The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

DEEP SUBMICRON (DSM) DESIGN AUTOMATION TECHNIQUES TO MITIGATE PROCESS VARIATIONS

A Thesis in

Computer Science and Engineering

by

Feng Wang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2008

The thesis of Feng Wang was reviewed and approved* by the following:

Yuan Xie
Assistant Professor of Computer Science and Engineering
Thesis Adviser
Chair of Committee

Mary Jane Irwin
Professor of Computer Science and Engineering
Evan Pugh Professor and A. Robert Noll Chair in Engineering

Vijaykrishnan Narayanan
Professor of Computer Science and Engineering

Zhiwen Liu
Assistant Professor of Electrical Engineering

Mahmut Kandemir
Associate Professor of Computer Science and Engineering
Chair of Graduate Program

*Signatures are on file in the Graduate School

Abstract

Technology scaling provides an integration capacity of billions of transistors and continuously enhances system performance. However, fabricating transistors at feature sizes in the deep sub-micron regime is increasingly challenging and leads to significant variations in such critical transistor parameters as transistor channel length, gate-oxide thickness, and threshold voltage. This manufacturing variability consequently causes substantial performance and power deviations from nominal values in identical hardware designs. As technology scales down relentlessly, the impact of these variations becomes even more pronounced. The established practice of designing for the worst case scenario is no longer a viable solution, since it yields overly pessimistic design estimates that unnecessarily increase design and manufacturing costs.

This realization has led to a marked shift from deterministic to statistical design methodologies across all levels of the design hierarchy [14]. In this thesis, statistical design techniques ranging from gate level to system level have been proposed to mitigate the impact of the process variations. At the gate level, an efficient method is proposed to compute the criticality for paths and arcs/nodes simultaneously by a single breadth-first graph traversal with linear complexity in circuit size. At the module level, variation aware resource sharing and assignment techniques in high level synthesis is proposed. In addition to the design-time techniques, a module selection algorithm with joint post-silicon tuning and design-time optimization is proposed to further reduce the parametric yield loss. At the system level, the classical task scheduling and allocation algorithm is augmented to be variation aware. Analysis results indicate these proposed techniques are very powerful in tackling the process variability issue.

Table of Contents

List of Tables viii

List of Figures ix

Acknowledgments xiii

Chapter 1. Introduction 1

1.1 The Trend in Process Variations 1

1.2 The Failure of Current Solutions for Process Variations 3

1.3 Contribution and Organization 6

Chapter 2. Overview of the Prior Art 9

2.1 Gate-level Statistical Analysis and Optimization 9

2.2 Architectural and System Level Design Approaches 13

2.3 Variation Aware High level synthesis 15

Chapter 3. A Criticality Computation Method in Statistical Timing Analysis 17

3.1 Introduction and Motivation 18

3.2 Background 19

3.2.1 Canonical Model 19

3.2.2 Timing Graph 20

3.2.3 Atomic Operations 20

3.2.4 Concept of the Criticality 23

3.2.5 Criticality of Arc/Node 24
3.3 Criticality Computation Method 25
3.3.1 Critical Region Computation for Arc/Node 25
3.3.2 Critical Region Computation for Path 27
3.3.3 Computation Algorithm 31
3.4 Analysis Results 33
3.5 Summary 35

Chapter 4. Variation-aware High Level Synthesis 36
4.1 Introduction and Motivation 37
4.2 Related Work in Variation-aware High Level Synthesis 39
4.3 Statistical Analysis For DFG 40
4.3.1 Function Unit Delay and Power Modeling 41
4.3.2 Statistical Timing Analysis in HLS 42
4.3.3 Statistical Power Analysis in HLS 43
4.3.4 Statistical Performance Yield Analysis for DFG 44
4.3.5 Performance Yield Gradient Computation 45
4.3.6 Power Yield Gradient Computation 47
4.4 Process Variation Aware Resource Sharing and Binding 48
4.4.1 Preliminaries and Problem Formulation 48
4.4.2 Statistical Performance Improvement Computation for Resource Reallocation and Reassignment 53
4.4.3 Gain Function for Resource Reallocation and Reassignment 54

4.4.4 Variation-aware Resource Sharing and Binding Algorithm 55
4.4.5 Analysis Results 57
4.5 Module Selection with Design-Time Optimization and Post-Silicon Tuning 60
4.5.1 Preliminaries and Problem Formulation 61
4.5.2 Function Unit Delay and Power Modeling with Adaptive Body Biasing 65
4.5.3 Design Time Variation-aware Module Selection Algorithm 66
4.5.4 Post-silicon Tuning with ABB 67
4.5.5 Joint Optimization Algorithm 72
4.5.6 Analysis Results 72
4.6 Summary 76

Chapter 5. Variation-aware Task Allocation and Scheduling for MPSoC 83
5.1 Introduction and Motivation 84
5.2 Related work in variation aware task allocation and scheduling 88
5.2.1 Task allocation and scheduling for embedded systems 88
5.2.2 Complementing Existing Probabilistic Real-Time Embedded System Research 89
5.3 Preliminaries 90
5.3.1 Platform Specification and Modeling 90
5.3.2 Problem Formulation 91
5.4 Statistical Task Graph Timing Analysis 95

5.5 Process-Variation-Aware Task and Communication Mapping for NoC Architectures 96
5.5.1 Process-Variation-Aware Dynamic Priority 97
5.5.2 Yield Computation for Partially Scheduled Task Graphs 98
5.5.3 Mapping Algorithm 103
5.5.4 Fast Path Allocation Algorithm 105
5.6 Evaluation Results 109
5.7 Summary 113

Chapter 6. Conclusion and Future Work 114

References 117

List of Tables

3.1 Accuracy of Our Criticality Computation Methods 34

3.2 Run Time of Our Criticality Computation Methods 34

4.1 Area Reduction under 95% Performance Yield Constraint 59

4.2 Area reduction under 99% Performance Yield Constraint 59

4.3 Run Time Overhead of Variation Aware Method 60

4.4 Power Yield Under 99% Performance Yield Constraint 74

4.5 Power Yield Under 99% Performance Yield Constraint with Power Constraint Relaxation 75

5.1 Yield Improvement over a Deterministic Mapper with 99% Performance Yield Target 110

5.2 Yield Improvement over a Deterministic Mapper with 90% Performance Yield Target 111

5.3 Cost Saving over a Deterministic Mapper with 99% Yield Target 112

5.4 Cost Saving over a Deterministic Mapper with 90% Yield Target 112

List of Figures

1.1 Process variations cause large variations in performance and leakage power (courtesy Intel [13]). 2

1.2 Performance variations at different technology nodes (courtesy IBM [41]). . . 2

1.3 The execution times of X and Y have independent Gaussian distribution $N(\mu, \sigma^2)$. The WCET is calculated as $\mu + 3\sigma$. X+Y follows ($N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$). Therefore, $WCET_{X+Y} < WCET_X + WCET_Y$. 5

3.1 Circuit diagram 20

3.2 The circuit timing graph. 21

3.3 The critical regions. 27

3.4 The example of the segment of the path in a timing graph. 28

3.5 The pseudo code of the criticality computation 31

4.1 Yield computation for a synthesized DFG. The multiplication operation is bound to a two-clock-cycle module and two additions share the same module. 46

4.2 An example of resource sharing for two addition operations and the comparison of worst-case execution time (WCET) based and statistical analysis based approaches. Two addition operations are mapped to an adder, and an additional multiplexer is required. The delay distributions of the adder and the multiplexer are shown as PDF (probability distribution function). The delay distribution of the critical path after resource sharing is also shown as PDF. 50

4.3 An example illustrates the effectiveness of the performance yield metric. The critical path delay distributions of two synthesis resultant hardware are shown in PDF. When clock cycle time is set to T1, the synthesis result with D1(t) is better than that with D2(t) in terms of performance yield. However, when clock cycle time is set to T2, the synthesis result with D2(t) is better than that with D1(t). However, if we use worst-case delay to evaluate the results, we always choose the synthesis result with D1(t). 77

4.4 Accurately evaluating the performance improvement requires to consider both the path delay improvement and the criticality. Four cases are shown with different path delay improvements and values of the criticality. For case 1 and case 4, it is obvious that the path delay improvement and the criticality can represent the performance yield improvement. In case 2 and case 3, one metric, i.e. either the path delay improvement or the criticality, is not sufficient to represent the performance yield improvement. 77

4.5 The pseudo code of variation aware optimization of DFG 78

4.6 The pseudo code of generate moves for resource sharing of DFG 78

4.7 The pseudo code of generate moves for resource binding of DFG 78

4.8 (a) The delay variation (normalized sigma/mean) for 16-bit adders in IBM Cu-08(90nm) technology; (b) The delay variation (normalized sigma/mean) for 16-bit adders in IBM Cu-08(90nm) technology. (Courtesy of K. Bernstein, IBM [11]). 79

4.9 An example of module selection for an adder and the comparison of worst-case execution time (WCET) based and performance yield based module selection. The delay distribution of two different type of adders are shown as PDF (probability distribution function), and the area of adder 2 is smaller. 80

4.10 The adder delay distribution can be adjusted by post-silicon ABB techniques . . 80

4.11 The pseudo code of variation aware optimization in module selection 81

4.12 The pseudo code of optimal body biasing of DFG 81

4.13 The pseudo code of variation aware optimization of DFG 81

4.14 Power yield improvement over worst-case based deterministic module selection, with 90% performance yield constraint. 82

5.1 An example of process-variation-aware task scheduling for an NoC architecture. The NoC architecture contains four PEs, *PE1-4*, and these PEs are connected by routers. For illustration purpose, this simple example does not shows the communication mapping. Assume that *PE1* has larger delay variation than *PE2* due to intra-die variation. $M1$ and $M2$ are two schedules for the possible placement of *Task 3*. In schedule *M1*, *task 3* is scheduled onto *PE1*, which causes a large completion time variation. In schedule *M2*, *task 3* is scheduled onto *PE2*, which results in smaller completion time variation. In (a) and (b), the distributions of the completion times of the two different task schedules *M1* and *M2*, denoted as $CPT_{M1}(t)$ and $CPT_{M2}(t)$, are shown here as Probability Distribution Functions (PDF). 92

5.2 During scheduling, some tasks have not been mapped to PEs, and paths in both the *scheduled* and *unscheduled* sub-task graphs contribute to the delay of the entire task graph. The path going through *Task 4* consists of two paths, *p1* and *p2*, where *p1* is the path from the start task node, *Task 0*, to *Task 4*, and *p2* is the path from *Task 4* to the end task node, *Task 8*. 99

5.3 Yield computation for a partially scheduled task graph to address *structural dependency*. The *virtual arc* from task 1 to task 6 indicates that task 1 and task 6 are scheduled onto the same PE and there is no data transferring on the arc (1,6). Virtual arcs have to be included in the cut-set, since they may lie on the critical path. 101

5.4 The pseudo code of the proposed variation-aware task scheduling process . . . 103

5.5 An example of path allocation for the path from s to d in a mesh network is shown in two steps: 1) perform a BFS search in the shaded mesh network to identify best links; 2) extract the optimal path according to the best link flags (small red/dark square). 106

5.6 The pseudo code of routing algorithm. 108

Acknowledgments

I would like to thank my advisor, Professor Yuan Xie. I am most grateful for his enormous patience in guiding me on professional and personal matters since I came to Penn State. He has provided me a lot of opportunities to meet with other leading experts in my research area. As my role model, he always thinks more of others, and is eager to help me succeed.

I am extremely grateful to Professor Mary Jane Irwin for her continuous unconditional support and insightful suggestions even she is extremely busy. I would also like to thank Professor Vijay Narayanan for his invaluable advises, which help me grow in my research, and also introducing me to the leading experts. I also very grateful to Professor Zhiwen Liu for serving on my Ph.D thesis committee and providing me with insightful comments on my thesis.

I would like to thank Mr. Kerry Bernstein from IBM for his great advises on research and helping me out at various time. I am very grateful to Dr. Andres Tackach for his great support and invaluable advises while I was an intern at Mentor Graphics.

I also want to thank my colleges and friends, Raj Ramanarayanan, Guangyu Chen, Lin Li, Feihui Li, Chrys Nicopoulos, Ozcan Ozturk, Kevin Irick, Xiaoxia Wu, Guangyu Sun, Ding Yang, Mike Debole, Paul Falkenstern, Soumya Eachempati, Sri Hari Krishna Narayanan, Jungsub Kim and Ramakrishnan Krishnan, for their great help.

Finally, I'd like to thank my family, especially my mother, for their unconditional love and support.

Chapter 1

Introduction

Aggressive technology scaling enables integration of billions of transistors [12]. As a result, entire systems can now be integrated on a single chip die [67, 35]. Meanwhile, designers have also relied on technology scaling [109] to enhance system performance for three decades. However, this irreversible momentum toward deep sub-micron process technologies for chip fabrication has resulted in significant variations in key transistor parameters. The increased magnitude of the process variations leads to the failure of traditional design for worst case approaches. Thus, the limitations of the worst case design methodologies call for new design methodologies to mitigate the impact of process variations.

1.1 The Trend in Process Variations

Process variations mainly arise from random dopant fluctuations and sub-wavelength lithography [12]. As the feature size approaches the optical resolution limit, the process variations cause significant variations in the device parameters [74] [7]. These variations in key transistor parameters, in turn, cause significant performance and power deviations from nominal values in identical hardware designs. For example, Intel has shown in Fig. 1.1 that process variability can cause up to a 30% variation in chip frequency and up to a 20x variation in chip leakage power for a processor designed in 180 nm technology [13].

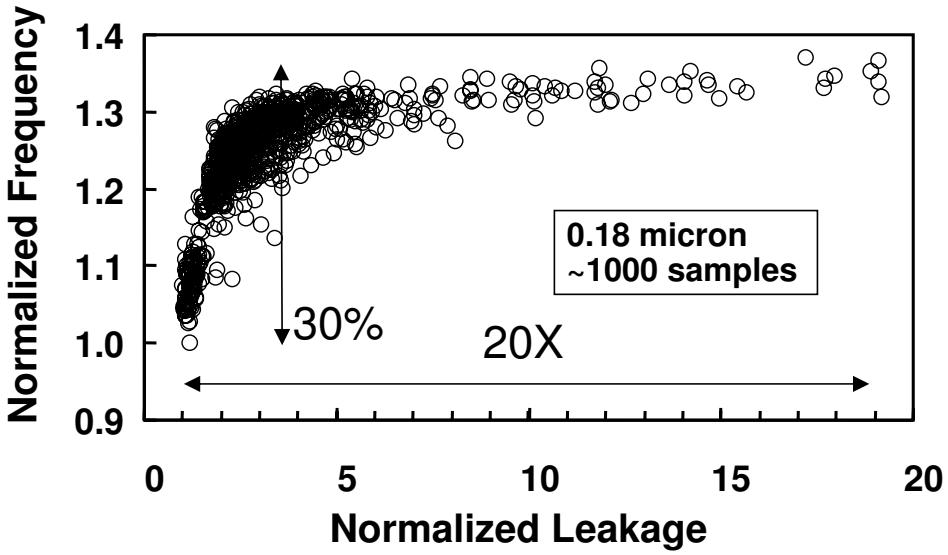


Fig. 1.1. Process variations cause large variations in performance and leakage power (courtesy Intel [13]).

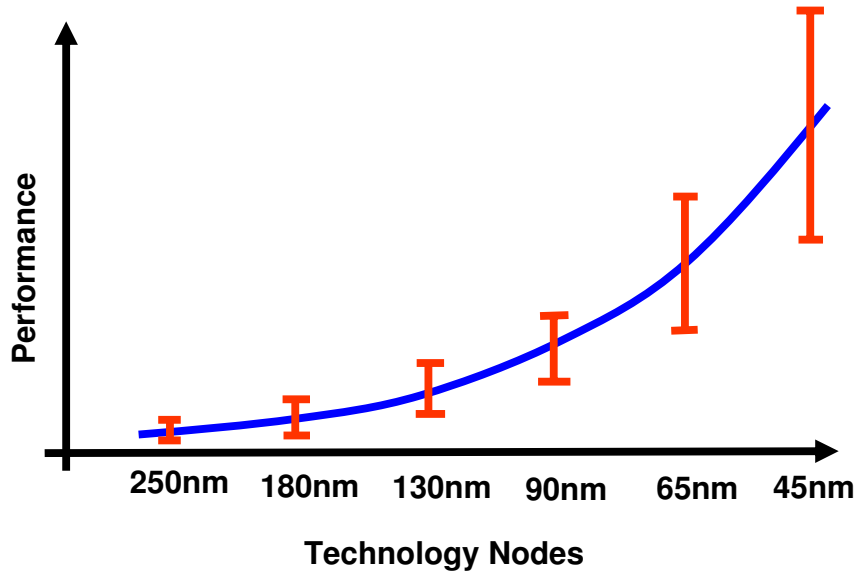


Fig. 1.2. Performance variations at different technology nodes (courtesy IBM [41]).

Parameter variation can be classified into two categories according to spatial characteristics: variation across identically designed neighboring transistors is called *within-die variation*, while variation across different identically designed chips is aptly called *inter-die variation*. Previously, the with-die variation was not significant, and could be ignored for the technology nodes above 130nm [15]. However, the with-die variation continuously grows and contributes significantly to the total variation as technology moves below 90nm [15] [75]. In addition to the classification according to spatial characteristics, the variations can also be categorized into systematic variations and random variations. The systematic variations can be predicted and modeled. In contrast to systematic variations, random variations come from fundamental randomness, or the mechanisms of random variations are not fully understood [73, 75]. As the feature size reduces, the magnitude of random variations is expected to grow compared to the systematic variations.

As technology scales further, performance variations become even more pronounced. For example, the worst case performance at 45nm technology could even offset the gain of technology scaling as shown in Fig. 1.2. The discrepancy between the expected and worst case performance parameters continuously increase, leading to significant performance loss. Thus, it has been predicted that the major design focus in sub-65nm VLSI design will shift to dealing with variability [13, 12].

1.2 The Failure of Current Solutions for Process Variations

The previous section described the trend in process variations as technology scales. In fact, designers have been aware of process variations for more than a decade. Current industry practice for coping with process variation relies on the corner based deterministic analysis and

design for worst case scenario. However, due to the increased magnitude of process variations, these deterministic approaches suffer from the following disadvantages:

- **Overly Conservative**: Due to the increased magnitude of the process variation, the discrepancy between the measurement and the worst-case analysis results continue to increase [12]. Worst-case analysis without taking into account the probabilistic nature of the manufactured components can also result in an overly pessimistic estimation in terms of performance. For example, Figure 1.3 shows that simply adding the Worst Case Execution Time (WCET) of two tasks can result in a pessimistic estimation of the total WCET, and may end up necessitating the use of excess resources to guarantee real-time constraints.
- **Computational prohibitive**: the increased magnitude of the parameter variations leads to the increase in the number of process corners. Timing closure requires designers to verify the setup and hold time at all possible corners. However, analyzing all possible process corners is computationally prohibitive since the computation complexity increases exponentially with the increase in the number of process corners [25].

As a result, a shift in the design paradigm, from today's worst-case deterministic design to statistical or probabilistic design [97, 71], is critical for deep sub-micron design. A classic approach to perform statistical analysis relies on Monte Carlo method. In this method, the analysis is performed for each sample of random variables. Although it is simple and accurate, its computation cost is prohibitive. Thus, statistical analysis and optimization has been extensively studied recently. A variety of gate level statistical timing analysis techniques, such as path based approaches and block based approaches have been developed [98] [16] [76] [3] [6] [72] [26] [88] [33] [61] [45] [23]. Based on the analysis, statistical optimization techniques, such as gate sizing,

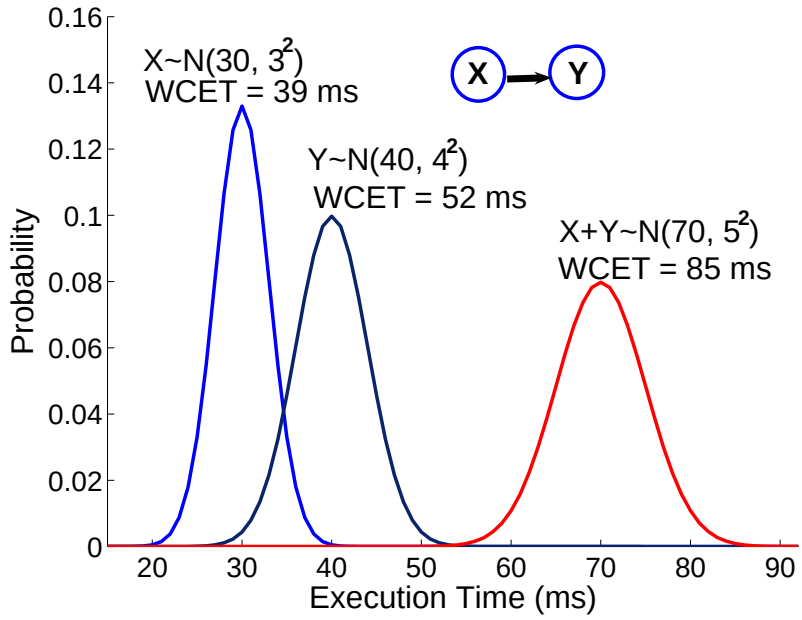


Fig. 1.3. The execution times of X and Y have independent Gaussian distribution $N(\mu, \sigma^2)$. The WCET is calculated as $\mu + 3\sigma$. X+Y follows $(N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2))$. Therefore, $WCET_{X+Y} < WCET_X + WCET_Y$.

vth assignment, and buffer insertion, have been proposed [87, 36, 19]. However, *the majority of the existing analysis and optimization techniques related to process variations are at the lower level (device or logic gate level).* In the domain of high-level synthesis and hardware/software co-synthesis, process-variation-aware research is still in its infancy [40] [68]. *It is important to raise the process variation awareness*, because the benefits from higher-level optimization often far exceed those obtained through lower-level optimization. Furthermore, higher-level statistical analysis enables early design decisions to take lower-level process variations into account, avoiding late surprise and possibly expensive design iterations.

1.3 Contribution and Organization

The previous section showed the importance of the shift from deterministic to statistical design methodology at all levels of the design hierarchy. Therefore, the research presented in this thesis will attempt to facilitate this design paradigm shift with a focus on high level synthesis. The aim is to develop a comprehensive solution to statistical analysis and optimization ranging from gate level to system level.

The following chapters start with an overview of the recent developments in state-of-art statistical design techniques at gate level, module level and architectural level in Chapter 2. In Chapter 3, the author focuses on developing a fast and accurate statistical criticality analysis method to lay down a foundation for higher level statistical design techniques [104].

Chapter 4 shifts the focus to statistical analysis and optimization in high level synthesis [101] [103] [100]. The proposed techniques consist of two parts:

- **Variation aware resource sharing and binding**: the author first presents an *efficient* variation-aware resource sharing and binding algorithm in high level synthesis to take into account the

performance variations for functional units. The performance yield, which is defined as the probability that the hardware meets the target performance constraints, is used to evaluate the synthesis result; and an efficient metric, called *statistical performance improvement*, is used to guide resource sharing and binding. The proposed algorithm is intergraded into a commercial synthesis framework. The effectiveness of the proposed algorithm is demonstrated with a set of industrial benchmark designs, consisting of blocks commonly used in wireless and image processing applications.

- **Joint optimization considering post silicon tuning at design time**:in addition to design time optimization, the parametric yield can be further improved by combining both *design-time* variation-aware optimization and *post silicon tuning* techniques (such as adaptive body biasing (ABB)). The author proposes a module selection algorithm that combines design-time optimization with post-silicon tuning (using ABB) to maximize design yield. A variation-aware module selection algorithm based on efficient performance and power yield gradient computation is developed. The post silicon optimization is formulated as an efficient sequential conic program to determine the optimal body bias distribution, which in turn affects design-time module selection.

In Chapter 5, the author investigates the impact of the process variations on task scheduling and allocation [99], and develops a variation-aware task and communication mapping for MPSOC (Multiprocessor System-on-Chips) that uses network-on-chip (NoC) communication architecture. The proposed mapping scheme accounts for variability in both the processing cores and the communication backbone to ensure a complete and accurate model of the entire system. A new design metric, called *performance yield* and defined as the probability of the assigned

schedule meeting the predefined performance constraints, is used to guide both the task scheduling and the routing path allocation procedure. An efficient yield computation method for this mapping complements and significantly improves the effectiveness of the proposed variation-aware mapping algorithm.

Finally, Chapter 6 concludes the thesis and discusses the future work.

Chapter 2

Overview of the Prior Art

The process variation problem has been recognized more than a decade ago [24, 28, 48, 49, 29], early approaches have relied on corner based analysis. As the variations in the device and interconnect parameters continuously increase, these approaches become too pessimistic and may end up using excess resources to guarantee desired yield. Thus, variation-tolerant methodologies have recently become a major research focus due to the economic motivations. Borkar et al. [13] first brought the attention of the process variations to the academia, and advocate a design paradigm shift from the deterministic to statistical design approach [12]. In this chapter, the author provides an overview of the recent developments in statistical design techniques. These techniques are categorized into three major types: gate level approaches, module level approaches and, architectural and system level approaches.

2.1 Gate-level Statistical Analysis and Optimization

Extensive researches have been done in statistical static timing analysis at gate level [98] [16] [76] [3] [6] [72] [26] [88] [33] [61] [45] [23]. These approaches can be categorized into two major types: path-based approaches and block-based approaches [98]. Path-based approaches perform timing analysis with a depth first traversal of a timing graph, by choosing a set of paths with high probability being critical for analysis. The correlation due to the path sharing and global sources of variation can be easily captured using path-based approaches [56] [98].

