NORTHWESTERN UNIVERSITY

Timing Analysis and Optimization Techniques for VLSI Circuits

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical and Computer Engineering

By

Ruiming Chen

EVANSTON, ILLINOIS

December 2007

ABSTRACT

Timing Analysis and Optimization Techniques for VLSI Circuits

Ruiming Chen

With aggressive scaling down of feature sizes in VLSI fabrication, process variations, crosstalk and buffering have become critical issues to achieve timing closure in VLSI designs. Timing analysis and optimization techniques need to consider each of them and also their interactions. There are many statistical timing analysis researches to handle the problems introduced by process variations, but how to get the bounds of timing yield and how to use these techniques to verify the clock validity still need investigations. Timing budgeting is an essential step in optimization problems (e.g., gate sizing, dual-Vth assignment), and with process variations, how to reduce the design overhead in optimization becomes important. Dynamic programming is a useful technique to handle optimization problems, such as buffering, technology mapping and slicing floorplan problems, where many maxplus merge operations of solution lists are involved. So the speed-up of the maxplus merge operations will benefit many algorithms. In addition, prior researches were focusing on how to buffer a single net, but because of the huge number of buffers, the buffering of a whole circuit is required. With process variations on devices and interconnects, deterministic buffering techniques become

pessimistic. How to efficiently get a good buffering solution needs investigations. The hierarchical design has become a useful technique for large circuit design, so IP reuse becomes a common practice. But since coupling capacitance becomes dominant, the delay from a input pin to a output pin is not constant any more. How to accurately specify the timing behavior of IPs with crosstalk is urgent. In this research, we investigate these analysis and optimization problems.

In our first work, we propose two algorithms to compute the bounds of statistical delays. In our second work, a statistical checking of the structural conditions for correct clocking in latch-based designs is proposed, where the central problem is to compute the probability of having a positive cycle in a graph with random edge weights. In our third work, we consider the changes of both means and variances of delays in optimization, and formulate the timing budgeting under process variations as a linear programming problem using a robust optimization technique. In our fourth work, we propose a flexible data structure that can achieve universal speed-up under all cases for the merge operations. In our fifth work, the timing constrained minimal buffer insertion problem is formulated as a convex-cost flow dual problem, and we propose two algorithms based on *convex-cost flow* and *min-cut* techniques, respectively, to solve it in combinational circuits. In our sixth work, solutions from deterministic buffering are used to guide the statistical buffering in order to reduce the solution space in dynamic programming and achieve predictable results. Our last work presents two conservative macro-models that specify the timing behaviors of combinational hard IP blocks with crosstalk effects.

Acknowledgments

Prof. Hai Zhou, my advisor, teacher, mentor all rolled into one. From you I learnt the beauty of algorithm design, the need of strictness, high quality and perseverance in research, and the skill of technical writing. Thanks also for the financial and moral support over years.

To my dissertation committee, Professors Robert Dick, Yehea Ismail and Seda O. Memik, thank you for the inspiring discussions in the past four years, and the suggestions on my work. I would also like to thank Prof. Yan Chen for the suggestions on the network security project.

To Dr. Vladimir Zolotov and Dr. Chandu Visweswariah, my experience as a summer intern in IBM was very great with the guide and the inspiration from them. I would also thank all the other EinsStat team members in IBM for their help, especially to Kerim Kalafala, Jinjun Xiong and Natesan Venkateswaran.

To the members of NuCAD group, specifically Chuan Lin, Debasish Das, Nikos Liveris, Debjit Sinha and Jia Wang, thank you for the enourmous help in my research and daily life. I am very proud that I am a member of this wonderful group.

To my parents, brother and sisters, this thesis is the result of as much effort (if not more) on your parts than mine.

Last, but not the least, I thank Zheng Yu, my wife, for her great support during the last four years.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

With aggressive scaling down of feature sizes in VLSI fabrication, process variations, buffering and crosstalk have become critical issues to achieve timing closure in VLSI designs. This disseration considers how to handle each of these issues and also their interactions.

1.1. Process variations

The first issue we are handling is process variation. The corner-based deterministic static timing analysis (STA) becomes pessimistic and inefficient because of the complicated correlations among component delays and the huge number of corners. Timing optimization techniques based on corners also become pessimistic and expensive.

Many statistical static timing analysis (SSTA) approaches have emerged, and greatly speeded up the analysis by propagating the distributions instead of single values. Although it was shown that SSTA approaches have good accuracy, it is not guaranteed that the computed yield is either lower or higher than the actual yield. Without this information, the designers have to over design in order to make sure that the yield is satisfied. So the computation of the lower bound and the upper bound of the yield is desired. We propose two correlation-aware block-based statistical timing analysis approaches that keep these necessary conditions, and prove that our approaches always achieve tight lower bound and upper bound of the yield. Especially, our approach always gets the tight upper bound of the yield irrespective of the distributions that random variables have.

Usually having level-sensitive latches for high speed, high-performance IC designs need to verify the clock schedules. With process variations, the verification needs to compute the probability of correct clocking. Because of complex statistical correlations, traditional iterative approaches are difficult to get accurate results. Instead, in our work, a statistical checking of the structural conditions for correct clocking is proposed, where the central problem is to compute the probability of having a positive cycle in a graph with random edge weights. The proposed method only traverses the graph once to avoid the correlations among iterations, and it considers not only data delay variations but also clock skew variations.

Timing budgeting under process variations is an important step in a statistical optimization flow. We propose a novel formulation of the problem where budgets are statistical instead of deterministic as in existing works. This new formulation considers the changes of both the means and the variances of delays, and thus can reduce the timing violation introduced by ignoring the changes of variances. We transform the problem to a linear programming problem using a robust optimization technique. Our approach can be used in late-stage design where the detailed distribution information is known, and is most useful in early-stage design since our approach does not assume specific underlying distributions. In addition, with the help of block-level timing budgeting, our approach can reduce the timing pessimism.

1.2. Buffering

The second issue we are handling is the huge number of buffers. Saxena et al. [**83**] predicted synthesis blocks will have 70% of their cell count dedicated to interconnect buffers within a few process generations. Consequently, there is an increasing demand for *efficient* buffer insertion approaches.

Dynamic programming is a useful technique to handle buffering, technology mapping and slicing floorplan problems, where many maxplus merge operations of solution lists are involved. We propose a flexible data structure that can achieve universal speed-up under all cases for the merge operations.

The problem of buffer insertion in a single net has been the focus of most previous researches. However, efficient algorithms for buffer insertion in whole circuits are generally needed. The timing constrained minimal buffer insertion problem is formulated as a convex-cost flow dual problem, and propose two algorithms based on *convex-cost flow* and *min-cut* techniques, respectively, to solve it in combinational circuits.

Then, we consider how to handle process variations and huge number of buffers simultaneously. Because it is hard to be 100% positive to satisfy the prune condition when process variations are considered, the main diffculty comes from the huge number of solutions in the direct extension of van Ginnekin's buffering algorithm. We use deterministic solutions based on corners to guide the statistical pruning, and the solution space is greatly reduced while the accuracy is still reasonable as demonstrated by experiments.

1.3. Crosstalk

The third issue we considered is crosstalk. IP reuse is becoming a common practice in the modern VLSI designs. The last work presents two conservative macro-models for specifying the timing behaviors of combinational hard IP blocks with crosstalk effects. The gray-box model keeps a coupling graph and lists the conditions on relative input arrival time combinations for couplings not to take effect. The black-box model stores the output response windows for a basic set of relative input arrival time combinations, and computes

the output arrival time for any given input arrival time combination through the union of some combinations in the basic set.

1.4. Dissertation organization

In this dissertation, we present our results [**20**–**31**] for the problems mentioned above. The rest of this dissertation is organized as follows. In Chapter 2, we propose two efficient algorithms to compute the bounds of delays under process variations. In Chapter 3, we propose our timing verification algorithms to handle the clock schedule verification problem with the consideration of process variations in the latch-based designs. In Chapter 4, an algorithm that considers the changes of both means and variances during timing budgeting is proposed. In Chapter 5, a flexible data structure for maxplus merge operations in dynamic programming is proposed. In Chapter 6, we present two algorithms based on convex-cost flow and min-cut techniques, respectively, to solve the buffering problem in a whole combinational circuits. In Chapter 7, we illustrate an efficient statistical optimization technique that is guided by deterministic buffering solutions. In Chapter 8, two timing macro-models for hard IP with crosstalk effects are proposed.

CHAPTER 2

Statistical Static Timing Analysis without Moment Matching

With aggressive scaling down of feature sizes in VLSI fabrication, process variation has become a critical issue in designs. The corner-based deterministic static timing analysis (STA) becomes pessimistic and inefficient because of the complicated correlations among component delays and the huge number of corners.

The emerging statistical static timing analysis (SSTA) approaches [**14**, **15**, **53**, **73**, **103**, **113**, **114**] greatly speed up the analysis by propagating the distributions instead of single values. An essential problem in SSTA is how to compute the maximum of random variables. Assuming that process variations are not very prominent, [**14**] and [**103**] used Clark's approach [**33**] to approximate the maximum of two random variables with Gaussian distribution as a Gaussian variable, and achieved good efficiency and accuracy. Random variables are represented in a linear canonical form, and the first two moments (that is, the mean and the variance) are matched for the maximum. The essential of the approach is the least-squares fitting.

The delay of a gate or a wire is affected by more than one type of process variations, and a linear form may not be accurate enough to capture important information. So [**15**, **113**, **114**] extended the linear model to non-linear models. For example, random variables in [**114**] are represented in a quadratic model. These approaches are shown to be more accurate than those based on the linear model.

With the development of SSTA tools, many statistical timing optimization works also emerged. These works optimize the timing yield (the probability that a circuit satisfies timing constraints) using SSTA approaches to compute timing information. But how accurate is the timing yield estimation? Without this information, the designers have to over-design in order to make sure that the yield objective is satisfied. Monte Carlo simulation, an expensive approach, is widely used in the existing literatures to verify the results from SSTA. As shown in Fig. 2.1, if we can quickly estimate the lower bound and upper bound of the yield, comparing these bounds with the yield estimation by moment matching based SSTA approaches will tell how accurate the estimation is. Agarwal *et al.* [**2**, **3**] proposed techniques to compute the bounds on yield, but they did not consider correlations: [**3**] ignored the correlations between components, while [**2**] ignored the correlations due to path reconvergence, therefore it is not clear if the computed bounds are close to the actual yield when correlations are considered.

In this chapter, we consider how to compute the lower bound and upper bound of timing yield. The existing SSTA works use the linear model or the second order model to *approximate* process variations. Even the yield computed by the Monte Carlo simulation is not the exact yield. However, the designers can select parameters in the models such that the described process variations are the lower bound or the upper bound of actual process variations. As shown in Fig. 2.1, our approach computes the bounds of timing yields based on a model that bounds process variations instead of the fitting techniques widely used in literatures. Thus, the accurate computation of the lower and upper bounds of the yield can tell whether the yield objective is satisfied. Enforcing two necessary conditions for the statistical "max" operation that were not satisfied by moment-matching based approaches [**14**, **103**],

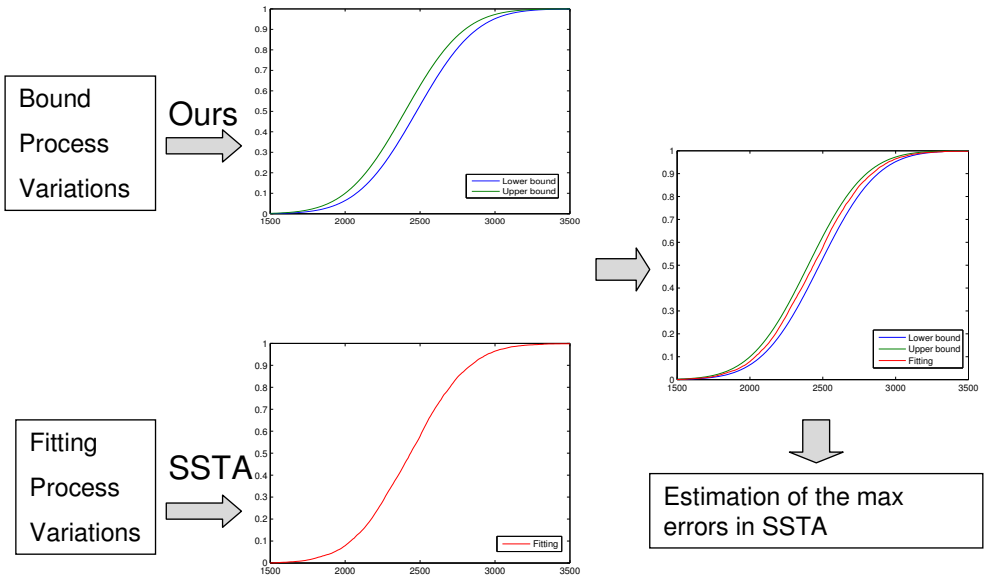


Figure 2.1. Our approach is useful for the fast estimation of the maximal errors in moment matching based SSTA approaches.

our approaches achieve tight bounds of yield. Furthermore, for upper bound computation, our techniques can also be used with the second-order model.

The rest of this chapter is organized as follows. Section 2.1 briefly reviews the existing works on SSTA. Section 2.2 presents the relations between the results and the operands in the statistical "Max" operation, and discusses problems in moment matching based approaches. Section 2.3 presents our correlation-aware approaches for the statistical "Max" operation. The experiments on the proposed approaches and their comparison with the Monte Carlo simulation are reported in Section 2.4. Finally, the conclusions are drawn in Section 2.5.

2.1. Preliminary

The combinational circuit is represented by a directed acyclic graph (DAG) $G(V, E)$ with a vertex (or node) set V and an edge set E. Each vertex represents a primary input (PI),

a primary output (PO) or a gate; each edge represents an interconnection from the source vertex to the target vertex; and the edge weight gives its delay. Two dummy nodes s and t are introduced into the graph: s is connected to all the primary inputs, and t is connected from all the primary outputs. The weights of the edges from s to PIs are the arrival time of the corresponding PIs, and the weights of the edges from POs to t are the negative of the required arrival time at the corresponding POs.

All the delays (or weights), slacks and arrival time are represented in a first-order canonical form as in [**14**]:

$$c_0 + \sum_{i=1}^{n} c_i X_i,$$

where c_0 is the mean value, X_i's are the principal components [**59**], and c_i's are their coefficients. Principal component analysis [**59**] may be performed to get this canonical form [**14**].

We define the following for two Gaussian random variables X and Y with correlation coefficient ρ.

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2/2), \tag{2.1}$$

$$\Phi(y) = \int_{-\infty}^{y} \phi(x)dx, \tag{2.2}$$

$$\theta_{XY} = \sqrt{\sigma_X^2 + \sigma_Y^2 - 2\rho\sigma_X\sigma_Y}, \tag{2.3}$$

$$\alpha_{XY} = \frac{\mu_X - \mu_Y}{\theta}. \tag{2.4}$$

Given any two random variables X and Y, [**103**] defined the *tightness probability* T_X of the variable X as the probability that it is larger than Y, and $T_Y = 1 - T_X$. Thus,

$$T_X = \Phi(\frac{x_0 - y_0}{\theta_{XY}}),$$

when $X \neq Y$.

In block-based SSTA, the moment matching is performed to compute the canonical form representing $\max(X, Y)$. For example, [**103**] matches the mean, variance and covariance, while [**113**] matches the raw moments.

Chang *et al.* [**14**] compute the maximal of two Gaussian random variables as follows. Suppose

$$\begin{aligned} A &= a_0 + \sum_{i=1}^{n} a_i X_i, \\ B &= b_0 + \sum_{i=1}^{n} b_i X_i. \end{aligned}$$

Let C represent $\max(A, B)$. Then according to [**33**],

$$\mu(C) = T_A \mu(A) + (1 - T_A)\mu(B) + \theta\phi(\alpha), \tag{2.5}$$

$$\begin{aligned} \sigma^2(C) &= [\sigma^2(A) + \mu^2(A)]T_A \\ &\quad + [\sigma^2(B) + \mu^2(B)](1 - T_A) \\ &\quad + [\mu(A) + \mu(B)]\phi(\alpha) - \mu^2(C). \end{aligned} \tag{2.6}$$

The moment matching [**14**] gives

$$C = \mu(C) + \frac{\sigma(C)}{s_0} \sum_i \beta_i X_i,$$

where

$$\beta_i = T_A a_i + (1 - T_A) b_i,$$

and

$$s_0 = \sqrt{\sum_i \beta_i^2}.$$

2.2. Statistical "Max" operation

We introduce two concepts for relations among random variables.

Definition 2.1 (Dominance relation)**.** *A random variable A dominates variable B when*

$$Pr(A \geq B) = 1.$$

Definition 2.2 (Comparison relation)**.** *A random variable C has comparison relation with variables A and B when*

$$\begin{aligned} Pr(C > A) &= Pr(B > A), \\ Pr(C > B) &= Pr(A > B). \end{aligned}$$

The following theorem shows that both the dominance relation and the comparison relation are necessary conditions for the statistical "Max" with its operands.

Theorem 2.1. *Suppose A and B are random variables and* $C = \max(A, B)$*, then C dominates A and B, and has the comparison relations with them.*

Proof. Since C is the maximum of A and B, $C \geq A$ and $C \geq B$, so C dominates A and B.

$$Pr(C > A) = Pr(\max(A, B) - A > 0)$$

$$\begin{aligned} &= Pr(\max(A-A, B-A) > 0) \\ &= Pr(\max(0, B-A) > 0) \\ &= Pr(B-A > 0) \end{aligned}$$

Similarly, we can prove

$$Pr(C > B) = Pr(A > B).$$

□

The block-based SSTA approaches [**14**, **103**] assume that all the random variables have Gaussian distribution. They use a canonical form

$$c_0 + \sum_{i=1}^{n} c_i X_i$$

to represent a random variable, where c_0 is the nominal value, and X_i's are independent random variables with standard normal distribution. When they compute the maximum of Gaussian variables, they use Clark's approach [**33**] to match the mean and the variance. But during this match, the dominance and comparison relations are not kept. For example, compute the maximum of the following two Gaussian random variables using the approach in [**103**]:

$$A = 30 + x_1,$$

and

$$B = 30.5 + 0.5x_1.$$

Suppose $C = max(A, B)$, then theoretically,

$$Pr(C \geq A) = 1 \; and \; Pr(C \geq B) = 1.$$

But the results computed from the moment matching based approach in [**103**] are

$$Pr(C \geq A) = 89.46\% \; and \; Pr(C \geq B) = 62.57\%.$$

So the dominance relations are not kept. Also theoretically,

$$Pr(A > B) = 15.84\%,$$

but the moment matching based approach gets

$$Pr(C > B) = 62.57\%,$$

which has a big difference from $Pr(A > B) = 15.84\%$. So the comparison relations are not kept either.

We also took the approach in [**113**] to approximate the maximum of two Gaussian variables as a non-Gaussian variable, and find that neither the dominance nor comparison relation is kept. For example, using the approach in [**113**], for the dominance relations, we get

$$Pr(C \geq A) = 63.43\% \; and \; Pr(C \geq B) = 49.17\%,$$

and for the comparison relations, we get

$$Pr(C > B) = 49.17\% \neq Pr(A > B) = 15.84\%.$$

Thus, the existing approximation approaches have not kept the necessary conditions in the statistical "Max" operation.

A timing analysis approach may be used in timing optimizations. In statistical timing optimization, we need to compute the yield, that is, the probability that the constraint is satisfied. Since the moment matching based SSTA approaches [**14**, **103**] are approximation approaches, there is no guarantee whether they are conservative or optimistic. For example, given a timing constraint for the maximal delay from the primary input to the primary output, we do not know if the computed yield is larger or smaller than the actual yield.

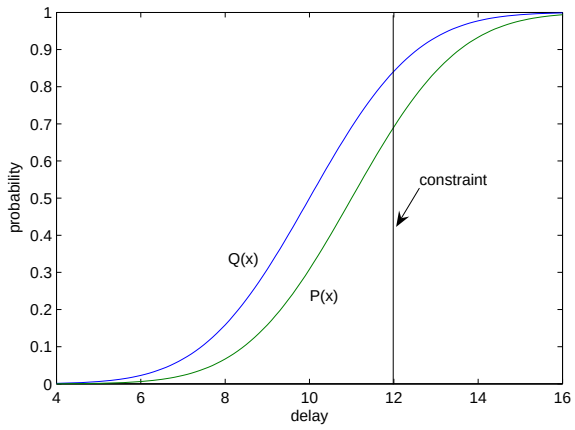


Figure 2.2. CDF $Q(x)$ is an upper bound of CDF $P(x)$.

Definition 2.3. *For any two cumulative distribution functions $P(x)$ and $Q(x)$, $Q(x)$ is the upper bound of $P(x)$ (and $P(x)$ is the lower bound of $Q(x)$) if and only if $\forall x : Q(x) \geq P(x)$.*

As shown in Fig. 2.2, using the upper bound of $P(x)$, the yield $Pr(x \leq constraint)$ according to the upper bound of $P(x)$ is higher than the yield according to $P(x)$. We will

show later that the approaches based on the dominance relations or the comparison relations give the lower bound or the upper bound of the yield, respectively.

2.3. SSTA without moment matching

"Max" and "Add" are the two fundamental operations in timing analysis. In SSTA, all random variables are represented in the canonical form. The "Add" operation is simple and exact. For the "Max" operation, we plan to enforce either the dominance relation or the comparison relation.

2.3.1. Theory

Our SSTA approach traverses a circuit in the topological order, and computes the distribution of the arrival time at each node. Depending on what relations the procedure keeps, our approach has two variants. The first one, denoted as LBDomSSTA, keeps the dominance relations, while the second one, denoted as UBCompSSTA, keeps the comparison relations.

Note that the theory in this subsection holds for random variables of any distributions, not only limited to Gaussian.

For the dominance relation, we have the following theorem.

Theorem 2.2. *In a combinational circuit, when a "*max*" operation is encountered, if a random variable that dominates the operands is used for their maximum, the computed yield will be a lower bound of the actual yield.*

Proof. Suppose A and B are operands of the "max" operation, and C dominates A and B. Then $C \geq A$ and $C \geq B$, so $C \geq \max(A, B)$. If C is used as the maximum of A and

B, the computed maximal delay is no less than the actual maximal delay, so the yield is not larger than the actual yield. □

Therefore, LBDomSSTA is guaranteed to get the lower bound of yield.

The comparison relations can be transformed into

$$Pr(C \leq A) = Pr(B \leq A), \tag{2.7}$$

$$Pr(C \leq B) = Pr(A \leq B). \tag{2.8}$$

For the comparison relation, we have

Theorem 2.3. *Suppose A and B are two random variables, and let*

$$C = \beta A + (1 - \beta)B,$$

where $\beta \in [0, 1]$, then C always satisfies the comparison conditions:

$$Pr(C \leq A) = Pr(B \leq A),$$

$$Pr(C \leq B) = Pr(A \leq B).$$

Now we prove the following lemma.

Lemma 2.1. *Suppose A and B are two random variables, and*

$$C = \beta A + (1 - \beta)B,$$

where $\beta \in [0, 1]$, then

$$\max(A, B) \geq C.$$

Proof.

$$\begin{aligned}\max(A,B)-C &= \max(A-C,B-C)\\ &= \max(A-(\beta A+(1-\beta)B),\\ &\quad B-(\beta A+(1-\beta)B))\\ &= \max((1-\beta)(A-B),\beta(B-A))\end{aligned}$$

Thus, if $A \geq B$, $(1-\beta)(A-B) \geq 0$, so $\max(A,B) \geq C$; if $A \leq B$, $\beta(B-A) \geq 0$, so $\max(A,B) \geq C$. □

According to Lemma 2.1, we know that the "max" of two random variables as computed in Theorem 2.3 is not larger than their actual maximum. Therefore, we have the following lemma based on the monotonicity property of the "max" operation.

Lemma 2.2. *The maximal delay from the primary inputs to the primary outputs computed in UBCompSSTA is not greater than the actual maximal delay.*

For two random variables A and B, if $P_r(A \leq B) = 1$, then $P_r(A \leq D) \geq P_r(B \leq D)$ for any constant D. Thus, we have the following theorem.

Theorem 2.4. *The yield computed by UBCompSSTA gives the upper bound of the actual yield.*

2.3.2. Lower bound

Most of the existing SSTA approaches assume that random variables have Gaussian distributions. In this subsection, we consider the LBDomSSTA under this assumption. Suppose

A and B are two Gaussian variables. Let

$$A = a_0 + \sum_{i=1}^{n} a_i x_i,\ B = b_0 + \sum_{i=1}^{n} b_i x_i,$$

and

$$C = \max(A, B) \approx c_0 + \sum_{i=1}^{n} c_i x_i.$$

Unfortunately, we have the following theorem.

Theorem 2.5. *Let A and C be two Gaussian variables represented in the first order canonical form. Then*

$$Pr(C \geq A) = 1$$

cannot be satisfied unless $C = A + d$ with d a non-negative constant number.

Proof. Suppose

$$\begin{aligned} A &= a_0 + \sum_{i=1}^{n} a_i X_i, \\ C &= c_0 + \sum_{i=1}^{n} c_i X_i. \end{aligned}$$

If $C = A + d$,

$$Pr(C \geq A) = Pr(A + d \geq A) = Pr(d \geq 0) = 1,$$

If $C \neq A + d$, obviously $\sum_{i=1}^{n} (a_i - c_i)^2 \neq 0$. So

$$\begin{aligned} Pr(C \geq A) &= \Phi(-\mu_{A-C}/\sigma_{A-C}) \\ &= \Phi(-\frac{a_0 - c_0}{\sqrt{\sum_{i=1}^{n} (a_i - c_i)^2}}) \end{aligned}$$

But $\Phi(x)$ is not equal to 1, so

$$Pr(C \geq A) < 1.$$

□

Therefore, it is almost impossible to find a Gaussian random variable to dominate all the operands simultaneously. However, if the dominance relation is relaxed to

$$Pr(C \geq A) \geq \eta,\ Pr(C \geq B) \geq \eta, \tag{2.9}$$

with $0 < \eta < 1$, it is possible to find a C satisfying this condition. With increasing η, we have increasing confidence that the computed yield is a lower bound.

Clark [**33**] stated that the covariance between $C = max(A, B)$ and any random variable Y can be expressed in terms of covariances between A and Y and between B and Y as

$$Cov(C, Y) = Cov(A, Y)T_A + Cov(B, Y)(1 - T_A).$$

As suggested in [**103**], in order to preserve the covariance, for every $Y = x_i$, we must have

$$c_i = a_i T_A + b_i(1 - T_A)\ i = 1, 2, \ldots n. \tag{2.10}$$

Our approach adjusts the mean value (c_0) to satisfy the dominance relations given by the following inequalities.

$$Pr(C \geq A) = \Phi(\frac{c_0 - a_0}{(1 - T_A)\sqrt{\sum_i (a_i - b_i)^2}}) \geq \eta \tag{2.11}$$

$$Pr(C \geq B) = \Phi(\frac{c_0 - b_0}{T_A\sqrt{\sum_i (a_i - b_i)^2}}) \geq \eta \tag{2.12}$$

Let ζ be a constant satisfying $\Phi(\zeta) = \eta$. Then the two inequalities can be transformed to

$$\frac{c_0 - a_0}{(1 - T_A)\sqrt{\sum_i (a_i - b_i)^2}} \geq \zeta, \tag{2.13}$$

$$\frac{c_0 - b_0}{T_A\sqrt{\sum_i (a_i - b_i)^2}} \geq \zeta. \tag{2.14}$$

By solving this inequality set, we can find the minimal c_0. The dominance relations are then satisfied.

2.3.3. Upper bound

2.3.3.1. Gaussian variables. In this subsection, we also assume that random variables have Gaussian distributions. According to Theorem 2.3 and the discussion in the previous subsection, if we select $\beta = T_A$, the comparison relations are kept, and the covariance is also preserved.

In order to check whether the upper bound is tight, we compare the mean and the variance computed by our approach and those by the moment matching based approach. Let C represent the maximum of the two Gaussian random variables A and B computed by our approach, and D represent $\max(A, B)$ computed by [**33**].

$$\begin{aligned}
\mu(D) - \mu(C) &= (T_A\mu_A + (1 - T_A)\mu_B + \theta\phi(\alpha)) \\
&\quad -(T_A\mu(A) + (1 - T_A)\mu(B)) \\
&= \theta\phi(\alpha) \geq 0.
\end{aligned}$$

Assuming that all the random variables have at most 10% deviation (3σ) from their nominal values, we get

$$\begin{aligned}
\theta^2 &= \sigma^2(A) + \sigma^2(B) - 2\rho\sigma(A)\sigma(B) \\
&\leq \sigma^2(A) + \sigma^2(B) + 2\sigma(A)\sigma(B) \\
&\leq (0.10\mu(A)/3)^2 + (0.10\mu(B)/3)^2 \\
&\quad +2(0.10\mu(A)/3)(0.10\mu(B)/3) \\
&\leq (0.10/3)^2(\mu(A) + \mu(B))^2.
\end{aligned}$$

Thus,

$$\begin{aligned}
\frac{\theta^2}{\mu^2(D)} &\leq \frac{\theta^2}{(T_A\mu_A + (1-T_A)\mu_B)^2} \\
&\leq (0.10/3)^2 \frac{(\mu(A) + \mu(B))^2}{(T_A\mu(A) + (1-T_A)\mu(B))^2}
\end{aligned}$$

Since the random variables in our problem represent delay or arrival time, if we set the arrival time at the PIs to 0, their mean values should be non-negative. Without loss of generality, we assume $\mu(A) \geq \mu(B) > 0$. So $T_A \geq 0.5$. Let $\mu(A) = \gamma\mu(B)$, so $\gamma \geq 1$. Thus,

$$\begin{aligned}
\frac{\theta^2}{\mu^2(D)} &\leq (0.10/3)^2 \frac{(\gamma\mu(B) + \mu(B))^2}{(\gamma T_A\mu(B) + (1-T_A)\mu(B))^2} \\
&= (0.10/3)^2 \frac{(1+\gamma)^2}{(\gamma T_A + 1 - T_A)^2} \\
&\leq (0.10/3)^2 \frac{(1+\gamma)^2}{(0.5 + 0.5\gamma)^2} \\
&= 4(0.10/3)^2
\end{aligned}$$

$$= \quad 0.0044 \tag{2.15}$$

In addition, $\phi(\alpha) \leq 1/\sqrt{2\pi}$, so the relative error of the mean is at most

$$\frac{\sqrt{0.0044}}{\sqrt{2\pi}} = 2.66\%.$$

From this derivation, we can see that if the variance is smaller, or the correlation is positive and larger, the result will be more accurate. In practice, this relative error is even smaller because of the highly positive correlation between delays and the small variance.

Even though the mean plays a major role in the yield, we will also estimate the error on the variance.

$$\begin{aligned} \sigma^2(D) - \sigma^2(C) &= T_A(1-T_A)[\sigma^2(A) + \sigma^2(B) \\ &\quad -2\rho\sigma(A)\sigma(B) + (a_0 - b_0)^2] - \theta^2\phi^2(\alpha) \\ &\quad +\theta\phi(\alpha)[(a_0 - b_0)(1 - 2T_A)] \end{aligned} \tag{2.16}$$

If $\theta = 0$,

$$\begin{aligned} 0 &= \sqrt{\sigma^2(A) + \sigma^2(B) - 2\rho\sigma(A)\sigma(B)} \\ &= \sqrt{\sum_i (a_i - b_i)^2} \end{aligned}$$

Thus

$$a_i = b_i \ \forall i = 1 \ldots n,$$

and

$$T_A = 0 \text{ or } T_A = 1.$$

So

$$\sigma^2(D) - \sigma^2(C) = 0.$$

If $\theta > 0$ (note $\theta \geq 0$),

$$\begin{aligned} \alpha^2 &= \frac{(\mu(A) - \mu(B))^2}{\sigma^2(A) + \sigma^2(B) - 2\rho\sigma_A\sigma_B} \\ &\geq \frac{(\gamma - 1)^2}{(0.10/3)^2(\gamma + 1)^2} \\ &= 900\frac{(\gamma - 1)^2}{(\gamma + 1)^2}. \end{aligned} \tag{2.17}$$

According to [**92**], when $\alpha \geq 3$, the right hand side of Eq. (2.16) approaches 0. So when $\gamma \geq 1.22$, the error approaches 0.

When $\gamma < 1.22$, according to [**92**],

$$\sigma^2(D) - \sigma^2(C) \leq 0.091\theta^2.$$

While according to Eq. (2.15), when $\rho \geq 0$, and $\gamma < 1.22$,

$$\theta^2 \leq 0.0022\mu^2(D).$$

Thus,

$$\begin{aligned} \sigma^2(D) - \sigma^2(C) &\leq 0.091 * 0.0022\mu^2(D), \\ &= 0.0002\mu^2(D). \end{aligned} \tag{2.18}$$

Therefore,

$$\frac{\sigma^2(D) - \sigma^2(C)}{\mu^2(D)} \leq 0.02\%.$$

Therefore, the error on variance is at most 0.02% of the square of mean value. If the correlation coefficient (ρ) is more positive, this error gets even smaller. For example, when $\rho = 1$, the error is less than 0.01%. Note that moment matching based approaches are doing approximations, so the errors computed here are not the errors from the actual values.

In summary, the results computed in UBCompSSTA are very close to the results from the moment-matching approach. Note that the moment-matching approach is also an approximation approach, so it is possible for UBCompSSTA to have better results than the moment-matching approach.

2.3.3.2. Non-Gaussian variables. In this part, we consider the cases where the random variables do not have Gaussian distributions. The delay of a gate or an interconnect may be affected by not only one kind of process variation, so there may exist non-linear relations between the delays and the process variations. For example, the delay of a wire is affected by the process variations on the wire length, the wire width and the wire thickness. Zhang *et al.* [**114**] has proposed a quadratic delay model for a wire. The delay random variable D is represented in the following quadratic model:

$$D = m + \alpha\delta + \delta^T \Upsilon \delta + \gamma^T r,$$

where $r = (R_1, R_2, \ldots, R_p)^T$ represents the local variances, $\delta = (X_1, X_2, \ldots X_q)^T$ represents the global variances, α and γ are sensitivity vectors, and Υ is a sensitivity matrix. All these R_i's and X_j's are independent and have the standard Gaussian distribution. The random variables represented in this form do not have a Gaussian distribution. We will show that our approach also gets results close to the results from [**114**].

Suppose random variables A and B are represented in the quadratic model:

$$A = m_A + \alpha_A \delta + \delta^T \Upsilon_A \delta + \gamma_A^T r, \tag{2.19}$$

$$B = m_B + \alpha_B \delta + \delta^T \Upsilon_B \delta + \gamma_B^T r, \tag{2.20}$$

In the computation of the maximum of two random variables A and B represented in the quadratic model, Zhang *et al.* [**114**] approximated the random variables as Gaussian variables by moment matching and computed the skewness of the output. If the skewness is greater than a threshold, the "max" operation is delayed, otherwise, the approach got

$$\max(A, B) = m_C + \alpha_C \delta + \delta^T \Upsilon_C \delta + \gamma_C^T r,$$

where

$$m_C = T_A m_A + (1 - T_A) m_B + \theta \phi(\alpha) \tag{2.21}$$

$$\alpha_C = T_A m_A + (1 - T_A) m_B \tag{2.22}$$

$$\Upsilon_C = T_A \Upsilon_A + (1 - T_A) \Upsilon_A \tag{2.23}$$

$$\gamma_C = T_A \gamma_A + (1 - T_A) \gamma_B \tag{2.24}$$

The output of our approach differs from the output of [**114**] only in the m part. Our approach gets

$$m = T_A m_A + (1 - T_A) m_B.$$

Since m affects only the mean value, we only need to compute the difference on the mean value in our approach. Let C and D represent the outputs of [**114**] and our approach

respectively. The difference between $\mu(C)$ and $\mu(D)$ is $\theta\phi(\alpha)$. We can show similarly that the relative error of the mean is at most 2.66%.

Thus, our approach can also be applied to methods where the variables do not have Gaussian distributions, and get an upper bound of the yield that is close to the result from [**114**].

2.4. Experimental results

We have implemented LBDomSSTA and UBCompSSTA in C++. Experiments were performed on the large cases in ISCAS85 benchmark. We use the cases where all the random variables have the Gaussian distributions as examples to show the effectiveness of our approaches. We also implemented the Monte Carlo simulation to compute the maximal delay from s to t. We made 10,000 trials for each test case. All the random variables have at most 10% deviation from their nominal values. All the experiments were run on a Linux PC with a 2.4 GHz Xeon CPU and 2.0 GB memory.

