

University of Passau

Delay Testing in Nanoscale Technology under Process Variations

Author:

Jie Jiang

Supervisor:

Prof. Dr. Ilia Polian

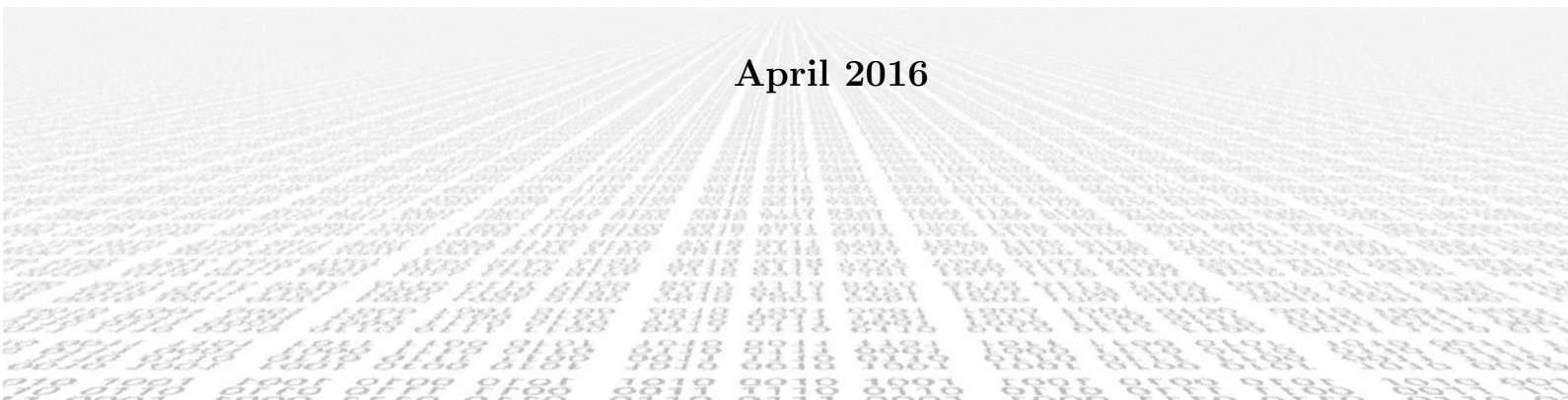
Prof. Dr. Joan Figueras

A thesis submitted in partial fulfillment for the
degree of Doctor of Computer Science

in the

Faculty of Computer Science and Mathematics
Department of Computer Engineering

April 2016



Declaration of Authorship

I, JIE JIANG, declare that this thesis titled, ‘Delay Testing in Nanoscale Technology under Process Variations’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Passau.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at the University of Passau or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

“Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.”

Albert Einstein

Abstract

In modern CMOS technology, process variations have significantly increased impact on the circuit behavior with continuously scaled transistor sizes. Manufactured devices tend to have different performances due to parameter variations during manufacturing and in the operating context. Conventional tests generated regardless of variations could fail to rule out devices with low performance and even functional failure caused by extreme variations; the unreliability in shipped products is in turn raised. To tackle the problem, many existing test approaches have focused on identifying and testing a number of critical paths in the circuit, and aimed at the efficiency of the searching process. However, the statistical circuit model, which better describes the circuit timing behavior under variations, is not yet sufficiently investigated and employed by existing testing methodologies.

This thesis work proposes Opt-KLPG and MIRID, which can be utilized by a statistical delay testing flow. Opt-KLPG—a **K Longest Paths Generation (KLPG)** algorithm for optimal solutions under memory constraints—can pin-pointedly generate tests for small delay defects, which are common small timing deviations under process variations, based on the traditional KLPG algorithm. In contrast to KLPG, Opt-KLPG guarantees the optimality of the solution (the K longest sensitizable paths indeed). MIRID is a mixed-mode timing-aware simulator, incorporating effects of power-supply noise and combining an event-driven logic simulation engine with interfaces to provided electrical models. MIRID aims at evaluating delay tests in presence of process variations efficiently yet accurately, by performing logic simulation at the gate level while determining the gate delays using simplified electrical modes. The electrical models applied by the simulator focus on the IR drop effect. Electrical parameters mainly contributing to the effect are incorporated into the model. The simulator is generic and flexible to be adapted by modifying the interfaces with minor effort. Both applications were verified in various aspects by experiments for academical/industrial circuits, and turned out to have satisfiable effectiveness and performance.