However, the set of the paths selected for statistical timing analysis is hard to determine. Existing approaches choose the set of paths based on the results of static timing analysis, which might miss some important paths. In comparison, the block-based approaches perform timing analysis by traversing the timing graph in a levelized "breadth-first" manner [98]. In such approaches, the parameterized delay model approximates the timing quantities as normal distributions. With this approximation, the block-based approaches can effectively handle the correlation due to the path sharing and the variation of global parameters [56] [98] .

After the statistical timing analysis, the statistical delay information of the circuits is available. The slack is not a deterministic value and many paths could be critical in some process subspaces. Thus, slack and critical path metrics used in the deterministic static timing analysis become less effective in guiding the circuit optimization. Visweswariah et al. [98] first defined the concepts of the *criticality* of the path and the arc/node. The criticality of the path is defined as the probability of the path being critical. The criticality of the node/arc is defined as the probability of the node/arc on the critical path.

In addition to the statistical timing analysis, the statistical leakage power analysis has also become a import research topic, as the variation of the leakage power increases dramatically and the leakage power has become a major component of total chip power consumption. A variety of techniques have been developed to model the leakage power distribution. Based on the Monte-Carlo simulation results, Srivastava et al. develop an analytical model to estimate the mean and variance of the leakage power distribution due to gate length and Tox [90]. Mukhopadhyay et al. [70] extend Srivastava et al.'s work by including gate and reverse biased junction band-to-band-tunneling (BTBT) leakage. In these works, the leakage power of a gate is modeled as a log-normal distribution, and the total leakage power is the summation of the leakage power of all

the gates in the circuit [57]. This model is a first-order approximation and may lead to significant estimation errors when the variation of the leakage power increases as technology moves below 90nm [57]. A quadratic approximation method can be used to improve the accuracy, however, the computation cost of this approach is prohibitive. Observing the sparse structure of the problem in the quadratic approximation approach, a projection-based algorithm is developed by Xin Li et al. [57] to reduce the computation cost. In this algorithm, a compaction method reducing the dimension of the matrix is developed to further reduce the computational complexity to be linear in the circuit size.

Similar to the analysis techniques, the traditional optimization techniques are deterministic oriented and based on the corner based analysis. After the statistical analysis, the power and timing results are not deterministic, thus, the statistical optimization techniques have to be developed. Many works have been done to augment the deterministic optimization techniques to be variation aware. Previously, the slack is used to guide the optimization, such as gate sizing and vth assignment, however, performance yield is proved to be a more effective metric after the statistical approaches have been adopted. A variety of yield driven gate sizing techniques have been proposed [87, 36, 19]. In these yield driven statistical optimization approaches, the yield computation is performed at each decision-making step of the optimization. Consequently, efficient yield analysis and perturbation computation methods have been developed. In addition to gate sizing, dual vth assignment is an effective optimization techniques to minimize the power while meeting performance requirements. In Srivastava et al.'s work [90], the timing constraint is derived from the statistical timing information while selecting the threshold value for the gate. The sensitivity based greedy approach is used to determine optimal threshold voltage assignment, and the sensitivity computation considers both the mean and variance of the performance

and power distribution. In Srivastava et al.'s approach, the sensitivity is computed based on one path each time. Davoodi et al. [22] extend this approach by computing the sensitivity for a set of paths simultaneously and developing a dynamic programming based approaches. The proposed method is proved to be more effective and efficient. Recently, other optimization techniques, such as buffer insertion, have been proposed to augment the deterministic approaches to be variation aware. Buffer insertion is proved to be an effective approach to reduce the wire delay. In a classic buffer insertion algorithm, the buffer is inserted based on the sensitivity of the cost function and the cost function is a fixed value. Khandelwal et al. extend this approach by using a probabilistic cost function and use a probabilistic optimization criteria to determine the optimal locations of the buffers [50]. The optimization techniques discussed belong to design time approaches, compared to the post silicon tuning, such as ABB techniques. As the variation continuously grows, these design time optimization techniques are insufficient to ensure the performance and power yield. Adaptive body biasing, a classic powerful post silicon tuning technique, adjusts the transistor parameters to effectively compensate for the process variations after the chip is fabricated. Thus, to further improve the parametric yield, a joint optimization approach taking into account post silicon tuning at design time can be applied to counter the effects of process variations. A more recent work by Mani [64] et al. developed an optimization framework to take advantage of both design time optimization and post silicon tuning techniques.

The clock skew plays an import role in high performance system design. Similar to the timing analysis, clock skew analysis relied on Monte Carlo simulation or corner based approaches. Due the limitations of these two approaches, Agarwal et al. [2] formulate statistical clock skew analysis problem and propose an efficient algorithm with linear run-time in circuit

size. The proposed clock skew analysis technique takes into account the intra-die variation and topological correlation. The analysis is performed in a bottom-up fashion in the clock tree.

2.2 Architectural and System Level Design Approaches

The impact of process variability can be effectively reduced if it is considered from the very early stage of the design [14]. In this section, the author presents very recent works focusing on variation aware pipeline design, variation tolerant cache architecture, and system level variation aware techniques.

- **Variation Tolerant Pipeline Design:** Recently, Kim et al. [51] investigate the impact of the process variation on the optimal combination of pipeline depth and and parallel processing width and conclude that shallower pipelining and narrower parallel is preferred due to the process variations. From a different perspective, Tiwari et al. [93] find that the process variability could exacerbate the unbalance between the pipeline stages. Based on this observation, cycle time stealing is applied to reduce the performance loss due to process variations by clock skewing. Further, a empty stage is added to the pipeline stage to provide the additional slacks, which is also used to reduce the power consumption.
- **Yield-aware micro-architecture data cache design**. Agarwal et al. [5] attempt to identify the cache failure mechanisms and classify them into four types of failures. Based on this study, Agarwal et al. propose a new architecture: 1) adaptively resizing the cache to avoid the faulty cache blocks after the self test 2) storing the information of faulty cache blocks in config storage, which is obtained using BIST techniques and loaded when the system is rebooted. Unlike Agarwal et al.'s work, Ozdemir et al. [77] focus on the power issues and propose four schemes to improve the cache yield: 1) powering down the cache ways with large leakage

and delay violation; 2) powering down cache lines with large leakage and delay violation; 3) allowing variable cache access latency to adapt the delay variability in cache; 4) applying the first and third approach simultaneously. Liang et al. [60] propose a circuit-architectural approach to design a process variation tolerant memory architecture. In the proposed approach, 3T1D (3-transistor, 1-diode dynamic memory) DRAM cell design is proposed to replace the 6T (6-transistor static memory) SRAM. The proposed memory cell design can tolerate large parameter variations and is only subject to the data retention time variation, which can be compensated using micro-architectural approaches.

- **System level variation aware techniques** Borkar et al. investigate the effects of different architectural design decisions, such as logic depth, number of micro-architecture critical paths, on the process variations. Based on this study, possible solutions, such as the body biasing techniques, vdd variation reduction methods and temperature control techniques, are presented to mitigate the impact of process variations. Marculescu et al. [66] develop a micro-architectural model to investigate the impact of the process variations at architectural level. A joint energy, performance and variability metric is employed to compare the process variability resilient capability of a fully synchronous system and that of a asynchronous system. This work show that the globally asynchronous and locally synchronous processors is more resilient to the process variability. Liang et al. [59] observe that the process variations could introduce a large gap in the delay between the register files and the execution unit. Based on this observation, a variable latency execution unit and register file scheme is proposed to close the gap between these components. In addition, variable latency register files can improve the yield of the register files with a small IPC penalty. Further, a test strategy to apply the proposed technique is described for the practical considerations.

2.3 Variation Aware High level synthesis

Although we have seen some very recent work considering the impact of process variations at the architectural and system level [51, 66, 58], high-level synthesis research related to process variations is still in its infancy.

Recently, Hung et al. [40] first investigate the impact of the process variation on the high level synthesis. In their work, a discrete timing model is used to model the timing variability of the function units. Base on this model, the timing analysis was integrated into the high level synthesis flow and the performance yield was used to evaluate the synthesis results. The key limitations of this work are: 1) each step of the synthesis is still deterministic oriented; 2) the proposed simulated annealing method is hardly to guarantee the performance yield, resulting in significant run time overhead.

Observing these limitations, a recent proposed work by Jung et al. [47]attempts to investigate the effects of the synthesis decisions on the performance yield. Based on the study, three rules are derived: 1) cascading the function units trades off the schedule length and performance; 2) resource sharing always leads to performance yield gain; 3) yield equivalent pattern can be used to increase the performance yield and maximize the resource sharing opportunities. However, these rules are not always true. For example, resource sharing requires additional multiplexors, and this may result in significant timing overhead. Thus, resource sharing could potentially reduce the performance yield. Therefore, the proposed algorithm can generate misleading results. In addition, identifying the yield-equivalent pattern significantly complicates the algorithm, thus, the proposed scheme may not be able to find the optimal solution.

These two works represent the first attempts to incorporate the process variability into the synthesis framework. However, these two works only consider the timing variability due to the parameter fluctuations. As technology moves below 90nm, the power consumption is a limiting factor for further technology scaling. Consequently, power consumption has to be taken into account in the synthesis framework. A recent work [68] attempts to take into account the power variability and propose a simulated annealing based algorithm to minimize the power under the performance constraint. In their work, the synthesis flow consists of four engines to perform the characterization, process input and generate the output results. A library of data path components was built based on characterization of equivalent NAND logic gates. Although the approach might simplify the library characterization, it might leads to significant errors. The proposed simulated annealing algorithm tries to minimize both the mean and variance of the leakage power. However, the proposed algorithm ignores the timing variability and timing analysis is not integrated into the synthesis flow. Consequently, the algorithm fails to utilize the additional ”performance slack” obtained by using the statistical performance analysis.

Chapter 3

A Criticality Computation Method in Statistical Timing Analysis

Chapter 2 reviews the state-of-art statistical static timing analysis techniques to effectively predict the circuit performance and verify the timing under large process variations. After the statistical timing analysis, the statistical delay information of the circuits is available. The slack is not a deterministic value and many paths could be critical in some process subspaces. Thus, slack and critical path metrics used in the deterministic static timing analysis become less effective in guiding the circuit optimization. Visweswariah et al. [98] first defined the concepts of the *criticality* of the path and the arc/node. The criticality of the path is defined as the probability of the path being critical. The criticality of the node/arc is defined as the probability of the node/arc on the critical path.

In this chapter, the author develops a critical region concept for the path and arc/node in a timing graph, and proposes an efficient method to compute the criticality for paths and arcs/nodes simultaneously by a single breadth-first graph traversal during the backward propagation to facilitate the criticality computation considering the correlation. Instead of choosing a set of paths for analysis prematurely, the author develops a new property of the path criticality to prune those paths with low criticality at very earlier stages, so that the proposed path criticality computation method has linear complexity with respect of the timing edges in a timing graph.

To improve the computation accuracy, cutset and path criticality properties are exploited to calibrate the computation results. The experimental results on ISCAS benchmark circuits show that the proposed criticality computation method can achieve high accuracy with fast speed.

3.1 Introduction and Motivation

Visweswariah et al. proposed a very simple fast method to compute the criticality for the path and the node/arc [98]. However, this approach is valid under the assumption of independence among the paths, which is not true due to the path convergence and the variation of global parameters [56] [110] [108]. Li et al. [56] and Zhan et al. [110] first pointed out the limitations of that approach. Li et al. [56] proposed a sensitivity based criticality computation method for arcs in the timing graph. Path sensitivity and arc sensitivity are defined as the mean value of the maximal circuit delay over the individual path delay and arc delay, respectively. The criticality of node/arc is computed based on sensitivity propagation using chain rule through a timing graph. Xiong et al. [108] computed the criticality as the probability of the edge slack being larger than its complement edge slack. Binary partition tree was used to speed up the criticality computation. Zhan et al. [110] proposed two techniques to compute the path criticality based on *max* operations and Monte Carlo integration. In his method, paths are chosen from a preselected path set for criticality computation. However, similar to the path-based statistical timing analysis, it is not clear how to determine the preselected path set. If the paths are chosen based on the static timing analysis, important paths could be missed.

In this chapter, the author proposes a new approach to compute the *path* and *arc/node* criticality *simultaneously* by a single breadth-first graph traversal based on the concept of the *critical region* (will be defined in Section 3.3). Instead of selecting the paths using static timing

information as [110], the author develops a new *path criticality property* to prune the paths with low criticality at very earlier stages. Pruning paths based on the statistical information significantly reduces the computation costs without sacrificing the accuracy. Existing *cutset and path criticality properties* are exploited to improve the accuracy of the criticality computation.

3.2 Background

In this section, the author first gives a brief introduction on the first order parameterized statistical timing analysis (SSTA). We then introduce the concept of statistical criticality. Finally we present the methods to determine the criticality of the node/arc without the simple independence assumption.

3.2.1 Canonical Model

In first order SSTA, the timing quantity (such as delay, arrival/required time and slew) is approximated as a linear function of random variables. In Visweswariah's paper [98], the first order approximation of the timing quantity is expressed as a canonical form:

$$a_0 + \sum_{i=1}^{n} a_i \Delta Y_i + a_{n+1} \Delta R_a \tag{3.1}$$

where a_0 is the nominal delay computed at the nominal values of the process parameters. ΔY_i represents the correlated variation, and ΔR_a represents the independent random variation. ΔY_i and R_a are independent and normally distributed random variables with zero mean and unit variance. a_i and a_{n+1} are the sensitivities to their corresponding sources of the variation.

3.2.2 Timing Graph

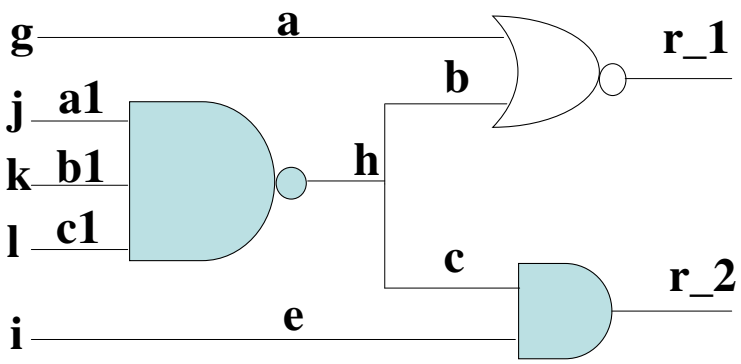


Fig. 3.1. Circuit diagram

In statistical timing analysis, the circuit is modeled as a timing graph G=(V,E). Fig. 3.1 and Fig. 3.2 depict a circuit diagram and the corresponding timing graph. The node, $v_i \in V$, in the timing graph represent the gate, primary input, primary output or the interconnects. The edge, $e(i,j) = (v_i, v_j) \in E$, represents the arc between the node v_i and v_j. The weight associated with the arc $e(i,j)$ in a graph represents the delay of that arc. Virtual source node and virtual sink node are connected to the primary inputs and primary outputs, respectively.

3.2.3 Atomic Operations

In statistical timing analysis, the distribution of the timing quantity is computed using two atomic functions *max* and *sum*. Assume that we have three timing quantities, A, B, and C in canonical forms (equation 3.2-3.4). The result of sum operation $sum(A, B)$ or max operation

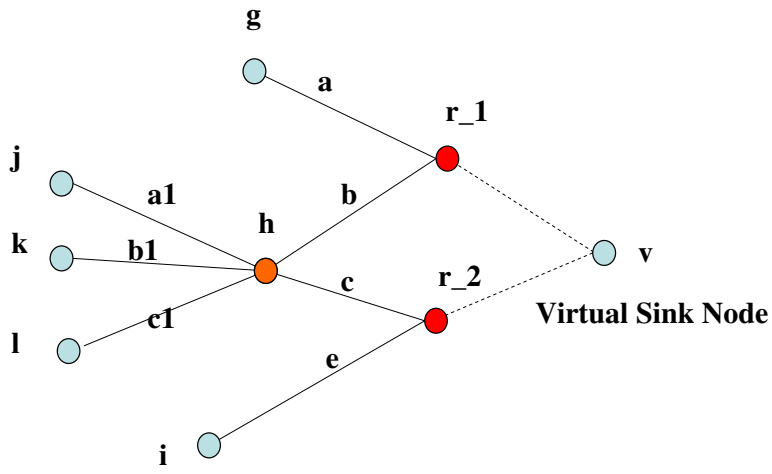


Fig. 3.2. The circuit timing graph.

$max(A, B)$ for the timing quantities is denoted as C.

$$A = a_0 + \sum_{i=1}^{n} a_i \Delta Y_i + a_{n+1} \Delta R_a \tag{3.2}$$

$$B = b_0 + \sum_{i=1}^{n} b_i \Delta Y_i + b_{n+1} \Delta R_b \tag{3.3}$$

$$C = c_0 + \sum_{i=1}^{n} c_i \Delta Y_i + c_{n+1} \Delta R_c \tag{3.4}$$

- *Sum* operation.

  The sum operation is simple. $C = sum(A, B)$ can be computed as $c_i = a_i + b_i$, where $i \in [0, n]$ and $c_{n+1} = \sqrt{a_{n+1}^2 + b_{n+1}^2}$.

- *Max* operation.

The *max* operation is quite complex. The maximum C of A and B is approximated as a normally distributed random variable with the same canonical form as shown in equation (3.4). Tightness probability and moment matching techniques [20] can be used to determine the corresponding sensitives to the process parameters. Tightness probability is defined as the probability of a random variable being larger than another. Clark's paper [20] provide an analytical equation to compute the tightness probability.

We first give the definition of the terms, which are used in the calculation of the $max(A, B)$

.

$$\phi(x) = exp(-\frac{x^2}{2}) \tag{3.5}$$

$$\Phi(x) = \int_{-\infty}^{y} exp(-\frac{x^2}{2}) \tag{3.6}$$

$$\theta = (\sigma_a^2 + \sigma_b^2 - 2\rho\sigma_a\sigma_b) \tag{3.7}$$

$$\alpha = \frac{a_0 - b_0}{\theta} \tag{3.8}$$

$$v_2 = (a_0^2 + \sigma_a^2)\Phi(\alpha) + (b_0^2 + \sigma_b^2)(1 - \Phi(\alpha)) + \theta\phi(\alpha) \tag{3.9}$$

We then show the calculation of the sensitivity values of C. c_0 is used to match the first order moment.

$$c_0 = a_0\Phi(\alpha) + b_0(1 - \Phi(\alpha)) + \theta\phi(\alpha) \tag{3.10}$$

The sensitivity to the correlated random variables can be obtained as a weighted sum of the a_i and b_i regarding to the tightness probability.

$$c_i = a_i\Phi(\alpha) + b_i(1 - \Phi(\alpha)) \ \ \forall\, i \in \ 1, 2, ..., n \tag{3.11}$$

Finally, the sensitivity to the independent random variable is calculated to match the variance, v_2, which is given by Clark [20].

$$c_{n+1} = (v_2 - \sum_{i=0}^{n}(c_i^2))^{\frac{1}{2}} \tag{3.12}$$

With the atomic operations defined, the timing quantities of the arrival time and the required time can be computed in *forward* and *backward* operations respectively. Our criticality computation is based on the results of the arrival time in SSTA, but it is not restricted to the first order SSTA. Note that our criticality computation method is *NOT* limited to first order SSTA, and using a *higher order* SSTA can improve its accuracy.

3.2.4 Concept of the Criticality

Visweswariah [98] proposed the following concepts of the criticality of the path and the criticality of arc/node.

- The criticality of the path is defined as the probability of the path being critical.
- The criticality of the node/arc is defined as the probability of the node/arc on the critical path.

These concepts can be used to guide the designer to identify the critical arc/node to perform the optimization. The gates with high criticality on the critical path can be sized up to improve the performance, while the arc/node with lower criticality can be sized down to save the power. In performance optimization, it is also very important for the circuit designer to identify the most important paths, which have highest probability being critical [110].

3.2.5 Criticality of Arc/Node

In this section, we show the arc/node criticality computation method, which is *NOT* restricted to the *independence assumptions* [98]. In Section 3.3, we will show the criticality computation for paths, which is based on the critical region computation for the node/arc.

For arc (h, r_i) in a timing graph, its criticality is expressed as:

$$Prob((AT(h) + d(h, r_i) > max_{g \neq h}(AT(g) + d(g, r_i)))| \\ r_i \; critical) \times Prob(r_i \; critical) \tag{3.13}$$

where:

- g represent all the other fan-in nodes of node r_i;
- $d(h, r_i)$ is the delay of the arc from node h to node r_i;
- r_i $(i \in [1, m])$ represents the set of the fan-out nodes of node h;
- $Prob(r_i \; critical)$ represents the probability of r_i being critical.

According to the lemma in [98], for node h in a timing graph, the criticality of node h can be calculated as the summation of the criticality of arcs originating from node h.

$$\sum_{i=1}^{m}\{Prob(arc(h,r_i)\ is\ critical)\} \tag{3.14}$$

3.3 Criticality Computation Method

In the previous section, the criticality is expressed as the form of $P(A|B)P(B)$. According to probabilities theory [79], $P(A|B)P(B)$ is equal to $P(A \cap B)$. With the help of Venn diagrams, $P(A \cap B)$ can be represented as the intersection of the two sets A and B [79]. To facilitate the criticality computation considering the correlations, we introduce the concept of the *critical region* for nodes/arcs and paths. We define the critical region of a node/arc as the process subspace where the node/arc is on the critical path. The critical region of a path is defined as the process subspace where the path becomes critical. The critical region is computed when we perform the backward operation on a timing graph. *With the critical regions determined, the criticality can be calculated using tightness function [98] and the equations in Section 3.2.* We classify the nodes in a timing graph into two types: 1) the nodes with a single fan-out 2) the nodes with multiple fan-outs. We first show the critical region computation for these two types of nodes and their corresponding arcs. We then introduce a Lemma to determine the critical regions of paths in a timing graph and presents a path criticality property. Finally, we shows our criticality computation method based on the concept of the critical region.

3.3.1 Critical Region Computation for Arc/Node

From Section 3.2, the criticality of the arc can be expressed as equation (3.13). Then, the critical region of the arc is the intersection of the critical region of its fan-out node and the

region where the arrival time (AT) of that arc determines that of the fan-out node. So the critical region of arc (h, r_i) is the intersection of the following two regions:

- The region where the arrival time of the arc (h, r_i) determines the arrival time of fan-out node r_i, i.e., $(AT(h)+d(h, r_i) > max_{g \neq h}(AT(g)+d(g, r_i)))$. We rewrite this condition as $(AT(h)+d(h, r_i)-max_{g \neq h}(AT(g)+d(g, r_i))) > 0$ and it is denoted as $F(h, r_i) > 0$
- The region where fan-out node r_i is critical, and it is denoted as $F(r_i) > 0$

So the critical region for arc (h, r_i) can be expressed as $min(F(r_i), F(h, r_i)) > 0$. With the critical region available, we use the *tightness function* [98] to compute the criticality of arc (h, r_i) as the probability of $min(F(r_i), F(h, r_i)) > 0$.

From equation (3.14), the criticality of the node can be expressed as the summation of those of its fan-out arcs. Thus, the combination of the critical regions of the arcs originating from that node is the critical region of the node. For example, in Fig. 3.2, assuming the critical regions for the fan-out nodes r_1 and r_2 are known, we show how to determine the critical region for node h. In Fig. 3.3, the dark areas, $F(r_1)$ and $F(r_2)$, represent the critical region of node r_1 and node r_2 respectively. Assuming that the process subspace where arc (h, r_1) determines the delay of the node r_1 is denoted as the region above dashed line b, the intersection of these two regions is the critical region of arc (h, r_1). We denote this intersection as $F(b)$. Similarly we obtain the critical region $F(c)$ for arc (h, r_2). For node h, its critical region is simply the combination of these two regions, $F(b)$ and $F(c)$.

Thus, for the internal node with multiple fan-outs, we first compute the critical regions of the arcs from that node to its corresponding fan-outs individually and then calculate the criticality over those regions for all its fan-out arcs. We then compute the criticality of the node by simply

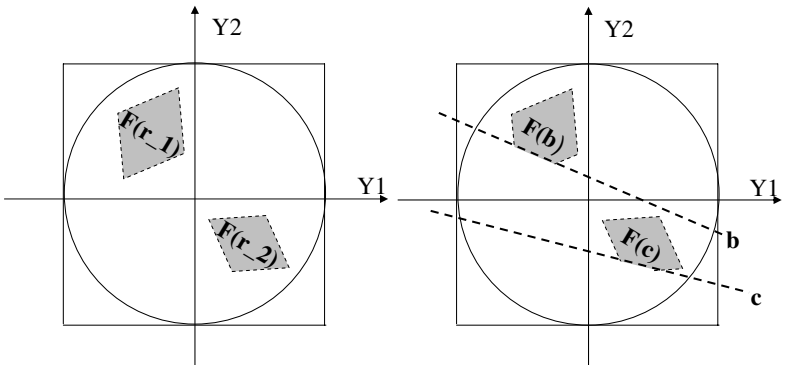


Fig. 3.3. The critical regions.

summing up the criticality of each arc originating from that node. For the internal node with a single fan-out, its the *critical region*/criticality is the same as that of the arc originating from the node to its fan-out.

