TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Entwurfsautomatisierung

# Hierarchical Statistical Static Timing Analysis Considering Process Variations

**Bing Li**

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktor-Ingenieurs**

genehmigten Dissertation.

| | | |
|---|---|---|
| Vorsitzender: | | Univ.-Prof. Dr.-Ing. Jörg Eberspächer |
| Prüfer der Dissertation: | 1. | Univ.-Prof. Dr.-Ing. Ulf Schlichtmann |
| | 2. | Univ.-Prof. Dr. sc. Samarjit Chakraborty |

Die Dissertation wurde am 27.01.2010 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 15.07.2010 angenommen.

## Acknowledgments

This thesis is the result of my working at the Institute for Electronic Design Automation, Technische Universität München as a research and teaching assistant.

First of all, I thank Professor Ulf Schlichtmann for admitting me to his research group. He has patiently guided me to enter the research field of statistical timing analysis and given me constructive advices on my specific topics since the beginning. He carefully reviewed all my papers and his insightful suggestions helped me not only improve my academic writing but also form a professional research style. Additionally, he also spent much time to help me overcome all other problems during my PhD studying. Moreover, I thank him for giving me the chance to establish the VLSI design lab. For me this is a precious experience in teaching and communication with students.

Many thanks are due to the committee members Professor Samarjit Chakraborty and Professor Jörg Eberspächer for their interest in my thesis.

From PD Dr. Helmut Gräb I gained much after each of our talks. I thank him for his generous help and advices. I thank Walter Schneider and Dr. Manuel Schmidt for our fruitful discussions. I give my thanks to Ning Chen for the numerous talks and the collaboration in writing papers. Christoph Knoth gave me lots of help in writing; Xin Pan gave me invaluable suggestions as I prepared my presentations; Qingqing Chen worked with me in teaching the VLSI design lab and took over it finally. I am grateful to all of them. I thank all the other PhD students in the institute for maintaining such a creative atmosphere, which is important for me to finish my thesis.

Since I joined the institute, Dr. Bernd Finkbein, Hans Ranke, Werner Tolle, Jürgen Zenz, Susanne Werner and Gertraude Kallweit have given me all sorts of support and I thank them gratefully.

Last but not the least, I give my deepest gratitude to my wife Xue Zhao. Without her patient support and encouragement I could not finish this thesis.

Munich, January 2010

Bing Li

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

With ubiquitous presence, Integrated Circuits (IC) have become an essential part of life and economy. As a fundamental implementation method, IC design as well as manufacturing has affected most industry branches and has been involved in nearly every innovation of the present era. Meanwhile, IC design methodology and manufacturing technology are compelled to innovate themselves to meet the increasing technical and economic requirements of the rapidly advancing industry.

In the past 40 years, the IC industry has met the requirements from different application areas by keeping the pace of device scaling of Moore's law [Moo65, Moo03]. With smaller devices, more functions are integrated into a chip without increasing the die size. This integration enables the trend of System-on-Chip (SoC) design, where a chip provides the functionality of a complete system. For example, such integration happens continuously in the field of consumer electronics [Lee05], where more and more functions are integrated in each new product generation. With more devices in a chip, design complexity of an SoC chip increases drastically. Facing the pressure of more Non-Recurring Engineering (NRE) expenses and shorter time to market, new design methodologies, such as Electronic System Level (ESL) design and Intellectual Property (IP) integration, are considered as the keys to solve design issues in managing the exploding number of transistors and functions in a design [CK08, MP03, Hen03].

The Electronic Design Automation (EDA) companies assume the role to provide methodologies and tools for IC design and verification. With these commercial tools, different design flows for digital circuits are constructed. These design flows share some common steps: circuit descriptions in Hardware Design Language (HDL) are translated into netlists at synthesis step; logic gates included in the netlists are distributed on the die at layout step and interconnects between them are routed thereafter; before sign-off, designs are verified against functionality, timing and power specifications.

Standing at the center of these design flows, a widely used development model is data abstraction. From high level abstraction, e.g., ESL, to low level parasitic extraction, details are considered gradually in the design flow. In this way, tasks of design and verification become manageable at each step. Consequently, this abstraction reduces the complexity not only for designers but also for design flow engineers. As examples of such abstraction, preverified modules are considered as black boxes in the SoC design flow; geometries instead of internal transistor structures of digital gates are used during the layout step.

At the boundary between design and manufacturing, similar data abstraction happens to hide manufacturing details from designers. These manufacturing details are abstracted into design rules, which should be abided by designers and checked by EDA tools before sign-off. For example, at the manufacturing side process variations exist since the beginning IC industry because of the inaccuracy of process control in foundries. Consequently, parameters of transistors on a chip after manufacturing deviate from design values or nominal values. To isolate such variations from design engineers, worst-case values of transistor parameters are used to create delay models of gates. By evaluating the performance of a circuit using worst-case delay models, IC chips after manufacturing are guaranteed to work properly.

As the transistor feature size enters the deep submicron realm, process variations become more significant [Nas01]. The reason of such aggravation is that the scaling of accuracy control of manufacturing dose not match the scaling of transistors during the evolution of technology generations. Therefore, the relative variation, e.g., the ratio of standard deviation to the nominal value of a parameter, becomes much larger than before [Nas01]. With this variation increase, traditional worst-case analysis methodology faces new challenges.

Although it successfully served the IC industry for many years, the worst-case analysis flow is too pessimistic when process variations become large. In this design flow, gate delays are calibrated by setting all process parameters to worst-case values. Circuit performance is evaluated using these worst-case gate delays, and therefore may be overly underestimated. This pessimism drives engineers to further optimize designs which may have met timing specifications but are incorrectly reported as violating timing constraints by the worst-case analysis method. This type of overdesign increases design cost drastically because it consumes much more resources to boost circuit performance in further design iterations. Additionally, the time-to-market window of the product may be missed because of such pessimistic design evaluation.

As a promising new methodology, statistical static timing analysis (SSTA, or statistical timing analysis for simplicity) emerges. In this method, process variations are modeled as random variables directly. Consequently, the performance of a circuit becomes a distribution, from which timing and yield information can be evaluated

by engineers. By avoiding modeling gate delays to worst case, the pessimism in traditional timing analysis is reduced. Because many steps in digital IC design flow depend on the results of timing analysis, the new analysis methodology demands a thorough renovation both during design and during manufacturing.

## 1.1 Challenges in SoC Design

The rapidly evolving semiconductor industry enables integration of more functions onto one die after each advance of its technology node. With all processing and interfacing units integrated, functions which were traditionally fulfilled by a complete system can now be achieved by only one chip.

The emergence and prevalence of SoC designs are driven by the customer market. Drastic competition requires the integration of more functions into one chip in each new product generation. From the side of system integration companies the size of the printed circuit board and further the complete bill of materials can be reduced with SoC chips. From technology side, system performance, e.g., clock frequency, can be improved because off-chip interconnects between functional modules are moved into one die. Additionally, overall power consumption can also be reduced by this integration because the components inside an SoC chip consume less power than their discrete counterparts.

An example of SoC design [KCJ+00] is illustrated in Figure 1.1. This chip integrates typical functions for customer market. The programmable processor fulfills general control functions. To accelerate processing of specific data, e.g., audio and video, corresponding hardware cores are integrated. These processing units are connected through peripheral buses and interfacing logic to the outside environment. With such bus structure, more functions can be easily added into an existing design.

As shown in Figure 1.1, SoC designs exhibit differences from traditional IC designs. With more functions integrated into one chip, the number of components in an SoC chip increases fast. In order to add a new function, both the processing unit and the interfacing logic should be integrated. Furthermore, the new components need to exchange data with other components inside the chip. While keeping the efficiency and performance, the increase of components makes the design and implementation of the communication logic, e.g., crossbar, more complex and resource-consuming. Another difference between SoC and traditional IC designs is heterogeneity. Unlike traditional IC designs, whose modules are nearly all implemented in-house, the design of an SoC chip is far beyond the boundary of traditionally defined design groups because of system complexity.

With higher integration, challenges arise for SoC designs [BS99]. The first challenge is the pressure of time to market. In these days, commercial electronics market

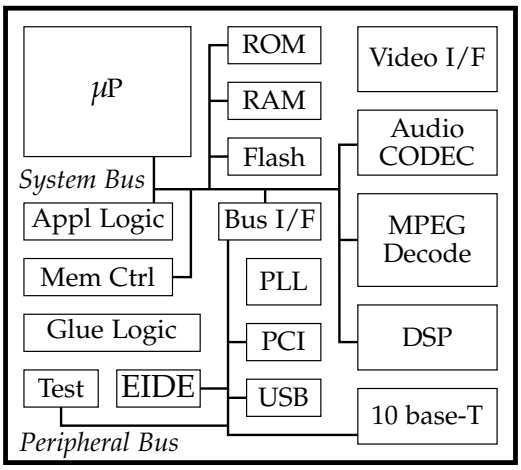


**Figure 1.1:** System on Chip Example [KCJ+00]

evolves much faster than before. Each product springs into the market in a hurry and fades away with the flood of the next generation of successors quickly. In each product generation, new functions integrated into the SoC chip increase the design complexity. Consequently, designers face the challenge to finish a more complex design in shorter time.

The second challenge is from the system heterogeneity. In order to gain competitive advantages, designers must try hard to integrate more functions in every new product. This integration reforms the definition of traditional end products for consumers so that new market demands are stimulated to keep continuous interest. An example of such integration is the cell phone market. Compared to 1990s, camera, audio and video processing units have already become necessary parts of a mobile phone. Instead of being completely designed by a company, new function components are normally obtained from different sources. Because they lack the knowledge of specific applications, designers face the new challenge of integrating these heterogeneous components and verifying the interactions between them.

The third challenge is the design flow innovation. The traditional digital design flow is successful for in-house designs. Each module is designed using a high level description language and synthesized into netlist. The layout and routing of the complete design are run using a flat netlist. So is the verification of performances, e.g., timing and power. With the evolution of SoC designs, the number of transistors increases drastically [Cla06]. In this case, the capacity of existing EDA tools faces strong challenges if the traditional flat design flow is still maintained.

With time-to-market pressure and more complex SoC designs, it is not feasible any more to design and implement each SoC chip completely from scratch. According to the customer market, the demand of a complete new SoC chip is rare, i.e., SoC designs are normally incremental in the timeline. For example, when camera, audio

and video processing units are added to an SoC for a cell phone, the basic function blocks for communication need not to be redesigned completely. This indicates that the solution for the dilemma between time-to-market pressure and design complexity of SoC chips is design reuse. In the traditional flow, the complete design is partitioned into small blocks and implemented separately. In the SoC scenario, once a subfunction module is implemented, it is used in future designs without or with incremental revision. This design reuse can be in the scope of design teams and IC companies, where the reused components are called macros or cores.

A more general ecosystem of SoC design is in the scope of the complete IC industry, where preverified components are sold for revenue. Moreover, some companies take the role of supplying SoC design components exclusively as their main business. These reusable components are called Intellectual Property (IP). Examples of IP companies are ARM, MIPS etc., from which general purpose processor cores are supplied for integration. Compared to the evolution of design methodology from full custom design to standard cell design, the design flow with preverified components continues the upward shift of abstraction [Cla06]. As the size of SoC designs continues to increase, higher level abstraction becomes the only way to divide and conquer the exploding design complexity.

Typical IP components are available in three types: hard, soft, and firm [BS99]. In hard cores, the layout and routing of the IP cores are all defined. By this way, the problem of nonoptimal layout and routing can be avoided to guarantee predictable performance. Examples of hard cores are memory blocks for their regularity, ADC and DAC for predictable performance such as bandwidth and signal integrity. With layout and routing defined, hard cores impose restrictions on application context, e.g., the number of meta layers. Because of such constraints, hard cores lose flexibility in application partially. In order to overcome the limitation of hard cores, soft cores are provided with synthesizable netlists. They can be integrated seamlessly into the standard IC design flow as a normal module and adapted to different design context easily. Compared to hard cores, the performance of the soft cores heavily depends on SoC designers, who perform the flow from logic to physical synthesis to translate the provided netlists to real circuits. Between hard and soft cores, firm cores are available with floorplanning or placement but routing information, as trade-off between predictable performance and flexibility.

Facing the challenge to integrate heterogeneous IP cores into SoC designs, traditional design flows should become IP-aware. At first, the design flow should allow the use of system-level functional models for hard cores. These models provide the same behaviors of hard cores with high level description languages, e.g., SystemVerilog [IEE07] and SystemC [IEE05]. With these models, interactions between IP cores and in-house designed blocks can be verified during an early design stage.

In addition to functional verification, the performance of a SoC design should also be

verified in module based style, or hierarchically. As an example, timing of the SoC design determines if the chip can work properly at a given clock period. If the timing constraints of a block are violated, the function of this block becomes unpredictable. At standard cell level, timing constraints include setup and hold time of flip-flop and latches. At module level, the constraints of flip-flops and latches inside hard cores are abstracted. For most of such flip-flops and latches, their timing constraints are guaranteed by the layout and routing of the hard cores themselves. Therefore, only the timing constraints at their interfaces should be verified. These constraints together form the timing models of the modules or IP blocks, and are commonly provided accompanying with functional models by IP vendors.

At system level, the design flow should have the ability to use the provided timing models to verify the performance of the complete system instead of using the hard cores directly. Using the latter for timing verification may be impossible because IP vendors may be reluctant to provide the details of the hard cores for the reason of IP protection. Even if the design details of IP cores are provided, directly verifying the complete system is infeasible, or at least not economic, because of runtime.

Another characteristic the SoC design flow should have is the ability of incremental design and verification. During design iterations, small revisions should be verified locally instead of invoking the verification of the complete design, for the sake of design time reduction and IP evaluation. With functional and timing information contained in abstracted models, a module based SoC design flow, or hierarchical SoC flow, can effectively reduce the time of design iterations, therefore accelerate the complete system development.

## 1.2 Contributions of This Work

Facing increasing process variations, statistical timing analysis is introduced into the EDA industry to remove innate pessimism in traditional worst-case analysis. Because the results of timing analysis are used by many steps in a digital design flow, the adoption of this new emerging and fast evolving methodology demands a thorough investigation of individual methodology steps.

In this thesis, statistical timing analysis methodology is investigated in the context of hierarchical verification, i.e., hierarchical statistical timing analysis. In the first step of such a flow, timing models for submodules are extracted by IP vendors or design teams. When variations are considered, the statistical representation of gate delays makes timing model extraction methods for static timing analysis infeasible. As a solution, this thesis proposes a criticality based method to remove noncritical edges from combinational circuits. This removal not only reduces the size of the

timing model directly, but also increases the chance to apply the basic merge operations inherited from static timing model extraction. For flip-flop based circuits, the classical extracted timing model [ALS+02] is extended to incorporate variations into a standard flow of statistical timing analysis. For latch based circuits, the limitation of transparency assumption in static timing models is overcome in the compact timing models proposed in this thesis. With such new statistical timing models, timing verification of a hierarchical design can be accelerated by several orders of magnitude.

After modules are integrated into a hierarchical design, correlation between these modules becomes a new problem in hierarchical statistical timing analysis. In contrast to the assumption of full correlation between delays in static timing analysis, delays from modules have correlation determined by relative layout of gates. Only knowing the layout of gates inside a module, a timing model can only contain correlation information inside the module. For timing analysis of the complete design, this thesis proposes a method to substitute independent variables in modules by the ones for the complete hierarchical design. The correlation between modules after instantiation is represented by sharing the same set of random variables, therefore the accuracy of timing analysis is maintained.

From the work of this thesis, preliminary results are published in [LKS+08,LCS+09b, LCS09a]. In [LKS+08] the basic idea of noncritical delay edge removal is investigated in the context of static timing analysis. This idea is extended in [LCS+09b] to identify noncritical gate delays by computing criticalities of them. The challenge to apply the proposed timing models is also investigated in [LCS+09b], where the variable substitution method for the timing analysis of the complete design is proposed. For flip-flop and latch based circuits, the method in [LCS09a] extracts the statistical interfacing constraints of a module. The minimum clock period of such a module is compressed into one random variable in the timing model. Therefore the runtime of hierarchical analysis can be remarkably reduced. The methods described above together form a complete solution for hierarchical statistical timing analysis.

## 1.3 Organization of This Dissertation

The structure of this thesis is as follows. The fundamentals of static timing analysis are introduced in Section 2. Process variations, existing statistical timing analysis methods, timing model extraction methods for static timing analysis, and state of the art in hierarchical statistical timing analysis are described in Section 3. With standard statistical timing methods as engine, timing model extraction for combinational, flip-flop based and latch based circuits are explained in Section 4. The variable substitution method handling correlation between modules is described in

Section 5. Thereafter, experimental results are shown in Section 6. In the end, this thesis is concluded in Section 7.

## 1.4 Summary

The complexity of integrated circuits increases drastically as more functions are merged into one chip. In order to handle such design complexity, hierarchical design flows are adopted. Modules are implemented and verified separately; SoC designs are created from modules with abstract models to reduce the runtime of verification. As the IC manufacturing technology scales further, process variations become relatively large and therefore cause the traditional worst-case design methodology to be too pessimistic. As a solution, statistical timing analysis is introduced to evaluate the timing performance of a circuit more accurately. The problems of applying such a statistical analysis method in hierarchical design flows are investigated in this thesis. Fast and accurate methods are proposed to extract statistical timing models for different types of circuits and to apply these timing models in a statistical hierarchical design framework.

# Chapter 2

# Static Timing Analysis

In the previous section, an SoC design is described as a combination of subsystems and submodules. Viewing from the aspect of circuit structure, sequential circuit assumes overwhelming dominance in implementing submodules. In this section, the basics of sequential circuits, with focus on timing characteristics, are reviewed.

## 2.1 Sequential Circuits and Timing Graphs

The typical structure of a digital circuit is shown in Figure 2.1. The input combinational logic generates the data for the registers and the output logic for the primary outputs. The outputs of registers are connected back to the input logic, forming combinational paths between registers. The registers store the data at their inputs when the triggering signal, called clock, is valid.

Based on when the data are stored, registers can be classified into two types: flip-flops and latches. A flip-flop transfers the data at its input to its output only when the predefined clock edge appears. On the contrary, a latch transfers the data when the clock is active, high or low according to the type of the latch. Therefore, a flip-flop is normally called edge-triggered and a latch level-triggered. Without losing generality, all flip-flops are assumed to be triggered at the rising clock edge, and all latches when the clock signal is high, in the rest of this thesis. After the clock signal of a latch switches to inactive, the data at its input does not affect the output value. That is, the data is locked into the latch at the falling clock edge, called *latching edge*. After the clock signal of a latch switches to active, the data at its input can start to propagate instantly. This edge is therefore named as *enabling edge*. Similarly, the latching edge of a flip-flop is defined as the rising edge of the clock signal. The enabling edge of a flip-flop is the same as its latching edge, because its input data can be transferred to its output only at the active clock edge.

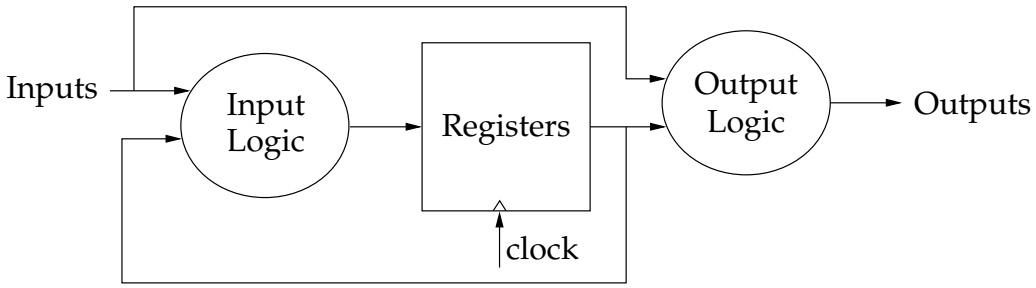


**Figure 2.1:** Sequential Circuit Structure

In order to lock the data correctly, a register has special requirements to the data at its input. Assuming the latching clock edge is at time $t_e$, the data at the input of the register should be stable between $(t_e - s_i)$ and $(t_e + h_i)$, where $s_i$ is called setup time and $h_i$ hold time, for register $i$. During this time range, any data change at the input of the register may cause it to enter metastability state with a certain probability [Cad04]. In this state the output of the register stays between 0 and 1. This is considered to be a circuit failure. A hold time constraint violation happens when the signal from a register propagates to the next stage too fast. It can be corrected easily, e.g., by delay insertion and padding [SBSV93]. On the contrary, setup time constraints determine the maximum clock frequency and should be checked when verifying a circuit against different clock frequencies. In the following, only setup time constraints will be discussed for simplicity.

The setup time constraints are for individual registers. Additionally, multi-cycle paths [ADM92, YC06] are also specified as timing constraints. Such a path allows a signal from the output of a register to reach the input of the next register in more than one clock cycle without affecting the correct function of the circuit. When designing digital circuits, multi-cycle paths are normally not specified by design engineers in hardware programming language, but by system engineers when verifying the timing of the circuit [Syn09]. This makes multi-cycle path information hard to be integrated into the design flow. Therefore they are often ignored in practice. Without specifying multi-cycle paths, timing tools normally consider all paths between registers as one clock cycle path [Syn09]. This assumption is very conservative but guarantees that the final chip can function properly. In the rest of this thesis, multi-cycle paths will not be discussed. The proposed methods, however, can be easily extended to include such information.

For convenience to explain timing specifications of sequential circuits, a *reduced timing graph* [ZTC+06] is used, with an example illustrated in Figure 2.2. In a reduced timing graph, a node represents a register, a primary input of the circuit, or a primary output. An edge represents the maximum delay between a pair of nodes, while only setup time constraints are considered. A reduced timing graph simplifies the corresponding sequential circuit by representing the combinational components

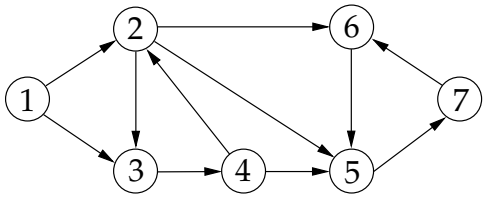


**Figure 2.2:** Example of Reduced Timing Graph

between nodes directly with delay edges. To compute these delays, combinational circuits between registers should be traversed. For such type of circuits, a *timing graph* is used to represent its structural timing properties. Figure 2.3 shows an example of the timing graph of the circuit c17 from ISCAS85 benchmarks. A node in a timing graph corresponds to a pin of a gate if interconnects are considered. Otherwise, a node corresponds to a net in the circuit, e.g., in Figure 2.3. Additionally, primary inputs and outputs are also represented by nodes. An edge represents the delay $W_{ij}$ between two nodes in the timing graph. In the following, primary inputs and outputs of a circuit will be called only inputs and outputs if no ambiguity is caused.

Notations concerning timing characteristics of circuit components, timing graph, and reduced timing graph are listed in Table 2.1. $\Delta_{ij}$ and $W_{ij}$ both represent delays. The difference is that $\Delta_{ij}$ is the maximum delay of combinational paths between two nodes; $W_{ij}$, however, is the delay of a combinational component, e.g., the pin-to-pin delay of a gate or the delay of an interconnect; additionally, $W_{ij}$ represents the delay of a single combinational path. These notations will be used in the rest of this thesis to explain timing analysis of flip-flop based and latch based circuits as well as timing model extraction.

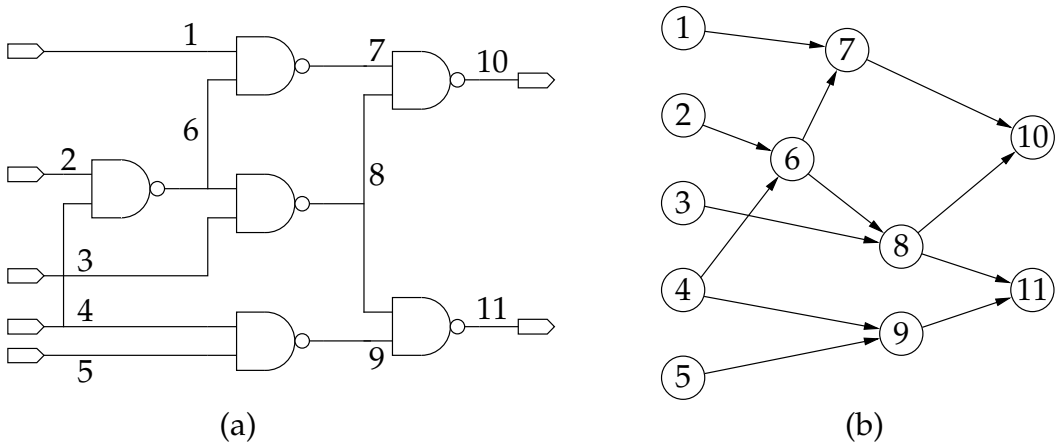


**Figure 2.3:** c17 Benchmark and Timing Graph

| | |
|---|---|
| $s_i$: | setup time of register $i$ |
| $q_i$: | propagation delay of register $i$ |
| $W_{ij}$: | delay between nodes $i$ and $j$ in timing graph |
| $\Delta_{ij}$: | maximum delay of combinational paths between $i$ and $j$ |
| $A_{ij}$: | latest arrival time from node $i$ to node $j$ |
| $A_j$: | latest arrival time to $j$ from all its fanin nodes |
| $D_i$: | latest departure time at latch $i$ |
| $\psi_i$: | set of all fanin registers of $i$ in timing graph or reduced timing graph |
| $\varphi$: | set of all registers in timing graph or reduced timing graph |
| $\phi$: | set of register pairs with combinational paths in the reduced timing graph |
| $T$: | clock period |
| $t_{c,n}$: | time of the $n$th latching clock edge |
| $r_i$: | time of the enabling clock edge of latch $i$ in local time zone |
| $\varepsilon_{ij}$: | clock phase shift of latch $i$ and $j$ |
| $m_{ij}$: | delay shift |

**Table 2.1:** Notation Definition for Timing Analysis

## 2.2 Timing of Flip-flop Based Circuits

Because of the simplicity of their design and verification, flip-flop based circuits are the most popular circuit type used in industry. The simplicity comes from the fact that a large circuit is split into small units by flip-flops. The output of each unit is used only at the latching clock edge.

Right after the $n$th active clock edge, a signal starts to propagate to the output of a flip-flop $i$ and further to the input of flip-flop $j$ at the next stage. The latest time that this signal reaches $j$ is $t_{c,n} + q_i + \Delta_{ij}$. The data change at the input of $j$ must meet its setup time constraint, so that

$$A_{ij} = t_{c,n} + q_i + \Delta_{ij} \leq t_{c,n+1} - s_j \tag{2.1}$$

For flip-flop $j$, there is normally more than one fanin node in the reduced timing graph. After a latching clock edge, data signals propagate from all these fanin nodes to $j$. Each arrival time must meet the setup constraint described in (2.1). Consequently, the maximum of these arrival times should meet the setup time constraint of $j$, i.e.,

$$\max_{i\in\psi_j}\{A_{ij}\} = \max_{i\in\psi_j}\{t_{c,n} + q_i + \Delta_{ij}\} \leq t_{c,n+1} - s_j \Longleftrightarrow \tag{2.2}$$

$$t_{c,n} + \max_{i\in\psi_j}\{q_i + \Delta_{ij}\} + s_j \leq t_{c,n+1} \Longleftrightarrow \tag{2.3}$$

$$\max_{i\in\psi_j}\{q_i + \Delta_{ij}\} + s_j \leq t_{c,n+1} - t_{c,n} = T \tag{2.4}$$

where $\psi_j$ is the set of all fanin nodes of $j$ in the reduced timing graph. Clock skew is not considered in (2.4) for simplicity. The constraint (2.4) should be met at all flip-flops in the circuit. With $\varphi$ defined as the set of all flip-flops, the setup time constraint for the circuit is

$$\max_{j\in\varphi}\{\max_{i\in\psi_j}\{q_i+\Delta_{ij}\}+s_j\}\leq T \tag{2.5}$$

The constraint (2.5) defines that the arrival time from any flip-flop node in the reduced timing graph to each of its sink nodes should meet the setup time constraint. With $\phi$ defined as the set of flip-flop pairs between each of which there is at least a combinational path, the constraint (2.5) is written as

$$\max_{(i,j)\in\phi}\{q_i+\Delta_{ij}+s_j\}\leq T \tag{2.6}$$

where $(i,j)$ denotes the flip-flop pair between node $i$ and $j$.

## 2.3 Timing of Latch Based Circuits

Because arrival times can propagate through latches, timing analysis of latch based circuits is more complex than that of flip-flop based circuits. When a data signal reaches the input of a latch during the active period of its clock, this signal can propagate through the latch instantly. This property is called latch transparency. With such property, the delay of a combinational path in a latch based circuit can be larger than the clock period [MS99]. This is different from the property of flip-flop based circuits, where any path delay between flip-flop pairs must be smaller than the clock period.

As for flip-flop based circuits, similar notations defined in Table 2.1 are used for latch based circuits. At each latch, the arrival time must meet the setup time constraint. The arrival time from a latch can start to propagate at any time when the clock signal is active, therefore, causes dependency between arrival times of latch stages. This is the source of the complexity of timing analysis for latch based circuits, and will be formulated in following.

To evaluate the timing performance of a latch based circuit, the complete timing constraints allowing multiphase clocks are specified in [SMO90b]. In this section, a short review of these timing constraints will be given. These constraints will be restructured to deduce the constraint set used for timing model extraction.

In timing analysis for latch based circuits, all arrival times are represented in the *local time zone* [MS99], i.e., relative to the starting time of the local time zone. In this

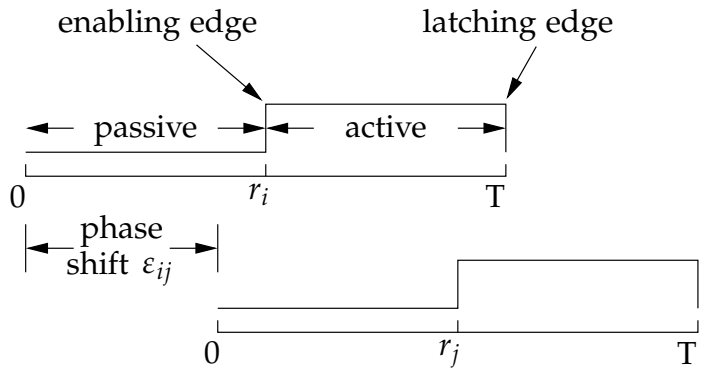


**Figure 2.4:** Local Time Zone and Clock Phase Shift

thesis, the active clock level of latches is 1, and the starting time of each local time zone is the time when the clock signal switches from 1 to 0. For two latches $i$ and $j$, where $i$ is a fanin of $j$, their clock phases are illustrated in Figure 2.4, where $\varepsilon_{ij}$ is the *phase shift* of the two clock phases, and will be used to transform an arrival time from the local time zone of $i$ to the local time zone of $j$.

