \mathcal{L} \cup \{\phi\}$ where ϕ means no buffering, an optimal buffer allocation for r can be recursively expressed as:

$$\begin{aligned}
C(b_{opt}(1, n)) = & \min\{C(b_{opt}(1, f_L(i, b_x))) \\
& + C(b_{opt}(f_R(i, b_x), n)) + C(b_{opt}(i, b_x)) \\
& + size(b_x), \forall b_x \in \mathcal{L}'\}
\end{aligned} \tag{4.3}$$

in which $f_L(i, b_x)$ and $f_R(i, b_x)$ represent the closest spine positions to the left (or to the top) and to the right (or to the bottom) that are not driven by b_x . (If $b_x = \phi$, $f_L(i, b_x) = i - 1$ and $f_R(i, b_x) = i$ and $area(\phi) = 0$.) (See Fig. 4.11(f) for an illustration of the problem decomposition)

The recursive expression can be solved by using dynamic programming. The termination condition is, for every $i \in \{1, 2, \dots, n\}$, $C(b_{opt}(i, i)) = area(b_{min})$ where $b_{min} \in \mathcal{L}$ is the minimum sized buffer that can drives the sum of load capacitance from the middle position of $i - 1$ and i to i and the middle position of i and $i + 1$ to i .

4.3.5 Top-Level Tree Construction

To complete the structure of clock spine, a top-level clock tree that drives the clock buffers placed on the spines in Step 3. We can use any clock tree synthesis algorithm for this job by treating the clock buffers with sink nodes. However, since a post-stage clock gating on the internal nodes of the top-level clock tree may further reduce the power consumption, a kind of clock-gating aware clock tree synthesis is preferred in this step. In this work, we use the power-aware clock tree synthesis algorithm PACTS proposed by Lu, Chow, and Sham [51].

4.4 Experimental Results

The proposed clock spine synthesis methodology CSPINE is implemented in C++ and Python3, and tested by HSPICE simulation. The simulation is run on Linux machine with 8 cores of 3.5GHz Intel i7 CPU and 16GB memory. The tested benchmark circuits are from ISCAS89 [22] and ISPD2010 [56]. Since there is no logical information in ISPD benchmarks, the activity pattern of clock sinks are randomly generated according to the method proposed in [46].

ISCAS89 testcases (s-series in Table 4.1) are synthesized with *Synopsys IC Compiler* with 45nm Nangate Open Cell Library [24]. Clock gating is applied to those circuits to extract activity patterns. The Elmore delay model are imported to simulate the circuit and unit resistance and unit capacitance of the wire are $0.1\Omega/\mu m$ and $0.2fF/\mu m$. The characteristics of buffers and gates are the same.

We compare the structure of clock spine network produced by our CSPINE with the structures of clock tree and clock mesh. Since there exists no clock tree algorithm combined with clock gating in the HSPICE simulation level, we generate clock trees in which we used the clock topology generated by the nearest neighbor algorithm in [57]. (Note that the previous power-aware clock tree

synthesis algorithms (e.g., [51]) performed SPICE level simulation. However, they considered skew values only.) For the comparison with the results of clock mesh, the synthesis algorithm in [58] is used for the generation of clock meshes. The clock meshes produced for comparison in the experiments have local trees from the meshes and clock gating cells are inserted at the roots of the local trees.

Table. 4.1 summarizes the comparison of the results for clock trees produced by [59] with clock gating, clock meshes produced by [58], subsequently applying clock gating, and clock spines produced by CSPINE. Skew results of clock spines are shown in the middle of that of clock trees and clock meshes. It is observed that the standard deviation of clock spine is closer to that of clock mesh than that of clock tree. Nevertheless, its power consumption is not much larger than that of tree.

Table. 4.2 shows the comparison of the results when the slew constraints of clock tree synthesis algorithm are $100fF$ and $200fF$ (The constraint of tree results in the Table. 4.1 is $300fF$). Average ratio is computed by setting the value of tree results in the Table. 4.1 as 1. The results show that the standard deviation of skew is decreased when the slew rate constraint is decreased. But, the number of inserted buffers is larger than that of clock spine structure while the amount of skew reduction is smaller than that of clock spine structure.

Table. 4.3 shows the comparison of the results when the number of allocated spines is changed. The standard deviation of skew is decreased when the number of spines is reduced. Due to transmission gates which are inserted by the spine relaxation, the power consumption is also decreased.