Table 2.1. Comparison results of UBCompSSTA, LBDomSSTA, [**14**] and Monte Carlo simulation on Linear Model

name	UBCompSSTA				LBDomSSTA				Monte Carlo			[**14**]		
	time (s)	μ	σ	yield (%)	time (s)	μ	σ	yield (%)	μ	σ	yield (%)	μ	σ	yield (%)
c1355	0.01	1580	40	91.15	0.01	1585	40	89.07	1583	40	90.00	1583	40	89.62
c1908	0.01	4000	100	91.92	0.01	4019	101	88.49	4011	100	90.00	4018	101	88.49
c2670	0.01	2918	61	91.15	0.01	2926	61	89.25	2922	61	90.00	2923	61	89.80
c3540	0.03	4700	120	92.22	0.02	4727	120	88.30	4715	119	90.00	4718	120	89.44
c5315	0.03	4900	123	91.47	0.02	4919	123	88.69	4910	125	90.00	4913	123	89.25
c6288	0.03	12400	312	92.36	0.03	12477	314	87.90	12443	313	90.00	12464	314	88.69
c7552	0.05	4300	107	91.47	0.04	4320	107	88.30	4311	107	90.00	4313	107	89.44

The comparison results of UBCompSSTA, LBDomSSTA, [**14**], and the Monte Carlo simulation are shown in Table 2.1. We perform Monte Carlo simulations to compute the 90% percentile point of the maximal delay from s to t, and select this point as the timing

constraint. We have $\eta = 90\%$ in LBDomSSTA. The columns 2, 3, 4, and 5 show the running time, the mean of the maximal delay, the standard deviation of the maximal delay, and the yield computed by UBCompSSTA, respectively. The columns 6, 7, 8, and 9 show the running time, the mean of the maximal delay, the standard deviation of the maximal delay, and the yield computed by LBDomSSTA, respectively. The 10th and 11th columns show the mean and the standard deviation of the maximal delay from the Monte Carlo simulation, respectively. The results show that our approaches always get *tight* bounds of the yield. The errors on the yield are 1.68% and 1.43% on average for UBCompSSTA and LBDomSSTA, respectively. The relative errors on the mean and the variance are also quite small. The [**14**] gets more accurate results than ours since it is doing the fitting instead of the bounding. The significant contribution of our work is that it provides an *efficient* way to estimate the maximal errors of the SSTA approaches without doing the expensive Monte Carlo simulation. For example, for these testcases, if we do not know the Monte Carlo simulation results, we can also conclude that [**14**] gets very accurate results since the yields from [**14**] are very close to the bounds computed from our approaches.

Fig. 2.3 shows the cumulative distribution functions from LBDomSSTA, UBCompSSTA, and the Monte Carlo simulation for the case "c6288". The CDF from UBCompSSTA stays on the left side, while the CDF from LBDomSSTA stays on the right, and the actual CDF stays between them. It demonstrates that our approaches achieve the bounds in the whole range.

UBCompSSTA is also tested based on the quadratic model. Table 2.2 shows the comparison results between UBCompSSTA and the Monte Carlo simulation. The results show that UBCompSSTA has an error of 1.50% on average.

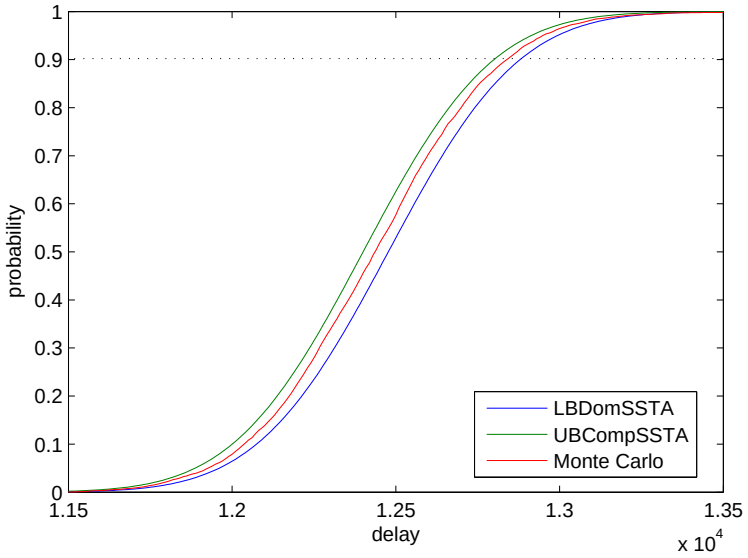


Figure 2.3. The CDFs from different approaches for "c6288".

Table 2.2. Comparison results of UBCompSSTA and Monte Carlo simulation on Quadratic Model

name	Monte Carlo			UBCompSSTA			
	μ	σ	yield (%)	time (s)	μ	σ	yield(%)
c1355	1629	75	90.00	1.18	1623	73	90.97
c2670	4088	151	90.00	1.54	4073	142	91.37
c2670	2960	78	90.00	0.63	2949	76	92.39
c3540	4810	169	90.00	3.05	4786	168	91.63
c5315	4997	168	90.00	6.14	4979	165	91.48
c6288	12700	472	90.00	5.14	12623	442	92.16
c7552	4397	156	90.00	5.99	4378	152	90.52

2.5. Conclusions

The state-of-the-art statistical static timing analysis approaches cannot tell whether the computed yield is smaller or larger than the actual yield. In this chapter, we proposed two block-based statistical static timing analysis approaches by satisfying each of the two necessary conditions for "max" operation. We showed that our approach achieves a tight

upper-bound of the actual yield. Furthermore, for the upper bound computation, our approach achieves the bound even without the assumption of Gaussian distributions for the random variables.

CHAPTER 3

Clock Schedule Verification under Process Variations

With shrinking geometries in deep sub-micron technology, process variation becomes a prominent phenomenon in fabrication. These variations introduce random variables into the timing of a fabricated integrated circuit. These delay variations and clock skew variations present a new challenge on timing verification and yield prediction.

There are many recent researches that deal with the timing analysis under process variations [**1**, **2**, **11**, **14**, **41**, **103**]. These researches are mainly focused on timing analysis of combinational circuits. However, the validity of a circuit really depends on whether the correct signal values are latched into the memory elements, and the results of timing analysis are to be used to check clocking conditions.

Level-triggered transparent latches are usually used in high-performance circuits because of their high performance and low power consumption [**42**]. The complexity introduced by latches is that a signal can pass transparently through a latch during its enabling period, which makes time borrowing across latch boundaries possible. Therefore, timing analysis can no longer be carried out only on the separated combinational part since the output time of a latch is now dependent on its input time.

In this chapter, we formulate the clock schedule verification problem under process variations as computing the probability of correct clocking. Considering applying iterative approaches to computing the arrival time on different corners, some will converge but others will not. As collections of the arrival time, the distributions will not converge. Furthermore,

since a delay will have a fixed value after fabrication, the distributions in different iterations are tightly correlated. These issues make the iterative approaches [**84**, **85**, **97**] difficult to be used under process variations. The accumulated inaccuracy of *max* and *min* operations on random variables is also an obstacle to iterative methods. For example, given three random variables A, B and C, suppose $A = \max(B, C)$. Theoretically,

$$A = \max(\max(A, B), C) = \max(A, \max(B, C)),$$

but in reality, this is not always true:

$$\max(\max(A, B), C) \neq \max(A, \max(B, C))$$

because of the errors in *max* operations. Then if we need to determine the convergence by checking the statistical conditions like follows:

$$f(X) \leq \beta$$

where $f(X)$ is a function on random variable vector X, and β is a threshold value, the selection of β may greatly influence the results. An *applicable* statistical iterative algorithm needs to tell the users the number of iterations before the execution, instead of determining when the algorithm stops by checking some statistical conditions during the execution. Based on this, we propose to use structural clock validity conditions with process variations. The relationship between valid clocking and the conditions of no positive cycle in the latest constraint graph or negative cycle in the earliest constraint graph is first established. Then these structural conditions with delays and clock skews being random variables are checked through non-iterative graph traversal techniques. One advantage of these techniques is that

each element in the circuit is traversed only several times and the effect of correlations and accumulated inaccuracy of statistical operations is thus greatly reduced.

There was some recent work on the statistical timing analysis of latch-based pipeline design [**16**, **115**], where the key problem is to compute the probability that the arrival time of the latches violates setup-time and hold-time constraints. Not involving any cycles, this problem is a special case of our statistical clock schedule verification problem, and can be efficiently and accurately solved by our algorithms.

The rest of the chapter is organized as follows. In Section 3.1, the models of latches, clocking schemes, and the conditions on correct clock schedules are given without consideration of process variations. With the consideration of process variations, Section 3.2 formulates the statistical clock schedule verification problem and presents the difficulties involved in iterative approaches under process variations. Section 3.3 establishes the structural conditions for valid clock schedules by extending Szymanski and Shenoy's work [**97**]. In Section 3.4, the probability that these structural conditions are held under random element delays and clock skews is computed to give the probability of valid clocking, and an important application of our algorithms in pipeline designs is proposed. The experiments on the proposed approaches and their comparison with the MC simulation are reported in Section 3.5. Finally, the conclusion is given in Section 3.6.

3.1. Deterministic clock schedule verification

3.1.1. Models of clock and transparent latches

A clock is a periodical signal used to regulate other signals in the circuit. Multiple phases of a clock may be used in a circuit. A *clock scheme* for a circuit is a set of periodical signals $\phi_1, \ldots, \phi_n$ with a common period c. Selecting a period of length c as the *global time*

reference, we can denote each phase ϕ_i by its starting and ending time (s_i, e_i) with respect to the reference. Note that it is possible to have $s_i > e_i$ based on the selection of global time reference. For simplicity, we assume that $s_i < e_i$ for all the phases, which can be easily done by shifting the global time reference. We generally order the phases such that $e_i < e_j$ if $i < j$. Also note that w_i is used to represent the width of phase i. A three-phase clocking scheme is shown in Fig. 3.1.

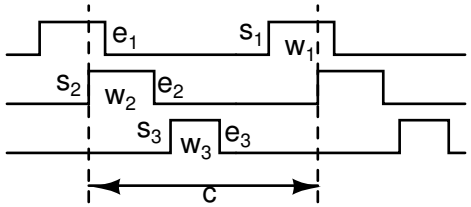


Figure 3.1. Three phase clocking with period c.

Memory elements are used in a circuit to store its state. Only under a certain condition of the clock input does the memory element respond to the data input. Memory elements can be categorized into two groups according to how they respond to the clock input: *flip-flops* store the data when the clock switches; *latches* let the output have the input value when the clock level is high. Figure 3.2 shows the symbol used for a memory element and the signal responses in a latch and a flip-flop.

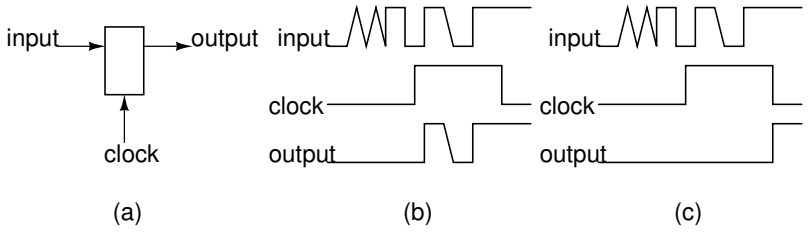


Figure 3.2. (a) A memory element; (b) Signal response in a latch; (c) Signal response in a flip-flop.

Because of this, clock schedule verification is easy in a circuit with only flip-flops but is very hard when latches are used. We only focus on the latter problem in the sequel.

3.1.2. Clock validity conditions

Without the consideration of process variations, the clock schedule verification problem can be formulated as

Problem 3.1 (deterministic clock schedule verification)**.**
Given a circuit and a clock schedule without process variations, check whether the clock schedule is correct in the fabricated chips.

Given the pin-to-pin delay of each gate, the maximal and minimal delays from the output of one latch to the input of another latch can be computed by traversing the topology of the gate connections. Let Δ_{ij} and δ_{ij} represent the maximal and minimal combinational delays from latch i to latch j, respectively. Also let A_i and a_i represent the latest and the earliest signal arrival time on the input of latch i, and D_i and d_i the latest and the earliest signal departure time on the output of latch i, respectively. N denotes the number of latches. The well-known SMO formulation [**79**] is

$$A_i = \max_{j \to i}(D_j + \Delta_{ji} - E_{p_j p_i}) \tag{3.1}$$

$$D_i = \max(A_i, c - w_{p_i}) \tag{3.2}$$

$$a_i = \min_{j \to i}(d_j + \delta_{ji} - E_{p_j p_i}) \tag{3.3}$$

$$d_i = \max(a_i, c - w_{p_i}) \tag{3.4}$$

where p_i is the clock phase controlling latch i and E_{ij} is defined as

$$E_{ij} = \begin{cases} e_j - e_i & \text{if } j > i \\ c + e_j - e_i & \text{otherwise} \end{cases}$$

We must also note that used here is local time referring to local periods that end with the phase falling edges.

Ignoring initial hold condition violations, the SMO formulation is too aggressive on earliest time calculation. A *deterministic conservative formulation* of earliest time constraints is presented in [**78**], as

$$a_i = \min_{j \to i}(d_j + \delta_{ji} - E_{p_j p_i}) \tag{3.5}$$

$$d_i = c - w_{p_i} \tag{3.6}$$

In practice, the aggressive formulation might yield a solution with a shorter period, but [**96**] showed that there are common situations, such as a latch driven by a qualified clock signal, in which the aggressive formulation is incorrect, and a similar problem arises in circuits which permit the clock to be stopped between adjacent latches to save power. Therefore, we choose the conservative formulation.

The solution of the above equations should satisfy the setup and hold time conditions:

$$A_i \leq c - S_i \tag{3.7}$$

$$a_i \geq H_i \tag{3.8}$$

where S_i and H_i are the setup time and hold time of latch i respectively.

Shenoy [**85**] proposed an iterative approach to solve the deterministic clock schedule verification problem. The general flow of this approach on the setup time condition is shown in Fig. 3.3.

Shenoy's Algorithm
1 Initialize A_i and D_i for each latch i
$A_i^0 = -\infty$
$D_i^0 = c - w_{p_i}$
2 **for** $m = 1, 2, \ldots n$
3 **do for** each latch i
4 **do** $A_i^m = \max_{j \to i}(D_j^{m-1} + \Delta_{ji} - E_{p_j p_i})$
5 $D_i^m = \max(A_i^m, c - w_{p_i})$
6 **if** converged
7 **then** Check the setup condition (Eq. 3.7)
8 **else** Report "invalid clock schedule"

Figure 3.3. The clock schedule verification algorithm in [**85**].

Shenoy has the following theory regarding the convergence of the algorithm in deterministic situation [**85**].

Lemma 3.1. *A_i^m and D_i^m are monotonically increasing with m at every latch i.*

Theorem 3.1. *The iteration procedure converges to a solution, if one exists, in at most N iterations.*

3.2. Problem formulation

3.2.1. Statistical clock schedule verification

Process variations influence not only the data delays but also the clock network, and there exist correlations among the process variations of all the devices including clock network. There are many recent researches that deal with the timing analysis under process variations

[**1**, **2**, **11**, **14**, **41**, **103**]. All these researches only dealt with timing analysis on combinational circuits. However, sequential circuits dominate the reality, and the validity of a circuit really depends on whether the correct signal values can be latched into the memory elements. So all these researches eventually should be used to check the clocking conditions, which is the focus of our work. Neves et al. [**71**] presented a graph-based algorithm to solve the clock skew optimization problem considering process variations, and it only considered the process variations of clock network while neglecting the dominating data delay variations. Rather, our work provides a framework for statistical clock schedule verification problems, which considers both data delay variations and clock network variations. The correlations among data delay variations and clock network variations are all considered here.

With the consideration of process variations, the clock schedule verification problem can be formulated as

Problem 3.2 (statistical clock schedule verification)**.**
Given a circuit and a clock schedule under process variations, compute the probability that the clock schedule is correct in the fabricated chips.

Since there exist process variations in the clock network, the starting and ending time (s_p, e_p) are not the same for the latches controlled by the same clock phase p, so we cannot use a *single* pair of variables (s_p, e_p) to represent this phase. A pair of random variables (s_i, e_i) is used to represent the clock phase controlling latch i. For simplicity, we assume that the influence of process variations on s_i and e_i is the same, so $w_{p_i} = e_i - s_i$ is constant, thus we only need one random variable e_i. Our method can be easily extended to more realistic models. Thus, we need n correlated random variables (e_i) and p constants (w_p) to represent all the clock phases instead of $2p$ constants (e_p, w_p) in the deterministic

clock schedule verification problem, where n is the number of latches, and p is the number of clock phases. Since the delays are influenced by process variations, $\Delta_{ij}, \delta_{ij}, A_i, a_i, D_i$ are random variables. Process variations also influence the setup time and hold time of latches, so S_i and H_i are also random variables. These random variables may be correlated. In the sequel, we use bold font to differentiate the random variables from the deterministic ones. In summary, the following variables in the conservative formulation are random variables: $\{\mathbf{A_i}, \mathbf{a_i}, \mathbf{D_i}, \mathbf{\Delta_{ji}}, \delta_{\mathbf{ji}}, \mathbf{E_{ji}}, \mathbf{S_i}, \mathbf{H_i}\}$. Changing these variables to random variables, we can translate the deterministic conservative formulation of time constraints to the *statistical conservative formulation* of time constraints.

3.2.2. Difficulties in traditional iterative methods

In statistical situation, we have

Theorem 3.2. *The probability that the iteration converges is*

$$Pr(\max_{i \in V}(D_i^{n+1} - D_i^n) = 0)$$

where $n = N$.

But according to Lemma 3.1, $D_i^{n+1} \geq D_i^n$, which means that the distribution of $D_i^{n+1} - D_i^n$ should be truncated at point 0. Then, if we use the moment matching approaches [**14**, **103**] to compute the D_i, the mean value of $D_i^{n+1} - D_i^n$ must be non-negative, so

$$Pr(\max_{i \in V}(D_i^{n+1} - D_i^n) \leq 0) \leq 50\%.$$

So the direct extension of the deterministic clock schedule verification algorithm based on Theorem 3.2 cannot get accurate results when the yield is larger than 50%.

Considering applying iterative approaches on different corners, some will converge but others will not. As collections of the arrival time, the distributions will not converge.

Also, many *max* and *min* operations on correlated random variables are involved in the conservative formulation. Since no accurate analytical formula exists for any of them, the current practice always uses approximation techniques to handle them. For example, some current statistical timing analysis algorithms [**14**, **103**] are based on the Clark's method [**33**] that assumes that the maximum of random variables with normal distribution is still with normal distribution. However, from our experience, the actual distribution may be far away from normal distribution in some cases. The accumulated inaccuracy of *max* and *min* operations on random variables introduces inaccuracy into the ultimate results of iterative methods.

Furthermore, in clock schedule verification with process variations, when traversing the same element in different iterations, the element delay is the same. This means that in an iterative approach the delays in different iterations are perfectly correlated, which makes some state-of-the-art statistical timing analysis techniques inappropriate to be used in the iterative approaches under process variations. For example, the technique in [**103**] used a canonical delay model that denotes the delay as the sum of global components and local components. The local components are used to improve the accuracy. But since there exist no local components in iterative methods, this technique loses correlation information, which results in much inaccuracy in the ultimate results.

Let $s_c^\infty = \max_i(A_i^\infty - (c - S_i))$. Then

Theorem 3.3. *The yield of a circuit with N latches is*

$$Pr(s_c^\infty \leq 0)$$

Proof. If the equation set has no solution, according to Lemma 2.1, A_i keeps increasing, and it is obvious that A_i has no upper-bound, so it will eventually be larger than $c - S_i$.

If the equation set has a converged solution, A_i is fixed after N iterations according to Theorem 3.1, which means $A_i^{\infty} = A_i^N$. So checking $\max_i(A_i^{\infty} - (c - S_i)) \leq 0$ tells us whether the clock is valid. □

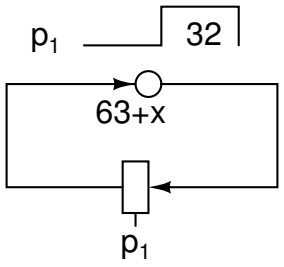


Figure 3.4. A circuit has only one latch and one gate.

Based on this theorem, we do sufficient iterations such that

$$Pr(s_c^k \leq 0)$$

converges. Then it gives the desired yield. This algorithm is denoted as SuffAlg. Since the s_c^{∞} does not have a truncated distribution, it can be approximated as a Gaussian variable. So this approach may get the accurate results. But how to decide when the algorithm stops? One choice is that when

$$|Pr(s_c^{k+1} \leq 0) - Pr(s_c^k \leq 0)| \leq \beta,$$

the algorithm terminates. But this may lead to much inaccurate results. For example, For example, as shown in Fig. 3.4, a circuit has only one gate and one latch. Suppose the clock cycle time is 64 units, the duty cycle is 50%, and the delay of the gate is $63 + x$, where x is

a random variable with standard Gaussian distribution. The actual yield is

$$Pr(63 + x - 64 \leq 0) = 84.13\%.$$

But the yield computed according to Theorem 3.3 is fixed at 100%.

We also tried to use some non-Gaussian distribution to approximate the arrival time. For example, we use the non-Gaussian approximation approach in [**113**] to compute the A_i's and D_i's for the circuit in Fig. 3.4. The yield is 74.32% when $\beta = 0.01$, while when $\beta = 0.0001$, the yield is 60.74%. This confirms that it is difficult to get the stable results when a statistical algorithm *dynamically* decides whether the algorithm should terminate.

Now we have seen that a direct extension of a deterministic algorithm to a statistical one has to consider many *subtle* issues. In deterministic situation, we can always compare variables and based on the comparison results, select a definite action. But in statistical situation, a random variable may not dominate others, thus it is impossible to select only one action. For example, if $x > y$, then $y := x$. Also the approximation errors should be taken into account. A theoretically valid algorithm may not get correct results in reality because of the approximation errors or the stop criteria.

One way to simplify the correlations among iterations and reduce the effect of the inaccuracy of statistical operations on ultimate results is to reduce the number of iterations without sacrificing the accuracy too much.

3.3. Structural verification of clock schedule

3.3.1. Deterministic situation

Having studied when the iterative approach to the deterministic clock schedule verification problem will converge and whether the converged solution is unique, Szymanski and Shenoy came up with some structural characterizations. A circuit C is modeled as a finite, edge-bi-weighted, directed graph $G = (V, E, \Delta, \delta)$. For every memory element, primary input or primary output, there is a vertex in V. If there is a path of combinational logic from one memory element or primary input i to another memory element or primary output j, there is a directed edge e_{ij} from the vertex representing element i to the vertex representing element j. The edge weight Δ_{ij} (δ_{ij}) is the maximum (minimum) delay of the combinational paths from i to j. G is called *latch graph*. The following two theorems were given in Shenoy's thesis [**85**].

Theorem 3.4. *The equation set composed by 3.1, 3.2 has a unique solution if and only if there is no zero Δ-weight cycle in the latch graph.*

Theorem 3.5. *If the latch graph has no zero δ-weight cycle, then the equation set composed by 3.3 and 3.4 has a unique solution.*

Since it is difficult to use an iterative approach when the delays become random variables, a theory of structural conditions for a valid clocking will be established.

First, we consider the deterministic clock schedule verification problem. The *latest equation set* is composed by 3.1, 3.2 and 3.7. The *earliest equation set* is composed by 3.5, 3.6

and 3.8. We first translate the latest equation set into a system of inequalities.

$$\begin{aligned} A_i - D_j &\geq \Delta_{ji} - E_{p_j p_i} \ \forall j \rightarrow i \\ D_i - A_i &\geq 0 \ \forall i \\ D_i &\geq c - w_{p_i} \ \forall i \\ -A_i &\geq S_i - c \ \forall i \end{aligned}$$

Based on the correspondence between a system of difference inequalities and the longest path problem on a graph [**35**], we can construct a *latest constraint graph* corresponding to this inequality set. For each constraint $x_i - x_j \geq b_{ij}$, vertices v_i and v_j are introduced for variables x_i and x_j, and an edge from v_j to v_i with weight b_{ij} is also introduced. Another vertex O will be introduced as the reference time 0.

Similarly, the earliest equation set can be translated into the following inequalities, from which an earliest constraint graph can be constructed.

$$\begin{aligned} a_i - d_j &\leq \delta_{ji} - E_{p_j p_i} \ \forall j \rightarrow i \\ d_i &\leq c - w_{p_i} \ \forall i \\ -a_i &\leq -H_i \ \forall i \end{aligned}$$

As an example, consider a circuit given in Fig. 3.5(a). Its latest and earliest constraint graphs are given in Fig. 3.5(b) and (c), where the setup time and hold time are assumed to be 0 for simplicity.

We denote the sum of the weights of all the edges in a path as the weight of this path. If the weight of a cycle is positive (negative), this cycle is called *positive (negative) cycle*.

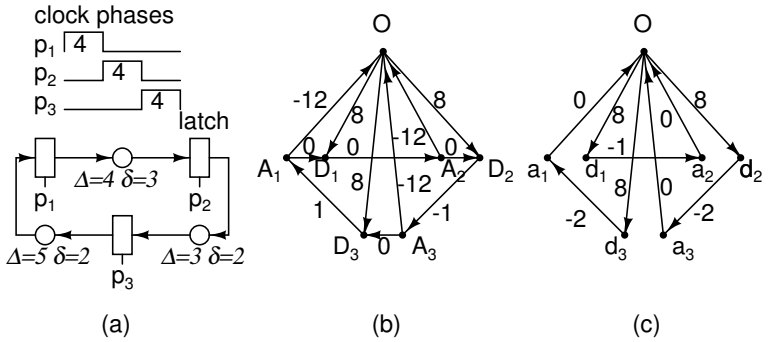


Figure 3.5. A clock schedule (a) and its latest constraint graph (b) and earliest constraint graph (c).

Based on the construction of the latest and earliest constraint graphs, the following theorem can be established.

Theorem 3.6. *A given clock schedule is valid if and only if the latest constraint graph has no positive cycle and the earliest constraint graph has no negative cycle.*

Proof. For the latest constraint graph, Sakallah et al. [**78**] have proved that the relaxation of the latest equation set is equivalent to the the latest equation set. So the constraints in latest equation set are satisfied if and only if the latest constraint graph has no positive cycle.

For the earliest constraint graph, Shenoy et al. [**84**] proved that for each edge from latch u to latch v, constraint

$$e_{p_v} \leq s_{p_u} + \delta_{uv} + (1 - K_{uv})c - H_u$$

is equivalent to the earliest time constraints in the earliest equation set, where

$$K_{uv} = \begin{cases} 0 & \text{if } e_{p_u} < e_{p_v} \\ 1 & \text{otherwise} \end{cases}$$

Provided

$$e_{p_u} - s_{p_u} = w_{p_u},$$

and

$$E_{p_u p_v} = e_{p_v} - e_{p_u} + K_{uv}c,$$

the constraint

$$\delta_{uv} - E_{p_u p_v} - H_v + c - w_{p_u} \geq 0$$

is equivalent to

$$e_{p_v} \leq s_{p_u} + \delta_{uv} + (1 - K_{uv})c - H_v,$$

which means that the constraints in earliest equation set are satisfied if and only if the earliest constraint graph has no negative cycle.

Therefore, the theorem is true. □

3.3.2. Statistical situation

In statistical clock schedule verification problem, we similarly construct the latest constraint graph and the earliest constraint graph based on the constraints, but the weights of edges in these two graphs are random variables. The following corollary can be easily established based on Theorem 3.6.

Corollary 3.6.1. *The probability that a given clock schedule is valid is equal to the probability that the latest constraint graph has no positive cycle and the earliest constraint graph has no negative cycle.*

In the deterministic situation, the negative (positive) cycle detection can be accomplished by the Bellman-Ford algorithm [**35**]. But to the best of our knowledge, there are no algorithms to deal with negative (positive) cycle detection problem when the edge weights are random variables. As mentioned before, it is difficult for iterative approaches to achieve *accurate* and *stable* results in the statistical situations because of the inaccuracy of the statistical operations. The following example confirms this. The Bellman-Ford algorithm can be extended to handle statistical situations: do the statistical min (max) operations instead of the deterministic comparison and assignment operations in the relaxation steps. Then the following theorem can be proved:

Theorem 3.7. *The probability of the existence of negative cycles in a graph $G(V, E)$ with random-weight edges is equal to*

$$1 - Pr(max_{(u,v)\in E}(d_v - d_u - w(u, v)) \leq 0),$$

where d_u is the computed distance label at vertex u after $|V| - 1$ iterations, and $w(u, v)$ is the weight of the edge (u, v).

But it is obvious that

$$max_{(u,v)\in E}(d_v - d_u - w(u, v)) \geq 0,$$

so

$$Pr(max_{(u,v)\in E}(d_v - d_u - w(u, v) \leq 0))$$

is actually equal to

$$Pr(max_{(u,v)\in E}(d_v - d_u - w(u, v) = 0)),$$

and the density distribution function of

$$max_{(u,v)\in E}(d_v - d_u - w(u,v))$$

is truncated at 0. The state-of-the-art SSTA algorithms [**14**, **103**] always assume that the maximum of two Gaussian distributed variables is still Gaussian distributed, then these methods must get the mean value of

$$max_{(u,v)\in E}(d_v - d_u - w(u,v))$$

larger than 0, which implies

$$Pr(max_{(u,v)\in E}(d_v - d_u - w(u,v)) \leq 0) < 50\%.$$

So when the actual yields are no less than 50%, the iterative approach cannot get *accurate* and *stable* results. We use non-iterative approaches to solve the statistical clock schedule verification problem.

3.4. Statistical checking of structural conditions

3.4.1. Statistical static timing analysis

The statistical static timing analysis algorithm introduced in [**14**] is used in our work to calculate the maximal delay between vertices. It used the PCA technique to transform a set of correlated parameters into an uncorrelated set. It assumed that the delay of a gate or an interconnect is normally distributed. After doing PCA, a delay can be represented as a linear function of principal components (independent random variables with standard

normal distributions):

$$\mathbf{d} = d_0 + k_1 \times \mathbf{p_1} + ... + k_m \times \mathbf{p_m},$$

where d_0 is the mean value, $\mathbf{p_i}$'s are independent principal components, and k_i's are coefficients. The *sum* function and *max* function of normally distributed random variables were provided, which can maintain the correlation information. Especially for the *max* function, Clark's method [**33**] was used to approximate the result, which assumes that the maximal of two random variable with normal distribution is also normally distributed. Then it used a PERT-like traversal on the circuit graph to calculate the latest arrival time of primary outputs.

3.4.2. Latest time constraints

Statistical verification of the latest time constraints needs to calculate the probability that all the cycles are non-positive in the latest constraint graph.

Let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of cycles, and $|\mathbf{c_i}|$ be the delay of cycle c_i. Let $F(|\mathbf{c_1}|, |\mathbf{c_2}|, \ldots, |\mathbf{c_n}|)$ be the joint cumulative distribution function (JCDF) of the delays of cycles, then the probability that all the cycles are non-positive is $F(0, 0, \ldots, 0)$, which is equal to

$$P_r(|\mathbf{c_1}| \leq 0, |\mathbf{c_2}| \leq 0, \ldots, |\mathbf{c_n}| \leq 0).$$

However, we know that

$$P_r(|\mathbf{c_1}| \leq 0, |\mathbf{c_2}| \leq 0, \ldots, |\mathbf{c_n}| \leq 0) = P_r(\max_{j=1}^{n}(|\mathbf{c_j}|) \leq 0).$$

Thus, if we can get the distribution of the maximum delay of all the cycles, we can get the probability of the circuit satisfying the latest time constraints. But the number of cycles

is often exponential in the number of vertices of a graph, so the methods based on the enumeration of cycles [**98**] are prohibitive.

However, if we can classify the edges of a directed graph into two disjoint sets such that each cycle is just formed by two simple paths, one composed of edges from each set, then the enumeration of cycles is not necessary.

Based on this idea, we perform depth-first search on the latest constraint graph, and classify the edges into two sets: one contains all the backward edges, and the other contains the rest. Then if the backward edges are deleted from the graph, the graph becomes a DAG, and we can use PERT traversal to compute the longest path between any two points. When they are combined with the weights of backward edges, we can get the weight distribution of the longest cycle.

Unfortunately, our study shows that some cycles may be missed in the above approach.

Theorem 3.8. *The edges in a directed graph cannot* always *be divided into two disjoint sets, such that any simple cycle is formed by two simple paths, one composed of edges from each set.*

A simple example to show the correctness of this theorem is shown in Fig. 3.6. Enumerating all the bi-partitions of the cycle A (a, b, c, d, e, f, a) of the graph in this example, we will see that all the partitions do not satisfy that all the simple cycles in this graph can be formed by two simple paths, one composed of edges from each set.

This theorem tells us that if more than one backward edge is involved in a cycle, they perhaps do not form a simple path on this cycle, and the traversal in any topological order of vertices ignoring these backward edges does miss this cycle. For example, for the graph in Fig. 3.7, if the topological order is (a, c, b, d), edges (b, c) and (d, a) are backward edges,

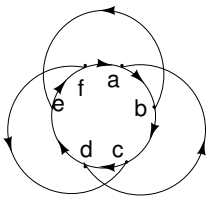


Figure 3.6. A case against simple path bi-partitioning.

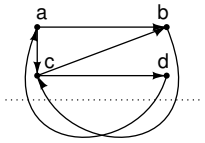


Figure 3.7. A cycle containing two backward edges.

and the cycle (a, b, c, d, a) involves these two edges that do not form a simple path, so this order misses this cycle.

Based on these discussions, we design a heuristic method by performing depth first search many times, and the order of choosing edges in each search is *randomly* selected. When the search is performed many times, the overall probability of missing cycles can be smaller. This algorithm PCycle is shown in Fig. 3.8.

We assume that the random variables $\mathbf{\Delta_{ji}}$, $\delta_{\mathbf{ji}}$, $\mathbf{e_i}$, $\mathbf{S_i}$, $\mathbf{H_i}$ in our statistical conservative formulation satisfy normal distribution. Any one of these random variables $\mathbf{r}$ can be represented by the linear function of independent variables:

$$\mathbf{r} = \mu + \sum_{i=1}^{m} a_i \mathbf{l_i}$$

where μ is the mean value of $\mathbf{r}$, $\mathbf{l_i}$'s are independent random variables, a_i's are constants, and m is the number of independent random variables. This representation can be always achieved using the models presented in [**1**] or [**14**]. Correlated variables can be represented by the linear function of independent random variables with standard normal distribution

Algorithm PCycle
Input: constraint graph $G(V, E)$
Output: probability of no positive cycles existing in G
Notations:
L_e: the weight of edge e
D_e: the maximal weight between the end vertex and start vertex of edge e
M: the maximal number of depth first search iterations.

Procedure:
1. Depth first search graph G to find all backward edges, stored in set $\mathcal{E}$;
2. For each edge e in $\mathcal{E}$ {
   t := 1;
   LOOP: While t<M+1 {
    depth first search graph G starting from the end vertex of e to find backward edges, stored in set $\mathcal{E}'$;
    topological sort graph G starting from the end vertex of e after ignoring all the edges in $\mathcal{E}'$;
    clear $\mathcal{E}'$;
    t := t+1;
    if the resulting order is ever found goto LOOP;
    calculate D_e using statistical timing analysis method;
    $C_e^t = D_e + L_e$;
   }
   $C_e := \max_{t=1}^{M}(C_e^t)$;
  }
3. Compute $\max_{e\in\mathcal{E}} C_e$ using statistical timing analysis method;
4. Probability of no positive cycles is equal to $P_r(max_{e\in\mathcal{E}}(C_e) \leq 0)$

Figure 3.8. The PCycle Algorithm.

using PCA. So for simplicity, we assume that each $\mathbf{l_i}$ is a standard normally distributed variable. So in PCycle, we can use statistical timing analysis methods [**14**, **103**] to compute the maximal delay of cycles.

An important property of PCycle is that it can get a good estimation of yield very efficiently. In section 3.5, we will see that it can get pretty accurate estimation of yield in much shorter time compared with Monte Carlo simulations. We also implemented a cycle-enumeration algorithm based on the algorithm proposed by [**98**], and found that it is only 4 times faster than the Monte Carlo simulation in most cases. So although PCycle may miss cycles, it is still a good algorithm to estimate the yield.

The complexity of this algorithm is $O(MN(V+E))$, where M is the number of depth first search iterations for each backward edge, N is the number of edges in backward edge set $\mathcal{E}$, V is the number of vertices in G, and E is the number of edges in G.

3.4.3. Earliest time constraints

Statistical verification of the earliest time constraints needs to calculate the probability that all the cycles are non-negative in the earliest constraint graph.