Acknowledgements

Firstly, I would like to express the deepest appreciation to my advisor Prof. Dr. Ilia Polian for the continuous support of my Ph.D study and related research, for his patience, motivation and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to offer my special thanks to Prof. Dr. Joan Figueras to be the second supervisor of this thesis.

My sincere thanks also goes to my collaboration partners: Prof. Dr. Michel Renovell, Prof. Dr. Mariane Comte and Dr. Marina Aparicio Rodriguez from Montpellier; Prof. Dr. Bernd Becker, Dr. Matthias Sauer, Dr. Alexander Czutro from Freiburg; and Prof. Dr. Hans-Joachim Wunderlich, Michael E. Imhof and Abdullah Mumtaz from Stuttgart, not only for their precious collaboration but also for their insightful comments and encouragement.

Last but not the least, I would like to thank my family: my parents Yanfen Song and Ping Jiang, and my husband Yujiong Chen, for supporting me spiritually throughout writing this thesis and my life in general.

Contents

Declaration of Authorship	i
Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	x
Abbreviations	xi
1 Introduction	1
1.1 Background and State-of-the-art	2
1.1.1 Process Variation in Nanoscale Technologies	2
1.1.2 Delay Testing Under Process Variations	5
1.1.2.1 Delay Fault Models	6
1.1.2.2 Timing Analysis	8
1.1.3 State-of-the-art Approaches	8
1.1.3.1 Statistical Design Approaches	9
1.1.3.2 Statistical Testing Approaches	10
1.2 Statistical Testing flow	11
1.2.1 Overview of a Statistical Testing Flow	11
1.2.2 Statistical Fault Coverage	12
1.2.3 SDD Tesing and KLPG Algorithm	14
1.2.4 Variation-aware Fault Simulation	16
2 KLPG: K Longtest Path Generation	19
2.1 Motivation	19
2.2 Preliminaries	21
2.2.1 Classification of Path Delay Faults	21
2.2.2 SAT-solver and SAT-based ATPG	24
2.3 Walker’s KLPG Algorithm	25
2.4 Opt-KLPG: K Longest Path Generation for Optimal Solutions Under Memory Constraints	27
2.4.1 Algorithm Overview	28
2.4.2 Esperance Computation	30

2.4.3	KLPG Module	35
2.4.4	Sensitization check	38
2.4.4.1	Direct Implication	39
2.4.4.2	ATPG using TIGUAN	42
2.5	Application of Opt-KLPG by Variation-aware Fault Grading	45
2.5.1	Modeling for Variation-aware Delay Testing	45
2.5.2	Variation-aware Fault-grading Procedure	46
2.5.3	Path-oriented Delay ATPG	47
2.6	Experimental Results	48
2.6.1	On the Optimality of KLPG Algorithms	48
2.6.2	Fault-grading Procedure using Opt-KLPG	53
3	MIRID: Mixed-mode IR Drop Induced Delay Simulator	56
3.1	Motivation	56
3.1.1	Mixed-mode Simulation	58
3.2	Preliminaries	59
3.2.1	Power Distribution Network (PDN)	59
3.2.2	IR Drop Effect	60
3.2.3	Power-aware Test	61
3.3	Simulation Overview	63
3.3.1	Event-driven Logic Simulation	63
3.3.2	Interfaces to Electrical Models	65
3.4	Electrical Models	68
3.4.1	PDN Configuration	68
3.4.1.1	Structural Assumption	68
3.4.1.2	Parasitic Elements in PDN	70
3.4.1.3	Mapping to the Logic Block	71
3.4.2	Current Distribution in PDN Grids	72
3.4.3	Electrical Models at the Gate Level	76
3.4.3.1	Dynamic Current Model	77
3.4.3.2	Gate Delay Model	80
3.5	Simulator	81
3.5.1	Logic Library	82
3.5.2	Simulation Preprocessing	87
3.5.3	Logic Simulation	90
3.5.3.1	Zero-delay Logic Simulation	91
3.5.3.2	Timing-aware Event-driven Simulation	93
3.5.4	Interface Functions For Electrical Models	98
3.5.4.1	Estimation of Dynamic Currents	98
3.5.4.2	Distribution of Currents in PDNs	100
3.5.4.3	Estimation of Gate Delays	102
3.6	Application to Self-adaptive Better-than-worst-case Design	103
3.6.1	Short-path Invalidation	104
3.6.2	Detection Conditions	107
3.6.2.1	Simplified Variability Model	107
3.6.2.2	Mitigation by Buffer Padding	108
3.6.2.3	Mitigation by Latch Placement	109