Unlike at a flip-flop, where a data signal starts to propagate right after the latching clock edge, the time that a data signal at a latch starts to propagate to the next stage can be at any time when the clock is active. This time is called *departure time* and denoted as $D_i$. Similar to arrival time, $D_i$ is also defined in the local time zone of latch $i$, i.e., it uses the origin of the local time zone as reference time 0. If only the signal propagation from $i$ to $j$ is considered, the latest arrival time $A_{ij}$ is computed as

$$A_{ij} = D_i + q_i + \Delta_{ij} - \varepsilon_{ij} = D_i + m_{ij} \tag{2.7}$$

where $-\varepsilon_{ij}$ transforms $A_{ij}$ to the local time zone of $j$. $m_{ij}$ is the *delay shift* from $i$ to $j$. Considering all fanin latches $i$ of $j$, the arrival time $A_j$ can be computed as

$$A_j = \max_{i \in \psi_j} \{D_i + m_{ij}\} \tag{2.8}$$

The enabling clock edge of latch $i$ is denoted as $r_i$, also in the local time zone, as shown in Figure 2.4. Because a data signal can start to propagate to the next latch stage only after the enabling clock edge, the latest departure time of $i$ can be written as

$$D_i = \max\{A_i, r_i\} \tag{2.9}$$

By substituting $D_i$ in (2.8) with $\max\{A_i, r_i\}$, the departure time can be eliminated from (2.8),

$$A_j = \max_{i \in \psi_j} \{\max\{A_i, r_i\} + m_{ij}\} \tag{2.10}$$

The setup timing constraint for a latch $j$ is that the data signal at its input must be stable at least $s_j$ time before the latching clock edge. Therefore, the timing constraint for latch $j$ becomes

$$A_j \leq T - s_j \Longleftrightarrow A_j + s_j \leq T \tag{2.11}$$

Because $A_j$ depends on the arrival times of previous stages recursively, the constraint (2.11) for all latches can not be merged further like (2.6) for flip-flop based circuits.

## 2.4 Static Timing Analysis of Combinational Circuits

The performance of a sequential circuit is normally represented by the maximum clock frequency. This clock frequency is determined by the minimum clock period which can meet the timing constraint described in (2.6) for flip-flop based circuits, or the constraint (2.11) for latch based circuits. To verify these constraints, the maximum delays $\Delta_{ij}$ used in (2.6) and (2.11) should be computed first.

In contrast to sequential circuits, a combinational circuit consists of no storage components but only combinational gates. If a signal reaches an input of such a gate, it continues to propagate instantly and reaches the output of this gate after the time equal to the delay from the corresponding input to the output. Therefore, $\Delta_{ij}$ in a reduced timing graph is the maximum path delay from an input to an output of the combinational circuit.

To compute $\Delta_{ij}$ for register $i$ and $j$ in the reduced timing graph, the timing graph of the combinational circuit between the two latches should be traversed. Two types of traversal methods exist to compute the maximum delay of a combinational circuit: path-based and block-based. In a path-based method, the paths from inputs to outputs of the timing graph are enumerated [PS73, KYC+81, SYA+81]. The delay of a path is computed by summing the edge delays on the path together. Although this path enumeration method is feasible to evaluate small designs, it can not handle all paths in large designs, where the number of paths increases exponentially with circuit size.

The second method is the block-based method, or block-oriented method [Hit82, HSC82, MS99]. Regardless of computing the delay from one or more than one input of the circuit, a block-based method visits each node in the timing graph no more than once, so that the runtime is much shorter than that of the path-based method.

In Algorithm 1, the computation of maximum delays from all inputs are listed. As initialization (lines 4-12), the arrival times at inputs are set to given values determined by the application context. Thereafter, all nodes whose fanins are only inputs of the circuit are added to a node queue $\mathcal{Q}$. These nodes are candidates to be processed in the main loop, in the same order as they are appended.

**Algorithm 1**: Maximum Delay Computation from All Inputs

```
   // variables
1  Q: FIFO-like queue of nodes to be visited;
2  n_*: nodes;
3  A_*: arrival times;
   // algorithm initialization
4  foreach primary input n_i do
5      set arrival time of n_i;
6      mark n_i as visited;
7      foreach fanout node n_j of n_i do
8          if all fanin nodes of n_j are primary inputs then
9              append n_j to Q;
10         end
11     end
12 end
   // main loop
13 while Q is not empty do
14     n_i ← head node of Q;
15     A_i ← 0;
16     foreach fanin node n_j of n_i do
17         A_t ← A_j + W_ji;
18         A_i ← max{A_t, A_i};
19     end
20     mark n_i as visited;
21     foreach fanout node n_j of n_i do
22         if all fanin nodes of n_j are visited then
23             append n_j to Q;
24         end
25     end
26 end
```

In the main loop (lines 13-26) in Algorithm 1, the timing analysis is performed in an iterative way. In each iteration, a node is taken from the queue as the current node. The arrival time from each fanin node is summed with the edge delay between the fanin node and the current node. The arrival time of the current node is then calculated by the maximum of all the newly computed arrival times (lines 16-19). After the arrival time of a node is computed, its fanout nodes are checked. If a fanout node has no unvisited fanin node, this fanout node becomes a new candidate to be processed, and therefore is appended to the node queue. The main algorithm termi-

**Table 2.2:** Arrival Time Propagation of c17

| Iteration Number | Current Node | $\mathcal{Q}$ |
|---|---|---|
| initialization | | $n_6$, $n_9$ |
| 1 | $n_6$ | $n_9$, $n_7$, $n_8$ |
| 2 | $n_9$ | $n_7$, $n_8$ |
| 3 | $n_7$ | $n_8$ |
| 4 | $n_8$ | $n_{10}$, $n_{11}$ |
| 5 | $n_{10}$ | $n_{11}$ |
| 6 | $n_{11}$ | |

nates when the node queue is empty, which means all nodes are already visited and all the arrival times at outputs of the circuit are already computed. The maximum delay of the circuit is the maximum arrival times at all these outputs.

To illustrate Algorithm 1, the timing graph in Figure 2.3 is used as an example. The content of the node queue and the current node are shown in Table 2.2, where the last column shows the nodes in $\mathcal{Q}$ after processing the current node.

From this example, it can be observed that the nodes are processed in the order of topological level. The topological level of a node is defined as the maximum of all the levels of its fanin nodes plus 1. From this sense, the algorithm shown in Algorithm 1 is a breadth-first traversal [Knu98]. A variant of this algorithm is explained in [SYA+81, Hit82] to traverse the nodes in depth-first order. In this algorithm, the fanout node is checked if all its fanin nodes are visited. If this is true, this fanout node is directly processed, in contrast to appending it to the node queue for later processing in Algorithm 1.

The algorithm described till now can handle timing analysis from all inputs of the circuit efficiently. After the algorithm terminates, the arrival times at an output is the maximum delay from all inputs to it. This information is enough for timing analysis of flip-flop based circuits, described in Section 2.5 later. However, the maximum delay between each register pair is required to analyze the timing of latch-based circuits in Section 2.6. For a combinational block between latches, this requires arrival time propagation from each input separately.

In the main loop of Algorithm 1, a fanout node is appended to the queue only when all its fanin nodes are already visited. If the arrival time is propagated from only one input, some nodes with fanin node driven only by other inputs can never be processed. As an example the circuit in Figure 2.3 is used to explain such problem. If the delay from node 1 to node 10 is needed, node 7 should be processed directly after node 1. But node 7 has unvisited fanin node 6, so that the propagation can not be performed further. A bypass of this problem is to set the arrival times of unrelated inputs of the circuit to $-\infty$ [Fis90]. But this method is slow because the

**Algorithm 2**: Maximum Delay Computation from Primary Input $n_p$

```
   // variables
 1 Q_k: FIFO-like node queue for level k;
 2 L_m: maximum level number;
 3 n_*: nodes;
 4 A_*: arrival times;
   // algorithm initialization
 5 for k ← 1 to L_m do
 6 |  empty Q_k;
 7 end
 8 append n_p to Q_1;
 9 set A_p to predefined value;
   // main loop
10 for k ← 1 to L_m do
11 |  while Q_k is not empty do
12 |  |  n_i ← head node of Q_k;
13 |  |  if n_i is not a primary input then
14 |  |  |  A_i ← 0;
15 |  |  end
16 |  |  foreach fanin node n_j of n_i do
17 |  |  |  if n_j is visited then
18 |  |  |  |  A_t ← A_j + W_ji;
19 |  |  |  |  A_i ← max{A_t, A_i};
20 |  |  |  end
21 |  |  end
22 |  |  mark n_i as visited;
23 |  |  foreach fanout node n_j of n_i do
24 |  |  |  l ← level of n_j;
25 |  |  |  if n_j is not in Q_l then
26 |  |  |  |  append n_j to Q_l;
27 |  |  |  end
28 |  |  end
29 |  end
30 end
```

complete timing graph is traversed although only the nodes in the fanout cone of the input of interest affect the delay. These nodes are normally only a small part of the complete circuit [CSH95]. The method proposed in [Sap96] uses this observation to propagate arrival times only through nodes in the timing graph when necessary.

**Table 2.3:** Arrival Time Propagation from $n_3$ in c17 Timing Graph

| Iteration number | Current level | Current node | $\mathcal{Q}_1$ | $\mathcal{Q}_2$ | $\mathcal{Q}_3$ | $\mathcal{Q}_4$ |
|---|---|---|---|---|---|---|
| initialization | | | $n_3$ | | | |
| 1 | 1 | $n_3$ | | | $n_8$ | |
| 2 | 3 | $n_8$ | | | | $n_{10}$, $n_{11}$ |
| 3 | 4 | $n_{10}$ | | | | $n_{11}$ |
| 4 | 4 | $n_{11}$ | | | | |

The complete arrival time propagation from a specified input in [Sap96] is shown in Algorithm 2. Compared to Algorithm 1, only the nodes driven by the input $n_p$ in the timing graph is traversed. These nodes are appended to their level queues when one of their fanin nodes is visited. The arrival time of a node is computed directly by maximizing the arrival times of its fanin nodes which are already visited. Because the queues are visited in level order (line 10), a fanin node is not in the fanout cone of the input if it is not visited. Therefore it does not affect the arrival time of the current node (line 17). Similar to Table 2.2, an example of applying Algorithm 2 to compute delays from node 3 in Figure 2.3 is shown in Table 2.3.

By initializing more than one input in Algorithm 2, this algorithm can be extended to compute maximum delays from these inputs at once. If all inputs are initialized, Algorithm 2 becomes a variant of the static timing algorithm shown in Algorithm 1.

The arrival time traversal in Algorithm 1 and 2 both can run in the reverse direction, from outputs to inputs of the circuit. The results are maximum delays from outputs to all internal nodes in the timing graph. These delays can be used to compute the slacks at internal nodes for circuit optimization.

## 2.5 Static Timing Analysis of Flip-flop Based Circuits

For flip-flop based circuits, the clock frequency is determined by the maximum delay between all flip-flops as specified in (2.5), where the inner maximum is performed with all fanin nodes in the reduced timing graph. Instead of computing the maximum delay $\Delta_{ij}$ between flip-flop $i$ and $j$ individually, the inner maximum in (2.5) is computed by one arrival traversal using Algorithm 1. For this purpose, a virtual combinational circuit is formed. All outputs of flip-flops are considered as primary inputs of the virtual circuit, and all inputs of flip-flops as primary outputs. All the combinational components between flip-flops together form the combinational logic in between.

During the initialization of Algorithm 1, the arrival times at primary inputs of the virtual combinational circuit are set to the propagation delays of the corresponding

registers. The resulting arrival times at the primary outputs of the virtual circuit are maximum arrival times from all primary inputs. In other words, the arrival time at a primary output is the maximum delay from all fanin flip-flops to the input of a flip-flop, i.e., the result of the inner maximum in (2.5). Therefore, the left side of (2.5) can be computed by the maximum of the sums of the arrival time and the setup time at all flip-flops. This maximum specifies the minimum clock period for the flip-flop based circuit without timing violation. The complete computation of the minimum clock period is shown in Algorithm 3, where Algorithm 1 in line 3 can be replaced by other propagation algorithms, e.g., Algorithm 2 with all primary inputs initialized.

**Algorithm 3**: Timing Analysis of Flip-flop Based Circuits

```
   // variables
1  A_j: arrival times at the input of flip-flop j;
2  T_Min: minimum clock period;
   // arrival time propagation
3  Run Algorithm 1 on the virtual circuit, with q_i as initial arrival times;
   // minimum clock period computation
4  T_Min ← 0;
5  foreach flip-flop n_j in the reduced timing graph do
6  |  A_t ← A_j + s_j;
7  |  T_Min = max{T_Min, A_t};
8  end
```

## 2.6 Static Timing Analysis of Latch Based Circuits

Unlike flip-flop based circuits, the minimum clock period can not simply be identified for latch based circuits. This is because there is latch transparency, which makes the arrival times depend on each other, as described with (2.10). Instead of computing the minimum clock period directly, the method proposed in [SMO90b] describes this as an optimization problem, as shown in Algorithm 4. The variables in this optimization problem are the arrival times for all latches. The constraints are from the relation between these arrival times specified by (2.10) and (2.11).

**Algorithm 4**: Minimum Clock Period of Latch Based Circuits

```
Minimize T;
Subject to (2.10) and (2.11) for all arrival times;
```

The optimization problem in Algorithm 4 is nonlinear because of the maximum computation in (2.10). In order to reduce the optimization complexity, the constraints (2.10) and (2.11) are relaxed to (2.12)-(2.14) in [SMO90b].

$$
\begin{aligned}
&for\ i \in \psi_j : \\
&A_j \geq A_i + m_{ij} && (2.12)\\
&A_j \geq r_i + m_{ij} && (2.13)\\
&m_{ij} = q_i + \Delta_{ij} - \varepsilon_{ij} && (2.14)
\end{aligned}
$$

Consequently, the optimization problem of Algorithm 4 can be rewritten as Algorithm 5.

| **Algorithm 5**: Minimum Clock Period with Constraint Relaxation |
|---|
| Minimize $T$; |
| Subject to (2.11)-(2.14) for all arrival times; |

It is proved in [SMO90b] that the solution of the relaxed problem is equal to the one from Algorithm 4. With this constraint relaxation, the optimization problem becomes linear. Therefore it can be solved with standard linear programming methods, e.g. simplex method [Dar91]. The optimal result is the minimum clock period for the latch based circuit without violating the setup time constraints. In the relaxed optimal problem, the constraints from hold time are not considered. This problem is addressed by the method proposed in [SMO90a] based on sensitivity analysis of linear programming.

The constraints (2.12)-(2.14) contain edge delays $\Delta_{ij}$ between latches. Similar to the method in Section 2.5, a virtual combinational circuits is created. Because the maximum delay of each latch pair is needed, Algorithm 2 instead of Algorithm 1 is used to propagate arrival times from each primary input of the virtual circuit. After each timing traversal, the arrival times at all primary outputs which are visited are the maximum delays from the corresponding latch to all its fanout latches respectively. All edge delays in the reduced timing graph are computed after repeating the computation from all latches. Thereafter the linear programming method can be performed to determine the minimum clock period of the latch based circuit. The step to empty all queues in lines 5-7 of Algorithm 2 guarantees the correct results after applying this algorithm to all inputs successively.

## 2.7 Summary

Timing performance determines the proper behavior of a digital circuit. In order to reduce the runtime of the timing evaluation of such a circuit, static timing analysis is

used. For flip-flop based circuits the delays between flip-flops can be computed by applying block-based or path-based timing analysis methods to the combinational circuits between all the flip-flops. The clock period of the circuit is determined by the resulting delays and the corresponding setup time and hold time constraints. Because of the interdependency between arrival times at successive latches, timing analysis of latch based circuits relies on more complex methods, e.g., linear programming, to determine the constraint for the clock period. These static timing analysis methods form the background of the timing analysis considering process variations in the following chapters.

# Chapter 3

# Problem Description

Traditional static timing analysis evaluates timing of a circuit by setting process parameters to their extreme (e.g., $3\sigma$) values, called corners. For example, the worst-case corner for checking the setup time constraint is the corner that all the process parameters are set so that gate delays and interconnect delays are the largest in all corners. With increasing process variations, this corner-based method faces new challenges, not only because the problem in selecting proper corners from the large parameter space, but also because it is over conservative in evaluating circuit performance [Sch02]. In this chapter variations are discussed at first. Thereafter an introduction to statistical timing analysis is given, where process variations are directly modeled to produce the yield and performance curve. Timing model extraction methods for static timing analysis without considering process variations are discussed later. As an overview, this discussion explains the steps in hierarchical timing analysis flow. These steps are the same as in the proposed methods in this thesis, where process variations and their correlation are handled. In the last part of this chapter, the state of the art of hierarchical statistical timing analysis is reviewed.

## 3.1 Variations

Process variations exist since the beginning of semiconductor industry. These variations are defined as the deviations of process parameters after manufacturing from design values. The source of variations lies in the limitation of process control. Examples of such variations are the inaccuracy in creating device shapes at photolithography step because of diffusion effect, the doping density fluctuation because of time control, and the interconnect thickness variation because of interconnect and device patterns during chemical-mechanical planarization (CMP) process. As the feature size scales into deep submicron realm, the relative process variations

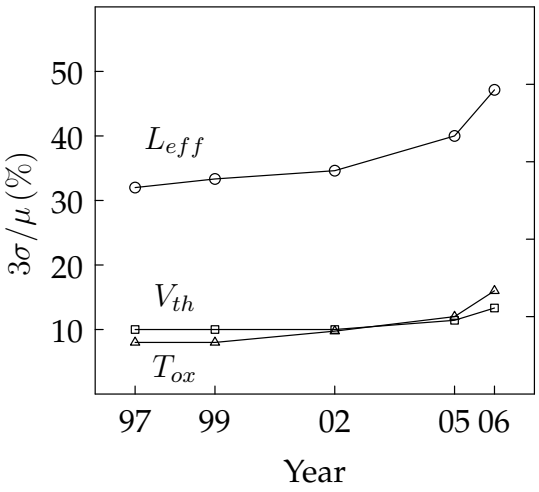


**Figure 3.1:** Relative Variation Increase, data from [Nas01]

become worse than at the earlier technology nodes. This relative variation can be defined as the ratio of three times the standard deviation of a process parameter to its mean value, i.e., $3\sigma/\mu$. According to [Nas01], the trends of increasing relative variations of effective gate length ($L_{eff}$), oxide thickness ($T_{ox}$) and threshold voltage ($V_{th}$) are shown in Figure 3.1.

Facing the increasing process variations, the traditional worst-case design method becomes too conservative. In this method, all parameters are set so that the delays of circuit components, e.g., gates and interconnects, are the worst in all corners. Circuits are designed according to the results of this worst-case performance evaluation. If the worst-case performance of a circuit can meet corresponding specification, the correct timing of chips after manufacturing can be guaranteed. In practice, however, process parameters of different gates and interconnects after manufacturing are not at the worst-case corner at the same time. For example, process parameters are not fully correlated according to the measurements in [CCBC06]. Therefore the circuit performance is underestimated by the worst-case design method. This underestimation becomes larger at newer technology nodes because the worst-case corner deviates further from the nominal one.

As the relative process variations increase, the worst-case design becomes too expensive because the underestimation of the performance causes unnecessary further optimization of circuits which may have met timing constraints after manufacturing. Such unnecessary further optimization is also called overdesign. To reduce the conservatism in the worst-case design and analysis, the first step is to investigate and model process variations more accurately than simply using worst-case or bounding methods. With the results of the variation analysis, both design and manufacturing steps can be optimized to improve yield.

### 3.1.1 Sources of Variations

In general process variations come from the limitation of process control. At each step of the manufacturing process, different factors affect peculiar characteristics of wafers as well as dies on them. According to manufacturing steps of integrated circuits, variation sources can be analyzed from device and interconnect sides. In this section, some sources which cause variations are explained.

The geometries of devices are determined during the photolithography step, where the gate length dimension affects delay dominantly, so that is called critical dimension (CD). The accuracy of the mask is the first source affecting the dimensions of transistors, because any inaccuracy and fluctuation during mask creation keeps its effects to final device dimensions; additionally, uncertainty is introduced when optical proximity correction (OPC) is applied to correct the dimensions of devices; the nonuniformity of lens used in photolithography also imposes some variations on the shape of devices. Another factor affecting devices is the variation of film thickness, which is caused by the variations in oxide film coating. Because of the fluctuation of implant dose, energy or angle of doping, the depth of dopant profiles can also be a distribution. This may have impact on effective gate length and threshold voltage as well [BN99]. Because electrical parameters depend on physical ones of transistors, the variations discussed above cause electrical parameters to deviate from their nominal values [BCSS08]. For example, threshold voltage ($V_{th}$) exhibits variations because it depends on film thickness, doping density and device geometries.

Like devices, the geometries of interconnect exhibit variations resulting from photolithography process. The accuracy of mask affects the line width and line space. The ensuing variations make electrical characteristics of interconnects, such as capacitance and cross talk, deviate from design specifications. The thickness of metal lines are affected by the fluctuations during CMP process. Additionally, the final thickness of interconnects as well as inter layer dielectrics differs in relation to the patterns of interconnects in different areas on the die. For example, less line dishing happens in the area with small pitch size than in the area with larger one [PTB+99].

### 3.1.2 Decomposition of Process Variations

Process variations are grouped into different categories, as shown in Figure 3.2. Systematic variations can be determined before manufacturing. Once physical synthesis is finished, the source of these variations can be determined. With accurate measurement, they can be modeled with fixed values and included into performance analysis. A typical example of systematic variations is the randomness of interconnect metal thickness. After layout and routing are finished, the patterns of

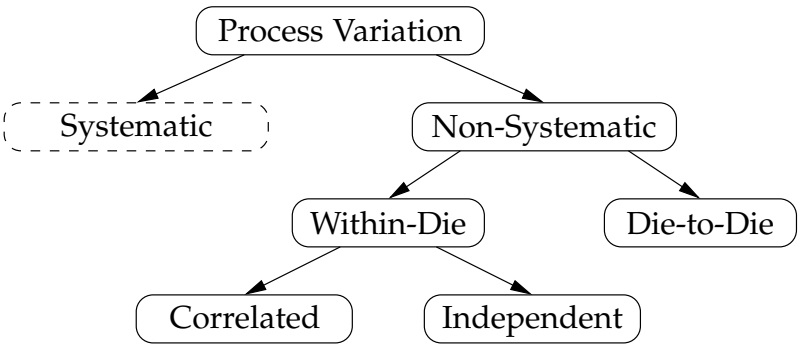


**Figure 3.2:** Variation Classification [BCSS08]

interconnects can be accurately analyzed. Therefore, the layout-related metal thickness dishing in different areas can be predicted. With this information, the resistance and capacitance of interconnects can be modeled more accurately in sign-off analysis. From device side, gate length is affected by variations in the step of optical proximity correction for mask optimization. These variations can be determined by computing the post-OPC gate lengths on the critical path thus more accurate timing analysis results can be achieved [YCS05]. In both cases, systematic variations are represented using fixed values instead of statistical variables. This is more accurate than simply analyzing circuit performance with variation assumption. Both measurements, however, can be fulfilled only after physical synthesis. During the first iteration of logic synthesis, the circuit can only be optimized corresponding to the performance from modeling systematic variations as random variables. Thereafter, the accuracy improvement by determining systematic variations can only happen in further design iterations

Unlike systematic variations, non-systematic variations can not be determined before manufacturing. These variations come from the inaccuracy of process control and are independent to circuit design. Therefore, they can only be modeled with random variables in the complete design flow. Examples are variations in doping density and in layout independent metal thickness of interconnects.

According to their spatial characteristics, non-systematic variations can be further partitioned into die-to-die variations (interdie variations) and within-die variations (intradie variations). Die-to-die variations affect all devices and interconnects on a die equally, i.e., all devices and interconnects have fully correlated random components. At wafer level, die-to-die variations come from the nonuniformity of process control across wafer surface. Therefore, chips on different positions of a die have different performances. For example, the chips in the center of a wafer are normally faster than the chips near the boundary of the wafer, because of better process control when the chips in the center are processed. Within-die variations affect devices or interconnects inside a die differently. For two devices their physical parameters can shift in different directions, i.e, they are not fully correlated. Within-die varia-

tions come from the inaccuracy of process control at die level. For example, there is still a variation residue after modeling the systematic and die-to-die variations of CD on a chip.

Furthermore, within-die variations can be partitioned into a correlated part and an independent part. Although within-die variations on devices or interconnects are not fully correlated, they still show a similar trend in some degree. This trend can be modeled by sharing the same variables as a part of within-die variations, or by establishing correlation between these variations directly. Besides the correlated variation component, within-die variations still exhibit purely random effect. The purely random variations come from the random fluctuation during manufacturing processes, which thus imposes its effect on each device without correlation. Because of the inaccuracy of manufacturing equipments and process control, purely random variations exist nearly in every processing step. Examples are the random distortion of the lens used during the photolithography step and the purely random variation of the doping speed.

### 3.1.3 Correlation Modeling

Process variations can normally be measured as a lumped distribution. Thereafter, the measured data are decomposed into different components [SBC97]. The overall variations are then modeled as sums of these decomposed variables. Die-to-die variations are shared by all devices or interconnects on the chip. These variations make parameters of the devices and interconnects exhibit some correlation, called global correlation or die-to-die correlation. Because of the uncertainties during manufacturing process vary continuously, within-die variations exhibit proximity correlation. This correlation depends on the distance between two devices on the die [CCBC06]. The larger the distance is, the smaller the correlation becomes. For convenience, the correlation from within-die variation is called local correlation.

Different methods are proposed to model correlation between process parameters. Modeling die-to-die correlation accurately, the quadtree model is proposed in [ABZ03a, ABZ+03b], illustrated in Figure 3.3. In this model, different grid layers are used to model correlation between process parameters. For a process parameter, e.g., gate length, a variable is assigned to each grid cell at each level. The process parameter of a device is modeled as the sum of all the variables of the grid cells directly above this device. The correlation between process parameters is therefore established by sharing the same variables of the corresponding levels. Because the variable at level 0 is shared by all devices, it models the correlation from die-to-die variation. The local correlation is modeled by sharing the variables at lower levels. If two devices are nearby on the die, they share more variables so that have more correlation. If two devices are near enough to be located in the same grid cell at level 2, they become

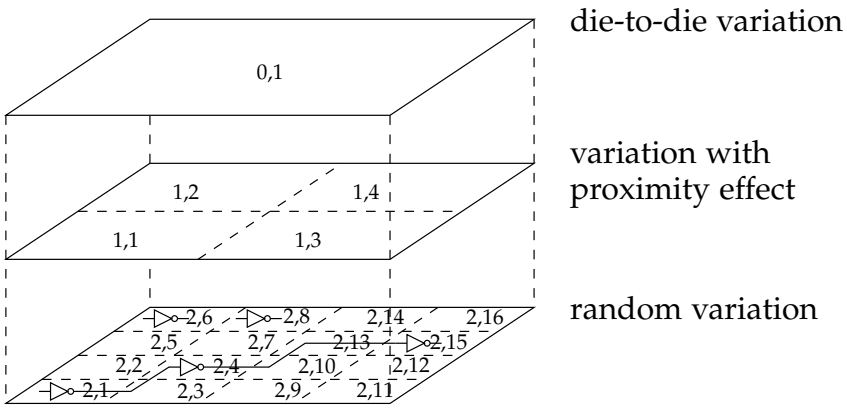


**Figure 3.3:** Quadtree Correlation Model [ABZ+03b, ABZ03a]

fully correlated. By increasing the number of grid layers, the accuracy of correlation modeling can be increased. This model, however, has an apparent limitation. By partitioning each layer into different grid cells, the local correlation can not be modeled uniformly. For example, the distances from (2,4) to (2,1) and from (2,4) to (2,13) are equal. From this model, a parameter in (2,4) and (2,1) share the same variable at layer 1, but the same parameter in (2,4) and (2,13) do not share such variable. Consequently, correlations between parameters with the same distance are unequal in this model. This contradicts the fact that within-die correlation depends on distance between devices because of the proximity effect during manufacturing process.

The second correlation model is proposed in [CS03]. In this model, the die area is partitioned into a uniform grid, as shown in Figure 3.4. For each grid cell, a random variable is assigned. The correlations between these random variables are computed or identified from the characterization of manufacturing technology, for example with the method in [XZH07]. For $n$ grid cells on the die, in total $n$ variables are assigned. For the convenience of statistical timing analysis algorithms, the $n$ correlated variables are decomposed into linear combinations of independent random variables, using an algorithm such as principal component analysis (PCA) [Jol02]. After this decomposition, only the independent variables with large coefficients are kept in the linear combinations, so that the number of variables modeling correlation can be drastically reduced. This correlation model is very flexible because it can handle any correlation between process parameters. The only reason to partition the die area to grid is to reduce the number of initial variables. For better modeling accuracy smaller cell size can be used, at the expense of larger number of variables and larger correlation matrix. A similar correlation model is proposed in [CZV+08], where hexagonal grid cells are used to partition the die area. The advantage of such a model is that a grid cell in the partition has only one type of neighboring cell. Additionally, the distances from the neighbors of a cell to it are equal. This makes the hexagonal partition a better approximation in modeling proximity related cor-

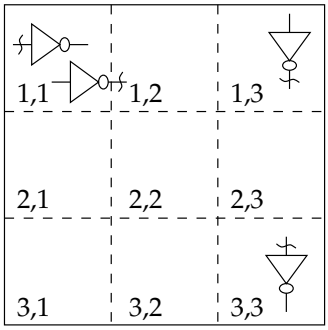


**Figure 3.4:** Uniform Grid Correlation Model [CS03]

relations.

Another correlation model is proposed in [KS05]. In this model, the die area is partitioned into grid with square cells. A process parameter in a grid cell is modeled as the sum of independent variables assigned to the corners of the grid cell. That is, each process parameter is decomposed into a linear combination of four independent random variables. This method can generate simple parameter decomposition, but no theoretical proof is provided for accuracy. Additionally, the method to map correlation data to the proposed model is not explained.

The correlation in the discussed models are all first-order. This is only enough to model the dependency between Gaussian random variables. To incorporate higher order dependency, methods like independent component analysis [HKO01] should be used, e.g., in [SS06, SS08].

### 3.1.4 Process Variation Handling

As pointed out in [BBC+08], methods should be deployed to handle increasing process variations. At the center of these methods, statistical metrology measures and analyzes process variations to generate corresponding models. With this information, methods can be applied during manufacturing to reduce deviations. For example, the results of fab-to-fab and lot-to-lot variation analysis expose the deviations of process control. Therefore they can provide indications for adjusting operation parameters of equipments. At design stage, variations should be modeled directly, in contrast to the traditional worst-case analysis. With the statistical analysis results, circuit components which are statistically critical to performance can be identified, thus providing more accurate candidates for optimization. Beyond statistical analysis and optimization, a further step is variation resistant design. An example of such design is proposed in [BBC+08], where the irregularity of layout is reduced by inserting dummy fill structures. Consequently, variations such as resulting from interconnect dishing during CMP process can be reduced.

## 3.2 Statistical Timing Analysis

Timing performance of circuits is an important measurement for optimization and sign-off. With process variations considered as variables, all gate delays become random variables. The timing graph traversal algorithms described in Chapter 2 can be adapted to compute the minimum clock period of a circuit. However, the resulting clock period is a random variable, denoted as $T_{min}$. For a given clock period $T$, the timing yield of a circuit is evaluated by computing the probability that $T_{min}$ is smaller than $T$, i.e.,

$$yield = Prob\{T_{min} \leq T\}, \quad 0 < T < \infty \tag{3.1}$$

where $Prob\{\cdot\}$ denotes the probability.

Because all gate delays are positive, the computed minimum clock period $T_{min}$ is also positive. According to probability theory, e.g., [FF91], yield computation in (3.1) is equivalent to the definition of *cumulative distribution function* (CDF) of the random variable $T_{min}$. The graphic representation of (3.1) is illustrated in Figure 3.5, where circuit yield approximates 0 when $T$ approximates 0, and 1 when $T$ is large enough. The latter case indicates that a sequential circuit can work properly at a reasonable low clock frequency, if no hold time constraint is violated. In this section, methods for process variations modeling, gate delay mapping and algorithms for analyzing circuit performance will be explained.

