Ph.D. DISSERTATION

METHODOLOGY FOR SYNTHESIZING CLOCK SPINE NETWORKS

클락 스파인 네트워크 합성 방법론

BY

Youngchan Kim

February 2018

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

Ph.D. DISSERTATION

METHODOLOGY FOR SYNTHESIZING CLOCK SPINE NETWORKS

클락 스파인 네트워크 합성 방법론

BY

Youngchan Kim

February 2018

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

METHODOLOGY FOR SYNTHESIZING CLOCK SPINE NETWORKS

클락 스파인 네트워크 합성 방법론

지도교수 김 태 환

이 논문을 공학박사 학위논문으로 제출함

2017 년 12 월

서울대학교 대학원

전기 컴퓨터 공학부

김 영 찬

김 영 찬의 공학박사 학위논문을 인준함

2018 년 1 월

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

Abstract

Clock distribution network is used to deliver a clock signal from clock source to clock sinks (flip-flops and latches) in synchronous digital systems. However, the process variation caused by the CMOS process technology scaling increases the global clock skew. This reduces the clock speed of circuits, often causing a circuit failure. Consequently, it is essential to minimize the clock skew or control the clock skew in a bound during the synthesis of clock networks. To mitigate the clock skew induced by process variation, clock mesh network is investigated. However, though the clock mesh network provides a high variation tolerance, the clock resource and power consumption on the mesh is unacceptably high. To compromise the clock resource with clock skew variation, the clock spine network can be used as an alternative. But there is not much works which addressed the clock spine network synthesis.

This dissertation addresses the problem of developing a synthesis method for clock spine networks, which is able to systematically explore the clock resource and clock skew variability. The main idea is to transform the problem of allocating and placing clock spines on a plane into a slicing floorplan optimization problem, in which every candidate of clock spine network structures is uniquely expressed into a postfix notation to enable a fast cost computation in the slicing floorplan optimization. Recursive bipartition method and tolerance metric for spine networks are proposed to reduce clock resource consumption while keeping tolerance to variation of clock spine network at a certain level. In addition, to explore the various types of clock spine structure, methodology for synthesizing crossed clock spines is proposed as well. With crossed clock spine

structure, clock skew, clock resource usage, and power consumption of clock networks can be controlled. Finally, a clock spine synthesis method that supports clock gating at spine level is proposed. Experimental results demonstrate that our proposed method successfully further reduces the clock skew, clock resource, and power consumption over the networks produced by the previous work.

Keywords: Clock network, clock spine, clock skew, delay variation

Student Number: 2013-30224

Contents

Abstract **i**

Chapter 1 Introduction **1**

1.1 Clock Distribution Network 1

1.2 The Effect of Process Variations on Clock Skew 2

1.3 Clock Network Topologies 5

1.4 Microprocessor Clock Distributions 8

1.5 Contributions of This Dissertation 16

Chapter 2 Algorithm for Synthesizing Clock Spine Networks **18**

2.1 Introduction 18

2.2 Preliminaries and Motivation 20

2.2.1 Proposed Clock Spine Structure 20

2.2.2 The benefits of shorting intermediate stages in reducing clock skew 20

2.2.3 Smoothing Effect of Clock Spine 25

2.3 Overall Flow 27

2.4 Determination of the Number of Clock Spines 28

2.5 Transformation of Clock Spine Allocation and Placement Problem 29

2.5.1 Clock Spine Allocation and Placement Problem 29
2.5.2 Slicing Floorplan Optimization Problem 30
2.5.3 Single-line Clock Spine Allocation and Placement Problem 31
2.5.4 Other Considerations for Single Clock Spine Allocation and Placement 37
2.6 Application of Slicing Floorplan Optimization Algorithm 40
2.6.1 Move Operations 40
2.6.2 Rules for Legal Postfix Expressions 43
2.6.3 Cost Function for Simulated Annealing 46
2.6.4 The Starting Values of N, m, n 46
2.7 Derivation of Solution for Clock Spine Network 46
2.8 Spine-based Recursive Bipartition 47
2.9 Refinement of Clock Spine Network 47
2.10 Experimental Results 49
2.10.1 Experimental Environments 49
2.10.2 Comparison with Clock Tree Structure 51
2.10.3 Comparison with Clock Mesh Structure 54
2.10.4 Skew-resource trade-off based on tolerance metric 55
2.10.5 Run Time Analysis 57
2.10.6 The Effect of Eliminating Isolated Flip-Flops 61
2.10.7 Analysis of Intermediate Results during Simulated Annealing Process 65
2.11 Summary 66

Chapter 3 Extensions of Algorithm for Synthesizing Clock Spine Networks 67

3.1 Crossed Clock Spine Allocation and Placement 67

3.1.1 Introduction 67
3.1.2 Proposed Method 68
3.1.3 Experimental Results 74
3.2 Application of Clock Gating 77
3.2.1 Introduction 77
3.2.2 Activity Pattern 79
3.2.3 Problem Definition 81
3.2.4 Proposed Method 81
3.2.5 Experimental Results 82
3.3 Summary 82

Chapter 4 Conclusion **85**

Appendices **87**

Chapter A The Experimental Results of ISPD2010 Benchmark Circuits with Different τ Values. **88**

초록 **100**

List of Figures

Figure 1.1 The structure of PST buffer which has digitally controlled analog delay line which consists of 20-bit delay control register, a two-stage variable delay circuit, and a push-pull style output buffer [1]. 4

Figure 1.2 Four types of clock network topologies: (a) clock tree structure, (b) clock mesh structure, (c) clock tree structure with links, and (d) clock spine structure. 6

Figure 1.3 H-tree and asymmetric clock trees based on delay matching from [2]. 9

Figure 1.4 Multilevel symmetric H-tree distribution and global clock grid from [3]. 10

Figure 1.5 Schematic diagram and 3D visualization of clock distribution from [4]. 11

Figure 1.6 Clock distribution network for Pentium 4(180nm) [5]. . . 12

Figure 1.7 Die plot showing horizontal straight, continuous clock stripes from [6]. 13

Figure 1.8 Clock distribution of low power IA processor consisting of centralized spines and binary trees from [7]. 14

Figure 1.9 The block diagram of clock network synthesized in [8]. . 15

Figure 2.1 The conceptual view of clock spine structure. 21
Figure 2.2 Clock skew attenuation factor from [6]. 22
Figure 2.3 RC network modeling for simple clock spine network. . . 23
Figure 2.4 The comparison of clock skew at the top-level tree (red triangles) with that at the top-level tree + spines + stubs (blue circles). 26
Figure 2.5 The proposed synthesis flow of clock spine networks. . . 27
Figure 2.6 An illustration of one-to-one correspondence between an instance of clock spine network and an instance of slicing floorplan. (a) A clock spine network with 6 spines. (b) A slicing floorplan with 6 blocks. Notations $*_i$ and $+_j$ represent the vertical and horizontal cutting lines of 5×5 grids, respectively. 30
Figure 2.7 Example to illustrate deciding the position of clock spine for a cluster. The x-coordinates of clock sinks are projected to calculate the position of vertical clock spine in the cluster. 33
Figure 2.8 Clock spine extension to improve the variation tolerance. (a) A clock spine that contains isolated clock sinks. (b) Clock spine is extended, resulting in placing a new buffer (b_4) to drive s_1 and s_2. 38
Figure 2.9 Examples to illustrate the tolerance metric for a spine segment. 40
Figure 2.10 Example showing the application of *move* operations to postfix expression. 42

Figure 2.11 Example of an invalid binary tree for constructing legal slicing floorplan. The slicing cut $*_4$ should be on the right side of $*_2$. 44
Figure 2.12 The indices of the binary tree can be examined by propagating the index constraints (in the properties) from root to leaf nodes. 45
Figure 2.13 Example showing the spine-based recursive bipartition. . 48
Figure 2.14 Two clock spines are merged with resized buffers b'_1, b'_2 and b'_3 to drive s_1, s_2 in a spine as well as some sinks in the other spine. 49
Figure 2.15 The global clock skew, total wirelength, total buffer area, and power consumption of clock spine networks synthesized for ISPD03 while changing τ from 100 to 800. . . . 59
Figure 2.16 Clock spine synthesis results for benchmark circuit ISPD06 with changing τ. 62
Figure 2.17 Comparison of the variation of arrival times to isolated sink #624 in circuit ISPD08 before and after the application of our refinement. The blue histogram indicates the latency distribution on our clock spine network without refinement while the orange one indicates the distribution on our spine network with refinement. 64

Figure 3.1 Single-line spine and crossed spine structures. 68

Figure 3.2 Example of crossed spine placement problem. (a) An instance of crossed spine placement problem of block b. (b) Block b in (a) is replicated and arranged in a 2×2 matrix. (c) A simple rectilinear polygon derived from (a). 70
Figure 3.3 The modified synthesis flow of clock spine networks. . . . 74
Figure 3.4 Layout of the clock spine network using single-line spines and the network using crossed spines as well as single-line spines, produced by our algorithm for circuit ISPD08. 76
Figure 3.5 An example to illustrate the activity patterns of clock sinks and clock spine. 80

List of Tables

Table 1.1 The properties of clock tree, clock mesh and clock spine networks. 16

Table 2.1 The number of clock sinks in benchmark circuits in [9] and the configurations of clock mesh networks [10] and clock spine networks. 51

Table 2.2 Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock tree networks by [11] and our spine networks under process variations. 53

Table 2.3 Comparison of global clock skew of clock tree networks by [11] and our spine networks without process variations. 54

Table 2.4 Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock mesh networks by [10] and our clock spine networks. 56

Table 2.5 Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock spine networks with $\tau = 800, 400, 200$. 60

Table 2.6 Runtime and starting values for synthesizing clock spine networks consisting of single-line spines. 63

Table 2.7 Clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of intermediate clock spine networks produced during the simulated annealing process on circuit ISPD08. Each intermediate solution is obtained at a fixed temperature. 65

Table 3.1 Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock spine networks with only single-line spines and clock spine networks considering crossed clock spines. 75

Table 3.2 Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock spine networks by [12] and ours considering clock gating. 83

Table A.1 Experimental results of ISPD03 with different τ values. . 89

Table A.2 Experimental results of ISPD04 with different τ values. . 89

Table A.3 Experimental results of ISPD05 with different τ values. . 90

Table A.4 Experimental results of ISPD06 with different τ values. . 90

Table A.5 Experimental results of ISPD07 with different τ values. . 91

Table A.6 Experimental results of ISPD08 with different τ values. . 91

Chapter 1

Introduction

1.1 Clock Distribution Network

Clock distribution network is used to deliver a clock signal from clock source to the clock sinks (e.g., flip-flops and latches) in synchronous digital systems. Because all clock sinks are synchronized and activated by the clock signal, design of clock distribution networks have to consider many design metrics.

First important design metric is clock frequency. It represents the number of oscillations of clock signal in a second. The data transfer rate of synchronous digital system depends on the clock frequency.

Second metric is clock skew. The maximum difference among the clock latencies, which is called *(global) clock skew*, causes adverse effects on high-speed digital systems. The increase of clock skew generally reduces the clock speed of circuits, often causing a circuit failure. Consequently, it is essential for the circuit designers to minimize the clock skew or control the clock skew in a bound during the synthesis of clock networks.

In addition to the problem of reducing or controlling clock skew, the power consumption also should be considered. As stated in works of [13, 14, 15], the power consumption of clock distribution network reaches up to 40% of total power. Recently, it is becoming harder to reduce power consumption of clock distribution network due to increased requirements for synthesizing robust clock distribution network.

Last metric is clock transition time. The transition time of clock signal is defined as switching time from V_l to V_h (or from V_h to V_l.) The V_l and V_h are 10% and 90% of supply voltage, respectively. Long transition time of clock signal can increase the delay and power consumption of corresponding buffer. Thus, the transition time of clock signal should be constrained.

1.2 The Effect of Process Variations on Clock Skew

Process variation—deviation in parameters such as gate width, device threshold voltage, channel length, oxide thickness and wire width/thickness—has significant impact on clock skew and must be taken into account in clock network synthesis/optimization. As the technology node shrinks, the process variation is rapidly increasing, which makes unacceptable to treat the clock skew as a deterministic value. The work [16] observed that the delay variation of interconnect produces up 25% variation of clock skew.

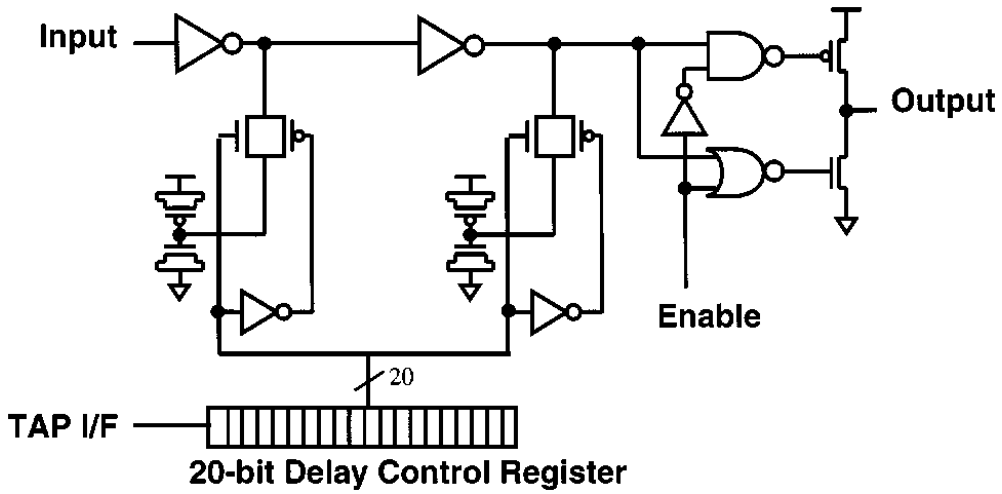
In addition, the continued CMOS process technology scaling to sub-100nm regime has made it difficult to predict the clock skew of the individual fabricated chips. According to work [17], the clock skew of H-tree can be increased from 15% to 30% of the clock cycle while technology scales from 180nm to 50nm technology due to variation.

Traditionally, worst-case based timing analysis is used to consider the ef-

fect of the delay variation caused by the process variation in signoff process. However, as the delay variation increases, the assigned timing margin in signoff process account for a significant portion of clock period, causing to degrade the circuit performance and excessive clock resource consumption. To cope with this problem, the statistical static timing analysis (SSTA) has been addressed. By considering the delays as random variables, the excessive margins in signoff process is removed [18, 19, 20].

To resolve the timing fault problem due to process variation, other researchers have used Post-Silicon Tunable (PST) clock buffers [1] at the post-silicon stage. The structure of a PST buffer is illustrated in Figure 1.1. It has digitally controlled analog delay line which consists of 20-bit delay control register, a two-stage variable delay circuit, and a push-pull style output buffer. At the post-silicon stage, already inserted PST buffers can help fixing timing violations by process variations, resulting in the increase of chip yield. When inserting the PST buffers into clock distribution networks to improve yield, minimizing the total tunable delay range of PST buffers to be allocated under the target yield constraint and minimizing the total number of PST buffers to be allocated under the target yield constraint are important. These problems are addressed in [21].

In addition, to mitigate the clock latency variations caused by process variations, various clock network structures such as clock mesh, clock tree with cross links, clock spines have been proposed and elaborated. These structures are elaborated in the next section.



(a)

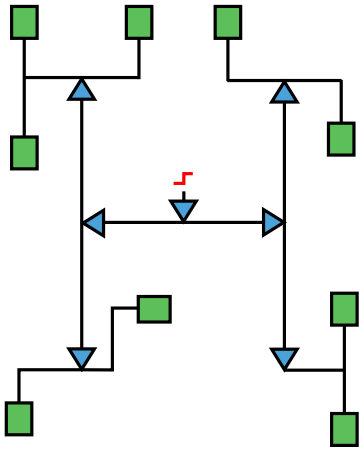
Figure 1.1: The structure of PST buffer which has digitally controlled analog delay line which consists of 20-bit delay control register, a two-stage variable delay circuit, and a push-pull style output buffer [1].

1.3 Clock Network Topologies

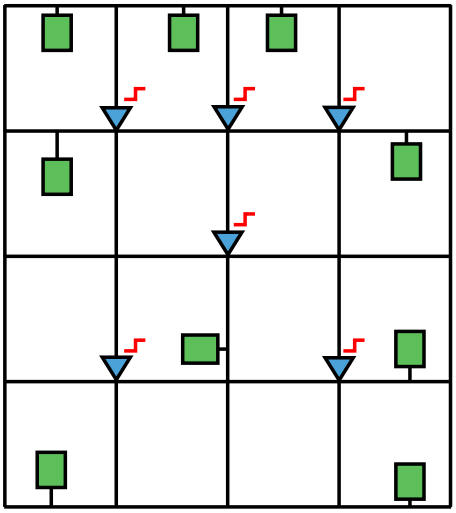
Being aware of the importance of mitigating clock latency variation caused by the CMOS process technology scaling, designers and researchers have investigated various clock network structures to deal with the clock skew variation problem. Clock tree (e.g., [22, 11, 23, 24]) has been the most widely used structure of clock network due to its simplicity. Due to its convenience of analysis and optimization, the clock tree network is commonly used in automatic synthesis. Figure 1.2(a) shows an instance of clock tree in which the green squares and blue triangles represent clock sinks and clock buffers respectively. Since every clock sink receives the clock signal from the clock source through exactly one clock path, there is a high risk of non-negligible clock skew variation.

One structure proposed to reduce the variability is clock mesh (e.g., [25, 26, 10, 27, 28, 29]). Figure 1.2(b) shows an instance of clock mesh, which has more than one clock path to a clock sink through the mesh grids. This redundancy of clock paths allows variations affecting one path to be compensated by other clock paths. Even though the clock mesh provides a high variation tolerance, the clock resource (wirelength and buffer area) and power consumption on the mesh is unacceptably high due to the massive parasitic capacitance of the clock resources of mesh. Since the clock mesh covers the area where the flip-flops are placed, the important things for designing clock mesh are the number of mesh grids, stub constraints, and buffer allocation. Thus, analysis and optimization of clock mesh network is harder than clock tree. In addition, the output pins of mesh buffers are connected to the same net (i.e., mesh wire.) It makes the analysis of clock mesh network difficult.

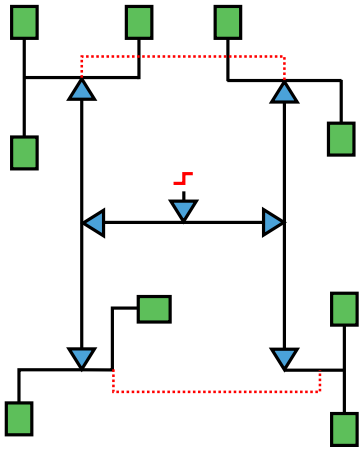
In the meanwhile, clock tree with links shown in Figure 1.2(c) affords a reasonable solution which compromises clock resource with clock skew variation



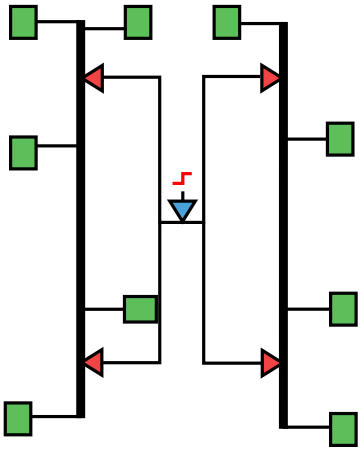
(a) Clock tree structure



(b) Clock mesh structure



(c) Clock tree structure with links



(d) Clock spine structure

Figure 1.2: Four types of clock network topologies: (a) clock tree structure, (b) clock mesh structure, (c) clock tree structure with links, and (d) clock spine structure.

by adding cross links to internal nodes on the clock tree (e.g., [30, 31, 32, 33]). Since the determination of the placement locations on clock tree at which links (dotted red lines in Figure 1.2(c)) are inserted directly affects the degree of clock skew variation, the main issue of the algorithms synthesizing clock tree with links is to find a minimal number of internal locations on the clock tree at which the links are inserted to meet the constraint of clock skew variation (e.g., [31, 32, 33]). Like the clock mesh, it is difficult to analyze the clock tree with cross links because there are nets driven by two or more buffers.

Clock spine shown in Figure 1.2(d) is another alternative to the clock tree with links, but the structure is totally different. The clock spine network contains vertical and horizontal clock wires, shown as thick lines in Figure 1.2(d), which are called *clock spines* and every clock sink is attached to its nearby spine. Since a spine has multiple clock buffers to drive the clock sinks connected to the spine, shown as red triangles in Figure 1.2(d), the clock sinks receive clock signals from multiple paths. The automatic synthesis of clock spine networks is not as easy as synthesizing clock tree networks because many parameters have to be determined such as the number of spine wires, the position and length of spine wires, which clock sinks have to be connected to corresponding spine segment, constraints for stub wires, and placement of spine buffers.

Contrary to the clock tree with links, so far, no work has addressed the problem of automating the synthesis of clock spine networks. Recently, the work in [12] addressed the clock spine synthesis, in which the creation of spines is totally based on the activity pattern of the individual FFs which can be obtained from power mode information. Because the main objective of their proposed algorithm is reducing the total energy consumption through spine-level clock gating based on the activity patterns of FFs, it is not enough to optimize the clock resource and power consumption in the absence of power

mode information.