3.6.3	Strategies of Mitigation	112
3.7	Experimental Results	113
3.7.1	Accuracy Validation	113
3.7.2	Investigation on IR Drop induced Delay	120
3.7.3	Performance for Large Circuits	123
3.7.4	Mitigation of the Short-path Problem using Buffers and Latches	125
4	Conclusion	128
 Bibliography		 134

List of Figures

1.1	Overview of the variation classification	3
1.2	Overview of the statistical test flow [1]	11
2.1	An example of the longest path	20
2.2	Testability classification of PDFs [2]	22
2.3	An example of the invalidation by a non-robust test	23
2.4	Opt-KLPG overview	28
2.5	Path store initialization	29
2.6	PERT delays under different delay assignments	30
2.7	Computation of PERT delays for a signal driving a fan-out	31
2.8	Iterative computation of PERT delays	34
2.9	KLPG module	36
2.10	New Path submodule	37
2.11	KLPG flowchart	38
2.12	Examples of direct implication	40
2.13	An example of sensitization check using direct implication	40
2.14	A false path that cannot be identified using direct implication	43
2.15	An example of CMS@ derivation	44
2.16	Experimental results for circuit p35k. Runtimes are shown by lines; number of gates with shorter paths are shown by bars.	53
3.1	A two-layer power distribution grid	60
3.2	Schematic illustration of the simulator	65
3.3	Two sets of Vdd and Gnd lines	69
3.4	PDN grids	70
3.5	Gates connected to Vdd and Gnd power supply nodes	72
3.6	Current distribution in Vdd PDN	73
3.7	Central, corner and band areas in a 100×100 grid	74
3.8	A high-level view of the power grid over a chip with 9 blocks	75
3.9	Supply and input voltage swings and gate delay	80
3.10	Definition of the <i>gate</i> object	83
3.11	Definition of the <i>signal</i> object	84
3.12	Definition of the <i>pin</i> object	84
3.13	A circuit containing an inverter and two NAND gates	85
3.14	Examples of constructed gate objects	85
3.15	Examples of constructed signal and pin objects	86
3.16	Symbolic presentation of operations for matrices in different areas in PDN	88
3.17	Event-driven simulation applying an event queue	95

3.18	Calculation of voltage level at PDN node $N(i, j)$	99
3.19	Update current array $current'$	102
3.20	Short-path invalidation in Razor design	105
3.21	Mitigation by latch placement	106
3.22	Input-to-latch invalidation	110
3.23	Strong latch-to-output invalidation	110
3.24	Weak latch-to-output invalidation	111
3.25	14-inverters model for validation	114
3.26	Current validation for 14-inverters chain	115
3.27	Voltage validation for 14-inverters chain	115
3.28	Current validation for c17	117
3.29	Voltage validation for c17	118
3.30	Current validation for c17 with 10^{-1} ps precision	119
3.31	Voltage validation for c17 with 10^{-1} ps precision	119
3.32	8-inverter chains connected to a 2×2 grid	121
3.33	Model of 9 inverter chains	122
3.34	Correlation of the IR drop induced delay and X	122