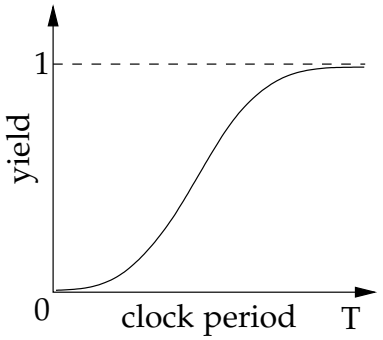


**Figure 3.5:** Graphic Representation of Yield Computation

### 3.2.1 Process Parameter Modeling

The first step of statistical timing analysis is to model process variations in a form which can simplify modeling of gate delays and arrival time propagation. As described in Section 3.1, a process parameter is a sum of components modeling die-to-die variations, within-die variations and purely random variations. The additive

attribute is determined by applied variation decomposition methods, e.g., [SBC97]. The additive form of a process parameter $p$ is written as

$$p = p_0 + p_g + p_l + p_r \tag{3.2}$$

where $p_0$ is the nominal value of the parameter. $p_g$ models the die-to-die variation and is shared by all gates. $p_l$ is the within-die variation specific to each gate and is correlated with each other. $p_r$ is an independent variable modeling the purely random effect in manufacturing processes.

Depending on statistical measurements, the parameter $p$ for a device may have Gaussian or non-Gaussian variations. In [CS03, VRK+04, KPR05], all process variations are assumed as Gaussian in order to reduce the complexity of timing analysis. The Gaussian assumption, however, can not provide enough accuracy because only the first two moments of process parameters are captured with Gaussian variables. To improve modeling accuracy, non-Gaussian variables are used in [CZNV05]. Additionally, the independent component analysis based non-Gaussian model is proposed in [SS06, SS08]. In both methods, the random variables representing process variations can be in any form in addition to Gaussian.

### 3.2.2 Gate Delay Representation

Statistical timing analysis uses abstracted gate delays to evaluate circuit performance. A gate delay is defined as the time difference between points of measurement of the input and output waveforms. With given input waveform, the output waveform of a gate depends on transistor parameters of the gate. For example, the effective gate length affects the gate delay dominantly. Assuming all process parameters denoted as a vector $\boldsymbol{p}$, a gate delay $W$ is expressed as

$$W = f(\boldsymbol{p}) \tag{3.3}$$

where $f$ denotes the mapping function from process parameters to the gate delay. The mapping function is theoretically very complex. Therefore SPICE simulation is often used to obtain accurate samples of gate delays. To provide fast delay estimation, different models, such as Elmore delay [Elm48], are used to compute gate delays from lower level parameters [RPH83].

With process variations considered, a gate delay becomes a random variable. Because of the correlation between process variations, gate delays are correlated with each other. For example, the delays of two gates vary in a similar way when these gates are near on the die. When their distance is large, both gate delays exhibit more randomness. In order to incorporate the correlation from process variations, gate delays are described as simplified functions of process parameters, instead of

identifying the numeric characteristics of their distributions directly. In other words, the mapping function $f$ in (3.3) is replaced with a simpler form at the expense of accuracy.

The first popular delay description method uses linear functions [CS03, VRK+04]. A gate delay in this method is expressed as

$$W = \boldsymbol{k}\boldsymbol{p} \tag{3.4}$$

where $\boldsymbol{k}$ is the coefficient vector and can be computed by sensitivity analysis numerically [ABZ+03b], or identified by linear regression [Seb77] from the results of SPICE based Monte Carlo simulation.

According to (3.2) a parameter can be partitioned into different parts. If each variable in (3.4) is replaced into the form of (3.2), the gate delay is transformed as

$$W = \boldsymbol{k}\boldsymbol{p}_0 + \boldsymbol{k}\boldsymbol{p}_g + \boldsymbol{k}\boldsymbol{p}_l + \boldsymbol{k}\boldsymbol{p}_r = W_0 + \boldsymbol{k}\boldsymbol{p}_g + \boldsymbol{k}\boldsymbol{p}_l + p_\tau \tag{3.5}$$

In (3.5) $\boldsymbol{p}_0$ represents nominal values of parameters and all its elements are fixed, so that $\boldsymbol{k}\boldsymbol{p}_0$ can be merged into a constant $W_0$. Because the first order moments are merged into $W_0$, the means of $\boldsymbol{p}_g$, $\boldsymbol{p}_l$ and $p_r$ are all zero. Representing die-to-die variations, $\boldsymbol{p}_g$ is shared by all gate delays. $\boldsymbol{p}_r$ models purely random manufacturing effects, so that it can be merged into one random variable $p_\tau$. Unlike the other vectors in (3.5), $\boldsymbol{p}_l$ models within-die variations and needs further processing.

As discussed in Section 3.1.3, correlation exists between within-die variables because of proximity effects. Consider two gate delays $W_a$ and $W_b$,

$$W_a = W_{0,a} + \boldsymbol{k}_a\boldsymbol{p}_g + \boldsymbol{k}_a\boldsymbol{p}_{l,a} + p_{\tau,a} \tag{3.6}$$

$$W_b = W_{0,b} + \boldsymbol{k}_b\boldsymbol{p}_g + \boldsymbol{k}_b\boldsymbol{p}_{l,b} + p_{\tau,b} \tag{3.7}$$

where $\boldsymbol{k}_a\boldsymbol{p}_{l,a}$ and $\boldsymbol{k}_b\boldsymbol{p}_{l,b}$ are correlated random variables. During arrival time propagation, these random variables can not be merged because of their correlation. Therefore, the number of variables describing an arrival time may increase very fast if the gate delays in (3.6) and (3.7) are directly used to evaluate the performance of a circuit. Additionally, the correlation between $\boldsymbol{p}_{l,a}$ and $\boldsymbol{p}_{l,b}$ also causes the computation of the correlation between $W_a$ and $W_b$ to be very slow, as explained later.

In order to reduce the runtime of timing analysis, principal component analysis [Jol02] is used to decompose correlated random variables. Assume that variable vector $\boldsymbol{p}_l$ with $m$ elements is the vector containing all the correlated random variables modeling within-die process variations so that $\boldsymbol{p}_{l,a}$ and $\boldsymbol{p}_{l,b}$ both are parts of $\boldsymbol{p}_l$. The correlation matrix of $\boldsymbol{p}_l$ is denoted as $\boldsymbol{C}$. Under Gaussian assumption, each element in $\boldsymbol{p}_l$ can be expressed as a linear combination of a set of independent components after applying PCA.

$$\boldsymbol{p}_l = \boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{A}^r\boldsymbol{x}^r \tag{3.8}$$

where $\boldsymbol{A}$ is the orthogonal transformation matrix formed by the eigenvectors of $\boldsymbol{C}$. $\boldsymbol{x} = [x_1, x_2, \ldots x_m]^T$ is a vector of independent Gaussian random variables with zero mean. The standard deviation vector of $\boldsymbol{x}$ is formed by the square root of eigenvalues of $\boldsymbol{C}$ corresponding to the eigenvectors in $\boldsymbol{A}$. If there are eigenvalues which are very small compared to other larger eigenvalues, the corresponding variables in $\boldsymbol{x}$ contribute relative less than other variables in (3.8). Therefore, these variables can be discarded to reduce the number of the independent variables. Assume $\boldsymbol{x}$ is truncated to $\boldsymbol{x}^r$ with $n_r$ variables. The original within-die variations can be approximated by linear combinations of the $n_r$ independent random variables $\boldsymbol{x}^r$. $\boldsymbol{A}^r$ is a column truncated matrix of $\boldsymbol{A}$.

Because any random variable from $\boldsymbol{p_l}$ can be approximated by a linear combination of $\boldsymbol{x}^r$ by selecting the row of $\boldsymbol{A}^r$ corresponding to the random variable, as shown in (3.8), the gate delay in (3.5) can be written as

$$W = W_0 + \boldsymbol{k}\boldsymbol{p}_g + \boldsymbol{k}\boldsymbol{A}_s^r\boldsymbol{x}^r + p_\tau \tag{3.9}$$

$$= c_0 + \sum_{i=1}^{n} c_i v_i + c_r v_r \tag{3.10}$$

where $\boldsymbol{A}_s^r$ is formed by the rows of $\boldsymbol{A}^r$ corresponding to the variables in $\boldsymbol{p}_l$ in (3.5). The gate delay in (3.9) can be generalized into the canonical linear delay form [VRK+04] as in (3.10), where $v_i$ are independent random variables and shared by all gate delays. $v_r$ is the purely random variably specific for each delay. $c_0$ is the nominal value of the delay. $c_i$ and $c_r$ are the coefficients of the random variables. The correlation between gate delays are represented by sharing the same set of random variables $v_i$.

In the canonical delay model (3.10), the mapping function $f$ from parameters to delays is assumed as linear. With such linear delay form, arrival times can be propagated very fast with simple computations [VRK+04]. The drawback of this simple assumption, however, is the loss of accuracy [LLGP04, ZSL+05]. To improve approximation accuracy, quadratic timing models are proposed in [ZSL+05, ZCH+05, KS05], where a gate delay is mapped as a second order function of process parameters. If principal component analysis is still used to decompose correlated random variables, a parameter in the quadratic form is replaced by a linear combination of uncorrelated random variables. For a second order term in the quadratic form, this replacement results in many cross terms, which make timing analysis complex and slow. To reduce the number of cross terms in a quadratic model, orthogonalization method is used in [ZSL+05]. In addition to quadratic models, a gate delay is mapped as a linear function of independent Gaussian and non-Gaussian variables in [SS06, SS08]. A more general delay mapping method is proposed in [CZNV05]. In this model, a gate delay is mapped as a sum of linear and nonlinear functions of Gaussian and non-Gaussian random variables. Therefore, it can handle any delay functions without limitation. Using non-Gaussian random variables can improve

the modeling accuracy of process variations; using non-linear functions can improve the accuracy of approximating the mapping from process parameters to gate delays and interconnect delays. Both methods, however, cause complexity in the following steps of statistical timing analysis.

### 3.2.3 Statistical Timing Analysis of Combinational Circuits

Similar to static timing analysis, the target of statistical timing analysis is to compute the timing performance of the circuit. With all gate delays modeled as functions of random variables, arrival times are propagated across the circuit using the algorithm described in Section 2.4. In this propagation, two computations are involved: maximum and sum. When multiple edges converge to a node, the maximum of the incoming arrival times is computed. Thereafter, this arrival time is propagated to the next node by adding the edge delay. In statistical timing analysis, this timing graph traversal is completely the same as in static timing analysis. The two computations, however, must be adapted to handle random gate delays.

In statistical timing analysis, arrival times are computed from gate delays. In order to use the same sum and maximum computations at all nodes, arrival times are usually represented in the same form of gate delays. When an arrival time is added with a gate delay, corresponding coefficients of different variables are summed directly, whether linear or quadratic gate delays are used. Because of the complexity in computing the maximum of two random variables and the requirement that the result of the maximum should have the same form as a gate delay, such computation is always approximated in statistical timing analysis. In the following, only the sum and maximum computations of two random variables are discussed. The complete statistical timing analysis can be implemented by replacing the sum and maximum computations in Algorithm 1 or 2.

Using the linear delay model, [VRK+04] introduces an arrival time propagation method which can process the maximum computation efficiently, meanwhile keeping the correlations between arrival times accurately. Consider two random variables $A$ and $B$

$$A = a_0 + \sum_{i=1}^{n} a_i v_i + a_r v_{r_a} \tag{3.11}$$

$$B = b_0 + \sum_{i=1}^{n} b_i v_i + b_r v_{r_b} \tag{3.12}$$

The sum of $A$ and $B$ is computed as

$$A + B = (a_0 + b_0) + \sum_{i=1}^{n}(a_i + b_i)v_i + (a_r v_{r_a} + b_r v_{r_b}) \tag{3.13}$$

$$= s_0 + \sum_{i=1}^{n} s_i v_i + s_r v_{r_s} \tag{3.14}$$

where $s_r$ is identified by matching the variances of $s_r v_{r_s}$ and $a_r v_{r_a}$+$b_r v_{r_b}$.

To compute the maximum of A and B, denoted as $\max\{A, B\}$, the tightness probability ($T_P$) [VRK+04] is first computed. In [VRK+04], $T_p$ is defined as the probability that $A$ is larger than $B$. If $A$ and $B$ are both Gaussian, $T_P$ can be computed by

$$T_P = Prob\{A \geq B\} = \Phi(\frac{a_0 - b_0}{\theta}) \tag{3.15}$$

where $\Phi$ is the cumulative distribution function of the standard Gaussian distribution. $\theta = \sqrt{\sigma_A^2 + \sigma_B^2 - 2Cov(A, B)}$, where $\sigma_A^2$ and $\sigma_B^2$ are the variances of $A$ and $B$ respectively. $Cov(A, B)$ is the covariance between $A$ and $B$, and is computed according to [FF91] as

$$\begin{aligned} Cov(A, B) = &\sum_{i=1}^{n}\sum_{j=1}^{n} a_i b_j Cov(v_i, v_j) + \sum_{i=1}^{n} a_i b_r Cov(v_i, v_{r_b}) \\ &+ \sum_{i=1}^{n} b_i a_r Cov(v_i, v_{r_a}) + a_r b_r Cov(v_{r_a}, v_{r_b}) \end{aligned} \tag{3.16}$$

Because the random variables $v_{r_a}$, $v_{r_b}$ and $v_i$ in (3.11) and (3.12) are independent of each other, (3.16) can be simplified as

$$Cov(A, B) = \sum_{i=1}^{n} a_i b_i Cov(v_i, v_i) = \sum_{i=1}^{n} a_i b_i \sigma_{v_i}^2 \tag{3.17}$$

Compare (3.16) and (3.17), the computation is drastically simplified because the random variables are uncorrelated. This is the motivation that the correlated random variables in Section 3.2.2 are decomposed.

According to [Cla61], the mean ($\mu$) and variance ($\sigma^2$) of $\max\{A, B\}$ can be computed by

$$\mu = T_P a_0 + (1 - T_P) b_0 + \theta\phi(\frac{a_0 - b_0}{\theta}) \tag{3.18}$$

$$\begin{aligned} \sigma^2 = &T_P(\sigma_A^2 + a_0^2) + (1 - T_P)(\sigma_B^2 + b_0^2) \\ &+ (a_0 + b_0)\theta\phi(\frac{a_0 - b_0}{\theta}) - \mu^2 \end{aligned} \tag{3.19}$$

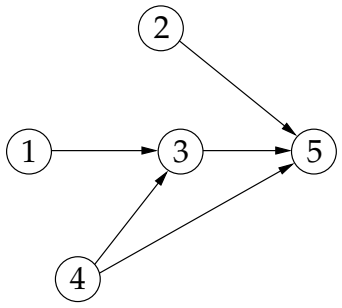


**Figure 3.6:** Correlation Example in Statistical Arrival Time Propagation

where $\phi$ is the probability density function of the standard Gaussian distribution. In order to apply the sum and maximum computations iteratively to propagate arrival times, $\max\{A, B\}$ is approximated in the same form of (3.10) as

$$max\{A, B\} \approx M_{A,B} = m_0 + \sum_{i=1}^{n} m_i v_i + m_r v_{r_m} \tag{3.20}$$

where $m_0$ is equal to $\mu$. $m_i$ is computed by $m_i = T_P a_i + (1 - T_P) b_i$. $m_r$ is computed by matching the variance of the linear form (3.20) and $\sigma^2$ in (3.19).

The sum and maximum computations discussed till now process correlation between arrival times implicitly. An example of such correlation is illustrated in Figure 3.6. The arrival times from nodes 2 and 3 to 5, denoted as $A_{25}$ and $A_{35}$, can be expressed as

$$A_{25} = A_2 + W_{25} \tag{3.21}$$
$$A_{35} = \max\{A_1 + W_{13}, A_4 + W_{43}\} + W_{35} \tag{3.22}$$

where $A_1$, $A_2$ and $A_4$ are arrival times at node 1, 2 and 4 respectively. In the method from [VRK+04], the computation of the maximum of $A_{25}$ and $A_{35}$ requires the covariance between them. This covariance can be computed as

$$\begin{aligned} Cov(A_{25}, A_{35}) =& Cov(A_2 + W_{25}, \max\{A_1 + W_{13}, A_4 + W_{43}\} + W_{35}) && (3.23)\\ =& Cov(A_2, \max\{A_1 + W_{13}, A_4 + W_{43}\}) + \\ & Cov(W_{25}, \max\{A_1 + W_{13}, A_4 + W_{43}\}) + \\ & Cov(A_2, W_{35}) + Cov(W_{25}, W_{35}) && (3.24) \end{aligned}$$

In [VRK+04] the maximum in the first two terms in (3.24) is approximated with a linear form. In order to compute the covariance correctly, the covariance computed with this linear form approximation should be equal to the covariance computed with the original maximum. This requirement is met in [VRK+04] by guaranteeing that the linear approximation has the same covariance to any other random variable.

That is, for a third random variable $C$ in linear form, written as

$$C = c_0 + \sum_{i=1}^{n} c_i v_i + c_r v_{r_c} \tag{3.25}$$

the maximum and its linear approximation $M_{A,B}$ in (3.20) of two random variables $A$ and $B$ defined in (3.11) and (3.12) should meet

$$Cov(\max\{A, B\}, C) = Cov(M_{A,B}, C) \tag{3.26}$$

According to [Cla61], the left side of (3.26) can be computed by

$$Cov(\max\{A, B\}, C) = T_p Cov(A, C) + (1 - T_p) Cov(B, C) \tag{3.27}$$

$$= T_p \sum_{i=1}^{n} a_i c_i \sigma_{v_i}^2 + (1 - T_p) \sum_{i=1}^{n} b_i c_i \sigma_{v_i}^2 \tag{3.28}$$

Similar to (3.16) and (3.17), the right side of (3.26) can be computed by

$$Cov(M_{A,B}, C) = \sum_{i=1}^{n} m_i c_i \sigma_{v_i}^2 = T_p \sum_{i=1}^{n} a_i c_i \sigma_{v_i}^2 + (1 - T_p) \sum_{i=1}^{n} b_i c_i \sigma_{v_i}^2 \tag{3.29}$$

From (3.27) to (3.29) the equation of (3.26) is proved, so that the arrival time computation of the method in [VRK+04] can handle correlation correctly.

The property (3.26) guarantees that the linear approximation in the maximum computation of [VRK+04] can preserve the correlation of the maximum to any random variable. Therefore, the correlation of the maximum to any independent variable $v_i$ is also preserved. This is the basis of the method proposed in [CS03]. The advantage of the method in [VRK+04] is that the correlation is handled implicitly and the computation of (3.15) and (3.17)-(3.19) need only to be fulfilled once in a maximum computation. Therefore this method is more efficient than [CS03].

In addition to the correlation between gate delays, reconvergent structures in the circuit cause further correlation. In Figure 3.6, the arrival time $A_4$ at node 4 has a purely random variable $v_{r_4}$. The two arrival times from node 3 and 4 to 5 are partially correlated because $v_{r_4}$ becomes a part of the arrival time of node 3 after the maximum computation at node 3. This correlation, however, is discarded in [VRK+04], because the sum of the purely random variables is merged into one variable in the maximum computation. At node 5, all the random parts of the incoming arrival times are assumed as independent. This assumption is not true because a purely random part may converge from different paths at following nodes, thus causing structural correlation [AZB03, DK03]. To solve this reconvergence problem, the canonical delay model (3.10) in [VRK+04] is extended in [ZHC05]. Instead of merging the initial purely random variables of gate delays, these variables are kept

separately in arrival times during propagation. Therefore, the correlation from these random variables can be incorporated.

The linear timing analysis methods require that gate delays are approximated by linear combinations of Gaussian random variables. As in modeling gate delays, statistical timing analysis methods using nonlinear or non-Gaussian gate delays or both are proposed to improve timing accuracy. In [ZSL+05] gate delays and arrival times are represented as quadratic functions of independent Gaussian random variables. The maximum computation is performed in a way similar to [CS03], where the covariances between the maximum and each term in the quadratic form are matched. As in [CS03], the first order correlation between the maximum and other variables are preserved. The disadvantage of this method is that numerical integration is needed for each coefficient identification, which makes the proposed method slow. In order to reduce the runtime of [ZSL+05], a parameter dimension reduction technique is proposed in [FLZ07]. Another method with a quadratic model is proposed in [ZCH+05]. This method still uses the tightness probability from [VRK+04], but only when the maximum of two quadratic variables is Gaussian. This Gaussian property is evaluated by computing the skewness of the maximum using the formula in [Cla61]. If the skewness is smaller than a threshold, the maximum is assumed to be Gaussian and is approximated by a linear combination of the two quadratic inputs. If the skewness is larger than the threshold, the maximum is not computed but the corresponding arrival times are directly propagated as a collection of quadratic forms. At each maximum computation, the skewness is evaluated so that the collections of quadratic forms can be compressed as soon as possible.

Representing gate delays as linear combinations of non-Gaussian variables, the method in [SS06,SS08] approximates the maximum of two variables also using tightness probability. The difference from [VRK+04] is that the tightness probability is computed from two non-Gaussian random variables, with the formulas proposed in [LLGP04]. This method has high efficiency, but the correlation between random variables is compromised during the maximum approximation. In the nonlinear non-Gaussian case, the method in [CZNV05] samples the nonlinear non-Gaussian parts of the variables so that the rest part of the arrival times are linear combinations of Gaussian variables, which can therefore be processed with the method in [VRK+04]. The accuracy of this sampling based method depends heavily on the number of samples. If the distributions of non-Gaussian variables are very complex and the number of them is large, this method faces runtime problem for moderate accuracy.

From the discussion above, correlation handling is always the source of complexity for statistical timing analysis. To avoid this complexity, correlation is simply discarded in [ABZV03b], where it is proved that the result without considering correlation is an upper bound of the result with correlation after the maximum computation. Without considering correlation, the statistical bounds in [ABZV03b] are

very loose. Therefore, selective enumeration is deployed in [ABZV03a, AZB03] to improve the bounding accuracy.

The algorithms discussed above are all block-based. Similar to static timing analysis, path-based methods are also explored to process statistical gate delays, e.g., in [ABZ+02, OB04]. To apply these methods, critical paths should be first identified. However, without a statistical timing method, the critical paths identified from static timing analysis can not be guaranteed to be critical [LLCP08]. Additionally, any path in the circuit contributes to the circuit delay distribution with certain probability. Consequently, it is not very clear how many paths should be selected for path-based methods to cover the paths which are statistically critical. Furthermore, it is very hard to implement incremental timing analysis with path-based methods, because any revision in the circuit can change the critical paths. With these disadvantages, path-based methods are currently limited in specific areas of application.

### 3.2.4 Statistical Timing Analysis of Sequential Circuits

Timing analysis of flip-flop based circuits is similar to the method for static timing analysis, Algorithm 3 in Section 2.5. The maximum and sum computations in arrival time propagation are replaced by statistical computations discussed in the previous section. The result $T_{min}$ is a random variable, whose properties define the performance distribution of the circuit. The clock feeding to all flip-flops must have a period larger than $T_{min}$ to guarantee the proper behavior of the circuit. Therefore, timing yield of a flip-flop based circuit at clock period $T$, defined as the probability that the circuit works correctly with clock period $T$, can be computed by (3.1).

For latch based circuit, the reduced timing graph introduced in Section 2.1 is first established by computing edge delays with statistical engines described in the previous section. Because of latch transparency described in Section 2.3, statistical timing analysis for latch based based circuits is more complex. From the timing specification (2.10) and (2.11), the arrival times in each local time zone depends on the arrival times in the previous local time zones. To compute the minimum clock period for a latch based circuit, a linear programming method is used in [SMO90b], as described in Section 2.6. However, this method does not work when variations are taken into account, because all arrival times become random variables and the optimization target in Algorithm 4 and 5 is also a random variable.

In order to identify the minimum clock period of a latch based circuit, direct timing specifications are derived in [CZ04, CZ06, ZTC+06]. The timing specification in common in these methods is that there should be no positive loop in the reduced timing graph. The *cumulative delay shift* ($M_c$) of a loop is defined as the sum of all

delay shifts when the loop is traversed in timing analysis, i.e.,

$$M_c = \sum_{e_{ij} \in \boldsymbol{E}} m_{ij} \tag{3.30}$$

where $\boldsymbol{E}$ is the set of edges on the loop. The nonpositive loop specification is defined as that there should be no loop in the reduced timing graph whose cumulative delay shift is positive.

A proof of this nonpositive loop specification is shown in [ZTC+06]. Assume the arrival time at node $i$ is denoted as $A_i^0$. After traversing across a loop $k$ times, the arrival time at $i$ is denoted as $A_i^k$. According to (2.10), the arrival time across a latch is no smaller than the one with the latch assumed transparent. Therefore, the relation between $A_i^k$ and $A_i^0$ can be established as

$$A_i^k \geq A_i^0 + k \sum_{e_{ij} \in \boldsymbol{E}} m_{ij} = A_i^0 + kM_c \tag{3.31}$$

If $M_c$ in (3.31) is positive, the arrival time returning to the initial node will eventually become infinity after sufficient loop traversals and violate the setup time constraint of the latch. Therefore, the nonpositive constraint can be expressed for each loop as

$$M_c = \sum_{e_{ij} \in \boldsymbol{E}} m_{ij} \leq 0 \tag{3.32}$$

The delay shift $m_{ij}$ in (3.35) is defined as $q_i + \Delta_{ij} - \varepsilon_{ij}$ in (2.7), where $q_i$ and $\Delta_{ij}$ are random variables. By replacing $m_{ij}$, the nonpositive constraint can be expressed as

$$\sum_{e_{ij} \in \boldsymbol{E}} (q_i + \Delta_{ij} - \varepsilon_{ij}) \leq 0 \tag{3.33}$$

In [ZTC+06] the clock phase shift $\varepsilon_{ij}$ is assumed as constant times the clock period, i.e., $\varepsilon_{ij} = \varsigma_{ij} T$. Therefore the constraint becomes

$$\sum_{e_{ij} \in \boldsymbol{E}} (q_i + \Delta_{ij}) \Big/ \sum_{e_{ij} \in \boldsymbol{E}} \varsigma_{ij} \leq T \tag{3.34}$$

The left side of (3.34) is a random variable and defines a statistical lower bound for the clock period, equivalently, minimum clock period for the circuit.

To compute the minimum clock period, all loops in the reduce timing graph should be enumerated. This is prohibitive when the number of latches is large in the circuit [CZ06]. To accelerate the enumeration, a loop breaking algorithm is proposed in [CZ04, CZ06]. In the first step, the reduced timing graph is searched in a depth-first order. All backward edges which form loops are removed from the graph and saved as a separate edge set $\boldsymbol{E}_r$. The remaining graph therefore becomes a directed acyclic

graph without any loop, denoted as $G_r$. For each edge $e$ from $\boldsymbol{E}_r$, $G_r$ is traversed with a statistical timing engine to compute the arrival time from the end node to the start node of $e$. Thereafter, the maximum of cumulative delay shifts of loops across $e$ is computed by adding the delay shift from the start node to the end node of $e$ to the arrival time after the traversal of $G_r$. After all edges in $\boldsymbol{E}_r$ are enumerated, the minimum clock period is computed as the maximum of the lower bounds, which are the left sides of the constraints in form of (3.34) from all loop traversals.

As stated in [CZ04, CZ06], the method above may miss loops when there are more than one backward edge in the loop. To increase loop coverage, the reduced timing graph is searched several times in random order to create the backward edge set. After each search, the minimum clock period is updated as described above. In this way, the probability of missing edges can be reduced, at the expense of runtime. The result from this heuristics is still an approximation and no guarantee can be made about the completeness of the loop enumeration.

To avoid the heuristics in [CZ04, CZ06], a method based on *cycle mean* and *iteration mean* is proposed in [ZTC+06]. For a loop with $n$ edges in the reduced timing graph, the cycle mean for the loop is defined as

$$m_c = M_c / n \tag{3.35}$$

where $M_c$ is the cumulative delay shift defined in (3.30). After initialization arrival times at all latches are updated using (2.10) repeatedly. For latch node $i$, the arrival time after the $k$th iteration is denoted as $A_i^k$. The *iteration mean* $O_i^k$ of the $k$th iteration is defined in [ZTC+06] as

$$O_i^k = \frac{A_i^k}{k+1} \tag{3.36}$$

In [ZTC+06] it is proved that when the iteration number $k$ becomes large enough, the iteration mean is equal to the cycle mean $m_c$ in (3.35) if the node belongs to a loop or is affected by an arrival time from a loop. Otherwise, the iteration mean is equal to 0. Therefore the nonpositive constraint can be described as that the maximum of the iteration means of all latches is no larger than 0 after sufficient iterations. The exit condition of the iterative arrival time update is that the first and second order moments of the iteration means of all latches do not change after an arrival time update. This is also an approximation because the correlation between arrival times is not considered in this exit condition. Additionally, the clock period should be given during the iteration because the computation of the first two moments of the arrival times requires that the clock period is known. This limitation restricts the method in [ZTC+06] to be used to compute the yield of the circuit against a given clock period. The complete cumulative probability function of the yield can only be achieved by sampling the clock period range and run the method in [ZTC+06] for each sample.

According to the reasoning of the nonpositive loop constraint, it is only a necessary condition for latch based circuits. To complete the timing constraint, it is specified in [ZTC+06] that the arrival times should meet the timing constraints of corresponding latches after sufficient iterations. This condition guarantees that the circuit works correctly after sufficient clock periods from reset, but still can not guarantee the correct function of the circuit directly after reset. To overcome this limitation, the timing constraints of latches are checked at the first time when arrival times are propagated through them in [LCS09a].

## 3.3 Timing Model Extraction for Static Timing Analysis

Similar to migrating from transistor level to gate level, hierarchical design style is adopted for further abstraction to overcome increasing design complexities. In a hierarchical flow, a design is composed of a series of modules at different levels. In designs using IP macros from third-party vendors, the complete netlists of these macros are not always available because of IP protection. Instead, timing models are provided as substitutes of the original netlists. Thereafter, timing analysis is run using these models to evaluate the performance of the complete system.

Timing models in hierarchical timing analysis are extracted from original circuits created by IP vendors or other design groups. These models contain only the timing information needed by the timing verification of the complete design. Therefore they are much smaller compared to original netlists. Another advantage of using timing models is the runtime reduction, because most of the timing constraints inside a module are compressed into a very simple form and only the interfacing constraints should be verified individually. For static timing analysis, different methods are already proposed for timing model extraction. In this section, these methods are reviewed because they may be partially reused in timing model extraction for statistical timing analysis.

### 3.3.1 Static Timing Model for Combinational Circuits

To derive the timing model for a combinational circuit, the requirements for such a circuit are specified first. The timing information of a combinational circuit is represented using a timing graph, as explained in Section 2.1. An arrival time assigned to a node in a timing graph saves the maximum delay from inputs of the circuit to this node. When a combinational circuit is used as a module, there is more than one

path from input $i$ to output $j$. The arrival time $A_j$ at output $j$ can be computed by

$$\begin{aligned} A_j &= \max_{i \in I}\{ \max_{p_{ij_k} \in P_{ij}} \{\tilde{A}_i + W_{ij_k}\}\} \\ &= \max_{i \in I}\{\tilde{A}_i + \max_{p_{ij_k} \in P_{ij}} \{W_{ij_k}\}\} \qquad (3.37) \\ &= \max_{i \in I}\{\tilde{A}_i + M_{ij}\} \qquad (3.38) \end{aligned}$$

where $I$ is the set of all inputs of the module. $\tilde{A}_i$ is the arrival time at input $i$ and depends on the application context. $W_{ij_k}$ is the delay of $p_{ij_k}$, which denotes the $k$th path between input $i$ and output $j$. The set of all paths between $i$ and $j$ is denoted by $P_{ij}$. $M_{ij}$ denotes the maximum path delay between $i$ and $j$.