Similar to the verification of latest time constraints, let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of cycles, and $|\mathbf{c_i}|$ be the weight of cycle c_i. Let $F(|\mathbf{c_1}|, |\mathbf{c_2}|, \ldots, |\mathbf{c_n}|)$ be the JCDF of the weight of cycles, then we can calculate the probability that all cycles are non-positive, that is,

$$P_r(|\mathbf{c_1}| \geq 0, |\mathbf{c_2}| \geq 0, \ldots, |\mathbf{c_n}| \geq 0).$$

However, we know that

$$P_r(|\mathbf{c_1}| \geq 0, |\mathbf{c_2}| \geq 0, \ldots, |\mathbf{c_n}| \geq 0) = P_r(\min_{j=1}^{n}(|\mathbf{c_j}|) \geq 0).$$

Algorithm NCycle

Input: constraint graph $G(V, E)$
Output: probability of no negative cycles existing in G

Procedure:
1. Split vertex O into O_1 and O_2, all the outgoing edges of O in original G are outgoing from O_1, and all the incoming edges of O in original G are incoming edges of O_2;
2. PERT-traversal the new G to calculate the shortest distance $\mathbf{D}$ from O_1 to O_2;
3. Probability of no negative cycles is equal to $P_r(\mathbf{D} \geq 0)$

Figure 3.9. The NCycle Algorithm.

This problem is similar to the latest time verification: we only need to calculate the distribution of the minimal weight of all the cycles in the earliest constraint graph. However, since the earliest constraint graph is much simpler, where every cycle in the earliest constraint graph includes the vertex O, the verification can be done optimally. NCycle, the earliest time constraints verification algorithm, is shown in Fig. 3.9. Since all cycles are considered in NCycle, it can check the earliest time constraints accurately. Since

$$\min(\mathbf{A}, \mathbf{B}) = -\max(-\mathbf{A}, -\mathbf{B}),$$

the min operations involved here can be transformed to max operations, thus we can still use Clark's method here.

3.4.4. Combined verification

It is obvious that the cycles in the latest constraint graph share some elements with the cycles in the earliest constraint graph. So there exist correlations between the probabilities that the clock schedule does not violate the latest or earliest time constraints. In the above two subsections, we have shown how to calculate the probability that the clock schedule does not violate the latest or earliest time constraints respectively. However, since the earliest time and the latest time are correlated, multiplying the probabilities of correct conditions in them does not give the right answer. Therefore, we need a combined verification.

In PCycle, let

$$\mathbf{A} = \max_{e \in \mathcal{E}} \mathbf{C_e}$$

and in NCycle, let

$$\mathbf{B} = \min_{e \in \mathcal{E}} \mathbf{C_e}$$

then the probability that latest constraint graph has no positive cycles and earliest constraint graph has no negative cycles is equal to

$$P_r(max(\mathbf{A}, -\mathbf{B}) \leq 0).$$

When the correlation coefficient of A and $-B$ is more negative, the Clark's method is more inaccurate. For example, $A = (-1, 4)$, $-B = (-1, 4)$, and their correlation coefficient is -1. We compute $Pr(max(A, -B) \leq 0)$, then the result of Clark's method has a 7% error compared with the result of MC simulation. Here, $\mathbf{A}$ and $-\mathbf{B}$ often fall into this situation. Furthermore, we do not need to keep the linear function format for the results in this step.

So in order to improve the accuracy, we use MC simulation to accurately calculate the probability that the maximum of $\mathbf{A}$ and $-\mathbf{B}$ is not positive.

3.4.5. Latch-based pipeline designs

Recently, some researches are focusing on the statistical static timing analysis for latch-based pipeline designs [**16**, **115**]. Besides the computation of the latest and earliest arrival time at the primary outputs, we need to compute the probability that the arrival time on latches violates setup-time and hold-time constraints, which is the *data transmission error probability*. Not involving any cycles, this problem is similar with the earliest constraint verification problem, and can be efficiently and accurately solved by our algorithm.

For the pipeline design with level-sensitive latches, since the pipeline design does not involve cycles, we can use PERT-like traversal to compute the arrival time at the primary outputs. The remaining task is to compute the data transmission error probability. We first construct the latest constraint graph G_p and the earliest constraint graph G_n. Similar with NCycle algorithm, vertex O in G_p (G_n) is split into two vertices O_1 and O_2: all the outgoing edges of O in original graph G_p (G_n) are outgoing from O_1, and all the incoming edges of O in original G_p (G_n) are incoming edges of O_2. Then G_p (G_n) becomes a DAG, so we can traverse the new G_p (G_n) from O_1 to calculate the maximal (minimal) weight from O_1 to O_2, which is also the maximal (minimal) weight of all cycles in the original G_p (G_n). Then we can use the combined verification technique to compute the data transmission error probability.

3.4.6. Cycle-breaking strategy

PCycle may miss some cycles, so it may over-estimate the yield. Another important aspect of the statistical clock schedule problem is to *conservatively* estimate the yield.

Inspired by the cycle-breaking ideas proposed in [**13**], we break the cycles at the positions of latches instead of at the backward edges. For each cycle in the latch graph, we select one latch to be treated as a flip-flop. These flip-flops are called *virtual flip-flops*. In order to get a tight lower bound of the yield, we need to select an appropriate set of latches to be treated as virtual flip-flops, and set appropriate departure time for each virtual flip-flop.

Let X_i be the weight of the edge from O to D_i, and Y_i be the weight of the edge from A_i to O. If we select latch i to be treated as a virtual flip-flop, the physical meanings of X_i and $-Y_i$ are the departure time and the arrival time of the flip-flop respectively.

Definition 3.1 (Cycle break operation)**.** *In the latest constraint graph for the clock schedule verification, for latch i, delete the edge from A_i to D_i, and set X_i and Y_i such that X_i and Y_i satisfy the following three conditions:*

$$X_i + Y_i = 0,$$

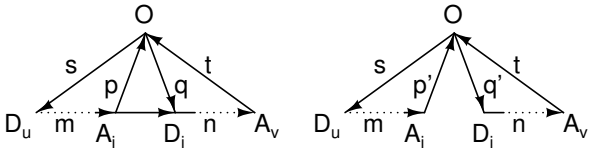
$$Y_i \geq S_i - c,$$

and

$$X_i \geq c - w_{p_i}.$$

This operation on the graph is called Cycle Break Operation (CBO).

An important property of CBO is that



(a) The original latest constraint graph (b) The constraint graph after CBO

Figure 3.10. Cycle break operation.

Theorem 3.9. *CBO is conservative, that is, if there are no positive cycles in the graph after CBO, there are no positive cycles in the original graph.*

Proof. Part of a latest constraint graph and its corresponding graph after the CBO on latch i are shown in Figure 3.10(a) and (b), where s, t, p, q, m, n, p' and q' are the weights of the corresponding paths. According to the definition of CBO, p' and q' satisfy

$$p' + q' = 0,$$

$$p' \geq S_i - c,$$

and

$$q' \geq c - w_{p_i}.$$

Also according to the weight definition of the latest constraint graph, $p = S_i - c$ and $q = c - w_{p_i}$. So $p' \geq p$ and $q' \geq q$.

For any cycle L in the original constraint graph,

- If L does not pass through the edge from A_i to D_i, since the edge weights do not decrease in the CBO, if L is not a positive cycle in the new graph, L is not a positive cycle in the original graph either, which means that the CBO is conservative.
- If L passes through the edge from A_i to D_i, there are two situations.

- If L does not pass vertex O, we can replace the edge from A_i to D_i in L by one edge from A_i to O and another edge from O to D_i, and it is obvious that the weight of the new cycle is the same with the weight of L.
- If L passes vertex O, we do the above replacement, and we get two new cycles L_1 and L_2. As shown in Figure 3.10, the weight of L is equal to

$$s + t + m + n,$$

and the weights of L_1 and L_2 are equal to

$$s + m + p'$$

and

$$t + n + q',$$

respectively. If

$$s + m + p' \leq 0$$

and

$$t + n + q' \leq 0,$$

we get

$$(s + m + p') + (t + n + q') \leq 0,$$

while $p' + q' = 0$, so $s + m + t + n \leq 0$, which means that if L_1 and L_2 are not positive cycles, L is not a positive cycle either.

So the CBO is always conservative. □

Let $\mu_{\mathbf{a}}$ and $\sigma_{\mathbf{a}}$ denote the mean value and the deviation of random variable $\mathbf{a}$ respectively. Let $PreSet(i)$ be the set of latches that can reach latch i through a combinational path. Let $Best_i$ and $Worst_i$ represent the earliest and the latest arrival time of latch i respectively when the combinational delays between any two latches j and i are $\mathbf{\Delta_{ji}}$, that is,

$$Best_i = \min_{j \in PreSet(i)} (c - w_{p_j}) + \mu_{\mathbf{\Lambda_{ji}}} - 3\sigma_{\mathbf{\Lambda_{ji}}},$$

and

$$Worst_i = \max_{j \in PreSet(i)} c + \mu_{\mathbf{\Lambda_{ji}}} + 3\sigma_{\mathbf{\Lambda_{ji}}},$$

where $\mathbf{\Lambda_{ji}} = \mathbf{\Delta_{ji}} - \mathbf{E_{ji}}$.

We define the weight of a latch as

$$LW_i = \min(Worst_i, c) - \max(Best_i, c - w_{p_i}).$$

Now, we propose our latch selection algorithm. We need to select as few latches as possible in breaking the cycles. We split the vertex O into two vertices O_1 and O_2 to break the cycles involving the vertex O: all the outgoing edges of O in the original graph are outgoing from O_1, and all the incoming edges of O in the original graph are incoming edges of O_2. So the cycles involving the vertex O are broken at O, so in the following we are focusing on how to select latches to break the cycles not involving vertex O. For each latch i, the corresponding vertices A_i and D_i are collapsed to be a single vertex i: all the incoming edges of A_i in the original graph are incoming edges of i, and all the outgoing edges of D_i in the original graph are outgoing edges of i. We set the weight of vertex i to be LW_i. Then our latch selection problem becomes the well-known minimum weighted feedback vertex set

problem, which has been proved to be NP-complete [**45**]. We use the heuristic algorithm proposed by Demetrescu and Finochi [**39**] to compute the solutions.

After the selection of latches, we need to set the departure time for each virtual flip-flop, or determine X_is and Y_is. The problem to determine X_is and Y_is can be formulated as:

$$Maximize \;\; Pr(L \leq 0),$$

s.t.

$$L = max_{i \mapsto j}(X_i + Y_j + L_{ij})$$

$$c - w_{p_i} \leq X_i \leq c \; \forall i \in Sel$$

$$Y_i = -X_i \; \forall i \in Sel$$

$$X_i = c - w_{p_i} \; \forall i \notin Sel$$

$$Y_i = S_i - c \; \forall i \notin Sel$$

where L_{ij} is the maximal weight of the paths from i to j, $i \mapsto j$ means that latch i can reach latch j, and Sel is the set of virtual flip-flops (the selected latches).

From the proof of Theorem 6, we can see that the yield pessimism comes from the increase of the probabilities that the cycles not through the edge (A_i, D_i) in CBO are positive. Thus, if we can reduce this probability increase, the yield pessimism is expected to be reduced. Furthermore, since the correlations between path weights are typically positive, we may increase the $Pr(L \leq 0)$ by separately increasing

$$Pr(X_i + Y_j + L_{ij} \leq 0)$$

on each path $i \mapsto j$.

Based on these, we propose a novel approach to determine X_is. Starting from each vertex a connected with O_1, we traverse the graph to compute the maximal weights of the paths from a to the vertices connected with O_2. Since all the edge weights involved in this computation are known, for each pair of vertices, we can compute the probability that the maximal weight of the paths between them is positive, and insert it into a max-heap. Then, we use a greedy algorithm to set the unknown X_is. We assume that the setup time of latches is 0. The algorithm selects the paths one by one in the non-increasing order of the probabilities that the path weights are positive. For each selected path from D_i to A_j in the graph after CBOs, there exists a cycle $\{O \rightarrow D_i \rightarrow A_j \rightarrow O\}$ with weight

$$L_{ij} + (c - w_{p_i}) - c = L_{ij} - w_{p_i}$$

in the original constraint graph, so the greedy algorithm always sets X_i and Y_j such that

$$Pr(X_i + Y_j + L_{ij} \leq 0)$$

is as close to

$$Pr(L_{ij} - w_{p_i} \leq 0)$$

as possible, which means that the increase of the probabilities that the cycles not through (A_i, D_i) are positive is minimized.

After the determination of the X_is, we can traverse the constraint graph from O_1 to calculate the distribution of the maximal weight of the paths from O_1 to O_2, then we can compute the lower-bound of the timing yield using the combined verification algorithm.

Table 3.1. Comparison results of PCycle and Monte Carlo Simulation Method

circuit				PCycle		Monte Carlo		error%
name	inputs#	latches#	gates#	yield%	time(s)	yield%	time(s)	
s27	4	3	10	94.41	0.01	94.39	96	0.02
s298	3	14	119	95.35	0.36	95.41	170	-0.06
s349	9	15	161	97.72	0.93	97.72	498	0.00
s526	3	21	193	95.35	0.22	95.25	287	0.10
s820	18	5	289	92.79	0.24	93.07	3,381	-0.28
s1238	14	18	508	96.71	0.21	96.67	3,312	0.04
s1488	19	6	653	87.49	4.43	87.86	3,651	-0.37
s1494	8	6	647	88.69	6.47	88.32	3,050	0.37
s5378	35	179	2,779	83.15	10.98	82.99	31,965	0.16
s9234	19	228	5,597	92.92	175.55	93.24	66,351	-0.32
s13207	31	669	7,951	93.94	1,024.79	93.48	118,046	0.46
s15850	14	597	9,772	88.49	1,154.83	87.70	164,130	0.79
s35932	35	1,728	16,065	96.25	938.28	96.12	392,806	0.13

Table 3.2. Comparison results of NPCycle and Monte Carlo Simulation Method

circuit	NPCycle		Monte Carlo		error
	yield%	time(s)	yield%	time(s)	%
s27	92.15	0.05	92.13	180	0.02
s298	93.60	0.48	93.47	273	0.13
s349	92.91	2.00	92.71	970	0.20
s526	93.59	1.86	93.71	580	-0.12
s820	90.68	1.82	90.66	4,310	0.02
s1238	89.30	1.81	89.47	4,416	-0.17
s1488	83.88	6.82	83.89	4,261	-0.01
s1494	84.91	8.23	84.59	4,710	0.32
s5378	78.08	18.72	77.65	41,813	0.43
s9234	88.13	207.22	88.00	104,935	0.13
s13207	0.00	140.12	0.00	113,321	0.00
s15850	0.00	160.31	0.00	153,625	0.00
s35932	94.08	1,342.00	-	>10 d	-

3.5. Experimental results

PCycle, NCycle, and the combined verification algorithm NPCycle have been implemented and tested on the ISCAS89 benchmark circuits where the flip-flops are replaced by

level-sensitive latches. For simplicity, we have only considered the single phase clock, and since PCA can transform the correlated variables to uncorrelated variables, any random variable **r** in statistical conservative formulation can be represented by a linear function of independent random variables with standard normal distribution:

$$\mathbf{r} = r_0 + k_1 \times \mathbf{p_1} + ... + k_m \times \mathbf{p_m},$$

where r_0 is the mean value, $\mathbf{p_i}$'s are independent random variables with standard normal distribution, and k_i's are coefficients. Then a random number generator is used to generate r_0 and k_i for $i = 1, ..., m$. All the random variables are normally distributed with 10-20% deviation. In our test cases, the number of random variables (m) is between 10 and 100, and $|k_i| \leq 5$. This test case generator introduces the spatial correlations of process variations of gates, interconnects and the clock network. For example, if for some $1 \leq i \leq m$, $k_i \neq 0$ for one gate, and $k_i \neq 0$ for another gate, there exists spatial correlation between the delay of these two gates. All experiments are run on a Linux PC with a 2.4G Hz CPU and 2.0 GB memory.

To verify the results of the PCycle, we used MC simulation as a comparison. In each iteration, MC simulation assigns values to $\mathbf{p_i}$ for $i = 1, ..., m$, then calculates the gate delays and clock schedule, then performs Bellman-Ford shortest-path algorithm to detect positive cycles. Here, the assigned values of $\mathbf{p_i}$ for $1 \leq i \leq m$ satisfy standard normal distribution. We run 10,000 iterations for each case in the MC simulations. A comparison of results on the latest time constraints is shown in Table 3.1. The number of depth first search (M) in PCycle is always less than 10. We can see that the results of PCycle are very close to

the MC results with an average error of 0.24%, which confirms that the influence of missed cycles in PCycle is insignificant. PCycle is also much faster than the MC simulations.

A comparison of the results by the NPCycle with those by the MC simulations is shown in Table 3.2. For each case, MC simulation performs the Bellman-Ford algorithm to detect positive cycles in the latest time constraint graph, if no positive cycles are found, then it performs Bellman-Ford algorithm to detect negative cycles in the earliest time constraint graph. Since there exist a pair of latches between which the minimal combinational delays are 0 in s13207 and s15850, there always exist earliest constraint violations in them, so the timing yields of them are 0%. We can see that the results from NPCycle are very close to the MC simulation results with an average error of 0.16%. The circuit with the largest run time, s35932, is verified in 1342 seconds, while the MC simulation cannot finish within 10 days.

We also implemented the conservative timing yield estimation algorithm CBVerify. CBVerify computes the yields considering only the latest time constraints. Many cycles in ISCAS89 benchmark circuits contain only one flip-flop, so in order to test the performance of our latch selection strategy, we transformed the circuits by replacing edge-triggered devices with pairs of level-sensitive latches controlled by a clock with dual phases, and then the circuits are retimed to minimize cycle time using the retiming algorithm proposed by Maheshwari and Sapatnekar [**66**]. A comparison of the results by the CBVerify with those by the MC simulations is shown in Table 3.3, where the "Strat. A" is the latch selection strategy to select latches with the least LWs, the "Strat. B" is the latch selection strategy to select latches with the largest LWs. Since it is obvious that our algorithm is much more efficient than MC simulations, we do not report the running time here. We can see that both strategies get the conservative yields. Especially, Strategy A gets much more accurate yield

Table 3.3. Comparison results of CBVerify and Monte Carlo simulation

circuit		MC	Yield of CBVerify (%)	
name	#latch	Yield(%)	Strat. A	Strat. B
s5378	844	98.87	98.86	74.86
s9234	990	91.25	88.30	83.13
s13207	1,295	89.50	89.25	89.07
s15850	1,719	100.00	100.00	22.06
s35932	5,207	98.87	98.81	98.30

estimation than Strategy B, which confirms that the latch selection strategy influences the accuracy of the yield estimation greatly.

3.6. Conclusion

In statistical clock verification problem, because of the complex statistical correlations and the accumulated inaccuracy of statistical operations, traditional iterative approaches are difficult to get accurate results. Instead, a statistical checking of the structural conditions for correct clocking is proposed, where the central problem is to compute the probability of having a positive cycle in a graph with random edge weights. We proposed two algorithm to handle this problem. The proposed methods traverses the graph several times to reduce the correlations among iterations. Although the first algorithm is a heuristic algorithm that may miss cycles, experimental results showed that it has an error of 0.16% on average in comparisons with the MC simulations. As an important application, this algorithm can accurately solve the statistical static timing analysis problem in latch-based pipeline design. The second algorithm is based on a cycle-breaking technique, and can conservatively estimate the timing yield. Both algorithms are much more efficient than the Monte Carlo simulation.

CHAPTER 4

Timing Budgeting under Arbitrary Process Variations

Process variation has become a critical problem in modern VLSI fabrication. The traditional corner-based analysis and optimization techniques become prohibitive. Statistical static timing analysis (SSTA) techniques as [**14**, **103**] propagate distributions instead of single values, and achieve good efficiency. Based on them, many statistical optimization techniques [**29**, **32**, **38**, **48**, **67**, **92**] also emerged. We are considering the following statistical optimization problem. Given the constraint on the maximal delay from primary inputs to primary outputs in a combinational circuit, minimize the worst total cost (e.g., leakage power) such that the timing yield is at least a given value. Fig. 4.1 shows a general flow of statistical circuit optimization. The key sub-problem is the timing budgeting, i.e., redistribute the timing slacks to elements. Fig. 4.2 shows an example of timing budgeting.

There are already several works [**23**, **46**, **67**, **94**] solving the timing budgeting problem. Most of them do not consider process variations. Mani *et al.* [**67**] transformed the timing budgeting problem under uncertainty to a second order conic programming (SOCP) problem. All these works put budgets on the mean of the delay, but when a gate is sized up or down, both the mean and the variance of the delay change. Without considering the change of the variance, the final result may suffer a big timing violation. For example, as shown in Fig. 4.2, the gate g needs to be sized down. Suppose the delays before sizing and after sizing are d_g and d'_g respectively, and the budget on the mean of the delay is b. After the gate is

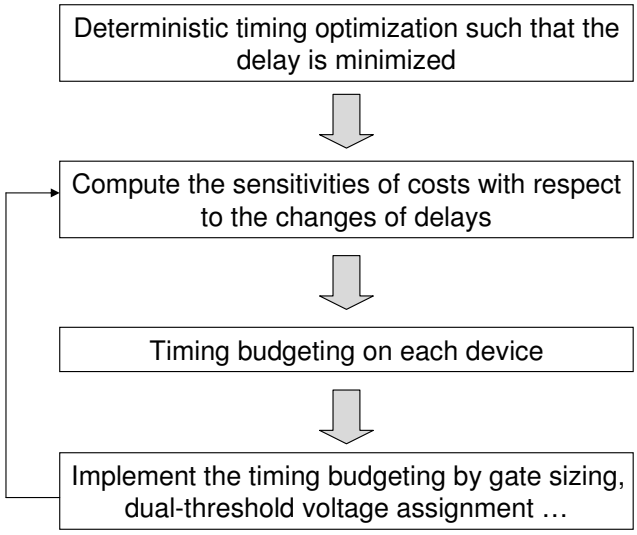


Figure 4.1. A general flow of statistical optimization.

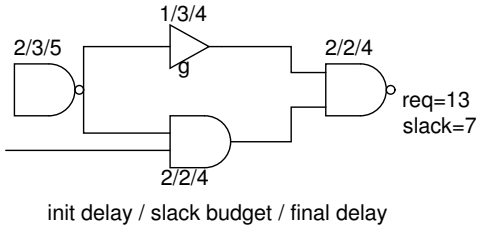


Figure 4.2. Timing budgeting is the redistribution of slacks.

sized down, the delay budget is fully utilized, so the worst delay of the gate is expected to satisfy

$$\mu(d_g') + 3\sigma(d_g') = (\mu(d_g) + b) + 3\sigma(d_g).$$

But the variance may increase, then the actual worst delay of the gate

$$\mu(d_g') + 3\sigma(d_g') > (\mu(d_g) + b) + 3\sigma(d_g),$$

which means the actual worst delay is greater than the expected worst delay. So the timing constraint is violated. Thus, considering only the change of the mean in the budgeting step cannot guarantee that the timing constraint is satisfied after sizing.

Most of existing works optimize the cost at a late stage: they assume that the distributions of variables are known. An early-stage design can improve design greatly since it has more flexibility. But the detailed information of the distributions is not known at the early stage. How to explore limited information (e.g. the mean and the bounds of the distributions) in an early-stage design is the focus of this chapter. There are already some works done on this topic [**65**, **91**, **105**, **106**], but all of them are on the analysis or modeling, not the optimization.

The contribution of our work includes:

- Our new formulation considers the budgeting of both the mean and the variance of the delay, so the timing violation is greatly reduced.
- The underlying distribution is allowed to be any symmetric distribution. So our technique can be used at both an early stage and a late stage.
- The timing budgeting problem is transformed to a linear programming problem through a pessimism controllable robust optimization technique, and is efficiently solved.
- Our block-level timing budgeting reduces the timing pessimism compared with gate-level timing budgeting.

The rest of this chapter is organized as follows. Section 4.1 briefly discusses how timing budgeting should be conducted. Section 4.2 presents our theory to handle timing budgeting under uncertainty. Section 4.3 proposes the application of our theory in gate sizing, and also

proposes the block-level timing budgeting technique to reduce timing pessimism. Section 4.4 shows the experimental results. Finally, the conclusions are drawn in Section 4.5.

4.1. Timing budgeting for optimization

The general timing budgeting problem without uncertainty is

$$\text{Maximize} \quad \sum_{(i,j)\in E} c_{ij} b_{ij} \tag{4.1}$$

$$\text{s.t.} \quad t_j - t_i \geq d_{ij} + b_{ij} \quad \forall (i,j) \in E \tag{4.2}$$

$$0 \leq b_{ij} \leq \delta \qquad \forall (i,j) \in E \tag{4.3}$$

where d_{ij} is the existing delay on edge (i,j), b_{ij} is the delay budget assigned on edge (i,j), δ is a small positive real number, and c_{ij} is the sensitivity of the cost to the change of the delay. For example, in the leakage power minimization problem,

$$c_{ij} = \frac{\Delta P}{\Delta D},$$

which represents the leakage power reduction for one unit of delay change. The power-delay curve is not linear in general, so in order to make the linear approximation valid, we need to restrict the range of the delay change within a small range, so we have $b_{ij} \leq \delta$. $b_{ij} \geq 0$ implies the delays of gates cannot decrease, so the current solution is always a feasible solution in the next iteration. If b_{ij} can be negative, because of the discrete nature of gate delays, we may not find a feasible solution in the budgeting implementation step.

Suppose the maximal delay from primary inputs to primary outputs should not be greater than D_{constr}. We introduce two dummy nodes s and t: s is connected to all the primary inputs, and t is connected from all the primary outputs. We also introduce an edge from t

to s with delay $-D_{constr}$. Then

$$t_s - t_t \geq -D_{constr}$$

is put into the formulation to enforce the timing constraint.

As shown in Fig. 4.1, after timing budgeting, we need to size up/down gates or assign different threshold voltages to gates to implement the computed budgets. This is just a local optimization problem, and thus can be efficiently solved.

When process variations are considered, sensitivities (c_{ij}) and delays are no longer deterministic. Let $\hat{D}$ represent an element (e.g., a constant or a function) D with uncertainty. The timing budgeting problem under uncertainty was formulated as:

$$\text{P1: Maximize} \quad \sum_{(i,j)\in E} \hat{c}_{ij} b_{ij} \tag{4.4}$$

$$\text{s.t.} \quad t_j - t_i \geq \hat{d}_{ij} + b_{ij} \quad \forall (i,j) \in E \tag{4.5}$$

$$0 \leq b_{ij} \leq \delta \qquad \forall (i,j) \in E \tag{4.6}$$

Note that budgets (b_{ij}) are deterministic here.

The computation of $\hat{c}_{ij}$ is as follows. For each gate in the circuit, we replace it by another gate of the same type from the library, and compute the statistical cost difference $\Delta\hat{P}$ of these two gates, and also the difference ΔD between the mean delays of these two gates. Then we compute

$$\frac{\Delta\hat{P}}{\Delta D},$$

and finally select the one with maximal mean value as the $\hat{c}_{ij}$. Note that the delay difference is deterministic. The reason is that the $\hat{c}_{ij}$ is the cost sensitivity to the *deterministic* shift (b_{ij}) of the mean value.

4.2. Budgeting by robust optimization

The formulation P1 does not consider the change of variance when the mean of the delay is shifted by b_{ij}, so as mentioned before, it may introduce the violation of timing constraints.

We propose the following general formulation that can handle the changes of the variances of delays.

$$\text{Maximize} \qquad \sum_{(i,j)\in E} \hat{f}_{ij}(b_{ij}) \tag{4.7}$$

$$\text{s.t.} \quad t_j - t_i \geq \hat{d}_{ij} + \hat{g}_{ij}(b_{ij}) \quad \forall (i,j) \in E \tag{4.8}$$

$$0 \leq b_{ij} \leq \delta \qquad \forall (i,j) \in E \tag{4.9}$$

where functions $\hat{f}_{ij}(b_{ij})$ and $\hat{g}_{ij}(b_{ij})$ give the reduction of cost and the change of the edge delay when the mean of the delay on edge (i,j) is shifted by b_{ij}, respectively.

We assume that $\hat{f}_{ij}(b_{ij})$ is a linear function of b_{ij}, and

$$\hat{f}_{ij}(b_{ij}) = \hat{c}_{ij} b_{ij}.$$

For the function $\hat{g}_{ij}$, [**48**] shows that the sensitivity of the delay of a gate to the variance of a parameter can be approximated as the product of a constant and the mean value of the delay, i.e., suppose the delay is represented in the linear form:

$$\hat{d}_{ij} = d_{ij}^{(0)} + \sum_{1\leq k\leq n} d_{ij}^{(k)} \hat{\Delta}_k,$$

where $d_{ij}^{(0)}$ is the mean of d_{ij}, $\hat{\Delta}_k$ is the variance of the kth parameter, and $d_{ij}^{(k)}$ is the sensitivity of $\hat{d}_{ij}$ to $\hat{\Delta}_k$, then

$$d_{ij}^{(k)} = \alpha_{ij}^{(k)} d_{ij}^{(0)},$$

where $\alpha_{ij}^{(k)}$ is a constant when $d_{ij}^{(0)}$ changes. Based on this, we get

$$\hat{g}_{ij}(b_{ij}) = \left[1 + \sum_{k} (\alpha_{ij}^{(k)} \hat{\Delta}_k)\right] b_{ij}.$$

Then Eq.(4.8) becomes

$$t_j - t_i \geq (d_{ij}^{(0)} + b_{ij}) + \sum_{1 \leq k \leq n} \left[\alpha_{ij}^{(k)} (d_{ij}^{(0)} + b_{ij})\right] \hat{\Delta}_k,$$

from which we can see that the variance also changes when the budget b_{ij} on the mean value changes. Thus, the new formulation captures the change of the variance when the mean of the delay shifts.

In summary, the formulation is

$$\text{P2: Maximize} \sum_{(i,j) \in E} \hat{c}_{ij} b_{ij}$$

s.t. $\forall (i,j) \in E$:

$$t_j - t_i \geq (d_{ij}^{(0)} + b_{ij}) + \sum_{1 \leq k \leq n} \left[\alpha_{ij}^{(k)} (d_{ij}^{(0)} + b_{ij})\right] \hat{\Delta}_k \tag{4.10}$$

$$0 \leq b_{ij} \leq \delta \tag{4.11}$$

Robust optimization is a generic technique to handle this kind of mathematical programming problem under uncertainty. Bertsimas and Sim [**10**] proposed a robust optimization technique to transform a linear programming problem under uncertainty to a linear programming problem without uncertainty. We use this technique to transform our timing budgeting problem to a linear programming problem.

Without loss of generality, we assume that $\hat{\Delta}_k \in [-1, 1]$. In reality, it is unlikely that all of the parameters will change simultaneously. A real number $\Gamma_{ij} \in [0, n]$ is used to control the robustness of the constraint Eq.(4.10). If only $\lfloor \Gamma_{ij} \rfloor$ of parameters are allowed to change, and one parameter changes by at most $(\Gamma_{ij} - \lfloor \Gamma_{ij} \rfloor)$, the following robust optimization technique guarantees the feasibility of the solution in the problem P2.

$$
\begin{aligned}
\text{P3: Maximize } & \sum \hat{c}_{ij} b_{ij} \\
t_j - t_i \geq & (d_{ij}^{(0)} + b_{ij}) + \\
& \max_{\{S_{ij} \cup \{t\} | S_{ij} \subseteq \{1..n\}, |S_{ij}| = \lfloor \Gamma_{ij} \rfloor, t \in \{1..n\} \setminus S_{ij}\}} \left\{ \sum_{k \in S_{ij}} y_k + (\Gamma_{ij} - \lfloor \Gamma_{ij} \rfloor) y_t \right\} \\
y_k \geq & |\alpha_{ij}^{(k)} (d_{ij}^{(0)} + b_{ij})| \\
0 \leq & b_{ij} \leq \delta
\end{aligned}
$$

After transformation, the following formulation is equivalent to P3.

$$
\begin{aligned}
\text{Maximize } & \sum \hat{c}_{ij} b_{ij} \\
t_j - t_i & \geq d_{ij}^{(0)} + b_{ij} + \Gamma_{ij} z_{ij} + \sum_{1 \leq k \leq n} q_{ij}^{(k)} \\
z_{ij} + q_{ij}^{(k)} & \geq \left| \alpha_{ij}^{(k)} (d_{ij}^{(0)} + b_{ij}) \right| \\
z_{ij} & \geq 0 \\
q_{ij}^{(k)} & \geq 0 \\
0 \leq & b_{ij} \leq \delta
\end{aligned}
$$

In this formulation, all the constraints are linear. Similarly, we transform the objective function to a linear function without uncertainty. The final formulation is

P4: Maximize Y

$$
\begin{aligned}
Y &\leq c_{ij}^{(0)} b_{ij} - \Gamma_0 z_0 - \sum_{1 \leq k \leq n} q_0^{(k)} \\
t_j - t_i &\geq d_{ij}^{(0)} + b_{ij} + \Gamma_{ij} z_{ij} + \sum_{1 \leq k \leq n} q_{ij}^{(k)} \\
z_0 + q_0^{(k)} &\geq \left| \sum_{(i,j) \in E} c_{ij}^{(k)} b_{ij} \right| \\
z_{ij} + q_{ij}^{(k)} &\geq \left| \alpha_{ij}^{(k)} (d_{ij}^{(0)} + b_{ij}) \right| \\
z_0 &\geq 0 \\
q_0^{(k)} &\geq 0 \\
z_{ij} &\geq 0 \\
q_{ij}^{(k)} &\geq 0 \\
0 \leq b_{ij} &\leq \delta
\end{aligned}
$$

which is a linear programming problem. Thus, we can efficiently solve this.

If the problem does not satisfy the constraint on the maximal number of parameters allowed to change, [**10**] proved that if the distributions of those variables under uncertainty are symmetric, the probability that the timing constraint on edge (i, j) is violated is upper-bounded by

$$B(n, \Gamma_{ij}),$$

where

$$B(n,\Gamma_{ij}) = \frac{1}{2^n}\left\{(1-\mu)\binom{n}{\lfloor v\rfloor} + \sum_{l=\lfloor v\rfloor+1}^{n}\binom{n}{l}\right\} \tag{4.12}$$

where $v = \frac{\Gamma_{ij}+n}{2}$ and $\mu = v - \lfloor v\rfloor$. So if

$$\forall(i,j) \in E, B(n,\Gamma_{ij}) \leq 1-\eta,$$

where η is the yield constraint on the path delay, the timing yield is satisfied. Thus, our objective is to select Γ_{ij} such that $B(n,\Gamma_{ij}) \leq 1-\eta$. We can use a look-up-table (LUT) to store the values of $B(n,\Gamma_{ij})$ for a set of Γ_{ij}s, then the determination of Γ_{ij} is straight forward. A good property of this bound is that it does not depend on the input data (e.g., the distribution of d_{ij}), so we can use it for arbitrary symmetric distributions.

4.3. Application to gate sizing

In order to verify the effectiveness of our budgeting technique, we apply it to the statistical gate sizing problem. The problem is described as follows.

Problem 4.1. *Given a combinational circuit and a library of gates with different sizes, optimize the circuit such that the worst total cost (e.g., leakage power) is minimized while the timing yield is at least a given value.*

This problem can be formally formulated as:

$$Minimize \sum_{(i,j)\in E} \hat{C}(\hat{d}_{ij})$$

$$\text{s.t. } Pr(\max_{path}(\hat{D}_{path}) \leq D_{constr}) \geq \eta$$

$$\hat{D}_{path} = \sum_{(i,j)\in path} \hat{d}_{ij}$$

where function $\hat{C}(\hat{d}_{ij})$ gives the cost when the delay on edge (i,j) is $\hat{d}_{ij}$, $\hat{D}_{path}$ is the maximal delay of the *path*, D_{constr} is the delay constraint, and η is the timing yield.

The global timing constraint on all the paths is transformed to local timing constraint on each path:

$$\begin{aligned} &Minimize \sum_{(i,j)\in E} \hat{C}(\hat{d}_{ij}) \\ &\text{s.t. } Pr(\hat{D}_{path} \le D_{constr}) \ge \eta \qquad \forall\text{path from PIs to POs} \\ &\hat{D}_{path} = \sum_{(i,j)\in path} \hat{d}_{ij} \end{aligned}$$

Note that mathematically it is not guaranteed that this transformation is conservative, but as shown in [**68**], it is conservative in practice because of high correlations between path delays.

Since the number of paths is exponential with respect to the number of nodes and edges, we need to transform the timing constraint on each path to more local constraints. Previous research [**67**] distributed the constraint to each gate. In that case, there is no difference between the worst-case approach and this gate level approach in the timing part since they both use $\mu + \Phi^{-1}(\eta)\sigma$ as the delay of a gate. We propose the block-level timing budgeting that can reduce the pessimism on timing.