Fig. 4.12 shows the synthesized clock mesh and clock spine structures for circuit s13207. It is shown that clock spine structure consists of spines of various shapes and lengths while clock mesh consists of regular mesh wires. This flexible

Table 4.1: Comparison of the clock skew variability (μ, σ), wirelength and buffer area (WL, BA), and power consumption (PWR) of the clock trees with clock gating by [59], the clock mesh by [58] and a subsequent application of clock gating, and clock spines by CSPINE.

Circuit	# sinks	Tree				
		Skew (ps)		WL	BA	PWR
		μ	σ	(μm)	(μm^4)	(mW)
s1423	74	12.04	4.03	718.31	132.16	32.23
s13207	330	31.77	13.05	3294.29	446.04	125.37
s15850	134	9.9	5.19	1017.73	165.5	21.41
s38417	1564	50.84	6.78	17917.96	2864.73	866.11
s38584	1168	40.42	4.84	18707.4	1453.76	695.92
03	1200	58.43	9.59	79411.3	1329.86	846.68
04	1845	41.52	11.02	138833	1928.71	1140.65
05	1016	20.59	4.28	60647.1	929.25	541.98
06	981	97.04	34.31	56328.4	1474.41	355.68
Avg.ratio		1	1	1	1	1
Circuit	Grid	Mesh				
		Skew (ps)		WL	BA	PWR
		μ	σ	(μm)	(μm^2)	(mW)
s1423	3×3	5.47	1.32	2083.52	204.424	48.65
s13207	5×5	4.35	1.5	8444.79	1255.59	177.64
s15850	3×3	6.31	3.77	2036.12	422.44	27.87
s38417	6×6	6.6	2.06	39531.6	5551.74	1276.43
s38584	10×10	5.33	1.6	32076.1	4413.42	1051.13
03	22×12	27.93	2.93	118489	5064.17	2048.22
04	20×20	61.63	4.15	262776	8881.42	3562.25
05	20×20	51.58	4.78	181908	9056.57	1421.69
06	19×8	40.68	2.29	116317	2844.98	612.4
Avg.ratio		0.71	0.41	2.20	3.55	1.90

Circuit	Tree				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
s1423	12.04	4.03	718.31	132.16	32.23
s13207	31.77	13.05	3294.29	446.04	125.37
s15850	9.9	5.19	1017.73	165.5	21.41
s38417	50.84	6.78	17917.96	2864.73	866.11
s38584	40.42	4.84	18707.4	1453.76	695.92
03	58.43	9.59	79411.3	1329.86	846.68
04	41.52	11.02	138833	1928.71	1140.65
05	20.59	4.28	60647.1	929.25	541.98
06	97.04	34.31	56328.4	1474.41	355.68
Avg.ratio	1	1	1	1	1
Circuit	Spine				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
s1423	6.31	1.51	929.97	176.29	35.99
s13207	8.53	2.15	3577.96	745.236	139.78
s15850	8.22	4.74	1567.39	290.156	22.53
s38417	13.02	3.76	16762.6	3337.47	968.43
s38584	12.49	2.77	13555.5	2502.56	751.9
03	19.7	4.38	66521.3	1910.41	1150.78
04	24.33	5.61	124829	2791.77	1519.9
05	33.68	2.75	89497.7	1689.6	437.76
06	25.27	4.43	58119.5	1566.06	437.34
Avg.ratio	0.56	0.48	1.09	1.49	1.13

Table 4.2: Comparison of the clock skew variability (μ, σ), wirelength and buffer area (WL, BA), and power consumption (PWR) of the clock trees with clock gating by [59] when its slew rate constraints are changed.

Circuit	Tree (slew= $200fF$)				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
s1423	20.78	2.41	716.308	177.59	37.73
s13207	32.63	2.56	3434.97	578.2	135.2
s15850	9.9	1.97	1017.73	165.2	21.41
s38417	44.3	3.7	25763	2606.03	999.6
s38584	46.21	2.71	18330.6	1932.84	770.24
03	59.99	5.23	80709.5	1618.96	928.59
04	35.66	7.38	152250	2535.82	1390.88
05	20.57	4.22	66746.4	1379.42	335.79
06	99.25	7.04	55977.7	1734.6	388.81
Avg.ratio	1.08	0.78	1.05	1.19	1.10
Circuit	Tree (slew= $100fF$)				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
s1423	29.55	1.71	752.103	272.58	46.06
s13207	42.98	2.95	3546.64	1015.98	171.1
s15850	37.27	2.73	1096.16	408.87	28.62
s38417	59.72	3.36	26596.4	4621.47	1199.46
s38584	57.28	4.06	19459.1	3452.68	936.08
03	53.65	3.23	78527.7	2734.06	1229.55
04	46.83	6.45	139071	3745.91	1648.73
05	34.32	6.17	69396	1829.59	402.71
06	92.18	8.07	68872.7	2717.54	542.27
Avg.ratio	1.64	0.83	1.10	2.02	1.39