3.3.2 Critical Region Computation for Path

To compute the path criticality, we first identify the critical regions of the paths, and Lemma 1 helps determine these critical regions. Then, we show that the critical regions of the paths are computed after a breadth first traversal of a levelized timing graph. Finally we develop a new path criticality property to prune the paths with low criticality to improve the efficiency of path criticality computation.

Lemma 1: A path's critical region is the intersection of all the critical regions of the arcs along the path.

Proof: We prove the statement with *proof by contradiction*. We can assume that 1) there exists at least one subspace in the process space other than the intersection where the path is critical or 2) there exists at least one subspace within the intersection where the path is *NOT*

critical. An arc is critical if the arc is on a critical path. All arcs along the path are critical when that path is critical, so that any subspace where a path is critical is part of the intersection of critical regions of all arcs along that path. Thus it contradicts the assumption 1). Since all the arcs on the path are critical, the slack of the path is equal to the slack of the arc. Thus the path has minimum slack, which means the path must be critical. It contradicts the assumption 2). Therefore, the path's critical region is simply the intersection of the critical regions of the arcs along that path.

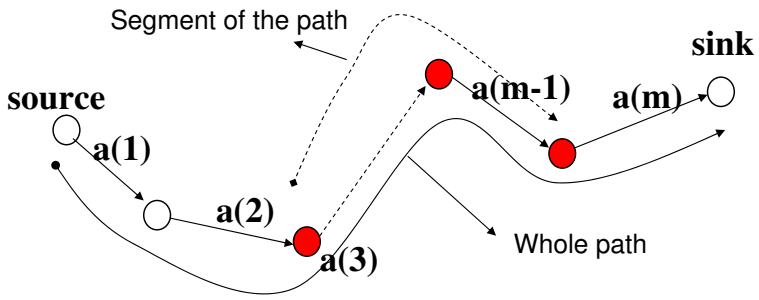


Fig. 3.4. The example of the segment of the path in a timing graph.

To compute the critical regions of the paths, we perform the critical region analysis for nodes/arcs in a breadth first traversal of a levelized timing graph. After this BFS traversal, the critical region of each PI contains the combinations of the critical regions of all the paths originating from the PI in the timing graph.

We show the correctness of the above statement with a stronger statement as the follows. We perform a breadth-first search for a levelized timing graph and compute the critical region

for node/arc from the highest level, m, to level 1, which consists of all the PIs. The nodes in this graph are levelized according to its distance to the primary inputs. We prove that: *at each level, the critical region of the node is the combination of all the critical regions of the paths originating from that node to the primary outputs*. We prove this statement with *proof by induction*.

Proof: Initially, at level m, the critical regions of the arcs from the primary outputs to the virtual sink node is computed. The statement is certainly true for level m.

Assume that the above statement is true for all level k, $n \leq k \leq m$. We prove that the statement is also true for level $n-1$. We classify the nodes into two types: 1) the nodes without any fan-outs 2) the nodes with one fan-out or multiple fan-outs. For the first type of the node, similar to the node at level m, we compute the criticality for the arc from the fan-out to the virtual sink node. Thus the statement is true. For the node with one or more fan-outs, since we assume that the statement is true for level k, where $n < k < m$, any fan-out of the node at level $n-1$ contains the critical regions of all the paths from that fan-out node to the primary outputs. We denote these regions as the set c_i, where $1 \leq i \leq p$, p is number of the path from the fan-out to the primary outputs. As shown in Section 3.3.1, the critical region of the node's fan-out edge is computed as the intersection of the critical region of its fan-out node and the region (denoted c_{arc}) where the AT of that arc determines the AT of the fan-out node. Effectively, the critical region of that fan-out edge is the combination of the intersections of c_i and c_{arc} for all the i, where $1 \leq i \leq p$. From Lemma 1, the intersection of c_i and c_{arc} is the critical region of the path from the node to the virtual sink node via its fan-out, because the critical region of the node is the combination of the critical regions of the arcs along the path. Any path originating from

the node is simply the path, from its fan-out to the virtual sink node, concatenated with the arc from the node to its fan-out. Thus, at level $n-1$, the statement is true.

In conclusion, since the base case is true and the inductive step is true, the statement is true for all the levels. Therefore, the critical region of a PI is the combination of the critical regions of all the paths originating from that PI to the virtual sink node. Since any path in the timing graph is from the PI to the virtual sink node, its critical region is determined when we perform criticality computation in a BFS manner. In addition, no path has been computed twice, since there is no single path going through any two fan-out arcs of that node.

To speed up the path criticality computation, we develop a new path criticality property. Assume that a path consists of a set of arcs: a_i, where i is from 1 to m, the segment of the path can be defined as a set of arcs: a_j, where j is from k to p, and $1 <= k < p <= m$. Fig. 3.4 shows an example of a segment of the path. With this definition, we have **property 1** as follows.

Property 1: The criticality of the path is not larger than the criticality of any segment of that path.

Proof: From lemma 1, the criticality of the path is equal to

$$Prob((a_1\ critical)\bigcap(a_2\ critical)\bigcap...\bigcap(a_m\ critical)) \tag{3.15}$$

and the criticality of the segment of the path criticality is calculated as

$$Prob((a_k\ critical)\bigcap(a_{k+1}\ critical)\bigcap...\bigcap(a_p\ critical)) \tag{3.16}$$

So the critical region of the path is the subspace of that of the segment. From the probabilities theory [79], the statement is true.

3.3.3 Computation Algorithm

In this section we first show the algorithm to compute the criticality of the paths and nodes in the timing graph. We then show a heuristic to improve the speed of the criticality computation. Finally we present a heuristic to improve the accuracy of the computation.

```
Criticality (netlist){
1. Compute the critical regions of the primary outputs (POs);
2. Compute the critical regions of the nodes/arcs;
3. Prune the nodes/arc with the criticality less than the criticality threshold;
4. Repeat 2 and 3 until the primary inputs are visited;
5. Compute the path criticality at the primary inputs;
6. }
```

Fig. 3.5. The pseudo code of the criticality computation

Fig. 3.5 shows the criticality computation algorithm. It takes the gate net-list as its input and compute the criticality for the arc/node and path simultaneously. The criticality computation involves a BFS traversal from the POs to the PIs. The critical regions of the nodes/arcs are determined from the critical regions of its fan-out arcs and nodes in the BFS traversal. After the traversal reaches the PIs, each PI contains the critical regions of all the paths starting from that PI. The path criticality is then computed over its critical region.

A brute-force path criticality computation approach leads to large computational overheads (its computation complexity is linear with respect to the number of the paths in a timing

graph). To speed up the computation, we use a heuristic to improve the performance of our algorithm. ***Property 1*** *enables us to prune the path/node/arc with a small criticality value at very earlier stages of path criticality computation.* Since a large portion of the paths in a timing graph has low criticality value, the computational complexity can be greatly reduced with the heuristic. As shown in Table 3.2 in Section 3.4, our path criticality computation method has a *linear computational complexity with respect of the timing edges*. Although the reduction of the computation cost depends on the designs, the experimental results demonstrate the effectiveness of our path pruning technique across the ISCAS benchmark circuits. In addition, we avoid the path selection problem in Zhan's approach [110]. We use the statistical timing information, instead of static timing information, to remove paths that are not important to the circuit designer.

The linear approximation of the gaussian distribution in *max* or *min* operation is a major source of the computation error in the criticality computation [98]. As the critical region computation proceeds to the level close to the primary inputs, the error due to the approximation accumulates. However, we extend the properties developed by Visweswariah et al. [98] and integrate them to calibrate the results.

Property 2: The sum of the criticality of the unpruned paths in a timing graph is *1.0-the sum of the criticality of the pruned paths*.

Property 3: The sum of the unpruned edge criticality of any cutset in a timing graph that separates the source from the sink node is *1.0-the sum of the criticality of the pruned edges of that cutset.*

From the properties in [98], the sum of the criticality of all the paths in a timing graph has to be 1.0. We record the sum of the criticality of the pruned paths as $prunedpath$. The sum of the unpruned path criticality denoted as $unprunedpath$ can be computed after the BFS traversal.

We normalize the criticality value of each path by multiplying a scaling factor $\frac{1}{pathtotal}$, where $pathtotal = prunedpath + unprunedpath$. Similarly, we compute the sum of the arc criticality of any cutset as $cutsettotal$, we normalize the criticality value of each arc belonging to that cutset with a factor $\frac{1}{cutsettotal}$.

3.4 Analysis Results

In this section, we present the analysis results and show our method can accurately compute the criticality with fast speed.

We implement our criticality computation method in C++ and integrate it into our statistical timing analysis tools. We conduct the criticality analysis on ISCAS 85 benchmark circuits to show the efficiency and accuracy of our method.

To demonstrate the accuracy of our method, we compare the simulation results against Monte Carlo simulation with 10,000 samples. We perform the statistical timing analysis and collect the statistical information of the critical path/node/arc for each sample. Table 3.1 shows the results of our method against the Monte Carlo techniques. In the second and third columns, we show the results of maximal and average criticality errors of the arcs with our methods. We also show the results of Li et al.'s [56] method in the fourth and fifth columns. In our method, the maximal and average criticality errors for arcs are less than 1.17% and 0.05% respectively. The maximal error of the path criticality against Monte Carlo techniques is less than 0.82% as shown in sixth column.

Table 3.2 shows the run time of our method against the basic statistical timing analysis. The circuit size in terms of the number of gates is given in column two. The run time of basic statistical timing analysis, the run time of criticality computation, and the relative overhead of

Table 3.1. Accuracy of Our Criticality Computation Methods

Circuit	critical node/arc		critical node/arc [56]		critical path
	max	avg	max	avg	max
c880	0.12%	0.010 %	1.3%	0.9 %	0.010 %
c1908	0.31%	0.006 %	3.4%	0.4 %	0.260 %
c2670	1.17%	0.043 %	2.6%	0.3 %	0.820 %
c5315	0.44%	0.016%	2.8%	1.8 %	0.340%
c6288	0.19%	0.047%	1.9%	0.6 %	0.100 %
c7552	0.43%	0.042 %	3.5%	1.1 %	0.400 %

Table 3.2. Run Time of Our Criticality Computation Methods

Circuit	Size	SSTA (sec)	Criticality Computation (sec)	Overhead
c880	0.50k	0.0052	0.0041	0.78
c1908	0.60k	0.010	0.007	0.7
c2670	0.78k	0.013	0.012	0.99
c5315	1.7k	0.026	0.016	0.62
c6288	3.8k	0.049	0.27	0.55
c7552	2.2k	0.042	0.015	0.38

criticality computation over statistical timing analysis are reported in column three, four and five, respectively. From column five in Table 3.2, we can see that the run time of the criticality computation for both paths and nodes/arcs is less than that of the corresponding SSTA. The run time overhead of our criticality computation over the basic statistical timing analysis tends to decrease as the size of the circuit increases. These results indicate that our path pruning technique reduces the computational complexity of path criticality computation to linear complexity with respect of the timing edges.

Compared to the previous work [56] [108] [110], our criticality computation method computes criticality for both paths and nodes/arcs. Our method has the same run time complexity as that of existing methods solely for the arc criticality computation [56] [108]. The results on the same benchmarks demonstrate that our method is more accurate in computing arc criticality compared to Li's approach [56]. In Zhan's approach [110], the path criticality computation is performed on a pre-selected set of paths, which might lead to missing some important paths. Our method avoid this problem by pruning the paths based on the statistical information.

3.5 Summary

In this Chapter, the author defines the critical region for paths and nodes/arcs in a timing graph. With this definition, the author develops an efficient method to compute the criticality for paths and the arcs/nodes simultaneously. A new property of the path criticality is used to prune the low criticality node/arc at the very early stages of computation to avoid the selection of the paths based on the static timing information. Cutset and path criticality prosperities are used to improve the accuracy in the criticality computation. Simulation results show our criticality computation method is very accurate and fast.

Chapter 4

Variation-aware High Level Synthesis

To make the design flow variation-aware, the statistical timing analysis has to be intergraded to the synthesis tool. Currently, hand coded register-transfer level (RTL) synthesis is widely used in the hardware design of complex systems. However, as the complexity of the system grows, this labor intensive synthesis technique is both time consuming and error prone. High-level synthesis (HLS) automates the process of generating RTL implementations from behavioral descriptions [81][31]. This automation also enables a wider exploration of the design space and has been reported to provide around five times faster design times as compared to manual RTL methodologies [30]. In addition, process variability could further complicates manual development of RTL. Thus, developing an automated RTL creation methodology that takes into account process variability becomes of primary importance as technology continues to shrink. Towards this goal, the author proposes variation aware high level synthesis design techniques. In this chapter, the author present two variation aware techniques in high level synthesis: a statistical resource sharing and binding technique; a module selection algorithm with joint design-time optimization and post-silicon tuning.

4.1 Introduction and Motivation

High-level synthesis consists of three steps: scheduling, resource sharing and resource binding. Traditionally, all these steps are performed based on deterministic worst-case performance analysis. However, technology scaling has resulted in significant variations in transistor parameters, such as transistor channel length, gate-oxide thickness, and threshold voltage. This manufacturing variability can cause significant performance deviations from nominal values in identical hardware designs. For example, IBM has shown that the worst case performance at 45nm technology node is very close to the nominal performance at 65nm technology node [42]. This substantial deviations from the nominal values makes the designing for the worst case infeasible. Worst case performance analysis without taking into account the process variation results in pessimistic estimation in terms of performance and end up using excess resources to guarantee design constraints.

To bring the process-variation awareness to the design flow, variation-aware performance analysis is integrated into the HLS process to cope with process variations. In addition, a new metric called *parametric yield* has been introduced [13, 64, 40]. The parametric yield is defined as the probability of the design meeting a specified constraint $Yield = P(Y \leq Y\_max)$, where Y can be performance or power. In Section 4.3, the statistical analysis for DFG, including the performance, power and yield analysis, is first presented to lay the foundations upon which our proposed variation-aware high level synthesis algorithm rests.

It is obvious that the parametric yield depends on all steps of high-level synthesis: scheduling, resource sharing and binding. These steps are usually interact with each other during high-level synthesis, and influence the final parametric yield calculations. However, the resource

sharing and binding have direct impact on the performance yield. Thus, in section 4.4, we focus on the variation aware resource sharing and binding.

In addition to these *design-time statistical optimization* approaches, to further maximize design yield, designers can rely on another complementary strategy: *post-silicon tuning*. Design time statistical optimization approaches, such as gate sizing and multiple vdd/vth selection, use statistical timing/power analysis to explore design space, and maximize parametric yield. The design decisions are the same for all fabricated dies and the decisions are made at the design-time (i.e., pre-silicon). As a result, some dies may inevitably miss the target power-delay envelop. In contrast to design time optimization techniques, post silicon optimization approaches are performed after the fabrication. Techniques such as adaptive body biasing (ABB) and adaptive supply voltage [64, 17, 94, 8] can be used to tune the fabricated chips, such that the variation in delay/power can be reduced. Compared to design-time solution, the post-silicon tuning decision is different for each differently fabricated die. For example, FBB (Forward Body Biasing) can be applied to slower dies such that the delay becomes faster at the expense of higher leakage power, and RBB (Reverse Body Biasing) can be applied to faster dies such that the circuit is slowed down but the power is reduced. Thus, in Section 4.5, we propose a variability-driven module selection algorithm that combines design-time optimization with post-silicon tuning (using adaptive body biasing) to maximize design yield. To the best of our knowledge, this is the first variability-driven high level synthesis technique that *considers post-silicon tuning during design time optimization*.

4.2 Related Work in Variation-aware High Level Synthesis

Related work pertaining to this paper can be divided into three different categories: traditional high-level synthesis approaches, statistical timing analysis and optimization, and recent developments in variation aware high-level synthesis.

Extensive research has been done on high-level synthesis for over two decades [81][31]. Research in high-level synthesis has focused on the following core steps: scheduling, resource sharing and binding. Scheduling is an NP-complete problem and while formulations based on ILP have been proposed [43], algorithms are in general based on heuristics to guide how operations are scheduled. For example, many scheduling approaches are based on some variation of list-scheduling with heuristics to guide how operations are scheduled based on their *urgency* [34], their *mobility* [78] or by attempting to balance their distribution [80] [52]. A number of resource sharing and binding techniques, such as clique partitioning algorithm [95] and the $LEFT\_EDGE$ algorithm [54], have been explored. Recently, a number of high-level synthesis techniques have been proposed to reduce the power, the temperature and interconnect delays [55, 63, 91] [69]. However, all these approaches are developed based on worst case timing analysis.

Recently, research on variation aware analysis and optimization techniques has received great attention both from academia and industry. Various techniques have been proposed for statistical timing analysis, such as path-based approaches and block based approaches [98]. Based on timing analysis, statistical optimization techniques, ranging from gate sizing to buffer insertion, have been explored. However, most of these techniques fall into either gate level approaches or device level approaches.

The impact of process variability can be effectively reduced if it is considered from the very early stage of the design [14]. Although we have seen some very recent work considering the impact of process variations at the architectural and system level [51, 66, 58], high-level synthesis research related to process variations is still in its infancy. Recently, Hung et al. [40] proposed to a *simulated-annealing* based HLS framework to take into account process variations. However, statistical timing information and yield analysis results are not used to guide the design exploration during synthesis, which may leads to suboptimal solutions. Jung et al. [47] attempted to use statistical timing information to perform high-level synthesis based on observations, which are only partly true. For example, one observation is that resource sharing always results in higher yield. This observation neglects the timing overhead of multiplexers introduced by resource sharing.

4.3 Statistical Analysis For DFG

In this section, we briefly describe our statistical timing/power analysis for a synthesized DFG [91] (in which all operations have been scheduled and bound to module instances selected from the resource library). The terminology and approach is similar to most of the gate-level statistical timing/power analysis approaches [24, 90, 84, 98, 4]. Although the fundamental idea is the same, that is, to consider process variations during timing/power analysis, the divergence occurs in that the allocated resource can be shared and that the sequencing order of operations with respect to clock cycle time must be enforced in HLS. This divergence makes statistical analysis at high-level synthesis a unique problem. In addition, we introduce parametric yield computation method for a synthesized DFG and present fast yield gradient computation methods.

4.3.1 Function Unit Delay and Power Modeling

In this section, the piecewise linearized delay model for function units is first introduced, and the exponential power model of function units is then presented. All these models are simple extension from the gate level models [45] and [90].

In the delay modeling, the delay of the function unit is expressed in terms of the gate length (l), and the threshold voltage (V_{th}). Piecewise linear approximation of the delay has been widely used in the gate level timing analysis. Thus, the delay of the function unit can also be expressed in a piecewise linear function. Suppose ΔV_{th} represents the deviation of the threshold voltage, and Δl represents the deviation of the gate length. The delay of a function unit, T_i, is expressed as:

$$T_i = a0_i + a1_i \Delta V_{th} + a2_i \Delta l \tag{4.1}$$

where $a0_i$ is the nominal delay computed at the nominal values of the process parameters without body biasing. $a1_i$, $a2_i$ represent the sensitivity to the deviation of threshold voltage and gate length respectively.

The power consumption of a function unit consists of dynamic power and leakage power. The dynamic power is relatively immune to process variation, while the leakage power is affected by process variation greatly, and it becomes a dominant factor in total power consumption as technology scales to nanometer region [90]. Our statistical leakage power model is based on the gate level model and the rms error of this gate level model is around 8% [90]. In this approach, the leakage power of each logic gate is expressed as a lognormal random variable in a canonical form, and the leakage power dissipation of a function unit, which consists of many gates, can

be computed as the sum of these random variables. This sum can be accurately approximated as a lognormal random variable using an extension of Wilkinson's method [90]. Consequently, the leakage power dissipation of a function unit can also be expressed as a lognormal random variable in a canonical form. Therefore, the leakage power of a function unit can be expressed as

$$P_i = exp(b0_i + b1_i \Delta V_{th} + b2_i \Delta l \tag{4.2}$$

where $exp(b_0)$ is the nominal leakage power computed at the nominal values of the process parameters. b_i are the sensitivities to their corresponding sources of the deviation.

4.3.2 Statistical Timing Analysis in HLS

In the statistical timing analysis for a synthesized DFG, the timing quantity is computed by using two atomic functions sum and max. Assume that there are three timing quantities, A, B, and C, which are random variables. The sum operation $C = sum(A, B)$ and the max operation $C = max(A, B)$ will be developed:

1. The sum operation is easy to perform. For example, if A and B both follow a Gaussian distribution, the distribution of $C = sum(A, B)$ would follow a Gaussian distribution with a mean of $\mu_A + \mu_B$ and a variance of $\sqrt{\sigma_a^2 + \sigma_b^2 - 2\rho\sigma_a\sigma_b}$, ρ is correlation coefficient.

2. The max operation is quite complex. Tightness probability [98] and moment matching [20] techniques could be used to determine the corresponding sensitivities to the process parameters. Given two random variables, A and B, tightness probability of random variable A is defined as the probability of A being larger than B. An analytical equation

in [20] to compute the tightness probability is used to facilitate the calculation of max operation.

The delay distribution of module instances can be obtained through gate-level statistical timing analysis tools [84][98] or Monte Carlo analysis in HSPICE. With the atomic operations defined, the timing analysis for the synthesized DFG can be conducted using PERT-like traversal [84].

4.3.3 Statistical Power Analysis in HLS

Our statistical leakage power analysis method is based on the gate level analysis approach [90]. In this approach, the leakage power of each logic gate is expressed as a lognormal random variable in a canonical form, the total power of the circuit can be computed as the sum of these random variables. This sum can be accurately approximated as a lognormal random variable using an extension of Wilkinson's method [90]. Since the leakage power dissipation of a function unit can also be expressed as a lognormal random variable in a canonical form as show in Section IV, the total power dissipation of the synthesized DFG is computed as the sum of the leakage power dissipation of the module instances in the DFG. Thus, this sum can also be approximated as a lognormal random variable in a canonical form using the extended Wilkinson's method. Therefore, the leakage power of each module instance can be expressed as

$$P_m = exp(m_0 + \sum_{i=1}^{n} m_i Y_i + m_{n+1} R_m) \tag{4.3}$$

where m_0 is the nominal value computed at the nominal values of the process parameters. Y_i represents the correlated variation, and R_m represents the independent random variation. Y_i

and R_m are independent and normally distributed random variables with zero mean and unit variance. m_i and m_{n+1} are the sensitivities to their corresponding sources of the variation.

The sum of the power dissipation of two modules is approximated as a lognormal random variable in the same format as expression (4.3). Assuming that $P_m = P_k + P_n$, the coefficient of P_m can be determined by moment matching [89],

$$m_i = log(\frac{E(P_k e^{Y_i}) + E(P_n e^{Y_i})}{(E(P_k) + E(P_n))E(e^{Y_i})}) \; \forall i \in [1, n] \tag{4.4}$$

$$m_0 = 0.5log(\frac{(E(P_k) + E(P_n))^4}{(E(P_k) + E(P_n))^2 + Var(P_k) + Var(P_n) + 2Cov(P_k, P_n)}) \tag{4.5}$$

$$m_{n+1} = [log(1 + \frac{Var(P_k) + Var(P_n) + 2Cov(P_k, P_n)}{(E(P_k) + E(P_n))^2}) - \sum_{i=1}^{n} m_i^2]^{0.5} \tag{4.6}$$

where $E(P)$ represents the mean of the random variable P, the $Var(P)$ represents the variance of the random variable P and $Cov(P, Q)$ is the covariance of the random variables P and Q.

4.3.4 Statistical Performance Yield Analysis for DFG

In a synthesized DFG, the operations are distributed to the clock cycles and bound to module instances selected from resource library. The operations in each clock cycle must finish execution within that clock cycle. The performance yield is calculated as the probability of the operations scheduled in each clock cycle meeting the clock cycle time constraints under the conditions that latency constraints and resource constraints are not violated. Assuming that the clock cycle time is T_clock, the latency constraints are N clock cycles, and the critical path delay

of the operations scheduled in clock cycle i is $Tmax_i$, the performance yield can be computed as

$$Yield_{delay}(DFG) = Prob(Tmax \leq T\_clock|constraints) \tag{4.7}$$

where $Tmax = max(Tmax_i), \forall i \in [1, N]$. $Tmax_i$ can be computed using the statistical timing analysis described in Section 4.3.2. Note that the max operation is defined in Section 4.3.2. The *constraints* represent the latency constraints and the resource constraints.