4.3.1. Block-level timing budgeting

We partition the circuit into many blocks; enumerate all the sub-paths in each block; write one timing constraint for each sub-path. Fig. 4.3 shows an example circuit. The dotted block has four gates, the five sub-paths within the block are enumerated, and each sub-path has one timing constraint:

$$t_4 - t_1 \geq d_{1,x} + d_{x,y} + d_{y,4} + b_{1,x} + b_{x,y} + b_{y,4}$$

$$t_4 - t_1 \geq d_{1,x} + d_{x,z} + d_{z,4} + b_{1,x} + b_{x,y} + b_{z,4}$$

$$t_4 - t_2 \geq d_{2,x} + d_{x,y} + d_{y,4} + b_{1,x} + b_{x,y} + b_{y,4}$$

$$t_4 - t_2 \geq d_{2,x} + d_{x,z} + d_{z,4} + b_{1,x} + b_{x,y} + b_{z,4}$$

$$t_4 - t_3 \geq d_{3,z} + d_{z,4} + b_{3,z} + b_{z,4}$$

where $d_{x,y}$ and $b_{x,y}$ are the delay and the delay budget on edge (x, y), respectively.

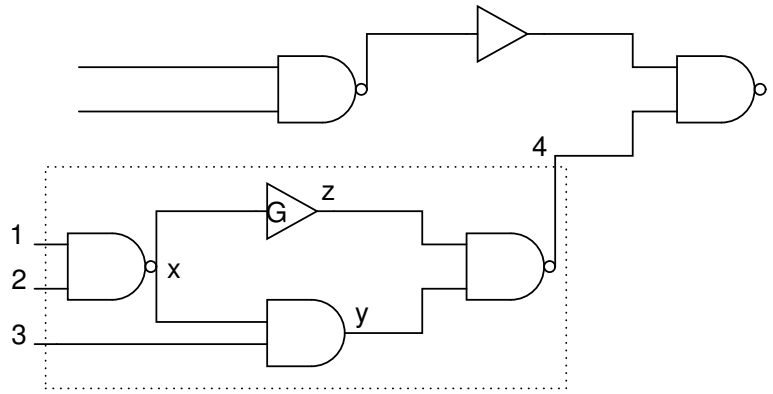


Figure 4.3. Block-level timing budgeting.

As shown in Fig. 4.4, two consecutive gates have delays $1 + \hat{x}$ and $1 - \hat{x}$, respectively, where $\hat{x}$ has a standard Gaussian distribution. The gate-level timing budgeting has the

following two constraints:

$$t_j - t_i \geq 1 + \hat{x} + b_{ij},$$

and

$$t_k - t_j \geq 1 - \hat{x} + b_{jk}.$$

Thus the minimal delay from i to k is 8 ($2(1+3\sigma) = 8$) through robust transformation. But the block-level budgeting combines these two gates together and thus has only one constraint:

$$t_k - t_i \geq 2 + b_{ij} + b_{jk}.$$

Thus the minimal delay from i to k is only 2. This example is just a demonstration: the sensitivities may have different signs for consecutive gates. When the sensitivities have the same signs, it can also be proved that combining gates together can reduce the timing pessimism.

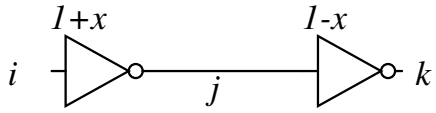


Figure 4.4. Block-level timing budgeting can reduce timing pessimism.

Let $p(u \rightsquigarrow v)$ represent a path p from the input u to the output v of a block. In summary, the block-level timing budgeting problem is formulated as

$$\text{P5: Maximize} \sum_{(i,j)\in E} \hat{c}_{ij} b_{ij}$$

s.t. $\forall (u \rightsquigarrow v) \in \{\text{paths in blocks}\}$:

$$t_v - t_u \geq \sum_{(i,j)\in(u\rightsquigarrow v)} (d_{ij}^{(0)} + b_{ij}) +$$

$$\sum_{1\leq k\leq n}\left\{\sum_{(i,j)\in(u\rightsquigarrow v)}\left[\alpha_{ij}^{(k)}(d_{ij}^{(0)}+b_{ij})\right]\right\}\hat{\Delta}_k$$

$$0\leq b_{ij}\leq\delta$$

Through the previously introduced robust optimization technique, P5 is converted to a linear programming problem:

P6: Maximize Y (4.13)

$$\text{s.t. } Y\leq c_{ij}^{(0)}b_{ij}-\Gamma_0 z_0-\sum_{1\leq k\leq n}q_0^{(k)} \tag{4.14}$$

$$t_v-t_u\geq\sum_{(i,j)\in p(u\rightsquigarrow v)}(d_{ij}^{(0)}+b_{ij})+\Gamma_p z_p+\sum_{1\leq k\leq n}q_p^{(k)} \tag{4.15}$$

$$z_0+q_0^{(k)}\geq\left|\sum_{(i,j)\in E}c_{ij}^{(k)}b_{ij}\right| \tag{4.16}$$

$$z_p+q_p^{(k)}\geq\left|\sum_{i,j\in p(u\rightsquigarrow v)}\alpha_{ij}^{(k)}(d_{ij}^{(0)}+b_{ij})\right| \tag{4.17}$$

$$z_0\geq 0 \tag{4.18}$$

$$q_0^{(k)}\geq 0 \tag{4.19}$$

$$z_p\geq 0 \tag{4.20}$$

$$q_p^{(k)}\geq 0 \tag{4.21}$$

$$0\leq b_{ij}\leq\delta \tag{4.22}$$

Now we are considering two special cases.

4.3.2. Deterministic budgets

When $\hat{g}_{ij}(b_{ij}) = b_{ij}$, the timing constraints in the timing budgeting problem can be converted to

$$t_v - t_u \geq \sum_{(i,j)\in p(u\rightsquigarrow v)} (d_{ij}^{(0)} + b_{ij}) + z_p\Gamma_p + \sum_{1\leq k\leq n} q_p^{(k)}$$

$$z_p + q_p^{(k)} \geq \left| \sum_{(i,j)\in p(u\rightsquigarrow v)} d_{ij}^{(k)} \right|$$

$$z_p \geq 0$$

$$q_p^{(k)} \geq 0$$

$$0 \leq b_{ij} \leq \delta$$

Its KKT optimality conditions include the following conditions:

$$\sum_{\text{path } p \text{ from } u} x^{(p)} = \sum_{\text{path } q \text{ to } u} x^{(q)} \qquad \forall \text{ I/O } u \text{ of blocks} \tag{4.23}$$

$$x^{(p)} \geq 0 \qquad \forall \text{ path } p \tag{4.24}$$

$$\sum_k \beta_p^{(k)} \leq x^{(p)}\Gamma_p \qquad \forall \text{ path } p \tag{4.25}$$

$$0 \leq \beta_p^{(k)} \leq x^{(p)} \qquad \forall \text{ path } p \tag{4.26}$$

The $x^{(p)}$ behaves like a network flow. Actually, it can be proved that the dual of the gate-level timing budgeting problem is a convex-cost flow problem.

The objective function of the dual problem has a term:

$$-\sum_{p}\left[\sum_{k}\beta_{p}^{(k)}\left|\sum_{(i,j)\in p}d_{ij}^{(k)}\right|\right].$$

Without loss of generality, we assume that

$$\forall 1\leq k\leq n-1:\left|\sum_{(i,j)\in p}d_{ij}^{(k)}\right|\geq\left|\sum_{(i,j)\in p}d_{ij}^{(k+1)}\right|, \tag{4.27}$$

then for the optimal $\beta_{p}^{(k)}$,

$$\beta_{p}^{(k)}=\begin{cases}x^{(p)} & \text{if } k\leq\Gamma_{p}\\ (\Gamma_{p}-\lfloor\Gamma_{p}\rfloor)x^{(p)} & \text{if } \Gamma_{p}\neq\lfloor\Gamma_{p}\rfloor \text{ and } k=\lceil\Gamma_{p}\rceil\\ 0 & otherwise\end{cases}$$

According to complementary slackness conditions, if $x^{(p)}\neq 0$, we know

$$z_{p}+q_{p}^{(k)}=\left|\sum_{(i,j)\in p}d_{ij}^{(k)}\right| \qquad \text{if } k\leq\lceil\Gamma_{p}\rceil \tag{4.28}$$

$$q_{p}^{(k)}=0 \qquad \text{if } k>\lceil\Gamma_{p}\rceil \tag{4.29}$$

$$q_{p}^{(k)}=0 \qquad \text{if } \Gamma_{p}\neq\lfloor\Gamma_{p}\rfloor, k=\lceil\Gamma_{p}\rceil \tag{4.30}$$

Thus,

$$z_{p}\Gamma_{p}+\sum_{k}q_{p}^{(k)}=\left[\sum_{k\leq\lfloor\Gamma_{p}\rfloor}\left|\sum_{(i,j)\in p}d_{ij}^{(k)}\right|\right]+z_{p}(\Gamma_{p}-\lfloor\Gamma_{p}\rfloor)$$

In order to make b_{ij} have more space to change, we need to reduce $z_{p}\Gamma_{p}+\sum_{k}q_{p}^{(k)}$ as much as possible, so we need to find a minimal z_{p}. If $\lfloor\Gamma_{p}\rfloor=n$, $z_{p}=0$. Otherwise, if $\Gamma_{p}\neq\lfloor\Gamma_{p}\rfloor$,

according to Eq.(4.28) and Eq.(4.30),

$$z_p = \left| \sum_{(i,j)\in p} d_{ij}^{(\lceil \Gamma_p \rceil)} \right|$$

If $\exists p : x^{(p)} = 0$, there might exist more than one optimal solutions that have the same objective value. It is obvious that if

$$z_p \Gamma_p + \sum_k q_p^{(k)}$$

is minimal, b_{ij} has more space to change, so the objective value is at least not worse than the optimal value. So similarly, we have

$$z_p = \begin{cases} \left| \sum_{(i,j)\in p} d_{ij}^{(\lceil \Gamma_p \rceil)} \right| & \text{if } \lfloor \Gamma_p \rfloor < n \\ 0 & \text{otherwise} \end{cases} \tag{4.31}$$

$$q_p^{(k)} = \begin{cases} \left| \sum_{(i,j)\in p} d_{ij}^{(k)} \right| - z_p & \text{if } k \leq \lceil \Gamma_{ij} \rceil \\ 0 & \text{if } k > \lceil \Gamma_{ij} \rceil \end{cases} \tag{4.32}$$

Based on these results, we can simplify the problem by eliminating those z_p and $q_p^{(k)}$.

If the budgets are statistical (i.e., $\hat{g}_{ij}(b_{ij}) \neq b_{ij}$), the right hand side of Eq.(4.17) depends on b_{ij}'s and $\alpha_{ij}^{(k)}$, and cannot be sorted as in Eq.(4.27), so the simplification cannot be used. We still need to directly solve the linear programming P6.

4.3.3. Known distribution

For a late-stage design, the distributions are already known. For example, the delays are often modeled to have Gaussian distributions.

If budgets are deterministic, and the delays have Gaussian distributions, our task is to find the minimal Γ_p such that

$$\sum_i z_p \Gamma_p + \sum_k q_p^{(k)} \geq \mu \left(\sum_{(i,j)\in p} \hat{d}_{ij} \right) + \Phi^{-1}(\eta)\sigma \left(\sum_{(i,j)\in p} \hat{d}_{ij} \right).$$

We have known z_p and $q_p^{(k)}$ from Eq.(4.31) and Eq.(4.32), so Γ_p can be easily computed.

If budgets are not deterministic, and the delays have Gaussian distributions, the standard deviation of the delay is

$$\sqrt{\sum_k \left(\sum_{(i,j)\in p} \alpha_{ij}^{(k)} (d_{ij}^{(0)} + b_{ij}) \right)^2},$$

which involves the non-linear operation on variables b_{ij}'s. How to determine Γ_p becomes difficult. We use the original bound Eq.(4.12) that ignores the distribution information to determine Γ_p.

For the objective function, the sensitivities are often modeled to have Lognormal distributions. In reality, the budgeting computation step and the budgeting implementation step are not coupled very well because of the discreteness of gate delays. For example, the budgeting computation step assigns a budget $3ps$ to a gate G with delay d_G, but we cannot find a gate in the library with exactly $d_G + 3ps$ delay. Suppose we find a gate with delay $d_G + 2ps$, then actually 1ps budget is not used, so there exist a discrepancy between the actual cost after the implementation step and the expected cost computed in the budgeting step. So the accurate model of the objective function may not benefit the final result greatly. Thus, we always set $\Gamma_0 = n$ in P6.

4.4. Experimental results

Our approach is implemented in C++ in our timer called ChiruTimer. We use MOSEK linear programming solver [**70**] as a sub-routine to solve the linear programming problem. The number of parameters is 10. We assume that parameters have at most 30% deviation from their means. The means and the bounds of the distributions of costs and delays are known.

Each block in the block-level budgeting has a gate and all its immediate fan-out gates. The yield constraint on timing is 100%. We tested our approach on ISCAS85 benchmarks that are synthesized to a UMC 90nm technology library. All the experiments were run on a Linux Redhat PC with 2.4GHz CPU and 2.0GB memory.

4.4.1. Timing yield

Table 4.1. Comparison between statistical approaches that use deterministic budgets and statistical budgets on worst case delay

circuits			DET-BUDGET	ChiruTimer
name	#gates	D_{constr} (ns)	delay (ns)	delay (ns)
c1355	384	5.00	5.23	5.00
c1908	430	4.50	4.57	4.49
c2670	563	5.00	5.12	4.99
c3540	1075	8.00	8.18	8.00
c5315	1453	6.00	6.17	6.00
c6288	2913	17.00	17.69	17.00
c7552	1624	6.00	6.19	6.00

First, we test the effectiveness of statistical budgets (i.e., $\hat{g}_{ij}(b_{ij}) \neq b_{ij}$) on improving timing yield. Table 4.1 shows the comparison results between two statistical gate sizing approaches: the DET-BUDGET approach uses deterministic budgets (i.e., $\hat{g}_{ij}(b_{ij}) = b_{ij}$), and ChiruTimer uses statistical budgets. The "delay (ns)" columns show the worst maximal

delay from primary inputs to primary outputs. The results indicate that all the solutions from DET-BUDGET violate the timing constraints. For example, the solution for c6288 has a timing violation of 690 ps. ChiruTimer does not have any timing violations.

4.4.2. Cost improvement

Second, we test our approach on the cost improvement. Table 4.2 shows the comparison results between worst case deterministic approach and ChiruTimer. The worst-case deterministic approach uses worst-case delays and sensitivities. Its formulation of budgeting step is

$$\text{Maximize } \sum_{(i,j)\in E}(c_{ij}^{(0)} - \sum_k |c_{ij}^{(k)}|)b_{ij} \tag{4.33}$$

$$\text{s.t. } t_j - t_i \geq d_{ij}^{(0)} + \sum_k |d_{ij}^{(k)}| + (1 + \sum_k |\alpha_{ij}^{(k)}|)b_{ij} \tag{4.34}$$

$$0 \leq b_{ij} \leq \delta \tag{4.35}$$

It guarantees that this worst-case deterministic approach gets solutions without timing violation since the coefficients of b_{ij} in the constraints Eq.(4.34) are their worst values. The results indicate that ChiruTimer achieves 17.50% savings on the worst leakage power on average. These savings come from two parts: the first part is that the cost sensitivities are statistical, so the correlations between cost sensitivities are considered; the second part is the block-level timing budgeting, so the correlations between delays are partially considered. The blocks in ChiruTimer are small: a gate and its immediate fan-out gates form a block. With bigger blocks, more cost savings can be expected. For the running time, ChiruTimer finished most of these cases in several minutes.

Table 4.2. Comparison between worst case deterministic approach and ChiruTimer on worst case cost

circuits		Worst Deterministic		ChiruTimer		
name	#gates	leakage (uw)	time (s)	leakage (uw)	time (s)	savings (%)
c1355	384	138.91	14.10	128.17	57.85	7.73
c1908	430	131.66	16.90	114.36	57.61	13.14
c2670	563	90.86	26.88	69.27	71.78	23.76
c3540	1075	188.44	60.08	148.94	296.54	20.96
c5315	1453	220.73	88.57	164.22	326.19	25.60
c6288	2913	684.46	163.20	592.04	1512.54	13.50
c7552	1624	311.38	70.95	256.02	429.51	17.78
Avg						17.50

4.5. Conclusions

We propose a novel formulation of the timing budgeting problem to consider the changes of both the means and the variances of delays, and transform the problem to a linear programming problem using a robust optimization technique.

Our approach can be used not only in late-stage design where the detailed distribution information is known, but also in early-stage design since our approach does not assume any specific underlying distributions. In addition, we propose a block-level timing budgeting to capture the correlations between devices and reduce timing pessimism. The experimental results demonstrate that our approach not only increases the timing yield, but also improves the leakage power by 17.50% on average.

CHAPTER 5

A Flexible Data Structure for Maxplus Merge Operations in Dynamic Programming

Dynamic programming is an effective technique to handle buffering [**101**], technology mapping [**57**] and slicing floorplan [**93**] problems. For example, van Ginneken [**101**] proposed a dynamic programming method to complete buffer insertion in distributed RC-tree networks for minimal Elmore delay, and his method runs in $O(n^2)$ time and space, where n is the number of legal buffer positions. Many works based on his work [**6**, **54**, **61**, **62**, **72**, **88**, **117**] emerged. So the speed-up of van Ginneken's algorithm can benefit many other algorithms. An essential operation in van Ginneken's algorithm is to merge two candidate lists to be one list where inferior candidates have been pruned. Shi [**87**] proposed an efficient algorithm that improves Stockmeyer's algorithm [**93**] for the similar merge operations in slicing floorplan. Based on that, Shi and Li [**88**] presented an $O(n \log^2 n)$ algorithm for the optimal buffer insertion problem. In these two researches, a balanced binary search tree is used to represent a list of solution candidates, and it avoids updating every candidate during the merge of two candidate lists. However, as shown in [**87**], the merge of two candidate lists based on balanced binary search trees can only speed up the merge of two candidate lists of much different lengths (*unbalanced situation*), but not the merge of two candidate lists of similar lengths (*balanced situation*).

Fig. 5.1 illustrates the best data structure for maintaining solutions in each of the two extreme cases: the balanced situation requires a linked list that can be viewed as a totally

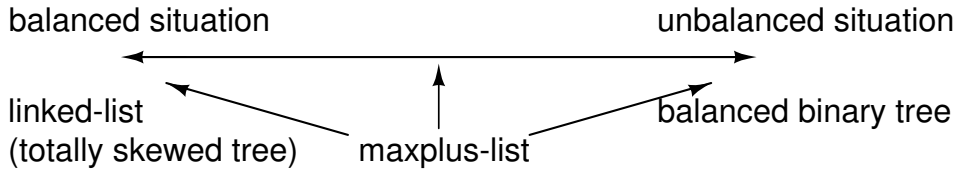


Figure 5.1. The flexibility of maxplus-list.

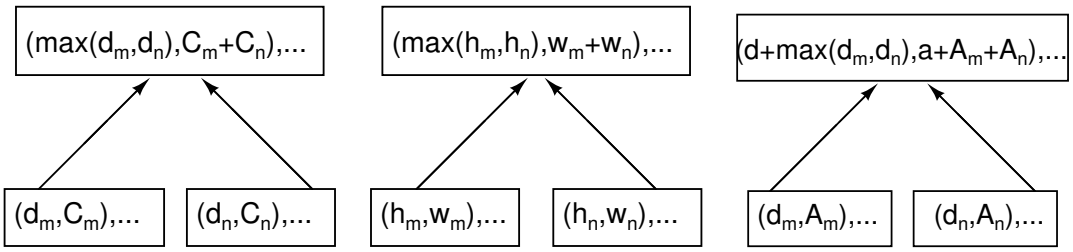
skewed tree; the unbalanced situation requires a balanced binary tree. However, most cases in reality are between these extremes, where neither data structure is the best. As we can see, the most balanced situation requires the most skewed data structure while the most unbalanced situation requires the most balanced data structure. Therefore, we need a data structure that is between a linked list and a balanced binary tree for the cases in the middle. We discovered that a skip-list [**77**] is such a data structure as it migrates smoothly between a linked list and a balanced tree. In this chapter, we propose a flexible data structure called *maxplus-list* based on the skip-list and corresponding algorithms for operations in the dynamic programming. As shown in Fig. 5.1, we can migrate the maxplus-lists to suitable positions based on how balanced the routing tree is: a maxplus-list becomes a linked-list in balanced situations; it behaves like a balanced binary tree in unbalanced situations. So the performance of our algorithm is always very good. The maxplus-merge algorithm based on maxplus-list has the same time complexity with the merge algorithm used in [**87**, **88**]. The experimental results show that it is even faster than the algorithm based on balanced binary search tree in unbalanced situations, and especially, it is much faster in balanced situations. Besides, maxplus-list data structure is much easier to understand and implement than balanced binary search tree.

The rest of this chapter is organized as follows. In Section 5.1, the general problem of merging two candidate lists is formulated, and the skip-list data structure is introduced. In

Section 5.2, maxplus-list data structure and an efficient algorithm to merge two maxplus-lists are shown. In Section 5.3, the approach for finding the optimal solutions after the bottom-up merge operations are done is shown. The experimental results are reported in Section 5.4, and the chapter is concluded in Section 5.5.

5.1. Preliminary

5.1.1. Maxplus problem



(a) buffer insertion for maximal source required time

(b) cell orientation in minimal area vertical slicing floorplan

(c) time-driven technology mapping

Figure 5.2. The similar merge operations in three different problems.

Given a routing tree as a distributed RC network, a dynamic programming approach to buffer insertion will build non-inferior solutions bottom-up from the sinks to the root. The objective is to insert buffers such that the maximal delay D_{source} from the root to sinks is minimized. Each solution (d_v, C_v) at a node v represents a buffering of the subtree at v having d_v as the maximal delay to the sinks and C_v as the loading capacitance. When a tree at u is composed of a wire (u, v) and a subtree at v, its solution (d_u, C_u) can be computed as follows.

$$d_u = d_v + r(u,v)(C_v + c(u,v)/2)$$

$$C_u = C_v + c(u, v)$$

where $r(u, v)$ and $c(u, v)$ are the resistance and capacitance of wire (u, v), respectively. When a buffer is inserted at the node v, the new solution (d'_v, C'_v) can be computed similarly:

$$d'_v = d_v + d_b + r_b C_v$$

$$C'_v = c_b$$

where d_b, r_b and c_b are the internal delay, output resistance and input capacitance of the buffer, respectively. The most interesting case happens when two branches are combined at a node. As shown in Fig. 5.2(a), assuming that (d_m, C_m) is a solution in one branch and (d_n, C_n) in the other, the combined solution is given as follows.

$$d = \max(d_m, d_n)$$

$$C = C_m + C_n$$

The optimal structure of dynamic programming requires that there is no solutions (d_1, C_1) and (d_2, C_2) such that $d_1 \leq d_2$ and $C_1 \leq C_2$ for the same subtree.

It is very interesting to note that such an operation also appears in many other optimization problems. For example, the area minimization problem in slicing floorplan. Given a slicing tree representing a floorplan, the problem of area minimization is to select the size of each module such that the chip area is minimized [**93**]. The dynamic programming approach [**93**] builds the solutions bottom up. Each solution (h_v, w_v) at a node v represents a floorplan at v having h_v as the height and w_v as the width. As shown in Fig. 5.2(b), given

the solutions $(h_m, w_m), (h_n, w_n)$ of the two subtrees and a parent node with vertical cut, a candidate solution at the parent node can be constructed as $(\max(h_m, h_n), w_m + w_n)$.

Given a generic gate-level netlist, the technology mapping needs to map the circuit into a netlist composed of cells from a library. A popular heuristic [**57**] is to decompose the circuit into trees and apply a dynamic programming on each tree. Each solution (d_v, A_v) at a node v represents a technology mapping at v having d_v as the maximal delay to the sinks and A_v as the area. When the objective is to minimize the area under a delay constraint [**17**], the generation of solutions at a node from those of its subtrees is shown in Fig. 5.2(c), where (d_m, A_m) and (d_n, A_n) are the solutions at the fan-ins of a possible mapping, and d and a are the delay and area of the mapped cell at node v respectively. The mapping can be decomposed into two steps: first compute $(\max(d_m, d_n), A_m + A_n)$ and then add d and a to its elements.

The common operation in the above dynamic programming approaches can be defined as follows.

Problem 5.1 (Maxplus problem)**.** *Given two ordered lists* $A = \{(A_1.m, A_1.p), \ldots, (A_a.m, A_a.p)\}$ *and* $B = \{(B_1.m, B_1.p), \ldots, (B_b.m, B_b.p)\}$*, that is,* $A_i.m > A_j.m \wedge A_i.p < A_j.p$ *and* $B_i.m > B_j.m \wedge B_i.p < B_j.p$ *for any* $i < j$*, compute another ordered list* $C = \{(C_1.m, C_1.p), \ldots, (C_c.m, C_c.p)\}$ *such that it is a* maxplus merge *of* A *and* B*, that is, for any* $0 < k \leq c$ *there are* $0 < i \leq a$ *and* $0 < j \leq b$ *such that*

$$C_k.m = \max(A_i.m, B_j.m)$$

$$C_k.p = A_i.p + B_j.p$$

and for any $0 < i \leq a$ *and* $0 < j \leq b$ *there is* $0 < k \leq c$ *such that*

$$\max(A_i.m, B_j.m) \geq C_k.m,$$

and

$$A_i.p + B_j.p \geq C_k.p.$$

5.1.2. Stockmeyer's algorithm

A straight-forward approach to the maxplus problem is to first compute $(\max(A_i.m, B_j.m), A_i.p + B_j.p)$ for every $0 < i \leq a$ and $0 < j \leq b$ and then delete all inferior solutions. However, it takes at least $\Omega(ab)$ time. Stockmeyer [**93**] proposed a $O(a + b)$ time algorithm where inferior solutions can be directly avoided. The idea is as follows. First, since $A_1.p + B_1.p$ is the smallest, the solution $(\max(A_1.m, B_1.m), A_1.p + B_1.p)$ must be assigned to C_1. Then if $A_1.m = C_1.m$, any solution $(\max(A_1.m, B_i.m), A_1.p + B_i.p) = (C_1.m, A_1.p + B_i.p)$ for any $1 < i \leq b$ is inferior to C_1, thus should not be generated. Since all combinations with A_1 have been considered (even though not generated), we can proceed with A_2. This process can be iterated and the pseudo-code is given in Fig. 5.3, where ϕ represents an empty list.

5.1.3. Skip-list

The advantage of a balanced binary search tree over a linked list is its capability to quickly find an item ranked around the middle. Skip list [**77**] is an alternative data structure to a balanced binary search tree. It can be viewed as a combination of multiple linked lists, each on a different level. The list on the lowest level includes all the items while that on a higher level has fewer items. An example skip-list is illustrated in Fig. 5.4. An item on k

Algorithm STOCKMEYER(A, B, C)

```
C ← φ
while A ≠ φ ∧ B ≠ φ
{
    if A_1.m ≥ B_1.m
    {
        P ← A_1
        Q ← B_1
    }
    else
    {
        P ← B_1
        Q ← A_1
    }
    P.p ← P.p + Q.p
    Append P to C
    Delete P from its original list
    if P.m = Q.m
        delete Q
}
return C
```

Figure 5.3. Stockmeyer's Algorithm.

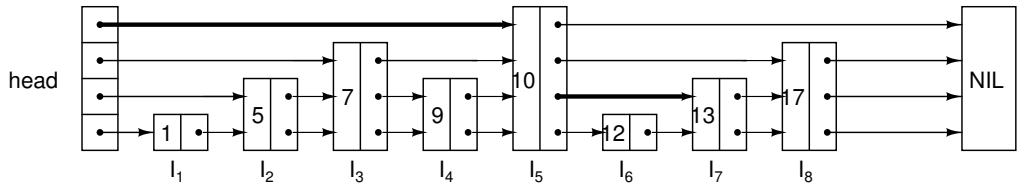


Figure 5.4. Skip-list.

linked lists, that is, with k forward pointers, is called a *level* k item. As we can see, if the level-k items are evenly distributed among the level-$(k-1)$ items, a skip-list can achieve the function of a balanced binary search tree, that is, finding any item in $O(\log n)$ time.

It is impractical to modify item levels during operations to maintain the balance among levels. An effective way is to randomly choose an item level during insertion and keep it fixed thereafter. Therefore, a skip-list has two parameters: the maximal permitted level

MaxLevel and the probability P_r that an item with level k $forward$ pointers also has level $(k+1)$ $forward$ pointers. Different values of *MaxLevel* and P_r may lead to different costs for operations. In [**77**], it is suggested that $P_r = 0.25$ and $MaxLevel = \log_{\frac{1}{P_r}}(N)$, where N is the upper bound on the number of items in the skip-list. Each skip-list has a *head* that has forward pointers at levels one through *MaxLevel*. The expected running time of search, insertion and deletion of one item in a skip-list with n items is $O(\log n)$ [**77**].

5.2. Maxplus-list

Even though Stockmeyer's algorithm takes linear time to combine two lists, when the merge tree is skewed, it may take n^2 time to combine all the lists even though the total number of items is n. For example, a list A of size $n/2$ and a list B of size one may be combined in one step, which may take $O(n/2)$ time in Stockmeyer's algorithm. For a skew tree with n items, the total number of merge operations for two lists is $n/2$, so the total running time is $O(n^2)$. However, if we can quickly find $0 < i \leq n$ such that $A_i.m \geq B_1.m$ and $A_{i+1}.m < B_1.m$, the new list will have the first i items in A with their p properties incremented by $B_1.p$. This is the idea explored by Shi [**87**]. He proposed to use a balanced binary search tree to represent each list so that the search can be done in $O(\log n)$ time. To avoid updating the p properties individually, the update was annotated on an item for the subtree rooted at it. Shi's algorithm is faster when the merge tree is skewed since it takes $O(n \log n)$ time comparing with Stockmeyer's $O(n^2)$ time. However, Shi's algorithm is complicated and also much slower than Stockmeyer's when the merge tree is balanced.

Instead of a balanced binary tree, we proposed the maxplus-list based on the skip-list for keeping candidate solutions. Since a maxplus list is close to a linked list, its merge operation is just a simple extension of Stockmeyer's algorithm. As shown in Fig. 5.3, during each

iteration of Stockmeyer's algorithm, the current item with the maximal m property in one list is finished, and the new item is equal to the finished item with its p property incremented by the p property of the other current item. The idea of the maxplus-list is to finish a sub-list of more than one items at one iteration. Assume that $A_i.m > B_j.m$, we want to find a $i \leq k \leq a$ such that $A_k.m \leq B_j.m$ but $A_{k+1}.m < B_j.m$. These items $A_i, \ldots, A_k$ are finished and put into the new list after their p properties are incremented by $B_j.p$. The speed-up over Stockmeyer's algorithm comes from the fact that this sub-list is processed (identified and updated) in a batch mode instead of item by item. The *forward* pointers in a maxplus-list are used to skip items when searching for the sub-list, and an *adjust* field is associated with each *forward* pointer to record the incremental amount on the skipped items.

5.2.1. Data structure

A maxplus-list is a skip-list with each item defined by the following C code.

```
struct maxplus_item{
    int level; /*the level*/
    float m, p; /*two properties*/
    float *adjust;
    struct maxplus_item **forward; /*forward pointers*/
}
```

The size of *adjust* array is equal to the level of this item, and $adjust[i]$ means that p properties of all the items jumped over by $forward[i]$ should add $adjust[i]$.

5.2.2. Merge operation

We define a *cut* after item I in a maxplus-list, denoted by cut_I, as an array of size *MaxLevel* with the ith item being the last item with its level larger or equal to i before item I (including I). For example, in Fig. 5.4, the *cut* after I_7 is: $cut[1] = I_7$, $cut[2] = I_7$, $cut[3] = I_5$, $cut[4] = I_5$. We can see that the items in a cut form stairs.

```
Algorithm ML-MERGE(A, B, C)
    C ← φ
    CUT ← C.head
    while A ≠ φ ∧ B ≠ φ
    {
        if A_1.m ≥ B_1.m
        {
                list ← A
                item ← B_1
        }
        else
        {
                list ← B
                item ← A_1
        }
        cut ← ML-SEARCHSUBLIST(list, item)
        sublist ← ML-EXTRACTSUBLIST(cut)
        ML-APPENDSUBLIST(sublist, CUT)
        CUT ← cut
        if cut[1].m = item.m
        {
                Clear the adjust array of item
                Delete item from the maxplus-list
        }
    }
    return C
```

Figure 5.5. Merge of two maxplus-lists.

ML-MERGE, the algorithm to solve Maxplus problem, is shown in Fig. 5.5. It is very similar to Stockmeyer's algorithm. As we have mentioned before, the basic idea is to find and update a sub-list with $A_i.m \geq B_j.m$ efficiently, or a sub-list with $B_u.m \geq A_v.m$ efficiently.

```
ML-SEARCHSUBLIST(list, item)
    cut ← list.head
    item1 ← NIL
    for i ← MaxLevel downto 1
    {
        while cut[i].forward[i].m ≥ item.m
        {
                cut[i].adjust[i] ← cut[i].adjust[i] + item.p
                cut[i].forward[i].p ← cut[i].forward[i].p + item.p
                item1 ← cut[i] ← cut[i].forward[i]
        }
        cut[i] ← item1;
    }
    return cut;
```

Figure 5.6. Procedure ML-SEARCHSUBLIST.

Suppose initially the maxplus-lists A and B are both sorted in the decreasing order of m and increasing order of p, and have no redundant items. The pseudocode of procedure ML-SEARCHSUBLIST(*list*, *item*) is shown in Fig. 5.6. It starts from the head of *list* to search for the longest sub-list R satisfying

$$\forall I \in R : I.m \geq item.m\,.$$

During this search, the p property of every visited item is increased by $item.p$, and the $adjust[i]$ of every visited item is also increased by $item.p$ if the corresponding *forward* pointer is used for jumping over. It returns a cut after the last item L in R.

Then all the items (including the head) before the next item of $cut[1]$ are moved to a new list *sublist* in procedure ML-EXTRACTSUBLIST. A new head of the original list is

generated, and for each level i, $adjust[i]$ in the new head is set to be $adjust[i]$ of $cut[i]$. Now we have successfully extracted the sublist. The number of *forward* pointers visited in ML-ExtractSublist is $O(MaxLevel)$.

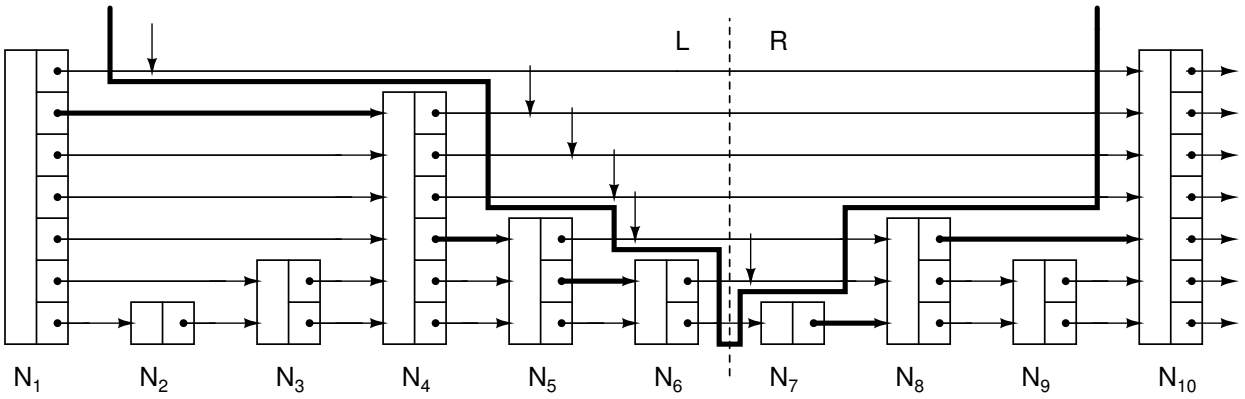


Figure 5.7. An example to show the flow of ML-AppendSublist.

```
ML-AppendSublist(sublist, CUT)
    for i ← MaxLevel downto 1
    {
        CUT[i].forward[i] ← sublist.head.forward[i]
        diff ← CUT[i].adjust[i] − sublist.head.adjust[i]
        CUT[i].adjust[i] ← sublist.head.adjust[i]
        item ← CUT[i]
        if i ≥ 2
        {
              while item ≠ CUT[i − 1]. forward[i − 1]
              {
                 item.adjust[i − 1] ← item.adjust[i − 1] + diff
                 if item.forward[i − 1] ≠ NIL
                       item.forward[i − 1].p ← item.forward[i − 1].p + diff
                 item ← item.forward[i − 1]
              }
        }
    }
```

Figure 5.8. Procedure ML-AppendSublist.