Table 4.3: Comparison of the clock skew variability (μ, σ), wirelength and buffer area (WL, BA), and power consumption (PWR) of the clock spines by CSPINE.

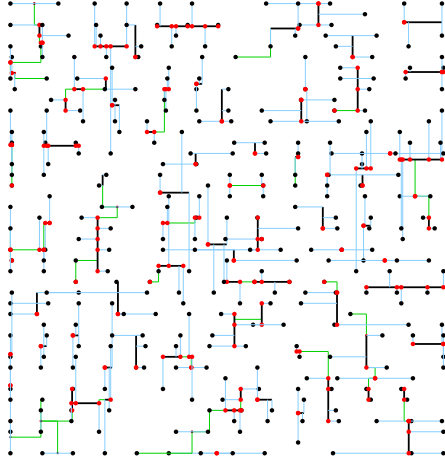
Circuit	# spines	Skew (ps)		WL	BA	PWR
		μ	σ	(μm)	(μm^2)	(mW)
s13207	62	9.72	2.12	3063.44	701.326	127.94
	31	5.33	1.42	2966.35	614.227	115.98
	8	2.63	1.53	3250.54	602.905	115.48
s15850	18	7.85	1.05	1384.77	257.226	20.8
	13	3.6	0.9	1453.87	248.917	20.01
	4	0.5	0.66	1348.93	233.994	19.06
s38417	271	14.2	2.22	19361.1	3235.82	944.49
	85	11.06	3.3	18121.6	2945.59	878.37
	37	20.57	1.85	31058.5	3003.99	951.04
s38584	196	14.27	2.09	15455.1	2329.31	717.19
	107	4.25	1.53	13309.8	2118.59	648.99
	30	4.37	1.52	13309.8	2118.59	649.14

structure of clock spine network decreases the resource usage significantly as well as decreases power dissipation.

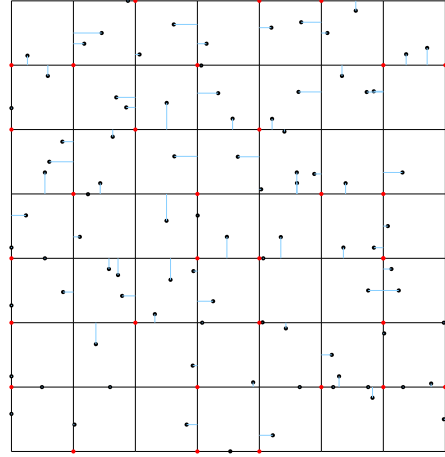
Fig. 4.13 shows the skew and power consumption for circuit 04 for all power modes. In most cases, the clock spine network produced by CSPINE uses smaller skew than that of clock tree network. Further, the clock gating on clock spine network decreases a significant amount of power consumption compared to the clock mesh network. For all tested results, it is clear that the clock spine structure is outstanding for achieving the clock skew tolerance, clock power saving, and clock wirelength reduction under multi-power mode designs, in comparison with the clock mesh and clock tree structures.

4.5 Summary

This work addressed the problem of automating the synthesis of power-aware (clock gated) clock spine networks, which has been semi-automated or never been automated in academia and industry so far, even though the usefulness and



(a) Clock spine: $\mu = 8.53$, $\sigma = 2.15$, WL = 3577.96, BA = 745.236, PWR = 139.78