List of Tables

2.1	Experimental results under functional sensitization criterion	52
2.2	Comparison of KLPG and Opt-KLPG under different model assumptions for circuit c7552	53
2.3	Comparison of KLPG and Opt-KLPG under different model assumptions for industrial circuit p45k	53
2.4	Statistical fault coverages for 101 instances, 100 random faults and 9 fault sizes	55
3.1	Difference of currents in 14-inverter chain reported by MIRID and SPICE	116
3.2	Difference of voltages in 14-inverter chain reported by MIRID and SPICE	117
3.3	Difference of current and voltage at $N(50, 50)$ in c17 reported by MIRID and SPICE	119
3.4	Simulation for ISCAS and NXP circuits	124
3.5	Experimental results for buffer padding and latch placement	126

Abbreviations

ATPG Automatic Test Pattern Generation

VLSI Very-Large-Scale Integration

CMOS Complementary Metal Oxide Semiconductor

VDSM Very Deep Sub-Micron

CUT Circuit Under Test

SDD Small Delay Defect

IC Integrated Circuit

TF Transition Fault

PDF Path Delay Fault

KLPG K Longest Paths Generation

PDN Power Distribution Network

MIRID Mixed-mode IR-drop Induced Delay Simulator

BUT Block Under Test

SAT Boolean Satisfiability Problem

CNF Conjunctive Normal Form

MLR Multiple-Linear Regression

NRMSE Normalized Root Mean Square Error

IFA Inductive Fault Analysis

Chapter 1

Introduction

Reliability is a very important issue in **Complementary Metal Oxide Semiconductor (CMOS)** manufacturing. Devices delivered to customers should operate not only correctly as required by the specification, but also reliably during the whole life time and in all expected operation conditions. However, nothing is perfect—due to the intrinsic physical limitations of the fabrication process, the uncertainties in the parameters of fabricated devices and in the operating environment during the lifetime are not avoidable; the situation is even getting more severe as technology scales. These uncertainties contribute to the source of *process variations*—variations in parameters of transistors or interconnects—and in turn impair the performance of devices or even cause operation failures.

In today's semiconductor manufacturing technology, up to 30% variation in operating frequency and 5 to 10 times variation in leakage power are expected [3]. Due to the effect of variations, the behavior of the fabricated design in terms of performance and power differs from what designer intended; the unreliability in the design rises. Numerous solutions that require change in design methodologies, such as adaptive body bias [4], or self-adaptive technique to lower the rate of dynamic errors [5], have been proposed to combat the variations. In context of testing, the impact of variations is also significant. Traditionally, test for **Integrated Circuit (IC)** uses fault models that abstract from defect-specific parameters affecting the circuit behavior. However, in presence of process variations, defects can manifest themselves differently, especially in affecting circuit timing; tests targeting the abstracted fault could fail to detect these defects. The

increased impact of process variations could lower the defect coverage and in turn impair the reliability of the device under test. Though a number of testing approaches have been proposed and contributed to raise the defect coverage under process variations [6–12], a statistical testing flow, which involves fault modeling, fault simulation and test generation with consideration of process variations, has not been sufficiently investigated.

The motivation of this thesis work is to propose variation-aware **A**utomatic **T**est **P**attern **G**eneration (ATPG) and simulation methodologies that could be integrated into the statistical testing flow. The objective is two-fold: on one hand, the methodologies have to be capable of dealing with benchmark/industrial circuits and thus efficiency is an important concern; on the other hand, to increase the defect coverage the impact of process variations need to be taken into account by the methodologies, which in turn requires the accuracy in estimating the impact. The remainder of this chapter is organized as follows: Section 1.1 introduces background knowledge and the state-of-the-art approaches; Section 1.2 presents a statistical testing flow with test methodologies that are especially of interest and could be employed by the flow.