Now we can start to append the *sublist* to the maxplus-list C. The pseudocode of ML-AppendSublist is shown in Fig. 5.8. The argument CUT is the cut after the last non-null item in C. From ML-SearchSublist and ML-ExtractSublist, we can see that the *adjust* fields of the head of a maxplus-list may be not equal to zero, so the most important step in ML-AppendSublist is to update the *adjust* fields of the items in CUT.

The flow of ML-AppendSublist is shown in Fig. 5.7. Here, we want to append sublist R to L. The current CUT is $\{N_6, N_6, N_5, N_4, N_4, N_4, N_1\}$. We walk down along the stairs formed by CUT. First we set $N_1.forward[7]$ to be N_{10}. We compute the difference *diff* between $N_1.adjust[7]$ and $R.head.adjust[7]$, and set $N_1.adjust[7]$ to be $R.head.adjust[7]$. If *diff* is not zero, then the actual p properties of the items from N_2 to N_6 become not correct. So we need to propagate this difference to the next level to correct the p properties. Increase $N_1.adjust[6]$ and $N_4.adjust[6]$ by *diff*, and now the p properties of N_2, N_3, N_5 and N_6 are correct. Increase N_4 *.p* by *diff*, then the p property of N_4 is correct. Now we have successfully appended R to L at level 7. At level 6, we set $N_4.forward[6]$ to be N_{10}, compute the difference between $N_4.adjust[6]$ and $R.head.adjust[6]$, and propagate the difference to level 5. Similarly, we can append R to L from level 5 to 1 using this difference propagation technique. From the flow we can see that at the end, for each level i, $adjust[i]$ of $CUT[i]$ is equal to $adjust[i]$ of the head of R. The reason that we choose to update the *adjust* fields of CUT instead of the *adjust* fields of the items in the rising stairs of R is that the *forward* pointers visited during the update of the CUT have already been visited by ML-SearchSublist, which can help the time complexity analysis.

After we finish all the merge operations, we need to evaluate the p properties of items in the candidate solution lists at the root of the tree. To evaluate the p property of any item

I, firstly we search for item I and simultaneously find the cut after I, then

$$I.p = I.p + \sum_{I.\,level < i \leq MaxLevel} cut[i].adjust[i].$$

5.2.3. Complexity analysis

Pugh [**76**] proposed an algorithm to merge two skip-lists. For two skip-lists with sizes n_1 and n_2 respectively, without loss of generality, assume that $n_1 \leq n_2$. The merge algorithm in [**76**] runs in $O(n_1 + n_1 \log n_2/n_1)$ expected time. Pugh [**76**] also claimed that in almost all cases the skip-list algorithms are substantially simpler and at least as fast as the algorithms based on balanced trees.

The merge procedure in our algorithm is similar with the merge procedure in [**76**], but we need to update the *adjust* fields in ML-AppendSublist. The running time of the skip-list merge algorithm in [**76**] is proportional to the number of jump operations. The running time of the maxplus merge algorithm in this section is also proportional to the number of the involved jump operations. Considering this, we have the following theorem concerning the number of jump operations in ML-Merge.

Theorem 5.1. *The number of jump operations in* ML-Merge *is a constant times of the number of jump operations in the skip-list merge algorithm in [**76**].*

Proof. Let $Pt(proc)$ denote the number of jump operations in an execution of the procedure *proc*. As mentioned before,

$$Pt(\text{ML-ExtractSublist}) \leq 2 * MaxLevel,$$

and

$$Pt(\text{ML-AppendSublist}) \leq Pt(\text{ML-SearchSublist}).$$

So

$$Pt(\text{ML-Merge}) = 2(MaxLevel + Pt(\text{ML-SearchSublist})).$$

While the number of jump operations in the corresponding iteration of the skip-list merge algorithm proposed by Pugh [**76**] is at most

$$MaxLevel + Pt(\text{ML-SearchSublist}).$$

So the number of jump operations in ML-Merge is no more than constant times of the number of jump operations in the skip-list merge algorithm in [**76**]. □

Further, since the skip-list merge algorithm proposed by Pugh takes $O(n_1 + n_1 \log n_2/n_1)$ expected time [**76**], we have the following corollary.

Corollary 5.1.1. *For two maxplus-lists with sizes n_1 and n_2 respectively, assume that $n_1 \leq n_2$, then the expected time complexity of* ML-Merge *is $O(n_1 + n_1 \log n_2/n_1)$.*

So the expected time complexity of the merge algorithm based on maxplus-list is the same with the time complexity of the merge algorithm based on balanced binary tree.

5.2.4. Determination of MaxLevel

Our experiments show that different values of *MaxLevel* may lead to much different running time. For example, when *MaxLevel* is equal to 1, the algorithm runs fastest in balanced situations, while it becomes much worse in unbalanced situations. As shown in Fig. 5.1, the method based on linked-list is much faster in balanced situations, while the method based on

binary search trees is faster in unbalanced situations. An important property of maxplus-list is that it is a *flexible* data structure, that is, when $MaxLevel = 1$, it becomes a linked-list, while when *MaxLevel* increases, it behaves like a binary search tree. In order to get the best speedup, we use different values of *MaxLevel* in different situations. Here we presented a simple strategy to determine the value of *MaxLevel*. Based on the results of the statistical experiments in [**77**], the value of P_r is always fixed at 0.25.

In the problems of buffering, floorplan, or technology mapping, the input is always a tree. We define *basic element*s as the buffer positions in buffer insertion, realizations of basic blocks in floorplan, and mappings in technology mapping. During the read of input files, we record the maximal and minimal depths of leaves in the tree: D_{max} and D_{min}. Then

$$MaxLevel = \begin{cases} 1 & \text{if } D_{min} \geq D_{max}/2 \\ \lfloor \log_{\frac{1}{P_r}}(n/8) \rfloor & \text{otherwise} \end{cases}$$

where n is the number of basic elements.

5.3. Solution extraction

We use the buffering problem as an example to show how to extract the best solution after the bottom-up dynamic programming procedure is finished.

In order to record the composition of each item, we modify our data structure to include a pointer array *comp* of size *level* in each item. During the bottom-up calculation of non-redundant candidates, we maintain a *configuration graph* to record the composition of each item. The pointers in *comp* array point to the vertices representing compositions in a configuration graph. Similar to the *adjust* array, *comp*[i] means that all the items between this

item and the next item with level larger or equal to i are composed partly by the composition that it points to.

Now we can update the *comp* array similar to the *adjust* array in the merge operations, and at the same time build the configuration graph. For example, as shown in Fig. 5.9, if the items from item m to item n in a maxplus-list are merged with an item k, then we create a new vertex v in the configuration graph, and add edges from v to the vertex that the original *comp*[3] of item m points to and all the vertices pointed by *comp* array of item k, then let *comp*[3] of item m point to v.

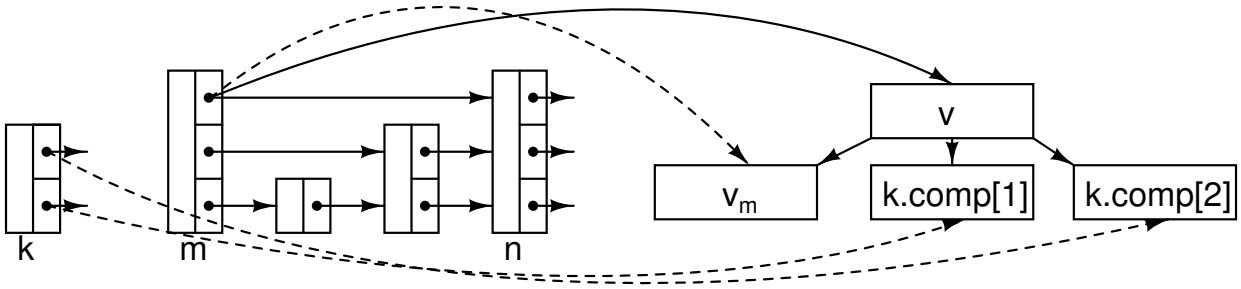


Figure 5.9. Configuration graph construction. (Dashed lines represent the original pointers, solid lines represent the pointers after merge.)

When we want to attach a buffer at the current location, we create a new vertex v in the configuration graph, and add edges from v to the vertex representing the buffer location and the vertex pointed by *comp*[1], then make *comp*[1] point to v.

We can see that the configuration graph is a directed acyclic graph that is similar to the well-known Binary Decision Diagram (BDD) [**12**]. Traversing from any vertex v in the configuration graph, the buffers visited represent part of a buffering solution.

After the bottom-up calculation of non-redundant candidates, we firstly evaluate the m and p properties of each item and select an optimal item. At the same time, we merge all the pointers in *comp* array for each item. So after evaluation, each item has a single pointer

pointing to its composition. We traverse the configuration graph starting from the vertex that the composition pointer of the optimal item points to, then we get all the inserted buffers in this optimal solution. During our merge algorithm, we only keep the maxplus-lists after merge, and the total space cost is the same with that in [**88**].

5.4. Experimental results

Table 5.1. Comparison results for unbalanced trees

Name	# Leaves	Stockmeyer's Alg. time (T_1 s)	Shi's Alg. time (T_2 s)	Our Alg. time (T_3 s)	T_1/T_3	T_2/T_3
U100	100	0.083	0.033	0.033	2.52	1.00
U200	200	0.233	0.083	0.067	3.48	1.24
U300	300	0.567	0.150	0.100	5.67	1.50
U400	400	1.033	0.200	0.133	7.77	1.50
U500	500	1.530	0.250	0.180	8.50	1.39
U600	600	2.080	0.333	0.217	9.59	1.53
U700	700	2.900	0.485	0.267	10.86	1.82
U800	800	3.533	0.433	0.317	11.15	1.37
U900	900	4.500	0.500	0.333	13.51	1.50
U1000	1,000	5.633	0.767	0.383	14.71	2.00

Table 5.2. Comparison results for balanced trees

Name	# Leaves	Stockmeyer's Alg. time(T_1 s)	Shi's Alg. time(T_2 s)	Our Alg. (1,0) time (s)	(4, 0.25) time(T_3 s)	T_1/T_3	T_2/T_3
B128	128	0.033	0.083	0.033	0.033	1.00	2.52
B256	256	0.100	0.317	0.133	0.100	1.00	3.17
B512	512	0.167	0.883	0.183	0.217	0.77	4.07
B1024	1,024	0.350	1.433	0.267	0.466	0.75	3.08
B2048	2,048	0.717	2.950	0.583	0.983	0.73	3.01
BLARGE	32,768	12.730	50.390	10.700	18.550	0.69	2.72

Table 5.3. Comparison results for mixed trees

Name	# Leaves	Stockmeyer's Alg. time(T_1 s)	Shi's Alg. time(T_2 s)	Our Alg. *MaxLevel*	time(T_3 s)	T_1/T_3	T_2/T_3
Mdata1	82	0.017	0.067	2	0.033	0.52	2.03
Mdata2	296	0.483	0.300	3	0.117	4.13	2.56
Mdata3	1236	0.633	1.750	4	0.616	1.03	2.84
Mdata4	2196	11.633	2.767	5	1.333	8.73	2.08
Mdata5	8046	3.317	11.383	5	4.550	0.73	2.50
Mdata6	21892	9.200	37.233	1	9.533	0.97	3.91

We are focusing on testing the performance of our maxplus-list based merge operation. We designed many test cases with each case corresponding to a tree, and every leaf in a tree has four basic options. We implement a bottom-up algorithm in C based on our merge algorithm to calculate the non-redundant candidate list at the root of each tree. We implement Stockmeyer's algorithm, and download the code of Shi's merge algorithm from Shi's web-page [**86**] for comparison. The running time is the total time for executing each algorithm 100 times, and does not include the time for reading input files and printing final results. All the experiments were run on a Linux PC with 2.4 GHz Xeon CPU and 2.0 GB memory.

For unbalanced trees, the comparison of our algorithm, Stockmeyer's algorithm and Shi's algorithm is shown in Table 5.1. Column 3 and Column 4 are the running time of Stockmeyer's algorithm and Shi's algorithm respectively. We use $MaxLevel = 4$, and $P_r = 0.25$ in the maxplus-list. The results indicate that our algorithm is much faster than Stockmeyer's algorithm, and with the increasing sizes of cases, it gets more and more speed-up. Most importantly, our algorithm is about 1.5 times faster than Shi's algorithm on average.

For balanced trees, the comparison of our algorithm, Stockmeyer's algorithm and Shi's algorithm is shown in Table 5.2. The 5th column is the running time of our method with $MaxLevel = 1$, $P_r = 0$, and the 6th column is the running time of our method with $MaxLevel = 4$, $P_r = 0.25$. The results shows that our algorithm with $MaxLevel = 1$ is even faster than Stockmeyer's algorithm in some cases. This is because when $MaxLevel = 1$, the skip-list becomes an ordinary linked-list, but our method moves a series of items in each iteration while Stockmeyer's algorithm moves item by item. When $MaxLevel = 4$, our algorithm is slower than Stockmeyer's algorithm but more than 2.5 times faster than Shi's algorithm.

For mixed trees that contain both balanced subtrees and unbalanced subtrees, we use our strategy mentioned before to determine the value of $MaxLevel$. The comparison of our algorithm, Stockmeyer's algorithm and Shi's algorithm is shown in Table 5.3. We can see that our algorithm is always more than 2 times faster than Shi's algorithm. Especially for Mdata6, our strategy for determining *MaxLevel* improved the efficiency greatly.

5.5. Conclusion

The common merge operations of solution lists in the dynamic programming technique for the buffering, technology mapping and slicing floorplan can be summarized as maxplus-merge operations. In this section, we presented a flexible data structure, the maxplus-list, to represent a solution list. With parameters to adjust automatically, our maxplus merge algorithm based on maxplus-list works better than Shi [**87**] under all cases: unbalanced, balanced, and mix sizes. Our data structure is also simpler to implement.

CHAPTER 6

Efficient Algorithms for Buffer Insertion in General Circuits Based on Network Flow

Buffer insertion is widely used to reduce interconnect delays [**6**, **7**, **62**, **89**, **101**, **118**]. Van Ginneken [**101**] proposed a dynamic programming method to buffering in distributed RC-tree networks for minimal Elmore delay. Lillis *et al.* [**62**] extended van Ginneken's algorithm to minimize the power consumption with more general delay models. However, most of the existing approaches considered the buffer insertion problem only on a single net. Recent projections of historical scaling trends by Saxena *et al.* [**83**] predict synthesis blocks to have 70% of their cell count dedicated to interconnect buffers within a few process generations. The power consumption of these buffers becomes prominent, so it is necessary to reduce buffers without sacrificing the performance of a circuit.

If the given timing constraint is greater than the minimal delay that can be achieved by buffering, assignment of various timing budgets on the critical paths would give multiple buffering solutions, among which we need to select the one with minimal cost. This problem also occurs on buffering the less-critical paths even when the timing constraint is tight. Different from the buffering on a single net where the required times at sinks are known, the buffering on a circuit needs to do the timing budgeting: assign required time at sinks of nets. Thus, the buffering in a circuit involves two related steps: budgeting and single net buffering. Net-based buffering approaches do not have a timing budgeting step, and consider

the buffering problem in a local view. As will be demonstrated in our experiments, a net-based buffering approach gets significantly sub-optimal solution. Liu *et al.* [**64**] presented a Lagrangian relaxation based algorithm to solve the buffer insertion problem in large networks. It was extended to consider multiple buffer types and feasible buffer locations in [**63**].

Our experiments show that the results obtained using Lagrangian relaxation based techniques are significantly sub-optimal. Sze *et al.* [**95**] proposed a path based buffering algorithm. The basic idea is to partition the whole circuit into a set of trees, treat the gates as specific buffers, and for each tree, use the Van Ginneken's algorithm to compute the solution. But its performance compared with the Lagrangian relaxation based technique [**63**] is not reported.

The input to our problem is a placed and routed netlist of modules. Our objective is to insert buffers into wires in the general combinational circuit such that the timing constraint is met and the area of buffers is minimized. We relate this problem to the network flow problem, and present an effective algorithm based on network flow theory. Experimental results show that the solutions from our algorithm have much less cost than those from [**63**, **64**] and a net-based buffering approach.

The delay change of one component affects the delays of many other components. For example, the delay d_{ij} of wire (i, j) in a multi-pin net depends on not only the buffering on (i, j), but also the buffering of the fan-outs of (i, j). Without this kind of interaction, the buffering problem can be formulated as a network flow problem. While with the interactions, the objective function is no longer separable, and thus the problem becomes much more difficult to solve.

6.1. Problem formulation

The input to our problem is a placed and routed netlist of modules with drivers and loads. Our objective is to insert buffers into wires in the general combinational circuit such that the timing constraint is met and the number of buffers is minimized.

As in [**64**], we use a directed acyclic graph (DAG) $G(V, E)$ to represent the circuit, of which the vertices correspond to the primary inputs, the primary outputs, tree junctions and module inputs/outputs. Two dummy nodes s and t are introduced: s is connected to all the primary inputs, and t is connected from all the primary outputs. The edge set E includes two disjoint sets of edges E^P and E^F, corresponding to the buffer allowable wires and the buffer forbidden edges (wires or modules), respectively.

Table 6.1. Notations

R_b	output resistance of a buffer
C_b	input capacitance of a buffer
t_b	intrinsic delay of a buffer
K_e	number of buffers on wire e
L_e	length of wire e
p_e	wire length from the input of wire e to the first buffer of wire e
q_e	wire length from the last buffer of e to the output of wire e
m_e	wire length between two adjacent buffers in wire e
$\hat{R}_e$	upstream resistance at the input of wire e
$\hat{C}_e$	downstream capacitance at the output of wire e
d_e	delay of the wire e
S_e	timing slack of the wire e
R	resistance of a unit-length wire
C	capacitance of a unit-length wire
S_{th}	timing slack threshold
δ_e	max delay change of wire e w/ one more buffer
c_e	capacity of wire e
f_e	flow through wire e

The notations used in this section are given in Table 6.1. The problem is formulated as:

$$Minimize \quad \sum_{(i,j)\in E^P} K_{ij} \tag{6.1}$$

$$\text{s.t.} \quad a_i + d_{ij} \leq a_j \quad \forall (i,j) \in E \tag{6.2}$$

$$a_t - a_s \leq REQ \tag{6.3}$$

where a_i is the arrival time at vertex i, and REQ is the timing constraint.

In the buffer insertion problem, the delay change of one component influences the delays of many other components. As shown in Figure 6.1, a net is composed by wires a, b and c. If a buffer B is inserted into wire c, the delay of c is changed, and according to Elmore delay model, the delay of a also changes, so the delay from s_0 to s_2 also changes. Actually, since the delay of the driver gate is related with the load capacitance, it may change too. This kind of relations between component delays make the buffer insertion problem very difficult.

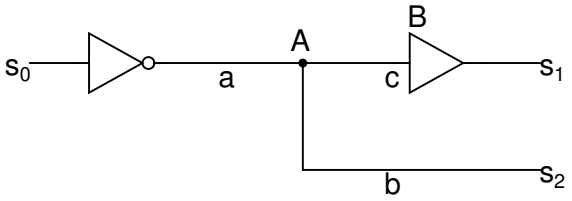


Figure 6.1. The influence of an inserted buffer.

We use the Elmore delay model for wires, modules and buffers as in [**64**]. Given a wire with buffers inserted, we can easily compute the delays. In this section, we only consider the single buffer type, then from [**64**], we know that the wire lengths between two consecutive buffers on one wire are equal. When $K_e > 0$, the delay of wire e is equal to

$$d_e \quad = \quad Rp_e(Cp_e/2 + C_b)$$

$$+(K_e - 1)(R_b(C_b + Cm_e) + t_b + Rm_e(C_b + Cm_e/2))$$

$$+t_b + R_b(\hat{C}_e + Cq_e) + Rq_e(\hat{C}_e + Cq_e/2), \tag{6.4}$$

We need to consider the contribution of the capacitance of e to the delay of the fanin edge of (e), so let

$$F_e = d_e + \hat{R}_e(Cp_e + C_b). \tag{6.5}$$

Then let $\frac{\partial F_e}{\partial p_e} = 0$, and $\frac{\partial F_e}{\partial q_e} = 0$. We can compute the optimal values of p_e and q_e for a given K_e such that F_e is minimized. The optimal values of p_e and q_e are given in [**64**]. When we get the optimal p_e and q_e for a given K_e, the delay of wire e for K_e is easily computed according to Eq. 6.4. Based on this, the maximal delay change of wire e when a new buffer is inserted into it, denoted as δ_e, can be computed. Since the wire delay is expected to decrease when a new buffer is inserted, we have $\delta_e < 0$. The *delay sensitivity* of component e is defined as $-1/\delta_e$.

In the reality, there exist some forbidden areas for buffer insertion. The buffering of the components in forbidden areas will not influence the delays of those components, which means $\delta_e = -0$ for any component e in the forbidden areas, and its delay sensitivity is infinity. Modules are forbidden areas for buffer insertion, so their delay sensitivities are infinity.

The buffer insertion problem can be viewed as a wire substitution problem. As shown in Figure 6.2, a wire has different delays for different number of inserted buffers. The objective is to select a point in the configuration of each wire such that the total number of buffers is minimized while the timing constraints are satisfied. The advantage of this idea is that during the buffer insertion, the positions of inserted buffers can change such that the required delay objective is achieved.

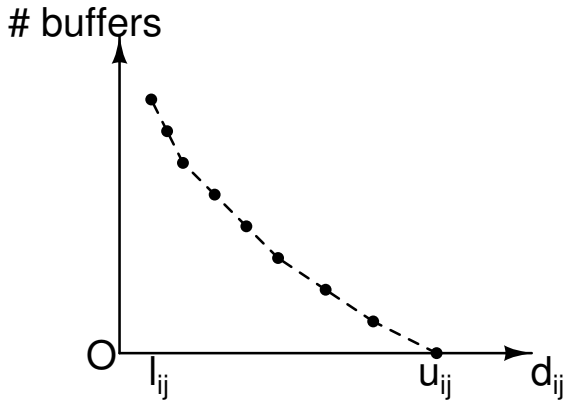


Figure 6.2. Number of buffers as a function of the wire delays.

6.2. Theory for a simplified separable model

We consider a simplified buffer insertion problem: the delay of a module is independent of the load of the module, and all the nets have only two pins, then the number of buffers on a wire does not have any relation with the delays of other components. Let $K_{ij} = \mathcal{F}_{ij}(d_{ij})$, where $\mathcal{F}_{ij}$ is the reverse function that gives the number of buffers K_{ij} for a given delay d_{ij}. In this paper, we only consider the buffer insertion in combinational circuits. Two dummy nodes s and t are introduced into the graph representing the circuit: s is connected to all the primary inputs, and t is connected from all the primary outputs. We introduce one edge from t to s with weight $-REQ$, where REQ is the timing constraint at t. This new graph $G(V, E)$ is called a *constraint graph*.

As shown in Fig. 6.2, on the same wire, with more and more buffers inserted, the amount of delay reduction by inserting a buffer gets smaller and smaller, and will finally become non-positive when the minimal delay is reached. Such a property of decreasing delay reduction with the increasing number of buffers is similar to the concept of convexity. The constraint graph excluding the edge from t to s is a DAG, the $\mathcal{F}_{ij}$ has only non-negative integer values, and d_{ij}s can only be discrete values. To the best of our knowledge, currently no algorithms

can solve this problem optimally. We connect each node to its nearest nodes as in Fig. 6.2 to get a piece-wise linear convex function $\mathcal{F}_{ij}$. For any edge (i,j), let l_{ij} be the minimal delay of edge (i,j) that can be achieved by buffering, and u_{ij} be the delay of edge (i,j) without buffers. Let $l_{ts} = u_{ts} = -REQ$, and $\mathcal{F}_{ij}(d_{ij})$ be equal to ∞ when $d_{ij} < l_{ij} \vee d_{ij} > u_{ij}$. It is obvious that if there is an optimal solution for the buffering problem, $l_{ij} \leq d_{ij} \leq u_{ij}$. Then the timing constrained buffering problem can be formulated as: given a constraint graph $G(V,E)$,

$$(P1:)\ Minimize \quad \sum_{(i,j)\in E} \mathcal{F}_{ij}(d_{ij})$$
$$\text{s.t.} \qquad t_j \geq t_i + d_{ij} \qquad \forall (i,j) \in E$$

We define $\mathcal{C}_{ij}(x) = \min_{x' \leq x} \mathcal{F}_{ij}(x')$, then the following problem formulation P2 can be proved to be equivalent to P1. Given a constraint graph $G(V,E)$,

$$(P2:)\ Minimize \quad \sum_{(i,j)\in E} \mathcal{C}_{ij}(d_{ij})$$
$$\text{s.t.} \qquad t_j = t_i + d_{ij} \qquad \forall (i,j) \in E$$

Suppose vector $t^* \in \mathcal{R}^V$ and d^*_{ij} form an optimal solution of P1. We have $\mathcal{F}_{ij}(d^*_{ij}) = \min_{x' \leq t^*_j - t^*_i} \mathcal{F}_{ij}(x')$. Otherwise, since $d^*_{ij} \leq t^*_j - t^*_i$, we can always adjust d^*_{ij} such that $\mathcal{F}_{ij}(d^*_{ij}) = \min_{x' \leq t^*_j - t^*_i} \mathcal{F}_{ij}(x')$, and thus objective value decreases. Thus, if $d_{ij} = t^*_j - t^*_i$ in problem P2, the objective value of P2 is equal to the optimal objective value of P1, so the optimal value of P1 should be greater than or equal to the optimal value of P2. Similarly, it can be proved that the optimal value of P2 should be greater than or equal to the optimal value of P1. Thus, the optimal value of P1 should be equal to the optimal of P2. So P2 is equivalent to P1.

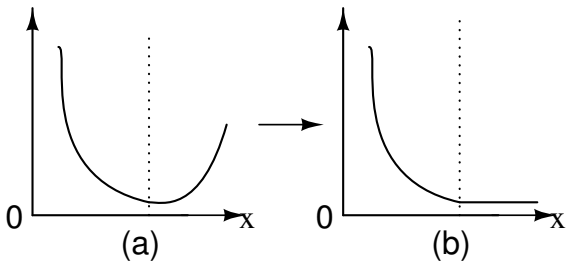


Figure 6.3. The transformation from (a) $\mathcal{F}_{ij}(x)$ to (b) $\mathcal{C}_{ij}(x)$.

Fig. 6.3 shows an example of the transformation from a general function $\mathcal{F}_{ij}(x)$ to $\mathcal{C}_{ij}(x)$.

For the simplified buffering problem, the $\mathcal{F}_{ij}(d_{ij})$ is non-increasing when d_{ij} is within the bounds. So

$$\mathcal{C}_{ij}(d_{ij}) = \begin{cases} \mathcal{F}_{ij}(d_{ij}) & \text{if } d_{ij} \leq u_{ij} \\ \mathcal{F}_{ij}(u_{ij}) & \text{otherwise} \end{cases}$$

The KKT optimality conditions become [**55**]: $\exists \mathbf{x} \in \mathcal{R}^E$, such that

$$t_j = t_i + d_{ij} \quad \forall (i,j) \in E \tag{6.6}$$

$$-\mathcal{C}_{ij}^{+}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(d_{ij}) \quad \forall (i,j) \in E \tag{6.7}$$

$$\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 \qquad \forall i \in V \tag{6.8}$$

where $\mathcal{C}_{ij}^{-}(x)$ and $\mathcal{C}_{ij}^{+}(x)$ represent the left derivative and the right derivative of function $\mathcal{C}_{ij}$ at point x, respectively.

Now we briefly explain these conditions. Suppose a cut C splits the vertices into two sets S and T. We define set

$$A = \{(i,j) | (i,j) \in C \wedge i \in S \wedge j \in T\},$$

and set

$$B = \{(i,j)|(i,j) \in C \wedge i \in T \wedge j \in S\}.$$

We decrease the t labels of the vertices in T by a small amount ϵ, then the change of the objective per unit of ϵ is

$$P(t,C) = -\sum_{(i,j)\in A} \mathcal{C}_{ij}^{-}(d_{ij}) + \sum_{(i,j)\in B} \mathcal{C}_{ij}^{+}(d_{ij}),$$

where $d_{ij} = t_j - t_i$. If this change is negative, we can always get a smaller objective value by decreasing the t labels of the vertices in T. So for the optimal solution, this change should be non-negative for all the cuts. Define

$$\lambda(t) = \max\left(0, -\min_{all\ cut\ C} \frac{P(t,C)}{|C|}\right),$$

where $|C|$ is the number of edges in the cut C. Karzanov and Mccormick [**55**] have proved that t is optimal iff $\lambda(t) = 0$. They also proved that

Theorem 6.1. *A real λ is an upper bound on $\lambda(t)$ if and only if there exist a vector $x \in \mathcal{R}^E$ such that*

$$\sum_{(i,j)\in E} x_{ij} - \sum_{(j,i)\in E} x_{ji} = 0 \ \forall i \in V,$$

$$\mathcal{C}_{ij}^{+}(d_{ij}) - x_{ij} \geq -\lambda,$$

and

$$x_{ij} - \mathcal{C}_{ij}^{-}(d_{ij}) \geq -\lambda.$$

Thus if $\lambda = 0$, we have

$$\mathcal{C}_{ij}^{+}(d_{ij}) \geq x_{ij},$$

and

$$\mathcal{C}_{ij}^{-}(d_{ij}) \leq x_{ij}.$$

Now we get all these conditions.

Eq.(6.6) shows that $t_j = t_i + d_{ij}$, but the delay of a gate or a wire may not be continuous but discrete, thus, this equation is hard, if not impossible, to satisfy in reality. So we relax it to $t_j \geq t_i + d_{ij}$. If this relaxed condition is satisfied, the timing constraint is satisfied.

Algorithm `ConvexCost-Buffering`
`ComputeTimingAndSlack(G);`
$d_{ij} \leftarrow u_{ij} \ \forall (i,j) \in E$
$c_{ij} \leftarrow -\mathcal{C}_{ij}^{-}(d_{ij}) \ \forall (i,j) \in E \wedge S_{ij} < 0$
$c_{ij} \leftarrow 0 \ \forall (i,j) \in E \wedge S_{ij} \geq 0$
while `there exist positive cycles in G`
  `Augment maximal flows using` s `as the source and` t `as the sink;`
  `Select a min-cut` M`;`
  `Insert a buffer into each wire in` M`;`
  `UpdateTimingSlack(G);`
  $\mathbf{c_{ij}} \leftarrow -\mathcal{C}_{\mathbf{ij}}^{-}(\mathbf{d_{ij}}) - \mathbf{f_{ij}} \ \forall (\mathbf{i},\mathbf{j}) \in \mathbf{E} \wedge \mathbf{S_{ij}} < \mathbf{0}$
  $c_{ij} \leftarrow 0 \ \forall (i,j) \in E \wedge S_{ij} \geq 0$

Figure 6.4. Convex-cost flow based buffering algorithm with the simplified separable model.

Now we design a heuristic to handle the buffering problem for the simplified model. The pseudo-code of this algorithm called ConvexCost-Buffering is shown in Fig. 6.4. Let c_{ij}, f_{ij}, and S_{ij} denote the capacity, the flow, and the slack of the edge (i,j), respectively. One thing we need to note is that when the flow flows through an edge, we do not introduce a residual backward edge, that is, the flows always flow through the forward edges. Since x is the network flow, condition (6.8) is always satisfied. When a new buffer is inserted into a wire e, the delay of e decreases, while based on the assumption that the interaction between the delays of components is ignored, we know that the delays of the fanin edges of e do not

change. Thus, when a buffer is inserted into each wire in the min-cut, the maximal delay decreases, so eventually the relaxed Eq.(6.6) is also satisfied. The buffers are always inserted into the wires in the min-cut, so the *accumulated* flow x_{ij} on any wire (i, j) in the min-cut reaches the capacity $(-\mathcal{C}_{ij}^{-}(d_{ij}))$ and is no less than $(-\mathcal{C}_{ij}^{+}(d_{ij}))$. While for the other wires, $-\mathcal{C}_{ij}^{+}(d_{ij}) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(d_{ij})$ is obviously satisfied. Thus, Eq.(6.7) is also satisfied. We need to note that according to Eq.(6.6), the Eq. (6.7) is equivalent to

$$-\mathcal{C}_{ij}^{+}(t_j - t_i) \leq x_{ij} \leq -\mathcal{C}_{ij}^{-}(t_j - t_i),$$

but since Eq.(6.6) is relaxed to an inequality in this part, these two conditions are not equivalent now. So our algorithm actually satisfies a relaxed Eq. (6.7).

Algorithm `MinCut-Buffering`
`maxdelay` $\leftarrow$ `ComputeTimingAndSlack(G);`
$d_{ij} \leftarrow u_{ij}\ \ \forall (i,j) \in E$
$c_{(i,j)} \leftarrow -\mathcal{C}_{ij}^{-}(d_{ij})\ \ \forall (i,j) \in E \wedge S_{ij} < 0$
$c_{(i,j)} \leftarrow 0\ \ \forall (i,j) \in E \wedge S_{ij} \geq 0$
while `maxdelay`$> REQ$
  `Find a min-cut` M `of` G`;`
  `Insert one buffer into` $(i,j)\ \ \forall (i,j) \in M$`;`
  `maxdelay` $\leftarrow$ `UpdateTimingSlack(G);`
  $\mathbf{c_{(i,j)} \leftarrow -\mathcal{C}_{ij}^{-}(d_{ij})\ \ \forall (i,j) \in E \wedge S_{ij} < 0}$
  $c_{ij} \leftarrow 0\ \ \forall (i,j) \in E \wedge S_{ij} \geq 0$

Figure 6.5. Min-cut based buffering algorithm with the simplified model: a greedy variant of ConvexCost-Buffering.

ConvexCost-Buffering computes the flows incrementally, so the flow through an edge before one iteration is the sum of the flows through this edge in the previous iterations. Intuitively, if we do not consider the existing flows, that is, the capacity of any edge (i, j) is set to be -$\mathcal{C}_{ij}^{-}(d_{ij})$ instead of the residual capacity, the number of inserted buffers might not be large, either. Based on this, we design a *greedy* variant of ConvexCost-Buffering

based on min-cut, called MinCut-Buffering as shown in Fig. 6.5, for comparison. The only difference between MinCut-Buffering and ConvexCost-Buffering is that the historical flows are not subtracted from the capacities of the edges after each iteration in MinCut-Buffering, so the capacities are *not* the *residual* capacities.

6.3. Implementation issues

In reality, the delay of a wire or a module is a function of the resistance, the capacitance and the *load*, thus, the number of buffers inserted into a wire also depends on the load. Therefore, the $\mathcal{C}_{ij}(d_{ij})$ may get different values for different loads. Since loads are not constants during the process, $\mathcal{C}_{ij}$ is a function of the delays of both the fanout edges and the wire (i, j). Thus, it is impossible to get a *separable* objective function. In this subsection, we consider these issues.

The general framework of network flow based buffering algorithms, called NetworkBIN, is shown in Fig. 6.6. At the beginning, there are no buffers in the circuit. We use Ford-Fulkerson algorithm [**44**] to compute the maximal flow from s to t, and simultaneously get the min-cut. The most significant difference between NetworkBIN and the previous two algorithms for the simplified model is that when a new buffer is inserted into a wire, NetworkBIN inserts buffers immediately after the other branches in order to decouple all the other branches.

When the relations between component delays are considered, the sensitivity $(-\mathcal{C}_{ij}^{-}(d_{ij}))$ computation becomes difficult. As shown in Fig. 6.7(a), a net in a circuit has four pins, the slacks on the wires (b, c), (b, d) and (b, e) are much different, and these slacks are all negative. If NetworkBIN finds a min-cut through (b, c), (b, d) and (b, e), it will insert one buffer into each wire. After that, their slacks might still be negative, so more buffers might be required to be inserted. This solution is shown in Fig. 6.7(b). But we may find a better solution by

```
Algorithm NetworkBIN
maxdelay ← ComputeTimingAndSlack(G);
SetCapacity(G);
while maxdelay> REQ
   Find a min-cut of G;
   for each wire (u, v) in the min-cut
      Insert one buffer into (u, v);
      Decouple the other wires that
         connect from u;
   maxdelay ← UpdateTimingSlack(G);
   UpdateCapacity(G);
```

Figure 6.6. Network flow based buffer insertion framework.

decreasing the delay of the wire (a, b) through decoupling some less critical branches: since the slacks of the branches are different, we may only need to insert one buffer into the most critical branch, and insert buffers into other branches to decouple them. This solution is shown in Fig. 6.7(c). The reason that NetworkBIN generates worse solutions is that the sensitivity computation is based on the assumption that no buffers will be inserted into all the other wires, but actually, the sensitivities of wires might change when new buffers are inserted into other branches. An approach to avoid this problem is to enforce that the min-cut contains at most one wire for each net, and the sensitivity computation also considers the delay changes of the fanin edges introduced by the decoupling buffers. Thus, during the computation of the delay sensitivity, we always assume that all the other branches are decoupled. In addition, in each iteration, for each net, we set the capacity of the most critical branch (if there are more than one most critical branches, pick either one of them) to be its delay sensitivity, and set the capacities of other branches to be 0. For the example of Fig. 6.7(a), when we compute the delay sensitivity of (b, e), we assume that there are buffers immediately after the branches (b, c) and (b, d), and based on this assumption, we compute the delay changes of the wire (a, b) and (b, e) when a new buffer is inserted into

(b, e). Then if the most critical branch (b, e) is in the found min-cut, we insert a buffer into (b, e), and decouple the branches (b, c) and (b, d) by inserting buffers near the split point of these branches, so we get the solution as shown in Fig. 6.7(c).