1.4 Microprocessor Clock Distributions

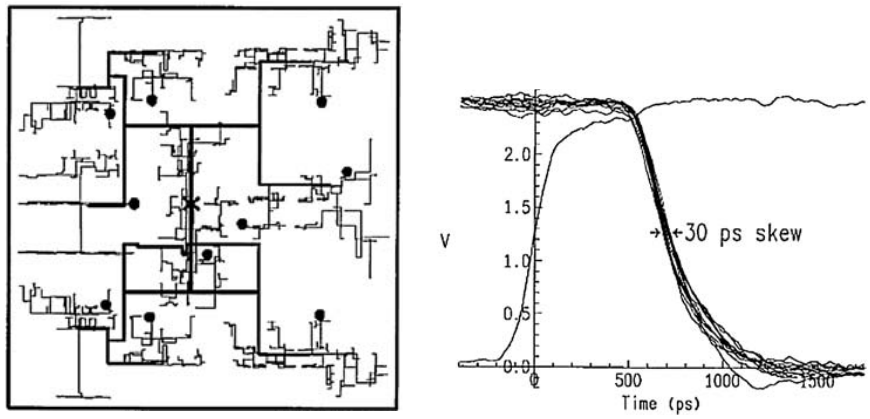
The clock topologies described in previous section have been constructed in hybrid combinations. For example, top-level clock tree and clock mesh is combined to drive overall circuit. Clock spines also have been synthesized as pre-driver of global clock grid network.

In IBM 400MHz S/390 microprocessor, symmetric and asymmetric clock trees are combined to deliver clock signals to all sequential elements [2]. In Figure 1.3(a), clock distribution network of IBM 400MHz S/390 microprocessor is described. Thick lines in Figure 1.3(a) represents H-tree structure constructed as first level of clock network which drives multiple secondary trees. This H-tree connects the global clock from the center of the die to 9 sector buffers. The 9 sector buffers distribute the clock to 580 global clock receivers through secondary trees. These secondary trees are asymmetric but delays are matched. The measured skew is described in 1.3(b).

For 1 GHz 64bit single issue PowerPC processor, multilevel symmetric H-tree and clock grid is synthesized for delay matching [3]. Figure 1.4 shows the delay tuned H-trees and main grid which covering most of the chip.

Figure 1.5(a) shows another example of hybrid tree and grid design from [4]. A multilevel H-tree structure delivers the clock from phase-locked loop (PLL) to 64 sector buffers which drive common grid through multiple second level tuned trees. The 3D visualization of the global clock distribution is illustrated in Figure 1.5(b).

The work in [5] illustrates the concept of binary distribution trees and clock spines in Intel 180nm Pentium® microprocessors. The clock distribution net-



(a) (b)

Figure 1.3: H-tree and asymmetric clock trees based on delay matching from [2].

work is synthesized to operate up to 4 GHz frequency. The binary trees connected to middle spine deliver the clock from the central PLL to other spines in balanced fashion. Other spines drive local binary trees. Final clock drivers are constructed as matched branches to drive local loads. The final drivers incorporate delay tuning capability to further optimize the skew through post silicon tuning. Figure 1.6 shows the implemented clock distribution network in work [5].

In work [6], global clock distribution scheme for 90nm multi-GHz IA microprocessor is introduced. This clock distribution scheme can operate up to 6.9GHz of clock frequency. In this processor, clock distribution network consists of Pre-Global Clock Network (PGCN), Global Clock Grid (GCG), and local clock network with clock gating structure. The PGCN are comprises 12 inversion stages from the PLL to the die center, and 15 stages to the input of GCG drivers. And every 2nd or 3rd stage of PGCN is shorted to keep the accumulated skew below a threshold. Figure 1.7 shows the global clock stripes

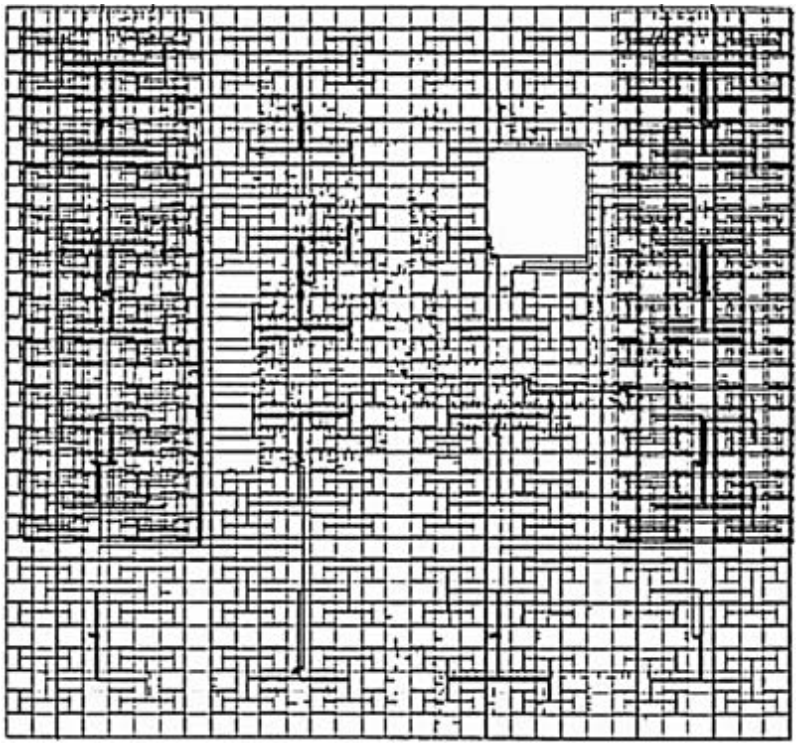
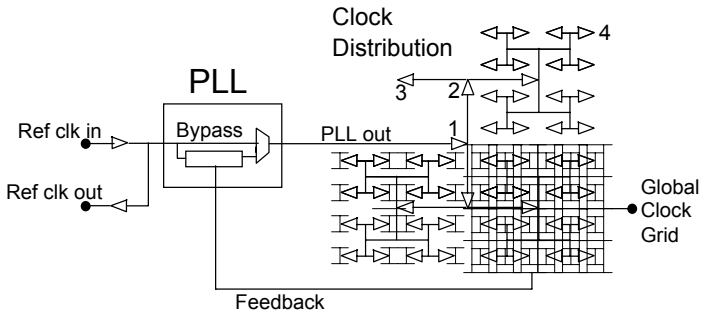
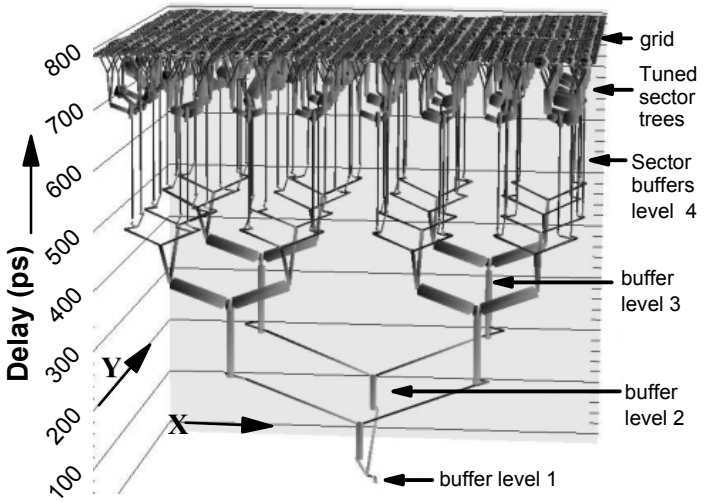


Figure 1.4: Multilevel symmetric H-tree distribution and global clock grid from [3].

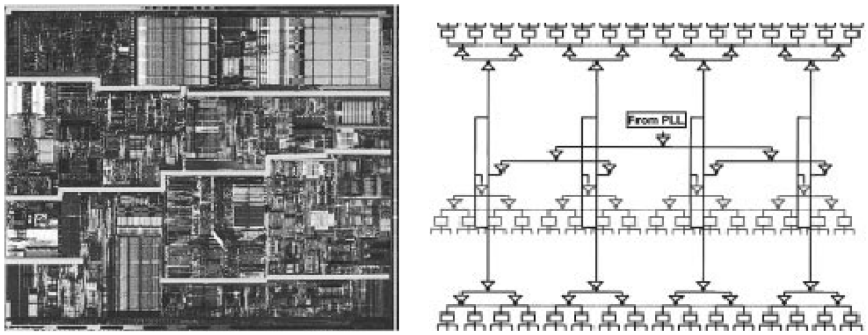


(a) Schematic diagram with PLL and 4 buffer levels.



(b) 3D visualization of the global clock distribution.

Figure 1.5: Schematic diagram and 3D visualization of clock distribution from [4].



(a) Triple spine distribution.

(b) Binary distribution tree with three clock spines.

Figure 1.6: Clock distribution network for Pentium 4(180nm) [5].

to short the intermediate stage of clock distribution network. A global clock skew of less than 10ps was achieved with this structure.

Another hybrid network with clock tree and clock spine is synthesized for sub-1W to 2W low power IA processor for mobile internet devices [7]. For global binary tree, clock recombination is limited and drives the clock signal to clock spines. The outputs of spine buffers are shorted with a high metal layer (M8) to equalize the driver delays. In this microprocessor, a grid-less clock topology is used to reduce the power consumption of clock network. Overall structure is illustrated in Figure 1.8.

Intel® ATOM™ processor for ultra low power SoC for handheld applications also adopted the hybrid network with clock tree and clock spine network [8]. In this processor, the clock is architected with power saving as a top priority. Clock recombination in global distribution is limited to a few critical stages after carefully investigating the power and skew trade-off. Figure 1.9 shows the block diagram of clock network of [8].

In summary, clock tree, clock mesh and clock spine topologies are adopted

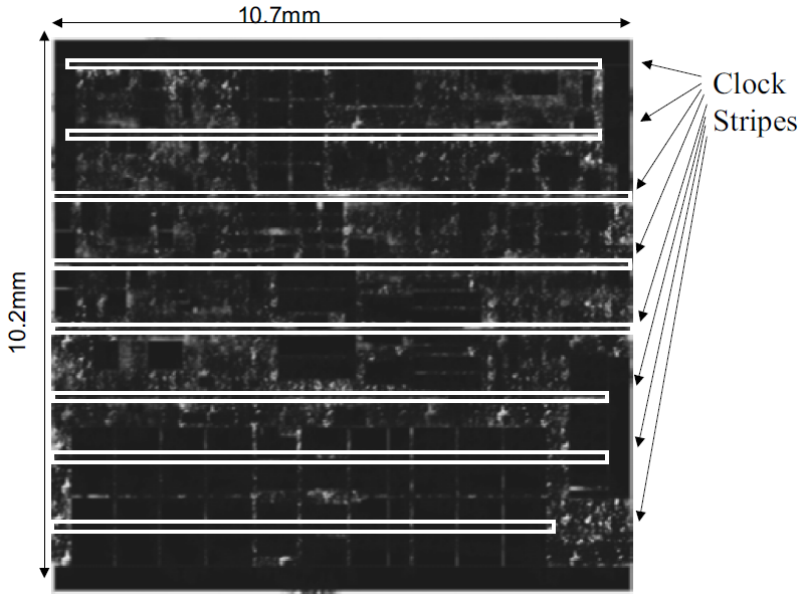


Figure 1.7: Die plot showing horizontal straight, continuous clock stripes from [6].

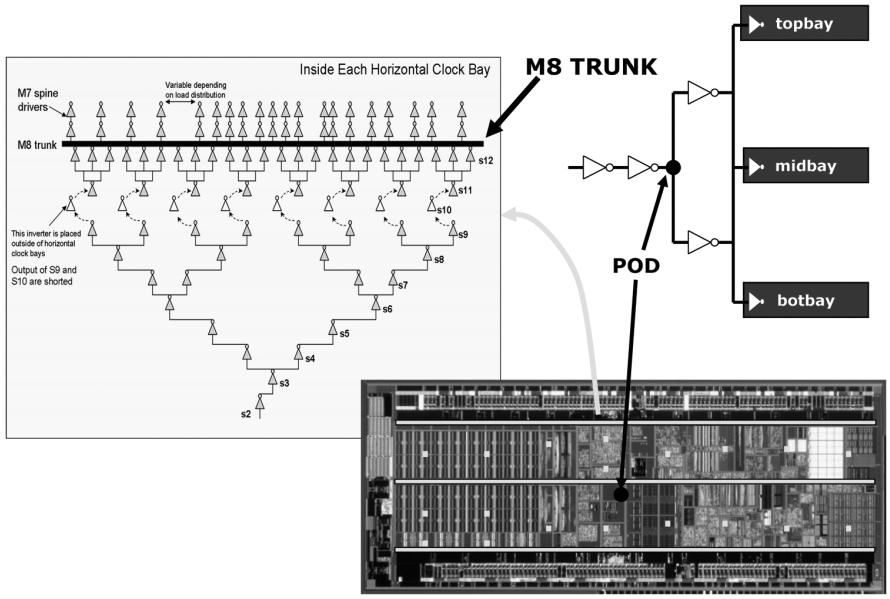


Figure 1.8: Clock distribution of low power IA processor consisting of centralized spines and binary trees from [7].

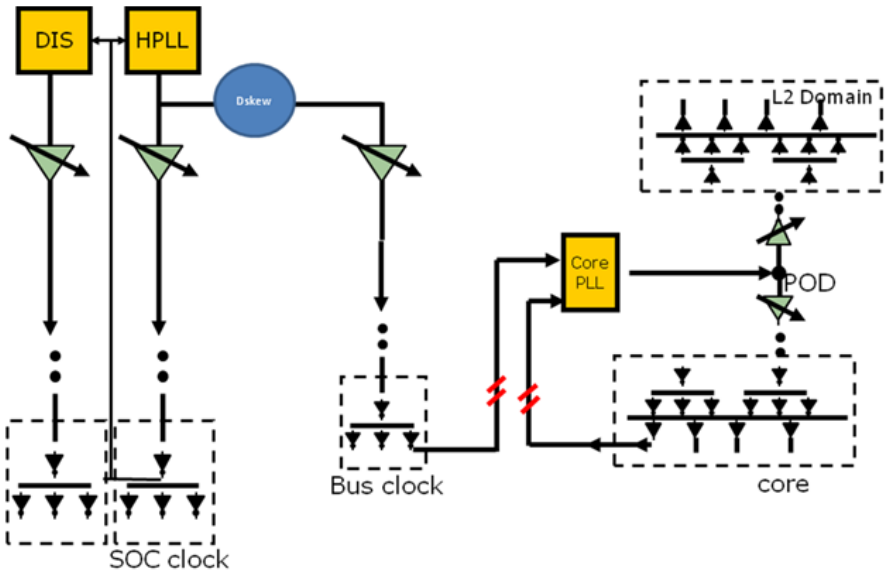


Figure 1.9: The block diagram of clock network synthesized in [8].

for various microprocessor designs. Multilevel H-tree or delay matched clock trees are usually used for top-level to drive clock signal from PLL to clock grids or clock loads. Also clock trees are synthesized as delay-matched branches to drive local loads.

Clock mesh is usually synthesized to drive local level. Since the clock skew of clock mesh can be controlled by just increasing or decreasing the number of mesh grids, synthesizing clock mesh in leaf level to drive local loads is easier and faster than synthesizing the clock spine for high performance processor.

To drive global clock mesh or local clock trees, clock spine structures are applied. Because the clock mesh structure outperforms than clock spine structure in high performance processor to reducing clock skew induced from process variations, clock spine structure is usually adopted for pre-grid clock networks. However, as demand for mobile devices grows, a low-power processor design

method was required. In this point of view, clock spine is adopted as an alternative for clock mesh in [7]. Clock spine network can reduce power consumption as compared to clock mesh network.

Table 1.1: The properties of clock tree, clock mesh and clock spine networks.

	Clock tree	Clock mesh	Clock spine
Adopted level	Top/local level	Local level	Top level (Local level for low-power design)
Power consumption	Low	High	Medium
Tolerance to variation	Low	High	Medium

1.5 Contributions of This Dissertation

In this dissertation, we propose a synthesis methodology for clock spine networks, which is able to explore the trade-off between the clock resource and clock skew from process variations. Also, fast algorithm for synthesizing crossed spine network is proposed to explore the more diverse clock spine networks which can reduce clock resource and power consumption. Finally, synthesis methodology for clock spine network exploiting activity pattern of FFs which can obtained from power mode information is proposed to consider reducing dynamic power consumption.

Specifically:

In Chapter 2, the benefits of shorting intermediates stages in reducing clock skew is illustrated. Also the algorithm for synthesizing and exploring clock spine networks is developed. The key idea of the development of our synthesis methodology is to transform the problem of allocating and placing single-line clock spines on a plane into a slicing floorplan optimization problem, in which

every candidate of clock spine network structure is uniquely expressed into a postfix notation to enable a fast cost computation in the slicing floorplan optimization. Also, recursive bipartitioning method and refinement method for initial solution are proposed. As a result, our synthesis methodology is able to explore diverse structures of clock spine network to find a globally optimal one within a reasonable computation time. Through experiments, we confirm that our methodology synthesizes clock spine networks with much less clock power over that of clock mesh at the expense of a little decrease of variation tolerance, and synthesizes clock spine networks with much higher variation tolerance over that of clock tree at the expense of a little consumption of clock power. (A short version of this chapter is in [34].)

In Chapter 3, the proposed algorithm for synthesizing and exploring clock spine networks is extended. First, our methodology is extended for solving the crossed spine allocation problem in order to explore more diverse clock spine structures. To adopt the spine-level clock gating technique, the new formulation considering activity patterns of clock sinks is proposed.

Chapter 2

Algorithm for Synthesizing Clock Spine Networks

2.1 Introduction

In the last few years, clock spine structures have been designed in some microprocessors. Intel 180nm Pentium® microprocessors used binary distribution trees and central clock spines as clock distribution network [5]. Local loads are driven by matched branches in local clock trees. The objective of clock spine structure is to reduce clock jitter induced by dynamic current in power supply network.

In contrast, the global clock distribution scheme for 90nm multi-GHz IA microprocessor [6] adopted clock spine structure for every 2nd or 3rd level of PGCN to reduce the clock skew from top-level.

However, as demand for mobile devices grows, a low-power processor design method while maintaining high performance was required. In this point of view, clock spine is adopted as an alternative for clock mesh in [7]. With synthesizing

clock spine network as an alternative to clock mesh network, the clock resource and power consumption can be reduced by sacrificing the tolerance to clock skew variation.

In these works, the synthesis of clock spine network is not automated. It's because many parameters have to be considered to synthesize effective clock spine networks, such as the length, position and connected sinks for each spine, the number of clock spines, and placement of stub wires and spine buffers. (In contrast, the number of mesh grids and placement of mesh buffers are dominant parameters for mesh construction.)

To cope with clock spine synthesis problems, the efficient synthesis method for clock spine networks which can satisfy clock skew and slew constraints, with minimal clock resource and power consumption is needed. In addition, the design space exploration method which can trade-off between tolerance to clock skew variation and resource consumption is needed to design efficient the clock distribution network which is well suited for the design specification and constraints. (In our knowledge, the automation of clock spine synthesis is first addressed in work [12]. In work [12], clock spine networks are synthesized through FF clustering algorithm based on K-means clustering. However, since the flip-flop clustering is conducted based on the activity patterns information of the individual FFs, their proposed algorithm is not enough to optimize the clock resource and power consumption in the absence of power mode information.)

In this chapter, the benefits of shorting intermediates stages in reducing clock skew is illustrated. And then, we propose a synthesis methodology for clock spine networks, which is able to explore the trade-off between the clock resource and clock skew from process variation, Precisely, (1) we define the clock spine allocation and placement problem; (2) we show how to transform the instance of clock spine allocation and placement problem into the instance

of slicing floorplan optimization; (3) we illustrate how to apply slicing floorplan optimization algorithm to solve clock spine allocation and placement problem. (4) Recursive bipartition method, tolerance metric for spine wires and spine refinement method are proposed to improve the performance of clock spine networks.

2.2 Preliminaries and Motivation

2.2.1 Proposed Clock Spine Structure

Clock spine networks we are considering consist of three parts: (1) top-level tree, (2) clock spines, and (3) stubs with sinks. Top-level tree delivers clock signal from clock source to clock spines. Clock spines are vertical or horizontal straight metal lines with multiple clock buffers to deliver clock signal to sinks through spines and stubs. At least two spine buffers have to be placed on each clock spine to maintain the tolerance to clock skew variability. Stubs are short wires connecting sinks to spines. (In this dissertation, we assume the clock sinks are connected to clock spine with straight stub wires.) Figure 2.1 shows the conceptual view of clock spine network we are targeting.

2.2.2 The benefits of shorting intermediate stages in reducing clock skew

In work [6], clock nets within a stage are strategically shorted to reduce skew over multiple stage due to random variations. And skew averaging effect is measured by *skew attenuation factor* α, which is defined as the ratio of observed output skew at the input of the receivers, to a given input skew at the input of the drivers. An example circuit is illustrated in Figure 2.2(a). In Figure 2.2(a),

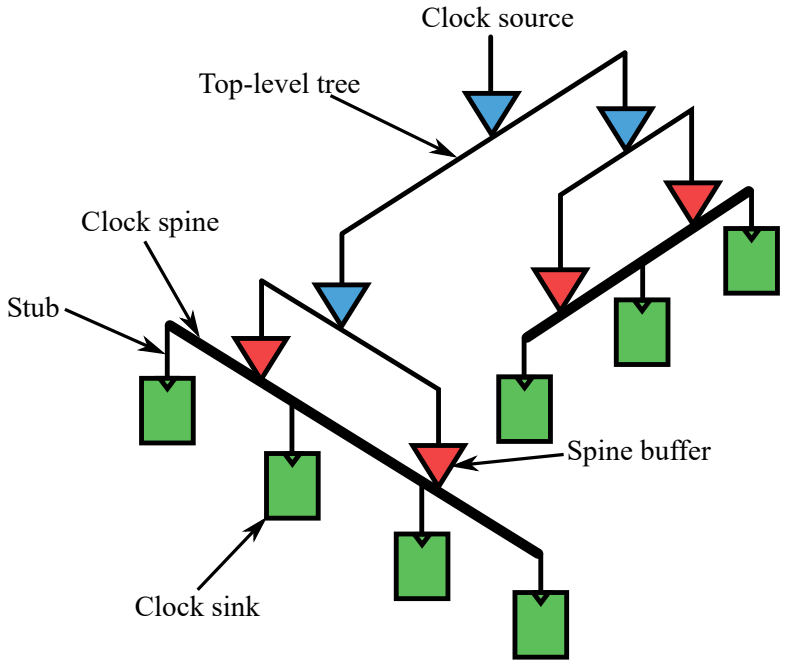


Figure 2.1: The conceptual view of clock spine structure.