4.3.5 Performance Yield Gradient Computation

Based on the yield analysis method in the previous sub-section, a yield gradient method is described in this sub-section. A brute-force approach to performance yield gradient computation requires computing the performance yield of the entire synthesized DFG twice. To facilitate the yield computation in the module selection, we employ a *divide and conquer* method to avoid the yield computation over all the clock cycles in a synthesized DFG. The synthesized DFG is divided into blocks and each block contains minimum number of clock cycles (time steps) such that resource shared operations or operations bound to multiple-clock-cycle modules are in the same block. For example, as shown in Fig. 4.1, the multiplication operation is bound to a two-clock-cycle module and two addition operations share the same module. The $block1$ consists of two clock cycles, $CC2$ and $CC3$, such that the complete two-clock-cycle multiplication operation/two addition operations are in the same block. Assuming that the correlation between the different module instances is relatively small compared to the resource shared and multiple clock cycle operations, we can compute the yield of each block separately and approximate the

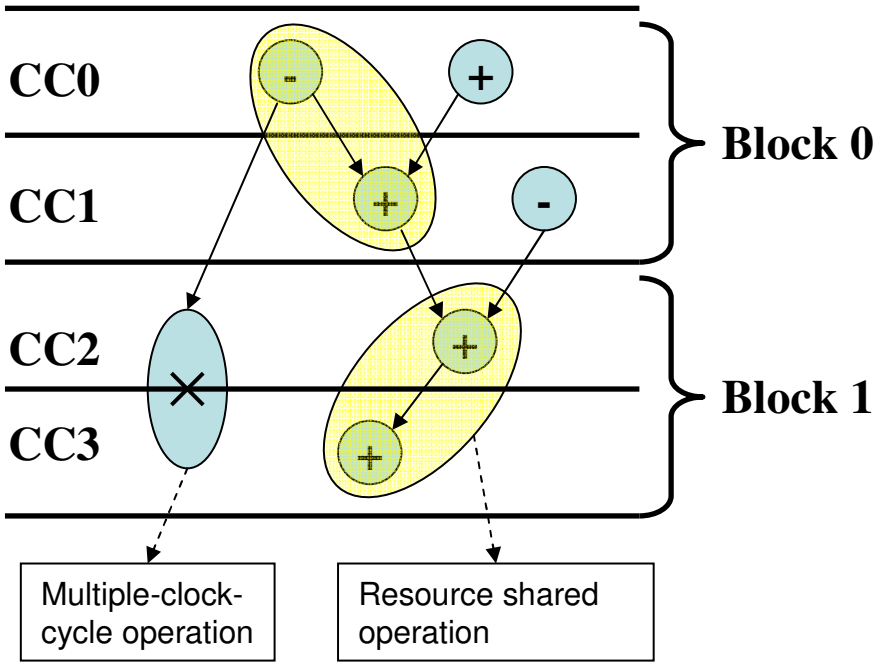


Fig. 4.1. Yield computation for a synthesized DFG. The multiplication operation is bound to a two-clock-cycle module and two additions share the same module.

performance yield of the entire DFG as

$$Yield_{delay} = \prod_{i=1}^{M} Yield_{delay}(b_i) \tag{4.8}$$

where $Yield_{delay}(b_i)$ is the yield value of $block\ i$ and it can be computed as $Prob(Tmax\_block_i \leq T\_clock|constraints)$, and M is the total number of blocks in the DFG. Thus, assuming that an operation in block j is rebound to a new module, the performance yield gradient of the module change can be computed as

$$\Delta Yield_{delay} = \prod_{i=1, i\neq j}^{M} Yield_{delay}(b_i) \times \Delta Yield_{delay}(b_j) \tag{4.9}$$

Thus the yield gradient computation for the entire DFG is reduced to the yield gradient computation for a single block in the DFG.

4.3.6 Power Yield Gradient Computation

The statistical power computation is performed by summing the power dissipation of each module instance in the synthesized DFG as described in Section 4.3.3. To perform power yield gradient analysis for an operation rebinding, we first perform statistical power analysis of the DFG after the rebinding:

$$P^{new}_{DFG} = P^{old}_{DFG} - P^{old}_{opt_k} + P^{new}_{opt_k} \tag{4.10}$$

where P^{new} and P^{old} refer to the power dissipation distribution after rebinding and before rebinding, respectively; P_{DFG} and P_{opt_k} denote the total power dissipation of synthesized DFG

and the power of module instance bound to operation k, respectively. With the distributions of the power dissipation before rebinding and after rebinding determined, the power yield gradient can be computed as

$$\Delta Yield = Yield(P^{new}_{DFG}) - Yield(P^{old}_{DFG}) \tag{4.11}$$

where $Yield(P)$ is computed as the probability of P less than the power limit.

4.4 Process Variation Aware Resource Sharing and Binding

In this section, we propose an *efficient* variation-aware resource sharing and binding algorithm in behavioral synthesis to take into account the performance variations for functional units. The performance yield, which is defined as the probability that the hardware meets the target performance constraints, is used to evaluate the synthesis result; and an efficient metric called *statistical performance improvement*, is used to guide resource sharing and binding. The proposed algorithm is intergraded into a synthesis framework from behavioral descriptions to RTL netlist. The effectiveness of the proposed algorithm is demonstrated with a set of industrial benchmark designs consisting of blocks commonly used in wireless and image processing applications. The experimental results show that our method achieves average 33% area reduction over traditional methods based on worst-case delay analysis with average 10% run time overhead.

4.4.1 Preliminaries and Problem Formulation

- **High-level Synthesis**

In HLS, each operation (such as addition and multiplication) in CDFG is scheduled to one or more cycles (or control steps). Each control step corresponds to a time interval equal to the clock period. Each operation may be performed by more than one *compatible* resource type from the resource library. For example, the addition operation can be performed by either a ripple-carry adder or a carry look-ahead adder, which have different delay and area parameters. *Resource binding* decides the type of functional units to perform the operation in CDFG. *Resource sharing* is that the same resource (functional units or registers) can be shared to perform multiple operations or store more than one variable. Traditionally, high-level synthesis is performed under design constraints, which includes resource constraints, and performance constraints: the *resource constraints* require that the operations are performed with only a limited number of resources available; the *performance constraints* require that the operations in the CDFG to finish the execution in a number of clock cycles (*latency constraints*) with a particular clock rate (*clock cycle time(CCT) constraints*).

- **The Impact of Process Variation**

Traditionally, worst-case delay parameters for the resource are used to facilitate the resource sharing and binding. However, it is becoming inappropriate as larger variability is encountered in new process technologies. A recent publication [11] reports that the delay variations range from 12% to 27%, for 11 different types of 16-bit adders with different circuit architectures and logic evaluation styles. Some adders may run faster but with large variations (for example, Kogge Stone Passgate adder has -27% to +27% of 3σ delay variation), and some adders may run slower but more resistant to variations (such as Carry Select Static).

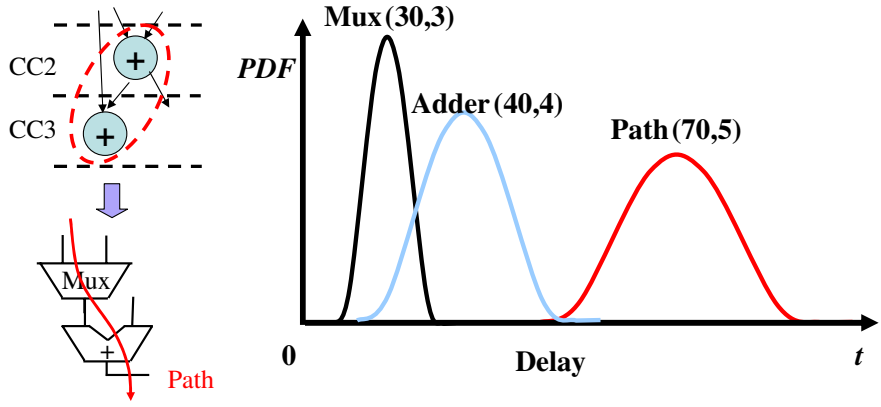


Fig. 4.2. An example of resource sharing for two addition operations and the comparison of worst-case execution time (WCET) based and statistical analysis based approaches. Two addition operations are mapped to an adder, and an additional multiplexer is required. The delay distributions of the adder and the multiplexer are shown as PDF (probability distribution function). The delay distribution of the critical path after resource sharing is also shown as PDF.

Due to large variations in delay, the existing deterministic worst-case design methodologies in HLS may result in unexpected performance discrepancy or a pessimistic performance estimation, or may end up using excess resources to guarantee design constraints, due to overly conservative design approaches. We illustrate this with an example of *resource sharing* shown in Fig. 4.2. Assume that the delay of an adder ($Dadd$) and a multiplexer ($Dmux$) have independent Gaussian distribution $N(\mu, \sigma)$, and the delay distributions of these two function units are given, $Dadd = (40ps, 4ps)$ and $Dmux = (30ps, 3ps)$, and the clock cycle time is $87ps$. In conventional worst-case analysis, the worst-case execution time (WCET) is calculated as $\mu + 3\sigma$. Assuming that we have two compatible addition operations, which can share the same adder, the worst-case timing analysis for the path delay after resource sharing is $D_{path\_WCET} = \mu_{Dadd} + \mu_{Dmux} + 3(\sigma_{Dadd} + \sigma_{Dmux})$, which is $91ps$. Consequently, the worst-case analysis prevents this resource sharing, because the path delay violates the

clock cycle time constraint. However, based on the statistical information, the delay after resource sharing D_{path_ST} follows ($N(\mu_{Dadd} + \mu_{Dmux}, \sigma^2_{Dadd} + \sigma^2_{Dmux})$), and the 3σ delay of D_{path_ST} is $85ps$. Thus, the statistical analysis allows the resource sharing to reduce area cost. Therefore, simply adding the WCET of two function units can result in a pessimistic estimation of the total delay, and may end up using excess resources to guarantee performance constraints.

- **Performance Yield as an Effective Evaluation Metric**

With large delay variations, it is unrealistic to guarantee 100% of the fabricated design to meet the performance constraint. A metric, called *parametric yield*, is introduced to bring the process-variation awareness to the high-level synthesis flow. The parametric yield is defined as the probability of the HLS resultant hardware meeting a specified constraint $Yield = P(Y \leq Y\_max)$, where Y represents performance. The *performance yield* is defined as the probability of the synthesis results meeting the clock cycle time constraints under the latency constraints and resource constraints.

Traditionally, the performance constraints is evaluated with the critical path delay. Under large process variations, the critical path delay is not a single value and it becomes a distribution. Thus, the yield metric is used to evaluate the synthesized the results. Fig. 4.3 demonstrates the effectiveness of the performance yield metric even for a simple example. Assume that we have two synthesized results with the critical path delay distribution of D1(t) and D2(t), respectively. When clock cycle time is set to T1, the synthesis result with D1(t) is better than that with D2(t) in terms of performance yield. However, when clock cycle time is set to T2, the synthesis result with D2(t) is better than that with D1(t). However, if we use worst-case delay to evaluate the results, we always choose the synthesis result with D1(t).

The detailed performance yield analysis for a synthesized DFG is described in Section 4.3 and 4.4.

- **Statistical Performance Improvement as an Effective Metric to Guide Optimization**

In this paper, we introduce two concepts, the *statistical path delay improvement* and the *criticality* to effectively guide the optimization in resource sharing and binding. The statistical path delay represents the magnitude of the path delay based on the statistical analysis. This statistical path delay avoids the pessimistic nature of the worst-case path delay based on the worst-case analysis. The statistical path delay improvement is the difference between the statistical path delay before and after resource sharing or binding. The criticality was first introduced in gate level statistical timing analysis [98]. In this work, we extend this concept to high-level synthesis. The criticality is defined as the probability of the operation being on the critical path. This metric represents the characteristics of the path delay distribution.

These two concepts combined together are able to capture the performance yield improvement due to resource sharing or binding. We illustrate this through an example shown in Fig. 4.4. Four cases are shown with different statistical path delay improvements and values of the criticality. For case 1 and case 4, it is obvious that either the path delay improvement or the criticality can represent the performance yield improvement. In case 2 and case 3, one metric, i.e. either the path delay improvement or the criticality is not sufficient to represent the performance yield improvement. In case 2, the statistical path improvement is large, and there is no difference between case 2 and case 4 if we only use the statistical path improvement as a metric. Similarly, if we use the criticality alone, case 3 would be treated as the same as case 4. Thus, we define the *statistical performance improvement* as the function of the path delay

improvement and the criticality to effectively represent the performance yield improvement. In Section 4.4, the computation method of this metric will be presented.

- **Variation Aware Resource Sharing and Binding** It is obvious that the parametric yield depends on all steps of high-level synthesis: scheduling, resource sharing and binding. These steps are usually interact with each other during high-level synthesis, and influence the final parametric yield calculations. However, the resource sharing and binding have direct impact on the performance yield. Thus, in this work, we focus on the variation aware resource sharing and binding. The variation aware resource sharing and binding is formulated as: *Given a scheduled data flow graph, a set of design constraints, and a resource library with statistical delay characterization, determine the type of function units to perform the operations in CDFG, to reduce the area while satisfying all the design constraints, which include the performance yield constraint.*

4.4.2 Statistical Performance Improvement Computation for Resource Reallocation and Reassignment

When resource reallocation and reassignment occurs, the distribution of the path delay changes, which include the changes in both the mean and the variance of the path delay. The statistical path delay is calculated as $\mu\_path\_delay + \alpha * \sigma\_path\_delay$, where α can is a weighting factor, which balances the optimization effort on reducing the mean and the spread of the performance distribution. The statistical path delay represents the magnitude of the path delay. This metric eliminates the pessimistic nature of the deterministic worst case path delay by computing the value based on the statistical performance analysis.

However, this metric alone is not sufficient to capture the distribution characteristics of the performance improvement as discussed in Section 4.4.2. Thus, the criticality concept is introduced. The criticality is computed as the probability of the delay distribution of the function units involved in the resource reallocation and rebinding determining the critical path delay.

With the statistical delay improvement and the criticality determined, the statistical performance improvement metric is computed as

$$\Delta st\_perf\_impr = st\_pathdelay\_impr * criticality \tag{4.12}$$

where Δst_perf_impr represents the statistical performance improvement, and $st\_pathdelay\_impr$ represents the statistical path delay improvement.

4.4.3 Gain Function for Resource Reallocation and Reassignment

With the statistical performance improvement metric determined, the gain function for the resource reallocation and reassignment can be computed. The gain function for performance improvement under resource constraints and other constraints is computed as $\Delta st\_perf\_impr/\Delta area\_cost$; the gain function for area cost reduction under performance yield constraint and other constraints is computed as $\Delta area\_cost/\Delta st\_perf\_impr$. The computation of $\Delta area\_cost$ is relatively simple and straightforward. For resource reassignment, the change of area cost is simply the area difference of the function units involved in the reassignment. For resource reallocation, the change of the area cost is the area difference of the function units plus the area cost of the additional multiplexers required.

4.4.4 Variation-aware Resource Sharing and Binding Algorithm

Our variation aware resource sharing and binding algorithm is integrated into a commercial synthesis tool flow. This high-level synthesis framework consists of four steps: first, the description of the algorithm in C++ to be synthesized is translated into a CDFG; next the operations of the CDFG are scheduled with the $constraints$, which include latency constraint, clock cycle time constraint, and other user specified constraints; then resource sharing and binding is performed, finally the RTL netlist is generated. In this work, we augment the third step to be process variation aware. Thus, our method takes a scheduled DFG, $constraints$ (latency constraint, resource constraint, and clock cycle time constraint), and a module $Library$ as inputs, and outputs a synthesized DFG that is either performance optimized or area optimized while satisfied those constraints. Note that the scheduled DFG (in which all operations have been scheduled and bound to function units selected from the resource library) meets the latency constraints in terms of the number of clock cycles.

The optimization algorithm can be configured as *performance optimization* or *area optimization*: 1) For performance optimization, the resource constraints is given as an additional constraint, and *the performance yield is maximized*; 2) For area optimization, the performance yield requirement (e.g., the probability of the synthesis result can run at 500 Mhz should be at least 95%) is given as an additional constraint, and the *area is minimized*.

For the sake of simplicity, we describe the *area optimization* algorithm in Fig. 4.5. Our resource sharing and assignment algorithm consists of two steps: 1) resource sharing; 2) resource binding to minimize area under the performance yield constraint. At the first step, we run the *Optimization* routine by finding the possible resource sharing for the compatible operations; at the

second step, we run the *Optimization* routine to map the operations to the resources to reduce the area cost. Both steps perform the the optimization use $\Delta area\_cost/\Delta performance\_improvement$ as the gain function. Detailed description is shown as follows:

1. At each optimization step, we first generate the possible moves, which could be resource sharing or resource binding. We identify possible moves, and form the *to_move_list* at Line 2; We then choose the resource sharing or binding from that list to minimize the area with least performance overhead, and apply these resource sharing or binding at Line 3. The optimization will be terminated when the performance yield constraint is violated.

2. In function *Generate_multiple_moves*, we compute the gain (i.e. $\Delta area\_cost/\Delta st\_perf\_impr$) for each possible move. The move can be resource sharing or resource binding. For resource sharing as shown in Fig 4.6, we first build a compatibility graph at Line 1. Then we compute the gain of the possible resource sharing as $\Delta area\_cost/\Delta st\_perf\_impr$ from Line 2 to Line 4. For resource binding as shown in Fig. 4.7, we identify the available function units for each operation and compute the gain of the rebinding as $\Delta area\_cost/\Delta st\_perf\_impr$ at Line 2. The possible resource sharing or resource binding are ranked according to the gain and penalty ratio, and are inserted to the *to_move_list* at Line 9.

3. In the function *Apply_multiple_moves*, we select the multiple resource sharing or binding from that list to maximize the yield (Line 7) and apply these resource sharing or binding decisions (Line 8).

4.4.5 Analysis Results

In this section, we present the analysis results and show that our method can effectively reduce the impact of the process variation and minimize the area cost under the performance yield constraint, as compared to the non-statistical method based on traditional worst-case performance analysis.

We implement our variation aware resource sharing and binding algorithm in C++ and integrated into a high level synthesis frame work from behavioral level description to RTL netlist. We conduct the experiments on twelve industrial design examples commonly used in wireless and image processing applications. The first ten designs are blocks from the IEEE 802.16 WiMAX standard: 1) Convolutional Coder (CC), 2) Deinterleaver (Deint), 3) Derandomizer (Derand) 4) Hamming windowing and 256-point FFT (RX-FFT), 5) 256-point IFFT and OFDM cyclic prefix insertion (IFFT-CPI), 6) QAM Mapper (QAMM), 7) QAM Slicer (QAMS), 8) Randomizer (Rand), 9) Reed Solomon encoder (RSEnc) and 10) Viterbi decoder (VitDec). The remaining two blocks implement an 8x8 DCT (SDCT) and an 8-tap FIR filter (SFIR). The set of benchmarks are part of Catapult C Synthesis [1] a commercial high-level synthesis tool. Since our focus was resource sharing and binding, we leveraged the tool for the remaining synthesis steps such as allocation/scheduling, register sharing, and datapath/controller generation. The sharing and binding that we use as reference for the comparisons are the ones performed by the tool. In this paper, we show the *area optimization* oriented result: i.e., the performance yield requirement is given as a constraint, and the area is minimized.

To demonstrate the effectiveness of our method, we compare our high-level synthesis results with process-variation aware(PV) resource sharing and binding against those of traditional deterministic methods using worst case (WC) delay models. Table 4.1 shows the results of our method (PV) against those of the worst case (WC) deterministic technique, under a 95% performance yield requirement. In the first column, we show the benchmarks we used in this analysis. From the second column to the third column, we show the absolute area cost results of the process variation aware method (PV) and the deterministic worst case technique (WC), respectively. In the fourth column, we show the relative area improvement of our method over the worst case method. As we can see from Table 4.1, significant area cost reductions are obtained when process variation is taken into account in resource sharing and binding. The results show that the WC based technique is pessimistic with an average 33% area penalty. The worst-case analysis results in a pessimistic estimation, and the consequence is sharing opportunities that are not leveraged to guarantee performance constraints. Of course, if we tighten the performance yield constraint, we may obtain less area reduction. For example, Table 4.2 shows that for the same 99% performance yield constraint, the area cost is slight increased compared to the results for 95% performance yield constraint (Table 4.1).

Table 4.3 shows the runtime overhead of our method (PV) over traditional worst case method (WC) for different benchmarks. From the second column to the third column, we show the runtime of the process variation aware method (PV) and the deterministic worst case technique (WC), respectively. In the fifth column, we show the runtime overhead of our method over worst case method. The maximum runtime overhead of the proposed method 21%. The average runtime overhead is 10% compared to the the traditional deterministic method.

Table 4.1. Area Reduction under 95% Performance Yield Constraint

Name	WC	PV	(PV-WC)/WC
CC	3731.675	5023.09	35%
Deint	6411.245	7514.372	17%
Derand	4636.388	5336.757	15%
RX-FFT	80390.443	101822.946	27%
IFFT-CPI	108258.755	144237.118	33%
QAMM	2856.15	7108.83	149%
QAMS	6515.371	6570.33	1%
Rand	6926.769	8102.694	17%
RSEnc	7992.56	9072.529	14%
SDCT	83482.4	95031.681	14%
SFIR	5402.935	6178.34	14%
VitDec	108880.481	172757.682	59%
Average			33%

Table 4.2. Area reduction under 99% Performance Yield Constraint

Name	WC	PV	(PV-WC)/WC
CC	3795.851	5023.09	32%
Deint	6660.507	7514.372	13%
Derand	4636.388	5336.757	15%
RX-FFT	83510.638	101822.946	22%
IFFT-CPI	124925.961	144237.118	15%
QAMM	2868.197	7108.83	148%
QAMS	6515.371	6570.33	1%
Rand	6939.646	8102.694	17%
RSEnc	8029.941	9072.529	13%
SDCT	83482.4	95031.681	14%
SFIR	5416.867	6178.34	14%
VitDec	110481.6	172757.682	56%
Average			30%

Table 4.3. Run Time Overhead of Variation Aware Method

Name	WC(s)	PV(s)	(PV-WC)/WC
CC	25	29	16%
Deint	11	12	9%
Derand	51	51	0%
RX-FFT	77	87	13%
IFFT-CPI	90	100	11%
QAMM	11	12	9%
Rand	41	45	10%
RSEnc	50	53	6%
SDCT	92	101	10%
SFIR	14	14	0%
VitDec	738	891	21%
Average			10%

4.5 Module Selection with Design-Time Optimization and Post-Silicon Tuning

The previous Section described design time approaches to maximize the parametric yield. In addition to *design-time* variation-aware optimization, *post silicon tuning* techniques (such as adaptive body biasing (ABB)) can be used to further improve the parametric yield. In this Section, we propose a module selection algorithm that combines design-time optimization with post-silicon tuning (using ABB) to maximize design yield. A variation-aware module selection algorithm based on efficient performance and power yield gradient computation is developed. The post silicon optimization is formulated as an efficient sequential conic program to determine the optimal body bias distribution, which in turn affects design-time module selection. The experiment results show that significant yield can be achieved compared to traditional worst-case driven module selection technique. To the best of our knowledge, this is the first variability-driven high level synthesis technique that considers post-silicon tuning during design time optimization.

4.5.1 Preliminaries and Problem Formulation

- **The Influence of Process Variation on HLS**

In HLS, each operation (such as addition and multiplication) in a control data flow graph (CDFG) is scheduled to one or more cycles (or control steps). Each control step corresponds to a time interval equal to the clock period. Each operation may be performed by more than one *compatible* resource type from the resource library. For example, the addition operation can be performed by either a ripple-carry adder or a carry look-ahead adder, which have different delay, power, and area parameters. *Module selection* decides the type of functional units to perform the operations in the CDFG. The same resource (functional units or registers) can be shared to perform multiple operations or store more than one variable. Traditionally, high-level synthesis is performed under resource constraints and performance constraints. Resource constraints require that the operations are performed with only a limited number of resources available; performance constraints require that the operations in the CDFG finish execution in a number of clock cycles with a particular clock rate (clock cycle time). Note that in this paper, we focus on *data-flow intensive* applications, in which most of the computations performed in the design are arithmetic operations (such as addition and multiplication), even though the approach could be easily extended for *control-flow intensive* applications, which contain significant control-flow constructs such as nested loops and conditionals.

Traditionally, worst-case delay/power parameters for the resource are used to facilitate the module selection. However, it is becoming inappropriate as larger variability is encountered in the new process technologies. For example, Fig. 4.8 shows the delay variations (depicted as normalized sigma/mean) for 11 different type of 16-bit adders that span a range of circuit

architecture and logic evaluation styles, all of which are implemented in IBM Cu-08 (90nm) technology [11].

Due to the large variation in delay and power, the existing deterministic worst-case design methodologies in HLS may result in unexpected performance discrepancy or a pessimistic performance/power estimation, or may end up using excess resources to guarantee design constraints, due to overly conservative design approaches.

Furthermore, worst-case analysis without taking the probabilistic information into account can also result in a pessimistic estimation. For example, assume that the delay of an adder ($X = D_{add}$) and a multiplier ($Y = D_{mul}$) have independent Gaussian distribution $N(\mu, \sigma^2)$. In conventional worst-case analysis, the worst-case execution time (WCET) is calculated as $\mu + 3\sigma$ (3σ delay). Based on the statistical information, $X + Y$ follows ($N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2)$), and the WCET for $X + Y$ is $WCET_{X+Y} = \mu_X + \mu_Y + 3\sqrt{\sigma_X^2 + \sigma_Y^2}$, which is smaller than the sum of the WCET of each function unit ($WCET_X + WCET_Y = \mu_X + \mu_Y + 3(\sigma_x + \sigma_y)$). Therefore, simply adding the WCET of two function units can result in a pessimistic estimation of the total delay, and may end up using excess resources to guarantee performance constraints.

- **Variation-aware Module Selection**

Similar to gate-level optimization, we bring the process-variation awareness to the high-level synthesis flow with the *parametric yield* concept. The *performance yield* is defined as the probability of the synthesis results meeting the clock cycle time constraints under the latency constraints and resource constraints. The *power yield* is defined as the probability that the total power of the synthesis result is less than the power limit under latency and resource constraints. The performance and power yield analysis methods are described in Section 4.3.