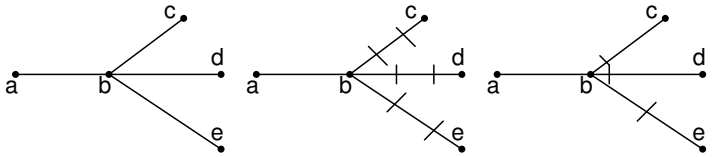


Figure 6.7. The sensitivity computation problem. The short line segments represent buffers.

We have two different algorithms sharing the NetworkBIN framework: one considers the existing flows, while the other one does not.

We first present our convex-cost flow based buffering algorithm called CostBIN, which is a realistic version of the ConvexCost-Buffering. The capacity setting step in CostBIN works as follows. The slacks and timing information are computed in the previous step ComputeTimingAndSlack with the timing constraint as the required time at sink t. Then, for any edge (i, j), if $S_{ij} \geq 0$, it should be excluded from the network, so its capacity is 0. In order to avoid the previously mentioned sensitivity computation problem, CostBIN assumes that all the other wire branches are decoupled in the delay sensitivity computation step. With this assumption, let $output((i, j))$ represent the set containing all the fanout edges of (i, j), then the maximal change of the sum of the delays of (i, j) and its fanin edge (k, i) is

$$T_{ij} = \delta_{ij} + \hat{R}_{ij}(number(output((k, i)))C_b + Cp_{ij} - \hat{C}_{ki}),$$

where function $number(S)$ represents the number of elements in set S, $\hat{R}_{ij}$ represents upstream resistance at the input of wire (i, j), $\hat{C}_{ki}$ represents downstream capacitance at the

output of wire (k, i), p_{ij} represents the wire length from the input of wire (i, j) to the first buffer of the wire, C_b represents input capacitance of a buffer, and C represents capacitance of a unit-length wire. If $T_{ij} < 0$, the capacity is $-1/T_{ij} - f_{ij}$; otherwise, it is infinity. Since the δ_{ij} is 0 for any edge in the forbidden area, its capacity is infinity.

If we do not consider the existing flows on edges, we have a greedy version of CostBIN. This variant is called CutBIN, which is a realistic version of the MinCut-Buffering. An approach to speed up the algorithm is to restrict the flow at a sub-network that contains only the most critical paths. CutBIN uses the current maximal delay from s to t as the required time at sink t, and computes the slacks of components. A subgraph containing only the components with slacks less than S_{th} is called the *critical subgraph* of the circuit. Besides the speeding up effect, S_{th} can also avoid the sensitivity computation problem. The S_{th} is a user specified value or adaptively computed using the following heuristic. For each component, the difference between the slacks of the most critical and the second most critical fanout edges is computed, then select the minimal difference for all the components as S_{th}; if S_{th} is less than a specified lower bound, it is set to be the lower bound. The lower bound is used to speed up the algorithm. This step is performed once before the capacity setting step in each iteration, so the S_{th} may change for different iterations. One thing we need to note is that this critical subgraph technique can also be used in CostBIN.

The capacity setting step in CutBIN works as follows. The slacks and timing information are already computed in the previous step ComputeTimingAndSlack that uses the current maximal delay from s to t as the required time at sink t. For any edge e, if $S_e \geq S_{th}$, it should be excluded from the critical subgraph, so we set its capacity 0. As mentioned before, we need to consider the influence of decoupling buffers on the wire delay, so if $T_e < 0$, the

capacity of wire e with $S_e < S_{th}$ is $-1/T_e$, otherwise, it is infinity. Since the δ_e is equal to -0 for any edge $e \in E^F$, its capacity is infinity.

6.4. Extensions

Now we extend these algorithms to handle more realistic situations: the buffers have several different sizes, and the candidate locations for buffering are specified. The objective is to minimize the total area of buffers.

The buffering candidate locations and the steiner points split the wires into many segments. So the buffering locations can only be at the starting point of segments. For each segment (i, j) with a buffering candidate location, we can easily compute the delay difference T_{ij} when a new buffer is inserted or the original buffer is replaced by a bigger buffer at the starting point. The capacity in CostBIN is set to be $|\Delta_S/T_{ij}| - f_{ij}$, where Δ_S is the buffer size difference. We can similarly set the capacities in CutBIN. The buffers with the minimal size are always selected as the decoupling buffers.

When the timing constraint is satisfied, we can refine the solution to save buffers even more. Since now the required arrival time at each sink of each net is known, for each net especially the large net, Lillis's min-cost buffering work [**62**] can be used to find a solution that has minimal buffer area and satisfies the timing requirement. It is obvious that this solution has at most the buffer area of the original solution. But we need to note that even with speed-up techniques as in [**90**], the min-cost buffering still introduces much memory overhead if the circuit has very big nets.

Table 6.2. Comparison results of CostBIN, CutBIN and [**64**]

circuit			[**64**]		CutBIN				CostBIN		
Name	Module	Wire	Buffer	Time (s)	Buffer	Time (s)	Reduce	Speed up	# Buffer	Time (s)	Reduce
c1	22	98	172	0	146	0.01	15%	1×	127	0.01	26%
c2	44	197	305	5	176	0.02	43%	250×	145	0.01	52%
c3	81	398	606	16	306	0.06	50%	266.67×	276	0.04	54%
c4	159	799	887	10	464	0.14	48%	71.43×	449	0.11	49%
c5	258	1037	1096	28	767	0.25	30%	112.00×	709	0.34	35%
c6	505	2039	2140	20	1251	1.04	42%	19.23×	1137	1.80	47%
c7	2514	10039	10297	170	5612	20	45%	8.50×	5206	40	49%
c8	5034	20038	21201	344	10059	58	53%	5.93×	9403	142	56%
Avg							41%				46%

6.5. Experimental results

We have implemented the CostBIN and CutBIN algorithms in C. We use the parameters from 100-nanometer technology [**34**]. We got four test cases from Liu, and generated additional four cases using the case generator in [**64**]. All experiments are run on a Linux PC with a 2.4 GHz Xeon CPU and 2.0 GB memory.

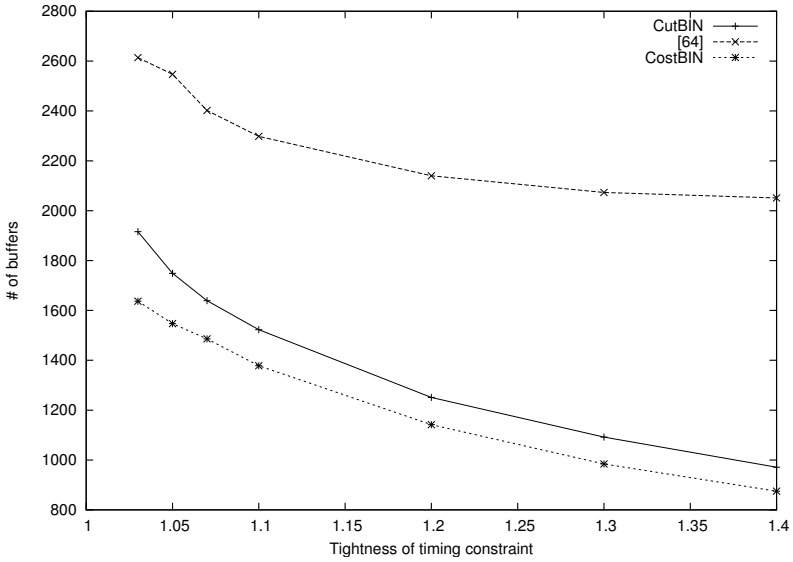


Figure 6.8. The influence of the tightness of timing constraint on the number of inserted buffers.

In order to test the benefit of the network flow based buffer insertion, we got the source code of the algorithm in [**64**] from Liu for comparison. The comparison results of CutBIN, CostBIN and [**64**] are shown in Table 6.2. We set the timing constraints to be 0.2 times larger than the minimal delays from s to t that can be achieved by buffering, which are computed by the min-delay buffering algorithm in [**64**]. The four sub-columns under "CutBIN" column show the number of buffers inserted by CutBIN, the running time of CutBIN, the reduction percentage of the number of buffers inserted by CutBIN compared with the number of buffers inserted by [**64**], and the speed-up of CutBIN over [**64**], respectively. The three sub-columns under "CostBIN" column show the number of inserted buffers by CostBIN, the running time of CostBIN, and the reduction percentage of the number of buffers inserted by CostBIN compared with the number of buffers inserted by [**64**], respectively.

We can see that CutBIN achieves 41% reduction of the number of inserted buffers on average compared with [**64**] and is much more efficient than [**64**], and CostBIN even achieves 46% reduction of the number of inserted buffers on average compared with [**64**]. We can also see that CostBIN always inserts less buffers than CutBIN. The reason behind this is that CostBIN considers the relations of different iterations by incremental flows, but CutBIN is only a greedy algorithm.

Using the case with 505 modules and 2,039 wires as an example, we test the influence of the tightness of timing constraint on the number of inserted buffers. As shown in Figure 6.8, both CostBIN and CutBIN save even more buffers than [**64**] when the timing constraints are looser, which means that they are more effective buffering approaches. Especially, CostBIN always inserts less buffers than CutBIN does under the same timing constraint, and when the timing constraint is tighter, CostBIN saves more and more buffers than CutBIN does. During our experiments, we observed that the results of [**64**] are very sensitive to the selection

of the weights in the objective functions in the Lagrangian relaxation, and it is required to select the weights *manually* in order to get a good result. In order to compare fairly, for each case, we run the approach in [**64**] several times and select the weights such that the timing constraint is satisfied while the number of buffers is the minimal among these trials. The objective function of the Lagrangian relaxation approach is a weighted sum of the number of buffers and the maximal delay. When the weight of the number of buffers increases, the approach in [**64**] takes more effort on the reduction of the number of buffers, while the delay keeps increasing. Thus, we start from a small weight, and increase the weight until we find a situation where the timing constraint is violated.

One thing we need to note is that our approaches may not get a valid solution when the tightness is 1.0. For this situation, there might be only one valid solution. Since we have not introduced the backward flows in our approaches, if our approaches make the wrong decision in one iteration, our approaches do not have the opportunity to correct it, so the valid solution cannot be found. This is a limitation we are trying to avoid. But because of the interaction of the delays of components, how to get a converged solution when the backward flows are introduced is still a hard problem.

Table 6.3. Comparison results of CostBIN, CutBIN, NetBIN and [**63**]

circuit			[**63**]		CostBINv1				CostBINv2	NetBIN	CutBIN	
Name	Cand #	#Segs $S<0$	Area	Time (s)	Area	Time (s)	Reduce	Speed up	Area	Area	Area	Time (s)
c1	695	689	598.5	151	333.5	0.17	44%	888×	301.5	629.0	340.5	0.17
c2	1278	1269	659.0	90	368.0	0.37	44%	243×	321.5	773.5	381.0	0.40
c3	2564	2564	1955.0	256	643.5	2.58	67%	99×	560.5	1473.0	660.0	2.98
c4	5168	4821	2636.0	979	1089.0	5.54	59%	177×	940.0	3096.0	1092.0	8.49
c5	5579	5507	2933.5	859	1594.5	16.43	46%	52×	1414.0	-	1668.0	11.05
c6	11163	11126	4842.5	1855	2604.0	32.83	46%	57×	2262.5	-	2762.0	25.00
c7	53612	53561	24272.0	4662	11234.0	682.73	54%	7×	-	-	11823.0	555.51
c8	107931	107847	50004.0	18550	21191.5	2399.47	58%	8×	-	-	22418.0	1716.93
Avg							52%					

The dynamic programming based work [**63**] extends the Lagrangian relaxation based buffering algorithm in [**64**] to handle the cases where there are multiple types of buffers and the candidate locations for buffering are specified. We also implemented a net-based buffering approach called NetBIN. NetBIN buffers nets one by one until the timing constraint is satisfied. We also use a refinement step at the end of NetBIN to reduce the buffers, and our experience is that the refinement can reduce the buffers at least 50%. Table 6.3 shows the comparison results of CostBIN, CutBIN, NetBIN and [**63**]. There are four types of buffers, and the buffering candidate locations are specified. The timing constraints are 0.2 times greater than the minimal achievable delays from s to t by buffering, computed by the min-delay buffering algorithm in [**63**]. We observed that NetBIN inserts more buffers than [**63**] for most of those cases. NetBIN does not budget the timing in a global view, so the required time at sinks may mislead the buffering. Thus, buffering each net such that the required time at the root of the net is maximized does not guarantee that the delay from s to t is minimized. For example, for the last four testcases, it is difficult for NetBIN to get a feasible solution in 5 hours. We also observed that the final solutions computed by the min-cost buffering algorithm in [**63**] often do not satisfy the timing constraint, so we select the solution that satisfies the timing constraint and has the minimal buffer area *during* the iterations. We also implemented the refinement step in CostBIN. The CostBIN without the refinement is denoted as CostBINv1, and the CostBIN with the refinement is denoted as CostBINv2. Column 3 shows the number of wire segments with negative slacks. Note that the wires in these test cases are separated into many small wire segments in this part, and the number of buffering locations (shown in Column 2) is equal to the number of wire segments. The results indicate that CostBINv1 achieves 52% reduction on the total area of buffers on average. The refinement step in CostBINv2 reduces additional 12% of the

buffering area on average, but "c7" and "c8" cannot be finished for CostBINv2 because of the big memory overhead. Also, the CostBIN inserts fewer buffers than the CutBIN in this situation.

6.6. Conclusions

With the development of VLSI technology, buffering becomes an effective technique to the problem of growing interconnect delays in modern designs. Most previous researches were focusing on the problem of buffer insertion in a single net, which may introduce the over-buffering problem in a whole circuit. In this section, we relate the timing constrained minimal buffer insertion problem to the convex-cost flow dual problem, and propose two algorithms CostBIN and CutBIN based on convex-cost flow and min-cut techniques, respectively, to solve the buffering problem in large combinational circuits. We compare our approaches to a Lagrangian relaxation based buffer insertion algorithm proposed by Liu et al. [**64**]. Experimental results demonstrate that our approaches are efficient and on the average, achieve 46% and 41% reduction, respectively, on the number of buffers inserted in comparison to the latter. The CostBIN algorithm always inserts less buffers than the CutBIN algorithm does, which implies the relations between iterations should be considered.

CHAPTER 7

Efficient Algorithms for Buffer Insertion under Process Variations

Interconnect delays have become dominant in modern deep sub-micron (DSM) designs with the continuously shrinking VLSI feature sizes. Buffer insertion is widely used to reduce the interconnect delays [**6**, **7**, **21**, **62**, **89**, **117**]. These researches are focusing on the buffering on a single net. Saxena *et al.* [**83**] predicted that 70% of cells will be dedicated to buffers within a few process generations. An effective buffer insertion in a whole circuit is needed. Recently, Sze *et al.* [**95**] proposed a path-based buffering technique; Chen and Zhou [**23**] proposed a network flow based buffering technique; Waghmode *et al.* [**104**] proposed a look-ahead approach to handle the buffering of a whole circuit. But none of them considered process variations.

Process variations become prominent in fabrication. The traditional corner-based analysis and optimization techniques become prohibitive. Nowadays, many statistical static timing analysis (SSTA) approaches [**14**, **103**] emerged. Propagating distributions instead of single values, these techniques are much more efficient. There are some work that considered the buffering problem under process variations [**37**, **40**, **58**, **109**, **110**]. Khandelwal *et al.* [**58**] considered only the wire length variation, and the pruning approaches are expensive. Davoodi *et al.* [**37**] considered the correlations between the delay and the downstream capacitance, but the pruning approach is still prohibitive. Both techniques use a two-phase scheme for the merge of solution lists: the solutions are generated in the first phase, and pruned in the second phase. Then for the merge procedure that merges two solution lists with m solutions

and n solutions respectively, these techniques run in at least $O(m^2n^2)$ time (the number of solutions before pruning is $O(mn)$, and the prune techniques compute the relation between each pair of solutions), which is prohibitive for large m and n. Xiong *et al.* [**110**] proposed to compute the joint probability density function (JPDF) of solutions numerically, which is demonstrated to be very inefficient [**109**]. Xiong and He [**109**] ignored correlations between the delay and the downstream capacitance, tried to propose a transitive closure technique for probabilistic buffering, and claimed that a direct extension of the deterministic buffering algorithms as in [**88**, **101**] can be used in the statistical situation. But the ordering property[1] is not generally true, and as shown in our experimental result part, the quality of buffering solutions from [**109**] is not good compared with the deterministic buffering that assumes that all the random variables have the nominal values. Deng *et al.* [**40**] claimed that the consideration of process variations is not necessary for the buffering of two-pin nets. But the conclusion does not hold for nets with multi pins according to the results in the other existing statistical buffering work. In addition, all of them are considering delay minimization. For the cost minimization problem, the solution space is much bigger, and [**90**] has proved that the problem even without process variations is NP-complete. The pruning approaches in [**37**, **58**, **110**] are expensive.

Buffer insertion is generally a discrete optimization problem: there are only limited types of buffers in a library, and the possible buffering locations are restricted because of forbidden areas. For discrete optimization problems, statistical optimization techniques as in [**32**, **48**, **92**] may not get much better solutions than deterministic optimization techniques.

[1]For any two random variable A and B with Gaussian distributions, either $P_r(A \geq B) \geq \gamma$ or $P_r(B \geq A) \geq \gamma$, where γ is a given constant number in (0.5, 1].

Sinha *et al.* [**92**] analyzed the situation where the statistical gate sizing is necessary. It is well-known that statistical optimization is much less efficient than its deterministic counterpart in general. Therefore, for discrete optimization problem, if we do the deterministic optimization first, and then an iterative refinement based on the deterministic results may get decent results efficiently.

The slew constrained buffering is done before the delay constrained buffering in the IBM physical synthesis methodology, and most nets may already satisfy the delay constraints after the slew constrained buffering [**74**]. Recently, some slew constrained buffering researches [**51**, **75**] emerged. But they did not consider process variations. He *et al.* [**50**] considered the slew propagation, but it did not consider the variances on the interconnects and the correlations among parameter variations.

This chapter considers both the delay constrained and the slew constrained min-cost buffering problems. With the huge number of buffers, the power consumption of those buffers becomes prominent, and is proportional to the total buffer area [**62**]. Thus, our objective is to minimize the total buffer area. Our approaches can be easily extended to the optimization of other cost metrics. The rest of this chapter is organized as follows. Section 7.1 presents the delay modeling of interconnects, and Section 7.2 elaborates how to compute the multiplication of two random variables, which will be used in our statistical buffering algorithm. Section 7.3 proposes our buffering approaches to handle the delay constrained and the slew constrained cost minimization problem in buffering with the consideration of process variations. Section 7.4 proposes our buffering approach for a combinational circuit with the consideration of process variations. Then, Section 7.5 shows the experimental results on our approaches, and finally, the conclusion is drawn in Section 7.6.

7.1. Preliminary

Given a routing tree with a distributed RC network, the classic van Ginneken's algorithm [**101**] computes non-inferior solutions bottom-up from the sinks to the root during the buffering. The objective is to insert buffers such that the maximal delay from the root to sinks is minimized.

Lillis *et al.* [**62**] extended van Ginneken's algorithm to consider the cost minimization. Let (P_v, D_v, C_v) represent a buffering solution of the subtree rooted at node v, where P_v represents the cost, D_v represents the maximal delay to sinks, and C_v is the downstream capacitance. If there are two solutions (P_1, D_1, C_1) and (P_2, D_2, C_2) satisfying $P_1 \leq P_2$, $D_1 \leq D_2$ and $C_1 \leq C_2$ at one node, (P_2, D_2, C_2) is inferior, and (P_1, D_1, C_1) *dominates* (P_2, D_2, C_2).

When process variations are considered, the wire length, wire width and wire height are no longer deterministic values. [**52**] shows that the introduced variation on the interconnect can be more than 30% of the nominal value. The parameters (e.g., Leff) of buffers are no longer deterministic values either. We assume that all the random variables have Gaussian distributions. Through principal component analysis (PCA) [**14**, **59**], each random variable X is represented as a canonical form: $x_0 + \sum_{i=1}^{n} x_i \epsilon_i + x_{n+1} R_x$, where ϵ_i's are independent random variables with the standard Gaussian distribution, x_0 is the mean value of X, x_i's are coefficients, and R_x is an independent random variable with the standard Gaussian distribution.

When a wire or a buffer is attached to a node, the delay computation of the new solution involves the multiplication of two Gaussian random variables for the Elmore delay model. Assuming that the result still has the Gaussian distribution, [**109**] gets analytic formula for

this operation. We will develop a new analytic approach to compute the multiplication of Gaussian variables in Section 7.2. For the D2M delay model, the approach in [**5**] is used.

In this chapter, we are considering the following four related buffering problem.

Problem 7.1. *Delay minimization on a net: given a routing tree where possible buffering locations are specified, insert buffers such that the probability that the maximal delay from the root to sinks is no greater than a given number is maximized.*

Problem 7.2. *Delay constrained min-cost buffering on a net: given a routing tree where possible buffering locations are specified, insert buffers such that the probability that the maximal delay from the root to sinks is no greater than a given number is no less than a given constraint η and the cost (e.g., total buffer area) is minimized.*

Problem 7.3. *Slew constrained min-cost buffering on a net: given a routing tree where possible buffering locations are specified, insert buffers such that the probability that the input slew at each buffer or sink is no greater than a given number is no less than a given constraint η and the cost (e.g., total buffer area) is minimized.*

Problem 7.4. *Delay constrained min-cost buffering in a combinational circuit: given a routing combinational circuit where possible buffering locations are specified, insert buffers such that the probability that the maximal delay from the primary inputs to the primary outputs is no greater than a given number is no less than a given constraint η and the cost (e.g., total buffer area) is minimized.*

Delay minimization problem is just a special case of cost minimization problem, so we are focusing on how to solve the last three problems. Note that it is not necessary to consider the delay constraints and the slew constraints simultaneously, because most of nets satisfy

the delay constraints when the slew constraints are satisfied, and those nets violating the delay constraints after the slew constrained buffering can easily satisfy the delay constraints through the delay constrained buffering [**51**].

7.2. Statistical multiplication

With the consideration of the process variations, the delay and capacitance parts of each solution become random variables. Here, we assume that all the random variables ($r(u,v)$, $c(u,v)$, d_b, r_b, c_b, D_v and C_v) have the Gaussian distribution. Each random variable X is represented in a canonical first-order form:

$$x_0 + \sum_{i=1}^{m} x_i \epsilon_i + x_{m+1} R_x,$$

where ϵ_i's are independent random variables with the standard Gaussian distribution, x_0 is the mean value of X, x_i's are coefficients, and R_x is an independent random variable with the standard Gaussian distribution. Here, R_x represents the independent variation. This canonical form is available through principal component analysis (PCA) [**14**, **59**].

When a wire $w(u,v)$ is attached to a node v, the computation of the maximal delay involves the multiplication between the random variables ($r(u,v)C_v$ and $r(u,v)c(u,v)$). Keeping all the random variables in a first-order form as

$$x_0 + \sum_{i=1}^{m} x_i \epsilon_i,$$

Xiong and He [**109**] used the moment-matching approach to compute the multiplication between the Gaussian random variables. If many random variables have their independent variations, the number of non-zero coefficients may become larger and larger during the

computation. Thus, we prefer the canonical form that models independent randomness. We need to keep that canonical form during the buffering.

Let

$$\begin{aligned} C_v &= C_{v0} + \sum_{i=1}^{m} \alpha_i \epsilon_i + \alpha_{m+1} R_{C_v}, \\ D_v &= D_{v0} + \sum_{i=1}^{m} \beta_i \epsilon_i + \beta_{m+1} R_{D_v}. \end{aligned}$$

Also suppose

$$\begin{aligned} r(u,v) &= r_0 + \sum_{i=1}^{m} \gamma_i \epsilon_i + \gamma_{m+1} R_r, \\ c(u,v) &= c_0 + \sum_{i=1}^{m} \zeta_i \epsilon_i + \zeta_{m+1} R_c. \end{aligned}$$

Then the solution after attaching a wire (u,v) to v has

$$C_u = C_{v0} + c_0 + (\alpha^T + \zeta^T)\Upsilon + \alpha_{m+1} R_{C_v} + \zeta_{m+1} R_c \tag{7.1}$$

where Υ represents the column vector $(\epsilon_1, ..\epsilon_m)^T$ (α, β, γ and ζ also represent the corresponding column vectors), For C_u, we get

$$\sigma^2(C_u) = \sum_{i=1}^{m} (\alpha_i + \zeta_i)^2 + \alpha_{m+1}^2 + \zeta_{m+1}^2.$$

By moment matching, we get

$$C_u = (C_{v0} + c_0) + (\alpha^T + \zeta^T)\Upsilon + \sqrt{\alpha_{m+1}^2 + \zeta_{m+1}^2} R_{C_u},$$

where R_{C_u} is an independent random variable representing the local variance. For D_u,

$$D_u = K + P\Upsilon + \Upsilon^T Q\Upsilon + R,$$

where

$$\begin{aligned}
K &= D_{v0} + r_0 C_{v0} + r_0 c_0/2 && (7.2)\\
P &= \beta^T + C_{v0}\gamma^T + r_0\alpha^T + 0.5c_0\gamma^T + 0.5r_0\zeta^T && (7.3)\\
Q &= \gamma\alpha^T + 0.5\gamma\zeta^T && (7.4)\\
R &= r_0\alpha_{m+1}R_{C_v} + C_{v0}\gamma_{m+1}R_r + \alpha^T\Upsilon\gamma_{m+1}R_r \\
&\quad +\gamma^T\Upsilon\alpha_{m+1}R_{C_v} + \gamma_{m+1}R_r\alpha_{m+1}R_{C_v} \\
&\quad +0.5(r_0\zeta_{m+1}R_c + c_0\gamma_{m+1}R_r + \zeta^T\Upsilon\gamma_{m+1}R_r \\
&\quad +\gamma^T\Upsilon\zeta_{m+1}R_c + \gamma_{m+1}\zeta_{m+1}R_rR_c) + \beta_{m+1}R_{D_v}. && (7.5)
\end{aligned}$$

Then

$$\begin{aligned}
E(D_u) &= K + tr(Q). && (7.6)\\
E(D_u^2) &= K^2 + PP^T + 2tr(Q^2) + tr(Q)^2 \\
&\quad +\beta_{m+1}^2 + \gamma_{m+1}^2 tr(\alpha\alpha^T) + \alpha_{m+1}^2 tr(\gamma\gamma^T) \\
&\quad +C_{v0}^2\gamma_{m+1}^2 + \gamma_{m+1}^2\alpha_{m+1}^2 + r_0^2\alpha_{m+1}^2 \\
&\quad +0.25r_0^2\zeta_{m+1}^2 + 0.25\gamma_{m+1}^2\zeta_{m+1}^2 \\
&\quad +0.25\zeta_{m+1}^2 tr(\gamma\gamma^T) + 0.25c_0^2\gamma_{m+1}^2
\end{aligned}$$

$$+0.25\gamma_{m+1}^2 tr(\zeta\zeta^T) + \gamma_{m+1}^2 tr(\zeta\alpha^T)$$

$$+C_{v0}c_0\gamma_{m+1}^2 + 2Ktr(Q). \tag{7.7}$$

Thus, we can compute $\sigma^2(D_u)$ according to

$$\sigma^2(D_u) = E(D_u^2) - E(D_u)^2. \tag{7.8}$$

And

$$\begin{aligned} cov(D_u, \epsilon_i) &= E(D_u\epsilon_i) \\ &= P_i. \end{aligned} \tag{7.9}$$

Similar as in [**103**],

$$D_u = (K + tr(Q)) + \sum_{i=1}^{m} P_i\epsilon_i + MR_{D_u} \tag{7.10}$$

where $M = \sqrt{\sigma^2(D_u) - PP^T}$.

Suppose

$$\begin{aligned} r_b &= r_{b0} + \sum_{i=1}^{m} \xi_i\epsilon_i + \xi_{m+1}R_{rb}, \\ c_b &= c_{b0} + \sum_{i=1}^{m} \theta_i\epsilon_i + \theta_{m+1}R_{cb}, \\ d_b &= d_{b0} + \sum_{i=1}^{m} \lambda_i\epsilon_i + \lambda_{m+1}R_{db}. \end{aligned}$$

When a buffer is attached to node v, the new solution has

$$C'_v = c_{b0} + \sum_{i=1}^{m} \theta_i \epsilon_i + \theta_{m+1} R_{cb}, \tag{7.11}$$

$$D'_v = L + J\Upsilon + \Upsilon^T \xi \alpha^T \Upsilon + H \tag{7.12}$$

where

$$J = \beta^T + \lambda^T + r_{b0}\alpha^T + C_{v0}\xi^T \tag{7.13}$$

$$L = D_{v0} + d_{b0} + r_{b0} C_{v0} \tag{7.14}$$

$$\begin{aligned} H = {} & \beta_{m+1} R_{D_v} + \lambda_{m+1} R_{db} + r_{b0}\alpha_{m+1} R_{C_v} \\ & + \xi^T \Upsilon \alpha_{m+1} R_{C_v} + \xi_{m+1} R_{rb} C_{v0} \\ & + \xi_{m+1} R_{rb} \alpha^T \Upsilon + \xi_{m+1} R_{rb} \alpha_{m+1} R_{cv}. \end{aligned} \tag{7.15}$$

Then, similarly, we get

$$E(D'_v) = L + \xi^T \alpha \tag{7.16}$$

$$\begin{aligned} \sigma^2(D'_v) = {} & JJ^T + \beta_{m+1}^2 + \lambda_{m+1}^2 + r_{b0}^2 \alpha_{m+1}^2 \\ & + \alpha_{m+1}^2 \xi^T \xi + C_{v0}^2 \xi_{m+1}^2 + \xi_{m+1}^2 \alpha^T \alpha \\ & + \xi_{m+1}^2 \alpha_{m+1}^2 + 2tr((\xi\alpha^T)^2). \end{aligned} \tag{7.17}$$

$$cov(D'_v, \epsilon_i) = J_i. \tag{7.18}$$

Similar as in [**103**],

$$D'_v = (L + \xi^T \alpha) + \sum_{i=1}^{m} J_i \epsilon_i + N R_{D'_v}, \tag{7.19}$$

where $N = \sqrt{\sigma^2(D'_v) - JJ^T}$.

7.3. Min-cost buffering on a net

7.3.1. Delay constrained buffering

With process variations, a straight-forward extension of the optimality condition of the deterministic min-cost buffering is that (P_1, D_1, C_1) is **inferior** iff there exist another solution (P_i, D_i, C_i) on the same node such that

$$Pr(P_1 \geq P_i, D_1 \geq D_i, C_1 \geq C_i) = 1.$$

In reality, using this prune criteria cannot prune many solutions since it is difficult for Gaussian random variables to satisfy the 100% probability, so we relax it to

$$Pr(P_1 \geq P_i, D_1 \geq D_i, C_1 \geq C_i) \geq \eta,$$

where η is a given real number in $(0.5, 1)$.

Xiong *et al.* [**110**] ignored correlations between delays and capacitances, and proved that the transitive ordering property is satisfied. Now we prove that the transitive ordering property does not always hold when the correlations between delay and capacitance are considered.

Theorem 7.1. *Suppose random variables C1, C2, C3, D1, D2, and D3 having a joint Gaussian distribution satisfy*

$$Pr(D_3 \geq D_2, C_3 \geq C_2) \geq \eta \text{ and } Pr(D_2 \geq D_1, C_2 \geq C_1) \geq \eta,$$

where $\eta \in [0.5, 1)$. *Then* $Pr(D_3 \geq D_1, C_3 \geq C_1) \geq \eta$ *does not always hold.*

Proof. Let A, B, C and D be random variables with a joint Gaussian distribution. We need to prove that if

$$Pr(A \leq 0, B \leq 0) \geq \eta \text{ and } Pr(C \leq 0, D \leq 0) \geq \eta,$$

where $\eta \in [0.5, 1)$, it is possible to have

$$Pr(A + C \leq 0, B + D \leq 0) < \eta.$$

We found the following case: the means of A and B are -0.68, the means of C and D are -0.32, and all of them have their standard deviations (σ) equal to 1. The covariance matrix is

$$\begin{pmatrix} 1.00 & -0.99 & 0.36 & -0.36 \\ -0.99 & 1.00 & -0.36 & 0.36 \\ 0.36 & -0.36 & 1.00 & 0.72 \\ -0.36 & 0.36 & 0.72 & 1.00 \end{pmatrix}$$

Then $Pr(A \leq 0, B \leq 0) = 0.5035$, and $Pr(C \leq 0, D \leq 0) = 0.5098$, while $Pr(A + C \leq 0, B + D \leq 0) = 0.4922$. □

Corollary 7.1.1. *Suppose* $\mathbf{X}$ *and* $\mathbf{Y}$ *are two vectors of random variables with a joint Gaussian distribution, and*

$$Pr(\mathbf{X} \leq 0) \geq \eta, \text{ and } Pr(\mathbf{Y} \leq 0) \geq \eta,$$

where $\eta \in [0.5, 1)$. *If both* $\mathbf{X}$ *and* $\mathbf{Y}$ *have more than one elements,* $Pr(\mathbf{X} + \mathbf{Y} \leq 0) \geq \eta$ *does not always hold.*

This implies if A is inferior to B, and B is inferior to C, A may not be inferior to C. An important result based on this implication is that the order of the pruning of inferior solutions affects the final solutions. For example, suppose B is inferior to C, and there are k_1 solutions inferior to B, and k_2 solutions inferior to C, then it is possible to have $k_1 > k_2$, thus if B is pruned by C first, those solutions inferior to B but not inferior to C can not be pruned because B is no longer in the solution list. Therefore, in order to get a minimal set of non-inferior solutions, we have to compute the number of solutions dominated by each solution, and keep those solutions that dominate more solutions. Thus, the prune in a solution list with n solutions needs to compute the dominance relation between each pair of solutions, which leads to n^2 time complexity, and a merge procedure that merges two solution lists with m and n solutions needs m^2n^2 computation of the dominance relation.

Another main difficulty in min-cost buffering is its big solution space. During the buffering, the mean value of the downstream capacitance of a statistical solution is always equal to the downstream capacitance of a deterministic solution since only the sum operation is involved in the capacitance computation. But for the delay, since the non-linear multiplication operation is involved, the mean value of the delay of a statistical solution may be different from the delay of the deterministic solution. In addition, because the number of

statistical solutions is always not less than the number of deterministic solutions, we can select a statistical solution that is prone to be better than the deterministic one. Suppose a set of deterministic solutions having the same cost at a node v are (P, D_1, C_1), (P, D_2, C_2), $\ldots$, (P, D_n, C_n), and they satisfy

$$C_1 < C_2 < \cdots < C_n \; and \; D_1 > D_2 > \cdots > D_n.$$

With process variations, the delay and capacitance of this set of solutions become random variables. Suppose their corresponding statistical representations are $\{(P, D_i', C_i') | 1 \leq i \leq n\}$, and then they must satisfy $\mu(C_i') = C_i$. Now if we can find another set of solutions $\{(P, D_i'', C_i'')\}(1 \leq i \leq n)$, such that for any (P, D_i', C_i'), there exist a (P, D_i'', C_i'') such that

$$(\mu(D_i'') \leq \mu(D_i')) \wedge (\mu(C_i'') \leq \mu(C_i') = C_i), \tag{7.20}$$

the yield is expected to be higher. If P is also random, we can pick those solutions satisfying

$$(\mu(P_i'') \leq \mu(P_i')) \wedge (\mu(D_i'') \leq \mu(D_i')) \wedge (\mu(C_i'') \leq \mu(C_i')). \tag{7.21}$$

So this is actually a greedy algorithm: always select the solution with higher probability to be better. As will be demonstrated in our experimental results, the deterministic buffering that fixes parameters at their worst case values gets much worse solutions on average. This is reasonable since the probability that a sample occurs in the region close to the nominal value is much bigger for Gaussian distributions. Therefore, here we select the solutions based on their mean value instead of their $\mu+3\sigma$ value, and the computation of deterministic solutions fixes parameters at their nominal values.