According to (3.38) the arrival time at an output of a module is determined by the arrival times at all inputs of the module and the maximum delays from all inputs to the output. When characterizing the timing model of a module, especially an IP block, the application context is unknown. For this reason, no assumption about the arrival times at the inputs should be made. On the contrary, the maximum input-output delays $M_{ij}$ in (3.38) are exclusively determined by the module.

For a module with $m$ inputs and $n$ outputs, the *delay matrix* is defined as an $m \times n$ matrix, with entries $M_{ij}$. From the analysis above, a precharacterized timing model must have the same delay matrix as the one of the original circuit to retain correct timing information. For a module with a large number of inputs and outputs, the delay matrix may be too large to be used as an efficient timing model directly. In the following, existing timing model extraction methods for combinational circuits will be reviewed.

Timing models of combinational circuits are normally classified into black-box and gray-box types [MKB02]. A black-box timing model does not rely on the internal structure of the circuit and contains the delay information directly. A gray-box timing model, however, transforms the original timing graph into a smaller one as the timing model. Therefore, the efficiency of the gray-box timing model extraction depends heavily on the structure of the original circuit.

For a combinational circuit, the black-box timing model contains the delays between all inputs and outputs, i.e., the delay matrix directly. An example of black-box is proposed in [ALS+02]. This type of time model has good accuracy because the delay matrix can guarantee accurate arrival time propagation in (3.38). Additionally, the model extraction algorithm is relatively easy, e.g., the fast algorithm described in [Sap96] can be used to compute all maximum delays in the delay matrix. However, the size of a black-box timing model may be much larger than the original circuit, in case of large numbers of inputs and outputs. To overcome the limitation of black-box timing models, gray-box timing models are extracted by compressing

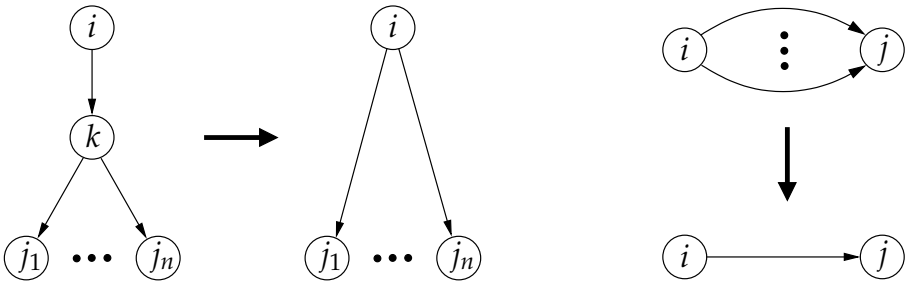


**Figure 3.7:** Basic Merge Operations [KM97, MKB02]

the original timing graphs. Some delay edges in the original timing graph are removed or merged and the resulting timing graph is used a gray-box timing model. The delay matrix of the gray-box timing model, however, is still accurate or a good approximation to the one of the original circuit.

In a gray-box timing model, two basic merge operations [KM97, MKB02] are applied to a number of edges and nodes in the timing graph. The serial merge operation is illustrated in Figure 3.7(a). If $n$ edges with sink nodes $j_1, \ldots j_n$ leave the same node $k$ and $k$ has only one fanin edge with source node $i$, $k$ can be removed and the edges can be merged so that there are only direct edges between $i$ and $j_1, \ldots j_n$. The delays of the new edges between $i$ and $j_1, \ldots j_n$ are the sums of the weights $W_{ik}$ and $W_{kj_1}, \ldots W_{kj_n}$, respectively. Similarly, this transformation can be applied in reverse direction, where $n$ edges meet at a node which has only one fanout edge. The parallel merge operation merges the edges with the same source and sink nodes, as illustrated in Figure 3.7(b). A new edge is created to replace the $n$ parallel edges, with delay equal to the maximum of all delays between $i$ and $j$, i.e., $\max\{W_{ij_1}, \ldots W_{ij_n}\}$.

After applying the serial merge operation, the delays between node $i$ and $j_1, \ldots j_n$ do not change. Because the delay between $i$ and $j$ in the parallel operation is determined by the maximum of all edge delays, the parallel operation also does not change the delay between $i$ and $j$. Consequently, applying the two basic operations in the original timing graph does not change the delay of any path going through a serial or parallel pattern, therefore guarantees the accuracy of the delay matrix. An example of applying the basic merge operations to the timing graph of c17 from ISCAS85 benchmarks is shown in Figure 3.8. A serial merge operation is applied to the subgraph defined by nodes 1, 6, 7 and 10; and to the subgraph defined by nodes 4, 5, 9 and 11.

The basic merge operations depend on specific structural patterns in the timing graph. In order to reduce the model size further, the butterfly-$\alpha$ transformation [KM97] is applied to increase the number of patterns for basic merge operations. This transformation is illustrated in Figure 3.9 and can only be applied when the

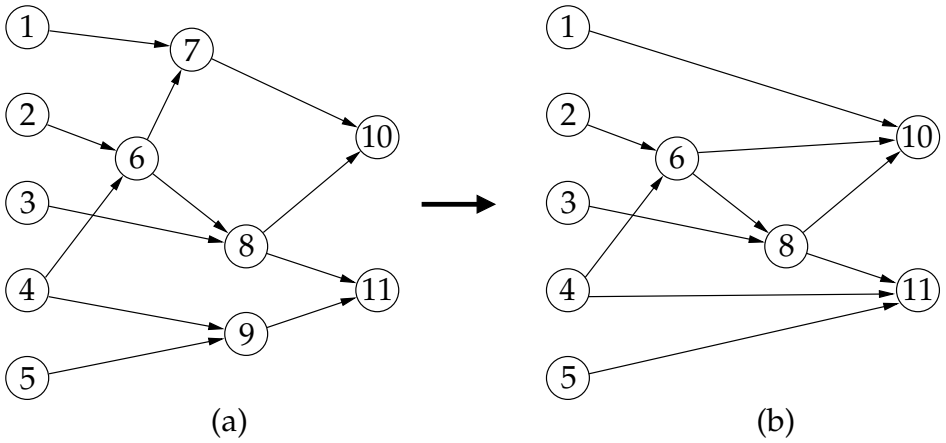


**Figure 3.8:** Example of Basic Merge Operations

condition $W_{13} \geq W_{14} + W_{23} - W_{24}$ holds in the original structural pattern. After this transformation, the weights of the newly created edges should meet the conditions that $W_{14} = W_{15} + W_{54}$, $W_{23} = W_{25} + W_{53}$ and $W_{24} = W_{25} + W_{54}$. By adding the first two conditions together and subtracting the last condition from the sum, the delay $W_{15} + W_{53}$ of the path formed by nodes $1 \rightarrow 5 \rightarrow 3$ is always equal to $W_{14} + W_{23} - W_{24}$. The condition in Figure 3.9 guarantees that the delay $W_{13}$ directly between nodes 1 and 3 dominates the path delay $W_{15} + W_{53}$. Therefore the maximum delay from node 1 to node 3 after this transformation is the same as the one in the original timing graph. A solution for the weights of the newly created edges is given in [KM97] as $W_{15} = W_{14} - W_{24}$, $W_{25} = 0$, $W_{53} = W_{23}$ and $W_{54} = W_{24}$. Additional solutions can be found by adding a value to the weights $W_{53}$ and $W_{54}$ and subtracting the same value from the weights $W_{15}$ and $W_{25}$. After this butterfly-$\alpha$ transformation, the edge between nodes 2 and 5 and the edge between nodes 5 and 4 can be merged with edges preceding or following this pattern. If the equal condition in Figure 3.9 holds, the direct edge between nodes 1 and 3 is not needed, because the path delay $W_{15} + W_{53}$ is equal to the delay $W_{13}$. In this case, the butterfly-$\alpha$ transformation can always yield very simple structure for the compression of the timing graph.

The butterfly-$\alpha$ transformation in [KM97] can be applied to patterns with two inputs

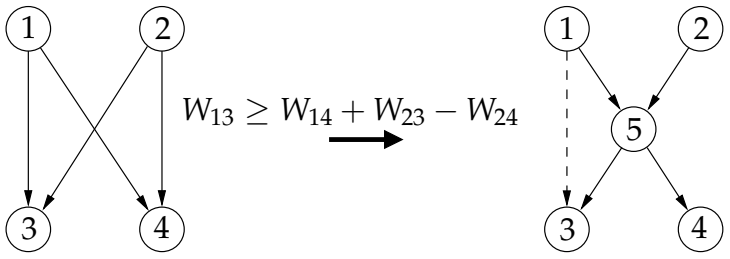


**Figure 3.9:** Butterfly-$\alpha$ Transformation [KM97]

and outputs. In [ZZH+06], a biclique-star replacement algorithm is introduced to transform graph patterns with more than two inputs and outputs. In this method, subgraphs are identified by an iterative search algorithm and transformed to star-like structure similar to the one in Figure 3.9 for better chance to apply the basic merge operations.

### 3.3.2 Static Timing Model for Sequential Circuits

Sequential circuits have two types: flip-flop based and latch based. For each of such circuit type, different methods to extract timing models will be reviewed in this section.

In flip-flop based circuits, a signal propagated from an input of a module stops at the inputs of its fanout flip-flops. The arrival times at the outputs of these flip-flops start to propagate only after the active clock edges, so they have no dependence on the arrival times at the inputs of these flip-flops. In other words, the internal circuit structure between flip-flops is separated from the application context by the flip-flops at the first level from inputs. The constraints from all paths between flip-flop pairs are compressed and represented by the minimum clock period, which can be computed similarly with the method discussed in Section 2.5. Similarly, the circuit part between the last flip-flops and outputs is also separated from the internal circuit structure. Consequently, only the constraints from inputs to flip-flops at the first level and the delays from the last flip-flops to outputs need to be extracted for timing models. When a module is used in a hierarchical design, the constraints inside the corresponding timing model are used to verify if the arrival times from previous modules can meet the timing constraints of the flip-flops at the first level of the module. The delays in the timing model are used to verify the timing constraints of the flip-flops inside the succeeding modules.

Normally there is more than one flip-flop which has at least one combinational path from input $i$. Because the arrival times at all these flip-flops must meet corresponding setup time constraints, the timing constraint for input $i$ can be written as

$$\max_j\{\tilde{A}_i + \Delta_{ij} + s_j\} \leq T \Longleftrightarrow \tag{3.39}$$

$$\tilde{A}_i + \max_j\{\Delta_{ij} + s_j\} \leq T \Longleftrightarrow \tag{3.40}$$

$$\tilde{A}_i + D_i \leq T \tag{3.41}$$

where $\tilde{A}_i$ is the arrival time at input $i$ and is only known after the module is instantiated. $\Delta_{ij}$ is the maximum path delay from $i$ to flip-flop $j$; $s_j$ is the setup time of flip-flop $j$. The maximum in (3.40) is performed with all flip-flops which have at least one combinational path from $i$. In order to verify the timing of the module,

only the result of the maximum computation for input $i$, denoted as $D_i$, needs to be contained in the timing model.

The first type of sequential timing model is Interface Logic Model (ILM) [ALS+02]. In this timing model, all circuit components which do not have a combinational path from an input or to an output are removed from the original circuit. The remaining circuit components are kept intact in their original status and used as the timing model. The combinational circuit components in the timing model are not compacted and contain original delay information corresponding to slope and load. All interconnects between gates are also kept inside the timing model with extracted parasitics unchanged. The advantage of ILM is its flexibility and simplicity. Because the timing model contains the original circuit information, it can be used in any, even including SPICE-based, design flow.

Another type of timing model for flip-flop based circuits is Extracted Timing Model (ETM) [ALS+02]. This timing model collapses the gate delays between flip-flops, inputs and outputs in the interface logic model. The constraint for input $i$ is represented by the inequality (3.41). For such a constraint, $D_i$ is computed using a static timing analysis engine, e.g., Algorithm 2. Because the slope-load information should be contained in the timing model, a lookup table is used to describe the compacted delay $D_i$.

The extracted timing model has a smaller size compared to the interface logic model because all circuit components from an input to all its fanout flip-flops are compressed. Similarly, circuit components from the last flip-flops to outputs are also compressed. This is different from the interface logic model, where all such circuit components are kept with their original timing information. The extracted timing model, however, can only be used for gate-level verification, because all information at transistor level is discarded during the delay collapse step. A similar method to generate extracted timing model is proposed in [MKB02], where graph based compression is used to identify the constraints and the minimum clock period. This method has the advantage of retaining delay edges with additional assertions in the circuit, but with more complex extraction computation.

For latch based circuits, the method in [MKB02] retains all latch nodes and collapses all gate delays. The resulting timing model still contains latches to allow arbitrary level of transparency. Because latches are not merged or discarded, the runtime of complex timing analysis algorithms, e.g., [SMO90b], is still large when using extracted timing models. Another method to extract timing models for latch based circuits is proposed in [VPMS97], where timing constraints at the inputs of a module are abstracted. Because latches can be transparent, this method substitutes the constraints iteratively across latches. To reduce complexity, latch transparency level is assumed to be a predefined value. This assumption is too strict because latch transparency can not be fixed during timing model extraction. In the interface logic

model [ALS+02], timing model extraction for latch based circuits is also covered. The level of latch transparency, however, is specified before timing model extraction so that this method assumes the same limitation as [VPMS97]. In the extracted timing model [ALS+02], the level of latch transparency is computed from prespecified arrival times at inputs of the module. Anytime these arrival times are out of range of the specification, the timing model should be regenerated for accuracy.

### 3.3.3 Timing Verification with Static Timing Models

With extracted timing models, the performance of a hierarchical design can be evaluated in shorter runtime and with reasonable accuracy. For combinational circuits, the acceleration of timing analysis of the complete hierarchical design comes from the smaller size of timing models compared to original circuits. For sequential circuits, timing verification of the complete design is performed only with the extracted timing constraints related to inputs and outputs. Compared to internal structures of original modules, the numbers of constraints in such timing models are much smaller. For instance, only one constraint is extracted for an input of a flip-flop based circuit. Therefore, the total number of constraints contained in the timing model is always one larger than the number of inputs, regardless of the number of circuit components inside the module. The additional one constraint specifies the minimum clock period for the paths between flip-flops.

When using the extracted timing models, the timing verification of the complete design may need adaption according to the types of timing models. For combinational circuits, the gray-box timing model is in the form of netlist or timing graph. The black-box timing model can also be represented with netlist or timing graph, with direct edges between inputs and outputs. Consequently, the verification algorithm for the complete hierarchical design needs no revision. However, the extracted timing models of sequential circuits are in the form of constraints and delays. After a module is instantiated in a hierarchical design, the arrival times, $\tilde{A}_i$ in (3.41) at the inputs of the module become known. Therefore, each constraint in (3.41) defines a lower bound of the clock period. Additionally, the minimum clock period $T_{min}$ specifying the constraints from paths between flip-flop pairs should also be verified.

Timing models for latch based circuits are more complex than for flip-flop based circuits. Because an arrival time at an input may propagate through several levels of latches, more than one constraint for the setup times of the transparent latches may be extracted for the input. All these constraints including the minimum clock period must be verified for the complete hierarchical design. Therefore the hierarchical timing verification flow becomes even complicated and needs further adaption to work with these timing models.

## 3.4 Hierarchical Statistical Timing Analysis

With process variations modeled by random variables directly, statistical timing analysis can overcome the pessimism in the worst-case design methodology. Because the traditional IC design flow heavily depends on the results of timing analysis, the application of statistical timing analysis demands renovation in the digital design methodology. In hierarchical timing analysis, new challenges arise when process variations are considered. Facing these variations, one of the statistical timing analysis algorithms discussed in Section 3.2 can be applied as a timing engine to extract timing models from different types of circuits. Because of correlation, timing verification of the complete hierarchical design with statistical timing models also needs further processing.

With process variations considered, all gate delays in a module are random variables. This causes most of the methods proposed for static timing model extraction not to work anymore. For example, the netlist transformation methods for combinational circuits proposed in [KM97, MKB02], such as butterfly-$\alpha$ and biclique-star replacement, depend on special patterns in edge delays. Because all delays are random variables and are represented by their moments of different orders, no simple relation similar to the one in Figure 3.9 can be established. However, some other transformations, e.g., basic merge operations in Figure 3.7, still work because only topological structures are required regardless of the relations between gate delays.

The second challenge in hierarchical statistical timing analysis is correlation handling between different modules. Because of spatial correlation, all gate delays are correlated with each other. Consider two gates in different modules, the correlation between their delays depends on their on-die distance. However, this on-die distance is unknown during timing model extraction because the positions of modules can only be fixed after they are instantiated into a design. Therefore, no knowledge about correlation between different modules can be established. Additionally, a module can be instantiated more than once, for example, in multi-core CPU systems. In this case, there is also correlation between delays in different instances of the same module. Consequently, correlation information between delays in different modules can not be incorporated into timing models. Instead, this correlation must be handled during timing verification of the complete design with extracted timing models. This is a completely new challenge in hierarchical statistical timing analysis, because the correlation between modules needs not to be considered in the traditional static hierarchical flow.

Hierarchical statistical timing analysis started to attract research attention only after the gradual maturing of statistical timing analysis. As a new research area, only one solution [GVTG08, GVTG09] is proposed to deal with the new challenges. For statistical timing model generation of flip-flop based circuits, this method extends

the classical extracted timing models in [ALS+02] to incorporate process variations. For timing verification of the complete design, a new variable substitution method is proposed to establish the correlation between modules. In the following, both algorithms will be explained in detail.

### 3.4.1 State of the Art in Statistical Timing Model Extraction

As the most widely used design style, flip-flop based circuits first gained attention in statistical timing model extraction. Similar to the extracted timing models in [ALS+02], a timing model in [GVTG08, GVTG09] contains the maximum delays from inputs of the module to their fanout flip-flops in the reduced timing graph. The delays from flip-flops to outputs are also included for timing verification of succeeding modules. These delays are represented as linear functions of process parameters in [GVTG08, GVTG09] extracted from the results of SPICE simulation.

A parameter can be split into three parts as in (3.2). If more than one process parameter is considered, the die-to-die parts of the random variables for different process parameters may have correlation between them. With a decomposition method, e.g., principal component analysis [Jol02], these random variables can be represented as linear combinations of independent ones. Assume there are $m$ independent random variables $\xi_i, i = 1, 2, \ldots m$ after decomposition modeling die-to-die variations of devices; and $n$ random variables $\eta_i, i = 1, 2, \ldots n$ for interconnects. The random variables $\xi_i$ and $\eta_i$ are shared by delays in all modules of a hierarchical design and need no additional processing during hierarchical statistical timing analysis. In addition to die-to-die variations, process parameters contain within-die variations. The correlation between these within-die variations depends on the distance of devices on the die. After the die area of the module is partitioned into a grid [CS03], the corresponding variables are also decomposed to independent random variables $\lambda_i, i = 1, 2, \ldots k$.

With all process parameters represented by independent random variables, the delay for path $p$ is assumed in linear form as

$$d_p = d_0 + \sum_{i=1}^{m} a_i \xi_i + \sum_{i=1}^{n} b_i \eta_i + \sum_{i=1}^{k} c_i \lambda_i + r\epsilon \tag{3.42}$$

where $a_i$, $b_i$ and $c_i$ are the coefficients of the independent random variables. $\epsilon$ represents the purely random part of the path delay, with $r$ as coefficient. $d_0$ is the nominal path delay without variation considered. The objective of statistical timing model extraction in [GVTG08, GVTG09] is to determine all coefficients of a critical path by SPICE simulation.

To reduce the complexity of path delay simulation, the coefficients for $\xi_i$, $\eta_i$, $\lambda_i$ and $\epsilon$ are processed separately. During the coefficient determination of each type of

random variables, the variations of other random variables are not considered. For example, if only the die-to-die variations of devices are considered, the path delay can be simplified as

$$d_p = d_0 + a_1\xi_1 + a_2\xi_2 + a_2\xi_3 + \cdots + a_m\xi_m \tag{3.43}$$

For a specific path, $2m+1$ path delays are obtained by setting the random variables in (3.43) to different corners. In the first run, all random variables are set to their nominal values. In the remaining simulations, each random variable from $\xi_i$ is selected and set to its value at $\sigma$ with other variables in $\xi_i$ at their nominal values. Similarly this variable is sampled at its $-\sigma$ for another simulation. After the $m$ variables are processed, $2m$ samples in total are created. Thereafter, the transformation matrix for correlation decomposition is used to transform these sample values back to the values of the original process parameters. These parameters are applied to run SPICE simulations for path delay samples. With the total $2m+1$ simulation results, the regression method proposed in [MR06] is used to determine the $m$ coefficients $a_i$ in (3.43), by minimizing the error of mean square. Similar to the steps for $a_i$, the coefficients $b_i$ and $c_i$ in (3.42) are also determined considering one type of variation once a time.

The determination of the purely random part in (3.42) is more complex because of the large number of random variables used to model mismatch inside digital gates. In [GVTG08, GVTG09], an independent random variable is assigned to each parameter per transistor to model mismatch effect from manufacturing process. In order to evaluate the sensitivity of a path delay to these mismatch random variables, process variations from die-to-die and within-die variations are not considered, i.e., $\xi_i$, $\eta_i$ and $\lambda_i$ in (3.42) are assumed to nominal values. This is similar to the way to determine $a_i$, $b_i$ and $c_i$ in (3.42). Consider a path with $l$ transistors on it and $q$ independent random variables are used to model the mismatch of parameters for each transistor. The path delay is expressed in a linear form as

$$d_p = d_0 + \sum_{i=1}^{q} r_{1i}\epsilon_{1i} + \sum_{i=1}^{q} r_{2i}\epsilon_{2i} + \cdots + \sum_{i=1}^{q} r_{li}\epsilon_{li} \tag{3.44}$$

$$= d_0 + r_p\epsilon_p \tag{3.45}$$

where $\epsilon_{ji}$ is the random variable used to model the purely independent process variation of the $i$th parameter of $j$th transistor on the path. $r_{ji}$ is the coefficient of $\epsilon_{ji}$. The effect of all these independent random variables $\epsilon_{ji}$ is merged into one random variable $\epsilon_p$, corresponding to $\epsilon$ in (3.42).

To determine the coefficients $r_{ji}$ in (3.44), each random variable $\epsilon_{ji}$ should be sampled. Because all random variables are independent, the number of them can not be compressed using parameter decomposition. With a large number of independent

random variables and different slope-load combinations, the identification of sensitivities of the path delay to mismatch random variables using SPICE simulation is very expensive. In [GVTG08, GVTG09], SPICE simulations of a path corresponding to different slope-load combinations are accelerated by ignoring random variables which are not significant to the path delay. As the initialization step, SPICE simulations corresponding to one slope-load combination with all random variables are fulfilled. From the results of these simulations, the coefficients $r_{ji}$ are determined for this slope-load combination. In the following steps to determine $r_{ji}$ corresponding to other slope-load combinations, the number of random variables is reduced. First, the random variables with coefficients equal to 0 in the result of the first step are discarded. Thereafter, the significance of other variables is investigated by setting them to worst-case points and running SPICE simulations. These worst-case corners are determined by the standard deviations of the random variables and the sign of their coefficients from the first step. If some random variables do not affect the variance of the path delay significantly, they are also discarded in following simulations to determine coefficients $r_{ji}$ corresponding to further slope-load combinations.

After the coefficients for die-to-die, within-die and the purely random variables are determined, all delays in the extracted timing model are in the linear form of (3.42). When such a module is instantiated in a hierarchical design, the successive delays are summed up and verified against corresponding timing constraints. The correlation between delays is contained in these linear forms by sharing the same random variables.

As the first method proposed for statistical timing model extraction, this method can effectively capture the delays related to inputs and outputs of a flip-flop based module. However, timing model extraction with SPICE simulation is very slow, especially when different parameter corners are simulated at transistor level. Additionally, the constraints from paths between flip-flop pairs are not considered. Because the number of these paths are very large in industrial designs, the SPICE simulation based method can not extract corresponding constraints in reasonable runtime.

Specifically in the details of this method, the paths are preselected for statistical evaluation. For example, to extract the setup constraint at a flip-flop which has a combinational path from an input, the critical path from this input to the flip-flop is characterized. However, the selected path may not be the critical one in the context of statistical analysis, because any path has a certain probability to become critical after manufacturing [XZVV06, LLCP08, MQSB09]. Another limitation of this method is in the step to determine coefficients of independent variables. These variables are only sampled at the corners of standard deviation. Thereafter, the coefficients are determined by a regression method using the results of SPICE simulations. However, the accuracy of such sampling and regression is not proved in [GVTG08, GVTG09]. For modeling mismatch effect, each parameter per transistor is assigned a purely

random variable. Consequently a large number of random variables are involved into simulation. This is because the gate modeling level is omitted in the discussed method, so that identifying a path delay needs to sample all purely random variables on the path, rather than characterizing the coefficients of the random variables for each type of gate.

### 3.4.2 State of the Art in Hierarchical Statistical Timing Analysis

In static timing analysis, extracted timing models can be easily integrated into a hierarchical flow. The timing verification algorithm for the complete hierarchical design needs only small revision to incorporate different timing models, as discussed in Section 3.3.3. Hierarchical statistical timing analysis, however, needs an additional step to handle the correlation between modules after they are instantiated into a design. In Figure 3.10 two modules A and B are illustrated on a die as an example.

Because of spatial correlation, the characterized delays in timing models are correlated. Consider the linear delay form in (3.42). The random variables $\xi_i$ and $\eta_i$ model die-to-die variations so that they are shared by all models. The correlation resulting from these variables can be computed using their coefficients $a_i$ and $b_i$ in (3.42). But the random variables $\lambda_i$ are generated from the correlation matrix of random variables which model the within-die variations inside the die area of the module. In different modules, the sets of variables $\lambda_i$ are different. Therefore no correlation from within-die variations between two modules can be established by variable sharing. This is reasonable because within-die correlation between modules can only be determined with layout information of module instances. During statistical timing model extraction, such correlation information can not be handled because even the number of grid cells for the complete hierarchical design is unknown.

To solve this problem, a variable replacement method is proposed in [GVTG08, GVTG09]. The basic idea can be summarized in the following. The die area of a module is partitioned in the same way during timing model extraction and during timing analysis of the complete design. Therefore, a random variable representing the variation of a process parameter is the same when assigned to the same grid cell during timing model extraction and hierarchical analysis. Assume that the $t$ correlated random variables modeling process variations in a module are denoted as $p_1, p_2, \ldots p_t$. For module A in Figure 3.10 $t$ is set to nine, equal to the number of grid cells after partition. During timing model extraction, the $t$ correlated random variables are decomposed into $k$ independent ones $\lambda_1, \lambda_2, \ldots \lambda_k$. In order to reduce the number of random variables, the smallest eigenvectors and eigenvalues are not included into the decomposition, so that $t > k$. Consequently, the decomposition

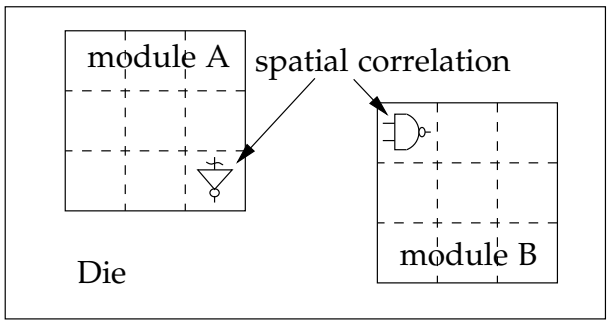


**Figure 3.10:** Correlation Between Modules

using PCA can be written as a set of linear equations.

$$\begin{aligned} p_1 &= \mu_1 + \alpha_{11}\lambda_1 + \alpha_{12}\lambda_2 + \cdots + \alpha_{1k}\lambda_k \\ p_2 &= \mu_2 + \alpha_{21}\lambda_1 + \alpha_{22}\lambda_2 + \cdots + \alpha_{2k}\lambda_k \\ &\vdots \\ p_k &= \mu_k + \alpha_{k1}\lambda_1 + \alpha_{k2}\lambda_2 + \cdots + \alpha_{kk}\lambda_k \\ &\vdots \\ p_t &= \mu_t + \alpha_{t1}\lambda_1 + \alpha_{t2}\lambda_2 + \cdots + \alpha_{tk}\lambda_k \end{aligned} \tag{3.46}$$

$$\Updownarrow$$

$$\boldsymbol{p} = \boldsymbol{A}\boldsymbol{\lambda} + \boldsymbol{\mu} \tag{3.47}$$

where $\boldsymbol{A}$ is the $t \times k$ coefficient matrix formed by $\alpha_{11}$ to $\alpha_{tk}$. $\boldsymbol{p} = [p_1, p_2, \ldots p_t]^T$ and $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots \lambda_k]^T$. Compared to (3.47) the decomposition in (3.8) does not include the mean vector $\boldsymbol{\mu}$ because the nominal values of process variations are already merged into the nominal gate delay in (3.5).

In the method proposed in [GVTG08, GVTG09], $k$ equations from (3.46) are stored in the timing model, where $k$ is the number of independent random variables after decomposition, used in (3.42). It is not explained in [GVTG08, GVTG09] how these $k$ equations are selected. Problems in such a selection will be discussed later in this section. The variables modeling process variations corresponding to the selected $k$ equations are denoted with $\boldsymbol{p}_k$, which is a subset of $\boldsymbol{p}$. The coefficients of the selected $k$ equations from a $k \times k$ matrix $\boldsymbol{A}_k$, which is a submatrix of $\boldsymbol{A}$. When the module is instantiated into a hierarchical design, the $k$ random variables $\lambda_1, \lambda_2, \cdots \lambda_k$ are mapped back to linear combinations of the original variables representing process variations by

$$\boldsymbol{p}_k = \boldsymbol{A}_k\boldsymbol{\lambda} + \boldsymbol{\mu}_k \tag{3.48}$$

where $\boldsymbol{\mu}_k$ are the nominal values corresponding to the selected transformation equations.

With (3.48) a delay in the timing model can be expressed as a linear combination of variables in $\boldsymbol{p}_k$, which are the variables assigned to the grid cells covering the die area of the module in the complete hierarchical design. After all random variables in the timing model are replaced by corresponding variables representing process variations, these variables are decomposed using the correlation matrix created from the grid for the die area of the complete design. Thereafter, all delays inside timing models are transformed into linear combinations of the new set of independent random variables. The correlation between delays inside different modules is established again by sharing the same set of random variables. Finally, a standard statistical timing analysis algorithm, e.g., [VRK+04], or Monte Carlo based method can be used to compute the timing performance of the complete design.

The discussed correlation handling method can effectively incorporate the correlation between modules after instantiation, thus guaranteeing the accuracy of timing analysis of the complete design [GVTG08, GVTG09]. However, this method still has limitations. The correlation is handled by selecting $k$ transformation equations, but it is not specified how these $k$ equations are selected. If not selected properly, the submatrix $\boldsymbol{A}_k$ may be rank deficient, because $t$ is larger than $k$ so that the $t$ row vectors in $\boldsymbol{A}$ are linearly dependent. When vector $\boldsymbol{\mu}_k$ is also considered, the new linear system (3.48) may have more than one solution or have no solution. In both cases, the backward transformation from the decomposed random variables to the original variables is not feasible. Additionally, there are also circuit components which are directly implemented in the top design. The correlation between these circuit components and the ones inside modules are not studied in the method in [GVTG08, GVTG09].