α is defined as

$$\alpha = \frac{skew(u,y)}{skew(a,b)} \tag{2.1}$$

where $skew(a,b)$ means the clock skew between node a and b. If output nodes u and y are unshorted (e.g. H-tree), this tree has no skew attenuation ($\alpha = 1$). If output nodes u and y are shorted with a resistor R_1 in Figure 2.2(a), the $skew(u,y)$ is attenuated. If R_1 is ideal resistor, $skew(u,y)$ will become zero, resulting in $\alpha = 0$. According to work [6], typical values of α ranges between 0.1 to 0.8 depending on the technology and circuit sizing trade-offs.

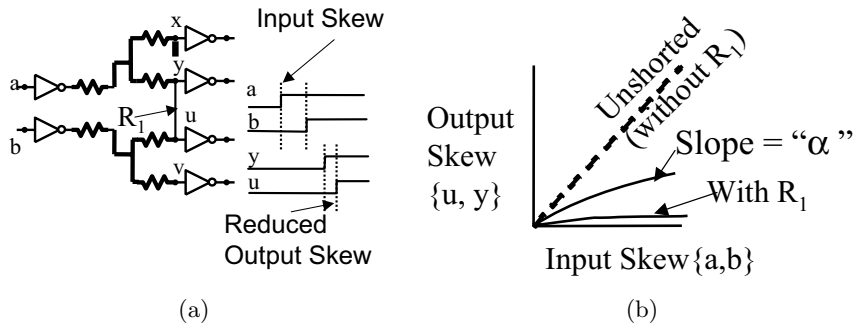


Figure 2.2: Clock skew attenuation factor from [6].

In addition, the benefits of shorting intermediates stages in reducing clock skew can be analyzed by modeling the small clock spine network and RC analysis. If the RC networks is given, the Elmore's delay [35] for i_{th} node in the RC network can be described as

$$t_{d,i} \equiv \frac{\sum_{j=1}^{n} R_{i.j} C_j [v_j(\infty) - v_j(0+)]}{v_i(\infty) - v_i(0+)} \tag{2.2}$$

or in matrix form

$$\mathbf{t_d} \equiv \frac{\mathbf{RC}[\mathbf{v}(\infty) - \mathbf{v}(0+)]}{\mathbf{v}(\infty) - \mathbf{v}(0+)} \tag{2.3}$$

where $\mathbf{R} \equiv \mathbf{G}^{-1}$, $\mathbf{C}$ is a diagonal matrix with node capacitance values. $\mathbf{v}(\infty)$ and $\mathbf{v}(0+)$ are the final and initial node voltages [36, 37].

According to work [38], the method for Elmore's delay update when two nodes in RC matrix is connected by resistors. If $\mathbf{t_d}$ is already calculated and node k and j are connected by resistance R_s, the updated Elmore's delay is

$$\hat{\mathbf{t_d}} = \frac{\mathbf{t_d} - \frac{t_{d,k} - t_{d_j}}{R_s + r_k - r_j}\mathbf{r}}{\hat{\mathbf{v}}(\infty)} \tag{2.4}$$

where $\mathbf{r} = \mathbf{R}\xi_{k,j}$ and $\xi_{k,j}$ is column vector with $+1$ in the k_{th} row and -1 in the j_{th} row and zeros everywhere else.

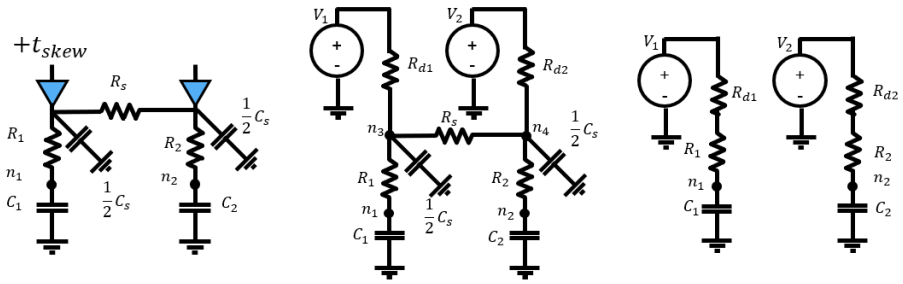


Figure 2.3: RC network modeling for simple clock spine network.

Simple clock spine network with two spine drivers and two clock sink can be represented as Figure 2.3(a). Clock skew due to random variations is included in t_{skew} value. This network can be translated into RC network in Figure 2.3(b). Figure 2.3(c) illustrates the original RC network without clock spine connection. The clock skew between node n_1 and n_2 in Figure 2.3(c) is

$$skew_{orig} = (R_1C_1 - R_2C_2) + (R_{d1}C_1 - R_{d2}C_2) + t_{skew}. \tag{2.5}$$

In this equation, $R_1C_1 - R_2C_2$ term corresponds to Elmore's delay induced by stub network and loading capacitances (e.g. capacitance values of clock

sinks). $R_{d1}C_1 - R_{d2}C_2$ term represents the clock skew induced by imbalance between loading capacitance and drivers' output resistance.

With (2.4) from [38], the clock skew between node n_1 and n_2 in Figure 2.3(b) can be calculated as following equation,

$$\begin{aligned} skew_{spine} &= (R_1C_1 - R_2C_2) \\ &+ \frac{R_s}{R_s + R_{d1} + R_{d2}}(R_{d1}C_1 - R_{d2}C_2 + t_{skew} + \frac{1}{2}(R_{d1} - R_{d2})C_s) \end{aligned} \tag{2.6}$$

where R_s is resistance of spine wire and C_s is capacitance of spine wire. Spine wire is modeled as π model.

The capacitance and resistance value of spine wire affects the clock skew after spine wire connection. First, spine wire capacitance increases the nominal clock skew between node n_1 and n_2. This increased nominal clock skew is affected by drivers' resistance value. If two drivers are mismatched, this clock skew would be increased. In addition, if C_s value become smaller, increased portion of nominal clock skew also become smaller. In summary, to reduce the increased portion of nominal clock skew induced by clock spine capacitance, using shorter spine can be advantageous and matching the resistance of driving network would be help reducing skew.

In the meanwhile, the proportion between spine resistance, R_s, and resistance values of spine drivers, R_{d1} and R_{d2} affects the reduction of clock skew transferred from top-level and clock skew induced by clock spine capacitance. According to (2.6), reducing resistance of spine wire can reduce the ratio $\frac{R_s}{R_s+R_{d1}+R_{d2}}$, resulting in reduction of overall clock skew between node n_1 and n_2.

In conclusion, the shorter the connection between two buffers, the larger reduction of clock skew induced by process variations.

In general CMOS technology, the resistance of driver is much larger than the resistance of the spine wire, the ratio $\frac{R_s}{R_s+R_{d1}+R_{d2}}$ dominates the clock skew increase by spine capacitance, resulting in the reduction of clock skew between connected two nodes.

2.2.3 Smoothing Effect of Clock Spine

The structure of spine networks, as opposed to that of tree networks, is able to mitigate the clock skew variation at the expense of more wire since multiple signal paths will be formed to every sink from clock spines, which we call the *smoothing effect* of clock skew by clock spines. To see how much the smoothing effect can be achieved by clock spines, we synthesized clock spine networks (i.e., top-level tree + spines + stub with sinks) for the ISPD2010 benchmark circuits [9], and 100 times of Monte Carlo simulation were conducted under delay variations. Then, for each spine s_i included in clock spine networks, we measured the maximum delay difference, denoted by $\delta_T(s_i)$, at the input pins of the spine buffers on s_i (i.e., the clock skew of the top-level tree driving s_i) and the maximum delay difference, denoted by $\delta_{T+S}(s_i)$, at the sinks attached to s_i (i.e., the clock skew of the network with top-level tree + spine s_i + stub.) Figure 2.4 shows the distribution of $\delta_T(s_i)$ and $\delta_{T+S}(s_i)$ values, which are represented by red triangles and blue circles, respectively. The vertical gap between $\delta_T(s_i)$ and $\delta_{T+S}(s_i)$ indicate that spines play a major role in lessening the skew variation. Furthermore, the smoothing effect is more prominent for longer spines with more buffers.

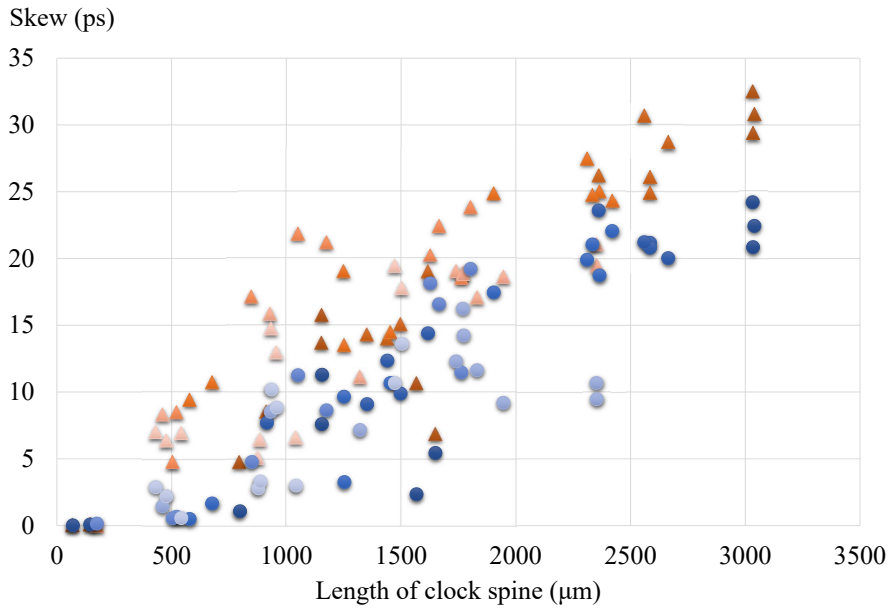


Figure 2.4: The comparison of clock skew at the top-level tree (red triangles) with that at the top-level tree + spines + stubs (blue circles).

2.3 Overall Flow

Figure 2.5 shows our overall clock spine synthesis flow. Our algorithm is performed in four steps: (1) compute an initial number of clock spines and the grid size; (2) generate a clock spine network, using the parameters obtained in Step (1), by transforming the clock spine allocation and placement problem into a slicing floorplan optimization problem and using a conventional slicing floorplanning algorithm to obtain a floorplan; (3) bipartition recursively each of the spines on the allocation structure obtained in Step (2); and (4) refine the clock spine network. Note that crossed clock spine as well as single straight-line clock spine is explored in Steps (1) and (3).

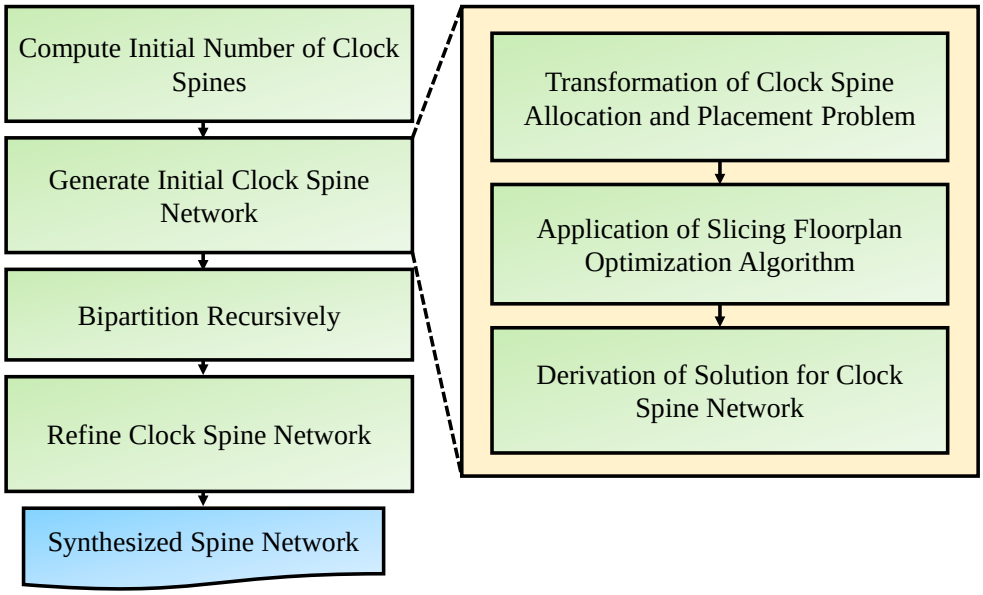


Figure 2.5: The proposed synthesis flow of clock spine networks.

2.4 Determination of the Number of Clock Spines

We adopt a nearest-neighbor (NN) based clustering approach to find the number of clock spines to be allocated, following the process in [11]. Initially we assume that every sink is allocated to a distinct spine. Then, we iteratively group a pair of spines, say c_i and c_j, that can be replaced by one spine, say c_z, and minimizes the quantity of $\Delta NN_{cost}(c_i, c_j)$:

$$\Delta NN_{cost}(c_i, c_j) = NN_{cost}(c_z) - (NN_{cost}(c_i) + NN_{cost}(c_j)). \tag{2.7}$$

The function $NN_{cost}(c_z)$ is defined as:

$$\begin{aligned} NN_{cost}(c_z) = & \, max(l_x(c_i, c_j), l_y(c_i, c_j)) \\ & + (|c_i| + |c_j|) \cdot min(l_x(c_i, c_j), l_y(c_i, c_j))/2 \end{aligned} \tag{2.8}$$

where $l_x(c_i, c_j)$ and $l_y(c_i, c_j)$ are the horizontal and vertical lengths of the bounding box of the sinks of c_i and c_j, respectively. $|c_i|$ ($|c_j|$) indicates the number of clock sinks in c_i (c_j). The equation form of $NN_{cost}(c_i)$ and $NN_{cost}(c_j)$ is identical to that of $NN_{cost}(c_z)$.

Note that the first term of $NN_{cost}(c)$ is used to estimate the length of spine for the sinks in c while the second term is used to estimate the total length of stubs for the sinks in c. For each iteration we check the following two (stub quality) constraints for the validity of every grouping candidate. Only for the groupings that meet the two constraints, they are considered as candidates for the selection of grouping based on their $\Delta NN_{cost}(\cdot)$ values.

B_{stub} : the maximum limit of the difference of the lengths of the longest and shortest stub wires to a spine.

l_{bound} : the maximum limit of the length of every stub wire.

If there is no candidate for grouping, the iteration stops. The number of clusters in the last clustering results is regarded as the initial number of clock spines for the next step.

2.5 Transformation of Clock Spine Allocation and Placement Problem

2.5.1 Clock Spine Allocation and Placement Problem

With the number of clock spines obtained in our NN based algorithm, we transform the clock spine allocation and placement problem into a slicing floorplan optimization problem. The clock spine allocation and placement (CSAP) problem is formally stated as:

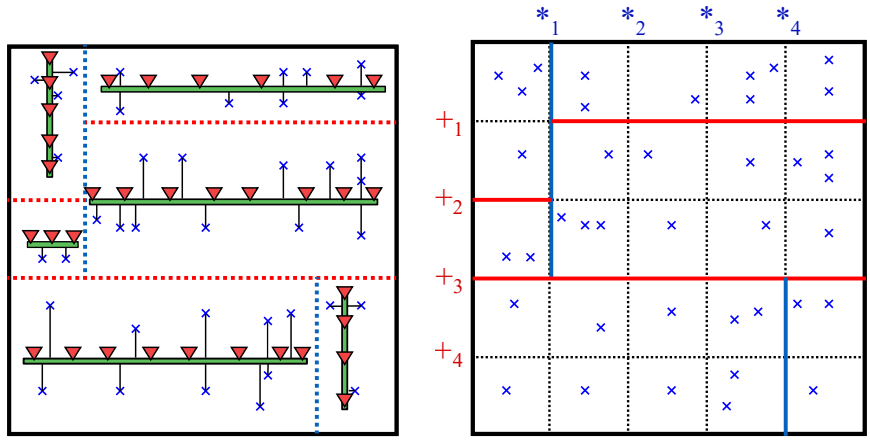
Problem 1. Clock spine allocation and placement (CSAP) problem: *For a set of clock sinks with their placement information, a buffer library $\mathcal{B}$, and parameters B_{stub}, l_{bound}, τ_{min} and N, generate a clock spine network $\mathcal{S}$ by (1) allocating N clock spines whose maximum difference between the stub lengths is not larger than B_{stub}, (2) placing spines and allocating stubs whose longest length is not longer than l_{bound}, and (3) allocating buffers from $\mathcal{B}$ and placing them on the spines, satisfying the tolerance constraint τ_{min} of delay variation (defined later) such that the following quantity is minimized:*

$$C(\mathcal{S}) = \alpha_1 \cdot WL_{sp}(\mathcal{S}) + \alpha_2 \cdot WL_{st}(\mathcal{S}) + \beta \cdot BA(\mathcal{S}) \tag{2.9}$$

where $WL_{sp}(\mathcal{S})$, $WL_{st}(\mathcal{S})$, and $BA(\mathcal{S})$ are the total length of the clock spines in $\mathcal{S}$, the total length of the stubs in $\mathcal{S}$, and the total area of the spine buffers in $\mathcal{S}$, respectively. α_1, α_2 and β are weighting factors.

For example, Figure 2.6(a) shows an instance, $\mathcal{S}$, of clock spine network with the allocation of 6 ($= N$) spines. The placement of the clock spines, sink

connection, spine buffer allocation/placement, and the tolerance metric of delay variation will be described later in this section.



(a) An instance of clock spine network.

(b) An instance of slicing floorplan corresponding to (a).

Figure 2.6: An illustration of one-to-one correspondence between an instance of clock spine network and an instance of slicing floorplan. (a) A clock spine network with 6 spines. (b) A slicing floorplan with 6 blocks. Notations $*_i$ and $+_j$ represent the vertical and horizontal cutting lines of 5×5 grids, respectively.

2.5.2 Slicing Floorplan Optimization Problem

a slicing floorplan optimization problem. The slicing floorplan optimization problem is formally stated as:

Problem 2. Slicing floorplan optimization (SFO) problem: *For a plane of $m \times n$ grids with a cost function $f(b)$ for a rectangle block b on the plane, and a parameter N, generate a slicing floorplan $\mathcal{F}$ by slicing the plane $N-1$ times to produce N blocks that minimizes the quantity:*

$$C'(\mathcal{F}) = f(b_1) + f(b_2) + \cdots + f(b_N) \tag{2.10}$$

where b_1, b_2, $\cdots$, b_N are the sliced blocks in $\mathcal{F}$.

For example, Figure 2.6(b) shows an instance, $\mathcal{F}$, of slicing floorplan partitioned into 6 ($= N$) grid blocks. The notations $*_i$ and $+_j$ respectively represent the vertical and horizontal slicing orders to produce $\mathcal{F}$. Specifically, $+_i$ and $*_j$ in Figure 2.6(b) represent the *i-th* horizontal cut and the *j-th* vertical cut, respectively, for two slicing cuts of the same type, the slicing cut with higher index must be placed on the right or bottom side of the slicing cut with the lower index.

Then, we can make a one-to-one correspondence between a solution, $\mathcal{S}$, of the CSAP problem instance and a solution, $\mathcal{F}$, of the SFO problem instance by setting $C'(\mathcal{F})$ to $C(\mathcal{S})$. Thus, the remaining issue for transforming an instance of CSAP problem into an equivalent instance of SFO problem is to derive a clock spine implementation, s_i, corresponding to a block, b_i, in $\mathcal{F}$ that minimizes the cost:

$$f(b_i) = c(s_i) = \alpha_1 WL_{sp}(s_i) + \alpha_2 WL_{st}(s_i) + \beta BA(s_i) \tag{2.11}$$

where $WL_{sp}(s_i)$, $WL_{st}(s_i)$, and $BA(s_i)$ are the length of spine s_i in b_i, total length of the stubs in b_i, and total area of spine buffers in s_i, respectively.