It is obvious that the parametric yield of the HLS resultant hardware depends on all steps of high-level synthesis: scheduling, module selection, resource sharing, and clock selection. These steps are usually interacted with each other during high-level synthesis, and influence the final parametric yield calculations. However, due to the space limitation, this paper will focus on the module selection problem, assuming other synthesis tasks have been performed. The variation aware module selection is formulated as: *Given a scheduled data flow graph, with the latency and resource constraints, and a resource library with statistical delay and power characterization, determine the type of function units to perform the operations in CDFG, to maximize the power yield subject to performance yield constraints.*

In process variation aware module selection, we take into account the distributions of the delay and the power for each resource type in the library. Fig. 4.9 shows the complexity of module selection problem even for a simple example, without considering power variation. The example also illustrates the difference between conventional worst-case based module selection and variation-aware module selection. Assume that the synthesis result is a single adder, and there are two types of adders with the delay distribution available in the resource library. In conventional module selection, the worst-case execution time analysis shows that *adder 1* is faster than *adder 2*. When the clock cycle time (CCT) constraint is large (e.g., $CCT = T1$), both adders meet timing constraint and the one with smaller area (adder 2) is selected; When we decrease the the CCT, WCET-based module selection has only one choice (adder 1) or no solution (when both adders' $WCET$ are larger than CCT), while performance yield based approach may select either adder 1 or adder 2, depending on the performance yield (*i.e.*, the probability of meeting the CCT constraint, which could be smaller than 100%).

The most interesting observation from Fig. 4.9 is that, the performance-yield based selection could be counter-intuitive. For example, a heuristic of using the product of sigma and mean ($\sigma \times \mu$) was proposed [40] to help module selection; however, Fig. 4.9(a) (when $CCT = T3$ and $CCT = T4$) shows that the heuristic may not be appropriate. Another example is illustrated in Fig. 4.9(b): adder 1 has smaller mean (μ) and smaller variation (σ), which means it is faster and resistant to delay variation. Therefore, intuitively, it should be a better choice than adder 2. However, if $CCT = T5$, adder 2 is actually a better choice since it has higher performance yield. Note that the example in Fig. 4.9 does not take into account power variation and is only for a single adder module selection.

- **Adaptive Body Biasing**

Post silicon tuning techniques can be applied to the fabricated dies after the fabrication. Adaptive Body Biasing (ABB) [94, 53, 64] is an effective technique to reduce the impact of the process variations by controlling the threshold voltage. With bidirectional adaptive body bias, the applied voltage can either raise the threshold voltage of the die (i.e., Reverse Body Biasing (RBB)), to reduce the leakage power at the expense of slowing down circuits, or lower the threshold voltage (i.e., Forward Body Biasing (FBB)), to increase the clock frequency at the expense of higher leakage power. ABB techniques can effectively tighten distribution of the performance and power, thus, the yield loss due to process variation can be minimized. For example, Figure 4.10 shows that by applying ABB techniques, the delay distribution can be adjusted and the spreading is narrowed. Note that the body bias V_{SB} for each individual die is different, and therefore V_{SB} is a probability distribution derived from the probability distribution of performance/power.

4.5.2 Function Unit Delay and Power Modeling with Adaptive Body Biasing

In this section, the piecewise linearized delay model for function units is first introduced, and the exponential power model of function units is then presented. All these models are simple extension from the model presented in previous section 4.3 [45] [90].

In the delay modeling, the delay of the function unit is expressed in terms of the gate length (l), the threshold voltage (V_{th}) and the body bias voltage (V_{SB}). Piecewise linear approximation of the delay has been widely used in the gate level timing analysis. Thus, the delay of the function unit can also be expressed in a piecewise linear function. Suppose ΔV_{th} represents the deviation of the threshold voltage, Δl represents the deviation of the gate length, and V_{SB} represents the applied body biasing, the delay of a function unit, T_i, is expressed as:

$$T_i = a0_i + a1_i \Delta V_{th} + a2_i \Delta l + a3_i V_{SB} \tag{4.13}$$

where $a0_i$ is the nominal delay computed at the nominal values of the process parameters without body biasing. $a1_i$, $a2_i$ and $a3_i$ represent the sensitivity to the deviation of threshold voltage and gate length, and applied body bias, respectively.

The power consumption of a function unit consists of dynamic power and leakage power. The dynamic power is relatively immune to process variation, while the leakage power is affected by process variation greatly, and it becomes a dominant factor in total power consumption as technology scales to nanometer region [90]. Our statistical leakage power model is based on the gate level model and the rms error of this gate level model is around 8% [90]. In this approach, the leakage power of each logic gate is expressed as a lognormal random variable in a canonical form, and the leakage power dissipation of a function unit, which consists of many gates, can

be computed as the sum of these random variables. This sum can be accurately approximated as a lognormal random variable using an extension of Wilkinson's method [90]. Consequently, the leakage power dissipation of a function unit can also be expressed as a lognormal random variable in a canonical form. Therefore, the leakage power of a function unit can be expressed as

$$P_i = exp(b0_i + b1_i \Delta V_{th} + b2_i \Delta l + b3_i V_{SB}) \tag{4.14}$$

where $exp(b_0)$ is the nominal leakage power computed at the nominal values of the process parameters. b_i are the sensitivities to their corresponding sources of the deviation and the bias voltage.

4.5.3 Design Time Variation-aware Module Selection Algorithm

In this subsection, the author first introduces a design time module selection algorithm based on the fast yield computation method. The author then presents the algorithm on how to decide optimal body biasing, and describe the module selection strategy of joint design time optimization and post silicon tuning in the following subsections.

Our module selection takes an initial scheduled DFG ($ISDFG$), $constraints$ (latency constraints, resource constraints, CCT constraints, and power constraints), and a module $Library$ as inputs, and output a synthesized DFG that is power optimized while satisfying performance constraints. In the initial scheduled DFG, the operations are bound to the fastest module in the module library, which meet the latency constraints in terms of the number of clock cycles. The iterative module selection algorithm consists of two steps: 1) performance yield maximization; 2) power yield improvement under the performance yield constraint.

Note that the optimization algorithm shown in Fig. 4.11 can be configured as performance optimization or power optimization depending on the *gain* function, and the gain is the change in total yield that results from the move (module selection) : for performance optimization, the gain is the performance yield gain, $\Delta Yield_{delay}$; for power optimization under the performance yield constraint (e.g., the probability of the synthesis result can run at 200 Mhz should be at least 90%), the gain is $\alpha * \Delta Yield_{delay} + \Delta Yield_{power}$, where α is weight factor.

Similar to [82], our iterative variation aware module selection algorithm is based on a variable depth search method. The algorithm starts with an initial scheduled DFG. It identifies the move with the maximum gain, and inserts that move to to_move_list (Line 9-12). The algorithm continues to identify the moves that give max gain until the maximum number of moves is reached (Line 8-12). After all these moves identified, we find a sequence of moves, which gives the best gain (Line 3). In other words, the algorithm find k consecutive moves, which give best $G_k = \sum_{i=1}^{i=k} g_i$. Note that the gain of some moves might be negative. The algorithm accepts these moves when $G_k > 0$. This makes our algorithm capable of escaping the local minimum. The k consecutive moves are then committed, and the performance yield or power yield are evaluated (Line 5-6). The iteration continues until the yield improvement is less than a preset small value or the delay/resource constraint is violated(Line 1).

4.5.4 Post-silicon Tuning with ABB

Once the module selection is decided at the design time, adaptive body biasing (ABB), which is a post-silicon tuning technique, can be applied to further reduce the parametric yield loss. ABB body biasing can be applied at the chip level (i.e., all the modules on a die will have a single body biasing V_{SB}), and the whole chip is either applied FBB or RBB; ABB can also be

applied at the module level (i.e., each module can have its own V_{SB}), such that each module can be applied FBB or RBB, achieving a finer granularity of tuning. After fabrication, the optimal body biasing for each die can be determined by speed/power binning. However, at design time, the optimal body biasing V_{SB} (either chip-level or module-level) is a probability distribution, depending on the statistical delay/power distribution of the chip.

In this section, we describe how to decide the optimal body biasing for a particular module selection decision, such that the power yield is maximized under the performance constraints (i.e., meeting a particular performance yield requirement). The optimization can be formulated as a sequential of a second order conic program problem [10]. To obtain the optimal V_{SB} to compensate the variability caused by the process variations, we formulate the optimization problem as follows:

$$minimize\ P_{sttot} \tag{4.15}$$

$$subject\ to\ P(Tmax < T_{clock}|constraints) > \alpha \tag{4.16}$$

where the objective function P_{sttot} is defined as $\bar{P}_{tot} + \beta * \sigma_{P_{tot}}$. $\bar{P}_{tot}$ and $\sigma_{P_{totl}}$ are the nominal and the variance of the total power, respectively. β is a weighting factor, which balances the optimization effort on reducing the mean and the spread of the power distribution. α is the required performance yield. The *constraints* represent the latency constraints and the resource constraints. $Tmax = max(Tmax_i), \forall i \in [1, N]$ as shown in equation (4.7).

In the optimization problem, the applied body biasing V_{SB}, is the optimization variable to be determined. Applied body bias is used to compensate the process variation. Thus, V_{SB} is a function of the random variables. In Mani et al.'s work [64], V_{SB} is set to be an affine function

of the parameter variations. To simplify the optimization problem, which can be solved using a conic programming, we set

$$V_{SB} = s^T Y \tag{4.17}$$

and the vector s is the compensation vector. Thus the vector s is the optimization vector to be determined. Y is the random variable vector used to model process variations, which consists of the random variables Y_i and R_m in equation (4.3).

The constraint function of this optimization problem is the integration of the delay distribution, $Tmax$. This random variable, $Tmax$, is obtained by performing max operation over the timing quantities, T_i. Thus, the constraint functions need to be transformed. First, since the V_{SB} is an affine function of the variation parameters, the delay of a module instance can be expressed as a linear function of optimization variables:

$$T_i = h_i^T s \tag{4.18}$$

Second, to compute $Tmax$, the max operation over the timing quantities T_i is performed. The max operation is complex and involves the integration of the exponential function. Thus, the max operation has to be approximated. According to Clark's work [20], the max operation can be computed as a linear function under specific constraints as follows:

Assuming that $Tmax = max(Tmax_0, ..., Tmax_n)$, we then have

$$Tmax = \Sigma_{i=1}^{n} C_i Tmax_i \tag{4.19}$$

where C_i is the probability of the timing quantity, $Tmax_i$, determining the $Tmax$. To make this approximation valid, a new constraint is introduced to limit the change of body bias V_{SB} to be less than a small value, ϵ. In this paper, the change of the body biasing, ΔV_{SB}, is set to be 0.01. From equation (4.18) and (4.19), we can express $Tmax$ as a linear combination of the optimization vector s,that is

$$Tmax = b^T s \tag{4.20}$$

Given that $Tmax$ is a gaussian random variable, the constraint, $P(Tmax < T_{limit}) < \alpha$, can be transformed to a quadratic function [10]:

$$\bar{T}max + \phi^{-1}(\alpha)(\sigma_{Tmax}) \leq T_{limit} \tag{4.21}$$

where $\bar{T}max$ and σ_{Tmax} are the mean and variance of $Tmax$. Thus, we can express the constraint function in terms of optimization vector s,

$$\bar{b}^T s + \phi^{-1}(\alpha)(s^T \Sigma s)^{1/2} \leq T_{limit} \tag{4.22}$$

The objective function, P_{sttot}, takes into account the mean and variation of the power. However, the objective function is an exponential function of random variables. To efficiently solve the optimization problem using second order conic programming, the objective function is transformed. According to probabilities theory, the mean and variance of the log-normal random variable can be expressed as a exponential function. Consequently, the objective function is a complex nonlinear function of the optimization vector s. Thus, Taylor expansion is used to

obtain its linear approximation. We then have:

$$\bar{P}_{DFG} = P_{DFG}(s_{ini}) + (s - s_{ini})^T \nabla_s(\bar{P}_{DFG}) \tag{4.23}$$

$$\sigma_{P_{DFG}} = \sigma_{P_{DFG}}(s_{ini}) + (s - s_{ini})^T \nabla_s(\sigma_{P_{DFG}}) \tag{4.24}$$

where s_{ini} represents the initial value of the optimization variable vector. To simplify the notation, $a1$ and $a2$ is set to be $\nabla_s(\bar{P}_{DFG})$ and $\nabla_s(\sigma_{P_{DFG}})$ respectively. We then express the objective function as a linear function of the optimization vector s:

$$a1^T s + \beta * a2^T s \tag{4.25}$$

Based on the above transformations in equations (4.25) and (4.22), the optimal body biasing assignment problem can be formulated as a second order conic program:

$$minimize \quad (a1 + \beta * a2)^T s \tag{4.26}$$

$$subject\ to \quad \bar{b}^T s + \phi^{-1}(\alpha)(s^T \Sigma s)^{1/2} \leq T_{limit} \tag{4.27}$$

$$c^T(s - s_{ini}) < \epsilon \tag{4.28}$$

The linear constraint is set to make the approximation of $Tmax$ valid.

Based on this second order conic program formulation, the sequential conic program (SCP) algorithm is shown in Fig. 4.12. In the sequential programming algorithm, the complex optimization problem is transformed to a sequence of the simplified subproblem. In this work,

the subproblem, CP^i, is used as an approximation of the optimal body biasing problem in a range, ϵ. In this work, the subproblem is solved using the above second order conic program. At the end of each iteration, the solution of the subproblem, CP^i, is evaluated for convergence. The iteration exits when the convergence test fails, or the number of maximal tries has been reached.

4.5.5 Joint Optimization Algorithm

Our joint design time module selection and post silicon tuning algorithm takes an initial scheduled DFG, $constraints$ (latency constraint, resource constraint, CCT constraint, and power constraint), and a module $Library$ as inputs, and outputs a synthesized DFG with optimal body bias that is power optimized while satisfying performance constraints. In the initial scheduled DFG, the operations are bound to the fastest module in the module library, which meet the latency constraints in terms of the number of clock cycles.

Our joint optimization algorithm consists of two steps: 1) design time module selection algorithm selects the module instance for the operations in DFG to maximize the power yield under power yield constraints, and the module selection is performed under the body bias determined in previous iteration; 2) the sequential conic program determines the optimal body bias for the module instances in the current iteration to tighten the delay and power distribution to further improve the power yield. The body bias is initially set to be zero. These two steps are iterated until no improvement can be obtained.

4.5.6 Analysis Results

In this section, we present the analysis results and show that our joint design time and post silicon time method can effectively reduce the impact of the process variation and maximize

the parametric yield, as compared to traditional worst-case based design-time module selection method.

We implement our joint design time module selection and post silicon tuning algorithm in C++ and conduct the experiments on six high level synthesis benchmarks: a 16-point symmetric FIR filter (FF), a 16-point elliptic wave filter (EWF), an autoregressive lattice filter (ARF), an algorithm for computing Discrete Cosine Transform (DCT), a differential equation solver (DES), and an IIR filter (IIR). The resource library contains different adders and multipliers, implemented in 90 nm technology, with statistical delay and power distributions. The library characterization is performed based on the results of Monte Carlo analysis in HSPICE. A few experiments are conducted to demonstrate the effectiveness of our algorithm, with the average runtime of algorithm for all benchmarks (on a 2GHz Pentium 4 Linux machine) less than 0.5 sec:

- *Design-time only variation-aware vs. Worst-case deterministic module selection.* First, we compare our variation-aware design time optimization approach (*without considering post-silicon tuning*) to the traditional worst case deterministic module selection method. The power optimization is performed under 90% performance yield constraints. As shown in Fig. 4.14, the reference design time approach has significant yield improvement over the worst case design time approach. An average 34% yield gain is achieved using our reference variation design time approach.

- *Joint design-time and post-silicon tuning Vs. design-time only variation-aware module selection.* We then compare the results of our joint design-time and post-silicon tuning

module selection algorithm (JT) against the variation-aware design-time only module selection method (DT). Table 4.4 shows the results of our method with a single chip-level body bias control (JTS) against those of the design time module selection technique (DT), under a 99% performance yield requirement. From the second column to the third column, we show the absolute power yield results of the joint optimization method (JTS) and the design-time only module selection technique (DT), respectively. In the fourth column, we show the absolute value of the yield improvement of our method over design-time only method. In the fifth column, we show the relative yield improvement of our method over the design time method. As we can see from Table 4.4, significant yield improvement could be obtained if we take into account post-silicon tuning techniques in high level synthesis. The yield results show that joint design time and post silicon optimization can achieve average 38% power yield improvement, compared to design-time only variation-aware module selection. If we relax the power constraint, we may have larger power yield. For example, Table II shows that for the same 99% performance yield constraint, if we relax the power constraint, joint design time and post silicon optimization can achieve average 11% power yield improvement.

Table 4.4. Power Yield Under 99% Performance Yield Constraint

Name	DT	JTS	JTS-DT	(JTS-DT)/DT
AR	47%	86%	39%	83%
DCT	60%	85%	25%	42%
DES	76%	90%	14%	18%
EWF	79%	90%	11%	14%
FF	75%	92%	17%	23%
IIR	58%	85%	27%	47%
Average	66%	88%	22%	38%

- *Module-level post-silicon tuning*. The more aggressive approach is module-level post-silicon tuning (i.e., each module can have its own V_{SB} control). This approach results in a finer granularity tuning, and can effectively address the *intra-die variation*). However, the additional overheads of the body bias generator and control circuitry can adversely affect the gain of this approach. Therefore, it is appropriate to apply post-silicon tuning at multiple-module level instead of single module level.

Table 4.5. Power Yield Under 99% Performance Yield Constraint with Power Constraint Relaxation

Name	DT	JTS	JTS-DT	(JTS-DT)/DT
AR	74%	90%	16%	22%
DCT	80%	89%	9%	11%
DES	88%	93%	5%	6%
EWF	90%	97%	7%	8%
FF	85%	94%	9%	11%
IIR	83%	90%	7%	8%
Average	83%	92%	9%	11%

- *Comparison against previous variation-aware HLS work.*

  We also compare our algorithm against previous variation-aware HLS work proposed by Hung et.al [40]. Their module selection is based on a heuristic of using the product of sigma and mean ($\sigma \times \mu$). However, as we have shown in Section III.B, Fig. 4.5.1, this heuristic may not be appropriate. In addition, their algorithm only considered the area reduction (using smaller number of resource to meet a specific performance yield) without considering power variations.

4.6 Summary

Process variation in deep sub-micron (DSM) VLSI design has become a major challenge for designers. Dealing with delay/power variations during high level synthesis is still in its infancy. Performance/power yield, which is defined as the probability of the synthesized hardware meeting the performance/power constraints, can be used to guide high level synthesis.

In this Chapter, we formulate the performance yield constraint resource sharing and binding problem and propose an efficient algorithm to solve it. We introduce an efficient metric called *statistical performance improvement* to facilitate the design space exploration in the resource sharing and binding. Based on this metric, the performance yield aware resource sharing and binding algorithm is developed and integrated into a commercial tool flow. Simulation results show that significant area cost reduction can be obtained with our variation-aware resource sharing and binding method with small amount of runtime overhead. In addition to the design time approach, the proposed research demonstrates that the yield can be effectively improved by combining both *design-time* variation-aware optimization and *post silicon tuning* techniques (adaptive body biasing (ABB)) during the module selection step in high level synthesis. The experiment results show that significant yield can be achieved compared to traditional worse-case driven module selection technique. To the best of our knowledge, this is the first variability-driven high level synthesis technique that considers post-silicon tuning during design time optimization.

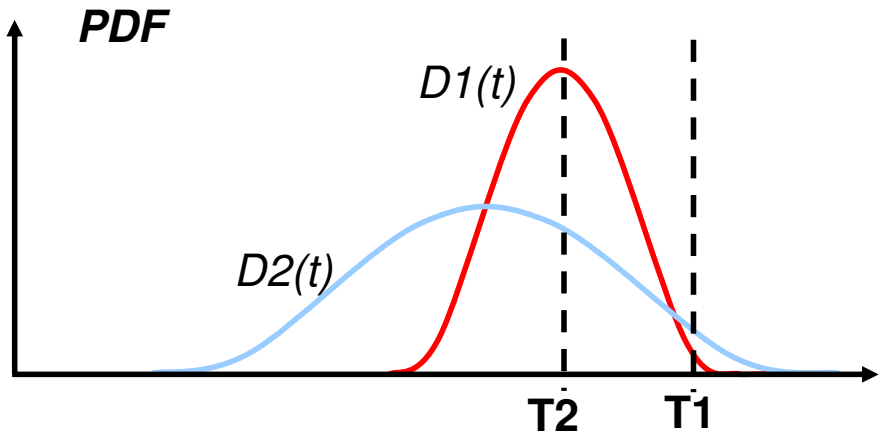


Fig. 4.3. An example illustrates the effectiveness of the performance yield metric. The critical path delay distributions of two synthesis resultant hardware are shown in PDF. When clock cycle time is set to T1, the synthesis result with D1(t) is better than that with D2(t) in terms of performance yield. However, when clock cycle time is set to T2, the synthesis result with D2(t) is better than that with D1(t). However, if we use worst-case delay to evaluate the results, we always choose the synthesis result with D1(t).

Cases	Path Delay Improvement	Criticality	Yield Improvement
Case1	small	small	small
Case2	large	small	medium
Case3	small	large	medium
Case4	large	large	large

Fig. 4.4. Accurately evaluating the performance improvement requires to consider both the path delay improvement and the criticality. Four cases are shown with different path delay improvements and values of the criticality. For case 1 and case 4, it is obvious that the path delay improvement and the criticality can represent the performance yield improvement. In case 2 and case 3, one metric, i.e. either the path delay improvement or the criticality, is not sufficient to represent the performance yield improvement.

```
Optimization (synthesizedDFG,constraints,Library){
1.  While (meet constraints){
2.    Generate_multiple_moves generates the to_move_list list;
3.    Apply_mutliple_moves applies multiple moves in to_move_list;
4.  }}

Apply_multiple_moves (to_move_list, constraints){
5.  While (to_move_list is not empty and meet constraints){
6.    For each resource sharing or rebinding in that list{
7.      Insert new move to to_update_list;
8.      Update Yield with new move in to_update_list;
9.  }}}
```

Fig. 4.5. The pseudo code of variation aware optimization of DFG

```
Generate_multiple_moves (syntheiszedDFG,Library, constraints){
1.  Build the compatible graph;
2.  For each edge in that compatible graph{
3.    Compute gain function for the possible resource sharing;
4.  }
5.  Rank the resource sharing in that list according to gain value;
6.  }
```

Fig. 4.6. The pseudo code of generate moves for resource sharing of DFG

```
Generate_multiple_moves (syntheiszedDFG,Library, constraints){
1.  For each operation{
2.    Compute gain function for different possible binding;
3.    }
4.  Rank the binding in that list according to gain value;
5.  }
```

Fig. 4.7. The pseudo code of generate moves for resource binding of DFG

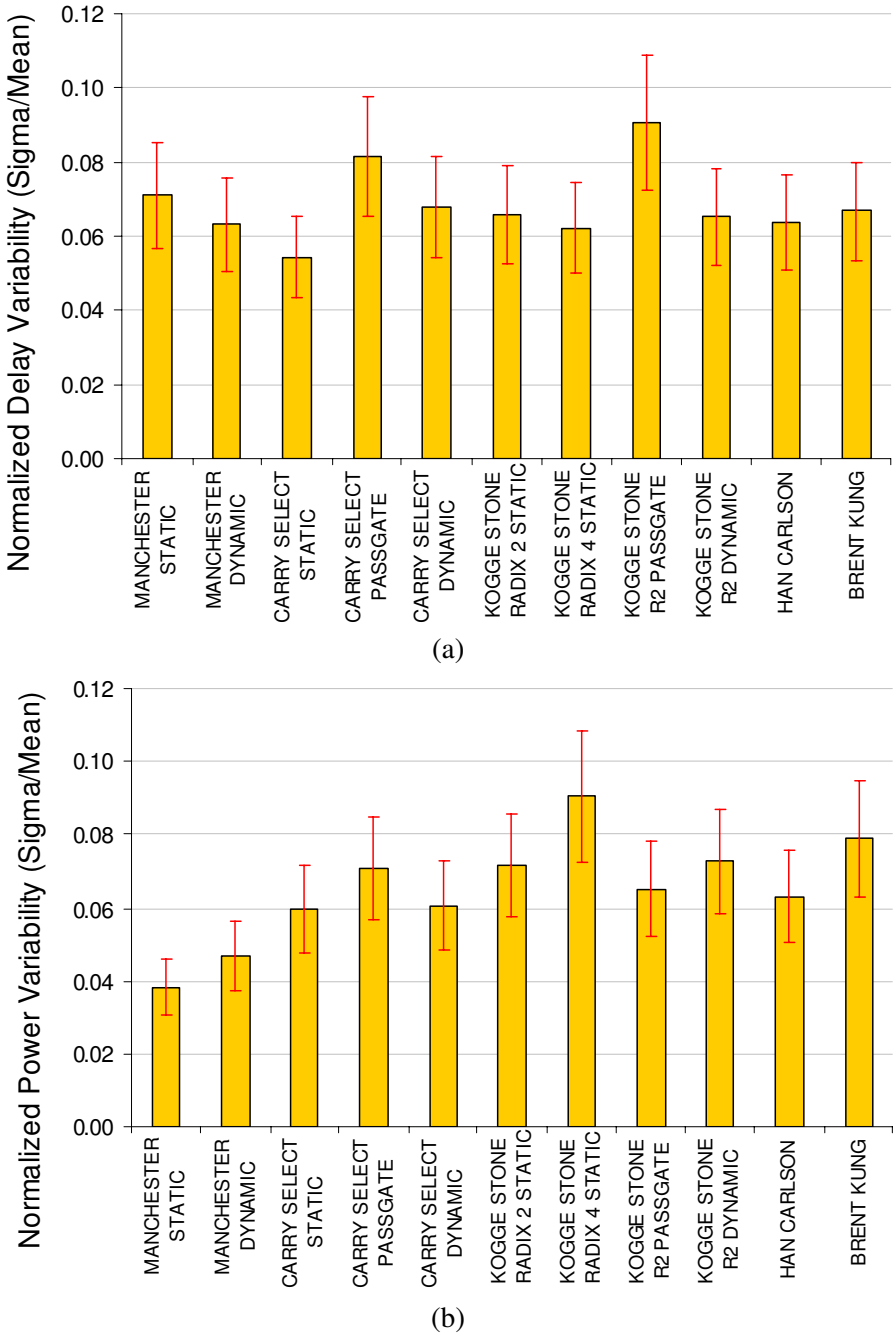
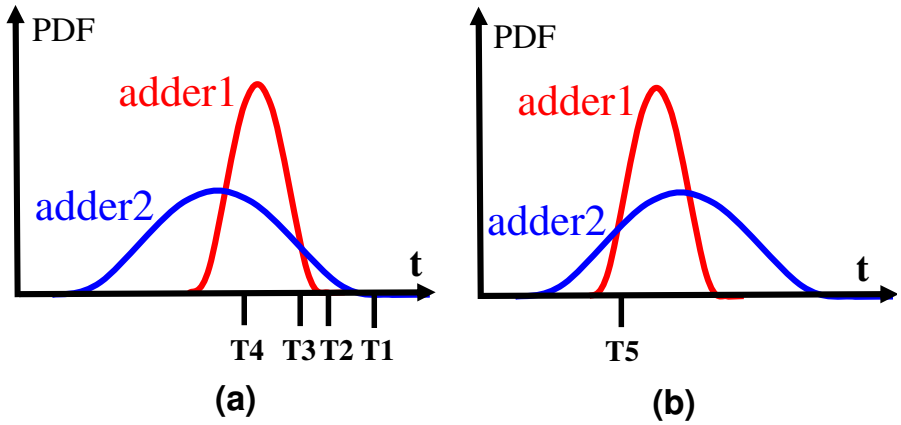


Fig. 4.8. (a) The delay variation (normalized sigma/mean) for 16-bit adders in IBM Cu-08(90nm) technology; (b) The delay variation (normalized sigma/mean) for 16-bit adders in IBM Cu-08(90nm) technology. (Courtesy of K. Bernstein, IBM [11]).