## 3.5 Summary

Relative process variations become large in deep submicron technology nodes. According to the characteristics of correlation, a variation can be modeled by a sum of die-to-die, within-die and independent variables. After decomposing the variables representing within-die variation, gate delays are modeled as functions of independent random variables. Different methods are proposed to evaluate the timing performance of a circuit from these statistical gate delays, with trade-off between runtime and accuracy. The first step of applying statistical timing analysis in a hierarchical design flow is to extract timing models. Methods for timing model extraction without considering process variations are reviewed. Based on these methods, process variations are considered in a state-of-the-art method of timing model extraction for flip-flop based circuits. To handle correlation between modules, an additional step is needed in the hierarchical statistical timing analysis flow. The

problems of the reviewed method of hierarchical statistical timing analysis will be addressed in the following chapter.

# Chapter 4

# Statistical Timing Model Extraction

In hierarchical statistical timing analysis, the extracted timing models affect the performance and accuracy of verification significantly. In this chapter, new methods to extract timing models for combinational circuits, flip-flop based and latch based circuits are explained. The preliminary results from these methods have been published in [LKS+08, LCS+09b, LCS09a].

For combinational circuits, the probabilities of delay edges on the critical paths between any input and output pairs are evaluated. Small probability indicates that an edge does not affect the delay matrix of the combinational circuit significantly. Therefore, edges with such probability smaller than a predefined threshold are removed to compress the timing model. For flip-flop based circuits, the extracted timing model [ALS+02] is extended with a standard statistical timing analysis engine [VRK+04]. The constraint from paths between flip-flop pairs missing in [GVTG08, GVTG09] is also extracted. For latch based circuits, the timing specification discussed in Section 2.3 is first reviewed. Thereafter, this specification is restructured to extract timing constraints for inputs of the module and paths between latches. The extracted timing models have small size and are adaptive to yield evaluation against arbitrary clock period.

## 4.1 Timing Model Extraction for Combinational Circuits

A combinational circuit contains only logic gates, such as AND, OR, NAND etc.. Unlike registers, the arrival times at logic gates start to propagate to the next gates instantly. Although purely combinational modules are rare in real designs, combinational paths between inputs and outputs exist widely in sequential circuits. For hierarchical statistical timing analysis, these paths should be processed as combinational circuits.

The extracted timing model for a combinational circuit should have the same maximum delays between inputs and outputs, as specified in (3.37) and (3.38). The basic idea of extracting a timing model for a combinational circuit is to transform the timing graph of the original circuit into a smaller one. Because of the smaller numbers of edges and nodes in the timing model, the runtime of timing verification for the complete design can be reduced. This concept is already used in [KM97, MKB02, ZZH+06], where the basic merge, butterfly-$\alpha$ and biclique-star replacement operations are used to reduce the number of edges in timing models for static timing analysis.

When process variations are considered, all edge delays in the timing graph become random variables. This change makes the delay pattern dependent transformation, e.g., butterfly-$\alpha$ in [KM97], infeasible. However, the two basic merge operations illustrated in Figure 3.7 can still be applied. In these two operations, only structural patterns are identified and transformed. The sum and maximum computations in these transformations can be performed statistically when variations are considered, so that the delay matrix of the combinational circuit is still maintained. In the result of Figure 3.8(b), the timing graph cannot be compressed further because no structural pattern of the basic merge operations exists. In this section, a new method to remove noncritical edges is proposed. This method can not only reduce the number of edges in the timing graph effectively, but also increase the number of structural patterns to apply the basic merge operations.

### 4.1.1 Concept of Noncritical Edge Removal for Static Timing Analysis

In this section, the concept of noncritical edge removal will be explained in the context of static timing analysis. In a combinational circuit there is normally more than one path from an input to an output in a module. In timing analysis, only the paths with dominant delays, called *critical paths*, determine the delay matrix of the module. From this observation, the edges which are never on critical paths can be removed without affecting the timing information of the module, therefore reducing model size while accuracy is still preserved. Note that the definition of critical path in this section is different from the classical one, where the critical path dominates the paths starting from all inputs to all outputs of a circuit. A critical path in this section, however, dominates all the paths starting from a specified input to a specified output.

Figure 4.1 illustrates the concept of the noncritical edge removal, where all edge delays are assumed as unit delay for simplicity. If all pairs of inputs and outputs of the circuit are investigated, it can be found that the edge between nodes 6 and 10 locates only on the paths from inputs 2 and 4 to output 10. However, the critical

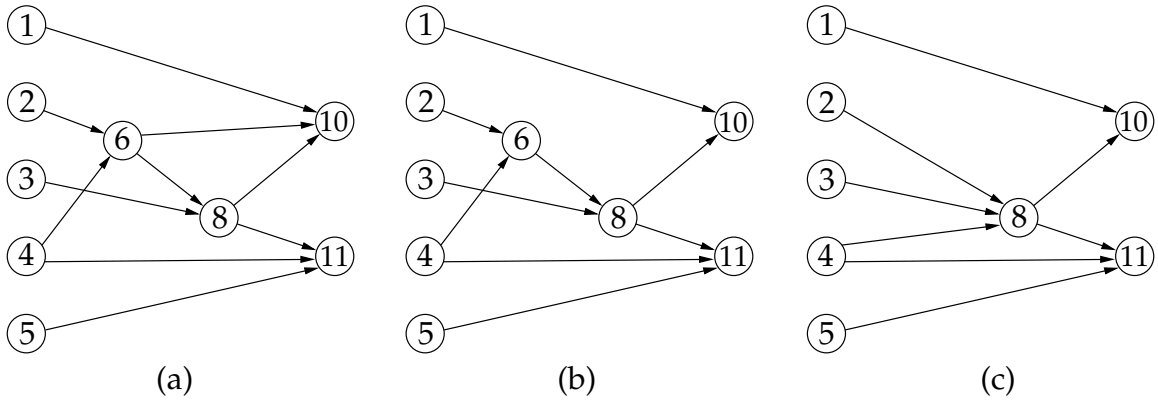


**Figure 4.1:** Example of Noncritical Edge Removal

paths of both input-output pairs pass through nodes 6 and 8. In other words, the edge between nodes 6 and 10 is dominated by the path delay between 6, 8 and 10. Therefore, the removal of the edge directly between nodes 6 and 10 does not affect the maximum delays between inputs and outputs of the circuit. After this removal, a basic serial merge operation can be applied to the subgraph defined by nodes 2, 4, 6 and 8 in Figure 4.1(b) to compress the timing graph further. The resulting timing graph is shown in Figure 4.1(c). Similarly the noncritical edge between nodes 4 and 11 can also be removed to compress the timing model.

To reduce the size of the timing graph, only edges never on the critical path of any input-output pair can be removed. To identify the noncritical edges, static timing analysis from each input using Algorithm 2 is run. This algorithm computes arrival time from an input to all nodes in its fanout cone in the timing graph. After each run of Algorithm 2, critical paths from the input to all outputs are identified by backward tracing from outputs. The complete algorithm is shown in Algorithm 6.

Lines 10-18 in Algorithm 6 identify the edges on the critical path between an input and an output. At a node in the timing graph, if its arrival time is determined by the arrival time at one of its fanin nodes and the corresponding edge delay, this edge is set as critical and the path is traced backwards further, as shown in lines 11-17. Note that the backward tracing selects only one fanin node in the iteration, because one path is enough to determine the maximum delay between the input and the output. Lines 4-6 clear all arrival times, so that only the nodes which are in the fanout cone of the current input have arrival times larger than 0 after running Algorithm 2. This guarantees that the backward critical path traversal is performed only in the fanout cone of the current input. After applying Algorithm 6, all edges which are never marked as critical can be removed from the timing graph without affecting the maximum delay between any input and output.

**Algorithm 6**: Critical Edge Identification in Static Timing Analysis

```
    // variables
1   n_*: nodes;
2   A_*: arrival times;
3   foreach primary input n_p do
4       foreach node n_c in the timing graph do
5           A_c ← −∞;
6       end
7       run Algorithm 2 from n_p;
8       foreach primary output n_q do
9           n_i ← n_q;
10          while n_i ≠ n_p do
11              foreach fanin node n_j of n_i do
12                  if A_j + W_ji = A_i then
13                      mark edge e_ji as critical;
14                      n_i ← n_j;
15                      break;
16                  end
17              end
18          end
19      end
20  end
```

### 4.1.2 Noncritical Edge Removal in Statistical Timing Analysis

When process variations are considered, the basic concept of the noncritical edge removal needs revision to handle probabilistic gate delays. In statistical timing analysis, all delays are random variables. A path delay can only dominate the delay of another path with certain probability. As an example, consider there are two paths with statistical delays. Because both path delays are described with nontruncated distributions in most statistical timing analysis algorithms, the probability that one path delay is larger than the other is always positive. This is different from that in static timing analysis, where a path dominates another always with either 100% or 0 probability. Because every path can be critical in statistical timing analysis, any edge can also be critical with certain probability.

If process variations are considered, the arrival time at a node in Algorithm 6 is computed using a statistical timing engine, e.g., [VRK+04], where the maximum of two arrival times is computed by linear combination of them with variance matching, shown with (3.20) in Section 3.2.3. This computation makes the backward critical

path traversal in Algorithm 6 infeasible, because the condition in line 12 can rarely be true even for all fanin nodes. Therefore, a new method is needed to identify the edges which are noncritical to the module.

Because the critical path problem is crucial for circuit optimization, several methods, e.g., [XZVV06, LLCP08, MQSB09], are already proposed for statistical timing analysis. In this section, the method proposed in [XZV08] is extended to identify critical edges in the timing graph with the concept of criticality.

The *criticality* $c_{ij}$ for an edge $e_{ij}$ between node $i$ and $j$ is defined in [XZV08] as the probability that the edge is on the critical path of the circuit after manufacturing. This criticality is first computed in [VRK+04] by forward and backward propagation. Because this method does not consider the correlation during the recursive computation, the resulting criticalities are inaccurate [XZV08]. To improve the accuracy of criticalities, cutset based methods are proposed in [XZVV06, MQSB09]. However, both methods are complex and time consuming. From circuit view, the method in [XZV08] directly computes edge criticalities after forward and backward traversals of the circuit with a standard statistical timing engine. Therefore this method is very fast and can yield accurate results.

The magnitude of the criticality defined in [XZV08] designates the probability that the edge affects the delay of the critical path of the complete circuit. In the noncritical edge removal method in Section 4.1.1, an edge can be removed only if it does not affect any maximum delay between inputs and outputs. In order to represent the probability of an edge on the critical path between a pair of input and output, the definition of criticality in [XZV08] is extended as $c_{ij}^{pq}$, which defines the probability that the edge is on the critical path between input $p$ and output $q$. If the maximum of $c_{ij}^{pq}$ corresponding to all pairs of inputs and outputs is smaller than a predefined probability threshold, the edge does not affect the delay matrix of the circuit with significant probability and can be removed from the timing graph.

The *maximum criticality* of an edge corresponding to all pairs of inputs and outputs is defined as

$$c_{ij}^{m} = \max_{p,q}\{c_{ij}^{pq}\} \tag{4.1}$$

where the maximum is computed with all pairs of inputs and outputs. Because the computation of $c_{ij}^{m}$ from all $c_{ij}^{pq}$ is straightforward, only the computation of the criticality $c_{ij}^{pq}$ will be explained in the following.

In [XZV08], the criticality of an edge is computed by splitting the paths into two sets. This method is applied to compute the criticality of an edge corresponding to an input-output pair in this section. The concept of path split is the same, but the paths in consideration are only between an input and an output.

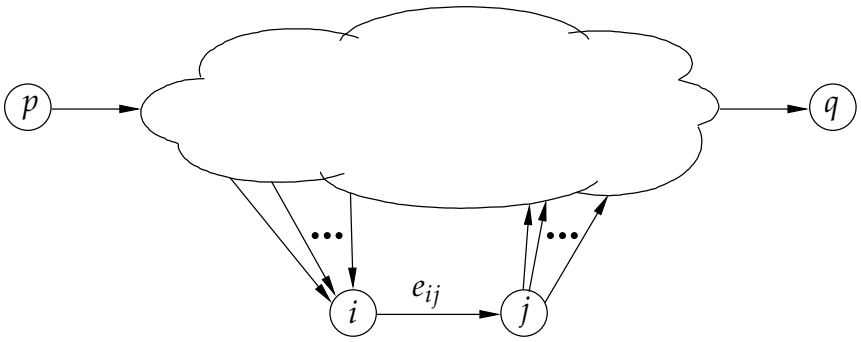


**Figure 4.2:** Path Partition according to an Edge

In a timing graph, there are many paths passing through edge $e_{ij}$. All paths in the circuit are categorized into two sets. All paths between an input $p$ and output $q$ and passing through $e_{ij}$ are in the set $\mathcal{P}_{ij}$. All paths not passing through $e_{ij}$ are denoted as a set $\overline{\mathcal{P}_{ij}}$. In the denotations of the two path sets, input and output indexes $p$ and $q$ are not included for simplicity. The concept of this partition is illustrated in Figure 4.2. The maximum of delays of all paths passing through $e_{ij}$ is denoted as $D_{ij}$; the maximum of delays of all paths not passing through $e_{ij}$ is denoted as $D_{\overline{ij}}$. If edge $e_{ij}$ is on the critical path, the longest path in $\mathcal{P}_{ij}$ dominates the longest path in $\overline{\mathcal{P}}_{ij}$, which means $D_{ij} \geq D_{\overline{ij}}$. This statement is also valid vice versa. According to its definition, the criticality $c_{ij}^{pq}$ can be computed as

$$c_{ij}^{pq} = Prob\{D_{ij} \geq D_{\overline{ij}}\} \tag{4.2}$$

As proposed in [XZV08], the criticality computation can be performed further as

$$\begin{aligned} c_{ij}^{pq} &= Prob\{D_{ij} \geq D_{\overline{ij}}, D_{ij} \geq D_{ij}\} && (4.3)\\ &= Prob\{D_{ij} \geq \max\{D_{ij}, D_{\overline{ij}}\}\} && (4.4) \end{aligned}$$

where the probability is computed as all the conditions inside the bracket are true at the same time. $\max\{D_{ij}, D_{\overline{ij}}\}$ is the maximum delay of all paths between the input $p$ and the output $q$ and is equal to the maximum input-output delay $M_{pq}$ defined in (3.38). This maximum delay can be evaluated very fast by applying Algorithm 2 to each input, but with all sum and maximum computations replaced by the statistical ones proposed in [VRK+04].

Similar to the explanation in [XZV08], the maximum delay of paths passing through edge $e_{ij}$ can be computed by

$$D_{ij} = A_i + W_{ij} + R_j \tag{4.5}$$

where $A_i$ is the maximum delay from input $p$ to node $i$ and equal to the corresponding arrival time exclusively from $p$. $R_j$ is the maximum delay from output $q$ to node

$j$ and equal to the corresponding negative required time exclusively from $q$, with the required time at $q$ set to 0. $W_{ij}$ is the statistical delay of edge $e_{ij}$.

A short proof of (4.5) is given in the following. The delay of a path $p_s$ from node $p$ to $i$ is denoted as $W_s$. The delay of a path $p_t$ from node $j$ to $q$ is denoted as $W_t$. A path from node $p$ to node $q$ combines three segments and its delay can be computed by $W_s + W_{ij} + W_t$. Assume there are $k_i$ paths from $p$ to $i$, and $k_j$ paths from $j$ to $q$. The total number of paths between $p$ and $q$ and passing through $e_{ij}$ is $k_i \times k_j$. The maximum delay $D_{ij}$ of these paths can be computed as

$$\begin{aligned} D_{ij} &= \max_{\substack{s=1,2,\ldots k_i \\ t=1,2,\ldots k_j}} \{W_s + W_{ij} + W_t\} \\ &= \max_{s=1,2,\ldots k_i} \{W_s + W_{ij} + \max_{t=1,2,\ldots k_j} \{W_t\}\} \\ &= \max_{s=1,2,\ldots k_i} \{W_s + W_{ij} + R_j\} \\ &= \max_{s=1,2,\ldots k_i} \{W_s\} + W_{ij} + R_j \\ &= A_i + W_{ij} + R_j \end{aligned} \tag{4.6}$$

Combined with statistical sum and maximum computations in [VRK+04], Algorithm 2 [Sap96] is used to propagate arrival times to all nodes from each input. If an output node is reached, the arrival time at it is the maximum delay between the current input and the output, i.e., $\max\{D_{ij}, D_{\overline{ij}}\}$ in (4.4). Similarly, this arrival time propagation can be performed backwards, so that the maximum delays to all nodes from the output can be computed. For each input-output pair, the criticality of each edge is computed using (4.2). The maximum criticality $c_{ij}^m$ of an edge is updated by the larger one of its current value and the newly computed criticality.

### 4.1.3 Timing Model Extraction with Noncritical Edge Removal

The computed maximum criticality designates the maximum of the probabilities that an edge affects the delays between all pairs of input and output of the circuit. If this probability is smaller than a predefined small threshold $\delta_c$ approximating 0, the removal of this edge does not affect the accuracy of the timing model significantly. The complete algorithm to compute criticality and noncritical edge removal is listed in Algorithm 7.

In the static version of the noncritical edge identification listed in Algorithm 6, the required time is not computed for every node. Only one critical path is traced backwards according to the arrival times computed in the forward propagation. This is different from the statistical case in Algorithm 7, because required times at

**Algorithm 7**: Statistical Model Extraction for Combinational Circuits

```
   // variables
1  n_*: nodes;
2  A_*: arrival times;
   // clear all maximum criticalities
3  foreach node n_c in the timing graph do
4  |  c_ij^m ← 0;
5  end
   // edge criticality computation
6  foreach primary input n_p do
7  |  foreach node n_c in the timing graph do
8  |  |  A_c ← −∞;
9  |  end
10 |  run Algorithm 2 from n_p;
11 |  foreach primary output n_q do
12 |  |  foreach node n_c in the timing graph do
13 |  |  |  R_c ← −∞;
14 |  |  end
15 |  |  run Algorithm 2 backwards from n_q;
16 |  |  foreach edge e_ij in the timing graph do
17 |  |  |  compute D_ij with (4.5);
18 |  |  |  compute criticality c_ij^pq with (4.2);
19 |  |  |  if c_ij^pq > c_ij^m then
20 |  |  |  |  c_ij^m ← c_ij^pq;
21 |  |  |  end
22 |  |  end
23 |  end
24 end
   // noncritical edge removal
25 foreach edge e_ij in the timing graph do
26 |  if c_ij^m < δ_c then
27 |  |  remove e_ij from timing graph;
28 |  end
29 end
   // compress timing graph with basic merge operations
30 repeat
31 |  merge serial and parallel patterns similar to Figure 3.7;
32 until no change in the timing graph ;
```

all nodes are needed in lines 16-22. In the backward traversal, only the nodes in the fanout cone of the input $n_p$ should be visited. This can be used to accelerate the computation instead of visiting all edges in the timing graph in Algorithm 7 (line 16). After noncritical edges are removed from the timing graph, the basic merge operations are applied to compress the timing model further. All the sum and maximum computations involved in this algorithm are from the statistical engine [VRK+04]. The final timing graph is used as the timing model for the verification of the complete design, with much fewer nodes and edges inside.

## 4.2 Timing Model Extraction for Flip-flop Based Circuits

For timing extraction of flip-flop based circuits, the most recently proposed method in [GVTG08, GVTG09] uses SPICE simulation based path delay extraction. This method directly establishes the path delay sensitivities to process variations ignoring the intermediate gate level modeling in the design flow. Because of the long runtime of SPICE simulation, this method can not capture the path delays between all flip-flop pairs. These delays are crucial in statistical timing analysis because the minimum clock period specified by them may dominate the circuit performance.

For hierarchical static timing analysis, the ILM and ETM methods are explained in Section 3.3.2. In the following, the basics of both methods are discussed to explain whether they are suitable to be enhanced to extract statistical timing models.

The ILM modeling method removes all circuit components which are not on the combinational paths from inputs to flip-flops or from flip-flops to outputs. The remaining part of the circuit is kept in the timing model intact. Therefore timing analysis at any level can be supported. Because the number of circuit components between flip-flops is large, they can not be kept in the timing model similar to the components in ILM. Therefore, the timing constraint from paths between all flip-flop pairs is usually specified by the minimum clock period in static timing analysis. When process variations are considered, the minimum clock period becomes to a random variable and should be checked during the timing verification of the complete design. If a standard statistical timing engine, e.g., [VRK+04], is used, the final result is a random variable representing the minimum clock period. With this random variable, the ILM timing model can be extended in a mixed style. This extension, however, conflicts with the concept of ILM, where combinational components are not collapsed for flexibility and accuracy, because the minimum clock period in this simple extension is computed at gate level and the more accurate timing information at transistor level is discarded.

The other timing model for flip-flop based circuits is ETM. This modeling method computes the maximum delay from each input to flip-flops and from flip-flops to

each output. The constraint between flip-flop pairs is also computed directly with a static timing engine. In this section, the ETM method is enhanced with a standard statistical timing engine to extract the constraints. The proposed method has short runtime during timing model extraction and timing verification, with accuracy still well maintained.

Unlike combinational circuits, a sequential circuit runs at a specified clock period. The target of statistical analysis is to compute the yield of the circuit at a given clock period, or at different clock periods for chip binning. For the most flexibility, the timing model of a sequential circuit should not be extracted against a specified clock period. Therefore, the clock period $T$ can only be assumed as an unknown fixed value during timing model extraction.

The minimum clock period extracted from each module should be verified for the complete design. In static timing analysis, this can be done by specifying the minimum clock period of the complete design and force each module to meet such constraint during their own development. When process variations are considered, this method can not work anymore because the minimum clock period of each module becomes a random variable. The final minimum clock period of the design must be computed from the constraints of all modules together; separately checking such constraint for each module against a specified clock period can not result correct yield. Additionally, this computation must be performed again anytime when a module is changed during design iteration, because such change may affect the correlation between the minimum clock periods of the modules. Consequently, the minimum clock period for a module should be included in its model when process variations are considered. This is a supplement to the method in [ALS+02, GVTG08, GVTG09], leading to more accurate statistical timing models for flip-flop based circuits.

When a flip-flop based module is instantiated in a hierarchical design, its inputs are connected to the outputs of previous modules, and its outputs to the inputs of following modules. Observed at the boundary of a module, the timing constraints can be split into three parts. The first part of the constraint is the setup constraints of flip-flops with combinational fanin paths from other flip-flops; the second part of the constraint specifies the setup constraints of flip-flops inside the module, but the fanin paths are through inputs; the last part of the constraint is for flip-flops inside following modules with partial fanin paths inside the current module.

Using Algorithm 3, the constraint from all flip-flop pairs is computed in one traversal of the virtual combinational circuit in Section 2.5. All sum and maximum computations in Algorithm 3 are performed using the method in [VRK+04] statistically. The resulting minimum clock period for the module, denoted as $D_F$, is in a parametrized statistical form. The constraint from flip-flop pairs is therefore simpli-

fied as constraint $C_F$.

$$C_F : \qquad D_F \leq T \tag{4.7}$$

This constraint is the one missed in [GVTG08,GVTG09].

The second split constraint specifies that the setup time constraints of flip-flops connected with inputs should be met. Similar to the timing constraint for flip-flop pairs, the arrival time from an input $k$ to a flip-flop $j$ can be written as

$$\tilde{A}_k + \Delta_{kj} \leq T - s_j \Longleftrightarrow \tilde{A}_k + \Delta_{kj} + s_j \leq T \tag{4.8}$$

where $\tilde{A}_k$ is the arrival time at input $k$ relative to the current clock phase of $j$. When extracting the timing model, no assumption should be made about $\tilde{A}_k$. For input $k$, there may be more than one fanout flip-flop. The arrival times at the inputs of all these fanout flip-flops must meet the constraint in the form of (4.8). By combining these constraints together, the timing constraint at input $k$ can be written as

$$C_{I_k} : \qquad \max_j\{\tilde{A}_k + \Delta_{kj} + s_j\} \leq T \Longleftrightarrow \tag{4.9}$$

$$\tilde{A}_k + \max_j\{\Delta_{kj} + s_j\} \leq T \Longleftrightarrow \tag{4.10}$$

$$\tilde{A}_k + D_{I_k} \leq T \tag{4.11}$$

where $D_{I_k}$ is computed for all flip-flops $j$ which have direct edges coming from input $k$ in the reduced timing graph.

For an input $k$, $D_{I_k}$ can be computed by propagating arrival times from $k$, where the arrival time $\tilde{A}_k$ is temporarily set to 0. Finally, each constraint of $C_F$ and $C_{I_1}, \ldots C_{I_m}$ is represented by a random variable respectively. These random variables need not to be updated if a module is not changed during design iteration. When verifying the timing of the complete design, only these $m+1$ variables are involved for the module.

If a circuit with $m$ inputs is used as a module in a hierarchical design, the probability that the complete circuit works properly with clock period $T$ can be computed as

$$Yield = Prob\{C_a, C_F, C_{I_1}, \ldots C_{I_m}\} \tag{4.12}$$

where $C_a$ represents the timing constraint for other modules in the design, and is also in the same form as $C_F$ and $C_{I_k}$ in their timing models. The probability in (4.12) is computed with all the constraints $C_a, C_F, C_{I_1}, \ldots C_{I_m}$ are true at the same time. The acceleration of timing verification with timing models comes from the fact that $D_{I_1}, \ldots D_{I_m}$ and especially $D_F$ are much simpler than their counterparts in the original circuit.

When a circuit is used as a module in a hierarchical design, its outputs are connected to the inputs of following modules. For example, when the output $l$ of a module is

connected to the input $k$ of another module, the arrival time at $k$ is determined by the arrival time at $l$. In order to verify the timing constraints for the fanout flip-flops of $k$, a timing model should also contain the delay information at all its outputs. Normally the output $l$ has more than one fanin flip-flop. After the latching clock edge, data signals are propagated from all these flip-flops $i$ to $l$. The data stable time or arrival time $D_{O_l}$ to $l$ is computed as

$$D_{O_l} = \max_i \{q_i + \Delta_{il}\} \tag{4.13}$$

Assuming there are $n$ outputs in the module, the $n$ arrival times $D_{O_1}, \ldots D_{O_n}$ are also included in the timing model. Combining with the setup time constraints, the timing model for a flip-flop based circuit contains only $m + n + 1$ random variables.

## 4.3 Timing Model Extraction for Latch Based Circuits

Latch based circuits have advantages compared to flip-flop based circuits when process variations are considered. Because of transparency, the path delay between a pair of latches can be compensated by the delays in the next stages. This is a remarkable advantage of latch based circuits because the delays of paths can be canceled statistically after manufacturing [HB06]. Although the levels of transparency in different chips after manufacturing may differ from each other, the functions of these chips are still correct. Because of latch transparency, an arrival time can propagate through several latch stages. At each stage, the arrival time must meet the setup time constraint of the corresponding latch. This leads to complexity in timing analysis of such type of circuits. By using extracted timing models, however, this timing analysis can be accelerated in several orders of magnitude. Therefore the application of such type of circuits can be expanded in practice.

For latch based circuits, different methods are already proposed to extract timing models for hierarchical static timing analysis. However, the limitations of these methods makes the direct extension of them to incorporate process variations difficult. The method in [MKB02] keeps all latches in the timing model to allow arbitrary latch transparency. Consequently, hierarchical timing analysis with such timing models is still time-consuming because the number of latches is not reduced. The second method is proposed in [VPMS97, ALS+02], where the level of latch transparency is assumed. This assumption is too strict because latch transparency can not be fixed during design time. After manufacturing, even the transparency levels in different chips may be different.

In this section, a statistical timing model extraction method for latch based circuits will be explained. This method does not make any assumption on the level of latch transparency. Instead, statistical conservative transparency level is used for

constraint extraction. Unlike other methods for statistical timing analysis of latch based circuits, such as [CZ04, ZTC+06], the proposed method only assumes that the clock period is an unknown fixed value. Therefore the extracted timing models can be used in designs with different clock specifications.

### 4.3.1 Timing Specification with Inputs for Latch Based Circuits

The basics of timing analysis for latch based circuits are explained in Section 2.3. All timing variables, including arrival times, departure times, times of enabling clock edge and latching edge etc., are specified with respect to the origin of the local time zone. With these variables, the timing constraint at a latch is expressed in (2.10)-(2.11). Because of latch transparency, arrival times at consecutive latch stages are dependent, as shown in (2.10). In static timing analysis, the minimum clock period is computed by transforming the timing specifications into a linear programming problem. This method dos not work when process variations are considered, because all constraints and the optimization target are specified with random variables. To solve this problem, structural methods, such as [CZ04, ZTC+06], are proposed for statistical timing analysis. However, both methods compute the yield of the circuit against a given clock period, so that are not usable in timing model extraction.

As used in (2.7), edge delays $\Delta_{ij}$ in the reduced timing graph are needed to specify timing requirements. These delays are computed using Algorithm 2. All sum and maximum computations during such processing are performed with a statistical timing engine. Consequently, all edge delays in the reduced timing graph become parametrized random variables.

The timing specification of (2.10) expresses that the arrival time at a latch depends on all the arrival times at its fanin latches. When used in a hierarchical design, the constraints from inputs of a latch based circuit should also be specified. Consider latch $j$ has an edge from input $k$ in the reduced timing graph. The arrival time at $j$ is determined by the arrival times at the input and all fanin latches. Consequently, the arrival time $A_j$ in (2.10) is revised as

$$A_j = \max\{\max_i\{\max\{A_i, r_i\} + m_{ij}\}, \max_k\{\tilde{A}_k + \Delta_{kj}\}\} \tag{4.14}$$

where $\tilde{A}_k$ is the arrival time at input $k$, and is expressed in the local time zone of $j$. Similar to timing model extraction for flip-flop based circuits, no assumption about $\tilde{A}_k$ can be made during timing model extraction. Similar to (2.11), the timing constraint at latch $j$ is written as

$$\max\{\max_i\{\max\{A_i, r_i\} + m_{ij}\}, \max_k\{\tilde{A}_k + \Delta_{kj}\}\} + s_j \leq T \tag{4.15}$$

### 4.3.2 Timing Constraint Restructuring for Latch Based Circuits

When a latch based circuit is used as a module in a hierarchical design, the constraint (4.15) should be checked for each latch inside the module. This constraint is restructured in this section to split the constraints from inputs and from latches inside the module. The latter are compressed into only one random variable, so that the size of the timing model can be reduced.

As the first step, (4.15) is equivalent to that each input of the maximum plus $s_j$ is smaller than the clock period $T$, i.e.,

$$\max_i\{A_i + m_{ij}\} + s_j \leq T \tag{4.16}$$

$$\max_i\{r_i + m_{ij}\} + s_j \leq T \tag{4.17}$$

$$\max_k\{\tilde{A}_k + \Delta_{kj}\} + s_j \leq T \tag{4.18}$$

where the first two maximum operations are performed with all fanin latches $i$ of $j$. The last maximum is performed with all fanin inputs $k$ of latch $j$.