1.1 Background and State-of-the-art

1.1.1 Process Variation in Nanoscale Technologies

The uncertainty in device parameters, which is called manufacturing-related variation, is caused by equipment imprecision and process limitations. As CMOS technology continues to scale, manufacturing process is getting more complex while process precision control needs to remain relatively accurate at the same time. As a result, a number of steps throughout the manufacturing process are prone to fluctuations [13], e. g., optical proximity effects (variations in the linewidth of a feature as a function of the proximity of other nearby features) can be caused by patterning features smaller than the wavelength of light [14]. These manufacturing imperfections can result in variations in physical parameters of devices and interconnects, such as gate length, gate-oxide thickness, channel doping concentration, and the thickness and height of interconnects. Furthermore, variations in physical parameters cause variations in electrical characteristics of

devices and interconnects, such as the threshold voltage, the drive strength of transistors, and the resistance and capacitance of interconnects, which finally affect the circuit timing behavior. It is important to note that correlations can exist between electrical variations that depend on common physical parameters. Also, physical parameters can be correlated themselves under the impact of a particular equipment variation. Analysis to determine the variations and correlations in physical parameters is very complex and impractical due to the high number of equipment-related parameters. Hence, physical parameters are often assumed to be random variables that are independent or have well-understood correlations by statistical timing analysis (analysis that determines the timing behavior of circuit elements statistically) [13].

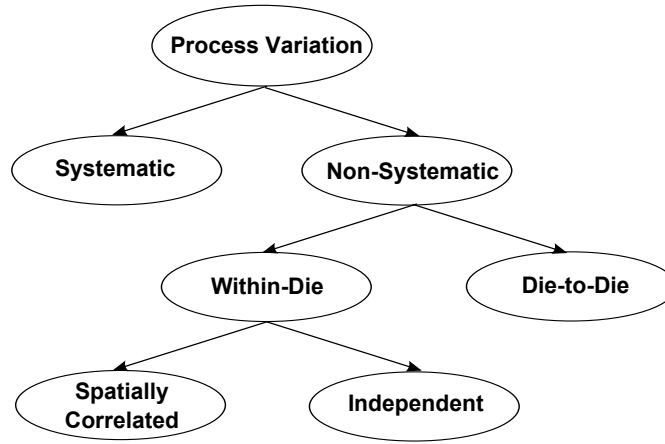


FIGURE 1.1: Overview of the variation classification

Manufacturing-related process variations can be further divided into two categories: *systematic* and *non-systematic*, as presented in Figure 1.1. Systematic variations are components of physical parameter variations that follow a well-understood behavior and can be predicted by analyzing the designed layout, e. g., the optimal proximity effects. Nonsystematic or random variations represent the truly uncertain component of physical parameter variations, such as line edge roughness and random dopant fluctuations. Nonsystematic variations can be further divided into groups of *die-to-die* and *within-die* variations according to their act on different spatial scales. In past technologies, it was sufficient to model die-to-die variations while in modern nanoscale technologies, within-die variations have become equally important [15]. Finally, within-die variations can be categorized into *spatially correlated* and *independent* variations. Many underlying processes tend to affect closely spaced devices in a similar manner, making these devices more likely to have similar characteristics than those placed far apart. The component

of variation that exhibits such a spatial dependence is known as spatially correlated variation. The residual variability of a device that is statistically independent from all other devices is referred to as independent variation. It has been observed that the contribution of independent within-die random variations is increasing with continued process scaling [13].

The other source of process variation is the uncertainty in the operating environment of a particular device during its lifetime, represented by variations related to the operating context, such as temperature, supply voltage and lifetime wear-out. These variations can depend on the functionality of individual circuit block and the applied input sequences. For example, the power density across a die varies for different rates of activities in a circuit block. Higher activity in a block puts more demand on the power distribution network, resulting in resistive and inductive voltage drops; the time-dependent supply voltage variation is in turn created. Moreover, the high rate of activities in a block raises the temperature in this area and creates the so-called hot spots, which in turn results in the temperature variation across the die. Also, aging has a significant impact on the transistor performance. Studies have shown that a transistor's saturation current degrades over years because of oxide wear-out and hot-carrier degradation effects [3]. This degradation is expected to become worse as the transistor size continues to shrink.