2.5.3 Single-line Clock Spine Allocation and Placement Problem

The problem of single spine allocation and placement for a block is described as:

Problem 3. Single-line clock spine allocation and placement problem: *For a set of clock sinks in a grid block b with their placement information, a buffer library $\mathcal{B}$, and parameters B_{stub}, l_{bound} and τ_{min}, generate a clock spine in b by (i) allocating a clock spine s whose maximum difference between the stub lengths not longer than B_{stub}, (ii) placing s and allocating stubs whose longest*

length is not longer than l_{bound}, (iii) allocating buffers from $\mathcal{B}$ and placing them on s, satisfying the tolerance constraint of delay variation τ_{min} such that the quantity of (2.11) *is minimized.*

We compute, for every grid block candidate b, its spine allocation cost by solving the single-line spine allocation and placement problem, performing the following three steps:

1.1 *Spine allocation*: The placement of spine determines the allocation of stubs. It affects the total length of stub wires, the maximum difference between stub lengths and the maximum length of stub wire in a spine s_i. Thus, the spine allocation method is proposed which leads to minimum length of stub wires while meeting l_{bound} and B_{stub} constraint.[1]

The position of a spine is restricted by l_{bound} and B_{stub} constraints. Suppose a vertical clock spine is allocated in the cluster cl with clock sink placement in Figure 2.7. To determine the position of the vertical spine, the coordinates of clock sinks are projected to x-axis because the lengths of stubs in one vertical spine are determined by x-coordinates of clock sinks and that of corresponding spine wire, x_{sp}. Let each x-coordinate of clock sink is indexed from 1 to $|cl|$, where $|cl|$ means the number of clock sinks in the current cluster. Let the x-coordinate of the leftmost clock sink, x_{s_1}, be L and the rightmost sink, $x_{s_{|cl|}}$, be R.

Then, x_{sp} is restricted by (2.12) to satisfy l_{bound} constraint.

$$\begin{aligned} x_{sp} &\geq R - l_{bound} \\ x_{sp} &\leq L + l_{bound} \end{aligned} \tag{2.12}$$

[1]In fact, the objective is simultaneously optimizing the cost in (2.11). However, minimizing wirelength reduces total loading capacitance, which possibly leads to minimizing total buffer area. Therefore, optimizations for wirelength and buffer area conducted independently.

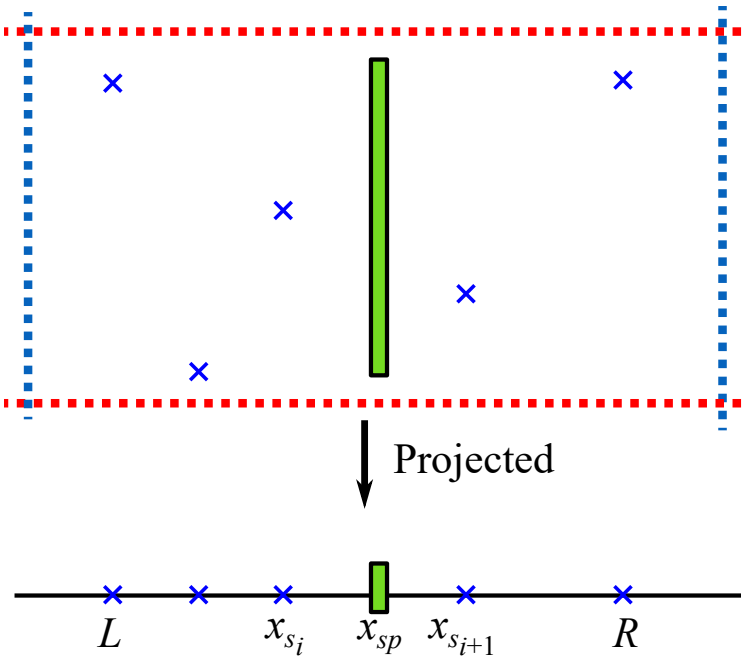


Figure 2.7: Example to illustrate deciding the position of clock spine for a cluster. The x-coordinates of clock sinks are projected to calculate the position of vertical clock spine in the cluster.

For horizontal clock spine placement, the restriction of y-coordinate of clock spine is:

$$\begin{aligned} y_{sp} &\geq U - l_{bound} \\ y_{sp} &\leq L + l_{bound} \end{aligned} \tag{2.13}$$

where U is y-coordinate of the top-most sink and L is that of the lowest sink in the cluster.

x_{sp} (y_{sp}) is also restricted by B_{stub} constraint. Let $M = (L+R)/2$. ($M = (U+L)/2$ is a counterpart for the horizontal spine.) Then, the length of the longest stub, L_{stub}^{max}, of vertical spine is

$$L_{stub}^{max} = \begin{cases} R - x_{sp}, & \text{if } L \leq x_{sp} \leq M \\ x_{sp} - L, & \text{if } M \leq x_{sp} \leq R \end{cases} \tag{2.14}$$

and L_{stub}^{max} for the horizontal spine is

$$L_{stub}^{max} = \begin{cases} U - y_{sp}, & \text{if } L \leq y_{sp} \leq M \\ y_{sp} - L, & \text{if } M \leq y_{sp} \leq U \end{cases} \tag{2.15}$$

When the x-coordinates of clock sinks are indexed like Figure 2.7, x_{sp} can exist within range $x_{s_i} \leq x_{sp} \leq x_{s_{i+1}}$ where i is an integer within range $1 \leq i \leq |cl|$. The lengths of shortest stub, L_{stub}^{min}, for vertical spine is

$$L_{stub}^{min} = \begin{cases} x_{sp} - x_{s_i}, & \text{if } x_{s_i} \leq x_{sp} \leq \dfrac{x_{s_i} + x_{s_{i+1}}}{2} \\ x_{s_{i+1}} - x_{sp}, & \text{if } \dfrac{x_{s_i} + x_{s_{i+1}}}{2} \leq x_{sp} \leq x_{s_{i+1}} \end{cases} \tag{2.16}$$

Then, the maximum difference between stub wires $\Delta_{stub}(x_{sp}) = L_{stub}^{max} -$

L_{stub}^{min} is obtained from the four cases in (2.17) for vertical spine:

$$\Delta_{stub}(x_{sp}) = \begin{cases} R - x_{s_i} - 2x_{sp}, & \text{if} \quad x_{sp} \leq M \quad and \\ & x_{s_i} \leq x_{sp} \leq \dfrac{x_{s_i} + x_{s_{i+1}}}{2} \\ R - x_{s_{i+1}}, & \text{if} \quad x_{sp} \leq M \quad and \\ & \dfrac{x_{s_i} + x_{s_{i+1}}}{2} \leq x_{sp} \leq x_{s_{i+1}} \\ x_{s_i} - L, & \text{if} \quad x_{sp} \geq M \quad and \\ & x_{s_i} \leq x_{sp} \leq \dfrac{x_{s_i} + x_{s_{i+1}}}{2} \\ 2x_{sp} - L - x_{s_{i+1}}, & \text{if} \quad x_{sp} \geq M \quad and \\ & \dfrac{x_{s_i} + x_{s_{i+1}}}{2} \leq x_{sp} \leq x_{s_{i+1}} \end{cases} \tag{2.17}$$

In addition, the minimum value of $\Delta_{stub}(x_{sp})$ is obtained at $x_{sp} = M$. The range of x_{sp} constrained by B_{stub} can be determined by the inequality:

$$\Delta_{stub}(x_{sp}) \leq B_{stub}. \tag{2.18}$$

Because Δ_{stub} is continuous function for x_{sp}, the lower bound and upper bound of x_{sp} can be explored using intermediate value theorem [39] using (2.19).

$$\begin{cases} (\Delta_{stub}(x_{s_i}) - B_{stub}) \cdot (\Delta_{stub}(M) - B_{stub}) \leq 0 \\ (\Delta_{stub}(x_{s_i+1}) - B_{stub}) \cdot (\Delta_{stub}(M) - B_{stub}) \leq 0, \end{cases} \quad \text{if} \quad x_{s_i} \leq M \leq x_{s_{i+1}}$$

$$(\Delta_{stub}(x_{s_i}) - B_{stub}) \cdot (\Delta_{stub}(x_{s_{i+1}}) - B_{stub}) \leq 0, \quad \text{otherwise.} \tag{2.19}$$

If the *acceptable range* is obtained from (2.12) and (2.18), the vertical clock spine can be placed while meeting B_{stub} and l_{bound} constraints. All counterparts of (2.16) $\sim$ (2.19) for horizontal spine can be similarly derived.

If the *acceptable range* is not considered, x_{sp} (or y_{sp}) is set to the median within range $[x_1, x_{|cl|}]$ (or $[y_1, y_{|cl|}]$) in order to minimize the total stub wirelength. Let the median of x-coordinates (y-coordinates) of clock sinks be x_{med} (y_{med}). Thus, if x_{med} is contained in the acceptable range, x_{sp} is set to x_{med}. Otherwise, x_{sp} is set to the nearest value from the acceptable range to minimize the total stub wirelength.

The length of clock spine is determined by clock sinks at the two end of clock spine. Then, the spine will be extended to provide independent clock paths for the clock sinks at the two ends of clock spine. It will be elaborated later.

1.2 *Stub allocation*: After the allocation and placement of clock spine, stub wires can be placed simply by connecting clock sinks to clock spine with straight wires.

1.3 *Buffer allocation/placement*: In this stage, buffers are allocated based on loading capacitance to meet the clock slew constraint. Precisely, the spine buffer allocation problem is to determine the ranges on a spine s and the types of buffers in $\mathcal{B}$ to be placed in the ranges so that all the capacitances of the spine wire in s and the stubs in s, and the inputs to the sinks are covered (i.e., driven by buffers to satisfy the clock slew constraints) by the allocated buffers while minimizing the $BA(s)$ cost in (2.11). With the buffer library $\mathcal{B} = \{buf_1, buf_2, \cdots, buf_B\}$, spine buffers can be placed on any location in the spine. Each buffer in $\mathcal{B}$ has maximum driving capacitance constraint, c_{buf_i}, to meet the output slew limitation. Then, to find the optimal-cost buffer allocation and placement, we used a dynamic programming approach proposed in [12] by segmentizing the spine in very short intervals. We recursively find an optimal buffer allocation and place-

ment of a portion of spine by exploring the set of optimal ones of smaller spines that were obtained in the previous iterations.

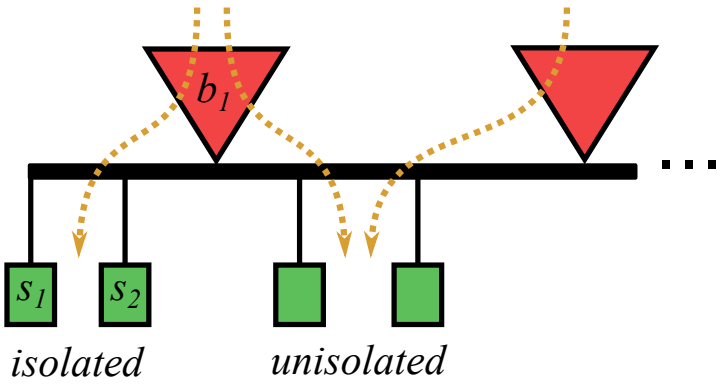
2.5.4 Other Considerations for Single Clock Spine Allocation and Placement

• *Spine extension*: One main benefit of using spine structure rather than tree structure is the improvement of the tolerance of clock latency variation by creating more than one clock signal path from the clock source to each sink. However, the allocation and placement of spine buffers may cause some sinks to be excluded from having the benefit of tolerating delay variation. As an illustration, Figure 2.8(a) shows a leftmost section of a spine with buffer allocation in which two clock sinks s_1 and s_2 would suffer the clock delay variability more than the other sinks, which are driven by multiple clock paths.

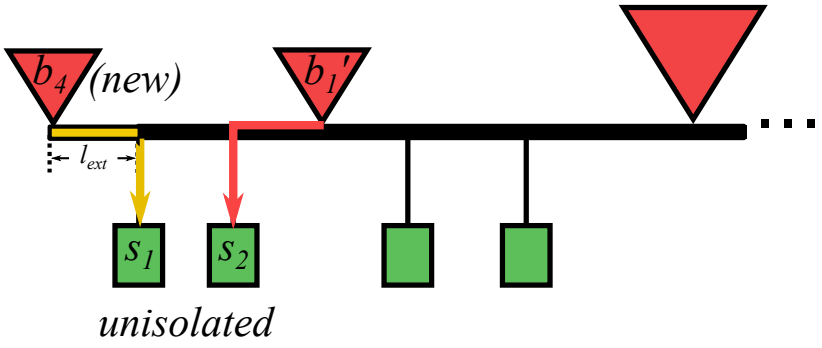
We call such clock sinks which are vulnerable to latency variation in a spine *isolated clock sinks*. The objective of this refinement step is to convert all isolated clock sinks in clock spines to unisolated ones so that the tolerance to delay variation to every sink are uniformly enhanced. To remove the isolated clock sinks, we propose *spine extension* method. Let the distance between the leftmost (rightmost) buffer and the nearest isolated clock sink from that buffer on the left (right) side be D_{min} and let the distance between the leftmost (rightmost) point of clock spine and the nearest isolated clock sink from the leftmost (rightmost) point of clock spine on its side be d_{min}. To balance the clock latencies of isolated clock sinks, the leftmost (rightmost) portion of clock spine is extended by l_{ext}:

$$l_{ext} = D_{min} - d_{min} \tag{2.20}$$

If $l_{ext} \leq 0$, then the spine is slightly extended to make a space for the new buffer.



(a) A clock spine with isolated clock sinks s_1 and s_2.



(b) Spine extension to convert the isolated sinks in (a) to unisolated ones.

Figure 2.8: Clock spine extension to improve the variation tolerance. (a) A clock spine that contains isolated clock sinks. (b) Clock spine is extended, resulting in placing a new buffer (b_4) to drive s_1 and s_2.

After extension, a new buffer is inserted that is enough to drive the extended portion of clock spine and isolated clock sinks. For example, in Figure 2.8(b), the lengths of red and yellow arrows correspond to D_{min} and d_{min}, respectively.

Given the driving strength of two buffers which are nearest to the isolated clock sinks, the extended portion of clock spine is not long enough to make the clock latency larger. Rather, it should be noted that the shortest clock latency become much smaller. Thus, the l_{ext} value is obtained based on the shortest path lengths from spine buffer to isolated sinks.

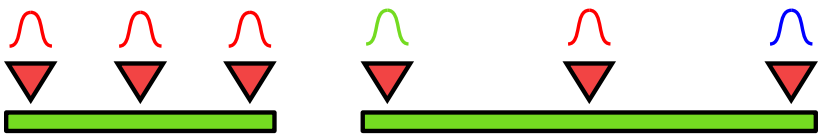
• *Spine directions*: For each option of vertical and horizontal directions, we perform our three-step spine allocation algorithm and compute $c(\cdot)$ in (2.11). We then choose the spine allocation and placement with the smaller one of $c(\cdot)$ values.

• *Tolerance metric of delay variation*: We proposed a tolerance metric, $\tau(s_i)$, for global clock skew variation for a spine segment s_i:

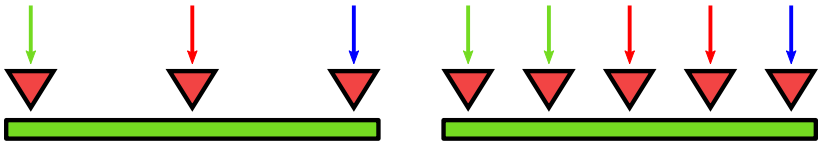
$$\tau(s_i) = \alpha_1 \cdot buf_{dist}(s_i) + \alpha_2 \cdot buf_{num}(s_i) \tag{2.21}$$

where $buf_{dist}(s_i)$ indicates the maximum distance among the spine buffers on s_i and $buf_{num}(s_i)$ represents the number of spine buffers on s_i. α_1 and α_2 are weighting factors.

As the value of $\tau(\cdot)$ for a spine is larger, it is more likely for the spine to tolerate the delay variation. For example, in Figure 2.9(a), process variation has the spatial correlation, short clock spine tends to be suffered by local process variation compared to longer clock spine wire. Also, as shown in Figure 2.9(b), the more spine buffers inserted in one spine, the more redundant paths exist, which provide more redundant paths. We want to make sure that the constraint $\tau(\cdot) \geq \tau_{min}$ is met for all spines.



(a) One example to compare the spine with different length.



(b) The more spine buffers are connected to the spine, the more redundant paths exist in that spine, which make the spine more tolerant to delay variation.

Figure 2.9: Examples to illustrate the tolerance metric for a spine segment.

2.6 Application of Slicing Floorplan Optimization Algorithm

In this section, we solve the instance of the SFO problem which is transformed from an instance of the CSAP problem. We can use any slicing floorplan optimization algorithm which is best suited for our design objectives and constraints. Here, we adopt the postfix expression based simulated annealing (SA) [40] method proposed by Wong and Liu [41].

2.6.1 Move Operations

Figure 2.10(a) shows an example of floorplan with 5 sliced blocks and corresponding binary tree representation, in which, the horizontal and vertical cuts (i.e., operators) are expressed by $*_i$ or $+_j$, and the sliced blocks are denoted by bullets. The binary tree in Figure 2.10(a) can be uniquely expressed into a postfix expression $\bullet\bullet\bullet *_1 +_2 \bullet\bullet *_4 +_3$ by performing the post-order traversal

on the tree.

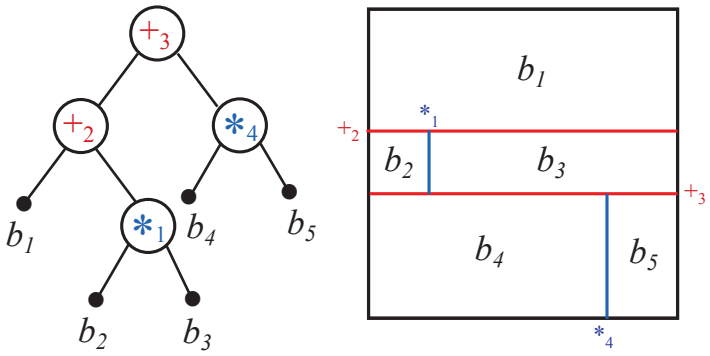
With the one-to-one correspondence between CSAP problem instances and SFO problem instances, the exploration of the various initial clock spine networks can be accomplished through consecutive changes in a postfix expression. The move operations to be used in the SA process are:

- *move-1*: Change the index of an operator: $*_i \rightarrow *_j$ or $+_i \rightarrow +_j$.
- *move-2*: Complement an operator: $* \rightarrow +$ or $+ \rightarrow *$.
- *move-3*: Switch positions of operators: $*_i \leftrightarrow +_j$, $*_i \leftrightarrow *_j$, or $+_i \leftrightarrow +_j$.
- *move-4*: Swap an operator with a bullet: $\{*, +\} \leftrightarrow \bullet$.

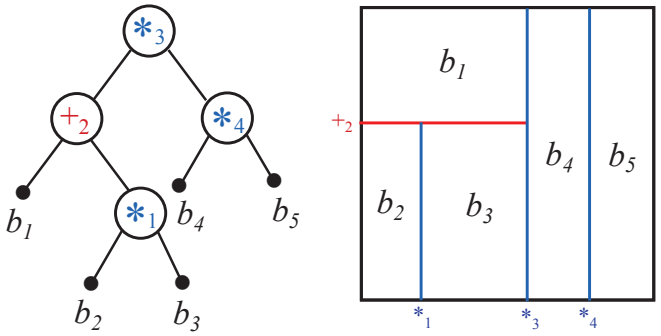
Figure 2.10(b) then shows the postfix expression updated by applying *move-2* to $+_3$ in Figure 2.10(a) and the corresponding floorplan. The operator $+_3$ at the root node changed to $*_3$. In the corresponding floorplan, long horizontal cut, $+_3$, is replaced by long vertical cut, $*_3$.

Meanwhile, Figure 2.10(c) shows the postfix expression updated by applying *move-3* to $*_1$ and $*_4$ and the resulting floorplan, which corresponds to switching the positions of operators $*_1$ and $*_4$.

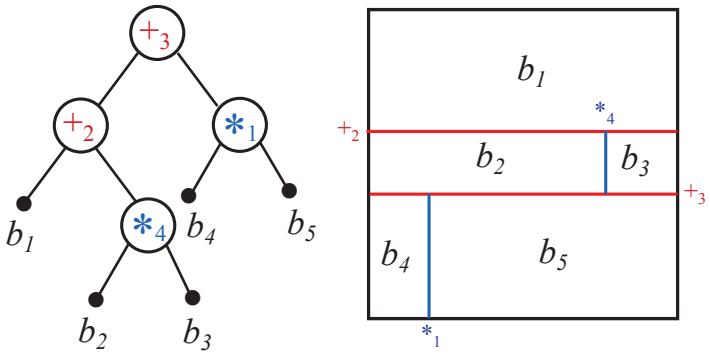
Since every legal move does not change the total number of operators in the postfix expression, the number of spine constraint ($= N$) is always maintained. It should be noted that the reason why we introduce parameter N in CSAP problem is that without the constraint, the one-to-one correspondence between CSAP problem instances $\mathcal{S}$ and SFO problem instances $\mathcal{F}$ is not well established. To explore the clock spine network with the different number of spine wires, we employ a spine-based recursive bipartitioning, which will be described in Sec. 2.8.



(a) Postfix and floorplan for the current state.



(b) Postfix and floorplan updated by the application of *move-2* to $+_3$ in (a).



(c) Postfix and floorplan updated by the application of *move-3* to $*_1$ and $*_4$ in (a).

Figure 2.10: Example showing the application of *move* operations to postfix expression.

2.6.2 Rules for Legal Postfix Expressions

The postfix expressions we have manipulated are required to meet a couple of rules to hold the one-to-one correspondence between a postfix expression and a solution of SFO problem.

- *Balloting sequence*
- *Positional relation validity*

The description of the balloting sequence rule is in [41]. If the rule is violated, the binary tree corresponding to the postfix expression cannot be constructed, thus not being transformed into a solution instance of CSAP problem. [41] claims that the balloting property can be tested in $O(1)$ time. In addition, the rule of positional relation validity guarantees that one postfix expression can be transformed into a solution of SFO problem. For example, by recalling $+_i$ and $*_j$ in Figure 2.6(b) represent the *i-th* horizontal cut and the *j-th* vertical cut, for two slicing cuts of the same type, the slicing cut with higher index must be placed on the right or bottom side of the slicing cut with the lower index. To represent this positional relation in a binary tree, the left and right subtrees of every node on the tree should satisfy the following properties.