For a fanin latch $i$, $A_i$ in (4.16) can be substituted further with the form of (4.14). The constraint after this substitution can be split into three parts similarly as

$$\max_p\{A_p + m_{pi}\} + m_{ij} + s_j \leq T \tag{4.19}$$

$$\max_p\{r_p + m_{pi}\} + m_{ij} + s_j \leq T \tag{4.20}$$

$$\max_q\{\tilde{A}_q + \Delta_{qi}\} + m_{ij} + s_j \leq T \tag{4.21}$$

where the first two maximum operations are performed with all fanin latches $p$ of $i$, and the last maximum is performed with all fanin inputs $q$ of $i$.

According to (4.16)-(4.21), the arrival time after each substitution is shifted by one latch stage backwards. The constraints (4.17) and (4.20) define that the data signals starting from the enabling clock edges of the latches in range of two stages before $j$ should meet the setup time constraint at $j$, where all latches in between are considered as transparent. Similarly, the arrival time from any input in this range should also meet such timing constraint, as defined by (4.18) and (4.21).

By repeating the substitution backwards through all fanin latches recursively, it can be observed that the arrival times starting from the enabling clock edges of all latches in the fanin cone of $j$ must meet the timing constraint of $j$, because new constraints similar to (4.20) are created after each substitution. For any inputs in the fanin cone of $j$, similar constraints can be inferred. Because each latch in the circuit has a constraint like (4.15), the recursive substitution above can be run for all latches. From the viewpoints of arrival times starting from enabling clock edges, the timing constraints for all latches together can be described as

$L_1$: The arrival time from the enabling clock edge of any latch to all latches in its fanout cone must meet the setup time constraints of these latches, with all intermediate latches assumed transparent.

Similarly the timing constraints for inputs can be described as

$L_2$: The arrival time from any input must meet the setup time constraints of all latches in its fanout cone, with all intermediate latches assumed transparent.

For simplicity, the arrival times mentioned in the following are all with the latch transparency assumption.

After each backward substitution from (4.16) to (4.21), a constraint similar to (4.19) is created. Therefore, this backward substitution can be performed infinitely from any latch across all paths. Because any signal propagation in a latch based circuit starts from the reset state, the infinite backward substitution can eventually reach the state just after reset. At this stage, the arrival time in the new created constraint similar to (4.19) is the time that the corresponding latch goes out of the reset state. As implicitly used in [ZTC+06, SMO90a], this arrival time is equal to the time of the corresponding enabling clock edge. Therefore, this new constraint can also be covered by $L_1$ and no further substitution is needed. Because $L_1$ and $L_2$ are derived from (4.15) and can cover all the constraints created from (4.15), they together specify the timing constraints of a latch based circuit completely.

In a reduced timing graph, there are loops across latch nodes. An example of such loop is illustrated in Figure 4.3. According to $L_1$, the arrival time starting from the enabling clock edge of latch $i$ must meet the timing constraints at all following latches. The arrival time from latch $i$ can go through the loop $i \rightarrow j_1 \cdots \rightarrow j_2 \cdots \rightarrow j_3$ and back to $i$. Thereafter, it can continue to propagate across the loop further.

After propagating across each latch, the delay shift defined in (2.7) is added to compute the arrival time at the latch of the next stage. For convenience of the following discussion, *cumulative delay shift* in (3.30) is extended for any path starting

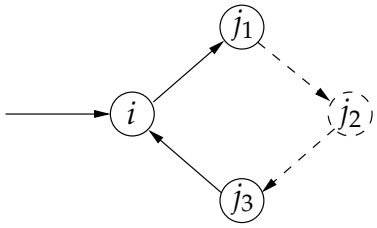


**Figure 4.3:** Loop Example in Reduced Timing Graph

from latch $i$ to $j$, denoted as $M_{i\to j}$, which is the sum of all delay shifts across the path and is formulated as

$$M_{i\to j} = \sum_{e_{st}\in \boldsymbol{E}_{ij}} (q_s + \Delta_{st} - \varepsilon_{st}) \tag{4.22}$$

where $\boldsymbol{E}_{ij}$ is the set of all edges on the specified path. $q_s$ is the propagation delay of latch $s$. $\Delta_{st}$ is the edge delay between $s$ and $t$ in the reduced timing graph. $\varepsilon_{st}$ is the phase shift of the clock phases of $s$ and $t$. If an arrival time traverses from node $i$ to $j$, the arrival time at $j$ can be expressed as

$$A_j = A_i + M_{i\to j} \tag{4.23}$$

For a loop in the reduced timing, e.g., Figure 4.3, the cumulative delay shift starting from $i$ and looping back to $i$ is denoted as $M_{i\to i}$. According to the proof in Section 3.2.4, any loop must be nonpositive, i.e., for any loop,

$$M_{i\to i} \leq 0 \tag{4.24}$$

From this observation, the third constraint for a latch based circuit can be described as

| |
|---|
| $L_3$: All loops in the reduced timing graph with statistical delays must be nonpositive. |

The constraint $L_3$ is also used in [ZTC+06], but not to simplify the complete timing specification of latch based circuits.

The constraint $L_1$ specifies that the setup time constraint at each latch should be checked even after infinite loops. With $L_3$ specified, the constraint checking of $L_1$ can stop after a loop is traversed only once. This is because that the arrival time from the enabling edge of latch $i$ to latch $j$ after traversing a loop is smaller than the arrival time when $j$ is reached the first time, so that the constraint after a loop is always dominated by the constraint before a loop is traversed, as formulated following

$$r_i + M_{i\to j} + M_{j\to j} + s_j \leq r_i + M_{i\to j} + s_j \leq T \tag{4.25}$$

where the loop is formed from $j$ and back to $j$. The property in (4.25) holds also for timing constraints propagated from inputs. With $L_3$ as condition, $L_1$ and $L_2$ can be revised to $L_{R1}$ and $L_{R2}$.

| |
|---|
| $L_{R1}, L_{R2}$: The constraints of $L_1$ and $L_2$ without visiting latches after loops, respectively. |

With $L_{R1}$, $L_{R2}$ and $L_3$ together, the yield of a hierarchical design using a latch based module can be written as

$$Yield = Prob\{L_a, L_{R1}, L_{R2}, L_3\} \tag{4.26}$$

where $L_a$ is the timing constraint set for the latches in other modules.

In following sections, statistical timing model extraction for latch based circuits will be explained. The basic idea is that each constraint $L_{R1}$, $L_{R2}$ and $L_3$ is replaced by a simpler form in the timing model to compress the constraints from the original circuit. As an example, the maximum loop cumulative delay shifts will be computed and used to represent $L_3$. During timing verification of the complete design, these loops need not to be enumerated again. Instead, only the provided variable is verified against the clock period.

### 4.3.3 Path Traversal and Clock Scheme

The constraint $L_{R1}$ defines that the arrival time starting from the enabling clock edge of any latch must meet the setup time constraints of all latches in the fanout cone of the latch without through loops, with all latches in between assumed as transparent. In the reduced timing graph in Figure 2.2, there are many paths starting from a latch. According to $L_{R1}$, arrival times should be propagated through all these paths.

Although loops need not to be traversed individually, they cause interdependence between arrival times during propagation using block-based methods. For instance, the propagation starting from node 1 stops at node 2 and 3. Because of the backward edge from node 4 to 2, the computation of arrival time at either node 2 or node 3 requires that the other to be visited first. If a path-based method is used, the paths $1 \to 3 \to 4 \to 2$ and $1 \to 2 \to 3 \to 4 \to 2$ can be enumerated separately. However, using a direct path-based method is prohibitive in large circuits for the exploding number of paths.

For short runtime and acceptable accuracy, a block-based method is used to approximate the path traversal from a node in the reduced timing graph. Such approximation is also used in other methods of statistic timing analysis for latch based circuits, e.g., [CZ06]. To solve the problem of interdependence between arrival times, the method in [CZ06] uses a feedback loop breaking algorithm with heuristics.

The basic idea of the feedback loop breaking is explained in the following. The reduced graph is searched in depth-first order. During this search, if some fanin edges of the current visited latch $i$ originate from latches which are in the fanout cone of $i$, these edges are removed from the reduced timing graph. The removed edges are called *feedback edges*, because there are loops through them starting from $i$

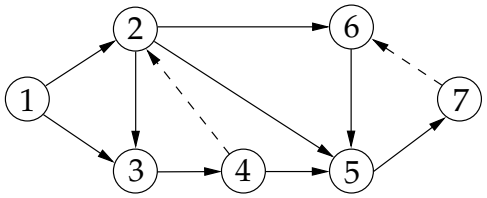


**Figure 4.4:** Reduced Timing Graph Example with Feedback Edge Removal

and ending at $i$. Consequently, the reduced timing graph becomes a directed acyclic graph. Arrival times can be propagated across this revised graph using a standard block-based statistical timing method. With latch 1 as starting latch, an example of the revised timing graph of Figure 2.2 is illustrated in Figure 4.4, where all nodes are assumed as latches and feedback edges are shown with dashed arrows. Note the result of feedback edge removal is not unique, depending on different traversal orders when searching feedback edges.

In Figure 4.4, the path $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 6$ is missing when visiting latches from 1. To solve this problem, it is proposed in [CZ06] to search the original reduced timing graph with different orders of nodes. Therefore, different feedback edges are broken during arrival time propagation. In this way, the probability of missing paths can be reduced.

In this thesis, arrival time propagation is simply run twice to reduce the runtime of timing model extraction. In the second run, the latch visiting order is the same as in the first run. The arrival times at source nodes of feedback edges created in the first run are updated to their sink nodes and propagated further. In this way, any path with one feedback edge is guaranteed to be traversed. An example is the path $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 7$ in Figure 4.4. The arrival time from node 1 to 4 is computed in the fist run. In the second run, this arrival time is used to update the arrival time at node 2 and propagated further.

Similar to the heuristic method in [CZ06], there are still missing paths with more than one feedback edge after this two-run traversal. However, traversing reduced timing graph twice already shows good accuracy for timing model extraction according to experimental results. In a reduced timing graph, paths with more than one feedback edge are relatively longer than other paths. As to be explained later, the arrival time propagation for $L_{R1}$ and $L_{R2}$ stops when the arrival time is smaller than the enabling clock edge of a latch. Therefore, long paths need not to be traversed completely. Additionally, the delays on a path compensate each other. As a result, long paths have less chance to affect the constraints. This explains why the accuracy is still acceptable when traversing the reduced timing graph only twice. For better path coverage, the traversal can be run more than twice, or the heuristic algorithm in [CZ06] can be used to replace the two-run traversal in this thesis.

Till now the latch traversal order for extracting timing constraints has been discussed. Like standard statistical timing methods, the sum and maximum computations are performed during the arrival time propagation. Each time when an arrival propagates across an edge, the delay of the edge is added to the arrival time. The arrival time at a latch is computed as the maximum of arrival times from all incident edges in the reduced timing graph. The difference in this computation from the standard statistical timing analysis method is that the delay shift $m_{ij}$ from latch $i$ to $j$ instead of a simple random variable is added to the arrival time, as shown in (4.14).

From the definition in (2.7), $m_{ij}$ is equal to $q_i + \Delta_{ij} - \varepsilon_{ij}$. $q_i$ and $\Delta_{ij}$ are known random variables, so that their sum can be computed easily. But $\varepsilon_{ij}$ can not simply be treated as a known random variable. As shown in Figure 2.4, $\varepsilon_{ij}$ is defined as the clock phase shift. If the clock phases are generated using an absolute delay based method, $\varepsilon_{ij}$ can be safely assumed as a known random variable. Therefore, the arrival time update is the same as in standard statistical timing analysis. The more complex case is that the clock phases are generated so that the relative clock phase shift is fixed, i.e., $\varepsilon_{ij}$ has a fixed ratio to the clock period. In the following, the second case will be studied only. This method can be easily adapted to handle the absolute clock phase shift.

In the second clock scheme, the clock phase shift changes proportionally when the clock period changes, so that $\varepsilon_{ij}$ is written as

$$\varepsilon_{ij} = \varsigma_{ij} T \tag{4.27}$$

where $\varsigma_{ij}$ is a positive constant. Similarly, the time of the enabling clock edge in the local time zone is also assumed as having a fixed ratio to the clock period, i.e.,

$$r_i = \varsigma_i T \tag{4.28}$$

where $\varsigma_i$ is a positive constant smaller than 1.

### 4.3.4 Timing Constraint Extraction from Enabling Clock Edges

To extract setup time constraints from the enabling clock edge of latch $i$, the arrival time $A_i$ is set to $r_i$ as initialization. The arrival time from $i$ to any following latch $j$ can be written as

$$A_j = r_i + M_{i \to j} = \varsigma_i T + \sum_{e_{st} \in \mathbf{E}_{ij}} (q_s + \Delta_{st} - \varsigma_{st} T) = D_{ij} + C_{ij} T \tag{4.29}$$

where $M_{i \to j}$ is the cumulative delay shift across the path from $i$ to $j$ defined in (4.22). All phase shifts in $M_{i \to j}$ are replaced by constant times of clock period, as assumed in (4.27). All known random variables are summed up and written as $D_{ij}$. The

same operation is done for the coefficients of clock period and the result is written as a constant $C_{ij}$. Because the extracted timing model is verified against different clock periods, $T$ can only be assumed as an unknown fixed value. Therefore, the part of clock period can not be merged with the random variables in (4.29). In the following, $D_{ij}$ and $C_{ij}$ in (4.29) are called *delay part* and *coefficient part* respectively.

When more than one arrival time in the form of (4.29) reaches latch $j$, their maximum should be computed. Because these arrival times can reach $j$ through different paths, their coefficients of $T$ may be different. Because $T$ is unknown, the arrival times with different coefficients of $T$ can not be merged simply. Instead, they are propagated in parallel. For the purpose of such parallel propagation, an arrival time at a latch is represented by a set of elements during timing model extraction for latch based circuits. Each element in such a set saves a delay part and a coefficient part shown in (4.29). When the maximum of two arrival times is computed, the coefficients of $T$ in the elements of these arrival times are first compared. The arrival time elements with the same coefficient of $T$ are merged by computing the maximum of their delay parts with a statistical timing engine. The coefficient of $T$ itself is unchanged in this computation. Thereafter, the resulting arrival time elements with different coefficients of $T$ are inserted into the arrival time set of the current latch.

When an arrival time propagates across an edge, the delay shift is added. Because the arrival time is a set of elements in the form of (4.29), the addition is performed to each element in the set. For each element, the random variables in the delay shift are added to the delay part. The coefficients of $T$ is also summed up in this operation.

The maximum of all the elements in an arrival time must meet the setup time constraint of the corresponding latch. For each element, the constraint can be written as

$$D_{ij} + C_{ij}T \leq T - s_j \iff \tag{4.30}$$

$$D_{ij} + s_j \leq (1 - C_{ij})T \iff \tag{4.31}$$

$$(D_{ij} + s_j)/(1 - C_{ij}) \leq T \tag{4.32}$$

where $1 - C_{ij}$ is positive because $C_{ij}$ is computed by subtracting the coefficient of $T$ from $\varsigma_i$ when traversing latch stages, as shown in (4.29).

In (4.32) all random variables and coefficients on the left side are known, so that $(D_{ij} + s_j)/(1 - C_{ij})$ can be treated as a known random variable. At each latch during arrival time propagation, such a constraint in the form of (4.32) for each element in the arrival time is created. After propagating arrival times from enabling clock edges of all latches, all these inequalities together form the constraint described by $L_{R1}$. Because all variables in these constraints should be smaller than $T$ to guarantee the correct circuit behavior with clock period $T$, these inequalities together are equivalent to the one that the maximum of all the random variables on the left

side of them is smaller than $T$. This maximum is denoted as $V_1$, with which the constraint $L_{R1}$ can be simply written as

$$V_1 \leq T \tag{4.33}$$

During arrival time propagation, each arrival time is represented by a set of elements in the form of (4.29). As the propagation recurs further, the numbers of elements in arrival times become large, so that the runtime to compute $V_1$ increases. In the following, the method to reduce the number of elements in an arrival time will be explained. Based on the discussion before, the arrival time from any enabling clock edge is propagated and setup time constraint is included implicitly in (4.33). During the propagation, if an element from an arrival time is smaller than the time of the enabling clock edge in that local time zone, the constraint created from propagating this element further is dominated by the constraint created from the arrival time propagation starting from the enabling clock edge. Therefore, such an element can be removed from the arrival time without affecting the timing constraint represented by (4.33). The condition for removing an element is written as

$$D_{ij} + C_{ij}T \leq r_j = \varsigma_j T \tag{4.34}$$

If $\varsigma_j - C_{ij}$ is positive, (4.34) is equivalent to

$$D_{ij}/(\varsigma_j - C_{ij}) \leq T \tag{4.35}$$

During arrival time propagation, $V_1$ increases gradually while the constraint (4.32) is merged to (4.33). To merge a constraint, the maximum of $V_1$ and the random variable at the left side of (4.32) is computed. $V_1$ is then updated with the result. When verifying the timing performance of a circuit, the constraint (4.33) will be true. Comparing (4.35) with (4.33), the former is dominated by the latter when (4.36) is true.

$$D_{ij}/(\varsigma_j - C_{ij}) \leq V_1 \tag{4.36}$$

Both sides of (4.36) are random variables, so that (4.36) can be true only with a certain probability. If the probability that (4.36) is true approximates 1, the removal of the corresponding arrival time element affects the timing model only with a very small probability. Therefore, the probability in (4.37) for each arrival time element during propagation is computed, as

$$p_r = Prob\{D_{ij}/(\varsigma_j - C_{ij}) \leq V_1\} \tag{4.37}$$

If $p_r$ is larger than a predefined constant $\delta_l$ approximating 1, the arrival time element can be removed from the arrival time.

Like the timing model extraction for flip-flop based circuits described in Section 4.2, the timing model for a latch based circuit should contain delays to the outputs of the circuit. During the arrival time propagation in this section, if the fanout of a latch is an output, the delay from this latch to the output and the arrival time are added together and stored as the output delay. From the analysis in Section 4.3.2, any arrival time can be considered as starting from an enabling clock edge or from an input initially. In the former case, if an arrival time element can reach an output without being removed at an intermediate latch, this arrival time element should be verified against the setup time constraint of the latches in the following modules. This explains the method to extract output delays from internal latches.

### 4.3.5 Timing Constraint Extraction from Inputs

After the timing constraint representing $L_{R1}$ is explained in Section 4.3.4, the timing constraint extraction from inputs, i.e., finding a simple form to represent $L_{R2}$, will be explained in this section.

The basic idea to extract timing constraint for an input is mostly the same as the one described in the previous section. The arrival time from an input is propagated across the reduced timing graph with feedback edge removal. At each latch, the maximum of the arrival times is computed and the setup time constraint is updated.

In the following, input $k$ is used as an example to extract timing constraints for it. Compared to (4.29), the starting arrival timing from input $k$ is $\tilde{A}_k$, which is unknown until the module is integrated into a hierarchical design. Similar to the method in Section 4.3.4, the arrival time at a latch becomes a set after $\tilde{A}_k$ is propagate across latches. Because $\tilde{A}_k$ can not be merged with the delay part or coefficient part, an element from the arrival time becomes

$$A_j = \tilde{A}_k + D_{kj} + C_{kj}T \tag{4.38}$$

If this arrival time is propagated across a latch stage further, the corresponding delay shift is merged with the right side of (4.38) by adding the random variables and the coefficients of $T$ respectively. Because all arrival time elements are in the form of (4.38) and share the same $\tilde{A}_k$, the maximum of two of them can be performed just like the maximum computation in Section 4.3.4 without considering $\tilde{A}_k$. The result of this maximum computation is still a set with $\tilde{A}_k$ implicitly appended.

At each latch, the setup time constraint from each element in the arrival time is extracted. An example of such a constraint is shown below.

$$\tilde{A}_k + D_{kj} + C_{kj}T \leq T - s_j \Longleftrightarrow \tag{4.39}$$

$$\tilde{A}_k + (D_{kj} + s_j) + C_{kj}T \leq T \tag{4.40}$$

Unlike (4.31)-(4.32), (4.40) can not be transformed similarly because this transformation causes the coefficients of $\tilde{A}_k$ to be different in the constraints from arrival time elements.

To represent $L_{R2}$, a constraint set $\mathcal{C}_k$ for the input $k$ is created. Each element from such a constraint set is in the form of (4.40). After an arrival time set is computed, timing constraint from each element of the set is created. The coefficient of $T$ in the new constraint is compared with the coefficient of each element in $\mathcal{C}_k$. If there is a match, only the random variable $D_{kj} + s_j$ in (4.40) is merged with the corresponding variable of the constraint element. Otherwise, a new constraint is simply inserted into $\mathcal{C}_k$.

Similar to compressing arrival times in Section 4.3.4, each arrival time element is compared with the time of the enabling clock edge. An example of such comparison for latch $q$ is shown in (4.41).

$$\tilde{A}_k + D_{kq} + C_{kq}T \leq r_q = \varsigma_q T \tag{4.41}$$

Consider that there is already a set of constraints $\mathcal{C}_k$ for input $k$, and each element in this set is in the form of (4.40). After subtracting both sides of (4.40) from (4.41), the result is written as

$$(D_{kq} - D_{kj} - s_j) + (C_{kq} - C_{kj})T \leq (\varsigma_q - 1)T \Longleftrightarrow \tag{4.42}$$

$$D_{kq} - D_{kj} - s_j \leq (\varsigma_q - 1 - C_{kq} + C_{kj})T \tag{4.43}$$

If (4.42) is true, the arrival time element can be removed because (4.41) is dominated by (4.40). If $\varsigma_q - 1 - C_{kq} + C_{kj}$ is positive, (4.43) is equivalent to

$$(D_{kq} - D_{kj} - s_j)/(\varsigma_q - 1 - C_{kq} + C_{kj}) \leq T \tag{4.44}$$

Similar to (4.35)-(4.37), if the probability $p_i$ is larger than $\delta_l$, the arrival time element can be removed, where $p_i$ is defined as

$$p_i = Prob\{(D_{kq} - D_{kj} - s_j)/(\varsigma_q - 1 - C_{kq} + C_{kj}) \leq V_1\} \tag{4.45}$$

As in Section 4.3.4, the delays to outputs are also created if a fanout is an output during the propagation. The only difference is that the output delays depend on the arrival time $\tilde{A}_k$ at the input.

### 4.3.6 Nonpositive Loop Constraint Extraction

The last constraint for a timing model is $L_3$, which specifies all feedback loops in the reduced timing graph should be nonpositive. In this thesis, the two-run traversal method used in Section 4.3.4 is adapted to compute the maximum loop delays.

Other loop breaking algorithms, e.g. [CZ06], can also be used for better path coverage.

The basic idea is to compute the maximum arrival time starting from each latch and looping back to it again. At first, the arrival time at the starting latch is set to 0. Arrival times are propagated using the two run traversal in Section 4.3.4, but without updating latch setup time constraints. During the propagation, if a direct fanout latch is the starting latch, a loop is formed. In this case, the delay shift between the current latch and the starting latch is added to the arrival time of the current latch to compute the maximum loop delay. As an example, assume that the fanout latch $j$ of the current latch $i$ is the starting latch. By summing the delay shift from $i$ to $j$ and the arrival time at $i$, the maximum of the cumulative delay shifts of the loops which are traversed can be computed. Because each loop should be nonpositive, this maximum should be less than or equal to 0. Consider an element $D_{ji} + C_{ji}T$ in the arrival time $A_i$, the loop constraint can be written as

$$D_{ji} + C_{ji}T + m_{ij} \leq 0 \Longleftrightarrow \tag{4.46}$$

$$D_{ji} + C_{ji}T + q_i + \Delta_{ij} - \varsigma_{ij}T \leq 0 \Longleftrightarrow \tag{4.47}$$

$$(D_{ji} + q_i + \Delta_{ij})/(\varsigma_{ij} - C_{ji}) \leq T \tag{4.48}$$

where $\varsigma_{ij} - C_{ji}$ is positive.

Similar to $V_1$, a random variable $V_3$ is created to represent the constraint that all loops are nonpositive. Each time when a constraint like (4.48) is created, the maximum of $V_3$ and the variable at the left side of (4.48) is computed. $V_3$ is thereafter updated with the result of this maximum computation. After the loop traversals from all latches are fulfilled, all loop constraints are merged into $V_3$. Therefore, the constraint $L_3$ can be represented by

$$V_3 \leq T \tag{4.49}$$

After the loop paths from a latch are traversed, this latch is marked as visited. This means the nonpositive constraints for all loops through this latch has been specified. Therefore, arrival times need not to be propagated through visited latches in the traversal starting from other latches. This can reduce the runtime of the loop constraint extraction remarkably.

### 4.3.7 Summary of Timing Model Extraction for Latch Based Circuits

As described by (4.26), constraints $L_{R1}$, $L_{R2}$ and $L_3$ are used to verify the timing of a latch based module. $L_{R1}$ and $L_3$ are specified in the proposed method simply by (4.33) and (4.49), where $V_1$ and $V_3$ are known random variables computed from the

original circuit during timing model extraction. (4.33) and (4.49) can also be merged into one constraint by computing the maximum of $V_1$ and $V_3$.

To specify $L_{R2}$ for an input $k$, the constraint set $\mathcal{C}_k$ is used. Each element in $\mathcal{C}_k$ is in the form of (4.40). When verifying the timing performance of a hierarchical design, $\tilde{A}_k$ is computed from the modules logically before the current module. As $\tilde{A}_k$ becomes known, (4.40) can be rewritten as

$$(\tilde{A}_k + D_{kj} + s_j)/(1 - C_{kj}) \leq T \tag{4.50}$$

As all the constraints can be written in the similar form in (4.33), (4.49) and (4.50), the constraints related to the current module in (4.26) can be easily represented by the maximum of all the random variables in (4.33), (4.49) and (4.50), where (4.50) is computed for the constraint elements of all inputs. Compared to directly verifying the timing performance of a latch based circuit, the timing constraints contained in an extracted timing model are very simple so that the statistical timing analysis of a hierarchical design can be accelerated drastically.

## 4.4 Summary

In this chapter timing model extraction methods considering process variations are proposed for common circuit types. Because gate delays are modeled as random variables, methods depending on special patterns in gate delays for combinational circuits are not feasible anymore. As a solution, delay edges with maximum criticalities smaller than a predefined threshold are removed from original circuits. After applying the basic merge operations, statistical timing models for combinational circuits are extracted. For flip-flop based circuits, the extracted timing model in [ALS+02] is extended with a statistical timing engine directly. For latch based circuits, the classical clock model proposed in [SMO90b] and discussed in Section 2.3 is restructured and corresponding timing constraints are extracted as the timing model. Because the circuit components between flip-flops or latches are compressed into only one constraint, the extracted timing models for sequential circuits have very small size compared to the original circuits.

# Chapter 5

# Correlation Handling in Hierarchical Statistical Timing Analysis

Spatial correlation exists between modules in hierarchical designs, as discussed in Section 3.4.2. During the timing model extraction explained in Section 4, a gate delay is a function of independent random variables. These variables are from the decomposition of correlated process parameters, e.g., using principal component analysis. In this decomposition, the correlation matrix of process parameters is needed. Because only the layout of the gates inside each individual module is known, the correlation matrix is generated in the range of the module, e.g., using the method in [XZH07]. Therefore, the gate delays used during timing model extraction contain the correlation information only inside the module. Because the correlation between modules in a hierarchical design can only be determined after these modules are instantiated, an additional step to handle such correlation during the timing verification of the complete design is needed to maintain the accuracy of timing analysis.

In this chapter, a method to incorporate spatial correlation between modules will be explained. This method substitutes the independent random variables in timing models by the variables for the complete design. The correlation between modules is thus represented by sharing the same set of independent random variables in the constraints of the timing models. Compared to the method proposed in [GVTG08,GVTG09], the method in this chapter always has a solution for the variable substitution, so that the limitation of the method in the previous approach is overcome. Details of this limitation are already explained in Section 3.4.2. With a heterogeneous grid, the method in this chapter can also handle the delay correlation of gates directly implemented in the top design.

## 5.1 Correlation Handling with Variable Substitution

As shown in Section 3.1.3, the die area is partitioned into a grid. For each grid cell, a random variable is assigned to represent the variation of a process parameter. A delay of a gate belonging to a grid cell is modeled as a function of the random variables for the corresponding grid cell. Because the same process parameters of different gates inside a grid cell are represented by the same random variables, the number of random variables used to model process variations is reduced. Owing to spatial correlation, the variables assigned to grid cells are correlated, with a precharacterized correlation matrix computed from the on-die distance.

The basic idea of correlation handling in this chapter is to establish the relation between the independent random variables for the complete design and the ones in timing models. This mapping is based on the same die partition mechanism of the die area covered by a module during model extraction and during the timing analysis of the complete design. The basic idea is illustrated in Figure 5.1. In the first step, the die areas covered by modules are partitioned with the same grids as during timing model extraction. In Figure 5.1 the die areas covered by module A and B are first partitioned using the default cell size and starting from their own origins respectively, as they are partitioned for timing model extraction. In the second step, the remaining die area which is not covered by modules is partitioned with the default cell size. All these grid cells together are considered as the grid partition for the complete design. Because the origins of modules may move freely during layout, the grid cells of the complete design may have irregular shapes and sizes. Examples are the ones in gray in Figure 5.1. Compared to the uniform partition in [CS03] illustrated in Figure 3.4, the result of the two-step partition is called *heterogeneous grid*. Because the size of each grid cell is no larger than the default cell size, this heterogeneous partition does not lose any modeling accuracy.

For each grid cell of the hierarchical design, a random variable is assigned to model the within-die variation, even though some grid cells are not regular. Assuming there are totally $m$ grid cells after partitioning the die of the top design and only

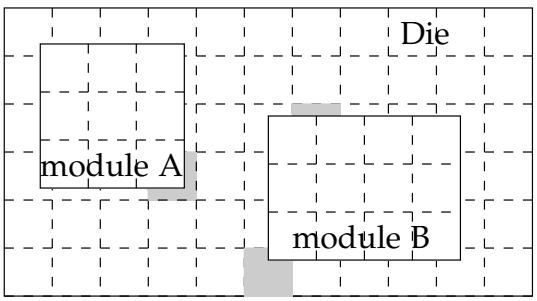


**Figure 5.1:** Heterogeneous Grid

one parameter is considered, $m$ random variables are assigned to these grid cells, written as a vector $\boldsymbol{p}^t$, with an $m \times m$ covariance matrix $\boldsymbol{C}^t$. By applying principal component analysis, $\boldsymbol{p}^t$ can be decomposed as

$$\boldsymbol{p}^t = \boldsymbol{B}^t \boldsymbol{x}^t \tag{5.1}$$

where $\boldsymbol{B}^t$ is the orthogonal transformation matrix. $\boldsymbol{x}^t$ is the vector of independent random variables.

In the following, module B is used as an example to explain the independent random variable substitution. Other modules can be processed similarly. Because the area covered by module B is partitioned in the same way as during model extraction, the number of grid cells inside this area is the same, denoted as $n$. Without losing generality, the random variables for grid cells of the die area covered by module B in the hierarchical design are indexed from 1 to $n$ in $\boldsymbol{p}^t$, and denoted as $\boldsymbol{p}_n^t$. The correlation between $\boldsymbol{p}_n^t$ is represented by the $n \times n$ submatrix $\boldsymbol{C}_n^t$ at the upper-left corner of $\boldsymbol{C}^t$. Considering only the first $n$ random variables in (5.1), the decomposition of $\boldsymbol{p}_n^t$ can be written as

$$\boldsymbol{p}_n^t = \boldsymbol{B}_n^t \boldsymbol{x}^t \tag{5.2}$$

where the $n \times m$ matrix $\boldsymbol{B}_n^t$ contains the first $n$ rows of the transformation matrix $\boldsymbol{B}^t$.