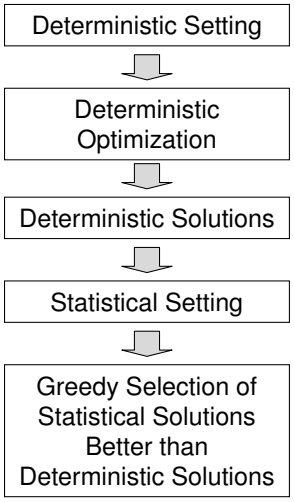


Figure 7.1. General flow of our statistical optimization framework.

The general framework of our statistical optimization approach is shown in Fig. 7.1. Let us see a specific application of the framework on buffering. The deterministic buffering that assumes parameters are at their nominal values is done first to get a set of non-inferior solutions at each node. Then we do the statistical buffering bottom-up from the sinks to the root. The merge of two solution lists with m and n solutions may generate mn new solutions, so we are focusing on the merge procedure. When a merge is encountered, for each deterministic solution K in each branch solution list, we pick a statistical solution with its cost less than or equal to the cost of K, its mean downstream capacitance less than or equal to the downstream capacitance of K, and its mean delay is the minimal[2]. So the number of picked statistical solutions to be merged is close to the number of deterministic non-inferior solutions, which can greatly improve the efficiency of the algorithm. This algorithm is denoted as DL-FSBI. Note that the framework in Fig. 7.1 has many different implementations.

[2]if costs are also random, we select the solution that has the biggest probability that its delay and cost are less than the delay and cost of K, respectively.

For example, the deterministic buffering may fix parameters at other corners, and also the statistical solutions can be selected based on the comparison on the $\mu + 3\sigma$ values.

Note that the pruning is an expensive step since it is at least needed to compute the dominance relations between each pair of non-inferior solutions. One variate of DL-FSBI is to do the solution picking without the pruning. This can greatly improve the efficiency. This variate is denoted as DL-PK-FSBI. Another big difference between DL-PK-FSBI and DL-FSBI is that DL-PK-FSBI picks solutions in the merged solution list, and those solutions that cannot be picked according to Eq.(7.21) are directly discarded, while DL-FSBI picks solutions in the solution lists waiting for merge, and all those newly generated non-inferior solutions are stored in a sorted list. Thus, the number of solutions in DL-PK-FSBI is always much smaller. Our experimental results will demonstrate that the quality of solutions does not degrade much in the efficient DL-PK-FSBI. DL-PK-FSBI-V2 is a variate of DL-PK-FSBI, and it picks solutions based on both their mean values and their $\mu + 3\sigma$ values.

7.3.2. Slew constrained buffering

The rising or falling time of a signal transition is its slew rate. We use the slew model in [**56**]. Let S_u represent the slew rate at node u. If a buffer b is attached to the node u, the slew rate at u's downstream node v is $S_v = \sqrt{S_{u,out}^2 + S_p^2}$, where $S_{u,out}$ is the output slew rate of the buffer, and S_p is the slew degradation along the path p from u to v. According to Bakoglu's metric [**9**], $S_p = (\ln 9)D_p$, where D_p is the Elmore delay of path p.

Hu *et al.* [**51**] proposes a deterministic slew constrained buffering approach that assumes the input slews of buffers are fixed at conservative values, and gets $S_{u,out} = P_b C_u + Q_b$, where P_b and Q_b are empirical fitting parameters. This assumption is also used in our approach

for simplicity, and our approach can be easily extended to handle the cases without this assumption. Let (P_v, S_v, C_v) represent a solution at node v. We prove the following theorem.

Theorem 7.2. *If the input slews of buffers are fixed at conservative values, there is only one type of buffer, and the input capacitances of buffers are no greater than sink capacitances, the costs of the non-inferior solutions at a potential buffering node have at most two distinct values for the buffer area minimization problem.*

Proof. One of the non-inferior solutions at the potential buffering node must have its downstream capacitance equal to the minimal input capacitance of buffers and its slew equal to 0 because this solution has the minimal slew and the minimal downstream capacitance. Suppose this solution is (P_k, S_k, C_k). This solution must have the biggest cost among those non-inferior solutions; otherwise it dominates some other solutions. Now suppose those non-inferior solutions have more than two distinct costs, then there exists a solution (P'_k, S'_k, C'_k) with its cost less than $P_k - 1$. If (P'_k, S'_k, C'_k) already have a buffer at the node, it is obvious that it dominates (P_k, S_k, C_k).

If (P'_k, S'_k, C'_k) does not have a buffer at the node, we can put a buffer with the minimal input capacitance at the node to get a new solution, and this solution dominates the (P_k, S_k, C_k) since its cost is less. □

With process variations, the delays and the downstream capacitance become random variables, so do the slew rates. Assuming that delays and downstream capacitance have Gaussian distributions, we know that path slew degradations S_p's also have Gaussian distributions. Let $D_p = D_0 + \sum_{i=1}^{n} D_i\epsilon_i + D_{n+1}R_{D_p}$, and $C_v = C_0 + \sum_{i=1}^{n} C_i\epsilon_i + C_{n+1}R_{C_v}$, then $S_p = \ln(9)D_0 + \sum_{i=1}^{n} \ln(9)D_i\epsilon_i + \ln(9)D_{n+1}R_{D_p}$, and $S_{u,out} = (P_bC_0 + Q_b) + \sum_{i=1}^{n} P_bC_i\epsilon_i + P_bC_{n+1}R_{C_v}$. Here, we assume that P_b and Q_b are deterministic values for simplicity. If they

also have process variations, the moment-matching approach in Section 7.2 can be used here to get a canonical form for $S_{u,out}$.

We use the same dynamic programming framework as in [**51**]. We traverse the tree bottom-up from sinks to the root, and at each node, we prune those inferior solutions. Since we are considering the slew instead of the delay, the inferior condition becomes:

Definition 7.1. (C_{v1}, S_{v1}, P_{v1}) *is* **inferior** *iff there exists another solution* (C_{v2}, S_{v2}, P_{v2}) *satisfying*

$$Pr(C_{v2} \leq C_{v1}, S_{v2} \leq S_{v1}, P_{v2} \leq P_{v1}) \geq \eta,$$

at the same node, where η *is a given constant real number in* $[0.5, 1]$.

When a buffer is attached to a node, we need to compute the probability that the output slew rate satisfies the slew constraints. If it does not satisfy the constraint, the solution needs to be discarded. The square root computation of random variables is involved in the computation S_v, which is not convenient for computation. So we compute S_v^2 instead of S_v.

Let Sq_v represent the S_v^2. Then according to $E(A^2) = E(A)^2 + \sigma(A)^2$, we have

$$\begin{aligned} E(Sq_v) &= ((P_b C_0 + Q_b)^2 + \sum_{i=1}^{n+1} P_b^2 C_i^2) + (\ln(9) D_0)^2 \\ &\quad + \sum_{i=1}^{n+1} (\ln(9) D_i)^2 \end{aligned}$$

Also

$$E(Sq_v^2) = E(S_{u,out}^4) + E(S_p^4) + 2E(S_{uout}^2 S_p^2).$$

Suppose X has a Gaussian distribution with mean μ and sigma σ, then the 4th raw moment of X is $E(X^4) = \mu^4 + 6\mu^2\sigma^2 + 3\sigma^4$. Suppose $X = x_0 + \sum_{i=1}^{n} x_i \epsilon_i + x_{n+1} R_x$, and $Y =$

$y_0 + \sum_{i=1}^{n} y_i\epsilon_i + y_{n+1}R_y$. Then

$$\begin{aligned} E(X^2Y^2) &= x_0^2y_0^2 + x_0^2\sigma_Y^2 + y_0^2\sigma_X^2 + 2\sum_i x_i^2y_i^2 \\ &\quad +\sigma_X^2\sigma_Y^2 + 4x_0y_0cov(X,Y) \\ &\quad +4\sum_{i=1,j=i+1,i\neq j}^{i=n-1,j=n} x_iy_ix_jy_j \end{aligned}$$

Now we get $E(Sq_v^2)$ and $E(Sq_v)$, and then $\sigma^2(Sq_v) = E(Sq_v^2) - E(Sq_v)^2$.

It is not necessary for the Sq_v to be represented in a canonical form since it is needed only when a buffer is inserted, and is never propagated. Once we know the mean and the variance of the Sq_v, assuming that Sq_v has a Gaussian distribution, we compute the probability that the slew constraint is satisfied and discard those solutions that violate the slew constraint. Our experiments demonstrate that this approximation is accurate enough for the estimation of the yield. This slew constrained min-cost buffering algorithm is called SW-FSBI.

Our approach can be easily extended to handle the cases where the output slew of a buffer is specified by a 2D look-up table (LUT): change the deterministic comparison in the approach of [**51**] to statistical comparison. For example, if the original comparison is $A \leq B$, we change it to $Pr(A \leq B) \geq \eta$, where η is a given value between 0.5 and 1.

7.3.3. Analysis

We observed that the delay constrained buffering is much less sensitive to process variations than the slew constrained buffering: it often occurs that the statistical delay constrained buffering gets the exactly same solution as the deterministic delay constrained nominal buffering, while the slew constrained buffering gets much better results. The merge of two solution lists with m solutions and n solutions respectively may generate mn new solutions

in statistical situation. But if the solutions in a solution list always have bigger delays than those in solution lists of other branches, the merged solutions always have the same (or similar) delays with those solutions in the dominating solution list, and thus, the number of merged solutions is much less than mn, which is not good for the statistical buffering. For whole circuit buffering, this situation often occurs since the criticalities of wires in different branches are often much different. In addition, most nets in a circuit are small, so the merge operation seldom occurs, and thus the number of newly generated solutions is not big, which makes the statistical delay-constrained buffering behave not much better either. Therefore, we can use the fast deterministic buffering as a pre-processing step to check if it is necessary to do the statistical delay-constrained buffering. Slew constrained buffering is a local optimization problem, and some works just used the wire length to do slew constrained buffering at the early stage. So slew constrained buffering is more sensitive to variations.

7.4. Min-cost buffering on a combinational circuit

The assignment of various timing budgets on less-critical paths in a combinational circuit would give multiple buffering solutions, among which we need to select the one with the minimal costs. Sze *et al.* [**95**] proposed a deterministic path-based buffering technique, where the circuit is partitioned into trees, and Lillis' min-cost buffering algorithm is used to buffer each tree. It is straight forward to extend that algorithm to consider process variations: our statistical min-cost delay constrained buffering algorithm is used to buffer each tree. But as shown in [**104**], the solutions from [**95**] have much more buffers than those from [**104**].

Our idea is that the deterministic optimization techniques are used to get a decent solution, and a statistical refinement is done to get a better solution. This algorithm is denoted

as CC-FSBI. The first step is the deterministic buffering of the circuit. Here as mentioned before, parameters use their nominal values. The whole circuit buffering techniques in [**23**,**104**] can be used here. The second step is the statistical refinement: we use DL-FSBI to buffer each net. Here the required times and the load capacitances of sinks use those deterministic values computed by the first step.

The second step of CC-FSBI buffers each net, and does not re-budget the timing between nets. If a gate has only one input, we can buffer the combination of its fanin net and its fanout net as a whole tree by viewing the gate as an existing buffer. If a gate has more than one inputs, we do not combine them, so the path-reconvergence problem as mentioned in [**95**] is avoided. A variate of CC-FSBI is to use this technique to re-budget the timing between neighbor nets.

7.5. Experimental results

7.5.1. Delay minimization

The delay minimization algorithm called FSBI is implemented in C++. We use the heuristic II that is claimed to be the best in the three heuristics in [**58**], and the approach in [**109**] for comparison. [**37**] also uses an expensive two-phase merge strategy, and the algorithm in [**110**] is shown to be very slow in [**109**], so we do not compare them with ours.

FSBI is tested on all the test cases from [**60**]. The characteristics of these test cases are shown in Table 7.1. For the nets with small number of sinks (e.g., the data nets), the existing approaches can be used. So we test our approach on only those nets with big number of sinks. Since the original test cases in [**60**] do not have the statistical information (e.g., deviation, correlation), we generate the statistical information by ourselves. For each random variable representing the d_b, the resistance, or the capacitance, we randomly generate the

coefficients of the ϵ_i's in the canonical form, and enforce that each random variable has 10% deviation from its nominal value. Note that although the test cases in [**109**] are also derived from the test cases in [**60**], since we cannot get those test cases with statistical information from [**109**], our test cases are different from those in [**109**]. $\eta = 0.90$ in the prune rule. All the experiments were run on a Linux PC with 2.4 GHz Xeon CPU and 2.0 GB memory.

Table 7.1. The characteristics of the test cases

name	# sinks	# nodes	# buffer locations
p1	269	537	268
p2	603	1205	602
r1	267	533	266
r2	598	1195	597
r3	862	1723	861
r4	1903	3805	1902
r5	3101	6201	3100

The heuristic II in [**58**] completes the merge procedure as follows. First, it merges each pair of solutions from left branch and right branch respectively, and then computes the prune probability between each pair of solutions in the new solution lists. A graph with the vertices representing the solutions and the edges representing the pruning relations between solutions is constructed. Then the vertex with the maximal out-degree is iteratively selected into the set that stores the vertices representing non-inferior solutions, and all the vertices that have edges from the selected vertex are deleted from the graph.

The comparison results of FSBI, [**58**], [**109**] and the deterministic buffering in nominal scenario are shown in Table 7.2. The solution with the minimal mean value of D (if tied, the solution with the minimal variance) is selected as the final solution. The timing constraints are randomly selected such that our algorithm and the other algorithms have the yields in the reasonable range [60%, 100%]. The approximation of the multiplication of Gaussian variables as a Gaussian variable may have 10% errors on the PDF [**109**], so we use Monte

Table 7.2. Comparison results of FSBI, [**58**], [**109**] and deterministic buffering

Nets		Van-Ginneken (Nominal)		[**58**]		[**109**]	FSBI			Gain
	Delay constr	Buffer #	Yield (%)	Time (s)	Yield (%)	Yield (%)	Buffer #	Time (s)	Yield (%)	(%)
p1	805	162	62.88	N/A	N/A	63.88	158	10.22	75.28	12.40
p2	2030	268	79.21	N/A	N/A	73.60	268	27.92	94.37	15.16
r1	335	166	62.60	198.75	77.81	59.10	169	5.26	79.38	16.78
r2	454	358	67.03	N/A	N/A	62.90	363	17.53	79.49	12.46
r3	620	517	81.30	N/A	N/A	79.30	523	14.20	92.48	11.18
r4	900	1187	78.56	N/A	N/A	79.26	1192	52.67	87.26	8.70
r5	1080	1893	70.65	N/A	N/A	71.03	1918	76.63	80.36	9.71
ave										12.34

Carlo simulation to compute the yield for the selected solution. Column 2 shows the timing constraints, Column 3 and 4 show the number of buffers and the yield, respectively, computed by the van Ginneken's algorithm in nominal scenario, Column 5 and 6 show the running time and the yield, respectively, computed by [**58**], and Column 7 shows the yield computed by [**109**]. Column 8, 9 and 10 show the number of buffers, the running time and the yield from FSBI, respectively. Column 11 shows the yield improvement of FSBI compared with the deterministic buffering. The "N/A" in the table means that the algorithm cannot finish the test case because of the memory constraint (2GB) or time limit (3 hours).

On average, FSBI achieves 12.34% improvement on the yield compared with the deterministic buffering. The yields from FSBI are always higher than those from the deterministic buffering for all these cases. For example, the FSBI achieves 16.78% yield gain for test case "r1". We also implemented an approach that prunes the solutions according to only the prune rule, so this approach keeps all the "non-inferior" solutions, and thus gives us a good estimation of the optimal yield. It can finish only the case "r1" because of the limitation of the memory. The computed yield of "r1" is 79.47%, which is quite close to the yield from FSBI. So the yield is not sacrificed much in FSBI. The results demonstrate that the statistical buffering is still needed for multi-pin nets, and it can use the information provided

by the deterministic buffering to achieve higher yields. The yields computed by [**109**] are higher than those computed by the deterministic buffering that assumes the worst situation (these results are shown in Table 7.2), which is consistent with the conclusion in [**109**]. But in general, they are not much higher, sometimes even lower, than the yields computed by the deterministic buffering that assumes the nominal situation. The FSBI always achieves much higher yields than the approach in [**109**]. The results also indicate that the FSBI is very efficient. It takes only 76.63 seconds for FSBI to finish the largest case "r5". While [**58**] cannot finish most of the test cases because of the memory constraint (2GB) or time limit (3 hours). Although the approach in [**109**] is efficient, we do not report its running time because of the quality of its solutions.

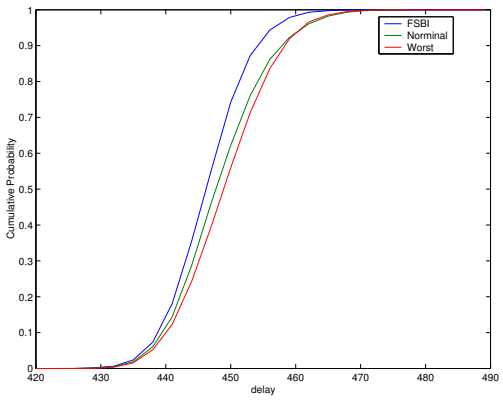


Figure 7.2. Comparison of solutions for "r2".

Through Monte Carlo simulation, Fig. 7.2 shows the cumulative distribution functions (CDF) of the maximal delays from the root to sinks for the final solutions from those algorithms. The "Nomina" curve and the "Worst" curve represent the CDFs of the solutions from the deterministic Van-Ginneken's algorithm that assumes all the parameters are at their nominal values or their $\mu + 3\sigma$ values, respectively. The curve representing the distribution from FSBI is obviously pushed to the left side, so FSBI gets a higher yield.

7.5.2. Cost minimization

DLC-FSBI, SWC-FSBI and CC-FSBI are implemented in C++ and tested on a set of big nets and combinational circuits. In the statistical buffering, process parameters are set to have at most 30% deviation ($3\sigma/\mu$) from their nominal values. We also implemented a deterministic delay-constrained buffering approach, denoted as DL-DETBI, that uses Lillis's algorithm and Shi's speed-up techniques [**90**] for delay-constrained min-cost buffering, and the deterministic slew-constrained min-cost approach [**51**], denoted as SW-DETBI, for comparison. In DL-DETBI and SW-DETBI, all the process parameters are fixed at their nominal values or their $\mu + 3\sigma$ values. All the experiments were run on a Linux Redhat machine with a 2.4 GHz Xeon CPU and 2.0 GB memory.

7.5.2.1. Delay constrained buffering. We also implemented a statistical buffering technique that does the pruning only according to the optimality condition, and does not do the greedy selection of statistical solutions. This is denoted as SBI. Therefore, we know if the quality of solutions degrades in DL-FSBI when we compare DL-FSBI and SBI. We tested DL-FSBI and SBI on those nets in the circuits from [**95**]. There are four types of buffers. These nets are generally very small (most of them have less than 4 sinks), so SBI can finish them in a short time. But we did not see any degradation on the yield for DL-FSBI on this set of nets.

For delay constrained buffering, we also compare the yields from DL-DETBI and DL-FSBI for each cost value. There is one type of buffer. Elmore delay model is used for the delay computation. DL-DETBI has two variates: DL-NOMBI has all the parameters at their nominal values, and DL-WSTBI has all the parameters at their worst case ($\mu + 3\sigma$) values. For each cost value, we set the delay constraint to the sum of the minimal delay

achieved by DL-NOMBI and its standard deviation. So the yield from DL-NOMBI is always close to 84.13%. Table 7.3 shows the comparison results. Note that SBI cannot finish any of these testcases in one day, so DL-FSBI is much more efficient than SBI. A good property of our approach is that the efficiency of DL-FSBI also depends on the number of non-inferior solutions in the deterministic buffering step, so with the help of existing speed-up techniques on deterministic buffering (e.g., sampling), DL-FSBI can achieve much better efficiency. The "# cost" column shows the total number of distinct costs; the "# diff" column shows the number of cost values on which the DL-FSBI/DL-PK-FSBI has more than 4% improvement on the yield compared with DL-NOMBI; the "Avg (%)" columns show the average improvement on the yields among those cost values where the DL-FSBI/DL-WSTBI has more than 4% improvement/degradation; the "Max (%)" column show the maximal improvement on the yields among all the cost values. Actually, we do not see any case where the DL-FSBI has yield degradation. Note that all the yields data here have been verified by Monte-Carlo simulation. The results indicate that the DL-FSBI can achieve big yield improvement (e.g., p1 has 15.87% maximal improvement) on a set of cost values. The DL-WSTBI always gets much worse solutions than DL-NOMBI. DL-PK-FSBI is much more efficient than DL-FSBI, and has only 8.99% runtime overhead on average compared with DL-NOMBI. So the statistical buffering step in DL-PK-FSBI is extremely fast. DL-PK-FSBI has only 6% degradation on the "# diff" values on average compared with DL-FSBI. Those statistical solutions better than deterministic solutions with respect to $\mu+3\sigma$ are also picked in DL-PK-FSBI-V2, and the degradation becomes even less. For example, the "# diff"s in "r2" and "r3" are both improved to 23.

On the other hand, comparing those values in the "# diff" column and the "# cost" column, we see that DL-FSBI can only improve the yield on a small set (max 15/131=11%)

of cost values. For "p2", the statistical buffering does not have any big improvement. Deng *et al.* [**40**] analyzed the buffering on a two-pin net and showed that it is not necessary to consider process variations based on some ideal conditions (e.g., no blockage). Alpert *et al.* [**6**] also showed that the density of wire segments does not change the solutions greatly. From our experiments, we see similar things.

Table 7.3. Delay constrained min-cost statistical buffering vs. deterministic buffering on a net for Elmore delay model

Nets	DL-NOMBI		DL-WSTBI	DL-FSBI				DL-PK-FSBI		DL-PK-FSBI-V2
	cost #	time (s)	avg (%)	# diff	max (%)	avg (%)	time (s)	# diff	time (s)	# diff
p1	131	35.31	-52.53	15	15.87	13.93	384.30	15	38.81	15
p2	164	251.81	-6.43	0	0	0	3861.59	0	272.17	0
r1	212	168.76	-20.79	9	9.19	5.54	2167.96	9	185.60	9
r2	374	1711.26	-26.63	24	14.50	5.65	23367.98	21	1801.77	23
r3	359	1607.01	-16.65	24	14.00	5.99	33149.76	20	1794.67	23

We also tested our approach on the much more accurate delay model D2M. The comparison results are shown in Table 7.4. The results indicate that DL-PK-FSBI-V2 achieves 8.51% improvement on timing yield on average.

Table 7.4. Delay constrained min-cost statistical buffering vs. deterministic buffering on a net for D2M delay model

Nets	DL-NOMBI		DL-WSTBI	DL-PK-FSBI-V2		
	#cost	time(s)	avg(%)	#diff	avg(%)	time(s)
p1	44	11.07	-8.49	9	11.06	19.04
p2	87	770.01	-12.89	24	11.75	877.12
r1	127	375.62	-32.59	35	3.59	413.21
r2	168	530.02	-25.36	52	7.51	588.35
r3	142	736.38	0.00	32	8.63	813.16

7.5.2.2. Slew constrained buffering. For the slew constrained min-cost buffering, we compare the yields from SW-FSBI and SW-DETBI [**51**]. The comparison results are shown in Table 7.5. The "Max" and "Min" columns show the maximal and the minimal number of inserted buffers, respectively. The objective yield in SW-FSBI is set to 97%. Here, we

Table 7.5. Slew constrained min-cost statistical buffering vs. deterministic buffering on a net

Nets		Nominal				Worst			SW-FSBI			
Name	Slew (ps)	Max	Min	Yield (%)	Time (s)	Max	Min	Yield (%)	Max	Min	Yield (%)	Time (s)
p1	600	57	55	54.8	0.01	57	55	100	57	55	100	0.03
	300	113	111	39.9	0.01	127	125	100	122	120	100	0.02
	200	191	189	38.7	0.01	219	217	100	211	209	100	0.02
p2	600	127	125	43.85	0.03	134	132	100	133	131	97.75	0.05
	300	264	262	48.31	0.03	283	281	100	270	268	99.04	0.05
	200	445	443	46.64	0.03	494	492	100	457	455	99.87	0.05
r1	1000	63	61	60.63	0.02	78	76	100	64	62	96.69	0.03
	800	75	73	58.00	0.02	87	85	100	78	76	97.26	0.03
	600	88	86	40.59	0.02	N/A	N/A	N/A	89	87	97.64	0.03
r2	1300	96	94	28.53	0.05	118	116	100	98	96	98.57	0.08
	1100	109	107	38.16	0.04	135	133	100	111	109	99.70	0.07
	1000	117	115	43.28	0.04	N/A	N/A	N/A	119	117	99.82	0.06
r3	1000	145	143	24.66	0.05	178	176	100	151	149	98.45	0.08
	800	170	168	32.45	0.05	211	209	100	176	174	98.95	0.08
	500	242	240	35.26	0.05	293	291	100	246	244	95.93	0.08

also used Monte-Carlo simulation to compute the yield for each solution, and observed that the computation of the yield using the proposed approximation technique is very accurate (difference is less than 1% on average). The results indicate that the SW-FSBI achieves the objective yield on all the cases except the last one, and achieves 56% yield improvement with 3.3% cost overhead on average compared with the deterministic buffering that uses nominal parameter values. The deterministic buffering that uses the worst case values achieves 100% yield on most cases, but it cannot get feasible solutions for two cases where the slew constraints are tight. In addition, it has 11.8% cost overhead on average compared with the SW-FSBI. The SW-FSBI is also efficient since the slew constrained buffering does not have many non-inferior solutions. In summary, the consideration of process variations on slew constrained min-cost buffering can greatly improve the yield.

7.5.2.3. Circuit buffering. We test CC-FSBI on a set of testcases from [**95**]. The nets in these testcases are very small, and most of those nets have only one sink. There are four types of buffers. The deterministic buffering approach called CC-DETBI works as follows:

the whole circuit buffering approach in [**23**] is done; then in order to improve the yield, the net buffering approach in [**90**] is used to buffer each net such that the maximal delay from the source to sinks is minimized while the total buffer area is also within a specified range. The comparison results on a testcase "a1" are shown in Table 7.6. Column "Cost" shows the cost from CC-FSBI over the cost from CC-DETBI. CC-FSBI does not have big improvement on the yield, and its reduced cost is also not very prominent. The other cases have similar results, which are omitted because of the space limit. Therefore, in general, a deterministic buffering approach is accurate enough to handle the delay constrained buffering of those circuits where most nets are small.

Table 7.6. Delay constrained min-cost statistical buffering vs. deterministic buffering on a combinational circuit

Circuits		CC-DETBI		CC-FSBI		
	Delay	Yield	Time (s)	Cost	Yield	Time (s)
a1	291 ps	83.52%	162.93	99.86%	85.44%	289.27

7.6. Conclusion

With the consideration of process variations, we proposed effective approaches to handle the delay minimization, delay constrained min-cost buffer insertion and the slew constrained min-cost buffer insertion on a net, and also proposed an approach to handle the combinational circuit buffering problem. We observed that process variations do not have great impact on the delay constrained buffering, especially for small nets, but they do have great impact on the slew constrained buffering that is mostly used in the current industry buffering practice [**74**].

CHAPTER 8

Timing Macro-Modeling of IP Blocks with Crosstalk

With the progress of deep sub-micron technologies, shrinking geometries have led to a reduction in self-capacitance of wires. Meanwhile coupling capacitances have increased as wires have a larger aspect ratio and are brought closer together. For present day processes, the coupling capacitance can be as large as the sum of the area capacitance and the fringing capacitance, and the trends indicate that the role of coupling capacitance will be even more dominant in the future as feature sizes shrink. This makes crosstalk a major problem in IC design. Crosstalk introduces noise between adjacent wires, and even alters the functions of circuits. If an aggressor and a victim switch simultaneously on the same direction, the victim will speed up. Likewise, if an aggressor and a victim switch on the opposite directions, the victim will slow down.

With the growing complexity of VLSI systems, the intellectual property (IP) reuse is becoming a common practice. There are previous researches dealing with the timing macro-modeling of IP blocks, and all of them are based on the concept of path delay [**36**, **43**, **69**, **102**, **111**, **112**]. The simplest of such models is the "pin-to-pin" delay model used to characterize standard cells and other complex combinational circuits. For example, given a simple combinational circuit shown in Figure 8.1(a), its "pin-to-pin" delay model is shown in Figure 8.1(b). The numbers shown on each arc give the minimal and maximal delays from one pin to another. In this case, if $a(x)$ and $A(x)$ are used to represent the earliest and

latest arrival time of signal x, respectively, we have

$$a(x) = a(a) + 3; A(x) = A(a) + 5;$$

$$a(y) = min(a(b) + 2, a(c) + 3); A(y) = max(A(b) + 4, A(c) + 4).$$

To model a sequential circuit with memory elements, the "pin-to-pin" model is extended to include timing constraints from a clock pin to an input pin (to model setup and hold conditions) and delay arcs from a clock pin to an output pin (to model the latch output to circuit output delay) [**36**]. Functionality may also be used to reduce the pessimism in these models [**111**, **112**].

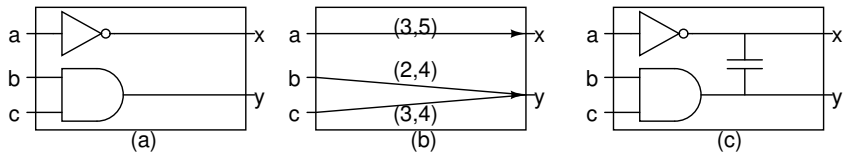


Figure 8.1. (a) A simple circuit; (b) Its "pin-to-pin" macro-model; (c) Coupling destroys path delay concept.

Unfortunately, coupling totally destroys the path delay concept. For example, as shown in Figure 8.1(c), if the wires x and y are coupled to each other, the arrival time on x and y cannot be decided through paths from the inputs. Instead, the relative switching time of a, b, c is important in deciding the arrival time on x and y. Suppose when

$$a(a) = A(a) = 0, a(b) = A(b) = 1, a(c) = A(c) = 0,$$

we have the maximal delay variation on the coupled signals x and y such that

$$(a(x), A(x)) = (2, 10), (a(y), A(y)) = (1, 11).$$

Then for any t such that

$$a(a) = A(a) = t, a(b) = A(b) = t + 1, a(c) = A(c) = t,$$

the delay variation will also be maximal and we have

$$(a(x), A(x)) = (t + 2, t + 10), (a(y), A(y)) = (t + 1, t + 11).$$

This means that the delays between pins are influenced by the relative arrival time of inputs. So the conventional "pin-to-pin" macro-model for IP blocks is pessimistic.

Many researches have been done on timing analysis or modeling considering crosstalk [**8**, **18**, **19**, **47**, **80**, **99**, **100**, **107**, **108**, **116**]. Most of them consider how to compute the delays of a set of coupled nets given their input time, so the resulting delays are accurate only for a specific input arrival time combination. Sasaki et al. [**81**, **82**] proposed relative window methods, and Agarwal et al. [**4**] proposed an analytical method, to depict the delay change curves for noise-aware static timing analysis (STA). However, they can only handle the simple cases with one victim and multiple aggressors.

So when considering crosstalk effects, all the current macro-modeling methods cannot generate an *accurate* model for a *complex* IP block. We propose two input-dependent models for specifying the timing behaviors of complex IP blocks. To the best of our knowledge, it is the first work dealing with timing characterization of IP blocks with crosstalk effects.

The rest of the section is organized as follows. Section 8.1 shows the requirements of a viable macro-model. Section 8.2 presents the extraction and application of SIMPMODEL. This fast gray-box macro-model computes and utilizes the conditions on the relative input

arrival time combinations for couplings not to take effect. Section 8.3 present the extraction and application of UNIONMODEL. This accurate black-box model stores the output response windows for a basic set of relative input arrival time combinations, and computes the output arrival time for any given input arrival time combination through the union of some stored output response combinations. Section 8.4 reports the experimental results on the proposed macro-models and their comparison with "pin-to-pin" model and STA results. Finally, the conclusion is discussed in Section 8.5.

8.1. Macro-model requirements

A viable timing macro-model of combinational hard IP block should satisfy the following requirements:

- Hiding of implementation details. It is a requirement for protecting intellectual property.
- Model accuracy. Given an input arrival time combination, the arrival time window of any primary output generated by the timing macro-model should be as close as possible to the corresponding actual output arrival time window.
- Conservative modeling. Given an input arrival time combination, the actual time window of any primary output should be in the range of the corresponding output time window generated by the timing macro-model.

When crosstalk effects are considered, the delay between pins depends not only on the structural detail of IP blocks, but also on the relative arrival time of primary inputs, or *input patterns*.

Definition 8.1 (input/output pattern)**.** *An input/output pattern of an IP block is an arrival time window vector where each element is an arrival time window on one primary input/output of the IP block.*

So an *input-dependent* model may greatly improve the accuracy. As is said before, when considering crosstalk effects, all the current macro-modeling methods *cannot* generate an *accurate* model for a *complex* IP block. We introduce two input-dependent macro-models that satisfy all the requirements.

8.2. SIMPMODEL

8.2.1. Delay model

The conventional "pin-to-pin" model assumed that the coupling effects were always active, which led to a very pessimistic estimation. In order to get a more accurate model, we compute and utilize the conditions under which the couplings do not take effect.

Hassoun [**49**] presented a dynamically bounded delay model to represent the delay of a net. We use a modified dynamically bounded delay model. A directed graph $G = (V, E, C)$ represents the IP block. Each vertex in V represents a primary input, a primary output, a gate, or a net. A connection connecting inputs, outputs, gates or nets is represented by one edge in E. Let set C_v be the set of aggressor nodes connected via a coupling capacitor to victim node v, W_v be the timing window of node v, and

$$C = \cup_{v \in V} \{C_v\}$$

Each node v has a dynamically bounded delay model consisting of a fixed delay range $[\delta_v, \Delta_v]$, and, for each coupling capacitor attached to v from an aggressor node a, a predicate $\gamma_{v,a}$

indicating whether the coupling takes effect. If the coupling does not take effect, the minimal delay should be increased by $\delta_{v,a}$, and the maximal delay should be decreased by $\Delta_{v,a}$. So the minimal delay of node v is

$$\delta_v + \sum_{a \in C_v} \gamma_{v,a} \delta_{v,a},$$

and the maximal delay of node v is

$$\Delta_v - \sum_{a \in C_v} \gamma_{v,a} \Delta_{v,a},$$

where $\gamma_{v,a}$ is defined as

$$\gamma_{v,a} = \begin{cases} 0 & \text{if } W_v \text{ overlaps with } W_a, \\ 1 & \text{otherwise.} \end{cases}$$

8.2.2. SIMPMODEL details

First, the SIMPMODEL reduces the original complex directed acyclic graph (DAG) G to a DAG called *coupling graph* that contains only the input pins, the output pins, and the nodes with coupling. Each edge in this coupling graph represents that the end node is reachable from the start node through a path without extra coupling nodes, and the weights of each edge represent the minimal and the maximal delays from the start node to the end node. This task can be accurately done by any conventional STA.

The remaining task is to determine the value of $\gamma_{v,a}$. When considering crosstalk effects, the determination of $\gamma_{v,a}$ becomes a chicken-and-egg problem, and the conventional way is to use iteration methods. With the coupling graph, the macro-model users can use STA tools to get the output pattern directly. Since the coupling relations are often complex,

this kind of macro-model leaves much work to users. Instead, we design a fast conservative approximation method to determine the value of $\gamma_{v,a}$.

It is obvious that the value of $\gamma_{v,a}$ is determined by the relations among the time windows of primary inputs that can reach v or a. Let set I_v be the set of primary inputs that can reach node v, v_i be the ith input that can reach node v, b_{v_i} and w_{v_i} be the minimal and maximal delays from input v_i to v respectively, $[x_{v_i}, y_{v_i}]$ is the arrival time window of input v_i. Then we have these two rules:

$$\min_{v_i \in I_v}(x_{v_i} + b_{v_i}) > \max_{a_j \in I_a}(y_{a_j} + w_{a_j}) \Rightarrow \gamma_{v,a} = 1,$$

and

$$\max_{v_i \in I_v}(y_{v_i} + w_{v_i}) < \min_{a_j \in I_a}(x_{a_j} + b_{a_j}) \Rightarrow \gamma_{v,a} = 1.$$

After getting the coupling graph, SIMPMODEL assumes that $\gamma_{v,a} = 0$ for all the coupling nodes. Then do a PERT traversal on the coupling graph to calculate the minimal and maximal delays from each primary input to each node, and put the results into a list L. Till now, a model composed by a coupling graph and a result list is extracted successfully.