Property 1. If a node $+_j$ $(*_j)$ is on the left subtree of node $+_i$ $(*_i)$, it should be $j < i$.

Property 2. If a node $+_j$ $(*_j)$ is on the right subtree of node $+_i$ $(*_i)$, it should be $j > i$.

Property 3. All leaf nodes are bullets (i.e., primitive sliced blocks).

For example, in Figure 2.11, the left subtree of node $*_2$ corresponds to blocks (b_1, b_2, b_3) and two slicing cuts $(+_3, *_4)$ that are on the left side of slicing cut $*_2$. However, node $*_4$ makes this binary tree to be infeasible because cut $*_4$ cannot

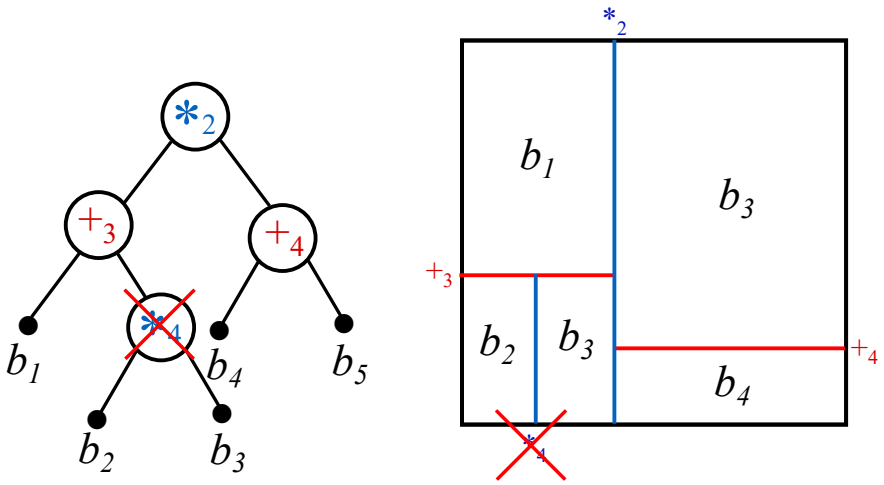


Figure 2.11: Example of an invalid binary tree for constructing legal slicing floorplan. The slicing cut $*_4$ should be on the right side of $*_2$.

be placed on the left side of cut $*_2$. The index of node $+_4$ is not constrained by $*_2$ because their operator types are different.

To check the positional relation validity for each time when a move operation in simulated annealing is performed, the resulting binary tree has to be constructed to examine the node indices. The construction of binary tree from postfix expression is done by performing the following steps, scanning the expression from right to left.

- *Step-1*: Set the rightmost literal as the root of the binary tree and mark the root.
- *Step-2*: Scan forward to the left direction to get one literal. If there is no literal, stop the process.
- *Step-3*: If the visited node doesn't have the right child, set the scanning

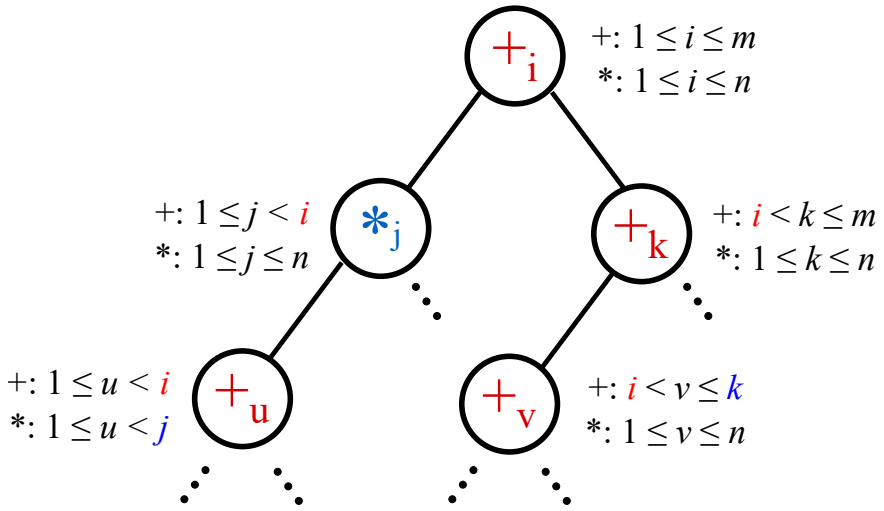


Figure 2.12: The indices of the binary tree can be examined by propagating the index constraints (in the properties) from root to leaf nodes.

literal in *Step-2* to the right child of the visited node and move on to the right child. Then, go to *Step-2*. If the visited node already has the right child and doesn't have the left child, set the scanning literal to the left child of the visited node and move on to the left child. Then, go to *Step-2*. If the visited node already has two children or is bullet, visit its parent node and repeat *Step-3*.

The three steps of constructing the binary tree from postfix expression of length l take $O(l)$ time.

The indices of nodes in the binary tree can be examined by propagating index constraints from root to leaf nodes, as illustrated in Figure 2.12. This examination takes a linear time as well.

2.6.3 Cost Function for Simulated Annealing

The cost function evaluated at each solution instance of SA process is $C(\cdot)$ in (3.3). To speed up the cost computation, we pre-compute an optimal spine cost (i.e., $c(\cdot)$) for every possible shape of blocks and store the cost in a cost table (in a form of hash table). Even though theoretically the number of block candidates is bounded by $O(m^2n^2)$ for $m \times n$ grids, the constraints (e.g., B_{stub} and l_{bound}) limit the number of candidate blocks to be explored in a practically controllable number. To determine an initial temperature T_0 for SA, we perform a sequence of random moves and calculate the cost changes in uphill moves. By evaluating the average cost change (Δ_{avg}) in uphill moves, we set T_0 to $e^{-\Delta_{avg}/T_0} = P \simeq 1$.

2.6.4 The Starting Values of N, m, n

The number of clock spines for an initial clock spine network is computed by our NN based algorithm in Sec. 2.4. The value of grid size $m \times n$ used for deriving an initial clock spine allocation is computed by

$$m = \left\lceil \frac{Ckt_w}{\frac{l_{bound}}{k}} \right\rceil, \qquad n = \left\lceil \frac{Ckt_h}{\frac{l_{bound}}{k}} \right\rceil \tag{2.22}$$

where Ckt_w and Ckt_h indicate the width and height of input circuit, respectively. l_{bound} is the upper bound of the length of stub wires to be met and k is a number in interval $[2, 4]$ that can be used by designer to trade runtime with quality of spine allocation.

2.7 Derivation of Solution for Clock Spine Network

From the postfix expression which has the least cost $C(\cdot)$ in (3.3), the corresponding slicing floorplan is obtained, from which a solution of spine placement and allocation is derived by synthesizing single-line clock spines for all sliced blocks.

2.8 Spine-based Recursive Bipartition

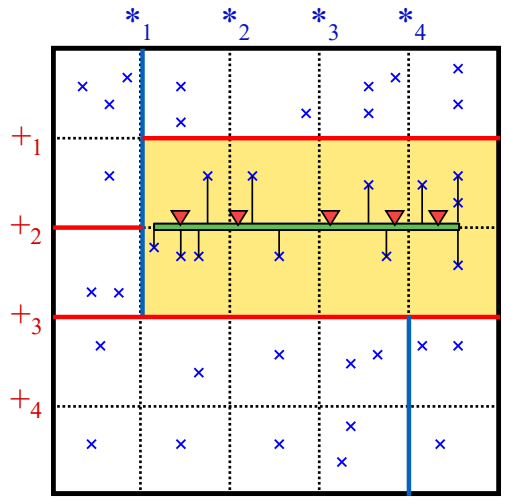
By incrementally updating the clock spine network produced by using the initial N and $m \times n$ values, we explore clock spine network structures while 'locally' increasing the values of N, m and n. Precisely, for the circuit block corresponding to a spine, we examine if there exists an allocation of two spines which satisfying the tolerance constraint τ_{min} with lower total cost by applying our spine allocation algorithm to the sinks on the block while doubling the size of $m \times n$ of the block. If there is, we replace the spine with the two-spine allocation of minimum cost. We then repeat this refinement process for each of the two spines until there is no bipartition result which can satisfy τ_{min} constraint or reduction on the cost.

This local bipartition approach is intended to manage the runtime, otherwise explosive, during the exploration of spine networks by varying the values of N, m, and n.

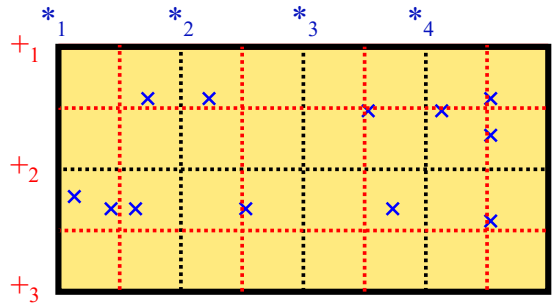
An example of recursive bipartition process is illustrated in Figure 2.13.

2.9 Refinement of Clock Spine Network

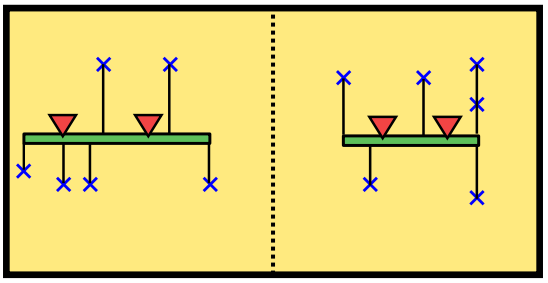
As a last step of our allocation algorithm, we perform so called *spine merging*. We attempt to connect multiple spines that are close to each other. The benefit is to improve the tolerance of delay variation at the expense of a little increase of spine wire. For two spines with distance less than a predefined threshold, we connect the spines and resize nearest buffers accordingly based on driving strength of nearest buffers. Figure 2.14 shows an example.



(a) A clock spine to be bipartitioned.



(b) Doubling the number of grids m and n for target sliced block.



(c) Found bipartition with minimum cost. Each bipartitioned spine meets the tolerance constraint.

Figure 2.13: Example showing the spine-based recursive bipartition.

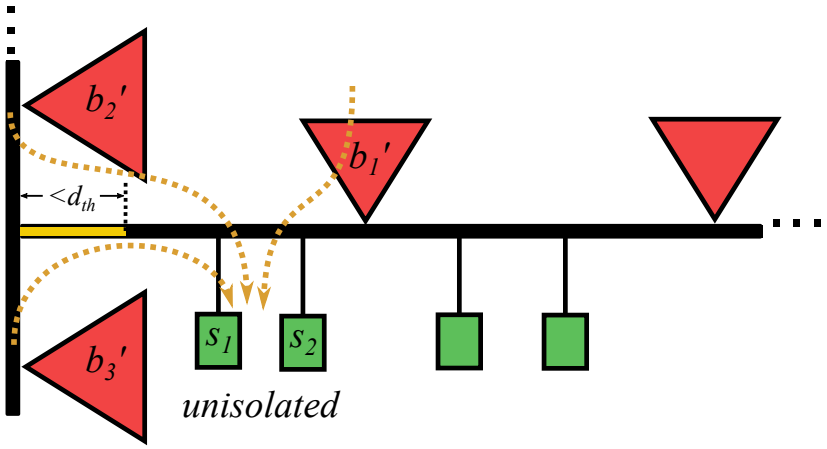


Figure 2.14: Two clock spines are merged with resized buffers b'_1, b'_2 and b'_3 to drive s_1, s_2 in a spine as well as some sinks in the other spine.

2.10 Experimental Results

2.10.1 Experimental Environments

Our proposed clock spine synthesis algorithm has been implemented with C++ and Python 3.4, and used HSPICE simulation to validate the effectiveness of clock spine networks synthesized by our algorithm. The simulation is performed by Linux machine with 8 cores of 2.67GHz Intel Xeon CPU and 52GB memory.

We tested our algorithm using ISPD2010 [9] benchmark circuits. To be a complete test for the clock spine networks synthesized by our algorithm, we implement a top-level binary clock tree to drive the clock spine buffers by applying the nearest neighbor clock tree synthesis algorithm in [11], in which spine buffers are regarded as sinks. We compared our clock spine networks with the clock tree networks synthesized by [11] and the clock mesh networks by [10]. Top-level binary clock trees for clock mesh networks also implemented for

complete test.

Technology parameters are based on 45nm predictive technology model [42]. We used the clock frequency of 2GHz and the supply voltage of 1.0V, while constraining maximum transition time to 50ps (10% of the clock period.) The unit resistance and unit capacitance of wire are set to $0.1\Omega/\mu m$ and $0.2fF/\mu m$, respectively. We use 20 buffers of different sizes with the maximum driving capacitance ranging from 19.88fF to 170.7fF under maximum slew constraint of 50ps.

When synthesizing clock spine networks, the values of α_1, α_2, and β in (2.11) are set to $0.025/\mu m$, $0.025/\mu m$, and $1/\mu m^2$ respectively. And the constraints B_{stub} and l_{bound} are set to $150\mu m$, $150\mu m$ respectively. The values of τ_{min}, α_1 and α_2 in (2.21) are set to 800, $1/\mu m$, and $10/\mu m^2$. For clock mesh synthesis, the length of mesh grid is constrained within range $[100\mu m, 150\mu m]$.

Table 2.1 summarizes the number of clock sinks in benchmark circuits in [9] and the configurations of clock mesh networks and clock spine networks. Each entry in the second column of Table 2.1 represents the number of clock sinks. The data in columns labeled *Grid* indicate the configuration of mesh grid. The *Init* and *Final* columns in the *Spine* column means the number of clock spines obtained by NN based clustering approach in Sec. 2.4 and recursive bipartition in Sec. 2.8, respectively.

We performed 100 times of Monte Carlo runs for each synthesized clock network in HSPICE simulation with random parameters for the wire width, the channel length of buffers, and the clock input capacitance of clock sinks in order to reflect the process variation environment. These random variables consist of nominal value and normally distributed variation. That is, $x_i \sim x_i^0 + N(0, \sigma_i^2)$ where x_i^0 is nominal value of random variable x_i and σ_i is set to 10% of nominal value. We also assume a spatial correlation on each parameter for which we

Table 2.1: The number of clock sinks in benchmark circuits in [9] and the configurations of clock mesh networks [10] and clock spine networks.

Circuit	# Sinks	Mesh	Spine	
		Grid	Init	Final
03	1200	22 × 12	5	14
04	1845	20 × 20	13	23
05	1016	20 × 20	9	18
06	981	19 × 10	6	14
07	1915	20 × 14	10	26
08	1134	20 × 20	7	17

apply principal component analysis (PCA) [18].

2.10.2 Comparison with Clock Tree Structure

We compare clock tree networks synthesized by [11] and clock spine networks by our algorithm. Top-level binary clock trees for clock spine networks are also synthesized by [11] for complete test.

• Comparison Results under Process Variations

Table 2.2 shows the comparison of the results produced by the clock tree synthesis algorithm in [11] and our clock spine synthesis algorithm. The data in the columns labeled *Skew*, WL, BA, and PWR indicate the global clock skew, total clock wire length, total clock buffer area, and total power consumption, respectively. The μ and σ columns in the *Skew* column means the average global clock skew and its standard deviation value calculated from Monte Carlo simulation of each clock distribution network. The last row shows the comparison of

the average (normalized) improvements of clock spine networks over the clock tree networks.

In comparison with clock tree networks, our clock spine networks decrease the clock skew by 30% while using 4% less wirelength and 21% more buffering area. The clock skew induced by process variation is reduced due to redundant clock signal paths. The average wire usage of clock spine is comparable to clock tree structure due to the wire snaking in zero-skew tree. The average power consumption of clock spine networks is 4% less than clock tree networks. These results indicate that the proposed clock spine synthesis algorithm can synthesize the clock spine networks which are more tolerant to process variations than clock tree networks with comparable wire, buffer area and power consumption.

• Comparison Results without Process Variations

We compared the global clock skew of clock tree networks and clock spine networks which are not suffered from the effects of process variations. Table 2.3 shows the experimental results of the same clock networks which are generated for the experiments in Table 2.2. The data in each column indicate the global clock skew of zero-skew clock tree networks synthesized by [11] and that of clock spine networks.

The average global clock skew of clock tree networks is 5.38ps. Non-zero skew results from HSPICE simulations are due to inherent inaccuracy and modeling error of the Elmore delay model and buffer delay mode. The average clock skew of clock spine networks is 8.33ps, which is 2.95ps larger than that of clock tree networks. Due to the fishbone structure of spine network, the difference between lengths of stub wires makes the nominal clock skew larger than skew of zero-skew clock tree networks. By comparing the clock skew values in Table 2.2 and Table 2.3, it can be observed that by limiting the length of stub wires with B_{stub}

Table 2.2: Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock tree networks by [11] and our spine networks under process variations.

Circuit	Clock tree networks by [11]				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	54.29	16.27	59401.3	1235.21	83.61
04	66.88	24.46	115178	1419.57	97.71
05	39.92	16.40	51227.4	622.22	42.86
06	54.16	17.52	67312.7	1027.81	70.59
07	52.28	21.82	102258	1571.67	106.15
08	39.54	14.29	62025.8	1060.07	70.54
Avg.ratio	1	1	1	1	1
Circuit	Our clock spine networks				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	39.71	19.27	41956.0	1286.97	65.47
04	43.05	21.97	112211	1710.32	91.56
05	35.66	18.97	63265.4	868.21	47.90
06	28.93	13.49	51035.7	1166.58	59.56
07	40.70	18.42	104411	1952.12	109.93
08	24.01	11.61	67669.4	1319.67	75.28
Avg.ratio	**0.70**	**0.94**	**0.96**	**1.21**	**0.96**

and l_{bound} constraints, the clock skew caused by the stub length difference can be prevented from increasing more than the clock skew due to process variations.

Table 2.3: Comparison of global clock skew of clock tree networks by [11] and our spine networks without process variations.

Circuit	Global clock skew (ps)	
	Clock tree networks by [11]	Clock spine
03	4.60	8.10
04	5.20	9.90
05	4.30	6.40
06	6.10	9.80
07	6.10	7.00
08	6.00	8.80
Average	**5.38**	**8.33**

2.10.3 Comparison with Clock Mesh Structure

We performed SPICE level simulation for clock mesh networks by [10], and clock spine networks by our algorithm. Top-level binary clock trees are also synthesized for clock mesh networks for complete test.

Table 2.4 shows the comparison of the results of clock mesh synthesis algorithm in [10] and our clock spine synthesis algorithm. In terms of the clock skew, our clock spine networks are almost comparable to the clock mesh ones but suffer clock skew variation. The clock spine networks show 21% increased global clock skew than clock mesh networks in average. Because of the greater number of redundant clock paths, the clock mesh structure is more stable and tolerant to clock latency variation than the clock spine structures.

On the other hand, our clock spine networks use 50% less wire resources 33% less buffering area. Power consumption also decreased by 46% over the clock mesh. In conclusion, with clock spine structure, a certain level of tolerance can be achieved with a low resource and power consumption.

2.10.4 Skew-resource trade-off based on tolerance metric

Tolerance metric τ for spine segment is proposed in (2.21). This tolerance metric is weighted sum of the length of clock spine segment and the total number of buffers in corresponding clock spine segment. If τ becomes large, each synthesized spine segment becomes longer and covers wider area of circuit to satisfy the tolerance constraint. Total number of clock spines would be smaller because long spine wire covers the clock sinks placed in a wide range of area. Longer spine segment is more likely to attenuate the global clock skew in a wide area of circuit while consuming more wire and buffer resources. In contrast, if τ becomes smaller, shorter spine segments would appear in clock spine network. Using short clock spines can reduce the clock resource consumption because shorter spine wires can be placed closer to the small clusters than long clock spine wires, reducing the total wirelength of stub wires. However, the degree of global clock skew attenuation is sacrificed because shorter clock spines can cover flip-flops in a smaller range of circuit area than the longer clock spines.

To evaluate the skew-resource trade-off, we synthesized clock spine networks by changing the τ to 200, 400, and 800. Each synthesized spine networks are merged with top-level zero-skew clock tree to evaluate the full clock networks. Conditions for process variations are same as previous experiments.

Table 2.5 shows the comparison of the results of clock spine networks with different τ values. The clock spine networks with $\tau = 800$ show smallest global clock skew in average compared to other clock spine networks with $\tau = 400, 200$.

Table 2.4: Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock mesh networks by [10] and our clock spine networks.