During timing model extraction, the $n$ random variables assigned to module $B$ are denoted as $\boldsymbol{p}_n$, with correlation matrix $\boldsymbol{C}_n$. Using principal component analysis, $\boldsymbol{p}_n$ can be decomposed as

$$\boldsymbol{p}_n = \boldsymbol{A}\boldsymbol{x} \tag{5.3}$$

where $A$ is the $n \times n$ orthogonal transformation matrix. $\boldsymbol{x}$ is the vector of independent random variables. All delays inside a timing model are represented as functions of the independent random variables $\boldsymbol{x}$.

The matrix $\boldsymbol{C}_n^t$ and $\boldsymbol{C}_n$ both represent the correlation between the variables for the grid cells inside the area of module B. The correlation of two grid cells is computed from the on-die distance between them, regardless of whether this computation is done during timing model extraction or the timing analysis of the complete design. Consequently, $\boldsymbol{C}_n^t$ and $\boldsymbol{C}_n$ are completely equal. With the same correlation matrix, the Gaussian random variable vectors $\boldsymbol{p}_n^t$ and $\boldsymbol{p}_n$ are statistically equal, because they represent the variations of the same process parameter in different grid cells and therefore have the same mean and variance vectors.

Comparing (5.2) and (5.3), $\boldsymbol{B}_n \boldsymbol{x}^t$ and $\boldsymbol{A}\boldsymbol{x}$ are decompositions of the same set of random variables. The difference between them is that in (5.2) the decomposition contains more random variables because $m$ is larger than $n$. Although some random variables in $\boldsymbol{x}^t$ are redundant in representing correlation defined by $\boldsymbol{C}_n^t$ or $\boldsymbol{C}_n$, they are essential for defining the correlation between modules and grid cells at the top

level. From (5.2) and (5.3), the relation between $\boldsymbol{x}^t$ and $\boldsymbol{x}$ can be established as

$$\boldsymbol{x} = \boldsymbol{A}^T \boldsymbol{p}_n \tag{5.4}$$
$$= \boldsymbol{A}^T \boldsymbol{B}_n^t \boldsymbol{x}^t \tag{5.5}$$

where $\boldsymbol{A}^T = \boldsymbol{A}^{-1}$ because $\boldsymbol{A}$ is orthogonal.

In statistical timing analysis, the decomposition is applied to accelerate the computation of the correlation between random variables and their variances, e.g., in [VRK+04]. As shown in (5.2) and (5.3), there exists more than one decomposition for the correlated random variables. In the statistical engine [VRK+04] used in this thesis, the statistical maximum computation needs only the correlation between two arrival times. Any set of decomposition can produce the same tightness probability in [VRK+04]. Therefore, whether the decomposition of $\boldsymbol{x}^t$ or $\boldsymbol{x}$ is used does not change the result of the timing model extraction except the different sets of random variables used.

With the transformation in (5.4), the independent random variables inside the timing model are transformed back to the random variables modeling process variations. These correlated random variables can be further replaced by (5.2), therefore the delays inside the timing model are represented by the variable set of the complete design. The two steps of the transformation can be performed together, as shown in (5.5). Conceptually, the timing model after this transformation can be thought as directly generated during the timing analysis of the complete design. In this case the die area is partitioned and random variables are assigned and decomposed with (5.1). Thereafter, the module itself is processed as explained in timing model extraction. The resulting timing model is used in hierarchical analysis, with the correlation contained by sharing the same set of independent random variables from decomposition.

With the transformation in (5.5), the correlation between all modules can be established. For combinational circuits, the extracted timing model is still a timing graph. For sequential circuits, the constraints at all inputs of each module should be checked. The arrival times at these inputs are computed from the delays to the outputs of the modules which logically precede to the current module. The minimum clock period is computed as a random variable, by merging all constraints, e.g., as described in Section 4.3.7.

## 5.2 Discussion

In this section several issues about the application of the proposed variable substitution method in hierarchical statistical timing analysis will be discussed. These

issues include the cooperation of the proposed method with existing statistical timing engines, the loop problem in hierarchical designs with modules and whether the variable substitution should be performed during timing model extraction or during the timing analysis of the complete design.

Conceptually, the independent random variables are first mapped back to the ones representing correlated process variations. This substitution is valid for the linear statistical methods [VRK+04] and [CS03]. In [VRK+04], the tightness probability is computed with the correlation of two arrival times. This correlation is the same whether the decomposed independent random variables or the original correlated random variables are used. The former is used in [VRK+04] to reduce the runtime of the correlation computation. Because the maximum and sum computations are both linear combinations of arrival times with variance mapping, the coefficients in the results from both computations do not depend on the decomposed variable set, but only on the correlation between these arrival times and their moments. From this analysis, either using $\boldsymbol{x}$ or $\boldsymbol{x}^t$ results in the same minimum clock period from timing analysis. The method in [CS03] computes the correlation between the maximum of two arrival times and each independent random variable using the formulas in [Cla61]. Because each independent random variable can be considered as a linear combination of the original correlated random variables, this method does not depend on a specific decomposition. Therefore it can use the proposed variable substitution to handle correlation between modules. For the second order statistical timing analysis methods, e.g., [ZSL+05, FLZ07], the variable substitution causes the number of terms in arrival times to increase drastically. For example, if a second order term in a quadratic form is replaced by a linear combination of another set of independent random variables, many new cross terms are created. To remove such cross terms, the orthogonalization in [ZSL+05] may be used. This transformation, however, increases the runtime of timing analysis when the number of cross terms is very large.

The second issue is about the loops in the hierarchical design with modules. For combinational circuits, all loops should be broken by registers during design stage. For latch based circuits, however, loops may exist because an arrival time can pass through different modules and loop back, with all latches in between in transparency. When using the timing model proposed in Section 4.3, the latch levels inside modules are compressed. The transparency from inputs to outputs are directly modeled by delays between them. This compression makes the loops smaller compared to the ones in the original circuits, and therefore accelerates the nonpositive loop verification of the complete design, as described in Section 4.3.2.

The last issue is when the transformation (5.4) from independent variables inside a module to the original variables should be performed. If this is done during timing model extraction, the transformation during the timing verification of the complete design can be simplified because the matrix product computation in (5.5)

needs not to be computed. This method, however, increases the runtime of timing model extraction. Because the runtime of the timing verification of the complete design is already very short, the increase of it by the variable transformation does not perceivably make the timing analysis of the complete design become slow. In practice, whether the variable transformation is run during the timing analysis of the complete design or during timing model extraction should be decided by engineers according to different design flows.

## 5.3 Summary

Because of the proximity effect during manufacturing, within-die variations exhibit correlation depending on the distances between circuit components on the die. With the method proposed in this chapter, the relation between the random variables inside timing models and the random variables for the complete hierarchical design is established. Therefore, the correlation between submodules are represented by sharing the same set of independent random variables after the variable substitution. In contrast to the method described in [GVTG08,GVTG08], the proposed method can always produce a solution during the variable substitution step and thus guarantee the validity of the correlation handling in any case.

# Chapter 6

# Experimental Results

In this chapter, the results of the proposed methods for timing model extraction and correlation handling are shown. The proposed methods in Section 4 are applied to ISCAS85 and ISCAS89 benchmark circuits to extract timing models. The sizes of the extracted timing models and of the original circuits are compared to prove the efficiency of proposed methods. To verify accuracy, maximum delays between inputs and outputs of combinational circuits are compared to the results from Monte Carlo simulation; for sequential circuits, timing models are tested in a random generated application context and also compared to the results from Monte Carlo simulation. Thereafter, the accuracy of the variable substitution method in Section 5 is evaluated by applying it to hierarchical designs with modules from ISCAS85 benchmarks. The focus of this test is the effect of the spatial correlation between modules and the accuracy of timing analysis using the proposed method.

## 6.1 Experiment Setup

The proposed methods in this thesis use a statistical timing analysis algorithm as engine, i.e., the maximum and sum mentioned in previous sections are all computed statistically. In the experiment, the algorithm proposed in [VRK+04] was used for such statistical computations. As explained in Section 3.2.3, this algorithm models gate delays as linear functions of process parameters. The maximum computation is very simple with tightness probability. The accuracy of this algorithm, however, is still very good by preserving correlation between arrival times and gate delays efficiently. Because the proposed timing model extraction methods do not depend on a specific statistical engine, methods with higher order or non-Gaussian gate delay models, e.g., [ZSL+05, ZCH+05, SS06], can also be used for better accuracy.

With a 90nm library from an industry partner, the delay of each type of gate was characterized as a linear function of transistor length, oxide thickness and threshold

**Table 6.1:** ISCAS85 Benchmarks

| Circuit | num. of inputs | num. of outputs | num. of gates | num. of edges | num. of nodes |
|---|---|---|---|---|---|
| c432 | 36 | 7 | 160 | 336 | 196 |
| c499 | 41 | 32 | 202 | 408 | 243 |
| c880 | 60 | 26 | 383 | 729 | 443 |
| c1355 | 41 | 32 | 546 | 1064 | 587 |
| c1908 | 33 | 25 | 880 | 1498 | 913 |
| c2670 | 233 | 140 | 1193 | 2076 | 1426 |
| c3540 | 50 | 22 | 1669 | 2939 | 1719 |
| c5315 | 178 | 123 | 2307 | 4386 | 2485 |
| c6288 | 32 | 32 | 2416 | 4800 | 2448 |
| c7552 | 207 | 108 | 3512 | 6144 | 3719 |

voltage. The standard deviations of transistor length, oxide thickness and threshold voltage were assigned to 15.7%, 5.3% and 4.4% of the nominal values respectively, according to [Nas01]. After running layout for each circuit, the area of the die was partitioned into rectangular grid [CS03] explained in Section 3.2. The number of gates in a grid cell was smaller than 100 to keep reasonable modeling accuracy of the spatial correlation as in [CS03]. The correlation between the random variables for the same type of parameter was set to 0.8 for two neighboring grid cells. This correlations decreases exponentially to 0.4 for grid cells with distance equal to 15 times the size of a grid cell. The correlation for parameters in grid cells which were further separated was set to 0.4, modeling global correlation. The correlations between different types of parameters were set to 0, i.e., independence, for simplicity.

The ISCAS85 benchmark circuits used in the experiment are first presented in [BF85] for testing. According to the results of the reverse engineering studying in [HYH99], these circuits are applications ranging from a 27-channel interrupt controller (c432) to a 32-bit adder/comparator (c7552). These circuits cover different design styles. Some have long critical paths (c6228); others have large span (c7552). For sequential circuits, ISCAS89 benchmark circuits are presented in [BBK89]. These circuits are collected from universities and industry over the world. Therefore most functions of these circuits are unknown. The number of registers in these benchmark circuits ranges from 3 to 1728; the number of combinational gates, e.g., AND, OR, NOT etc., from 10 to 22179. Because of the differences between these circuits, both ISCAS85 and ISCAS89 benchmarks are widely used in testing EDA algorithms. For example, they have been used in [CS03, KPR05, ZSL+05, ZCH+05, SS06]. An overview of ten ISCAS85 circuits is given in Table 6.1. Such information will be compared with the results from timing model extraction to verify the efficiency of the extracted timing models. The last two columns in Table 6.1 show the numbers of edges and nodes in

**Table 6.2:** ISCAS89 Benchmarks

| Circuit | num. of inputs | num. of outputs | num. of gates | num. of registers | num. of edges in TG | num. of nodes in TG | num. of edges in RTG |
|---|---|---|---|---|---|---|---|
| s298 | 4 | 6 | 119 | 14 | 224 | 137 | 86 |
| s526 | 4 | 6 | 193 | 21 | 445 | 218 | 167 |
| s820 | 19 | 19 | 289 | 5 | 757 | 313 | 185 |
| s1238 | 15 | 14 | 508 | 18 | 1041 | 541 | 219 |
| s1423 | 18 | 5 | 657 | 74 | 1164 | 749 | 2226 |
| s5378 | 36 | 49 | 2779 | 179 | 4212 | 2994 | 2126 |
| s9234 | 37 | 39 | 5597 | 211 | 7971 | 5845 | 3219 |
| s13207 | 63 | 152 | 7951 | 638 | 11165 | 8652 | 4584 |
| s15850 | 78 | 150 | 9772 | 534 | 13645 | 10384 | 16490 |
| s38584 | 39 | 304 | 19253 | 1426 | 32756 | 20718 | 20243 |

the corresponding timing graphs. Similar information for ISCAS89 benchmarks are listed in Table 6.2, with additional information about reduced timing graphs also listed, where TG is the abbreviation for timing graph and RTG for reduced timing graph. The number of gates is the count of the combinational gates in the original circuit. The number of the nodes in the reduced timing graph is the sum of the numbers of the registers, the primary inputs and the primary outputs in the circuit.

In order to verify the accuracy of the extracted timing models, the circuit performances were identified by Monte Carlo simulation with 10000 iterations. This simulation was performed at gate level, with gate delays as precharacterized mapping functions of process parameters. In each iteration of the Monte Carlo simulation, the random variables in gate delays were directly sampled; gate delays were computed from such samples with the mapping functions; the circuit performance of this iteration was computed using a static timing analysis engine. Algorithm 2 was run for combinational circuits to compute the maximum delays between primary inputs and outputs in verifying the accuracy of the timing models. With all primary inputs initialized, Algorithm 2 was also used to compute the maximum delay of the circuit in verifying the accuracy of the variable substitution method. For flip-flop based circuits Algorithm 3 was used to compute the minimum clock period in each iteration of the Monte Carlo simulation. For a similar purpose, Algorithm 5 was used for latch based circuits. Because Monte Carlo simulation and statistical timing analysis both used the same set of precharacterized gate delays, the error during the gate delay modeling step was not included. Therefore, the error in the comparison of this experiment was only from the statistical analysis engine and the proposed methods.

All methods mentioned in this experiment were implemented in C++ and tested on

a PC with a 2.33GHz CPU and 4G memory. The runtimes of the algorithms were measured with the *clock()* function in C++. Because the minimum time unit this function can measure is $10^{-6}$ second, shorter runtimes of some experiments, e.g., statistical timing analysis with extracted timing models for sequential circuits, were measured as 0. Such results in this section will be written as $<1\mu s$.

## 6.2 Results of Timing Models for Combinational Circuits

For combinational circuits, black-box timing models can be easily extracted with Algorithm 2, as discussed in Section 3.3. The size of black-box timing models, however, may be much larger than that of gray-box timing models. In order to show how efficient and accurate the black-box timing model is in statistical timing analysis, the number of edges and nodes in black-box models are listed in Table 6.3. Because the maximum delays between primary inputs and outputs are computed directly using a standard statistical timing engine, the accuracy of such delays compared to the results from Monte Carlo simulations will be used to verify the accuracy of the timing models extracted with the proposed noncritical edge removal method discussed in Section 4.1.

In Table 6.3 Columns II and III show the numbers of edges and nodes in the black-box timing models. As discussed in Section 3.3.1, all internal circuit structure is discarded in a black-box timing model and an edge is created if a path exists between a pair of input and output in the original circuit. Therefore, the number of nodes in a black-box timing model is equal to the sum of the numbers of primary inputs and outputs, as shown in Table 6.1. In the worst case, the number of edges inside a black-box model is equal to the product of numbers of inputs and outputs. This worst-case edge number is shown in column IV in Table 6.3. According to these results, the black-box timing model is much smaller in average than the worst case in the test circuits, and therefore can be used in practical design flows. However, the size of a black-box timing model is still large compared to the size of the corresponding gray-box timing model, the experimental results of which will be shown later in this section.

According to Section 3.3.1, a timing model for a combinational circuit should have the same delay matrix as of the original circuit. To verify the accuracy of black-box models, the edge delays were compared with the delay distributions from Monte Carlo simulation. In each iteration of the Monte Carlo simulation, all gate delays in the circuit were sampled. Thereafter Algorithm 2 was applied to compute the maximum delay from each input to all outputs. The comparison results of mean and standard deviation of input-output delays from timing model and from Monte Carlo simulation are listed in column V and VI. In order to compute the maximum

**Table 6.3:** Results of Black-Box Timing Models for Combinational Circuits

| Circuit | num. of edges | num. of nodes | worst num. of edges | max. mean error | max. std error |
|---|---|---|---|---|---|
| c432 | 225 | 43 | 252 | 0.46% | 0.72% |
| c499 | 1312 | 73 | 1312 | 0.35% | 0.76% |
| c880 | 419 | 86 | 1560 | 0.18% | 0.57% |
| c1355 | 1312 | 73 | 1312 | 1.02% | 1.64% |
| c1908 | 807 | 58 | 825 | 0.60% | 1.87% |
| c2670 | 1067 | 373 | 32620 | 0.81% | 1.09% |
| c3540 | 724 | 72 | 1100 | 1.50% | 0.89% |
| c5315 | 2978 | 301 | 21894 | 0.64% | 1.35% |
| c6288 | 784 | 64 | 1024 | 1.58% | 0.98% |
| c7552 | 3543 | 315 | 22356 | 1.21% | 1.83% |
| I | II | III | IV | V | VI |

mean error, the means of input-output delays from timing models were subtracted by the ones from Monte Carlo simulation. The absolute values of the results were divided by the means from Monte Carlo simulation. The maximum of such errors in all input-output delays are listed in column V of Table 6.3. The relative error of standard deviation (std) was computed similarly. Because the maximum delays in the timing models were computed directly by the statistical engine in [VRK+04] and no other approximation was applied in black-box timing models, the errors shown in Table 6.3 are totally from the statistical engine. For instance, the correlation due to path reconvergence is not considered and the maximum is approximated in [VRK+04]. In Table 6.3, both relative errors of mean and standard deviations are no more than 2%. This accuracy is acceptable based on the fact that the statistical engine in [VRK+04] is already used for industrial designs. If needed, the accuracy can also be improved by applying quadratic or independent component analysis based statistical engines.

In the gray-box timing models described in Section 4.1, the edges which have small probabilities on any critical paths corresponding to all input-output pairs are removed. The effectiveness of this method depends on the relative number of such edges in a circuit. To show the trend of the maximum criticalities, i.e., the distribution of the maximum criticalities of all the edges in each ISCAS85 benchmark circuit, Monte Carlo simulation was run for each original circuit. In each iteration of the simulation Algorithm 6 was applied. The criticality of an edge was computed by the number of iterations in which the edge was critical divided by the total number of the Monte Carlo iterations. The criticality is in the range from 0 to 1. This range is divided into 10 subranges and the percentages of edges with criticalities in these subranges are illustrated in Figure 6.1. According to this figure, a large percentage

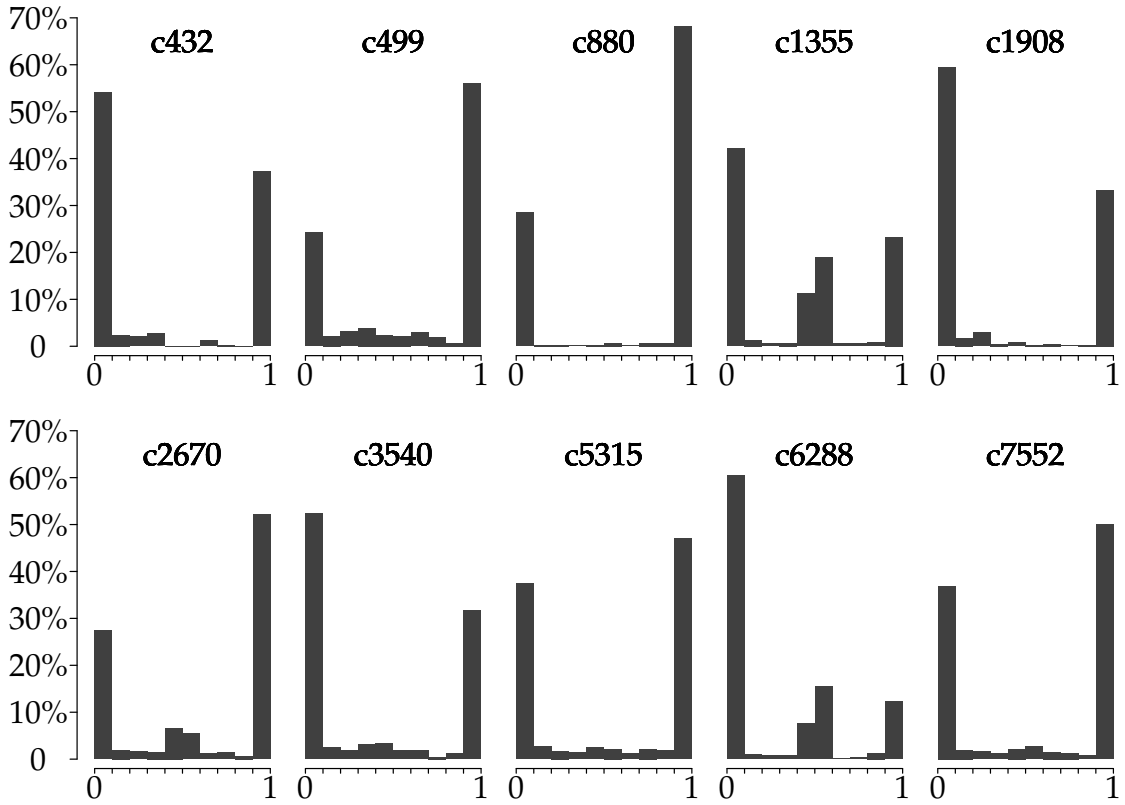


**Figure 6.1:** Criticality Distributions of ISCAS85 Benchmarks

of edges in the timing graph have criticalities in the subrange 0-0.1. This proves the concept of the noncritical edge removal in Section 4.1.

The edge criticalities from Monte Carlo simulation are accurate. This method, however, is much slower compared to the block-based statistical method proposed in Section 4.1.2, while [VRK+04] was used as the statistical timing engine. As already discussed, this engine may introduce computation error. In Table 6.4, the criticalities below $\delta_c$ from Monte Carlo simulation and from the method in Section 4.1.2 are compared. To keep enough accuracy, the threshold $\delta_c$ defined in Section 4.1.2 was set to 0.05. This threshold was so selected because most of the maximum criticalities in the 0-0.1 range are actually smaller than 0.05 according to the experimental results. Additionally, there is inaccuracy in the statistical computation of the maximum criticality. As the threshold was set to 0.05, even with such inaccuracy in the computed maximum criticalities, the accuracy of the timing models can still be preserved.

Because of the inaccuracy of statistical criticality computation, some edges with criticalities smaller than $\delta_c$ were not identified. The number of such edges is shown in column V of Table 6.4. These edges were not deleted from the timing graph. Therefore, the size of the timing model was increased unnecessarily. Column VI in Table 6.4 shows the number of edges which have criticalities larger than $\delta_c$, but

**Table 6.4:** Accuracy of Statistical Criticality Computation, $\delta_c = 0.05$

| Circuit | Monte Carlo simu. | | statistical criticality computation | | | | |
|---|---|---|---|---|---|---|---|
| | num. of removed edges | runtime (s) | num. of removed edges | num. of missing edges | num. of wrong edges | max. criticality error | runtime (s) |
| c432 | 178 | 86.55 | 172 | 6 | 0 | 0 | 0.02 |
| c499 | 98 | 183.99 | 73 | 26 | 1 | 0.0066 | 0.08 |
| c880 | 208 | 192.12 | 200 | 8 | 0 | 0 | 0.12 |
| c1355 | 444 | 504.77 | 443 | 2 | 1 | 0.0173 | 0.24 |
| c1908 | 875 | 668.17 | 848 | 27 | 0 | 0 | 0.24 |
| c2670 | 543 | 2323.14 | 529 | 14 | 0 | 0 | 6.05 |
| c3540 | 1480 | 1340.82 | 1323 | 159 | 2 | 0.0214 | 0.6 |
| c5315 | 1588 | 4256.43 | 1502 | 86 | 0 | 0 | 8.9 |
| c6288 | 2831 | 3657.48 | 2819 | 12 | 0 | 0 | 1.37 |
| c7552 | 2187 | 6142.62 | 2110 | 77 | 0 | 0 | 12.93 |
| I | II | III | IV | V | VI | VII | VIII |

were identified as noncritical by the statistical method. These edges were incorrectly removed from the timing model and consequently the accuracy of the timing model may be degraded. Comparing these two columns it can be found that the proposed statistical method missed some edges but identified only a few critical edges incorrectly.

The maximum deviation of the criticalities of the incorrectly recognized edges from $\delta_c$ is shown in column VII of Table 6.4. This error indicates that the maximum criticality of the wrongly deleted edges was 0.0714 ($\delta_c + 0.0214$). This shows that the wrongly deleted edges still had reasonable small criticalities. Therefore they did not affect the accuracy of the timing models significantly. In column II and IV, the numbers of edges identified as critical path by both methods are shown. Comparing these numbers with column V and VI, most of critical edges were recognized correctly by the proposed statistical method so that its effectiveness is proved. In column III and VIII, the runtimes of Monte Carlo simulation and the proposed method are shown. According to this comparison, the proposed method is at least two orders of magnitude faster than Monte Carlo simulation, and therefore is suitable for timing model extraction.

Finally, the results of timing model extraction for combinational circuits using the proposed method are listed in Table 6.5. In column II and IV, the numbers of edges and nodes in timing models are shown. These numbers are divided by the numbers of edges and nodes in the original timing graphs and the results are shown in column III and V. According to these compression ratios, the criticality based method

**Table 6.5:** Results of Gray-Box Timing Models for Combinational Circuits

| Circuit | num. of edges | edge ratio | num. of nodes | node ratio | max. mean error | max. std error | runtime (s) | ratio of compl. |
|---|---|---|---|---|---|---|---|---|
| c432 | 45 | 13.39% | 46 | 23.47% | 0.24% | 0.75% | 0.04 | 5.54 |
| c499 | 175 | 42.89% | 99 | 40.74% | 0.35% | 0.61% | 0.1 | 8.88 |
| c880 | 239 | 32.78% | 112 | 25.28% | 0.18% | 0.59% | 0.14 | 1.91 |
| c1355 | 143 | 13.44% | 99 | 16.87% | 0.36% | 1.65% | 0.26 | 11.37 |
| c1908 | 234 | 15.62% | 91 | 9.97 % | 0.60% | 1.87% | 0.27 | 3.88 |
| c2670 | 410 | 19.75% | 335 | 23.49% | 0.46% | 1.09% | 6.1 | 2.78 |
| c3540 | 448 | 15.24% | 143 | 8.32 % | 0.52% | 1.37% | 0.67 | 1.78 |
| c5315 | 960 | 21.89% | 421 | 16.94% | 0.58% | 1.35% | 9.02 | 3.48 |
| c6288 | 427 | 8.90 % | 187 | 7.64 % | 0.67% | 1.08% | 1.49 | 2.20 |
| c7552 | 1076 | 17.51% | 545 | 14.65% | 1.41% | 1.81% | 13.1 | 3.85 |
| I | II | III | IV | V | VI | VII | VIII | IX |

can effectively remove the edges which do not affect the maximum delays between primary inputs and outputs. Especially this method is most effective in circuits with long paths, e.g., c6288, where most of the paths are dominated by a small number of critical ones.

Similar to the comparison for black-box timing models, the accuracy of the extracted gray-box timing models was verified by comparing the maximum input-output delays with the ones from Monte Carlo simulation, shown in column VI and VII. From the comparison of these two columns to columns V and VI in Table 6.3, the extracted gray-box models also have reasonable accuracy. In column VIII of Table 6.5, the total runtimes for timing model extraction are shown. Comparing to column VIII in Table 6.4, most of the runtime of the algorithm was consumed by the criticality computation. The runtime of the basic merge operations in the last step of the timing model extraction was negligible.

In order to compare the black-box and gray-box timing models, the complexity of the timing model for a combinational circuit is first defined. Assume there are $m$ edges and $n$ nodes except the primary inputs in the timing model. Furthermore, assume a block-based statistical timing engine is used to verify the timing of the complete design when this timing model is applied. During such a timing analysis, the edge delays are added to arrival times. Therefore there are $m$ sum computations needed. At each node except primary inputs in the timing model, the maximum of all incident arrival times is computed. If at node $i$ there are $k_i$ fanin edges, the number of the maximum computations at this node is $k_i - 1$. The total number of maximum computations is then computed as $\sum_{i=1}^{n}(k_i - 1) = \sum_{i=1}^{n} k_i - n = m - n$, where $\sum_{i=1}^{n} k_i = m$ because each edge in the timing graph has only one sink node.

From this analysis, the complexity of a timing model is therefore defined as the number of the computations during a block-based arrival time propagation inside this model, i.e., the sum of the numbers of the sum and maximum computations, equal to $m + m - n = 2m - n$. The complexities of black-box and gray-box timing models for ISCAS85 benchmarks can be computed from the data in Table 6.1, 6.3 and 6.5. Their ratios are shown in column IX in Table 6.5. These ratios explain that using a black-box timing model the statistical timing engine needs to perform much more computations compared to using a gray-box timing model. This proves the efficiency of the proposed gray-box timing model extraction method.

## 6.3 Results of Timing Models for Sequential Circuits

Timing models for sequential circuits include the constraints for all inputs, the minimum clock period, and the delays to all outputs. For flip-flop based circuits, a constraint for each input and a statistical delay for each output is extracted; the minimum clock period for all flip-flop pairs is represented by only one random variable. Therefore, the size of a timing model for a flip-flop based circuit is always one larger than the sum of numbers of inputs and outputs. For latch based circuits, however, the numbers of constraints for an input and the delays to an output become larger than one because of latch transparency. The minimum clock period for the paths between latches is still represented by one random variable ($\max\{V_1, V_3\}$ in Section 4.3).

In Section 6.2, the accuracy of the timing models for combinational circuits was verified by comparing the maximum input-output delays with the results from Monte Carlo simulation. For sequential circuits, however, there is no such theoretical method to verify the accuracy of their timing models. Instead, the arrival times at the inputs of each sequential circuit were generated randomly to emulating a practical application context. For each circuit, the maximum mean $\mu_M$ and standard deviation $\sigma_M$ of the delays from all inputs to their fanout registers in the original netlist were computed. Thereafter, a random variable was generated for each input randomly, with mean in the range between 0 and $0.5\mu_M$ and standard deviation between $0.05\sigma_M$ and $0.15\sigma_M$. The correlations between these random variables were set between 0.4 and 0.8, for the purpose of experiment. The ranges of means and standard deviations of random generated arrival times were selected to guarantee that the performances of some circuits were dominated by the minimum clock period between registers, and others by the timing constraints for inputs.

The results of timing model extraction for flip-flop based circuits are listed in Table 6.6, where all registers in ISCAS89 benchmark circuits were assumed as flip-flops. The size of a timing model for a flip-flop based circuit was computed by

**Table 6.6:** Results of Timing Model Extraction for Flip-flop Based Circuits

| Circuit | size of model | relative error | | | runtime† | | |
|---|---|---|---|---|---|---|---|
| | | mean | std | 97% yield | Monte Carlo | SSTA with models | model extraction |
| s298 | 11 | 0.06% | 0.55% | 0.05% | 5.17 | <1$\mu s$ | <1$\mu s$ |
| s526 | 11 | 0.06% | 0.42% | 0.22% | 9.34 | <1$\mu s$ | 0.01 |
| s820 | 39 | 0.01% | 0.28% | 0.05% | 18.82 | <1$\mu s$ | 0.01 |
| s1238 | 30 | 0.14% | 0.99% | 0.35% | 25.37 | <1$\mu s$ | 0.02 |
| s1423 | 24 | 0.14% | 0.31% | 0.69% | 45.51 | <1$\mu s$ | 0.02 |
| s5378 | 86 | 0.32% | 0.27% | 0.06% | 246.41 | <1$\mu s$ | 0.13 |
| s9234 | 77 | 0.65% | 0.43% | 0.50% | 733.12 | <1$\mu s$ | 0.3 |
| s13207 | 216 | 0.36% | 0.91% | 0.22% | 981.93 | <1$\mu s$ | 0.48 |
| s15850 | 229 | 0.68% | 0.41% | 0.12% | 1258.53 | <1$\mu s$ | 0.93 |
| s38584 | 344 | 0.36% | 0.29% | 0.09% | 3403.98 | 0.01 | 1.19 |
| I | II | III | IV | V | VI | VII | VIII |

†unit is second if not specified

summing the numbers of inputs and outputs and plus 1, shown in column II. According to the comparison of this size to the number of combinational cells and the number of registers in the original circuits shown in Table 6.2, the extracted timing models have remarkable advantage in size.