CCT	WCET based	Performance yield based
T1	Adder 2 (WCET $\leq$ CCT, smaller area)	Adder 2 (100% yield, smaller area)
T2	Adder 1 (WCET $\leq$ CCT)	Adder 1 (100% yield)
T3	None	Adder 1 (better yield)
T4	None	Adder 2 (better yield)
T5	None	Adder 2 (better yield)

Fig. 4.9. An example of module selection for an adder and the comparison of worst-case execution time (WCET) based and performance yield based module selection. The delay distribution of two different type of adders are shown as PDF (probability distribution function), and the area of adder 2 is smaller.

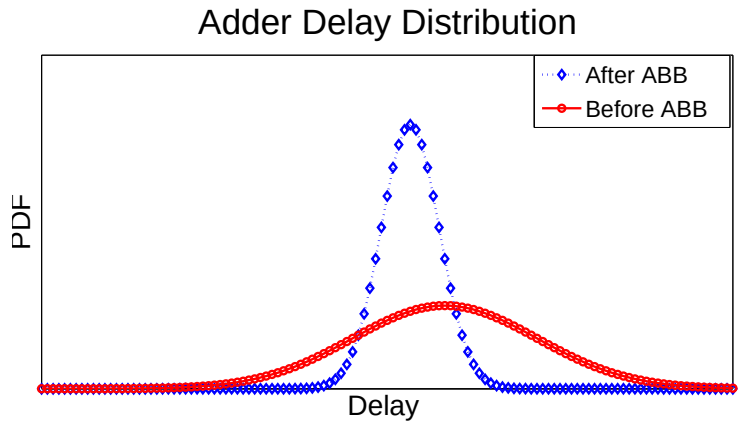


Fig. 4.10. The adder delay distribution can be adjusted by post-silicon ABB techniques

```
Optimization (ISDFG,constraints,Library){
1.  While (ΔYield > ε and meet constraints){
2.    Generate_multiple_moves generates the to_move_list list;
3.    Find k of to_move_list to maximizing the total gain G_k;
4.    If (total gain G_k > 0){
5.      Apply this sequence of moves;
6.      Evaluate the power and performance yield;
7.    }}}

Generate_multiple_moves (ISDFG,Library, constraints){
8.  While (maximum number of moves is not reached) {
9.    For (each possible move in the DFG){
10.     Evaluate the gain of that move;
11.     Save the move and gain to temp_move_list;}
12.   Insert the move with highest gain to to_move_list;
13.   }
```

Fig. 4.11. The pseudo code of variation aware optimization in module selection

```
SCP (ISDFG,constraints,s)
1.  While (convergent){
2.    setup the CP^i(ε)
3.    solve the CP^i(ε)
4.  }
```

Fig. 4.12. The pseudo code of optimal body biasing of DFG

```
JointOpt (ISDFG,constraints,Library)
1.  While (ΔYield > ε and meet constraints){
2.    Design time module selection under current body bias;
3.    Sequential Conic Optimization;
4.  }
```

Fig. 4.13. The pseudo code of variation aware optimization of DFG

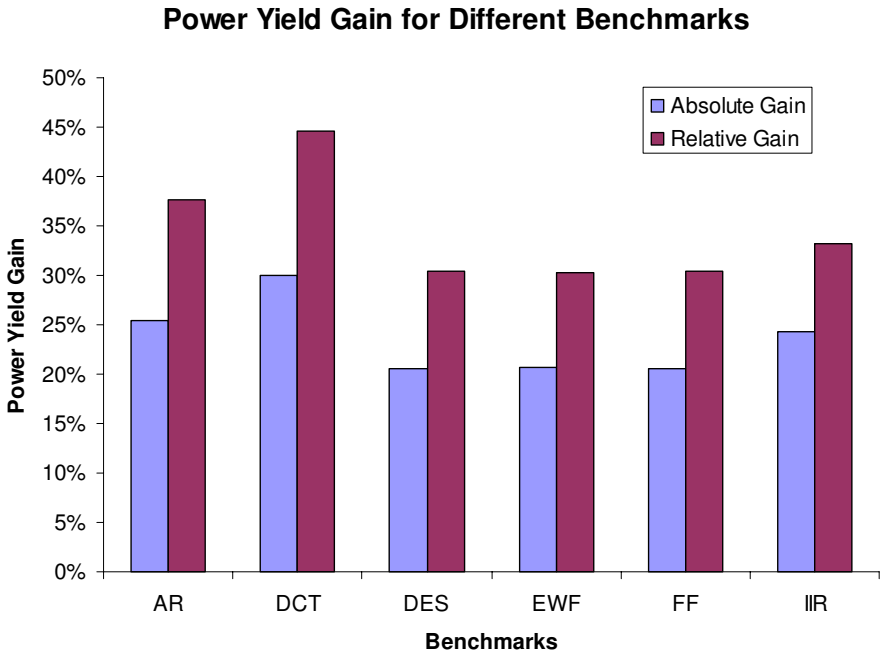


Fig. 4.14. Power yield improvement over worst-case based deterministic module selection, with 90% performance yield constraint.

Chapter 5

Variation-aware Task Allocation and Scheduling for MPSoC

Chapter 3 and 4 described the statistical design approaches at gate level and module level. Since the benefits from higher-level optimization often far exceed those obtained through lower-level optimization, it is important to raise the process variation awareness to a higher level. In this Chapter, the author focuses on system level statistical performance analysis and optimization. A variation-aware task and communication mapping for MPSOC (Multiprocessor System-on-Chips) that uses network-on-chip (NoC) communication architecture is developed in order to mitigate the impact of parameter variations. The proposed mapping scheme accounts for variability in both the processing cores and the communication backbone to ensure a complete and accurate model of the entire system. A new design metric, called *performance yield* and defined as the probability of the assigned schedule meeting the predefined performance constraints, is used to guide both the task scheduling and the routing path allocation procedure. An efficient yield computation method for this mapping complements and significantly improves the effectiveness of the proposed variation-aware mapping algorithm. Experimental results show that our variation-aware mapper achieves significant yield improvements. On average, 14% and 39% yield improvements over worst-case and nominal-case deterministic mapper, respectively, can be obtained across the benchmarks by using the proposed variation-aware mapper.

5.1 Introduction and Motivation

Currently, entire systems can be integrated on a single chip die (System-on-Chip, or SoC) [67, 35]. In fact, many embedded systems nowadays are heterogeneous multiprocessors with several different types of Processing Elements (PEs), including customized hardware modules (such as Application-Specific Integrated Circuits, or ASICs), programmable microprocessors, and embedded Field-Programmable Gate Arrays (FPGA), all of which are integrated on a single die to form what is known as a Multiprocessor System-on-Chip (SoC) [44]. Integrating multiprocessor on the same chip creates great challenges in the interconnect design. Technology scaling and reliability concerns, such as crosstalk and electromigration, motivate us to adopt the network-on-chip (NoC) architecture as a scalable approach for interconnect design [21] [9].

The Intellectual Property (IP) re-use approach has been widely advocated as an effective way to improve designer productivity and efficiently utilize the billions of available transistors in complex SoC designs [44]. In this approach, the designers are required to partition the logic functionality into hard and soft modules. By assigning the software functions to appropriate hardware PEs, the designer must then determine if the resulting embedded system can meet the real-time constraints imposed by the design specifications [44]. Design decisions taken at the early stages of the design process are critical in avoiding potentially high-cost alterations in the more advanced phases of the process. Consequently, the designer must conduct early analysis and evaluation to guarantee that the performance and cost targets are met. Early-stage evaluation tools with accurate system analysis capabilities enable the SoC designers to explore various high-level design alternatives that meet the expected targets. Traditionally, task scheduling and communication path allocation based on a worst-case timing analysis is used to evaluate these

possible designs during the early design stages. Hence, the deterministic slack can be used to assess the tightness of the timing constraints and guide the task scheduling and communication allocation in this early exploration process.

However, the challenges in fabricating transistors with diminutive feature sizes in the nanometer regimes have resulted in significant variations in key transistor parameters, such as transistor channel length, gate-oxide thickness, and threshold voltage. Parameter variation across identically designed neighboring transistors is called *within-die variation*, while variation across different identically designed chips is aptly called *inter-die variation*. This manufacturing variability can, in turn, cause significant performance and power deviations from nominal values in identical hardware designs. For example, Intel has shown that process variability can cause up to a 30% variation in chip frequency and up to a 20x variation in chip leakage power for a processor designed in 180 nm technology [13]. As technology scales further, performance variations become even more pronounced. It has been predicted that the major design focus in sub-65nm VLSI design will shift to dealing with variability [13, 12]. Designers have relied on technology scaling [109] to enhance performance. For example, embedded processors such as ARM's Cortex-A8 are already manufactured at the leading-edge 65 nm technology. Xilinx's Virtex-5 family of FPGAs (which include embedded hardware modules such as microprocessors and memory blocks) is also built on an advanced 65 nm technology node. The irreversible momentum toward deep sub-micron process technologies for chip fabrication has brought the ominous concerns of process variation to the forefront.

Designing for the worst case scenario may no longer be a viable solution, especially when the variability encountered in the new process technologies becomes very significant and causes substantial percentage deviations from the nominal values. Increasing cost sensitivity in the

embedded system design methodology makes designing for the worst case infeasible. Further, worst-case analysis without taking the probabilistic nature of the manufactured components into account can also result in an overly pessimistic estimation in terms of performance, as shown in Fig. 1.3. Consequently, design for worst case may end up necessitating the use of excess resources to guarantee real-time constraints. Under process variation, slack is no longer an effective metric, for it is no longer deterministic. Thus, the impact of large process variability requires a shift from deterministic design methodology to statistical design methodology at all levels of the design hierarchy [14].

In this Chapter, variation-aware performance analysis is integrated into the task scheduling and communication path allocation process for efficiently designing SoCs in the presence of unpredictable parameters. Accurate early analysis is very critical, because system-level performance evaluation influences early design decisions that later impact the overall design complexity and cost.

An important contribution of this work is the inclusion of the communication backbone in the variability-aware mapping scheme. Integrating multiprocessor on the same chip creates great challenges in the interconnect design. Technology scaling and reliability concerns, such as crosstalk and electromigration, motivate us to adopt the network-on-chip (NoC) architecture as a scalable approach for interconnect design [21] [9][37]. The proposed algorithm accounts for process variation in both the *processing cores* and the it packet-based communication network. NoC routers are expected to exhibit variations in performance, much like every other system component. Therefore, it is imperative for the analysis model to capture variability in all mission-critical modules of the entire SoC. Given the criticality of the interconnection network in any multicore SoC, our analysis and mapping algorithm include the NoC infrastructure. To the best

of our knowledge, this is the first paper to implement a variability-aware mapping scheme based on analysis of *both* the processing elements and the interconnection network.

In addition to variation-aware analysis, we introduce the new concept of *parametric yield* to accommodate the new reality of non-negligible variability in modern NoC architectures. Traditionally, yield has been viewed as a metric to determine hardware implementations which meet the pre-defined frequency requirements. Manufacturers reject the subset of dies that fail to meet the required performance constraints. Thus, classification based on manufacturing yield is very important from a commercial point of view. In this work, we extend the notion of yield to a higher abstraction level in the design flow: we define the *parametric yield* of the SoC design as the probability of the design meeting the real-time constraints imposed by the underlying system. Subsequently, designs that cannot meet the real-time constraints have to be discarded. Hence, this definition of yield relates directly to the design cost and design effort and is a direct corollary of decisions and choices made throughout the design process. Experimental results clearly indicate that the proposed process variation-aware approach in the early design exploration phase can lead to significant yield gains down the line. Specifically, our variation-aware scheduler can obtain 45% and 34% performance yield improvements over worst-case and nominal-case conventional (i.e. deterministic) schedulers, respectively, across a wide gamut of multiple processor benchmarks.

The contributions in this Chapter distinguish themselves in the following aspects: 1) The author first formulates the process-variation-aware task and communication mapping problem for NoC architectures and propose an efficient variation-aware scheduling algorithm to solve it; the novelty lies in the augmentation of process-variability-induced uncertainties in the mapping model. 2) The author subsequently introduces and employs the notion of *performance yield* in

the dynamic priority computation of the task and communication mapping process. 3) Finally, the author develops a yield computation method for the partially scheduled task graphs.

5.2 Related work in variation aware task allocation and scheduling

Related work pertaining to this paper can be divided into three different categories: traditional task and communication mapping for embedded systems, gate-level statistical analysis and optimization, statistical performance analysis approaches for the embedded system, and probabilistic analysis for real-time embedded systems.

5.2.1 Task allocation and scheduling for embedded systems

There have been extensive studies in the literature on task allocation and scheduling for embedded systems. Precedence-constrained task allocation and scheduling has been proven to be an NP-hard problem [83]; thus, allocation and scheduling algorithms usually use a variety of heuristics to quickly find a sub-optimal solution [86][106]. Recently, several researchers have investigated the task scheduling problem for Dynamic-Voltage-Scaling-enabled (DVS) real-time multi-core embedded systems. Zhang et al. [111] formulated the task scheduling problem combined with voltage selection as an Integer Linear Programming (ILP) problem. Luo et al. [46] proposed a condition-aware DVS task scheduling algorithm, which can handle more complicated conditional task graphs. Liu et al. [62] proposed a constraint-driven model and incorporated it into the power aware scheduling algorithm to minimize the power while meeting the timing constraints. Shang and Jha [85] developed a two-dimensional, multi-rate cycle scheduling algorithm for distributed embedded systems consisting of dynamically reconfigurations FPGAs. A recent

work [37] focused on the communication path allocation for regular NoC architectures and proposed a branch bound algorithm to solve it. Similar to the work [37], Hu et al. [38] proposed a energy aware task and communication scheduling for Network-on-Chip (NoC) architectures under real-time constrains. However, all these approaches are deterministic approaches without considering the manufactural variability.

Recently, Marculescu et al. developed a statistical performance analysis approach for the embedded system [32] [65]. Marculescu et al. [65] performed statistical performance analysis for single and multiple voltage frequency island systems, and compute performance bounds of these system based on the statistical analysis. Garg et al. [32] investigated the impact of process variations on the throughput for multiple voltage frequency island system and proposed a method compute the throughput considering the process variability.

5.2.2 Complementing Existing Probabilistic Real-Time Embedded System Research

The real-time community has recognized that the execution time of a task can vary, and proposed probabilistic analysis for real-time embedded systems [92, 39, 96], where the probability that the system meets its timing constraints is referred to as *feasibility probability* [39]. However, these statistical analyses are for execution time variations caused by software factors (such as data dependency and branch conditions), and hardware variations were not modeled. The actual meanings of *feasibility probability* and *performance yield* are quite different: for example, *feasibility probability = 95%* means that the application can meet the real-time constraints during 95% of the running time; *performance yield = 95%* means that 95% of the fabricated chips can meet the real-time constraints with 100% guarantee, while the other 5% of the chips may not meet the real-time constraints. Therefore, *feasibility probability* is a *time domain* metric and is

suitable for "soft real-time" systems, while *performance yield* is a *physical domain* metric (i.e., percentage of fabricated chips) and it can guarantee that good chips meet hard deadlines.

5.3 Preliminaries

This section lays the foundations upon which our proposed variation-aware task and communication mapping algorithm rests. We first describe the evaluation platform specification and the specifics of the assumed variation modeling. We then present the problem formulation, and finally we discuss how our proposed work complements and expands on existing probabilistic real-time embedded system research.

5.3.1 Platform Specification and Modeling

Heterogeneous multiprocessors tend to be more efficient than homogenous multiprocessor implementations at tackling inherent application heterogeneity [44], since each processing element is optimized for a particular part of the application running on the system. Motivated by this characteristic, the PEs of the NoC platform utilized in this work are assumed to be heterogeneous and interconnected with routers. For instance, the PE can be a Digital Signal Processor (DSP), general-purpose CPU or an FPGA. Note that field-programmable modules are starting to be integrated into larger SoCs to enable extra flexibility in the mapped design through customization of the embedded FPGA fabric [44].

Our delay distribution model for the PE and the router is based on the maximum delay distribution model of processors, as presented in [15]. According to this model, the critical path delay distribution of a processor due to inter-die and intra-die variations is modeled as

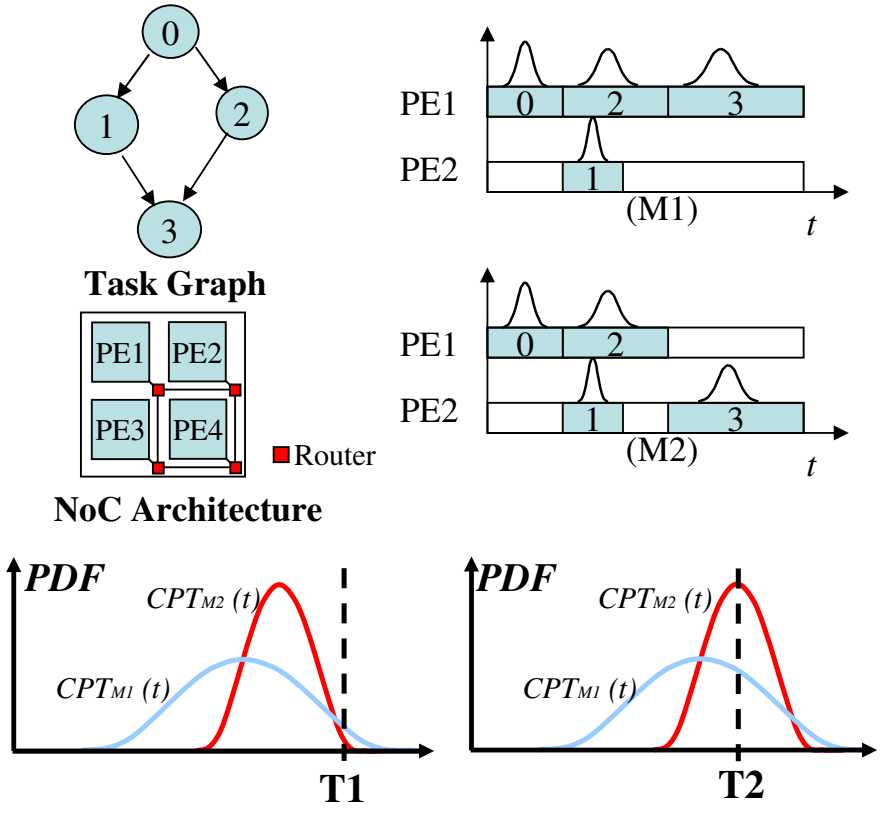
two normal distributions: $f_{inter} = N(T_{norm}, \sigma D_{inter})$ and $f_{intra} = N(T_{norm}, \sigma D_{intra})$, respectively. T_{norm} is the mean value of the critical path delay. The impact of both inter-die and intra-die variations on the chip's maximum critical path delay distribution is estimated by combining these two delay distributions. The maximum critical path delay density function resulting from inter-die and intra-die variations is then calculated as the convolution of $f_{T_{norm}}$, $f_{inter-dmax}$ and $f_{intra-dmax}$ [15]:

$$f_{chip} = f_{T_{norm}} * f_{inter-dmax} * f_{intra-dmax} \tag{5.1}$$

$f_{inter-dmax} = N(0, \sigma D_{D2D})$ is obtained by shifting the original distribution, f_{inter}. $f_{T_{norm}} = \delta(t - Tnorm)$ is an impulse at T_{norm}. The chip's intra-die maximum critical path delay density function is $f_{intra-dmax} = N_{cp} f_{intra} \times (F_{intra})^{N_{cp}-1}$, where F_{intra} is the chip's intra-die cumulative delay distribution and N_{cp} is the number of critical paths present in the design under evaluation.

5.3.2 Problem Formulation

In task scheduling and communication path allocation, the application is represented as a directed acyclic precedence graph G=(V,E), as shown in Fig. 5.1. The vertex, $v_i \in V$, in the task graph represents the computational module, i.e. task. The arc, $e(i,j) = (v_i, v_j) \in E$, represents both the precedence constrains and the communications between task v_i and task v_j. The weight $w(e(i,j))$ associated with the arc $e(i,j)$ in a task graph represents the amount of data that passes from task v_i to v_j.



Deadline	WC	NC	PV
T1	M2	M1	M2(higher yield)
T2	M2	M1	M1(higher yield)

Fig. 5.1. An example of process-variation-aware task scheduling for an NoC architecture. The NoC architecture contains four PEs, *PE1-4*, and these PEs are connected by routers. For illustration purpose, this simple example does not shows the communication mapping. Assume that *PE1* has larger delay variation than *PE2* due to intra-die variation. $M1$ and $M2$ are two schedules for the possible placement of *Task 3*. In schedule *M1*, *task 3* is scheduled onto *PE1*, which causes a large completion time variation. In schedule *M2*, *task 3* is scheduled onto *PE2*, which results in smaller completion time variation. In (a) and (b), the distributions of the completion times of the two different task schedules *M1* and *M2*, denoted as $CPT_{M1}(t)$ and $CPT_{M2}(t)$, are shown here as Probability Distribution Functions (PDF).

As previously stated, the target NoC architecture in this work contains heterogenous PEs, connected by routers. However, the methodology and algorithm presented in this paper can also be used with alternative interconnection fabrics, such as rings, crossbars, or shared bus with appropriate modifications to reflect the communication overhead of the interconnection protocol.

To account for process variation in task scheduling and communication path allocation,

1. An execution time distribution table captures the effects of process variability by associating each task node, t_i, in the task graph with execution time distributions corresponding to each PE in the system; i.e. element $delay[i][j]$ in the table stores the execution time distribution of task t_i if it is executed on the *jth* PE in the architecture.

2. The delay distributions of the routers capture the effects of process variability. The NoC routers are nominally expected to operate at a particular frequency. However, process variability may degrade this nominal performance. Therefore, some of the routers in the interconnection network may well be slower than others. Variation in speed between the routers will, inevitably, affect the inter-processor communication efficiency. Hence, when mapping tasks to specific processing cores on the basis of process variability robustness, one should also account for variability in the communication fabric. Our model accurately captures this phenomenon, leading to a more inclusive and complete modeling of the entire SoC.