Circuit	Clock mesh networks by [10]				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	29.08	11.32	107161	1839.87	146.22
04	35.04	18.72	212536	2794.67	232.70
05	31.91	18.97	134654	1607.61	100.08
06	19.46	6.37	94888.8	1496.39	120.40
07	37.10	8.59	184411	2836.98	206.13
08	24.49	4.57	130414.8	1821.85	174.74
Avg.ratio	1	1	1	1	1

Circuit	Our clock spine networks				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	39.71	19.27	41956.0	1286.97	65.47
04	43.05	21.97	112211	1710.32	91.56
05	35.66	18.97	63265.4	868.21	47.90
06	28.93	13.49	51035.7	1166.58	59.56
07	40.70	18.42	104411	1952.12	109.93
08	24.01	11.61	67669.4	1319.67	75.28
Avg.ratio	**1.21**	**1.81**	**0.50**	**0.67**	**0.46**

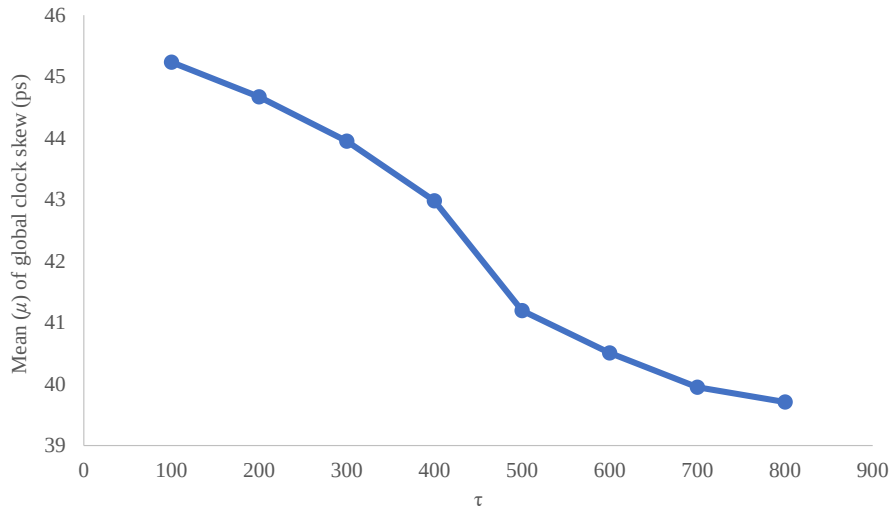
This indicates that the clock spine network with larger τ value are more tolerant to process variation than the clock spine network with smaller τ. On the other hand, clock spines with smaller τ show less clock resource consumption. With changing τ value, design space for clock spine networks can be explored while considering trade-off between global clock skew induced from process variations and clock resource consumption.

Figure 2.15 shows the global clock skew, total wirelength, total buffer area, and power consumption of clock spine networks synthesized for ISPD03 circuit with different τ values (from 100 to 800). These curves also show the trade-off between global clock skew and clock resource consumption. The experimental results for other benchmark circuits with different τ values from 100 to 800 are in Appendix A.

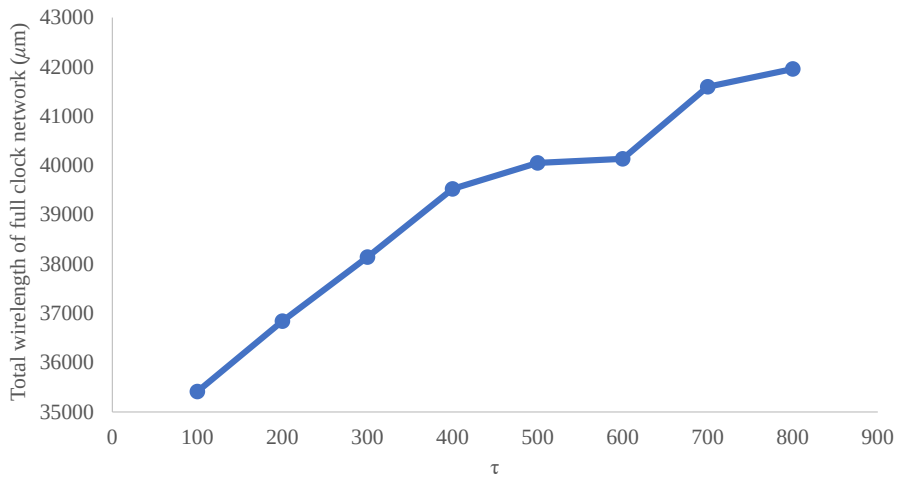
Figure 2.16 shows the synthesized clock spine networks synthesized for benchmark circuit ISPD06 with $\tau = 800$ and $\tau = 200$. Black lines represent the spine wire and stub wires. Red triangles are spine buffers which deliver the clock signal from top-level clock tree to clock spines. Green rectangles represent the placement of clock sinks in benchmark circuit. Shorter spine wires can reduce the lengths of spine wires as well as the lengths of stub wires, reducing total buffer area.

2.10.5 Run Time Analysis

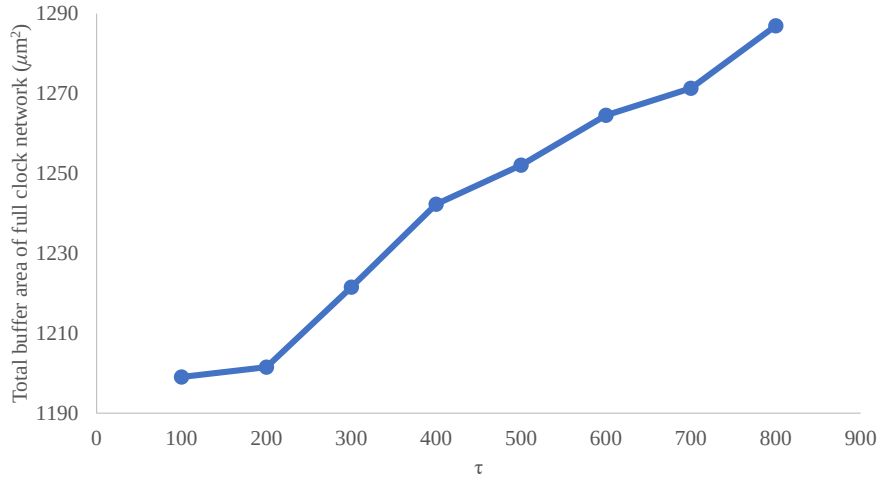
Table 2.6 shows the runtime and starting values for synthesizing clock spine networks consisting of single-line spines. The runtime taken for creating hash table is associated with the values of m, n (= grid size) and N (= the number of spines). Creating hash table to store spine information for every possible rectangle sub-block on $m \times n$ grid takes the longest time among all processes. The runtime for SA process is very short because of the number of literals in



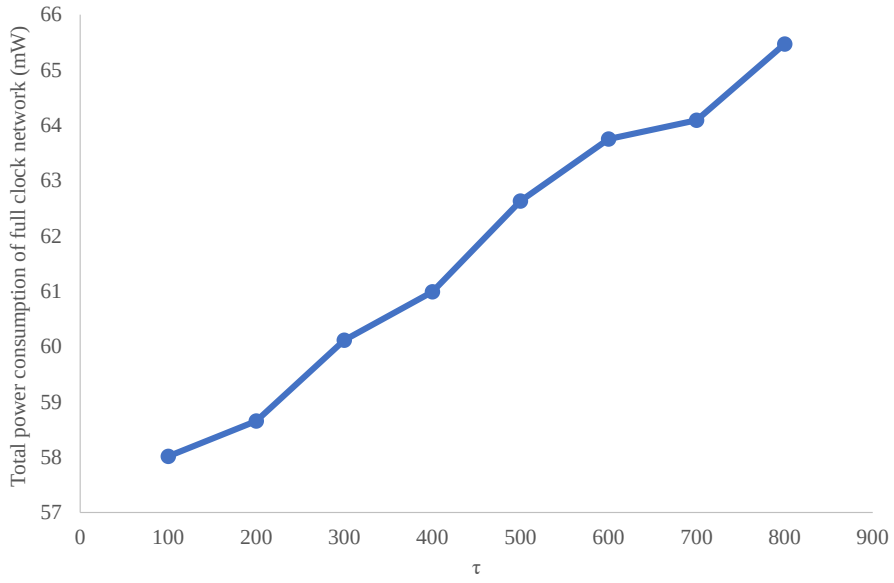
(a) Global clock skew



(b) Total wirelength



(c) Total area of buffers



(d) Total power consumption

Figure 2.15: The global clock skew, total wirelength, total buffer area, and power consumption of clock spine networks synthesized for ISPD03 while changing τ from 100 to 800.

Table 2.5: Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock spine networks with $\tau = 800, 400, 200$.

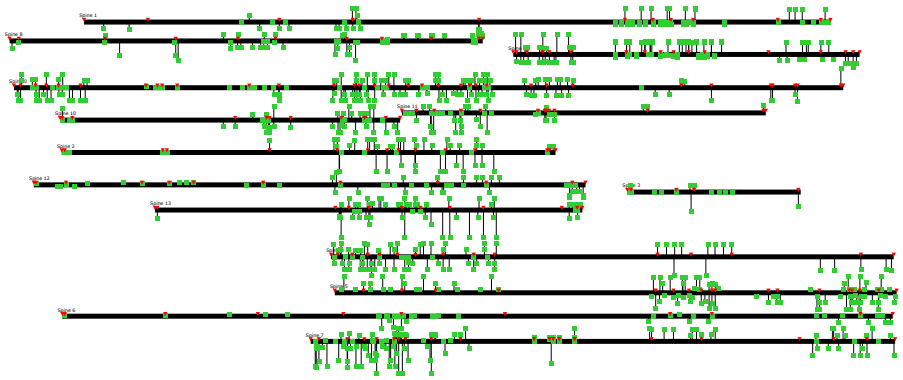
Circuit	$\tau = 800$				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	39.71	19.27	41956.0	1286.97	65.47
04	43.05	21.97	112211	1710.32	91.56
05	35.66	18.97	63265.4	868.21	47.90
06	28.93	13.49	51035.7	1166.58	59.56
07	40.70	18.42	104411	1952.12	109.93
08	24.01	11.61	67669.4	1319.67	75.28
Avg.ratio	1	1	1	1	1
Circuit	$\tau = 400$				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	42.98	20.83	39525.9	1242.28	60.99
04	47.88	22.53	99012.4	1479.73	81.62
05	37.18	18.37	56348.8	747.82	42.69
06	30.85	14.66	46434.3	1058.50	53.71
07	45.51	18.41	93566.3	1722.95	91.68
08	26.08	14.40	55665.6	1167.69	59.86
Avg.ratio	**1.08**	**1.07**	**0.89**	**0.89**	**0.87**

Circuit	$\tau = 200$				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	44.67	21.20	36841.3	1201.50	58.65
04	65.03	24.73	98881.4	1519.36	81.14
05	38.33	18.91	49999.3	698.20	39.04
06	32.22	15.92	42597.9	1038.70	51.84
07	47.48	19.78	82796.4	1605.96	85.41
08	29.65	15.53	51356.9	1112.07	56.90
Avg.ratio	**1.20**	**1.14**	**0.82**	**0.86**	**0.83**

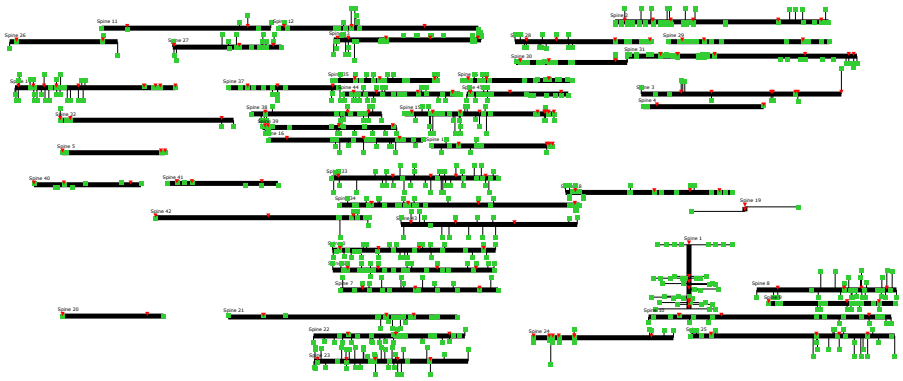
postfix expression is small. If the value of N becomes larger, the number of literals in postfix expression also becomes larger, which makes the runtime for SA process longer. The runtime for recursive bipartitioning is affected by τ_{min} since the N value increases according to the $\tau(\cdot)$ constraint. A small value of τ_{min} tends to increase the value of N during bipartition and it influences the runtime on the recursive bipartition.

2.10.6 The Effect of Eliminating Isolated Flip-Flops

Figure 2.17 shows the delay variation data before and after the application of our clock spine refinement of removing isolated clock sinks. The blue histogram shows the distribution of latency from clock source to sink #624 in circuit ISPD08, which is an isolated sink on the clock spine network produced by our algorithm without refinement. The orange histogram is counterpart produced after our refinement. (The latency values are extracted from 100 Monte Carlo simulations.) The red and blue curves are the normal distributions fitted to



(a) Clock spine network with $\tau = 800$.



(b) Clock spine network with $\tau = 200$.

Figure 2.16: Clock spine synthesis results for benchmark circuit ISPD06 with changing τ.

Table 2.6: Runtime and starting values for synthesizing clock spine networks consisting of single-line spines.

Circuit	Starting values			After bipartition	Runtime (sec.)		
	N	m	n	N	Hash table	S. Annealing	Bipartition
03	5	41	7	14	2503.08	1.99	196.40
04	13	29	36	23	1001.90	57.26	39.70
05	9	31	34	18	792.90	15.75	25.47
06	6	26	12	14	405.62	4.50	31.56
07	10	34	20	26	1810.39	17.35	117.50
08	7	25	22	17	416.23	12.58	50.53

the latency distribution of orange and blue histograms, respectively. The mean value of the latency on the original clock spine network is 166.37ps while the mean value after refinement is reduced to 163.81ps. Furthermore, the standard deviation is also reduced from 22.63ps to 18.94ps.

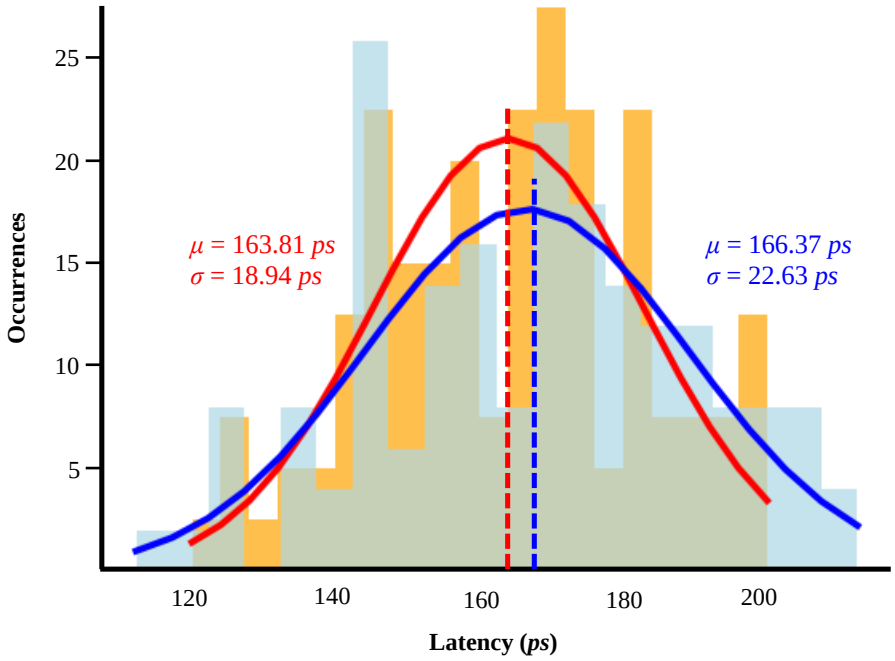


Figure 2.17: Comparison of the variation of arrival times to isolated sink #624 in circuit ISPD08 before and after the application of our refinement. The blue histogram indicates the latency distribution on our clock spine network without refinement while the orange one indicates the distribution on our spine network with refinement.

2.10.7 Analysis of Intermediate Results during Simulated Annealing Process

Finally, Table 2.7 shows the quality of intermediate clock spine networks produced during the SA process on circuit ISPD08. Each intermediate solution is extracted at the corresponding temperature in Table 2.7. Through the SA process using the resource cost in (2.11), many redundant resource allocation is removed while meeting B_{stub}, l_{bound} and τ_{min} constraints. In early stage of SA, wirelength, buffer area, and power consumption vary with no particular trend. However, as the temperature gradually gets down, the wirelength and buffer area as well as power consumption consistently decrease, but the clock skew swings within a certain range.

Table 2.7: Clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of intermediate clock spine networks produced during the simulated annealing process on circuit ISPD08. Each intermediate solution is obtained at a fixed temperature.

Temperature (K)	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
Initial	35.36	16.91	85382.3	1495.55	74.88
100000	31.43	15.68	94990.8	1582.05	79.70
10000	32.92	16.72	83837.7	1492.38	73.71
1000	34.06	16.08	85201.6	1482.02	74.44
100	31.44	17.21	76834.4	1361.48	69.61
Final	27.14	17.24	76012.1	1386.71	69.25

2.11 Summary

This chapter illustrated the benefits of shorting intermediates stages in reducing clock skew with RC network model and Elmore's delay model. The influence of spine capacitance and spine resistance on the clock latency was addressed. Also this chapter addressed the problem of automating the synthesis of clock spine networks, which has been semi-automated or never been automated in academia and industry yet, even though the usefulness and impact have been clearly noticed for a long time. Together with the structure of the clock tree with links, our automation of synthesizing clock spine networks would provide an exploration of diverse structures of hybrid clock networks to trade-off between the clock skew from process variations and clock resources (or power consumption).

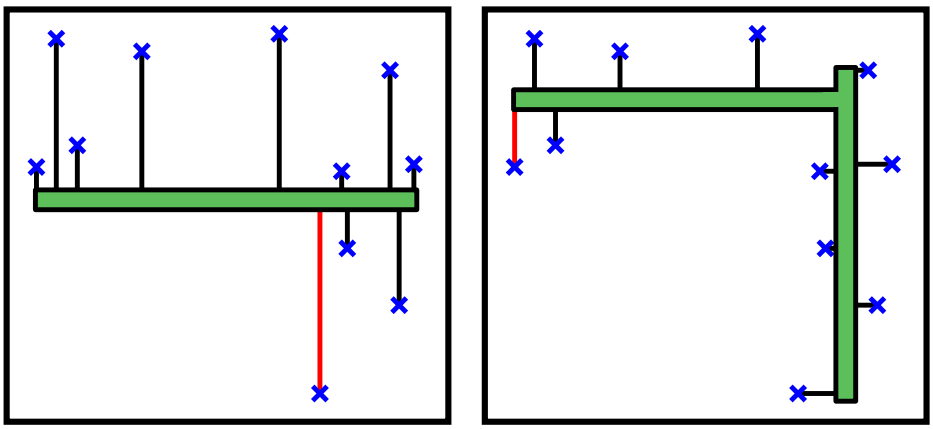
Chapter 3

Extensions of Algorithm for Synthesizing Clock Spine Networks

3.1 Crossed Clock Spine Allocation and Placement

3.1.1 Introduction

A crossed clock spine consists of one horizontal and one vertical spine wire, as shown in Figure 3.1(b). With crossed spine structure, the lengths of spine wires become longer, but the overall lengths of stub wires become shorter, which reduces clock resource consumption and power consumption. Also, if the overall stub length become shorter, the clock skew is likely to decrease because the difference between stub lengths tends to decrease. In order to explore the more diverse clock spine networks, algorithm for synthesizing crossed spine network is proposed in this section.



(a) Single-line spine allocation (b) Crossed spine allocation

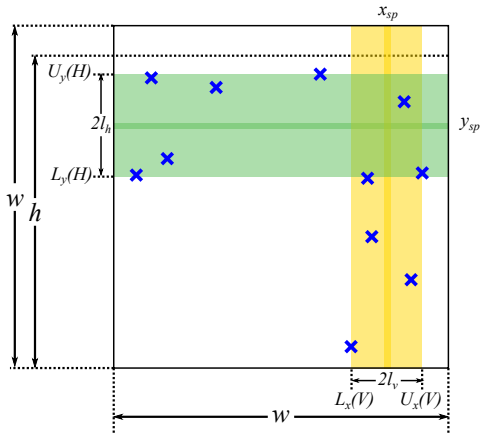
Figure 3.1: Single-line spine and crossed spine structures.

3.1.2 Proposed Method

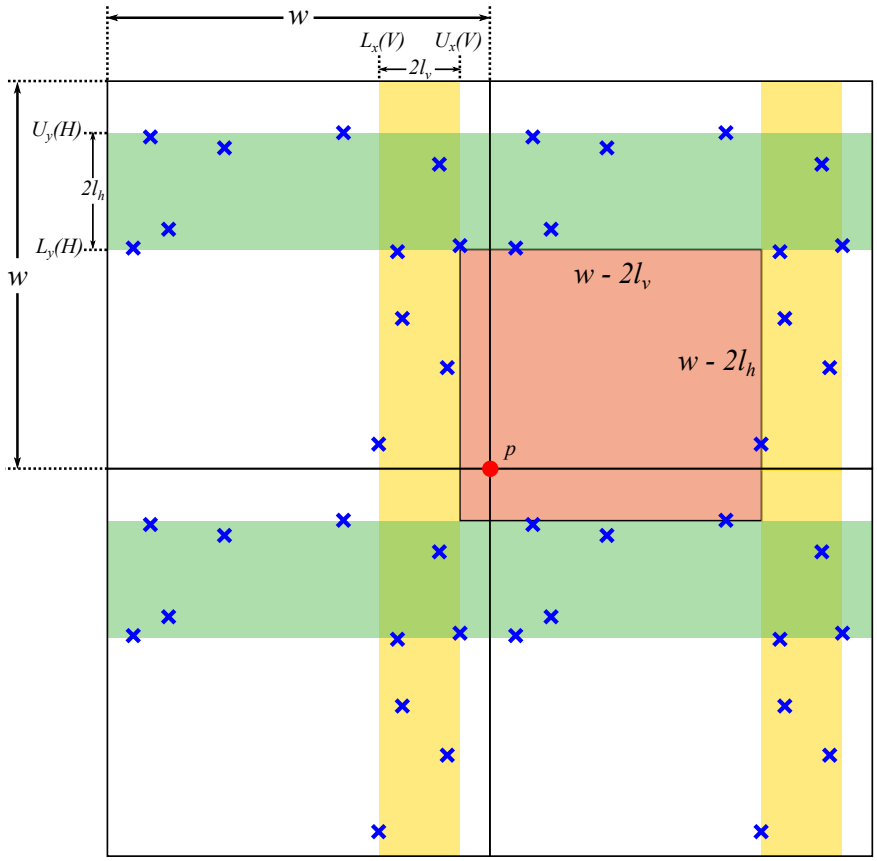
Previously, we have addressed the problem of single-line spines allocation and placement, shown in Figure 3.1(a). We extend our solution to the problem of crossed spines allocation and placement, shown in Figure 3.1(b).