Unlike the dynamic sources of variations that exhibit themselves during the operating cycle, such as voltage supply and temperature, variations caused by manufacturing imperfections and aging mechanisms are static or quasi-static, that is, they could exhibit long time constants. With respect to the exhibited time constant, the sources of process variations can be briefly divided into two groups from the time aspect:

- **Physical factors** including variations in the electrical and physical parameters characterizing the behavior of devices, mainly caused by manufacturing imperfections and various wear-out mechanisms. They exhibit long time constants, typically measured in years.
- **Environmental factors** including power supply voltage and temperature. These factors highly depend on the design and are typically measured by time constants in similar scale to the clock frequency.

1.1.2 Delay Testing Under Process Variations

The timing of **Very Deep Sub-Micron (VDSM)** devices is considerably affected by process variations. As reported in [16], defects that cause delay have typically represented 1% to 5% of the total defect population observed. Delay defects can cause delayed signal propagations exceeding the maximum operating frequency, i.e., the circuit fails to operate in time when these signal propagations are required. Under impact of process variations, which generally do not have fixed values and usually follow a probability distribution, delays induced by the variations will also follow a similar distribution and vary between different circuit instances.

To avoid the overall delay behavior of devices, raising the operation frequency is impractical because it contradicts the requirement for high performance. Typically, manufacturers will choose an operating frequency of the product to meet requirements from the customer and the market, as well as to make the economic trade-off between the yield loss below the operating frequency and the higher performance of the product. To screen the units that could operate slower than the desired frequency, the speed of the paths with the longest delays, called *critical paths*, are often tested under worst-case conditions.

The objective of tests is to rule out defective parts from manufacturing devices. which means, the tests should be designed such that by observing the responses from the **Circuit Under Test (CUT)** must lead to detection of the defects [17]. Classically, most delay testing techniques apply Boolean logic values to the CUT and observe its response at some clock frequency. Some delay defects, especially those resulting from process variation, can also be detected through analysis of the response of process monitors¹ [18]. For the mostly used Boolean delay-defect testing, two major related questions can arise:

- What kind of test patterns should be applied?
- An what clock frequencies to apply the test patterns and at what clock frequencies to capture the output responses?

¹Process monitors are carefully designed special circuit structures monitoring manufacturing-process characteristics, such as ring oscillators and trees of NAND gates with controlled delays.

However, due to the fact that an extraordinary volume of defects can exhibit themselves in a circuit (and not all possible defects may be known), it is almost infeasible to derive appropriate tests for defects individually. A fault model that extracts the faulty behavior common in the target defects is therefore used, to facilitate analysis of defects and the derivation of tests. Targeting on the modeled faults, dedicated tests can be generated, with time information when to apply the test and when to observe the output.

1.1.2.1 Delay Fault Models

Two fault models are often used by delay testing: the **Transition Fault (TF)** model and the **Path Delay Fault (PDF)** model [2]. Traditionally, the TF model is often used by the industry for delay testing. It assumes the delay caused by the defect to be larger than the specification time, which usually refers to the system clock period. Test generation for such a fault model takes advantage of the rather low number of faults, which is linear to the circuit size. Moreover, existing tools that test and simulate stuck-at faults—a very common and mostly used fault model—can be used with minor modifications to generate tests for TFs. However, the TF has significant limits. Under assumption of the defect size larger than the clock period, any path through the fault site can be selected to propagate the transition to the output. Indeed, ATPG tools, which are very often used by test generation, tend to seek the easiest path to excite the fault and propagate the fault effect. It implies that the selected path has relatively few gates and can be very short. Then, for defects with smaller sizes, that is, the extra delay caused by the defect is shorter than the clock period, propagation of the delay along such a short path is likely to finish during the clock period, resulting in the capture of a correct logic value at the output, i.e., the target defect escapes detection.

In fact, modern manufacturing process tends to produce defects inducing small delays. Many electrical phenomena in VDSM process technologies, such as process variations, crosstalk noise, power-supply noise, resistive shorts and opens, can induce small delay variations in the circuit components [19]. Such small delay variations are referred to as **Small Delay Defect (SDD)** [20, 21], which have received increased attentions over the last few decades. As reported in [22, 23], the distribution of delay-related failures is skewed toward the smaller delays. That is, small delay defects contribute to the failure

of the majority of devices that fail due to delay defects. Targeting these SDDs can lower the test escape rate and thus improve the reliability of the shipped products.