3. In addition, a new metric called *performance yield* is introduced to evaluate process scheduling. The performance yield is defined as the probability of the assigned schedule meeting the deadline constraint:

$$Yield = P(completion\_time\_of\_the\_schedule \leq deadline) \tag{5.2}$$

Thus the variation-aware task graph scheduling process is formulated as: *Given a directed acyclic task graph for an application running on an NoC architecture that contains heterogeneous PEs, find a feasible mapping, which includes the scheduling of the tasks to the PEs and the communication path allocation, and determine a mapping which maximizes the performance yield of the mapping under predefined performance constraints.*

Fig. 5.1 shows an example of mapping a task graph to a four-PE NoC platform. The example illustrates the difference between Process-Variation-aware (PV) task scheduling and conventional deterministic task scheduling based on Worst-Case (WC) or Nominal-Case (NC) delay models. Note that the communication path allocation is not considered in this simple example. The NoC platform contains four PEs, *PE1-PE4*. *PE1* has larger delay variation than *PE2* due to intra-die variation. $M1$ and $M2$ are two schedules for possible placement of *Task 3*. In schedule *M1*, *task 3* is scheduled onto *PE1*, which causes large completion time variation. In schedule *M2*, *task 3* is scheduled onto *PE2*, which results in smaller completion time variation. The distributions of the Completion Time (CPT) of task schedules, *M1* and *M2*, are denoted as $CPT_{M1}(t)$ and $CPT_{M2}(t)$, respectively. Given that the deadline of schedule M is T, with a completion time distribution of $CPT_M(t)$, the performance yield of schedule M, can

be computed as

$$Yield_M(T) = \int_0^T CPT_M(t)dt \tag{5.3}$$

It can be observed that deterministic scheduling techniques lead to inferior scheduling decisions. When the deadline of the task is set to *T1* as in Fig. 5.1(a), *task 3* should be scheduled onto *PE2* to achieve higher yield, i.e., $Yield_{M1}(T1) < Yield_{M2}(T1)$. However, deterministic scheduling based on nominal case delay models would choose *PE1* because the nominal case completion time of schedule *M1* is less than that of schedule *M2*. Meanwhile, when the deadline of the task is set to *T2*, as in (b), *task 3* should be scheduled onto *PE1* to obtain larger yield, i.e., $Yield_{M1}(T2) > Yield_{M2}(T2)$. However, deterministic scheduling based on worst case delay models would choose *PE2* because the worst case completion time of schedule *M2* is less than that of schedule *M1*. Thus, it is critical to adopt process-variation-aware scheduling, which takes into account the distribution of the execution time and not merely a nominal or worst case value.

In summary, the scheme proposed in this paper provides a complementary perspective to existing probabilistic real-time embedded system analysis by taking into account the underlying hardware variations during the task allocation and scheduling processes.

5.4 Statistical Task Graph Timing Analysis

In this section, we present our statistical timing analysis methodology for the application task graph.

In statistical timing analysis for task graphs, the timing quantity is computed by using two atomic functions sum and max. Assume that there are three timing quantities, A, B, and

C, which are random variables. The sum operation $C = sum(A, B)$ and the max operation $C = max(A, B)$ will be developed as follows:

1. The sum operation is easy to perform. For example, if A and B both follow Gaussian distributions, the distribution of $C = sum(A, B)$ would also follow a Gaussian distribution with a mean of $\mu_A + \mu_B$ and a variance of $\sqrt{\sigma_a^2 + \sigma_b^2 - 2\rho\sigma_a\sigma_b}$; ρ is the correlation coefficient.

2. The max operation is quite complex. Tightness probability [20] and moment matching techniques could be used to determine the corresponding sensitivities to the process parameters. Given two random variables, A and B, the tightness probability of random variable A is defined as the probability of A being larger than B. An analytical equation presented in [20] is used to compute the tightness probability, thus facilitating the calculation of the max operation.

The delay distribution of the PE and the router in the system can also be obtained through statistical timing analysis tools [98] [16]. With the atomic operations defined, the timing analysis for the resulting task graph can be conducted using PERT-like traversal [16].

5.5 Process-Variation-Aware Task and Communication Mapping for NoC Architectures

In this section, we first introduce the new metric of *performance yield* in the dynamic priority computation of task scheduling. We then present a yield computation method for task graphs. Finally, based on the new dynamic priority computation scheme, a new statistical scheduling algorithm is presented.

5.5.1 Process-Variation-Aware Dynamic Priority

In a traditional dynamic list scheduling approach, the ready tasks − for which the precedent tasks have already been scheduled − are first formed; the priorities of these ready tasks are then computed. Finally, the task node with the highest priority for scheduling is scheduled. The above steps are repeatedly executed until all the tasks in the graph are scheduled. In this approach, the priority of the task is recomputed dynamically at each scheduling step, thus we call the priority of the task as *dynamic priority*. In previous work in the literature, the dynamic priority of the ready task is computed based on deterministic timing information from the PEs. We refer to this technique as *deterministic task scheduling* (DTS).

However, under large process variations, the delay variations for a PE have to be taken into account in the dynamic priority computation, leading to *statistical task scheduling* (STS). To compute the dynamic priority in statistical task scheduling, we introduce a new metric, called conditional *performance yield* for a scheduling decision. The conditional performance yield for a task PE pair, $Yield(T_i, P_j)$, is defined as the probability of the task schedule meeting the predefined performance constraints. It is denoted as $Probability\ (D_{TaskGraph} < deadline|(T_i, P_j))$, where $D_{TaskGraph}$ is the completion time of the entire task graph under the condition that task T_i is scheduled onto PE P_j. The yield metric is an effective metric in guiding the task scheduling, as shown in the example in Fig. 5.1.

Based on the aforementioned performance yield definition, the process-variation-aware Dynamic Priority (DP) for task T_i can be computed as

$$PVaware\_DP(T_i) = Yield(T_i) + \Delta Yield(T_i) \tag{5.4}$$

where we assume that the largest yield is obtained when task T_i is scheduled to the *jth* PE. P_j, $Yield(T_i)$ is computed as:

$$Yield(T_i) = Yield(T_i, P_j) \tag{5.5}$$

Similar to deterministic scheduling [86], we define $\Delta Yield(T_i)$ as the difference between the highest yield and the second highest yield, which stands for the yield loss if task T_i is not scheduled onto its preferred PEs. We have

$$\Delta Yield(T_i) = Yield(T_i) - 2nd\_Yield(T_i) \tag{5.6}$$

where $Yield(T_i) = Yield(T_i, P_j)$, and $2nd\_Yield(T_i) = max\{Yield(T_i, P_k)\}, \forall k \in [1, N]\, and\, k \neq j$.

In finding the best PE for a specific task, T_i, instead of using the relative complex yield metric of the entire task graph, we compare two delay distributions of the path directly to speed up the scheduling process. We define $path\_yield(P_j)$ as $Probability(delay\_P_j < deadline|(T_i, P_j))$, where $delay\_P_j$ is computed as the longest path passing through task T_i.

5.5.2 Yield Computation for Partially Scheduled Task Graphs

To compute the performance yield at each scheduling step, we first estimate the delay of the path in the partially scheduled task graph, in which a part of the task graph has not been scheduled. During the scheduling, some tasks have not been mapped to PEs, and paths in both the *scheduled* and *unscheduled* sub-task graphs contribute to the delay of the entire task graph. For example, in Fig. 5.2, the path going through *Task 4* consists of two paths, *p1* and *p2*, where *p1* is the path from the start task node, *Task 0*, to *Task 4*, and *p2* is the path from *Task 4* to the end

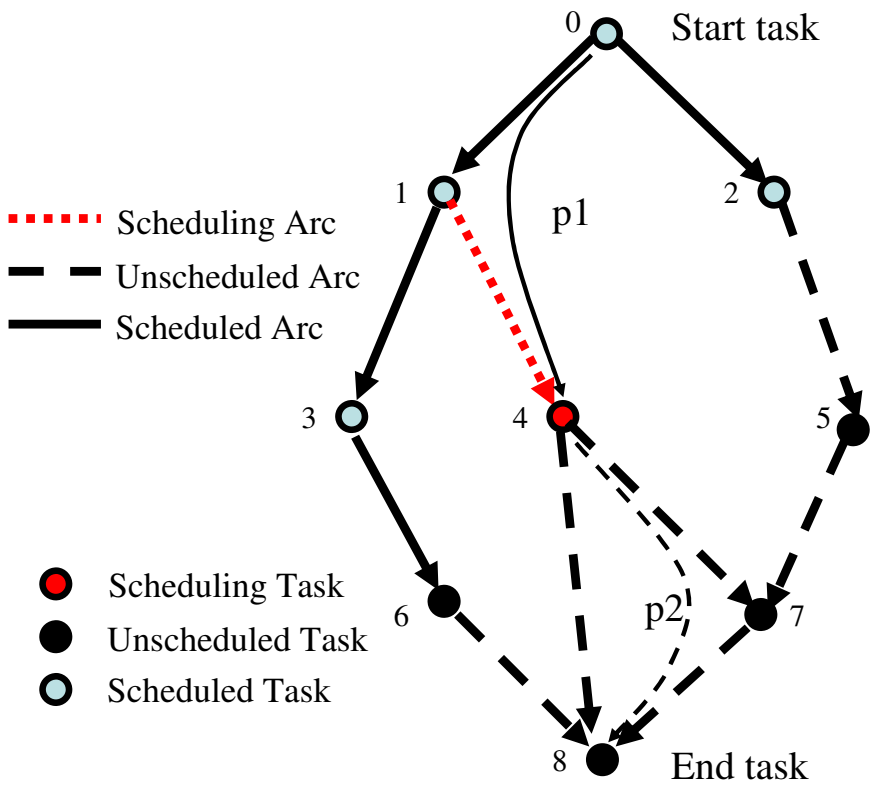


Fig. 5.2. During scheduling, some tasks have not been mapped to PEs, and paths in both the *scheduled* and *unscheduled* sub-task graphs contribute to the delay of the entire task graph. The path going through *Task 4* consists of two paths, *p1* and *p2*, where *p1* is the path from the start task node, *Task 0*, to *Task 4*, and *p2* is the path from *Task 4* to the end task node, *Task 8*.

task node, *Task 8*. The delay of that path can be computed as $delay(path\_T_4) = delay(p1) + delay(p2)$. *p2* is considered as a path in the unscheduled sub-task graph. Consequently, to compute the path delay, the timing quantities of the tasks in the *unscheduled* sub-task graph need to be determined. In this work, we develop two methods to approximate these timing quantities (delay distributions):

1. For each task, the *adjust mean* and *adjust variance* are used in computing the delay of the unscheduled task graph. We define an adjust mean and an adjust variance of the execution time, denoted $AdjMean(T_i)$ and $AdjSigma(T_i)$, respectively, for unscheduled task T_i as:

$$AdjMean(T_i) = \frac{\sum_{j=1}^{N} E(Delay(T_i, P_j))}{N} \tag{5.7}$$

$$AdjSigma(T_i) = \frac{\sqrt{\sum_{j=1}^{N} \sigma(Delay(T_i, P_j))^2}}{N} \tag{5.8}$$

   where N is the total number of PEs in the NoC platform and $Delay(T_i, P_j)$ is the execution time distribution of task T_i over PE P_j.

2. We first perform an initial scheduling via deterministic scheduling or variation-aware scheduling based on timing information obtained through method 1). We then evaluate the timing quantities for the tasks in the unscheduled sub-task graph based on these initial scheduling results.

Thus, with the timing quantities of the tasks nodes and edges in the unscheduled task graph determined, we compute the Critical Path Delay, CPD, from a particular task node to the end task node in the task graph. We perform statistical timing analysis to compute the CPD for each task node in a task graph through a single PERT-like graph traversal. After the path delay

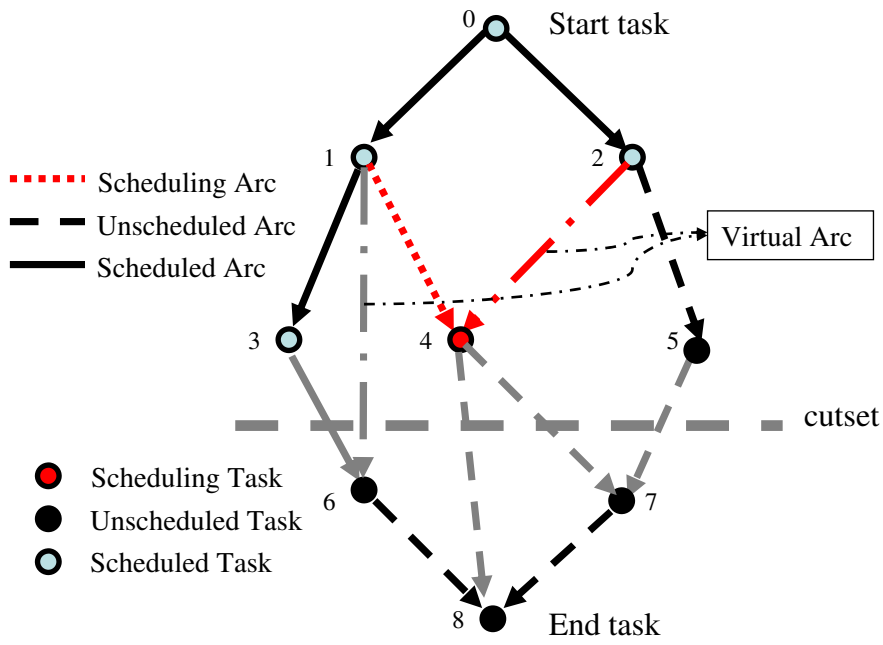


Fig. 5.3. Yield computation for a partially scheduled task graph to address *structural dependency*. The *virtual arc* from task 1 to task 6 indicates that task 1 and task 6 are scheduled onto the same PE and there is no data transferring on the arc (1,6). Virtual arcs have to be included in the cut-set, since they may lie on the critical path.

has been determined, the completion time of the partially scheduled task graph needs to be computed. Based on the same concept of cut-set as in [19], we develop a yield computation method for a partially scheduled task graph. From graph theory, all the paths from the source node to the sink node have to pass through any cut-set which separates the source and sink nodes in a directed acyclic graph. Consequently, the delay distribution of the entire task graph can be obtained by the Max operation over the delays of all the paths going through the edges in that cut-set.

However, the task graphs have to be modified to include *structural dependency*. We borrow the term *structural dependency* from pipeline design and define it in our context as two tasks being scheduled onto the same PE. As shown in Fig. 5.3, let us assume that task 6 and task 1 are scheduled onto the same PE. Although there is no data transfer from task 1 to task 6, *virtual arc* (1,6) is added to the task graph. Despite the fact that structural dependencies have been taken into account in the preceding timing computation, *one still has to include these virtual arcs in the cut-set extraction, as illustrated in Fig. 5.3, because these arcs may, in fact, lie on the critical path*. Given the longest path delay through each task node in the cut-set, the performance of the entire task graph can be computed as

$$D_{TaskGraph} = Max\{path\_delay(T_i)\},\ \forall T_i \in cutset \tag{5.9}$$

where $path\_delay(T_i)$ is the delay of the longest path from the start task node to the end task node passing through task T_i in the cut-set. At each step of task scheduling, the only changes to the timing quantities in a partially scheduled task graph are those of the task node being scheduled and its incoming arcs, as shown in Fig. 5.3. The approximate timing quantities of

the task being scheduled and its incoming arcs are replaced with the values computed using the delay information of the PE onto which the particular task is being scheduled. Assume task T_i is being scheduled onto PE P_j. The delay of the longest path from the start task node to task node h,$AVT(T_i, P_j)$, can be computed as $max[data\_available(T_i, P_j), PE\_available(P_j)] + execution\_time(T_i, P_j)$. $data\_available(T_i, P_j)$ represents the time the last of the required data becomes available from one of its parent nodes. $PE\_available(P_j)$ represents the time that the last node assigned to the jth PE finishes execution. Thus the longest path delay going through task T_i, $path\_delay(T_i)$, can be computed as

$$path\_delay(T_i) = AVT(T_i, P_j) + CPD(T_i) \tag{5.10}$$

5.5.3 Mapping Algorithm

```
PVschedule (Task Graph,Platform){
1.  Perform initial scheduling for Task Graph
2.  Perform statistical timing analysis and initialize the RTS;
3.  While (RTS is not empty){
4.    Select N most critical tasks and form CTS;
5.      For each task T_i in the CTS{
6.        Tentatively schedule T_i to PEs to obtain the best and
            2nd best (T_i, P_j) pair;
7.      }
8.    Select the (T_i, P_j) pair with maximum DP and Schedule T_i onto P_j;
9.    Add new ready tasks to RTS;
10.   }
11. Compute the yield;}
```

Fig. 5.4. The pseudo code of the proposed variation-aware task scheduling process

With the notion of variation-aware dynamic priority defined, the corresponding scheduling algorithm designed to mitigate the effects of process variation and improve the yield is shown in Fig. 5.4. Our scheduling algorithm takes the $Task\ Graph$ and the $specification$ of the NoC platform as inputs, and outputs the mapping of tasks to PEs. It also computes the performance yield of that mapping. The scheduling algorithm is described in detail below:

1. In the initial scheduling of the task graph, the tasks are scheduled with either a deterministic scheduling method or a variation-aware method with the assumption that the distribution of the delay of each task is determined by the *adjust mean* and the *adjust variance* over the PEs in the NoC architecture (Line 1).

2. Perform statistical timing analysis for the initial scheduled task graph and initialize the Ready Tasks Set (RTS) (Line 2).

3. Generate the RTS; that is, the tasks for which the precedent tasks have already been scheduled (Line 3). After the task is scheduled, the new ready tasks are added to the RTS (Line 9).

4. We choose N top-most critical tasks in the RTS as the Critical Task Set (CTS) and give higher priority to those tasks that have larger impact on the performance yield of the schedule (Line 4). The critical task is the task that has higher probability of being on the critical path. The CTS scheme enables us to focus on the most critical tasks. Thus, the computation cost can be greatly reduced, especially for large task graphs.

5. During tentative scheduling, task T_i is scheduled onto all the possible PEs in the NoC platform (Line 6). Two feasible mappings with the highest and second highest yield values

are then selected. Finally, after all the tasks in the *CTS* finish tentative schedule, the dynamic priority (DP) for each task is computed and the task-PE pair (T_i, P_j) with the maximum DP value is selected. Task T_i is then scheduled onto PE P_j (Line 8). During the tentative scheduling, the path allocation is performed for the communication between the task nodes. In this work, we propose a fast path allocation algorithm as shown in Section 5.5.4.

5.5.4 Fast Path Allocation Algorithm

From the scheduling algorithm presented in the previous subsection, the communication path allocation occurs in an inner loop of the tentative schedule to determine the scheduling priority. Thus, it is critical to reduce the computation complexity of the path allocation part, in order to improve the performance of the entire algorithm. In this section, we present our fast routing algorithm to allocate communication links for the inter-task communication arcs. With our routing algorithm, the minimal, deadlock-free, and optimal routing path can be found by a single breadth-first traversal of the interconnection mesh network, and a backward path extraction with linear complexity in the NoC size.

Hu and Marculescu [37] argue that the most appropriate routing technique for the NoC architecture of a specific application should be static, deadlock-free, wormhole based, and minimal due to resource limitations and latency requirements. Furthermore, the traffic patterns of the application itself should play a major factor in deciding the routing algorithm. Consequently, in this work, we employ the same routing techniques and assume registers as the buffer medium in the routers. To ensure deadlock freedom, we employ the odd-even turn model [18] in our routing algorithm. A 2D mesh interconnect network consists of four types of links: north, east, south

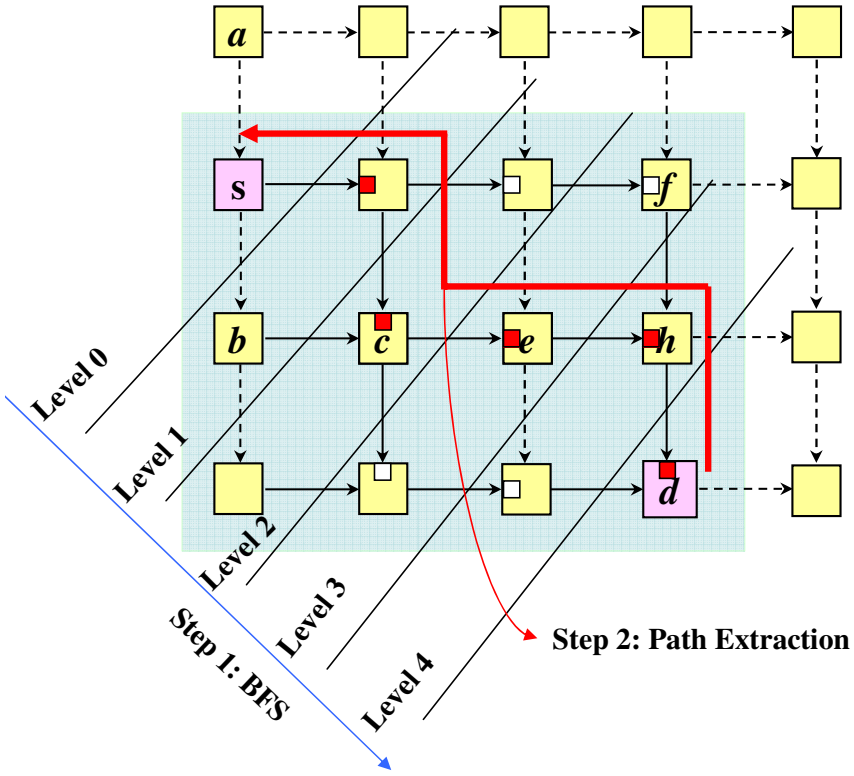


Fig. 5.5. An example of path allocation for the path from s to d in a mesh network is shown in two steps: 1) perform a BFS search in the shaded mesh network to identify best links; 2) extract the optimal path according to the best link flags (small red/dark square).

and west (a fifth link connects each router to a local processing element). The odd-even turn model prohibits east-north and east-south turns at the routers located in the even columns of the mesh, and prohibits north-west and south-west turns at the routers located in the odd columns.

The routing algorithm is shown in Fig. 5.6. Our algorithm takes the incoming communication arcs of task T_i as inputs, and allocates the communication links for these arcs to compute the latest data available time for task T_i. We use the *critical* and least *flexible* routing communication arc first policy because the communication arcs with higher criticality value have larger impact on the yield, and those arcs with least number of alternative path, i.e., lower path flexibility [37], have fewer routing options to obtain the optimal routing path [38].

In Fig. 5.6, the *getpath* function allocates the communication links for the path from a *source* PE to a *destination* PE in the NoC architectures. The path allocation function consists of two steps: 1) a BFS search from the source PE to the destination PE (From Line 2 to Line 8); 2) a backward path extraction (Line 9) from the destination PE to the source PE. At each intermediate node (PE), the best link available time (LVT) is obtained from its initial node of incoming links according to best link flag at Line 4. From Line 5 to Line 6, a for loop iterates over all the valid outgoing links of the current node. Valid links enforce the minimal, and deadlock-free routing requirements. We compute the new best LVT according to the available time and bandwidth of the current link at Line 6. If the terminal node of the current link has been visited, the new best LVT is compared with that of that node its best link flag is updated; otherwise, store the best LVT at that node and set its best link flag as from current node at Line 7. After the BFS traversal, we perform the path extraction according to the best link flags at Line 9. The computation complexity of the algorithm for getting the path is $O(N) + O(l)$, where N is the number of the PEs in the routing area and l is the level of the mesh network in the routing area.

In Fig. 5.5, we show an example of getting the path from source PE s to destination PE d. The dashed line represents the link we can not use if we enforce both the minimal and deadlock free routing requirements. For example, for minimal routing, we can not move to PE a at s; to be deadlock-free, we apply the old even rules, which prohibit routing from s to b. The best link flag is denoted as a small red square on the sides of the bigger square. At node e, we obtain the best LVT from c, as the best link flag is on the left side of e. The next valid move is to h. At h, we compute the new LVT, and if h has been visited, the new LVT at h is compared with the best LVT at h, and the best link flag is set to the left side, if the new LVT is chosen. After the BFS traversal reaches d, we start to trace back from d to extract the path according to the best link flag.

```
routing(incoming_arclist){
1.   for each incoming arc in the list{
2.      path=getpath(source, dst);
3.      update the data_available_time and tentative scheduling information;
4.   }}

getpath(source, dst){
1.   enqueue(Q, source);
2.   while (Q is not empty){
3.      u ← dequeue(Q);
4.      get the current link_available_time of u;
5.      for (each valid adjacent node v of u){
6.         update best link_available_time for v;
7.         update the current best_link_flag for v;
8.         enqueue(Q, v);}}
9.   extract the path according to the best link flag from dst;
}
```

Fig. 5.6. The pseudo code of routing algorithm.

5.6 Evaluation Results

In this section, we present our evaluation platform and analyze the simulation results. It will be shown that the proposed method can effectively reduce the impact of manufacturing process variability and maximize performance yield, significantly outperforming traditional deterministic task and communication mappers that use worst-case or nominal-case delay models.

Our variation-aware mapping algorithm was implemented in C++ and experiments were conducted using various benchmarks, including an MPEG2 benchmark [56], two embedded system synthesis benchmarks from the E3S suites [27] (these are based on data from the Embedded Microprocessor Benchmark Consortium (EEMBC)), and three benchmarks (BM1-BM3) from [107]. These benchmarks are allocated and scheduled onto an NoC platform with 4-16 heterogenous PEs, each of which has a Gaussian distribution of clock frequency to reflect the effects of process variation. Two sets of experiments are performed to demonstrate the effectiveness of our statistical task and communication mapping(STCM) algorithm. The first set of results is related to the yield improvement of our variation-aware task and communication mapping process. The second set of experiments evaluates our algorithm within the context of cost saving.