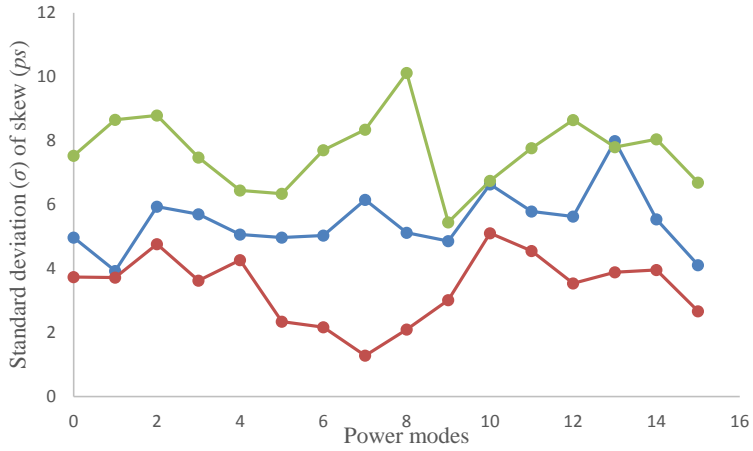
(b) Clock mesh: $\mu = 4.35$, $\sigma = 1.5$, WL = 8444.79, BA = 1255.59, PWR = 177.64

Figure 4.12: Comparison of clock spine and mesh structures of s13207. In (a) black dots, red dots, black lines, and green lines represent sinks, buffers, spines, and connection between spines. In (b) sky blue lines represent stubs.

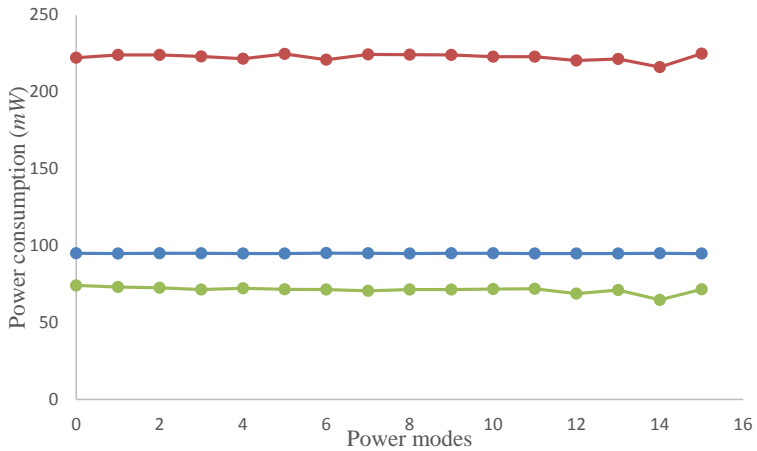
Table 4.4: The analyzed characteristics of clock spine.

	Clock tree	Cross links	Clock spine	Clock mesh
Performance	Low	Medium	High	Very high
Cost	Very low	Medium	Low	High
Analysis	Simple	Less simple	Less Complex	Complex

impact have been clearly noticed for a long time. In this chapter, we solved the problem by developing four solid synthesis steps, in which the key task was to identify and group the flip-flops with tight correlation of clock-gating operations together to form a spine while accurately predicting and maintaining clock skew and slew variations through the buffer insertion and stub allocation. Through experiments with benchmark circuits, it was confirmed that our power-aware synthesis for clock spines used significantly less power consumption compared to that of the conventional clock mesh synthesis algorithm at the expense of a little relaxed or the same constraint of clock skew. Together with the structure



(a) Standard deviation of skew of 04 in each power mode.



(b) Power consumption of 04 in each power mode.

Figure 4.13: Comparison of clock tree, clock mesh, and clock spine structures of circuit 04 for all power modes. Green lines, blue lines, and red lines represent clock tree, clock mesh, and clock spine networks, respectively.

of clock tree, our automation of synthesizing clock spines would provide a fast and diverse exploration of the structures of hybrid clock networks to trade-off between the clock skew/slew and power consumption.

Chapter 5

CONCLUSION

The contributions of this dissertation are summarized as follows:

5.1 Chapter 2

In the chapter, a comprehensive graph-based algorithm for PST buffer allocation to overcome the critical limitations of the prior works. The proposed PST-alloc explores PST buffer allocations systematically and finds minimized number of PST buffer allocation. Experimental results through benchmark designs show that the proposed PST-alloc uses 26% less number of PST buffers than that by the previous works while meeting the same yield constraint.