Then during the model application, once the input pattern is given, we can check the two rules to determine the value of each $\gamma_{v,a}$ according to the information in the list L. Once the delay of each node is determined, we traverse the coupling graph to get the desired output pattern.

We assume that $\gamma_{v,a} = 0$ for all the coupling nodes, so after checking these two rules, if $\gamma_{v,a} = 0$ for a pair of coupling nodes v and a, it maybe equal to 1 in reality, but if $\gamma_{v,a} = 1$ for a pair of coupling nodes v and a, it must be 1 in reality, so SIMPMODEL always achieves a conservative output pattern for a given input pattern.

The coupling graph unveils many implementation details when the coupling relations are very complex. Since macro-models need to hide the implementation details as many as possible, we need to simplify the coupling graph. We use the following two techniques to achieve that.

The first technique is pattern replacement, that is, some structures are replaced by simplified structures without much sacrifice of the accuracy of the models. For example, as shown in Figure 8.2, we can replace the structure in Figure 8.2(a) by that in Figure 8.2(b).

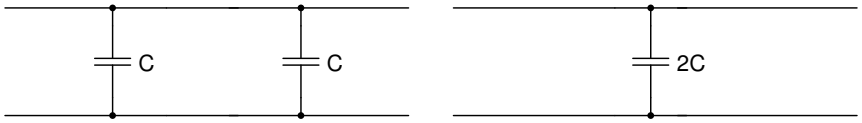


Figure 8.2. The original structure (a) and its structure after pattern replacement (b).

The second technique is coupling pruning, that is, if some coupling relations are much weaker than the others, they are directly deleted from the coupling graph. As shown in Figure 8.3(a), the coupling relation on the left side is much weaker than that on the right side, so it is deleted (shown in Figure 8.3(b).

Figure 8.3. The original structure (a) and its structure after coupling pruning (b).

8.3. UNIONMODEL

SIMPMODEL shows a method to fast estimate the output pattern, while its accuracy is sacrificed to get a high speed, and SIMPMODEL is a gray-box model, that is, it unveils

part of the implementation details of IP blocks. However, the accuracy of the model and the hiding of implementation details are strongly required in many situations. The following UNIONMODEL satisfies these two requirements, that is, UNIONMODEL is an accurate black-box timing macro-model.

Let A_i represent the time window of pattern A corresponding to pin i. First, we introduce some definitions.

Definition 8.2 (adjacent windows)**.** *If two windows a and b overlap with each other at only one common point, then a and b are adjacent, denoted as $a \diamond b$.*

Definition 8.3 (combinable input patterns)**.** *If the windows corresponding to the same pin in two patterns A and B are adjacent, and the other pairs of windows corresponding to the same pins are the same respectively, then A and B are combinable, denoted as $A \simeq B$*

Definition 8.4 (subwindow)**.** *If window a is contained in window b, then a is a subwindow of b, denoted as $a \subseteq b$.*

Definition 8.5 (subpattern)**.** *Let A and B be two patterns. If for any i, $A_i \subseteq B_i$, then A is a subpattern of B, denoted as $A \subseteq B$.*

Definition 8.6 (overlapped patterns)**.** *If each window in pattern A overlaps with the corresponding window in pattern B, then A and B are overlapped patterns, denoted as $A \cap B \neq \phi$.*

Definition 8.7 (union of patterns)**.** *The union of two overlapped patterns A and B, denoted as $A \cup B$, is a pattern where each window is the union of the two corresponding windows in A and B.*

For example, in Figure 8.4, time windows a, b, c, d are $[0, 10]$, $[10, 20]$, $[0, 10]$ and $[10, 20]$, respectively. Input patterns $A\{a, c\}$ and $B\{b, c\}$ are combinable input patterns that can be united into an input pattern $\{[0, 20], c\}$. But input pattern $\{a, c\}$ and $\{b, d\}$ are not combinable, because a and b are adjacent, and c and d are adjacent but not the same.

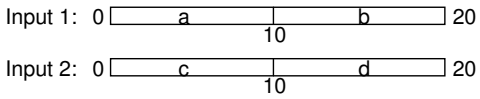


Figure 8.4. An example of input patterns.

It is obvious that combinable input patterns are also overlapped patterns. Combinable input patterns have another important property:

Lemma 8.1. *Suppose A and B are input patterns for an IP block, if $A \simeq B$, let input pattern $X = A \cup B$, then the output pattern for X is the union of the output patterns for A or B.*

For example, in Figure 8.4, the output window for input pattern {[0,20],[0,10]} is the same with the union of output patterns for input patterns {[0,10],[0,10]} and {[10,20],[0,10]}. UNIONMODEL uses this property to model the timing behavior of combinational IP blocks.

8.3.1. Model extraction

First, a wide range input pattern $\mathcal{P}$ is generated. Since sequential circuits dominate the reality, and combinational parts are embedded in sequential circuits, the arrival time of primary inputs of a combinational circuit is between 0 and T, where T is the clock period of the sequential circuit. We can make a reasonable assumption that T is upper-bounded by a value denoted as T_{max}, then we choose the arrival time window $[0, T_{max}]$ as $\mathcal{P}_i$, $i = 1, \ldots, n$, where n is the number of primary inputs.

Then, each window in $\mathcal{P}$ is evenly partitioned into k parts, where k is a positive integer. For each primary input, we pick one part from the corresponding k parts as an input window in the resulting small range input pattern, then we can construct k^n distinct input patterns, called *basic patterns*.

Performing STA or simulations on the IP block, we can get the output pattern for each basic pattern. The results are put into a table T: each entry corresponds to one basic pattern and its output pattern. This table T provides all the required information for UNIONMODEL application.

8.3.2. Model application

Given any input pattern I, the windows in I are shifted the same distance to make I a subpattern of $\mathcal{P}$. If failed, then this case cannot use UNIONMODEL, and the "pin-to-pin" model will be used. If succeeded, the output pattern can be easily calculated by union operations on output patterns in the table.

First, based on Lemma 8.1, if we can construct an input pattern P satisfying $I \subseteq P$ by union operations on basic patterns in the table T, then the output pattern constructed by uniting all the output patterns of these basic patterns is a conservative approximation of the output pattern of I. We designed a procedure called Pattern-Construction that can complete this construction successfully, which is shown in Figure 8.5. The subroutine Extr-CmbPattern(S, i, A, B) searches for a pair of combinable patterns A and B satisfying $A_i \diamond B_i$ in pattern set S. If this search succeeds, A and B are removed from S and *true* is returned. Procedure Pattern-Construction searches and puts all the basic patterns overlapping with I into pattern set S, then for each primary input i, it unites all the combinable patterns found

by Extr-CmbPattern. After the ith outer loop, the following condition is satisfied:

$$\forall P \in S : I_i \subseteq P_i.$$

Thus, at the end of this procedure, S contains only one pattern, and I is the subpattern of this pattern.

Algorithm `Pattern-Construction`

`Notations:`
T: `a table storing all the basic patterns`
I: `a given input pattern`
n: `the number of windows in` I

`Procedure:`
$S = \{P : (P \in T) \wedge (P \cap I \neq \Phi)\}$
`for` $i = 1$ `to` n
  `while(Extr-CmbPattern(`S, i, A, B`)=`$true$`)`
    `put` $X = A \cup B$ `into` S`;`
`return` S

Figure 8.5. Input pattern construction.

For example, in Figure 8.4, suppose the input pattern is

$$I\{[2, 19], [3, 18]\}.$$

Initially, S contains patterns $A\{a, c\}$, $B\{a, d\}$, $C\{b, c\}$ and $D\{b, d\}$, then in the first outer iteration, $i = 1$, we first unite A and C to get pattern $E\{[0, 20], [0, 10]\}$, then unite B and D to get pattern $F\{[0, 20], [10, 20]\}$, so before the second outer iteration, S contains E and F, and I_1 is a subwindow of E_1 and F_1. Then in the second outer iteration, $i = 2$, we unite E and F to get a pattern $X\{[0, 20], [0, 20]\}$. Obviously I is a subpattern of X. From this

procedure, we can see that the resulting input pattern is the same with the union of all the basic patterns overlapping with I. So based on Lemma 8.1, we have the following theorem:

Theorem 8.1. *The output pattern for an input pattern I can be conservatively calculated by uniting the output patterns for basic patterns that overlap with I.*

Thus, it is not required to do the expensive *input* pattern unions. Instead, we search for all the basic patterns overlapping with I, then look up the table T to find and unite all the *output* patterns corresponding to these basic patterns to get the desired output pattern directly. And during this union step, we do not need to search for the combinable patterns and unite them pair by pair. Instead, for each primary output, we can simply select a window composed by the earliest and the latest arrival time among the corresponding windows of these output patterns as the corresponding output window of the desired output pattern. Based on this union procedure, we know that UNIONMODEL achieves the conservative output pattern for any given input pattern.

Because of the large number of table entries, the most expensive step is to search for the basic patterns that overlap with I in the table. We need to design a method to find them efficiently. We label the primary inputs by $1 \ldots n$, and label the k parts in each window of $\mathcal{P}$ by $1 \ldots k$ in the increasing order of the earliest arrival time. Then each basic pattern can be represented by a distinct key with radix k: $(a_n \ldots a_1)_k$, where $0 \leq a_i \leq k-1$ for $i = 1 \ldots n$, and the ith window of this pattern is the $(a_i + 1)$th part in $\mathcal{P}_i$. Then we put the input patterns and their corresponding output patterns in a table in the increasing order of keys. For a given input pattern I, we can find the range of parts $[i_p, i_q]$ in $\mathcal{P}_i$ overlapping with I_i for $i = 1 \ldots n$, then the keys of the basic patterns that overlap with I are all the distinct

keys satisfying $i_p \leq a_i \leq i_q$ for $i = 1 \ldots n$. Obviously this method can easily find the basic patterns that overlap with I in the table.

An important contribution of UNIONMODEL is that it provides a framework for timing macro-modeling of combinational hard IP blocks with the consideration of crosstalk effects, that is, the obtainment of output patterns for basic patterns is not restricted to STA. Instead, other timing analysis or simulation methods to calculate the output patterns can be embedded into this macro-model. Also we can easily incorporate functional information into this macro-model.

8.3.3. Speed-up techniques

The bottleneck of UNIONMODEL is the large number of basic patterns. However, if input pattern A can be constructed by shifting input pattern B, the output pattern of A can also be constructed by shifting the output pattern for B. We call that A and B are *relatively the same*. Obviously many basic patterns are relatively the same. For example, in Figure 8.4, the corresponding windows in input patterns {a,c} and {b,d} are shifted 10 unit time respectively, so these two patterns are relatively the same. So it is not necessary to perform timing analysis on all the basic patterns respectively. We only perform timing analysis on the basic patterns that are not relatively the same with each other. Our experiments show that this technique can reduce the total number of basic patterns nearly half.

The number of basic patterns in UNIONMODEL is k^n, where k is the number of parts of an input window in $\mathcal{P}$, and n is the number of inputs. So it is very efficient to speed-up the extraction of UNIONMODEL if we can reduce the number of inputs. In reality, it is possible to partition the input set into several disjoint sets such that the relative arrival time of any two inputs in different sets has no influence on the same coupling nodes. Then the number

of basic patterns in UNIONMODEL is $\sum_{1\leq i\leq u} k^{n_i}$, where u is the number of disjoint sets, and n_i is the number of inputs in the ith disjoint set. This correlated input set partitioning technique will divide a large size problem to many small size problems, which will speed-up the overall running much. Our experiments show that this method can reduce the number of basic patterns greatly for some cases.

When the range of input windows in $\mathcal{P}$ is large, we need to partition each input window in $\mathcal{P}$ into many parts to guarantee the accuracy, so the number of basic patterns increases greatly. In order to avoid this problem, we randomly select some input patterns, do STA or simulation to get the output patterns, then we can find the range of input patterns where delay changes frequently, so we can shrink the range of input windows in $\mathcal{P}$ to this range, and for the input patterns that cannot be shifted into $\mathcal{P}$, we use conventional "pin-to-pin" model. This range shrinking step can greatly reduce the number of basic patterns, and benefit the model extraction and application speed.

8.4. Experimental results

Table 8.1. Comparison results of STA, SIMPMODEL and "pin-to-pin" model

circuit				STA	SIMPMODEL				"pin-to-pin" model	
name	#inputs	#outputs	#gates	time(s)	ETime(s)	ATime(s)	MaxE(%)	AveE(%)	MaxE(%)	AveE(%)
c17	5	2	6	0.00	0.00	0.00	0.00	0.00	0.06	0.03
c432	36	7	160	0.04	0.23	0.05	69.21	11.18	72.23	12.29
c499	41	32	202	0.05	0.42	0.11	35.00	1.62	46.21	6.72
c880	60	26	383	0.22	1.33	0.39	1309.11	141.15	8848.00	629.36
c1355	41	32	546	0.29	1.02	0.21	20.97	3.02	26.10	6.43
c1908	33	25	880	0.68	1.39	0.07	28.60	1.82	32.73	10.67
c3540	50	22	1669	2.30	3.82	0.16	212.72	11.49	212.72	20.54
c5315	178	123	2307	4.71	9.76	0.13	168.08	1.68	12458.40	107.48
c6288	32	32	2416	5.08	6.38	0.12	0.00	0.00	8.01	0.87
c7552	207	108	3513	10.2	16.26	0.12	32.06	0.41	13073.70	122.99

SIMPMODEL and UNIONMODEL have been implemented in C++ and tested on ISCAS85 benchmark circuits. For each circuit, we randomly designated the coupling between

wires, and input patterns are also randomly generated. All experiments were run on a Linux PC with a 2.4G Hz Xeon CPU and 2.0 GB memory.

To verify the results of our methods, we designed a STA tool based on an iterative switch factor method that works as follows. First, it uses the modified dynamically bounded delay model to model the delay of coupling nodes, then iteratively does PERT-traversal on the graph representing circuit, and after each iteration updates all the $\gamma_{v,a}$. It does not stop until there is no change on all $\gamma_{v,a}$, and the output pattern in last PERT-traversal is the desired. We also implemented a "pin-to-pin" model for comparison, which assumed that all couplings take effect. For each case, we compare the results from our model applications with the results from the "pin-to-pin" model. Suppose for the same primary input, $\{[d_1, d_2]\}$ is the output pattern from our models, $\{[a_1, a_2]\}$ is the output pattern from STA, then the error of our models for this primary output is defined as

$$[(d_2 - d_1) - (a_2 - a_1)]/(a_2 - a_1).$$

The average error is the summation of the errors for all the primary outputs divided by the number of primary outputs. The maximal error is the maximum of the errors for the primary outputs.

A comparison of results among STA, SIMPMODEL and "pin-to-pin" model is shown in Table 8.1, where ETime is the time of model extraction, ATime is the time of model application, MaxE is the maximal error, and AveE is the average error. We can see that the results of SIMPMODEL are much more accurate than the "pin-to-pin" model, and the running time of the application of SIMPMODEL is always less than 1 second, much

faster than STA in large cases. The errors are always non-negative, which confirms that SIMPMODEL is conservative.

Table 8.2. Comparison results of STA and UNIONMODEL

circuit		STA	UNIONMODEL			
name	RPI #	time (s)	# basic patterns	ETime (s)	ATime (s)	MaxE (%)
c17	5	0.00	27,000	1.99	8.94	0.00
c432	5	0.03	3,200,000	2133.00	204.63	0.01
c499	5	0.06	1,048,576	1374.00	245.58	0.01
c1355	4	0.33	65,536	129.24	14.80	0.00
c5315	5	4.65	256	5.52	0.65	0.00

Since UNIONMODEL can only deal with cases with a small number of correlated primary inputs, for the cases with a large number of correlated primary inputs, we designated the arrival time windows of most primary inputs to be invalid windows $[\infty, -\infty]$ and modeled only the timing behavior of the circuit stimulated by the remaining primary inputs that are denoted by RPI in our test. We calculated the output pattern for each basic pattern by STA. A comparison of the results from UNIONMODEL with STA is shown in Table 8.2. We can see that the results of UNIONMODEL are almost the same with the results of STA, that is, UNIONMODEL is an accurate model. From Table 8.2 we can also see that although the numbers of RPI in c499 and c5315 are the same, the number of basic patterns in c5315 is much less than in c499, which is the effect of less number of correlated inputs in c5315 than in c499. This confirms that our correlated input set partition improves the model extraction and application speed.

8.5. Conclusion

We present two macro-models to characterize the timing behavior of combinational hard IP block with the consideration of crosstalk effects. The first model, SIMPMODEL, keeps a coupling graph and lists the conditions on input patterns for couplings not to take effect. It

can fast estimate the output pattern for a given input pattern with the sacrifice of accuracy. The second model, UNIONMODEL, performs STA or simulations on basic input patterns, and based on these results constructs the output pattern for a given input pattern accurately. Both macro-models are conservative, and can greatly reduce the pessimism existing in the traditional "pin-to-pin" model. To the best of our knowledge, this is the first work to deal with timing macro-modeling problem with the consideration of crosstalk effects.

References

[1] A. Agarwal, D. Blaauw, S. Sundareswaran, V. Zolotov, M. Zhou, K. Gala, and R. Panda. Path-based statistical timing analysis considering inter- and intra-die correlations. In *ACM Intl. Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 16–21, 2002.

[2] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 900–907, San Jose,CA, November 2003.

[3] A. Agarwal, D. Blaauw, V. Zolotov, and S. Vrudhula. Computation and refinement of statistical bounds on circuit delay. In *Proc. of the Design Automation Conf.*, pages 348–349, 2003.

[4] K. Agarwal, Y. Cao, T. Sato, D. Sylvester, and C. Hu. Efficient generation of delay change curves for noise-aware static timing analysis. In *Proceedings of 15th International Conference on VLSI Design*, pages 77–84, 2002.

[5] K. Agarwal, D. Sylvester, D. Blaauw, F. Liu, S. Nassif, and S. Vrudhula. Variational delay metrics for interconnect timing analysis. In *Proc. of the Design Automation Conf.*, pages 381–384, 2004.

[6] C. Alpert and A. Devgan. Wire segmenting for improved buffer insertion. In *Proc. of the Design Automation Conf.*, pages 588–593, 1997.

[7] C. J. Alpert, M. Hrkic, and S. T. Quay. A fast algorithm for identifying good buffer insertion candidate locations. In *International Symposium on Physical Design*, pages 47–52, 2004.

[8] R. Arunachalam, K. Rajagopal, and L. T. Pilleggi. Taco: Timing analysis with coupling. In *Proc. of the Design Automation Conf.*, pages 266–269, Los Angeles, CA, June 2000.

[9] H. B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, 1990.

[10] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming B*, 98:49–71, 2003.

[11] A. Bhardwaj, S. B. Vrudhula, and D. Blaavw. Tau: Timing analysis under uncertainty. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 615–620, San Jose,CA, November 2003.

[12] R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.

[13] T. M. Burks and K. A. Sakallah. Optimization of critical paths in circuits with level-sensitive latches. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 468–473, 1994.

[14] H. Chang and S. Sapatnekar. Statisitical timing analysis considering spatial correlations using a single PERT-like traversal. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 621–625, San Jose, CA, November 2003.

[15] H. Chang, V. Zolotov, S. Narayan, and C. Visweswariah. Parameterized block-based statistical timing analysis with non-gaussian parameters, nonlinear delay functions. In *Proc. of the Design Automation Conf.*, pages 71–76, 2005.

[16] M. C. T. Chao, L. C. Wang, K. T. Cheng, and S. Kundu. Static statistical timing analysis for latch-based pipeline designs. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 468–472, 2004.

[17] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. In *Proc. of the Design Automation Conf.*, pages 492–498, 1992.

[18] P. Chen, D. A. Kirkpatrick, and K. Keutzer. Switching window computation for static timing analysis in presence of crosstalk noise. In *Proc. Intl. Conf. on Computer-Aided Design*, San Jose, CA, November 2000.

[19] P. Chen, Y. Kukimoto, C.-C. Teng, and K. Keutzer. On covergence of switching windows computation in presence of crosstalk noise. In *International Symposium on Physical Design*, pages 84–89, 2002.

[20] R. Chen and H. Zhou. Clock schedule verification under process variations. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 619–625, San Jose, CA, November 2004.

[21] R. Chen and H. Zhou. A flexible data structure for efficient buffer insertion. In *Proc. Intl. Conf. on Computer Design*, pages 216–221, 2004.

[22] R. Chen and H. Zhou. Timing macro-modeling of IP blocks with crosstalk. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 155–159, San Jose, CA, November 2004.

[23] R. Chen and H. Zhou. Efficient algorithms for buffer insertion in general circuits based on network flow. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 322–326, 2005.

[24] R. Chen and H. Zhou. An efficient data structure for max-plus merge in dynamic programming. *IEEE Transactions on Computer Aided Design*, 25(12):3004–3009, December 2006.

[25] R. Chen and H. Zhou. Statistical timing verification for transparently latched circuits. *IEEE Transactions on Computer Aided Design*, 25:1847–1855, September 2006.

[26] R. Chen and H. Zhou. An efficient algorithm for buffer insertion in general circuits based on network flow. *IEEE Transactions on Computer Aided Design*, 26(11), November 2007.

[27] R. Chen and H. Zhou. Fast buffer insertion for yield optimization under process variations. In *Proc. Asian and South Pacific Design Automation Conference*, pages 19–24, 2007.

[28] R. Chen and H. Zhou. Fast estimation of timing yield bounds for process variations. *IEEE Transactions on Very Large-Scale Integrated Systems*, 2007. Accepted for publication.

[29] R. Chen and H. Zhou. Fast min-cost buffer insertion under process variations. In *Proc. of the Design Automation Conf.*, pages 338–343, 2007.

[30] R. Chen and H. Zhou. New block-based statistical timing analysis approaches without moment matching. In *Proc. Asian and South Pacific Design Automation Conference*, pages 462–467, 2007.

[31] R. Chen and H. Zhou. Timing budgeting under arbitrary process variations. In *Proc. Intl. Conf. on Computer-Aided Design*, 2007.

[32] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester. Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 1023–1028, 2005.

[33] C. E. Clark. The Greatest of a Finite Set of Random Variables. *Operations Research*, 9(2):145–162, 1961.

[34] J. Cong and Z. Pan. Interconnect performance estimation models for synthesis and design planning. In *Workshop Notes of Intl. Workshop on Logic Synthesis*, pages 427–433, 1998.

[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[36] A. J. Daga, L. Mize, S. Sripada, C. Wolff, and Q. Wu. Automated timing model generation. In *Proc. of the Design Automation Conf.*, pages 146–151, 2002.

[37] A. Davoodi and A. Srivastava. Variability-driven buffer insertion considering correlations. In *Proc. Intl. Conf. on Computer Design*, 2005.

[38] A. Davoodi and A. Srivastava. Variability driven gate sizing for binning yield optimization. In *Proc. of the Design Automation Conf.*, pages 959–964, San Francisco, CA, USA, July 2006.

[39] C. Demetrescu and I. Finocchi. Combinatorial algorithms for feedback problems in directed graphs. *Information Processing Letters*, 86:129–136, 2003.

[40] L. Deng and M. D. Wong. Buffer insertion under process variations for delay minimization. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 317–321, 2005.

[41] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 607–614, San Jose,CA, November 2003.

[42] C. Ebeling and B. Lockyear. On the performance of level-clocked circuits. In *Advanced Research in VLSI*, pages 242–356, 1995.

[43] M. Foltin, B. Foutz, and S. Tyler. Efficient stimulus independent timing abstraction model based on a new concept of circuit block transparency. In *Proc. of the Design Automation Conf.*, pages 158–163, 2002.

[44] J. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[45] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman and Co., 1979.

[46] S. Ghiasi, E. Bozorgzadeh, P.-K Huang, R. Jafari, and M. Sarrafzadeh. A unified theory of timing budget management. *IEEE Transactions on Computer Aided Design*, 25(11):2364–2375, November 2006.

[47] P. D. Gross, R. Arunachalam, K. Rajagopal, and L. T. Pileggi. Determination of worst-case aggressor alighment for delay calculation. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 212–219, San Jose, CA, November 1998.

[48] M. Guthaus, N. Venkateswaran, C. Visweswariah, and Z. Zolotov. Gate sizing using incremental parameterized statistical timing analysis. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 1029–1036, 2005.

[49] S. Hassoun. Critical path analysis uing a dynamically bounded delay model. In *Proc. of the Design Automation Conf.*, pages 260–265, Los Angeles, CA, June 2000.

[50] L. He, A. Kahng, K. H. Tam, and J. Xiong. Simultaneous buffer insertion and wire sizing considering systematic CMP variation and random Leff variation. In *International Symposium on Physical Design*, pages 78–85, 2005.

[51] S. Hu, C. J. Alpert, J. Hu, S. Karandikar, Z. Li, W. Shi, and C. N. Sze. Fast algorithms for slew constrained minimum cost buffering. In *Proc. of the Design Automation Conf.*, pages 308–313, 2006.

[52] F. Huebbers, A. Dasdan, and Y. Ismail. Computation of accurate interconnect process parameter values for performance corners under process variations. In *Proc. of the Design Automation Conf.*, pages 797–800, 2006.

[53] J. A. G. Jess, K. Kalafala, S. R. Naidu, R. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. In *Proc. of the Design Automation Conf.*, pages 932–937, 2003.

[54] M. Kang, W. W.-M. Dai, T. Dillinger, and D. LaPotin. Delay bounded buffered tree construction for timing driven floorplanning. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 707–712, 1997.

[55] A. V. Karzanov and S. T. McCormick. Polynomial methods for separable convex optimization in unimodular linear spaces with applications. *SIAM*, 26:1245–1275, 1997.

[56] C. Kashyap, C. Alpert, F. Liu, and A. Devgan. Closed form expressions for extending step delay and slew metrics to ramp inputs. In *International Symposium on Physical Design*, pages 24–31, 2003.

[57] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *Proc. of the Design Automation Conf.*, pages 617–623, June 1987.

[58] V. Khandelwal, A. Davoodi, A. Nanavati, and A. Srivastava. A probabilistic approach to buffer insertion. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 560–567, 2003.

[59] W. J. Krzanowski. *Principles of Multivariate Analysis*. Oxford University Press, 2000.

[60] Z. Li and W. Shi. Fbi: Fast buffer insertion for interconnect optimization. `http://dropzone.tamu.edu/~zhuoli/GSRC/fast_buffer_insertion.html`.

[61] J. Lillis, C. K. Cheng, T. T. Lin, and C. Y. Ho. New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing. In *Proc. of the Design Automation Conf.*, pages 395–400, 1996.

[62] J. Lillis, C. K. Cheng, and T. T. Y. Lin. Optimal wire sizing and buffer insertion for low power and a generalized delay model. *IEEE Journal of Solid-State Circuits*, 31:437–447, 1996.

[63] I-M. Liu, A. Aziz, and D. F. Wong. Meeting delay constraints in DSM by minimal repeater insertion. In *Proc. DATE: Design Automation and Test in Europe*, pages 436–440, Paris, France, March 2000.

[64] I.-M. Liu, A. Aziz, D. F. Wong, and H. Zhou. An efficient buffer insertion algorithm for large networks based on lagrangian relaxation. In *Proc. Intl. Conf. on Computer Design*, pages 210–215, Austin, TX, October 1999.

[65] J. D. Ma, C. F. Fang, R. A. Rutenbar, X. Xie, and D. S. Boning. Interval-valued statistical modeling of oxide chemical-mechanical polishing. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 141–148, San Jose, CA, 2005.

[66] N. Maheshwari and S. S. Sapatnekar. Optimizing large multi-phase level-clocked circuits. *IEEE Transactions on Computer Aided Design*, 18(9):1249–1264, September 1999.

[67] M. Mani, A. Devgan, and M. Orshansky. An efficient algorithm for statistical minimization of total power under timing yield constraints. In *Proc. of the Design Automation Conf.*, pages 309–314, Anaheim, CA, June 2005.

[68] M. Mani and M. Orshansky. A new statistical optimization algorithm for gate sizing. In *Proc. Intl. Conf. on Computer Design*, pages 272–277, San Jose, CA, October 2004.

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

[70] MOSEK. Mosek aps optimization software. http://www.mosek.com.

[71] J. L. Neves and E. G. Friedman. Optimal Clock Skew Scheduling Tolerant to Process Variations. In *Proc. of the Design Automation Conf.*, pages 623–628, Las Vegas, NV, June 1996.

[72] T. Okamoto and J. Cong. Buffered Steiner tree construction with wire sizing for interconnect layout optimization. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 44–49, 1996.

[73] M. Orshansky and K. Keutzer. A general probabilistic framework for worst case timing analysis. In *Proc. of the Design Automation Conf.*, pages 556–561, 2002.

[74] P. J. Osler. Placement driven synthesis case studies on two sets of two chips: Hierarchical and flat. In *International Symposium on Physical Design*, pages 190–197, 2004.

[75] Y. Peng and X. Liu. Low-power repeater insertion with both delay and slew rate constraints. In *Proc. of the Design Automation Conf.*, pages 303–307, 2006.

[76] W. Pugh. A skip list cookbook. Technical Report CS-TR-2286.1, University of Maryland, College Park, 1990.

[77] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6), 1990.

[78] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. In *Proc. of the Design Automation Conf.*, pages 111–117, 1990.

[79] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. *check*T_c and *mint*$_c$: Timing verification and optimal clocking of synchronous digital circuits. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 552–555, November 1990.

[80] S. S. Sapatnekar. A timing model incorporating the effect of crosstalk on delay and its application to optimal channel routing. *IEEE Transactions on Computer Aided Design*, 2000.

[81] Y. Sasaki and G. De Micheli. Crosstalk delay analysis using relative window method. In *ASIC/SoC Conference*, 1999.

[82] Y. Sasaki and K. Yano. Multi-aggressor relative window method for timing analysis including crosstalk delay degradation. In *Custom Integrated Circuit Conference*, pages 495–498, 2000.

[83] P. Saxena, N. Menezes, P. Cocchini, and Desmond A. Kirkpatrick. The scaling challenge: Can correct-by-construction design help? In *International Symposium on Physical Design*, pages 51–58, 2003.

[84] N. Shenoy and R. K. Brayton. Graph algorithms for clock schedule optimization. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 132–136, Santa Clara, CA, 1992.

[85] Narendra V. Shenoy. *Timing Issues in Sequential Circuits*. PhD thesis, UC Berkeley, 1993.

[86] W. Shi. `http://ece.tamu.edu/~wshi`.

[87] W. Shi. A fast algorithm for area minimization of slicing floorplans. *IEEE Transactions on Computer Aided Design*, 15:550–557, 1996.

[88] W. Shi and Z. Li. An $O(n \log n)$ time algorithm for optimal buffer insertion. In *Proc. of the Design Automation Conf.*, pages 580–585, 2003.

[89] W. Shi and Z. Li. A fast algorithm for optimal buffer insertion. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 24(6):879–891, 2005.

[90] W. Shi, Z. Li, and C. J. Alpert. Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost. In *Proc. Asia and South Pacific Design Automation Conference*, pages 609–614, 2004.

[91] A. Singhee, C. F. Fang, J. D. Ma, and R. A. Rutenbar. Probabilistic interval-valued computation: toward a practical surrogate for statistics inside cad tools. In *Proc. of the Design Automation Conf.*, pages 167–172, San Francisco, CA, USA, July 2006.

[92] D. Sinha, N. Shenoy, and H. Zhou. Statistical gate sizing for timing yield optimization. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 1037–1041, San Jose, CA, 2005.

[93] L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 59:91–101, 1983.

[94] V. Sundararajan, S. S. Sapatnekar, and K. K. Parhi. Fast and exact transistor sizing based on iterative relaxation. *IEEE Transactions on Computer Aided Design*, 21(5):568–581, 2002.

[95] C.-N. Sze, C. J. Alpert, J. Hu, and W. Shi. Path based buffer insertion. In *Proc. of the Design Automation Conf.*, Anaheim, CA, June 2005.

[96] T. G. Szymanski. Computing optimal clock schedules. In *Proc. of the Design Automation Conf.*, pages 399–404, Anaheim, CA, June 1992.

[97] T. G. Szymanski and N. Shenoy. Verifying clock schedules. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 124–131, 1992.

[98] R. Tarjan. Enumeration of the elementary circuits of a directed graph. *J.Comput.*, pages 211–216, 1973.

[99] P. F. Tehrani, S. W. Chyou, and U. Ekambaram. Deep sub-micron static timing analysis in presence of crosstalk. In *International Symposium on Quality Electronic Design*, pages 505–512, 2000.

[100] B. Thudi and D. Blaauw. Non-iterative switching window computation for delay noise. In *Proc. of the Design Automation Conf.*, pages 390–395, 2003.

[101] L. P. P. P. van Ginneken. Buffer placement in distributed RC-tree networks for minimal Elmore delay. In *Proc. Intl. Symposium on Circuits and Systems*, pages 865–868, 1990.

[102] S. V. Venkatesh, R. Palermo, M. Mortazavi, and K. Sakallah. Timing abstraction of intellectual property blocks. In *Proc. Custom Integrated Circuits Conf.*, pages 99–102, 1997.

[103] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *Proc. of the Design Automation Conf.*, pages 331–336, San Diego, June 2004.

[104] M. Waghmode, Z. Li, and W. Shi. Buffer insertion in large circuits with constructive solution search techniques. In *Proc. of the Design Automation Conf.*, pages 296–301, 2006.

[105] W.-S. Wang, V. Kreinovich, and M. Orshansky. Statistical timing based on incomplete probabilistic descriptions of parameter uncertainty. In *Proc. of the Design Automation Conf.*, pages 161–166, San Francisco, CA, USA, July 2006.

[106] W.-S. Wang and M. Orshansky. Robust estimation of parametric yield under limited descriptions of uncertainty. In *Proc. Intl. Conf. on Computer-Aided Design*, pages 884–890, San Jose, California, November 2006.

[107] T. Xiao, C.-W. Chang, and M. Marek-Sadowska. Efficient static timing analysis in presence of crosstalk. In *Proceedings of 13th Annual IEEE International ASIC/SOC Conference*, pages 335–339, 2000.

[108] T. Xiao and M. Marek-Sadowska. Functional correlation analysis in crosstalk induced critical paths identification. In *Proc. of the Design Automation Conf.*, pages 653–656, 2001.

[109] J. Xiong and L. He. Fast buffer insertion considering process variations. In *International Symposium on Physical Design*, 2006.

[110] J. Xiong, K. Tam, and L. He. Buffer insertion considering process variation. In *Proc. DATE: Design Automation and Test in Europe*, 2005.

[111] H. Yalcin, M. Mortazavi, R. Palermo, C. Bamji, and K. Sakallah. Functional timing analysis for IP characterization. In *Proc. of the Design Automation Conf.*, pages 731–736, 1999.

[112] H. Yalcin, R. Palermo, M. Mortazavi, C. Bamji, and K. Sakallah. An advanced timing characterization method using mode dependency. In *Proc. of the Design Automation Conf.*, pages 657–660, 2001.

[113] Y. Zhan, A. J. Strojwas, X. Li, L. T. Pileggi, and D. Newmark. Correlation-aware statistical timing analysis with non-gaussian delay distributions. In *Proc. of the Design Automation Conf.*, pages 77–82, 2005.

[114] L. Zhang, W. Chen, Y. Hu, J. A. Gubner, and C. C. Chen. Correlation-preserved non-gaussian statistical timing analysis with quadratic timing model. In *Proc. of the Design Automation Conf.*, pages 83–88, 2005.

[115] L. Zhang, Y. Hu, and C. C. Chen. Statistical timing analysis in sequential circuit for on-chip global interconnect pipelining. In *Proc. of the Design Automation Conf.*, pages 904–907, 2004.

[116] H. Zhou, N. Shenoy, and W. Nicholls. Timing analysis with crosstalk as fixpoints on a complete lattice. In *Proc. of the Design Automation Conf.*, pages 714–719, 2001.

[117] H. Zhou, D. F. Wong, I-Min Liu, and A. Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. In *Proc. of the Design Automation Conf.*, pages 96–99, 1999.

[118] H. Zhou, D. F. Wong, I-Min Liu, and A. Aziz. Simultaneous routing and buffer insertion with restrictions on buffer locations. *IEEE Transactions on Computer Aided Design*, 19(7):819–824, July 2000.

Vita

Ruiming Chen was born in Yiyang, Hunan Province, China. He received his B.S. degree in Microelectronics in 2000 and M.S. degree in Computer Sience in 2003, both from Tsinghua University, China. In September 2003, he joined NuCAD group in Northwestern University with Walter P. Murphy Fellowship for graduate studuents. He has published more than 10 technical papers on leading journals and conferences in EDA area. He worked as a Technical Co-op from June 2006 to September 2006 in EinsStat Team at IBM T. J. Watson Research Center, Yorktown Heights, NY. He has been a member of PrimeTime group in Synopsys Inc since June 2007.