Obviously the TF model, which assumes the delay exceeds system clock period, does not match the faulty behavior caused by a SDD properly. PDF is more preferred by testing SDDs as it assumes that a path propagates the desired transition with a total amount of time exceeding the specification time. The small delays could be detected by exercising some paths through them so that the total path propagation delay exceeds the clock period. Thus, searching for such paths are often required by testing small delays, i.e., paths through the defect site that most probably propagate the small delays to the observation point are searched, and sensitized by tests under certain sensitization conditions. The sensitization condition is often related to the timing variations of side-inputs—inputs that are not on the path and drive the on-path gates, which can affect the fault detection. To avoid such situations, tests that guarantee the detection of delay faults regardless of timing variation in the rest of the circuit, called *robust* tests, are targeted. Robust tests are very powerful because they detect the target fault independent of the presence of other delay faults. However, as indicated in [24], robust tests only exist for a relatively small part of delay faults in today's **Very-Large-Scale Integration (VLSI)** circuits. Thus, non-robust tests should also be targeted by the test generation and applied with robust tests to raise the defect coverages totally. It should be noted that PDF is the fault model that can be used for testing SDD; where and how the delays are distributed on the path are not concerned by the model itself while a SDD is related to a specific defect site. The PDF model takes advantage of the scalability of delay sizes of detectable defects. Nonetheless, for VLSI circuits the test generation using PDF model could be very time-consuming since the number of paths is normally very large and even exponential to the circuit size in worst case.

Tests that detect a PDF need to launch the desired transition—assignments of opposite logic values in two consecutive clock cycles—at the input of the path, and propagate the transition through the path to its output. Furthermore, the propagation of the transition to the output implies a necessary condition that the output value settles in a logic value opposite to its initial value. Depending on sensitization condition for the test, such as a robust or non-robust test, more value constraints for signals, especially the side-inputs, need to be considered by the test generation. Moreover, a large part of the PDFs can be functionally redundant, i.e., no tests exist that propagate the transition along the

path. For example, it has been shown that up to 81% PDFs in ISCAS'85 circuits are functional redundant [25]. Thus, an efficient identification of redundant PDFs could help to improve the performance of test generation significantly.

1.1.2.2 Timing Analysis

Timing analysis that characterizes the timing behavior of circuit elements is very important for an accurate delay estimation required by timing-aware testing methodologies. It is performed during the design flow and can be static or dynamic. Static Timing Analysis checks static delay requirements of the circuit independent of input vectors whereas dynamic timing analysis verifies the functionality of the design without any timing violations for applied input vectors by simulating using timing information [13, 26].

In the past few decades, static timing analysis has been a widely adopted tool in the VLSI design and served as the timing engine in test generation, mainly because of its efficiency and conservatism in the sense that it takes the worst-case into account. However, static timing analysis is deterministic and calculates the circuit delay for a specific process condition, i.e., all parameters that affect the circuit timing are assumed to be fixed and are uniformly applied to all the devices in the design [13]. As process variations have become significantly more pronounced in nanoscale regime, the conventional characterization of circuit elements with fixed delays becomes inappropriate to be used. A *statistical* timing assignment [13, 26, 27] is preferred since process variations and their influences in circuit timing are statistically distributed in nature.

The need for an effective modeling of process variations in circuit timing has led to a number of statistical approaches, such as numerical integration over the process-parameter space to compute the delay of critical paths [28], path-based statistical timing analysis considering spatial correlations [27], and statistical dynamic timing analysis to effectively identify false paths [26].