5.2 Chapter 3

In the chapter, we proposed a careful clock skew tuning technique that exploited the inter-dependent time relation and able to incrementally and effectively increase the time margins of circuits. The effectiveness of the CSS-FT is shown through experiments with benchmark circuits, demonstrating that our method

relaxes the worst slack of circuits, so that the clock period (T_{clk}) is shortened by 4.2% on average, namely the clock speed is improved from 369MHz~2.23GHz to 385MHz~2.33GHz with no time violation. In addition, it reduces the total numbers of setup and hold time violations by 27.7%, 9.5%, and 6.7% when the clock periods are set to 95%, 90%, and 85% of the value of T_{clk} , respectively.

5.3 Chapter 4

The chapter shows four solid synthesis steps to synthesize power-aware clock spine networks, and the produced clock spine network minimize the energy consumption while satisfying clock skew and slew constraints. Through experiments with benchmark circuits, it is shown our power-aware synthesis for clock spines uses significantly 60% less power consumption compared to that of the conventional clock mesh synthesis algorithm at the expense of 17% relaxed constraint of clock skew.

Bibliography

- [1] P. Gronowski, W. Bowhill, R. Preston, M. Gowan, and R. Allmon, “High-performance microprocessor design,” *IEEE Journal of Solid-State Circuits*, vol. 33, no. 5, pp. 676–686, May 1998.
- [2] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, “Reducing power in high-performance microprocessors,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 1998, pp. 732–737.
- [3] Y. Liu, S. Nassif, L. Pileggi, and A. Strojwas, “Impact of interconnect variations on the clock skew of a gigahertz microprocessor,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 2000, pp. 168–171.
- [4] H. Chang and S. Sapatnekar, “Statistical timing analysis considering spatial correlations using a single pert-like traversal,” in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, November 2003, pp. 621–625.
- [5] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, S. Narayan, D. Beece, J. Piaget, N. Venkateswaran, and J. Hemmett, “First-order incremental block-based statistical timing analysis,” *IEEE Transactions on*

- Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2170–2180, October 2006.
- [6] Z. Feng, P. Li, and Y. Zhan, “Fast second-order statistical static timing analysis using parameter dimension reduction,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 2007, pp. 244–249.
- [7] S. Rusu and S. Tam, “Clock generation and distribution for the first IA-64 microprocessor,” in *Proceedings of IEEE International Solid-State Circuits Conference*, February 2000, pp. 176–177.
- [8] P. J. Restle, T. G. Mcnamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, S. Member, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. Mccredie, “A clock distribution network for microprocessors,” *IEEE Journal of Solid-State Circuits*, vol. 36, pp. 792–799, 2001.
- [9] D.-J. Lee and I. Markov, “Multilevel tree fusion for robust clock networks,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2011, pp. 632–639.
- [10] C. kok Koh, J. Jain, and S. F. Cauley, *Synthesis of clock and power/ground networks*, In L.-T. Wang, Y.-W. Chang, and K.-T. Cheng, editors, *Electronic Design Automation: Synthesis, Verification, and Test*, chapter 13. Morgan Kauffman, 2009.
- [11] S. Tam, S. Rusu, U. N. Desai, R. Kim, J. Zhang, and I. Young, “Clock generation and distribution for the first IA-64 microprocessor,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1545–1552, November 2000.

- [12] J.-L. Tsai, D. H. Baik, C.-P. Chen, and K. Saluja, “A yield improvement methodology using pre- and post-silicon statistical clock scheduling,” in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, November 2004, pp. 611–618.
- [13] J.-L. Tsai, L. Zhang, and C. Chen, “Statistical timing analysis driven post-silicon-tunable clock-tree synthesis,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2005, pp. 575–581.
- [14] V. Khandelwal and A. Srivastava, “Variability-driven formulation for simultaneous gate sizing and postsilicon tunability allocation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 4, pp. 610–620, April 2008.
- [15] K. Nagaraj and S. Kundu, “A study on placement of post silicon clock tuning buffers for mitigating impact of process variation,” in *Proceedings of Design, Automation Test in Europe Conference Exhibition*, April 2009, pp. 292–295.
- [16] J. Kim, D. Joo, and T. Kim, “An optimal algorithm of adjustable delay buffer insertion for solving clock skew variation problem,” in *Proceedings of ACM/IEEE Design Automation Conference*, May 2013, pp. 1–6.
- [17] Y.-S. Su, W.-K. Hon, C.-C. Yang, S.-C. Chang, and Y.-J. Chang, “Value assignment of adjustable delay buffers for clock skew minimization in multi-voltage mode designs,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2009, pp. 535–538.
- [18] —, “Clock skew minimization in multi-voltage mode designs using adjustable delay buffers,” *IEEE Transactions on Computer-Aided Design of*

Integrated Circuits and Systems, vol. 29, no. 12, pp. 1921–1930, December 2010.