To demonstrate the yield improvement of the proposed method, the simulation results are compared against those of traditional deterministic task and communication mapping (DTCM) procedures using Nominal-Case (NC) and Worst-Case (WC) delay models. In phase one of the evaluation process, deterministic and statistical mapping for the task graphs are performed. Phase two computes the yield after the timing analysis of the scheduled task graph, by using Equation (5.2). Table 5.1- 5.2 shows the results of the proposed statistical method against those

of deterministic scheduling techniques with 99% and 90% performance yield target, respectively. The first column shows the benchmarks employed in this analysis. From the second column to the fourth column, we show the absolute yield results of the Process-Variation-aware (PV) mapper, the deterministic Worst-Case (WC) mapper, and the deterministic Nominal-Case (NC) mapper, respectively. In the fifth and sixth columns, we show the yield improvement of our statistical method over worst-case (PV-WC) and nominal-case (PV-NC) methods. As illustrated in Table 5.1, significant yield improvement can be obtained through variation-aware task communication mapping. Although DTCM can result in high yield values for a few benchmarks, it is not able to guarantee the yield across all benchmarks. The yield results show that WC-based techniques are grossly pessimistic with an average 16% yield loss. Similarly, NC-based techniques result in a 43% yield loss on average. As we tighten the performance constraint and set performance yield target as 90%, average 62 % and 51% yield improvements over worst-case and nominal-case deterministic mapper, respectively, can be obtained across the benchmarks by using the proposed variation-aware mapper.

Table 5.1. Yield Improvement over a Deterministic Mapper with 99% Performance Yield Target

Benchmark	PV	WC	NC	PV-WC	PV-NC
MPEG2	99%	96 %	99 %	3 %	0 %
Telecom	99%	15 %	49 %	84 %	50 %
Auto-indust	99%	98 %	98 %	1 %	1 %
BM1	100%	93 %	0 %	7 %	100 %
BM2	100%	99 %	15 %	1 %	85 %
BM3	100%	100 %	80 %	0 %	13 %
Average	100%	88 %	62%	16 %	43%

Table 5.2. Yield Improvement over a Deterministic Mapper with 90% Performance Yield Target

Benchmark	PV	WC	NC	PV-WC	PV-NC
MPEG2	90%	82 %	90 %	8 %	0 %
Telecom	90%	0 %	3 %	90 %	87 %
Auto-indust	90%	83 %	83 %	7 %	7 %
BM1	90%	0 %	0 %	90 %	90 %
BM2	89%	0 %	0 %	89 %	89 %
BM3	90%	0 %	56 %	90 %	34 %
Average	90%	28 %	39%	62 %	51%

We define *cost* as the amount of additional performance required by the NoC architecture to meet a target yield of 99% when using a deterministic mapper (DTM), as compared to using the variation-aware mapper (STM): $cost = \frac{required\_frequency(DTS)}{required\_frequency(STS)} - 1$. This cost metric is an effective indicator of actual manufacturing cost, as higher performance requirements translate to an increase in design and manufacturing effort. Furthermore, a higher frequency would increase power density and result in even larger variations, larger voltage drops and elevated thermal issues. Table 5.3 and 5.4 shows the cost saving results based on this cost definition with 99% and 90% performance yield target, respectively. In the second and third columns, we show the percentage of additional speed required by the NoC architecture when using a deterministic WC mapper and a deterministic NC mapper, respectively. As can be seen from the table, as large as 6% and 5% cost can be saved over these two deterministic approaches by using the proposed process-variation-aware mapper.

Table 5.3. Cost Saving over a Deterministic Mapper with 99% Yield Target

Benchmark	WC	NC
MPEG2	3%	0%
Telecom	4%	4%
Auto-indust	3%	3%
BM1	3%	4 %
BM2	2%	5%
BM3	0%	4 %
Average	3%	4 %

Table 5.4. Cost Saving over a Deterministic Mapper with 90% Yield Target

Benchmark	WC	NC
MPEG2	3%	0%
Telecom	4%	4%
Auto-indust	3%	3%
BM1	5%	5 %
BM2	6%	5%
BM3	5%	4 %
Average	4%	4 %

5.7 Summary

In this Chapter, the author formulates a variation-aware scheduling process for heterogeneous multi-core NoC architectures and propose a statistical mapping algorithm to mitigate the effects of parameter variations. The proposed scheme accounts for variability in *both* the processing elements and the communication fabric. Just like the processing cores, the NoC routers are also assumed to be affected by process variation. Hence, the model employed by our algorithm captures this effect as part of the mapping and allocation process. By including both the *processing* and *communication* aspects of the SoC in the algorithm, we completely and accurately model the effect of variability in the entire SoC.

A new metric, known as *performance yield*, is used at each step of the iterative dynamic priority computation of the task and communication mapper. Further, an efficient yield computation method for task and communication mapping and a fast path allocation algorithm have been proposed to improve the performance of the scheduling algorithm. Simulation results show that significant yield gains can be obtained with our variation-aware scheduling methodology. More specifically, significant performance yield improvements over worst-case and nominal-case deterministic mappers, respectively, are reported.

Chapter 6

Conclusion and Future Work

Process variation has emerged as a major challenge as technology scales to nanometer regime. The continuously increased magnitude of the parameter variations due to the process variations causes a failure of the traditional corner-based analysis and design for worst case approaches. Thus, the statistical design approaches have surfaced as a promising solution to mitigate the impact of the process variations.

The statistical gate level analysis and optimization have become a major research focus, and a variety of techniques have been proposed. However, the statistical design techniques at module level and system level remains unexplored. Thus, the researches presented in this thesis focus on incorporating process variability into high level synthesis and task scheduling and allocation for embedded system. Toward this effort, statistical performance and power analysis are first developed to lay down the foundation for the design exploration at module and system level. Based on the analysis, the parametric yield concept is developed to perform the design space exploration in high level synthesis and hardware/software co-synthesis. In addition, the proposed work represents the first attempt to consider the post silicon tuning during design-time optimization in high level synthesis. Overall, the presented work makes an important contribution to move from the deterministic design methodologies to the statistical design methodologies in high level synthesis and hardware/software co-synthesis.

However, the work presented in this thesis constitutes the first attempt to incorporate the notion of manufacturing process variability at higher levels, many new topics could be potentially explored in the future. In particular, scheduling is an essential step in the high level synthesis. Scheduling determines the clock cycles where the operations could be assigned. Although scheduling does not directly affect the parametric yield, it has significant impact on the following steps, resource sharing and assignment, which have direct impact on the parametric yield. Thus, using the parametric yield to guide the scheduling is worth exploring. In addition to high level synthesis, the presented work in hardware/software co-synthesis have only accounted for the performance variations, the methodology could be extended to include a power model that captures variability in leakage power, which is expected to play a dominant role in future technology nodes. Therefore, the proposed future work in hardware/software co-synthesis is performing the hardware/software co-synthesis considering both the performance and power variations. In addition to the system level variation aware design techniques, FinFET technology has been proposed as a promising alternative for deep sub-micro bulk CMOS technology. The preliminary work [102] provides the dependability analysis of FinFET circuits, and demonstrates that the FinFet based circuit is more resilient to the process variations than the bulk CMOS based circuit as technology scales below 45nm. Thus, it can be predicted that replacing the bulk CMOS technology with FinFet technology in building the system could effectively counter the effects of process variations. Therefore, design of system with FinFet devices could be investigated.

Although process variations are considered as a major source of unreliability, other major reliability concerns faced by designers of systems include transient errors arising from internal and external noise sources and increasing on-chip temperature due to increasing power density

[12] [104] [105]. These two reliability issues also create great challenges for designers in designing a reliable system. In addition, the thermal issues, transient errors and process variations interact with each other, and this further complicates the design process. Thus, the research on building a robust system considering the interaction among these sources of unreliability is worth exploring.

References

[1] Catapult c synthesis. Technical report, Mentor Graphics Corporation, Products Overview,.

[2] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical clock skew analysis considering intra-die process variations. In *International Conference on Computer Aided Design*, pages 914–921, 2003.

[3] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 900–907, November 2003.

[4] A. Agarwal, K. Chopra, D. Blaauw, and V. Zolotov. Circuit optimization using statistical static timing analysis. In *Design Automation Conference*, pages 321–324, 2005.

[5] A. Agarwal, B. C. Paul, S. Mukhopadhyay, and K. Roy. Process variation in embedded memories: Failure analysis and variation aware architecture. *IEEE Journal of Solid-State Circuits*, 40(9):1804–1814, 2005. 0018-9200.

[6] A. Agarwal, V. Zolotov, and D. Blaauw. Statistical timing analysis using bounds and selective enumeration. *IEEE Trans. CAD*, (9):1243–1260, September 2003.

[7] V. Axelrad and J. Kibarian. Statistical aspects of modern IC designs. *Proceedings of the 28th European Solid-State Device Research Conference*, pages 309–321, September 1998.

[8] N. Azizi and F. N. Najm. Compensation for within-die variations in dynamic logic by using body-bias. In *IEEE-NEWCAS Conference*, pages 167–170, 2005.

[9] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano. Asymptotic Zero-Transition Activity Encoding for Address Buses in Low-Power Microprocessor-Based Systems. *Great Lakes Symposium on VLSI (GLSVLSI)*, pages 77–82, March 1997.

[10] J.L. Bentley, D.D. Sleator, R.E. Tarjan, and V.K. Wei. A Locally Adaptive Data Compression Scheme. *Communications of the ACM*, Vol. 29(4):320–330, 1986.

[11] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM J. Res. Dev.*, 50(4/5):433–449, 2006.

[12] S. Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *Micro, IEEE*, 25(6):10–16, 2005. 0272-1732.

[13] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter variations and impact on circuits and microarchitecture. In *Design Automation Conference*, pages 338–342, 2003.

[14] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[15] K. A. Bowman, S. G. Duvall, and J. D. Meindl. Impact of Die-to-Die and Within Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration. *Journal of Solid-State Circuits*, pages 183–190, February 2002.

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

[17] T. Chen and S. Naffziger. Comparison of adaptive body bias (abb) and adaptive supply voltage (asv) for improving delay and leakage under the presence of process variation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(5):888–899, 2003. 1063-8210.

[18] G. M. Chiu. The odd-even turn model for adaptive routing. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):729–738, 2000.

[19] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester. Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation. *International Conference on Computer-Aided Design*, pages 1023–1028, November 2005.

[20] C. Clark. The greatest of a finite set of random variables. *Operations Research*, pages 145–162, 1961.

[21] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. *In Proceedings of the 38th Design Automation Conference*, June 2001.

[22] Azadeh Davoodi and Ankur Srivastava. Probabilistic dual-vth leakage optimization under variability. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 143–148, New York, NY, USA, 2005. ACM.

[23] Liang Deng and Martin D. F. Wong. An exact algorithm for the statistical shortest path problem. *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 965–970, 2006.

[24] S. Devadas, H. F. Jyu, K. Keutzer, and S. Malik. Statistical timing analysis of combinational circuits. In *International Conference on Computer Design: VLSI in Computers and Processors*, pages 38–43, 1992.

[25] A. Devgan and C. Kashyap. Block-based static timing analysis with uncertainty. In *International Conference on Computer Aided Design*, pages 607–614, 2003.

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

[27] Robert P. Dick. Embedded systems synthesis benchmarks suite (e3s). *http://www.ece.northwestern.edu/ dickrp/e3s/*.

[28] S. W. Director, P. Feldmann, and K. Krishna. Statistical integrated circuit design. *Solid-State Circuits, IEEE Journal of*, 28(3):193–202, 1993. 0018-9200.

[29] H. Elzinga. On the impact of spatial parametric variations on mos transistor mismatch. In *IEEE International Conference on Microelectronic Test Structures*, pages 173–177, 1996.

[30] Tohru Furuyama. Keynote speeach: CHALLENGES OF DIGITAL CONSUMER AND MOBILE SoCs: MORE MOORE POSSIBLE? *DATE Conference*, April 2007.

[31] D. Gajski, N. Dutt, and A. Wu. *High-level synthesis: Introduction to chip and system design*. Kluwer Academic Publishers, 1992.

[32] S. Garg and D. Marculescu. System-level process variation driven throughput analysis for single and multiple voltage-frequency island designs. *Proc. IEEE Design, Automation and Test in Europe (DATE)*, Apr. 2007.

[33] A. Gattiker, S. Nassif, R. Dinakar, and C. Long. Timing yield estimation from static timing analysis. *IEEE International Symposium on Quality Electronic Design*, pages 437–442, 2001.

[34] E. F. Girczyc, R. J. Buhr, and J. P. Knight. Applicability of a subset of ada as an algorithmic hardware description language for graph based hardware compilation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-4(2):134–142, 1985.

[35] T. Givargis, F. Vahid, and J. Henkel. System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):416–422, 2002. 1063-8210.

[36] M. R. Guthaus, N. Venkateswarant, C. Visweswariaht, and V. Zolotov. Gate sizing using incremental parameterized statistical timing analysis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 1029–1036, 2005.

[37] J. Hu and R. Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular noc architectures. *date*, 01:10688, 2003.

[38] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. *date*, 1:10234, 2004.

[39] X. S. Hu, Z. Tao, and E. H. M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(6):833–844, 2001. 1063-8210.

[40] W.-L. Hung, X. Wu, and Y. Xie. Guarantee performance yield in high level synthesis. In *International Conference on Computer Aids Design*, 2006.

[41] P. Hurat, Y.-T. Wang, and N. K. Verghese. Sub-90 nanometer variability is here to stay. *EDA Tech Forum*, 2(3):26–28, 2005.

[42] P. Hurat, Y.-T. Wang, and N. K. Verghese. Sub-90 nanometer variability is here to stay. *EDA Tech Forum*, November 2005.

[43] Cheng-Tsung Hwang, Jiahn-Hurng Lee, and Yu-Chin Hsu. A formal approach to the scheduling problem in high level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-10(4):464–475, 1991.

[44] A. Jerraya and W. Wolf. *Multiprocessor systems-on-chips*. Morgan Kaufmann Publishers, 2005.

[45] J. Jess, K. Kalafala, S. Naidu, R. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. *IEEE/ACM DAC*, pages 932–937, 2003.

[46] L. Jiong and N. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *7th Asia and South Pacific Design Automation Conference*, pages 719–726, 2002.

[47] Jongyoon Jung and Taewhan Kim. Timing Variation-Aware High-Level Synthesis. *Proc. of ICCAD*, November 2007.

[48] H. Jyu and S. Malik. Statistical delay modeling in logic design and synthesis. In *Design Automation Conference*, pages 126–130, 1994.

[49] J.Zhang and M.Styblinski. *Yield and variability optimization of integrated cricuits*. Kluwer Publishers, 1995.

[50] V. Khandelwal, A. Davoodi, A. Nanavati, and A. Srivastava. A probabilistic approach to buffer insertion. In *International Conference on Computer Aided Design (ICCAD)*, pages 560–567, 2003.

[51] N. Kim, K. Taeho, K. Bowman, V. De, and T. Mudge. Total power-optimal pipelining and parallel processing under process variations in nanometer technology. In *International Conference on Computer Aided Design*, pages 535–540, 2005.

[52] H. Kramer and W. Rosenstiel. System synthesis using behavioural descriptions. In *Proceedings, European Design Automaton Conference*, pages 277–282. IEEE Computer Society Press, 1990.

[53] S.H. Kulkarni, D. Sylvester, and D. Blaauw. A Statistical Framework for Post-Silicon Tuning through Body Bias Clustering. *Proc. of ICCAD*, pages 39–46, Nov. 2006.

[54] F. Kurdahi and A. Parker. Real: A program for register allocation. In *Proceedings,* 24^{th} *Design Automation Conference*, pages 210–215. ACM Press, 1987.

[55] E. Kursun, A. Srivastava, S. G. Memik, and M. Sarrafzadeh. Early evaluation techniques for low power binding. In *International Symposium on Low Power Electronics and Design*, pages 160–165, 2002.

[56] X. Li, J. Le, M. Celik, and L. Pileggi. Defining statistical sensitivity for timing optimization of logic circuits with largescale process and environmental variations. *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 422–429, November 2005.

[57] Xin Li, Jiayong Le, and Lawrence T. Pileggi. Projection-based statistical analysis of full-chip leakage power with non-log-normal distributions. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 103–108, New York, NY, USA, 2006. ACM.

[58] X. Liang and D. Brooks. Performance optimal micro-architecture parameters selection under the impact of process variation. In *International Conference on Computer Aided Design*, 2006.

[59] Xiaoyao Liang and David Brooks. Mitigating the impact of process variations on processor register files and execution units. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 504–514, Washington, DC, USA, 2006. IEEE Computer Society.

[60] Xiaoyao Liang, Ramon Canal, Gu-Yeon Wei, and David Brooks. Process variation tolerant 3t1d-based cache architectures. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 15–26, Washington, DC, USA, 2007. IEEE Computer Society.

[61] J. Liou, K. Chen, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. *IEEE/ACM DAC*, pages 661–666, 2001.

[62] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi. Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems. *Proc. of the 38th Conference on Design Automation*, 2001.

[63] C.-G. Lyuh and T. Kim. High-level synthesis for low power based on network flow method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(3):364–375, 2003. 1063-8210.

[64] M. Mani, A.K Singh, and M. Orshansky. Joint design-time and post-silicon minimization of parametric yield loss using adjustable robust optimization. In *Proc. of ICCAD*, pages 19–26, 2006.

[65] D. Marculescu and S. Garg. System-level process-driven variability analysis for single and multiple voltage-frequency island systems. In *Proc. IEEE/ACM Intl. Conference on Computer-Aided Design (ICCAD)*, Nov. 2006.

[66] D. Marculescu and E. Talpes. Variability and energy awareness: A microarchitecture-level perspective. In *Design Automatin Conference*, pages 11–16, 2005.

[67] G. Martin and H. Chang. *Winning the SoC revolution: Experiences in real design*. Kluwer Academic Publishers, 2003.

[68] S. P. Mohanty and E. Kougianos. Simultaneous power fluctuation and average power minimization during nano-cmos behavioral synthesis. In *Proc. of VLSID*, pages 577–582, 2007.

[69] R. Mukherjee, S. Ogrenci Memik, and G. Memik. Temperature-aware resource allocation and binding in high-level synthesis. In *Design Automation Conference*, pages 196–201, 2005.

[70] S. Mukhopadhyay and K. Roy. Modeling and estimation of total leakage current in nano-scaled-cmos devices considering the effect of parameter variation. In *International Symposium on Low Power Electronics and Design*, pages 172–175, 2003.

[71] F. N. Najm. On the need for statistical timing analysis. In *Design Automation Conference*, pages 764–765, 2005.

[72] F. N. Najm. On the need for statistical timing analysis. *IEEE/ACM DAC*, pages 764–765, 2005.

[73] S. Nassif. Delay variability: Sources, impacts and trends. In *IEEE International Solid-State Circuits Conference*, pages 368–369, 2000.

[74] S. Nassif. Design for variability in DSM technologies. *IEEE International Symposium on Quality of Electronic Design*, pages 451–454, March 2000.

[75] S. R. Nassif. Modeling and analysis of manufacturing variations. In *IEEE Conference on Custom Integrated Circuits*, pages 223–228, 2001.

[76] M. Orshansky and K. Keutzer. A general probabilistic framework for worst case timing analysis. *IEEE/ACM DAC*, pages 556–561, 2002.

[77] Serkan Ozdemir, Debjit Sinha, Gokhan Memik, Jonathan Adams, and Hai Zhou. Yield-aware cache architectures. In *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 15–25, Washington, DC, USA, 2006. IEEE Computer Society.

[78] B.M. Pangrle. *A Behavioral Compiler for Intelligent Silicon Compilation*. PhD thesis, University of Illinois at Urbana-Champaign, 1987.

[79] A. Papoulis and S. Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw-Hill, 2001.

[80] P. Paulin and J. Knight. Force-directed scheduling for the behavioral synthesis of asic's. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 661–679, 1989.

[81] A. Raghunathan, N. K. Jha, and S. Dey. *High-level power analysis and optimization*. Kluwer Academic Publishers, 1998.

[82] Anand Raghunathan and Niraj K. Jha. An iterative improvement algorithm for low power data path synthesis. In *Proc. of ICCAD*, pages 597–602, 1995.

[83] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 06(4):412–420, 1995.

[84] S. Sapatnekar. *Timing*. Kluwer Academic Publishers, 2004.

[85] Li Shang, Li-Shiuan Peh, Amit Kumar, and Niraj K. Jha. Thermal Modeling, Characterization and Management of On-Chip Networks. *International Symposium on Microarchitecture (MICRO)*, pages 67–78, December 2004.

[86] G. C. Sih and E. A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 04(2):175–187, 1993.

[87] D. Sinha, N. V. Shenoy, and Z. Hai. Statistical gate sizing for timing yield optimization. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 1037–1041, 2005.

[88] D. Sinha and H. Zhou. A Unified Framework for Statistical Timing Analysis with Coupling and Multiple Input Switching. *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, November 2005.

[89] A. Srivastava, S. Shah, K. Agarwal, D. Sylvester, D.Blaauw, and S. Director. Accurate and efficient gate-level parametric yield estimation considering correlated variations in leakage power and performance. *Design Automation Conference (DAC)*, pages 535–540, 2005.

[90] A. Srivastava, D. Sylvester, and D. Blaauw. *Statistical analysis and optimization for VLSI: Timing and power*. Springer, 2005.

[91] X. Tang, H. Zhou, and P. Banerjee. Leakage power optimization with dual-vth library in high-level synthesis. In *Design automation conference*, pages 202–207, 2005.

[92] T. S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. W. S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium*, pages 164–173, 1995.

[93] Abhishek Tiwari, Smruti R. Sarangi, and Josep Torrellas. Recycle:: pipeline adaptation to tolerate process variation. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 323–334, New York, NY, USA, 2007. ACM.

[94] J. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De. Apaptive body bias for reducing imapcts of die-to-die and within-die parameter variaions on microprocessor frequency and leakage. *IEEE Journal of Solid State Circuits*, 37(11):1396–1402, 2002.

[95] C. J. Tseng and D. P. Siewiorek. Automated synthesis of data paths in digital systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-5(3):379–395, July 1986.

[96] G. D. Veciana, M. F. Jacome, and J.-H. Guo. Hierarchical algorithms for assessing probabilistic constraints on system performance. In *IEEE/ACM Design Automation Conference (DAC)*, pages 251–256, 1998.

[97] C. Visweswariah. Death, taxes and failing chips. In *Design Automation Conference*, pages 343–347, 2003.

[98] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, and S. Narayan. First-order incremental block-based statistical timing analysis. *Design Automation Conference (DAC)*, pages 331–336, June 2004.

[99] Feng Wang, C. Nicopoulos, Xiaoxia Wu, Yuan Xie, and N. Vijay. Variation-aware task allocation and scheduling for mpsoc. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 598–603, 4-8 Nov. 2007.

[100] Feng Wang, Guangyu Sun, and Yuan Xie. A variation aware high level synthesis framework. *To appear in Proceedings of Design Automation and Test in Europe (DATE)*, 2008.

[101] Feng Wang, Andres Tackach, and Yuan Xie. Variation-aware resource sharing and binding in high level synthesis. *Techincal Report*, 2008.

[102] Feng Wang, Yuan Xie, Kerry Bernstein, and Yan Luo. Dependability analysis of nano-scale finfet circuits. *IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2006)*, 2006.

[103] Feng Wang, Yuan Xie, and Hai Ju. Variability-driven module selection with joint design time optimization and post-silicon tuning. *in Proceedings of Asia-South Pacific Design Automation Conference (ASP-DAC)*, 2008.

[104] Feng Wang, Yuan Xie, N.Vijaykrishnan, and M.J.Irwin. On-chip bus thermal analysis and optimization. *Proceedings of the Design, Automation, and Test in Europe (DATE 2006)*, 2006.

[105] Feng Wang, Yuan Xie, R. Rajaraman, and B. Vaidyanathan. Soft error rate analysis for combinational logic using an accurate electrical masking model. *International Conference on VLSI Design*, 2007.

[106] W.H. Wolf. An architectural co-synthesis algorithm for distributed, embedded computing systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 5(2):218–229, 1997.

[107] Yuan Xie and Wei-Lun Hung. Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design. *J. VLSI Signal Process. Syst.*, 45(3):177–189, 2006.

[108] J. Xiong, V. Zolotov, C. Visweswariah, and N. Venkateswaran. Criticality Computation in Parameterized Statistical Timing. *ACM/IEEE international workshop on timing issues in the specification and synthesis of digital systems (TAU)*, pages 119–124, February 2006.

[109] T. Yuan, D. A. Buchanan, C. Wei, D. J. Frank, K. E. Ismail, L. Shih-Hsien, G. A. Sai-Halasz, R. G. Viswanathan, H. J. C. Wann, S. J. Wind, and W. Hon-Sum. Cmos scaling into the nanometer regime. *Proceedings of the IEEE*, 85(4):486–504, 1997. 0018-9219.

[110] Y. Zhan, A. J. Strojwas, M. Sharma, and D. Newmark. Statistical Critical Path Analysis Considering Correlations. *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, November 2005.

[111] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference*, pages 183–188, 2002.

Vita

Feng Wang is from China. He received BS and MS degrees from Fudan University and Ohio University, respectively. He is currently a PhD candidate in the Computer Science and Engineering Department at Penn State University. He works in the Microsystems Design Laboratory (MDL) in the Computer Science and Engineering Department. He has co-authored eleven conference papers, and three journal papers. He has also served as technical referee for journals and conferences in the field of VLSI design, Electronics Design Automation, including: IEEE Transactions on VLSI (TVLSI), IEEE Transactions on Computer Aided Design (TCAD), International Conference on Computer Aided Design (ICCAD), Design Automation Conference (DAC), Design Automation and Test in Europe Conference (DATE), etc.