Problem 4. Crossed clock spine allocation and placement problem: *For a set of clock sinks in a grid block b with their placement information, a buffer library $\mathcal{B}$, and parameters B_{stub}, l_{bound} and τ_{min} allocate a cross-shaped clock spine s whose maximum difference between the stub lengths not greater than B_{stub} as well as the longest length of stubs does not exceed l_{bound}, and allocate buffers from $\mathcal{B}$ and place them on s in a way to minimize the quantity in* (2.11)*, while satisfying the tolerance constraint.*

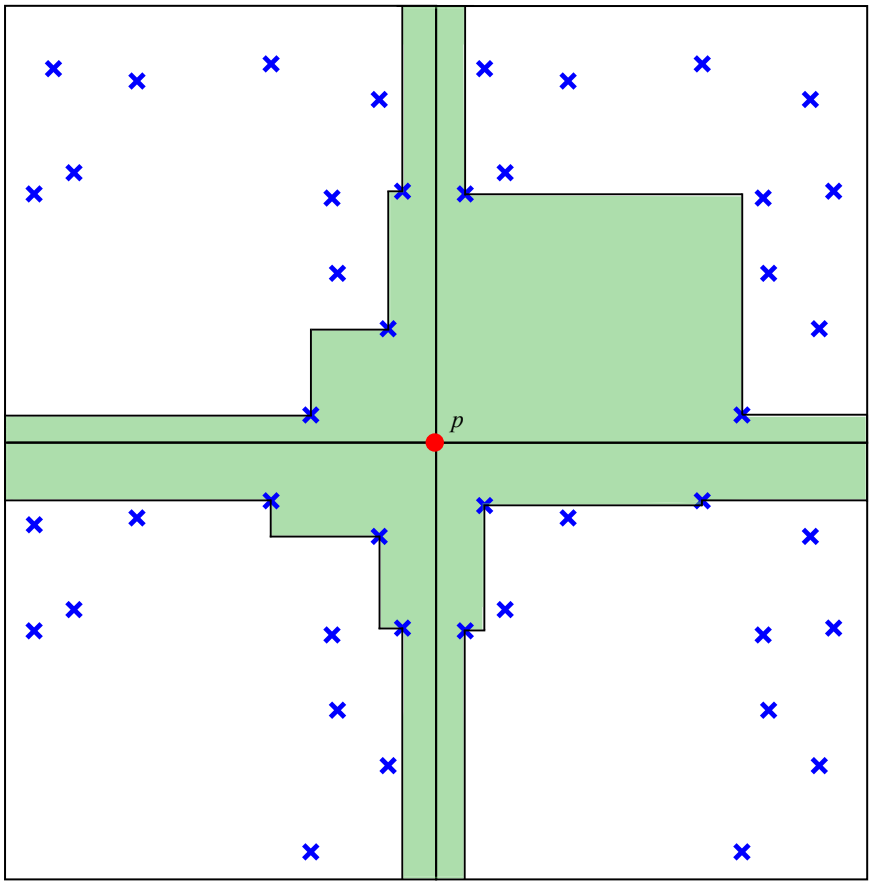
Note that $WL_{sp}(s)$ is the sum of lengths of horizontal and vertical spine wires. We compute the spine allocation cost by solving the crossed spine allocation and placement problem in the following:


x_{sp}
$U_y(H)$
$2l_h$
$L_y(H)$
y_{sp}
w
h
$2l_v$
$L_x(V)$
$U_x(V)$
w

(a)


w
$L_x(V)$
$U_x(V)$
$2l_v$
$U_y(H)$
$2l_h$
$L_y(H)$
w
$w - 2l_v$
$w - 2l_h$
p

(b)



(c)

Figure 3.2: Example of crossed spine placement problem. (a) An instance of crossed spine placement problem of block b. (b) Block b in (a) is replicated and arranged in a 2×2 matrix. (c) A simple rectilinear polygon derived from (a).

2.1 (*Location of horizontal/vertical spine wires*): The objective of using crossed spines rather than single-line spines is to reduce the length of stub wires. By assuming the clock sinks should be connected to the nearest spine wires, the longest stub length can be computed by the x-coordinate of vertical spine wire and the y-coordinate of horizontal wire. Thus, first we need to address the problem of finding H, which is the set of clock sinks connected to horizontal spine, and V, which is the set of clock sinks connected to vertical spine, that potentially lead to a minimal length of the longest stub. If the height of bounding box for H or the width of bounding box for V exceeds $2 \cdot l_{bound}$, the crossed spine cannot satisfy the l_{bound} constraint. Thus, finding H and V is an essential step for synthesizing crossed spines.

Problem 5. Assigning clock sinks to horizontal or vertical wires of crossed spine: *For a set of clock sinks in a grid block b, assign each clock sink to H or V that potentially leads to a minimal length of the longest stub.*

We transform an instance of Problem 5 into an instance of finding a maximum square in the simple rectilinear polygon[1] as follows.

Problem 6. Finding a maximum square in simple rectilinear polygon: *For a simple rectilinear polygon P and a point p, find a square Sq that has the longest edge length, denoted by l_{sq}, in P. In addition, Sq should contain p.*

The transformation process is that: Let $S = \{s_1, s_2, ..., s_n\}$ be the set of clock sinks in a grid block b. Let $(s_{i,x}, s_{i,y})$ be the coordinate of s_i and

[1] A rectilinear polygon is a polygon all of whose edge intersections are at right angles. A *simple* rectilinear polygon is a rectilinear polygon with no hole.

H and V be the sets of clock sinks that are connected to horizontal and vertical spine wires, respectively. Then, the upper and lower bounds of the bounding box for H are:

$$\begin{aligned} U_x(H) &= \max_{s_i \in H}(s_{i,x}) \\ L_x(H) &= \min_{s_i \in H}(s_{i,x}) \\ U_y(H) &= \max_{s_i \in H}(s_{i,y}) \\ L_y(H) &= \min_{s_i \in H}(s_{i,y}) \end{aligned} \tag{3.1}$$

Similarly, the upper and lower bounds for V can be obtained. Because of the B_{stub} and l_{bound} constraints, the horizontal (vertical) clock spine tends to be placed close to the vertical (horizontal) center and median of H (V). Thus l_h, the height of bounding box for H, and l_v, the height of bounding box for V, directly affect the maximum stub length of the horizontal and vertical spine, respectively. We minimize the value of $l_{potential}$ in (3.2):

$$l_{potential} = max(l_h, l_v) \tag{3.2}$$

Then, the problem of finding sets H and V that minimizes the value of $l_{potential}$, shown in Figure 3.2(a), is transformed into generating H and V that maximize the value of $min(w - 2l_h, w - 2l_v)$. The original rectangle block of size $w \times h$ is extended to $w \times w$ to transform the minimization objective function into the maximization objective function. Similarly, w also can be extended to h if $h > w$.

For example, we arrange the replicas of block b in 2×2 matrix as shown in Figure 3.2(b), in which $w - 2l_h$ and $w - 2l_v$ are the edge lengths of the red rectangle with no sink. Thus, sets H and V that leads to a maximum value of $min(w - 2l_h, w - 2l_v)$ can be found by finding a rectangle R_{max} which

has the longest among the shorter side edges of rectangles containing p and contains no sinks.

To find R_{max}, we define the simple rectilinear polygon P such as the green one in Figure 3.2(c) according to the sink placement. Then, we find the largest square Sq that contains p such that P has the longest edge of length l_{sq}. If Sq is found, R_{max} can be obtained by expanding Sq to horizontal or vertical direction. We applied a method based on Voronoi diagram [43] to find Sq in $O(|S|)$. With R_{max} we can easily find $U_x(V)$, $L_x(V)$, $U_y(H)$ and $L_y(H)$, as illustrated in Figure 3.2(a) and Figure 3.2(b). Then, each sink s_i in S is assigned to H and V by its coordinates: if $L_y(H) \leq s_{i,y} \leq U_y(H)$, s_i is assigned to H and if $L_x(V) \leq s_{i,x} \leq U_x(V)$, s_i is assigned to V.

The exact position of horizontal and vertical clock spines is independently determined for H and V. The position of clock spine is determined in the same way as that of single-line spine allocation.

2.2 (*Stub allocation*): If sink s_i is assigned to either H or V, but not both, connect s_i to the corresponding spine. If it is assigned to both, it is connected to the nearest spine.

2.3 (*Buffer allocation/placement*): We use a divide-and-conquer approach to the determination of size and placement location of spine buffers. We split each crossed spine into four single-line segments by removing the crossing point. We then apply the method of buffer allocation and placement for single-line spines we have developed in the previous section to each of the segments.

With the proposed allocation method for crossed clock spine, there are two

choices when allocating clock spine onto a sliced block: single-line clock spine and crossed clock spine. To make decision, the costs of single-line clock spine and crossed clock spine for a sliced block are compared. If the cost reduction with crossed clock spine is greater than threshold, crossed clock spine is allocated. Otherwise, single-line clock spine is selected.

The modified overall flow is shown in Figure 3.3.

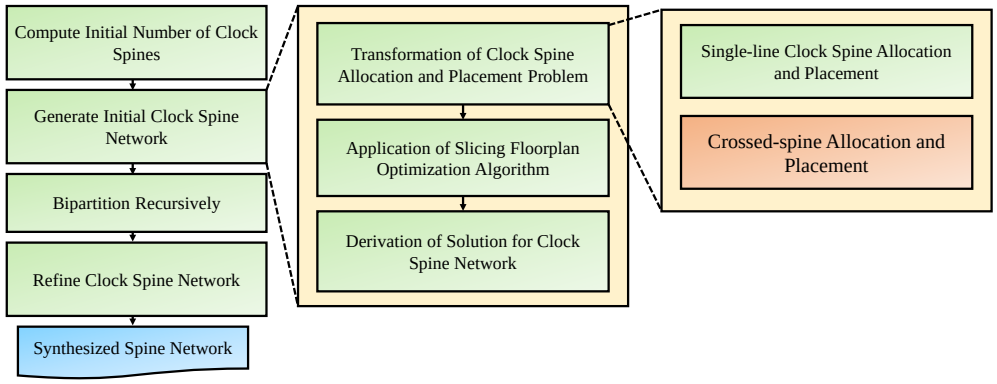


Figure 3.3: The modified synthesis flow of clock spine networks.

3.1.3 Experimental Results

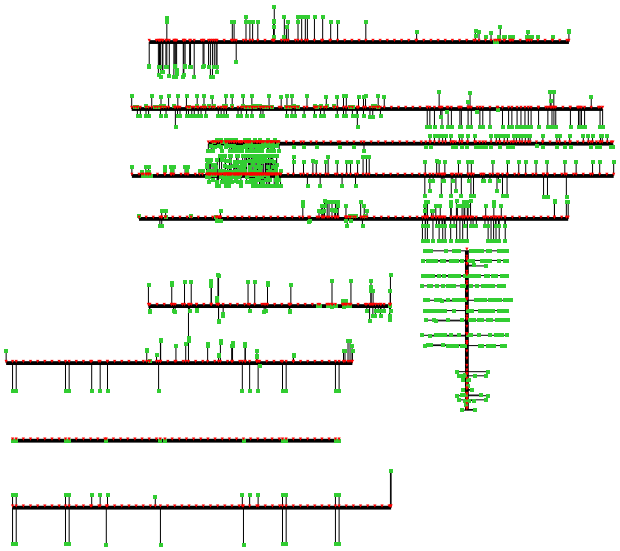
Table 3.1 summarizes the results of crossed clock spine networks. In comparison with the results of single-line spine networks, the clock skew is reduced by 11% on average with 3% decrease of wire and 5% decrease of buffer area. Power consumption is also reduced by 7% on average. Note that the wire and buffer overhead is reduced. This is because of the reduced length of stub wires, causing less loading capacitance. For all benchmarks, we observe the decrease of power consumption as well as the decrease of resource usage.

Figure 3.4(a) and Figure 3.4(b) show the layout of the clock spine network using single-line spines only and the network using a mixture of single-line and

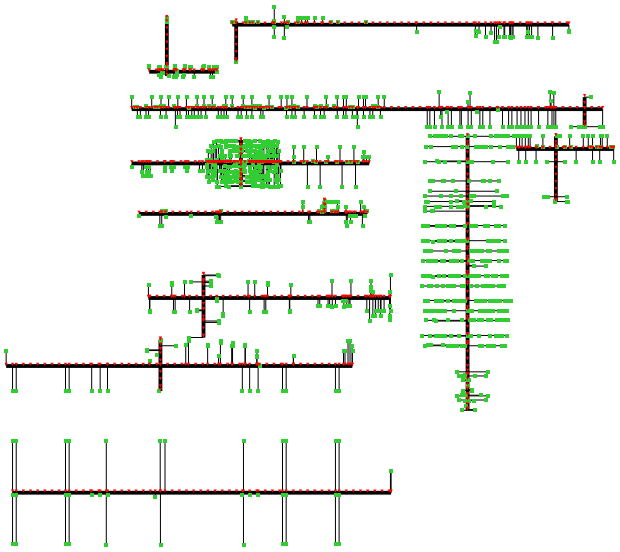
Table 3.1: Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock spine networks with only single-line spines and clock spine networks considering crossed clock spines.

Circuit	Clock spine networks using single-line spines				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	39.71	19.27	41956.0	1286.97	65.47
04	43.05	21.97	112211	1710.32	91.56
05	35.66	18.97	63265.4	868.21	47.90
06	28.93	13.49	51035.7	1166.58	59.56
07	40.70	18.42	104411	1952.12	109.93
08	24.01	11.61	67669.4	1319.67	75.28
Avg.ratio	1	1	1	1	1

Circuit	Clock spine networks using crossed spines				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	33.04	14.78	41277.9	1260.23	62.71
04	42.08	21.50	112105	1675.85	91.51
05	30.43	15.28	60410.8	833.952	46.13
06	27.56	12.23	49361.2	1064.77	55.22
07	30.82	14.10	100052	1837.04	95.19
08	23.70	13.70	64375.5	1217.6	65.91
Avg.ratio	**0.89**	**0.90**	**0.97**	**0.95**	**0.93**



(a) Clock spine network using single-line spines only.



(b) Clock spine network using the mixture of crossed and single-line spines.

Figure 3.4: Layout of the clock spine network using single-line spines and the network using crossed spines as well as single-line spines, produced by our algorithm for circuit ISPD08.

crossed spines, produced by our algorithm for circuit ISPD08 in Table 3.1. The thick and thin black lines indicate the spines and stubs, respectively. The red triangles represent the spine buffers and the green squares represent the clock sinks.

3.2 Application of Clock Gating

3.2.1 Introduction

In this section, we demonstrate the clock spine synthesis method considering clock gating method. Clock gating is a methodology of turning off some portions of clock distribution network for corresponding sequential blocks when there are no function required from those blocks for some duration. Clock gating method is a popular technique in synchronous circuits for reducing switching activity of flip-flops (FFs). With clock gating, the clock signals are gated with enabling signals, which comes from power management unit (PMU). Through restriction of switching activity with clock gating, the dynamic power consumed at the FFs and sequential blocks is saved. Many works are tried to apply clock gating in the register-transfer-level (RTL) [44], in the architecture level [45], and in the logic level [46].

Applying clock gating in RTL level is conducted before synthesizing clock networks. Thus, one drawback of clock gating in this stage is that unnecessary detours and wire snaking is involved because the placement information of clock sinks is unavailable [47]. To resolve this drawback, many works tried to integrate the clock gating into the clock network synthesis flow. In work [48], clock tree topology is constructed based on the placement information of clock sinks and pre-defined activity patterns of clock sinks in order to minimize the weighted sum of total wirelength of clock distribution network and estimated

total number of activities (i.e., the number of switching activities) of clock sinks. The activity pattern of a clock sink is a sequence of the per-cycle disable/enable (i.e., 0/1) status of the clock sink. The activity patterns of corresponding clock sinks can be obtained from the results of high-level synthesis. The work [49] elaborated the insertion of clock gating cells used in the work [48]. The recursive bitwise-OR operation is conducted based on the activity patterns of every pair of child nodes in the clock tree, thus computing the activity pattern of corresponding parent clock node. Computed activity patterns of parent nodes are referenced to determine whether clock gating cells are inserted at the corresponding clock nodes or not. In the works [50, 51], a probabilistic information of activity patterns of clock nodes is extracted from an instruction stream. The switching portion of downstream capacitance of each clock node is calculated to estimate the amount of power consumption of clock tree driven by corresponding node. Chao and Mak [52] constructed low-power gated clock trees with zero clock skew. Finally, the work [53] proposed fast algorithms for low-power gated clock tree which satisfies the clock skew and transition time constraints.

In the meanwhile, research on clock gating for other clock distribution network topologies, such as clock mesh, clock tree with cross links, and clock spine are rarely announced. To our knowledge, the work [54] is the only one which has addressed gated clock network for clock mesh. In this work, clock mesh is synthesized and clock gating is applied on their local subtrees attached to clock mesh. The proposed algorithm includes FF clustering and placement to form the local subtrees to reduce the dynamic power consumption. However, in this structure, the clock mesh wire, which shows highest power consumption in clock mesh networks, should be turned on when exactly one clock sink is activated, making overall power saving is very limited.

For the clock spine network topology, the work proposed by Seo, Kim, Kang,

and Kim [12] is the only one which has addressed the automated method for synthesis of clock spine considering spine-level clock gating. They proposed an algorithm for synthesis of gated clock spine networks. Since the clock spine wires which account for a large portion of total switching capacitance in clock spine network are clock gated, more dynamic energy savings can be achieved than work [54]. The main algorithm of work [12] is modified K-means clustering algorithm which clusters FFs and synthesizes clock spines based on energy consumption of clustered spine network with clock gating and activity pattern information. In this work, activity pattern of one clock spine segment is obtained by consecutive bitwise-OR operations for all activity patterns of clock sinks which are connected to corresponding clock spine segment. Then, clock spines are synthesized based on the energy consumption of one clock spine segment which can be estimated from the activity patterns of corresponding clock spine. Even though the algorithm considers the energy consumption based on predefined power modes, since only one FF is merged for each iteration, proposed algorithm is likely to fall into a local optimum and shows unnecessary clock resource consumption.

In this section, we propose the synthesis algorithm for clock spines which considers the spine-level clock gating. The spine wires are synthesized and gated to reduce the dynamic energy consumption based on the power mode information. Because our approach is based on the SA method proposed earlier, we are more likely to avoid locally optimal solutions.

3.2.2 Activity Pattern

Clock gating is a methodology of turning off corresponding portions of clock distribution network for sequential blocks when there is no function required from those blocks for some duration. Thus, it is essential to know the power

modes at which individual clock sinks require clock signal. This information is expressed by *activity patterns* [55]. Activity pattern [55] of clock sink can be expressed in sequence of binary numbers. If a value 1 is in i-th position of activity pattern of clock sink s, it means the sink s should be enabled at the power mode i. Otherwise, if a value 0 is in i-th position of activity pattern of clock sink s, the sink s should be disabled at the power mode i, saving dynamic power. For example, Figure 3.5 shows a clock spine in a design with four power modes, m_1, m_2, m_3 and m_4 respectively. The clock sink s_1 has activity pattern of [0110]. It means the clock sink s_1 is activated in power modes m_2 and m_3 and idle in power modes m_1 and m_4. If clock sinks are connected to a spine wire, the activity pattern of that spine wire is obtained by performing bitwise-OR operation for all activity patterns of clock sinks attached to the spine [12]. In Figure 3.5, the activity pattern of spine is [1110], which is result of bitwise-OR operation for activity patterns of clock sinks s_1, s_2, and s_3. It means, the spine buffers connected to the spine in Figure 3.5 are gated at power mode m_4, saving dynamic power consumption.

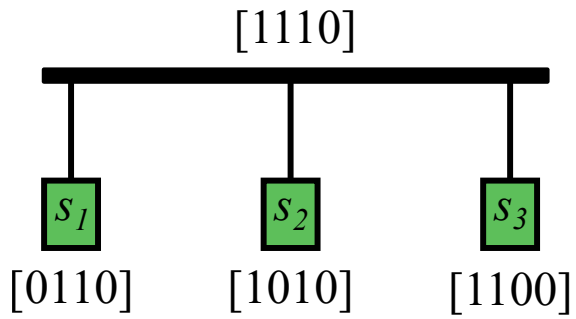


Figure 3.5: An example to illustrate the activity patterns of clock sinks and clock spine.