- [19] K.-H. Lim and T. Kim, “An optimal algorithm for allocation, placement, and delay assignment of adjustable delay buffers for clock skew minimization in multi-voltage mode designs,” in *Proceedings of IEEE Asia and South Pacific Design Automation Conference*, January 2011, pp. 503–508.
- [20] B. Li, N. Chen, and U. Schlichtmann, “Fast statistical timing analysis for circuits with post-silicon tunable clock buffers,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2011, pp. 111–117.
- [21] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.
- [22] F. Brglez, D. Bryan, and K. Kozminski, “Combinational profiles of sequential benchmark circuits,” in *Proceedings of IEEE International Symposium on Circuits and Systems*, May 1989, pp. 1929–1934 vol.3.
- [23] F. Corno, M. Reorda, and G. Squillero, “RT-level ITC’99 benchmarks and first ATPG results,” *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, July 2000.
- [24] “Nangate 45nm open cell library,” <http://www.nangate.com/>.
- [25] T. Susa, M. Murakawa, E. Takahashi, T. Furuya, T. Higuchi, “Post-Fabrication Clock-Timing Adjustment for Digital LSIs Ensuring Operational Timing Margins,” in *Proceedings of International Conference on Hybrid Intelligent Systems*, September 2008, pp. 907–910.

- [26] K. Nagaraj and S. Kundu, “An Automatic Post Silicon Clock Tuning System for Improving System Performance based on Tester Measurements,” in *Proceedings of IEEE International Test Conference*, October 2008, pp. 1–8.
- [27] D. Tadesse, J. Grodstein, and R.I. Bahar, “AutoRex: An automated post-silicon clock tuning tool,” in *Proceedings of IEEE International Test Conference*, November 2009, pp. 1–10.
- [28] Z. Lak and N. Nicolici, “A Novel Algorithmic Approach to Aid Post-Silicon Delay Measurement and Clock Tuning,” *IEEE Transactions on Computers*, vol. 63, no. 5, pp. 1074–1084, May 2014.
- [29] T. Okumura and M. Hashimoto, “Setup time, hold time and clock-to-Q delay computation under dynamic supply noise,” in *Proceedings of IEEE Custom Integrated Circuits Conference*, September 2010, pp. 1–4.
- [30] G. Rao and E. Howick, “Apparatus for optimized constraint characterization with degradation options and associated methods,” U.S. Patent App. 10/465,123. November 24, 2003.
- [31] S. Srivastava and J. Roychowdhury, “Interdependent latch setup/hold time characterization via Euler-Newton curve tracing on state-transition equations,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 2007, pp. 136–141.
- [32] —, “Independent and interdependent latch setup/hold time characterization via Newton-Raphson solution and Euler curve tracking of state-transition equations,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 817–830, May 2008.

- [33] E. Salman, A. Dasdan, F. Taraporevala, K. Kucukcakar, and E. Friedman, “Pessimism reduction in static timing analysis using interdependent setup and hold times,” in *Proceedings of IEEE International Symposium on Quality Electronic Design*, March 2006, pp. 159–164.
- [34] E. Salman and E. Friedman, “Utilizing interdependent timing constraints to enhance robustness in synchronous circuits,” *Microelectronics Journal*, vol. 43, no. 2, pp. 119–127, February 2012.
- [35] E. Salman, A. Dasdan, F. Taraporevala, K. Kucukcakar, and E. Friedman, “Exploiting setup–hold-time interdependence in static timing analysis,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 6, pp. 1114–1125, June 2007.
- [36] S. Hatami, H. Abrishami, and M. Pedram, “Statistical timing analysis of flip-flops considering codependent setup and hold times,” in *Proceedings of ACM Great Lakes Symposium on VLSI*, May 2008, pp. 101–106.
- [37] 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, September 2012.
- [38] A. Kahng and H. Lee, “Timing margin recovery with flexible flip-flop timing model,” in *Proceedings of IEEE International Symposium on Quality Electronic Design*, March 2014, pp. 496–503.
- [39] K.-H. Ho, X.-W. Shih, and J.-H. Jiang, “Clock rescheduling for timing engineering change orders,” in *Proceedings of IEEE Asia and South Pacific Design Automation Conference*, January 2012, pp. 517–522.