To verify the accuracy of the extracted timing models, the results of timing analysis using the extracted timing models and the results of Monte Carlo simulation were compared. The relative errors of mean and standard deviation by comparing the circuit performances from timing analysis with timing models and Monte Carlo simulation are listed in column III and IV in Table 6.6. These errors are defined as the absolute difference between the two circuit performances divided by the one from Monte Carlo simulation. Additionally, the relative errors of the clock periods at which the designs can achieve 97% yield are shown in column V for accuracy comparison. These results prove that timing analysis with the extracted timing models has very high accuracy.

To verify the quality of the timing models for latch based circuits, all sequential cells in the ISCAS89 benchmark circuits were assumed as latches. In experiment, all benchmark circuits were assumed with one clock phase and the enabling clock edges were set to 0.5 times the clock period. The predefined threshold $\delta_l$ for the probability comparison with (4.37) and (4.45) was set to 99.9%, which is very close to 1 so that the removal of arrival time elements during time constraint extraction affects the accuracy of timing models only with a very small probability.

For comparison, Monte Carlo simulation was run with 10000 iterations for each

**Table 6.7:** Results of Timing Model Extraction for Latch Based Circuits

| Circuit | size of model | | relative error | | | runtime† | | |
|---|---|---|---|---|---|---|---|---|
| | avg. cons. num. per input | avg. delay num. per output | mean | std | 97% yield | Monte Carlo | SSTA with models | model extraction |
| s298 | 2 | 5 | 0.09% | 0.05% | 0.05% | 21.27 | <1$\mu s$ | 0.01 |
| s526 | 4 | 11 | 0.10% | 1.08% | 0.16% | 42.87 | <1$\mu s$ | 0.01 |
| s820 | 2 | 36.79 | 0.04% | 0.69% | 0.21% | 44.18 | <1$\mu s$ | 0.03 |
| s1238 | 1 | 13.93 | 0.14% | 0.99% | 0.35% | 95.28 | <1$\mu s$ | <1$\mu s$ |
| s1423 | 4.35 | 31.4 | 0.19% | 1.04% | 0.91% | 556.4 | <1$\mu s$ | 1.94 |
| s5378 | 2.2 | 55.73 | 0.89% | 1.31% | 0.58% | 2445.7 | <1$\mu s$ | 2.73 |
| s9234 | 7.36 | 81.28 | 0.17% | 0.58% | 1.59% | 3578.1 | <1$\mu s$ | 18.32 |
| s13207 | 1.27 | 3.7 | 0.30% | 0.47% | 0.44% | 5031.7 | 0.01 | 8.58 |
| s15850 | 2.71 | 31.61 | 0.48% | 0.20% | 0.18% | 21439.4 | <1$\mu s$ | 174.08 |
| s38584 | 2.39 | 6.82 | 0.62% | 0.24% | 0.68% | 54610.5 | <1$\mu s$ | 307.32 |
| I | II | III | IV | V | VI | VII | VIII | IX |

†unit is second if not specified

benchmark circuit. In each iteration, Algorithm 5 was used to compute the minimum clock period of the original circuit from sampled gate delays. The results are shown in Table 6.7, where the errors are relative to the results of Monte Carlo simulation. The average number of the constraints for each input is shown in column II. The average number of the delay elements in the arrival time sets to an output is shown in column III. Because of latch transparency, arrival times from many internal latches and from inputs can reach outputs. Therefore, the numbers of delay elements are larger than the numbers of the constraints for inputs. The relative errors in column IV to VI prove that timing analysis with proposed timing models still maintains high accuracy.

The runtimes of Monte Carlo simulation and timing analysis using the extracted timing models are shown in Table 6.6 and Table 6.7. Because most of the circuit components are between registers in ISCAS89 benchmarks, the runtimes of the proposed timing model extraction method for flip-flop based circuits are roughly equal to the runtimes of the statistical timing analysis of the original circuits. The comparison between column VII and VIII of Table 6.6 shows that timing analysis using the proposed timing models gains many orders of magnitude in runtime acceleration for flip-flop based circuits. For latch based circuits, the speedup of the existing statistical timing analysis methods compared to Monte Carlo simulation is about three orders of magnitude [CZ06, ZTC+06]. According to the comparison of column VII and VIII in Table 6.7, using the proposed timing models can gain much larger acceleration to the Monte Carlo simulation than using the original circuits.

## 6.4 Results of Hierarchical Statistical Timing Analysis

To test the correlation handling method proposed in Section 5.1, four experimental hierarchical circuits were built with ISCAS85 benchmark circuits as modules. In each design four modules of the same circuit were placed with the layout illustrated in Figure 6.2.

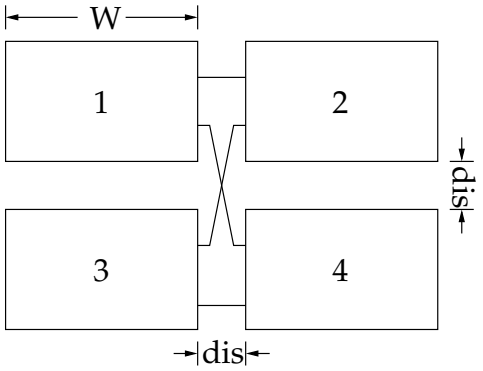


**Figure 6.2:** Layout of the Hierarchical Circuit

The outputs of the two modules in the first column were cross connected with the inputs of the other two modules in the second column. The distance between modules was changed in different tests, so that the effect of local correlation can be investigated. With circuit size and style as standard, the selected circuits as modules were c432, c3540, c6288 and c7552. The distances between modules were set to 0, one time the module width, and two times the module width. For each circuit and module distance, statistical timing analysis using timing models with random variable substitution (Case A) and without random variable substitution (Case B) were run to show the usefulness of correlation handling. In Case B, only the correlation from die-to-die variation was considered, because these variations were shared by all modules without the need of special processing. To verify the accuracy of the proposed method, Monte Carlo simulation was run with the original circuits (Case C). The results are listed in Table 6.8, where all runtimes without unit specified are in seconds.

In Table 6.8, the errors of mean and standard deviation are given as relative ratios. They are determined by comparing the maximum circuit delays computed from hierarchical statistical timing analysis to Monte Carlo simulation. Compare Case A and Case B in the different distance configurations. The accuracy with correlation handling is better than the accuracy without correlation handling in most cases. In other cases, e.g., dis=2W with c3540, c6288 and c7552, both tests have comparable accuracy. The inaccuracy in Case A is actually caused by the approximation during timing model extraction and statistical timing analysis, and is already acceptable for industrial application.

**Table 6.8:** Results of Hierarchical Statistical Timing Analysis

| | | | c432 | c3540 | c6288 | c7552 | |
|---|---|---|---|---|---|---|---|
| dis=0 | case A | mean error | 1.59% | 1.04% | 1.49% | 0.65% | I |
| | | std error | 1.27% | 0.34% | 1.41% | 1.43% | II |
| | | runtime | 0.01 | 0.06 | 0.14 | 0.75 | III |
| | case B | mean error | 5.35% | 2.69% | 4.82% | 0.17% | IV |
| | | std error | 19.76% | 11.68% | 13.74% | 8.04% | V |
| | | runtime | <1$\mu s$ | 0.01 | 0.01 | 0.02 | VI |
| | case C | runtime | 12.09 | 302.54 | 478.75 | 696.21 | VII |
| dis=1W | case A | mean error | 1.71% | 1.23% | 2.33% | 0.23% | VIII |
| | | std error | 1.21% | 1.05% | 2.80% | 2.20% | IX |
| | | runtime | <1$\mu s$ | 0.06 | 0.15 | 0.78 | X |
| | case B | mean error | 4.51% | 1.81% | 3.17% | 0.13% | XI |
| | | std error | 15.66% | 3.91% | 5.30% | 3.06% | XII |
| | | runtime | <1$\mu s$ | 0.01 | 0.01 | 0.02 | XIII |
| | case C | runtime | 12.58 | 314.95 | 496.36 | 747.85 | XIV |
| dis=2W | case A | mean error | 1.63% | 1.48% | 2.64% | 0.01% | XV |
| | | std error | 0.74% | 1.01% | 2.29% | 1.21% | XVI |
| | | runtime | <1$\mu s$ | 0.06 | 0.13 | 0.61 | XVII |
| | case B | mean error | 3.77% | 1.48% | 2.61% | 0.01% | XVIII |
| | | std error | 11.35% | 0.80% | 2.30% | 1.21% | XIX |
| | | runtime | <1$\mu s$ | 0.01 | 0.01 | 0.03 | XX |
| | case C | runtime | 13.13 | 322.05 | 513.72 | 748.24 | XXI |

Case A: Hierarchical SSTA with random variable substitution
Case B: Hierarchical SSTA without random variable substitution
Case C: Monte Carlo simulation

Another conclusion can be drawn by comparing case B in different distance configurations, e.g., rows V, XII, and XIX. The accuracy of these test cases increases as the distance between modules becomes larger. This is because the correlation between modules decreases with larger distance. Therefore, ignoring correlation from within-die variation affects the accuracy less. Compare the accuracy of Case B for different circuits. The accuracy of c7552 is obviously better than the other test cases. The first reason is that the width of c7552 is larger than the other modules, so that the correlation between modules is smaller when distance is set to 1W and 2W. In the test with distance 0, the accuracy of c7552 is also better than the other circuits.

This is because the area of c7552 is larger enough so that a part of circuit components inside the four instances, e.g., circuit components at the upper side of module 1 and the ones at the lower side of module 3, do not have correlation from within-die variation. Therefore, discarding such correlation has less impact on the accuracy of c7552 than on the accuracy of the other circuits with smaller area.

In Table 6.8 the runtimes of Case A are always larger than the ones of Case B because of the random variables substitution. Both runtimes, however, are less than one second and make no difference in a real design flow. Additionally, the proposed hierarchical analysis method using the extracted timing models in this experiment is faster by several orders of magnitude than Monte Carlo simulation with flat netlists. This proves the effectiveness of the proposed hierarchical statistical analysis in this thesis.

## 6.5 Summary

With the experimental results in this chapter, the efficiency and accuracy of the proposed timing model extraction methods are proved. The maximum criticalities in all ISCAS85 benchmark circuits exhibit the tendency to approximate 0 and 1. By removing edges with maximum criticalities smaller than the predefined threshold $\delta_c = 0.05$ and applying the basic merge operations, the sizes of extracted timing models are only about one fifth of the original circuits. By comparing the delays between primary inputs and outputs, the extracted timing models show about 1% inaccuracy, which is acceptable in statistical timing analysis because of the approximation during the maximum computation. For flip-flop based circuits, the extracted timing models have remarkable size compression ratios. Because of latch transparency, there is more than one constraint element for a primary input of a latch based circuit and more than one delay element to a primary output, as described in Section 4.3. Consequently the size of the extracted timing model is relative large but still has remarkable advantage compared to the size of the original circuit. The accuracy of the extracted timing models for sequential circuits was confirmed by applying them into a random generated test design individually.

The variable substitution algorithm handling correlation between modules in the complete hierarchical design was verified with hierarchical designs created from ISCAS85 benchmark circuits. The results show that the proposed method maintains good accuracy compared to Monte Carlo simulation and the accuracy degradation caused by discarding the correlation between modules is large, so that the usefulness of applying the proposed variable substitution is confirmed.

# Chapter 7

# Conclusion

With continuing feature size scaling semiconductor devices face more relative process variations than in past days. These variations cause the timing evaluation of a design more complex because a drastically increasing number of corners should be checked. To reduce the runtime and improve the accuracy of such evaluation, statistical timing analysis is introduced. In a parametrized statistical timing flow, gate delays are modeled as functions of process parameters; the performance of a circuit is computed by a propagation algorithm. The main challenge in such a propagation is to handle the correlation from reconvergent paths and the proximity effect from manufacturing.

As a method to conquer the complexities in design and verification, hierarchical flow is widely adopted. Moreover, this flow enables more independent cooperation between different design units. For example, more and more modules in SoC designs are provided as IP blocks by third-party vendors nowadays. The aggravation of process variations and the emergence of statistical timing analysis, however, demand a new renovation in the hierarchical design flow.

Traditionally, the hierarchical design flow contains two steps. At first, modules are designed and verified independently, with interface specifications defined by system engineers. Thereafter, these modules are integrated together to form a complete system. After this stage, the functional and timing verifications are performed for the complete design, without inspecting the internal details in each module. Base on this observation, timing models containing only interfacing constraints and delays are extracted for modules during the first step. Because these models are relative small and in a simple form, the timing verification of the complete design can be accelerated drastically. When process variations are considered, all gate delays inside a module become random variables. This new characteristic makes most existing algorithms proposed for timing model extraction without considering process variations infeasible. The second challenge is the correlation between modules. During

the timing model extraction step, the correlation between modules are unknown. Such information must be incorporated into timing verification of the complete design later.

The focus of this thesis is to solve the two problems above. For the three common circuit types timing model extraction methods were proposed. For combinational circuits, it was observed that many delays in such a circuit do not affect any critical paths between inputs and outputs. Therefore, the removal of these delays does not affect the interfacing timing characteristics. When process variations are considered, however, each path can affect the input-output delays of the module with a certain probability. To evaluate the importance of edges in timing analysis, the concept of maximum criticality was defined. Any edge with maximum criticality smaller than a predefined small threshold was removed. According to the experiment results, all ISCAS85 circuits exhibit the tendency of having a large portion of delays with criticalities approximating 0. This proves the effectiveness of the proposed noncritical edge removal algorithm.

For flip-flop based circuits, this thesis extended the extracted timing models in [ALS+02] to incorporate variations. A parametrized statistical timing analysis engine was used to extract all constraints and delays so that the resulting timing models can be seamlessly integrated into the statistical timing analysis flow. For latch based circuits, the classical clock model proposed in [SMO90b] was restructured to expose the timing requirements at the inputs of a module. Without assuming transparency level, timing constraints through different latch stages were extracted. The extracted models can be used to evaluate the yield of a circuit against any clock period and latch transparency. The accuracy of extracted timing models was verified by applying them into a random generated application context. Compared to Monte Carlo simulation, timing analysis with proposed models had several orders of magnitude of speedup. The accuracy, however, was still well maintained.

Because the correlation between modules can not be integrated into timing models, a random variable substitution method was proposed in this thesis to reestablished such correlation. The relation from the decomposed random variable sets inside timing models to the random variable set for the complete hierarchical design was established by a linear transformation. The correlation was therefore represented by sharing the same set of random variables in different modules. With a heterogeneous grid, the proposed method can handle the correlation between modules and circuit components in the top design at the same time. Experimental results proved that the correlation between modules had a significant effect on the accuracy of timing analysis. With the proposed random variables substitution method, the accuracy of hierarchical timing analysis can be well preserved in reasonable runtime.

With the proposed methods, a complete and effective hierarchical statistical timing analysis flow is established. This design flow not only maintains efficiency

and accuracy when process variations are considered, but also enables the modern system-on-chip designs in the new era of deep submicron realm.

# Bibliography


[ABZ+02] Aseem Agarwal, David Blaauw, Vladimir Zolotov, Savithri Sundareswaran, Min Zhao, Kaushik Gala, and Rajendran Panda. Path-based statistical timing analysis considering inter- and intra-die correlations. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, pages 16–21, 2002.

[ABZ03a] Aseem Agarwal, David Blaauw, and Vladimir Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 900–907, 2003.

[ABZ+03b] Aseem Agarwal, David Blaauw, Vladimir Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Statistical delay computation considering spatial correlation. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 271–276, 2003.

[ABZV03a] Aseem Agarwal, David Blaauw, Vladimir Zolotov, and Sarma Vrudhula. Computation and refinement of statistical bounds on circuit delay. In *ACM/IEEE Design Automation Conference (DAC)*, pages 348–353, 2003.

[ABZV03b] Aseem Agarwal, David Blaauw, Vladimir Zolotov, and Sarma Vrudhula. Statistical timing analysis using bounds. In *Design, Automation and Test in Europe (DATE)*, pages 62–67, 2003.

[ADM92] Pranav Ashar, Sujit Dey, and Sharad Malik. Exploiting multi-cycle false paths in the performance optimization of sequential circuits. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 510–517, 1992.

[ALS+02] Daga A.J., Mize L., Sripada S., Wolff C., and Qiuyang Wu. Automated timing model generation. In *ACM/IEEE Design Automation Conference (DAC)*, pages 146–151, 2002.

[AZB03] Aseem Agarwal, Vladimir Zolotov, and David T. Blaauw. Statistical timing analysis using bounds and selective enumeration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1243–1260, 2003.

[BBC+08] Duane S. Boning, Karthik Balakrishnan, Hong Cai, Nigel Drego, Ali Farahanchi, Karen M. Gettings, Lim Daihyun, Ajay Somani, Hayden Taylor, Daniel Truque, and Xie Xiaolin. Variation. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):63–71, 2008.

[BBK89] Franc Brglez, David Bryan, and Krzystof Kozminski. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1929–1934, 1989.

[BCSS08] David Blaauw, Kaviraj Chopra, Ashish Srivastava, and Louis Scheffer. Statistical timing analysis: from basic principles to state of the art. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(4):589–607, 2008.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 695–698, 1985.

[BN99] Duane S. Boning and Sani Nassif. Models of process variations in device and interconnect. In *Design of High Performance Microprocessor Circuits, chapter 6*. IEEE Press, 1999.

[BS99] Mark Birnbaum and Howard Sachs. How VSIA answers the SoC dilemma. *IEEE Computer*, 32(6):42–50, 1999.

[Cad04] Cadence. *Clock Domain Crossing (White Paper)*, Dec 2004.

[CCBC06] Brian Cline, Kaviraj Chopra, David Blaauw, and Yu Cao. Analysis and modeling of cd variation for statistical static timing. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 60 – 66, 2006.

[CK08] Yen-Kuang Chen and S. Y. Kung. Trend and challenge on system-on-a-chip designs. *Journal of Signal Processing Systems*, 53(1-2):217–229, 2008.

[Cla61] Charles E. Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, 1961.

[Cla06] Theo A. C. M. Claasen. An industry perspective on current and future state of the art in system-on-chip (SoC) technology. *Proceedings of the IEEE*, 94(6):1121–1137, 2006.

[CS03] Hongliang Chang and Sachin S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single PERT-like traversal. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 621–625, 2003.

[CSH95] Weitong Chuang, Sachin S. Sapatnekar, and Ibrahim N. Hajj. Timing and area optimization for standard-cell VLSI circuit design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(3):308–320, 1995.

[CZ04] Ruiming Chen and Hai Zhou. Clock schedule verification under process variations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 619-625, 2004.

[CZ06] Ruiming Chen and Hai Zhou. Statistical timing verification for transparently latched circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(9):1847–1855, 2006.

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

[CZV+08] R. Chen, L. Zhang, C. Visweswariah, J. Xiong, and V. Zolotov. Static timing: back to our roots. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 310–315, 2008.

[Dar91] Richard B. Darst. *Introduction to linear programming: applications and extensions*. Marcel Dekker Inc., 1991.

[DK03] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 607–614, 2003.

[Elm48] W. C. Elmore. The transient response of damped linear networks with particular regard to wide-band amplifiers. *Journal of Applied Physics*, 19:55–63, 1948.

[FF91] Dorian Feldman and Martin Fox. *Probability, The Mathematics of Uncertainty*. Marcel Dekker, Inc, 1991.

[Fis90] John P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39(7):945–951, 1990.

[FLZ07] Zhuo Feng, Peng Li, and Yaping Zhan. Fast second-order statistical static timing analysis using parameter dimension reduction. In *ACM/IEEE Design Automation Conference (DAC)*, pages 244–249, 2007.

[GVTG08] A. Goel, S. Vrudhula, F. Taraporevala, and P. Ghanta. A methodology for characterization of large macro cells and ip blocks considering process variations. In *International Symposium on Quality Electronic Design (ISQED)*, pages 200–206, 2008.

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

[HB06] Aaron P. Hurst and Robert K. Brayton. The advantages of latch-based design under process variation. In *International Workshop on Logic & Synthesis*, pages 241–246, 2006.

[Hen03] Jörg Henkel. Closing the SoC design gap. *IEEE Computer*, 36(9):119–121, 2003.

[Hit82] Robert B. Hitchcock. Timing verification and the timing analysis program. In *ACM/IEEE Design Automation Conference (DAC)*, pages 594–604, 1982.

[HKO01] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*. Wiley & Sons, 2001.

[HSC82] Robert B. Hitchcock, Gordon L. Smith, and David D. Cheng. Timing analysis of computer hardware. *IBM Journal Research Development*, 26(1):100–105, 1982.

[HYH99] Mark C. Hansen, Hakan Yalcin, and John P. Hayes. Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering. *IEEE Design & Test of Computers*, 16(3):72–80, 1999.

[IEE05] IEEE. *IEEE Standard SystemC Language Reference Manual*, 2005.

[IEE07] IEEE. *Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language*, 2007.

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

[KCJ+00] Ken Kundert, Henry Chang, Dan Jefferies, Gilles Lamant, Enrico Malavasi, and Fred Sendig. Design of mixed-signal systems-on-a-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1561–1571, dec 2000.

[KM97] Noriya Kobayashi and Sharad Malik. Delay abstraction in combinational logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(10):1205–1212, 1997.

[Knu98] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 1998.

[KPR05] Kunhyuk Kang, Bipul C. Paul, and Kaushik Roy. Statistical timing analysis using levelized covariance propagation. In *Design, Automation and Test in Europe (DATE)*, pages 764–769, 2005.

[KS05] V. Khandelwal and A. Srivastava. A general framework for accurate statistical timing analysis considering correlations. In *ACM/IEEE Design Automation Conference (DAC)*, pages 89–94, 2005.

[KYC+81] Ryotaro Kamikawai, Minoru Yamada, Tsuneyo Chiba, Kenichi Furumaya, and Yoji Tsuchiya. A critical path delay check system. In *ACM/IEEE Design Automation Conference (DAC)*, pages 118–123, 1981.

[LCS09a] Bing Li, Ning Chen, and Ulf Schlichtmann. Timing model extraction for sequential circuits considering process variations. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 336–343, 2009.

[LCS+09b] Bing Li, Ning Chen, Manuel Schmidt, Walter Schneider, and Ulf Schlichtmann. On hierarchical statistical static timing analysis. In *Design, Automation and Test in Europe (DATE)*, pages 1320–1325, 2009.

[Lee05] Ki Won Lee. SoC R&D trend for future digital life. *IEICE Transactions on Electronics*, E88-C(8):1705–1710, 2005.

[LKS+08] Bing Li, Christoph Knoth, Walter Schneider, Manuel Schmidt, and Ulf Schlichtmann. Static timing model extraction for combinational circuits. In *International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pages 156–166, 2008.

[LLCP08] Xin Li, Jiayong Le, Mustafa Celik, and Lawrence T. Pileggi. Defining statistical timing sensitivity for logic circuits with large-scale process and environmental variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(6):1041–1054, 2008.

[LLGP04] X. Li, J. Le, P. Gopalakrishnan, and L. T. Pileggi. Asymptotic probability extraction for non-normal distributions of circuit performance. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2004.

[MKB02] Cho W. Moon, Harish Kriplani, and Krishna P. Belkhale. Timing model extraction of hierarchical blocks by graph reduction. In *ACM/IEEE Design Automation Conference (DAC)*, pages 152–157, 2002.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), 1965.

[Moo03] Gordon E. Moore. No exponential is forever ... but we can delay 'forever'. In *presentation at ISSCC*, 2003.

[MP03] Philippe Magarshack and Pierre G. Paulin. System-on-chip beyond the nanometer wall. In *ACM/IEEE Design Automation Conference (DAC)*, 419-424, 2003.

[MQSB09] H.D. Mogal, Haifeng Qian, S.S. Sapatnekar, and K. Bazargan. Fast and accurate statistical criticality computation under process variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(3):350–363, March 2009.

[MR06] Douglas C. Montgomery and George C. Runger. *Applied Statistics and Probability for Engineers*. Wiley, 2006.

[MS99] Naresh Maheshwari and Sachin S. Sapatnekar. *Timing Analysis and Optimization of Sequential Circuits*. Kluwer Academic Publishers, 1999.

[Nas01] Sani R. Nassif. Modeling and analysis of manufacturing variations. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 223–228, 2001.

[OB04] Michael Orshansky and Arnab Bandyopadhyay. Fast statistical timing analysis handling arbitrary delay correlations. In *ACM/IEEE Design Automation Conference (DAC)*, pages 337–342, 2004.

[PS73] David J. Pilling and Henry B. Sun. Computer-aided prediction of delays in lsi logic systems. In *ACM/IEEE Design Automation Conference (DAC)*, pages 182–186, 1973.

[PTB+99] T. Park, T. Tugbawa, D. Boning, J. Chung, S. Hymes, R. Muralidhar, B. Wilks, K. Smekalin, and G. Bersuker. Electrical characterization of copper chemical mechanical polishing. In *International Conference on Chemical-Mechanical Polish Planarization for ULSI Multilevel Interconnection (CMP-MIC)*, pages 184–191, 1999.

[RPH83] J. Rubinstein, P. Penfield, and M. A. Horowitz. Signal delay in RC tree networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(3):202–211, 1983.

[Sap96] Sachin S. Sapatnekar. Efficient calculation of all-pairs input-to-output delays in synchronous sequential circuits. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 520–523, 1996.

[SBC97] Brian E. Stine, Duane S. Boning, and James E. Chung. Analysis and decomposition of spatial variation in integrated circuit processes and devices. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):24–41, February 1997.

[SBSV93] N.V. Shenoy, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constraints. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 156–161, 1993.

[Sch02] Lou Scheffer. Explicit computation of performance as a function of process variations. In *ACM/IEEE International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, pages 1–8, 2002.

[Seb77] G. Seber. *Linear Regression Analysis*. John Wiley & Sons, 1977.

[SMO90a] K.A. Sakallah, T.N. Mudge, and O.A. Olukotun. check$T_c$ and min$T_c$: Timing verification and optimal clocking of synchronous digital circuits. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 552–555, 1990.

[SMO90b] Karem A. Sakallah, Trevor N. Mudge, and Oyekunle A. Olukotun. Analysis and design of latch-controlled synchronous digital circuits. In *ACM/IEEE Design Automation Conference (DAC)*, pages 111–117, 1990.

[SS06] Jaskirat Singh and Sachin Sapatnekar. Statistical timing analysis with correlated non-Gaussian parameters using independent component analysis. In *ACM/IEEE Design Automation Conference (DAC)*, pages 155–160, 2006.

[SS08] Jaskirat Singh and Sachin S. Sapatnekar. A scalable statistical static timing analyzer incorporating correlated non-Gaussian and gaussian parameter variations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):160–173, 2008.

[SYA+81] Tohru Sasaki, Akihiko Yamada, Toshinori Aoyama, Katsutoshi Hasegawa, Shunichi Kato, and Shinichi Sato. Hierarchical design verification for large digital systems. In *ACM/IEEE Design Automation Conference (DAC)*, pages 105–112, 1981.

[Syn09] Synopsys. *PrimeTime Fundamentals User Guide*, 2009.

[VPMS97] S.V. Venkatesh, Robert Palermo, Mohammad Mortazavi, and Karem A. Sakallah. Timing abstraction of intellectual property blocks. In *IEEE Custom Integrated Circuits Conference (CICC)*, pages 99–102, 1997.

[VRK+04] C. Visweswariah, K. Ravindran, K. Kalafala, S.G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. In *ACM/IEEE Design Automation Conference (DAC)*, pages 331–336, 2004.

[XZH07] Jinjun Xiong, Vladimir Zolotov, and Lei He. Robust extraction of spatial correlation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(4):619–631, April 2007.

[XZV08] Jinjun Xiong, Vladimir Zolotov, and Chandu Visweswariah. Incremental criticality and yield gradients. In *Design, Automation and Test in Europe (DATE)*, pages 1130–1135, 2008.

[XZVV06] Jinjun Xiong, V. Zolotov, N. Venkateswaran, and C. Visweswariah. Criticality computation in parameterized statistical timing. In *ACM/IEEE Design Automation Conference (DAC)*, pages 63–68, 2006.

[YC06] Kai Yang and Kwang-Ting Cheng. Efficient identification of multi-cycle false path. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 360–365, 2006.

[YCS05] Jie Yang, Luigi Capodieci, and Dennis Sylvester. Advanced timing analysis based on post-OPC extraction of critical dimensions. In *ACM/IEEE Design Automation Conference (DAC)*, pages 359–364, 2005.

[ZCH+05] L. Zhang, W. Chen, Y. Hu, J. A. Gubner, and C. C.-P. Chen. Correlation-preserved non-Gaussian statistical timing analysis with quadratic timing model. In *ACM/IEEE Design Automation Conference (DAC)*, pages 83–88, 2005.

[ZHC05] Lizheng Zhang, Yuhen Hu, and C.C.-P. Chen. Block based statistical timing analysis with extended canonical timing model. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 250–253, January 2005.

[ZSL+05] Y. Zhan, A. J. Strojwas, X. Li, L. T. Pileggi, D. Newmark, and M. Sharma. Correlation-aware statistical timing analysis with non-Gaussian delay distributions. In *ACM/IEEE Design Automation Conference (DAC)*, pages 77–82, 2005.

[ZTC+06] Lizheng Zhang, Jengliang Tsai, Weijen Chen, Yuhen Hu, and Charlie Chung-Ping Chen. Convergence-provable statistical timing analysis with level-sensitive latches and feedback loops. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 941–946, 2006.

[ZZH+06] Shuo Zhou, Yi Zhu, Yuanfang Hu, Ronald Graham, Mike Hutton, and Chung-Kuan Cheng. Timing model reduction for hierarchical timing analysis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 415–422, 2006.

# Abstract in German

Mit fortschreitender Verkleinerung der Fertigungsgrößen von integrierten Schaltungen nehmen die relativen Schwankungen der Prozessparameter zu. Dies führt bei der konventionellen Statischen Timing Analyse zu einer erheblichen Überabschätzung der zu erwartenden Signallaufzeiten, da nur Extremwerte der Parameter berücksichtigt werden. Im Gegensatz dazu werden bei der Statistischen Timing Analyse Prozessparameter nicht auf ihre Extremwerte reduziert, sondern als Zufallsgrößen inklusive ihrer Korrelationen behandelt. In dieser Arbeit wurde die Anwendung der Statistischen Timing Analyse im Rahmen des hierarchischen Entwurfs digitaler Schaltungen erforscht. Dazu wurde eine Methode zur Generierung statistischer Timing Modelle für kombinatorische und sequentielle Schaltungen vorgestellt, die auch die Korrelationen der hierarchisch geschachtelten Module berücksichtigt.