3.2.3 Problem Definition

The clock spine allocation and placement problem which considering spine-level clock gating (CSAP-CG) is formally stated as:

Problem 7. Clock spine allocation and placement problem with activity patterns (CSAP-CG): *For a set of clock sinks with their placement information, a buffer library $\mathcal{B}$, and parameters B_{stub}, l_{bound}, τ_{min}, N, the set of activation pattern A for clock sinks, and a power mode set M with activation duration, generate a clock spine network $\mathcal{S}$ by (1) allocating N clock spines whose maximum difference between the stub lengths is not larger than B_{stub}, (2) placing spines and allocating stubs whose longest length is not longer than l_{bound}, and (3) allocating buffers from $\mathcal{B}$ and placing them on the spines, satisfying the tolerance constraint τ_{min} of delay variation such that the following quantity is minimized:*

$$C(\mathcal{S}) = \alpha_1 \cdot WL_{sp}(\mathcal{S}) + \alpha_2 \cdot WL_{st}(\mathcal{S}) + \beta \cdot BA(\mathcal{S}) + \gamma E(\mathcal{S}) \qquad (3.3)$$

where $WL_{sp}(\mathcal{S})$, $WL_{st}(\mathcal{S})$, $BA(\mathcal{S})$, and $E(\mathcal{S})$ are the total length of the clock spines in $\mathcal{S}$, the total length of the stubs in $\mathcal{S}$, the total area of the spine buffers in $\mathcal{S}$, and total energy consumption of $\mathcal{S}$, respectively. α_1, α_2, β, and γ are weighting factors.

3.2.4 Proposed Method

The energy consumption of spine wire segment according to the activity patterns of clock sinks can be considered by updating only the cost function in (2.11) while maintaining the overall synthesis flow in Figure 2.5. Also, the same is true for synthesizing crossed spine networks.

To synthesize clock spine network considering clock gating at spine level,

our algorithm updates the cost function of spine s_i in (2.11) as:

$$c(s_i) = \alpha_1 WL_{sp}(s_i) + \alpha_2 WL_{st}(s_i) + \beta BA(s_i) + \gamma E(s_i). \tag{3.4}$$

$E(s_i)$ can be represented as:

$$E(s_i) = PWR(s_i) * \frac{act(s_i)}{total\ number\ of\ cycles} \tag{3.5}$$

where $PWR(s_i)$ is the total power consumption of s_i during the time when at least one sink in s_i is active and $act(s_i)$ is the number of activated cycles of spine s_i which can be obtained based on activity pattern of clock sinks connected to s_i.

3.2.5 Experimental Results

Table 3.2 shows the comparison of the results produced by activation pattern aware clock spine synthesis in [12] and ours. Since no logical information is available in ISPD2010 benchmarks, we generated four power modes and 10 cycles of activity patterns of clock sinks globally in random (by partitioning into 20 subregions), but locally correlated (setting roughly 90% correlation in each subregion). In summary, ours shows 11% less clock skew, even using much less clock resources: 37% less wire length and 56% less buffer area, which lead to 40% less power consumption over that of [12].

Since only one FF is merged for each iteration, the local minimum is often derived from algorithm of [12] and shows unnecessary clock resource consumption. In contrast, our algorithm can effectively reduce clock resource consumption and total energy consumption based on activity patterns of clock sinks.

3.3 Summary

In this chapter, extensions of algorithm for synthesizing clock spine networks were presented. First, fast algorithm for synthesizing crossed spine network is

Table 3.2: Comparison of clock skew (*Skew*), clock wire length (*WL*), total clock buffer area (*BA*), and total clock power consumption (*PWR*) of clock spine networks by [12] and ours considering clock gating.

Circuit	Clock spine networks with clock gating by [12]				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	36.50	16.35	83862.8	3370.56	117.80
04	56.78	25.12	184611	4050.54	162.30
05	41.34	18.36	90747.4	1983.78	79.62
06	41.91	18.74	74736.2	2552.84	90.93
07	38.59	18.69	138645	4226.72	155.01
08	37.30	17.31	88818.3	2804.75	99.45
Avg.ratio	1	1	1	1	1

Circuit	Our clock spine networks with clock gating				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
03	36.31	15.11	41121.7	1346.89	64.08
04	44.04	17.02	113079	1786.94	89.86
05	37.61	15.46	60884.6	888.38	47.45
06	32.22	13.44	49887.5	1191.39	58.52
07	46.88	21.27	93011.5	1896.11	96.73
08	24.94	10.95	58075.2	1281.28	62.80
Avg.ratio	**0.89**	**0.82**	**0.63**	**0.44**	**0.60**

proposed. With crossed spine structures, the exploration of more diverse clock spine networks has become possible and clock skew and power consumption can be reduced. Also, the method of clock spine network synthesis considering clock gating at spine level is proposed with slight modification of cost function. In experiments, clock spine networks synthesized by our algorithm show more efficient resource allocation and power reduction compared to work [12].

Chapter 4

Conclusion

This dissertation analyzed the benefits of shorting intermediates stages in reducing clock skew is analyzed with RC network model and Elmore's delay model and the influence of spine capacitance and spine resistance on the clock latency was addressed. Then algorithm for synthesizing clock spine networks is proposed to automate the synthesis of clock spine networks. In our knowledge, the problem of automating the synthesis of clock spine network without clock gating information has never been addressed in academia and industry as yet. The main contribution of this dissertation is defining and solving clock spine allocation and placement (CSAP) problem. The solution of CSAP problem is obtained by transforming an instance of CSAP problem into an instance of slicing floorplan optimization (SFO) problem. To explore the solution space while meeting skew and slew constraints, NN-based clustering algorithm, metrics to find initial number of grids, and tolerance metric of delay variations are proposed. To eliminate the isolated FFs, clock spine refinement methods are also illustrated. Through experiments with ISPD2010 benchmark circuits,

clock spine networks synthesized by our proposed algorithm reduced the clock skew by 30% while using 4% less wirelength and 21% more buffering area in average compared to clock tree networks. In comparison with clock mesh networks, clock spine networks show 21% increased global clock skew while using 50% less wire resources and 33% less buffering area.

In addition, the design space exploration method which can trade-off between tolerance to clock skew variation and resource consumption is proposed to design efficient the clock distribution network which is well suited for the design specification and constraints.

With crossed clock spine structure, clock skew decreased by 11% with 3% reduction of wire usage and 5% reduction of buffering area compared to clock spine structures only with single-line spine structure.

Finally, clock gating at spine level is considered for automation of clock spine synthesis. Through slight change of cost function, ours shows 11% less clock skew, even using 37% less wire length and 56% less buffer area, which lead to 40% less power consumption compared to that of [12].

Appendices

Appendix A

The Experimental Results of ISPD2010 Benchmark Circuits with Different τ Values.

The experimental results for ISPD2010 benchmark circuits obtained with different τ values (from 100 to 800) are here. The data in the columns labeled *Skew*, WL, BA, and PWR indicate the global clock skew, total clock wire length, total clock buffer area, and total power consumption, respectively. The μ and σ columns in the *Skew* column means the average global clock skew and its standard deviation value calculated from Monte Carlo simulation of each clock distribution network. As τ increases, clock skew decreases and clock resource consumption and power consumption increases. This indicates that the design space exploration based on the trade-off between the clock resource and clock skew from process variations can be conducted by changing τ value.

Table A.1: Experimental results of ISPD03 with different τ values.

τ	ISPD03				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
100	45.24	11.92	35414.2	1199.06	58.02
200	44.67	9.20	36841.3	1201.50	58.65
300	43.95	12.28	38140.3	1221.55	60.12
400	42.98	14.83	39525.9	1242.28	60.99
500	41.19	11.66	40049.9	1252.09	62.63
600	40.51	11.53	40134.8	1264.53	63.75
700	39.95	9.11	41596.5	1271.32	64.09
800	39.71	19.27	41956.0	1286.97	65.47

Table A.2: Experimental results of ISPD04 with different τ values.

τ	ISPD04				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
100	65.29	25.74	94032.8	1417.45	77.61
200	65.03	24.73	98881.4	1519.36	81.14
300	54.87	24.93	98266.2	1475.05	81.99
400	50.88	22.53	99012.4	1479.73	81.62
500	50.39	22.41	101541	1561.21	85.16
600	48.93	22.77	101322	1536.32	83.56
700	46.93	22.95	103072	1571.79	87.16
800	43.05	21.97	112211	1710.32	91.56

Table A.3: Experimental results of ISPD05 with different τ values.

τ	ISPD05				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
100	41.56	19.84	49746.8	687.10	38.85
200	38.33	18.91	49999.3	698.20	39.04
300	37.48	18.27	53262.7	762.61	41.99
400	37.18	18.37	56348.8	747.82	42.69
500	36.64	18.94	60156.4	786.24	45.05
600	36.82	18.67	59734.0	781.83	44.51
700	35.22	18.12	64041.0	840.85	47.48
800	35.66	18.97	63265.4	868.21	47.90

Table A.4: Experimental results of ISPD06 with different τ values.

τ	ISPD06				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
100	35.11	16.11	39813.0	987.09	49.46
200	32.22	15.92	42597.9	1038.70	51.84
300	31.51	15.29	44271.7	1048.29	52.97
400	30.85	14.66	46434.3	1058.50	53.71
500	29.07	14.44	48757.7	1070.89	55.83
600	29.32	13.93	50004.1	1086.53	56.86
700	28.15	13.48	50018.1	1106.48	58.05
800	28.93	13.49	51035.7	1166.58	59.56

Table A.5: Experimental results of ISPD07 with different τ values.

τ	ISPD07				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
100	48.97	19.58	78351.0	1600.80	82.55
200	47.48	19.78	82796.4	1605.96	85.41
300	47.91	18.82	88912.2	1676.79	88.42
400	45.51	18.41	93566.3	1722.95	91.68
500	43.13	18.14	94768.3	1727.05	91.52
600	41.91	18.67	97471.1	1795.30	94.81
700	39.04	19.05	101750	1823.55	94.56
800	40.70	18.42	104411	1951.12	109.93

Table A.6: Experimental results of ISPD08 with different τ values.

τ	ISPD08				
	Skew (ps)		WL	BA	PWR
	μ	σ	(μm)	(μm^2)	(mW)
100	34.51	16.98	48284.8	1097.32	55.68
200	29.65	15.53	51356.9	1112.07	56.90
300	26.14	15.03	54877.2	1144.01	59.08
400	26.08	14.40	55665.6	1167.69	59.86
500	26.00	14.74	59300.8	1170.19	62.02
600	25.15	14.05	61816.7	1179.04	63.09
700	24.96	13.77	62006.7	1203.45	63.80
800	24.01	11.61	67669.4	1319.67	75.28

Bibliography

[1] G. Geannopoulos and X. Dai, "An adaptive digital deskewing circuit for clock distribution networks," in *1998 IEEE International Solid-State Circuits Conference. Digest of Technical Papers, ISSCC. First Edition (Cat. No.98CH36156)*, Feb 1998, pp. 400–401.

[2] C. F. Webb, C. J. Anderson, L. Sigal, K. L. Shepard, J. S. Liptay, J. D. Warnock, B. Curran, B. W. Krumm, M. D. Mayo, P. J. Camporese, E. M. Schwarz, M. S. Farrell, P. J. Restle, R. M. Averill, T. J. Slegel, W. V. Houtt, Y. H. Chan, B. Wile, T. N. Nguyen, P. G. Emma, D. K. Beece, C.-T. Chuang, and C. Price, "A 400-mhz s/390 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 11, pp. 1665–1675, Nov 1997.

[3] P. Hofstee, N. Aoki, D. Boerstler, P. Coulman, S. Dhong, B. Flachs, N. Kojima, O. Kwon, K. Lee, D. Meltzer, K. Nowka, J. Park, J. Peter, S. Posluszny, M. Shapiro, J. Silberman, O. Takahashi, and B. Weinberger, "A 1 ghz single-issue 64 b powerpc processor," in *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.00CH37056)*, Feb 2000, pp. 92–93.

[4] P. J. Restle, C. A. Carter, J. P. Eckhardt, B. L. Krauter, B. D. McCredie,

K. A. Jenkins, A. J. Weger, and A. V. Mule, "The clock distribution of the power4 microprocessor," in *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, vol. 1, Feb 2002, pp. 144–145 vol.1.

[5] N. A. Kurd, J. S. Barkarullah, R. O. Dizon, T. D. Fletcher, and P. D. Madland, "A multigigahertz clocking scheme for the pentium(r) 4 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1647–1653, Nov 2001.

[6] N. Bindal, T. Kelly, N. Velastegui, and K. L. Wong, "Scalable sub-10ps skew global clock distribution for a 90nm multi-ghz ia microprocessor," in *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC.*, Feb 2003, pp. 346–498 vol.1.

[7] G. Gerosa, S. Curtis, M. D'Addeo, B. Jiang, B. Kuttanna, F. Merchant, B. Patel, M. Taufique, and H. Samarchi, "A sub-1w to 2w low-power ia processor for mobile internet devices and ultra-mobile pcs in 45nm hi-k metal gate cmos," in *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2008, pp. 256–611.

[8] R. Islam, A. Sabbavarapu, R. Patel, M. Kumar, J. Nguyen, B. Patel, and A. Kontu, "Next generation intel(r) atom processor based ultra low power soc for handheld applications," in *2010 IEEE Asian Solid-State Circuits Conference*, Nov 2010, pp. 1–4.

[9] ISPD. (2010) ISPD 2010 High Performance Clock Network Synthesis Contest. [Online]. Available: http://archive.sigda.org/ispd/contests/10/ispd10cns.html

[10] 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, Jan 2010.

[11] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *IEEE/ACM Design Automation Conference*, 1993, pp. 612–616.

[12] H. Seo, J. Kim, M. Kang, and T. Kim, "Synthesis for power-aware clock spines," in *IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 126–131.

[13] P. Gronowski, W. Bowhill, R. Preston, M. Gowan, and R. Allmon, "High-performance microprocessor design," *Solid-State Circuits, IEEE Journal of*, vol. 33, no. 5, pp. 676–686, May 1998.

[14] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Design Automation Conference, 1998. Proceedings*, June 1998, pp. 732–737.

[15] N. H. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 3rd ed. USA: Addison-Wesley Publishing Company, 2005.

[16] Y. Liu, S. R. Nassif, L. T. Pileggi, and A. J. Strojwas, "Impact of interconnect variations on the clock skew of a gigahertz microprocessor," in *Proceedings of the 37th Annual Design Automation Conference*. ACM, 2000, pp. 168–171.

[17] V. Mehrotra and D. Boning, "Technology scaling impact of variation on clock skew and interconnect delay," in *Proceedings of the IEEE 2001 International Interconnect Technology Conference*, June 2001, pp. 122–124.

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

[19] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. 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, Oct 2006.

[20] Z. Feng, P. Li, and Y. Zhan, "Fast second-order statistical static timing analysis using parameter dimension reduction," in *2007 44th ACM/IEEE Design Automation Conference*, June 2007, pp. 244–249.

[21] J.-L. Tsai, L. Zhang, and C. C.-P. Chen, "Statistical timing analysis driven post-silicon-tunable clock-tree synthesis," in *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, Nov 2005, pp. 575–581.

[22] C.-K. Koh, J. Jain, and S. F. Cauley, "Synthesis of clock and power/ground networks," in *Electronic Design Automation: Synthesis, Verification, and Test*. Electronic Design Automation: Synthesis, Verification, and Test, 2009.

[23] T. Y. Kim and T. Kim, "Clock tree embedding for 3d ics," in *IEEE Asia and South Pacific Design Automation Conference*, 2010, pp. 486–491.

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

[25] P. J. Restle and et al., "A clock distribution network for microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, May 2001.

[26] M. Kang and T. Kim, "Integrated resource allocation and binding in clock mesh synthesis," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 3, pp. 30:1–30:28, 2014.

[27] M. R. Guthaus, X. Hu, G. Wilke, G. Flach, and R. Reis, "High-performance clock mesh optimization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 17, no. 3, pp. 33:1–33:17, 2012.

[28] J. Lu, X. Mao, and B. Taskin, "Clock mesh synthesis with gated local trees and activity driven register clustering," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 691–697.

[29] A. Rajaram and D. Z. Pan, "Meshworks: A comprehensive framework for optimized clock mesh network synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 12, pp. 1945–1958, Dec 2010.

[30] D. J. Lee and I. L. Markov, "Multilevel tree fusion for robust clock networks," in *IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 632–639.

[31] T. Mittal and C.-K. Koh, "Cross link insertion for improving tolerance to variations in clock network synthesis," in *ACM International Symposium on Physical Design*, 2011, pp. 29–36.

[32] I. L. Markov and D.-J. Lee, "Algorithmic tuning of clock trees and derived non-tree structures," in *IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 279–282.

[33] A. Rajaram, J. Hu, and R. Mahapatra, "Reducing clock skew variability via crosslinks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1176–1182, June 2006.

[34] Y. Kim and T. Kim, "Algorithm for synthesis and exploration of clock spines," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 263–268.

[35] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," vol. 19, pp. 55 – 63, 02 1948.

[36] T.-M. Lin and C. A. Mead, "Signal delay in general rc networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 4, pp. 331–349, October 1984.

[37] P. Chan, "An extension of elmore's delay," *IEEE Transactions on Circuits and Systems*, vol. 33, no. 11, pp. 1147–1149, Nov 1986.

[38] P. K. Chan and K. Karplus, "Computing signal delay in general rc networks by tree/link partitioning," in *26th ACM/IEEE Design Automation Conference*, June 1989, pp. 485–490.

[39] B. Bolzano, *Beyträge zu einer begründeteren Darstellung der Mathematik*. Caspar Widtmann, 1810.

[40] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[41] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *IEEE/ACM Design Automation Conference*, 1986, pp. 101–107.

[42] PTM. (2011) Predictive Technology Model. [Online]. Available: http://ptm.asu.edu

[43] M. Gester, N. Hähnle, and J. Schneider-Barnes, "Largest empty square queries in rectilinear polygons," in *International Conference on Computational Science and Its Applications*, 2015.

[44] M. Donno, A. Ivaldi, L. Benini, and E. Macii, "Clock-tree power optimization based on rtl clock-gating," in *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, June 2003, pp. 622–627.

[45] Y. Luo, J. Yu, J. Yang, and L. Bhuyan, "Low power network processor design using clock gating," in *Proceedings of the 42Nd Annual Design Automation Conference*, ser. DAC '05. ACM, 2005, pp. 712–715.

[46] 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 *2008 45th ACM/IEEE Design Automation Conference*, June 2008, pp. 714–719.

[47] D. Garrett, M. Stan, and A. Dean, "Challenges in clockgating for a low power asic methodology," in *Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No.99TH8477)*, Aug 1999, pp. 176–181.

[48] A. H. 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, Jun 2001.

[49] C. Chen, C. Kang, and M. Sarrafzadeh, "Activity-sensitive clock tree construction for low power," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2002, pp. 279–282.

[50] J. Oh and M. Pedram, "Gated clock routing minimizing the switched capacitance," in *Proceedings Design, Automation and Test in Europe*, Feb 1998, pp. 692–697.

[51] ——, "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, Jun 2001.

[52] W.-C. Chao and W.-K. Mak, "Low-power gated and buffered clock network construction," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 1, pp. 20:1–20:20, Feb. 2008.

[53] 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, Nov 2012.

[54] J. Lu, X. Mao, and B. Taskin, "Clock mesh synthesis with gated local trees and activity driven register clustering," in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2012, pp. 691–697.

[55] G. E. Tellez, A. Farrahi, and M. Sarrafzadeh, "Activity-driven clock design for low power circuits," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, Nov 1995, pp. 62–65.

초록

클락 분배 네트워크는 동기식 디지털 시스템에서 클락 신호를 클락 소스로부터 클락 싱크 (플립플랍이나 래치)에 전달하는 회로이다. 하지만 반도체 공정이 미세화 됨에 따라 발생하는 공정 변이는 글로벌 클락 스큐를 증가시켰다. 클락 스큐의 증가는 회로의 동작 속도를 느리게 하며 회로 오동작의 원인이 된다. 따라서, 클락 네트워크를 설계할 때 클락 스큐를 최소화하거나, 일정 수준으로 제한해는 것이 중요하다. 공정 변이로 인해 발생하는 클락 스큐를 줄이기 위해, 클락 메쉬 네트워크가 연구되었다. 클락 메쉬 네트워크는 변이에 강하다는 장점이 있지만, 클락 자원 소모와 전력 소모가 매우 크다는 단점이 있다. 클락 스큐 변이 문제와 클락 자원 소모 문제를 동시에 대응하기 위해서 클락 스파인 구조가 대안으로 이용될 수 있다. 하지만 클락 스파인 네트워크 자동 합성방법을 언급한 연구는 거의 없다.

이 논문에서는, 클락 자원 소모와 클락 스큐 변동성을 동시에 고려하여 클락 스파인 네트워크를 합성하는 문제에 대해 논한다. 논문의 핵심 아이디어는 클락 스파인 할당 및 배치 문제를 슬라이싱 플로어플랜 최적화 문제로 변형하여 해결한 것이다. 이 때, 합성될 수 있는 각각의 클락 스파인 네트워크 구조들은 단 하나의 후위 표기로 대응되며, 이를 통해 슬라이싱 플로어플랜 최적화 과정에서 빠른 시간 내에 플로어플랜의 비용 계산이 가능하다. 클락 스파인이 갖는 변이에 대한 저항성을 일정 수준으로 유지하면서 클락 자원 소모를 줄이기 위해, 재귀 이중분할 방법과 스파인 네트워크의 변이 저항성 정도를 측정하는 방법도 제안하였다. 또한, 다양한 형태의 클락 스파인 구조 탐색을 위해 교차형 클락 스파인을 합성하는 방법도 제안하였다. 교차형 클락 스파인 구조를 이용하면 클락 스큐, 클락 자원 소모, 그리고 전력 소모를 더욱 줄일 수 있다. 마지막으로, 클락 스파인 단계에서의 클락 게이팅을 고려한 클락 스파인 합성 방법을 제안하였다. 실험을 통해, 제안한 방법이 이전 연구에 비해 클락 스큐, 클락 자원 소모, 그리고 전력 소모를 효과적으로

줄일 수 있음을 확인하였다.

주요어: 클락 네트워크, 클락 스파인, 클락 스큐, 지연 시간 변이
학번: 2013-30224