- [40] MATLAB, *version 8.3.0.532 (R2014a)*. Natick, Massachusetts: The MathWorks Inc., 2014.
- [41] MOSEK ApS, “The MOSEK C optimizer API manual Version 7.1 (Revision 32),” <http://docs.mosek.com/7.1/capi/index.html>, 2015.
- [42] M. Donno, A. Ivaldi, L. Benini, and E. Macii, “Clock-tree power optimization based on RTL clock-gating,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 2003, pp. 622–627.
- [43] Y. Luo, J. Yu, J. Yang, and L. Bhuyan, “Low power network processor design using clock gating,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 2005, pp. 712–715.
- [44] C.-M. Chang, S.-H. Huang, Y.-K. Ho, J.-Z. Lin, H.-P. Wang, and Y.-S. Lu, “Type-matching clock tree for zero skew clock gating,” in *Proceedings of ACM/IEEE Design Automation Conference*, June 2008, pp. 714–719.
- [45] D. Garrett, M. Stan, and A. Dean, “Challenges in clockgating for a low power ASIC methodology,” in *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, August 1999, pp. 176–181.
- [46] A. Farrahi, C. Chen, A. Srivastava, G. Tellez, and M. Sarrafzadeh, “Activity-driven clock design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 6, pp. 705–714, June 2001.
- [47] C. Chen, C. Kang, and M. Sarrafzadeh, “Activity-sensitive clock tree construction for low power,” in *Proceedings of IEEE International Symposium on Low Power Electronics and Design*, August 2002, pp. 279–282.

- [48] J. Oh and M. Pedram, “Gated clock routing minimizing the switched capacitance,” in *Proceedings of Design, Automation and Test in Europe*, February 1998, pp. 692–697.
- [49] —, “Gated clock routing for low-power microprocessor design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 6, pp. 715–722, June 2001.
- [50] W.-C. Chao and W.-K. Mak, “Low-power gated and buffered clock network construction,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 13, no. 1, pp. 20:1–20:20, February. 2008.
- [51] J. Lu, W.-K. Chow, and C.-W. Sham, “Fast power- and slew-aware gated clock tree synthesis,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 11, pp. 2094–2103, November 2012.
- [52] J. Lu, X. Mao, and B. Taskin, “Clock mesh synthesis with gated local trees and activity driven register clustering,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 2012, pp. 691–697.
- [53] Synopsys, *IC Compiler Implementation User Guide Version J-2014.09-SP4*, 2014.
- [54] G. Tellez, A. Farrahi, and M. Sarrafzadeh, “Activity-driven clock design for low power circuits,” in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, November 1995, pp. 62–65.
- [55] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of Berkeley Symposium on Mathematical*

Statistics and Probability, Volume 1: Statistics. Berkeley, Calif.: University of California Press, 1967, pp. 281–297.

- [56] C. N. Sze, “ISPD 2010 high performance clock network synthesis contest: Benchmark suite and results,” in *Proceedings of International Symposium on Physical Design*, March 2010, pp. 143–143.
- [57] M. Edahiro, “A clustering-based optimization algorithm in zero-skew routings,” in *Proceedings of Conference on Design Automation*, June 1993, pp. 612–616.
- [58] G. Venkataraman, Z. Feng, J. Hu, and P. Li, “Combinatorial algorithms for fast clock mesh optimization,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 1, pp. 131–141, January 2010.
- [59] T.-Y. Kim and T. Kim, “Clock tree embedding for 3D ICs,” in *Proceedings of Asia and South Pacific Design Automation Conference*, January 2010, pp. 486–491.

초록

오늘날의 회로 설계에서 공정변이가 회로 클럭의 타이밍의 변이에 미치는 영향은 매우 커짐에 따라, 전통적으로 사용되던 클럭 트리 구조를 기반으로 한 클럭 네트워크를 사용하는 것은 한계에 부딪히게 되었고, 이를 극복하기 위한 여러가지 기술들이 제안되었다. 본 논문에서는 변이에 강한 클럭 네트워크를 설계하기 위해, 연구 및 사용되고 있는 세 가지 기술에 대해 소개하고, 이들을 개선한 연구들을 제안한다.

첫째로, 이 논문에서는 클럭의 타이밍 문제를 회로 제작 이후 단계에서 조정할 수 있는 포스트 실리콘 조정 클럭 버퍼를 배치하는 문제에 대해 서술한다. 포스트 실리콘 조정 버퍼는 클럭의 지연시간을 회로가 제작된 이후의 단계에서 조정하여 클럭의 타이밍 문제를 해결할 수 있지만, 버퍼 자체의 크기 때문에 최소한의 개수만 가장 효율적인 위치에 배치해야 하는 문제가 있다. 본 논문에서는 이전의 연구가 회로의 수율을 계산할 때 시간이 많이 걸리는 몬테-카를로 시뮬레이션을 사용하기 때문에 탐색 가능한 포스트 실리콘 조정 버퍼의 배치가 제한되는 문제가 있음을 지적한 후, 기존에 제안되었던 그래프 기반 회로 수율 계산 기법을 사용하여 효율적인 포스트 실리콘 조정 버퍼 배치를 찾을 수 있는 점진적이고 체계적인 방법을 제시한다.

다음은 클럭 시차 스케줄링 방법에 대한 연구를 서술한다. 최근의 연구에서 제안되었던, 플립-플롭의 클럭에서 출력까지의 딜레이가 클럭의 준비시간과 유지시간에 의존한다는 유연한 플립-플롭 타이밍 모델 연구는 기존의 플립-플롭의 타이밍 특성들이 고정된 값이라는 가정에 기반한 정적 타이밍 분석의 정확성 문제를 해결할 수 있는 중요한 연구이다. 본 논문에서는 새로운 모델을 고려하여, 이전에 고전적인 플립-플롭 타이밍 특성 모델을 기반으로 진행되었던 클럭 시차 스케줄링의 최적화 문제를 유연한 플립-플롭 타이밍 모델을 고려하여 해결하였

다. 본 연구에서는 주어진 회로의 준비시간과 유지시간의 여유시간을 반복적이고 체계적으로 최대화하여 문제를 해결하였다.

마지막으로 클럭 스파인 네트워크의 합성을 자동화하는 문제에 대해 서술한다. 전통적인 클럭 트리 구조가 공정변이 문제를 해결하지 못했기 때문에 클럭 메쉬를 포함하는 다양한 대안적 구조가 제안되었다. 클럭 메쉬의 경우 공정변이에 의한 클럭 시차를 줄일 수 있었지만 이를 위해 와이어나 버퍼 등의 자원을 많이 소모하는 문제를 가지고 있다. 두 구조의 중간적 구조에는 클럭 트리의 노드를 연결하는 크로스 링크를 삽입하는 구조와 클럭 스파인 구조가 있다. 클럭 트리에 점진적인 수정을 가하여 만드는 크로스 링크와 달리, 클럭 스파인 구조는 트리나 이후에 제안된 메쉬와는 완전히 별개의 구조로, 이를 합성하는 방법도 매우 다르다. 그렇기 때문에 클럭 스파인을 합성하는 알고리즘은 필수적이라고 할 수 있으나, 합성 방법론이나 이를 자동화하는 방법에 관한 연구는 아직 없다. 본 논문에서는 우선, 클럭-게이팅을 지원하는 클럭 스파인을 주어진 클럭 시차 및 클럭 슬루 조건을 만족하면서 자원 및 전력 소모량을 최소화하는 문제에 대해 서술한다. 그리고, 회로에서 주어진 플립-플롭들을 클럭-게이팅 조건에서의 연관성을 고려하고 조직화하여 클럭 스파인을 삽입한 후, 클럭 시차 및 슬루 조건을 고려하여 버퍼를 삽입하는 알고리즘을 제안한다.

요약하면, 본 논문에서는 클럭의 타이밍 문제를 해결하기 위해 포스트-실리콘 조정 클럭 버퍼를 사용하는 테크닉과 클럭 시차 스케줄링을 유연한 플립-플롭 타이밍 모델에서 적용하는 테크닉을 제시하고, 클럭의 타이밍 문제와 전력 소모 문제를 한번에 해결하기 위한 새로운 클럭 스파인 네트워크를 합성하는 자동화 알고리즘을 제시한다.

주요어: VLSI&CAD, 포스트 실리콘 조정 클럭 버퍼, 정적 타이밍 분석, 유연한 플립-플롭 타이밍 모델, 클럭 스파인 합성

학번: 2011-30237