1.1.3 State-of-the-art Approaches

Variability in circuit parameters has become a very important concern from both design and test sides as technology scales into the nano-regime. Designing for the worst-case conditions is unrealistic as it would contradict the performance requirement, and lead

to excessive area and power consumption. Thus, design approaches are often based on statistical timing assumptions, and aim at mitigating the effects of variability. However, even with the most sophisticated approaches, the electrical models employed are incapable of exactly representing the device performance, which means that there is always the possibility of variations being outside model predictions [17]. The reason is that causes of variability could be very sophisticated and depend on manufacturing technologies; and even if the causes are well understood, predicting the results of variations on devices is a serious challenge. Moreover, as variation has a continuous distribution, situations are inevitable that parameter variations will produce a small difference in specifications, among which circuit delay is particularly interested, from allowable extremes [17]. In brief, variation effects cannot be exactly predicted by design models and totally compensated. Moreover, delay variations in devices that could lead to malfunctions are inevitable. Test methodologies are therefore required by high-quality manufacturing, to rule out such devices that could present delay defects under process variations. In this section, the state-of-the-art approaches are briefly reviewed, in both the design (1.1.3.1) and the test areas (1.1.3.2).

1.1.3.1 Statistical Design Approaches

Numerous works have been contributed to the statistical-based design that considers the impact of process variations, both at the gate level [29] or at a high level [30]. Moreover, a number of techniques have been used to mitigate the impact of process variations in modern technologies [31]. The impact of process variations can be reduced by tuning certain operating parameters, such as the threshold voltage (“body bias”), the supply voltage and the frequency. Based on methods that adjust these parameters, self-calibration designs are proposed to compensate and correct the performance deviations induced by process variations adaptively [4, 32, 33]. Dynamic calibration during the operating process often requires additional hardware (e. g., sensors), and mechanisms of fault detection and correction. For example, the RAZOR processor [5] scales the supply voltage dynamically during the process according to the computed error rate in terms of the frequency of the delayed transitions. For the purpose of monitoring and correcting errors, the processor applies extra memory elements clocked later than the system clock to capture the correct values that are possibly delayed. A similar fault-tolerant architecture that combines latch-based design and time redundancy is presented in [34].

Also, other circuit techniques for variability mitigation have emerged recently, such as clock-tuning in Intel Montecito processor [35], the adaptive and self-healing architecture ElastIC [36], and the variation-tolerant network design using self-calibrating links [37].

1.1.3.2 Statistical Testing Approaches

Existing test approaches focus on identifying a number of testable critical paths for each possible fault site and generating tests that excite the transition propagated along the paths. These paths are commonly the longest testable ones given a static timing assignment, which implies, they have the minimum timing slack between the path propagation time and the operating frequency and thus most probably delay the propagated transitions under process variations. As the search space of paths for large circuits can be very huge, many approaches aim at the efficiency of the searching process, such as by pruning paths from the search space based on structural correlation [7, 38] and process-variation correlation [6]. Moreover, a number of approaches have explicitly considered process variations during test generation, usually based on statistical timing analysis that describes the timing behavior of circuit elements under process variations. In [8] an approach was proposed that intelligently selects n -detection test patterns (tests that detect each target fault at least n times) based on a gate-delay defect probability model. More sophisticated methods that employ a suitable defect-coverage metric were used to guide the test generation process. Park et al. were the first to propose the concept of statistical delay-fault coverage [9]. In [10] the impact of statistical variations including specific noise effects on circuit performance was analyzed, and further considered by a novel defect-driven path selection methodology for test generation. Shintani et al. introduced a metric called *parametric fault coverage* that was utilized to guide an adaptive test flow [11]. A process variation-aware test generation method targeting on resistive bridging faults was proposed in [12], which uses a metric called test robustness to quantify the impact of process variation on test quality.

1.2 Statistical Testing flow

1.2.1 Overview of a Statistical Testing Flow

Though variability has been considered by many existing testing approaches, a testing flow based on statistical timing analysis, which involves fault modeling, fault simulation and test generation, has not yet been sufficiently investigated. The basic concept of a statistical test flow [1] is illustrated in Figure 1.2. The “statistical” concept can be adopted by approaches at various abstraction levels of the test flow, from the fault modeling at electrical level to test strategies at system level.

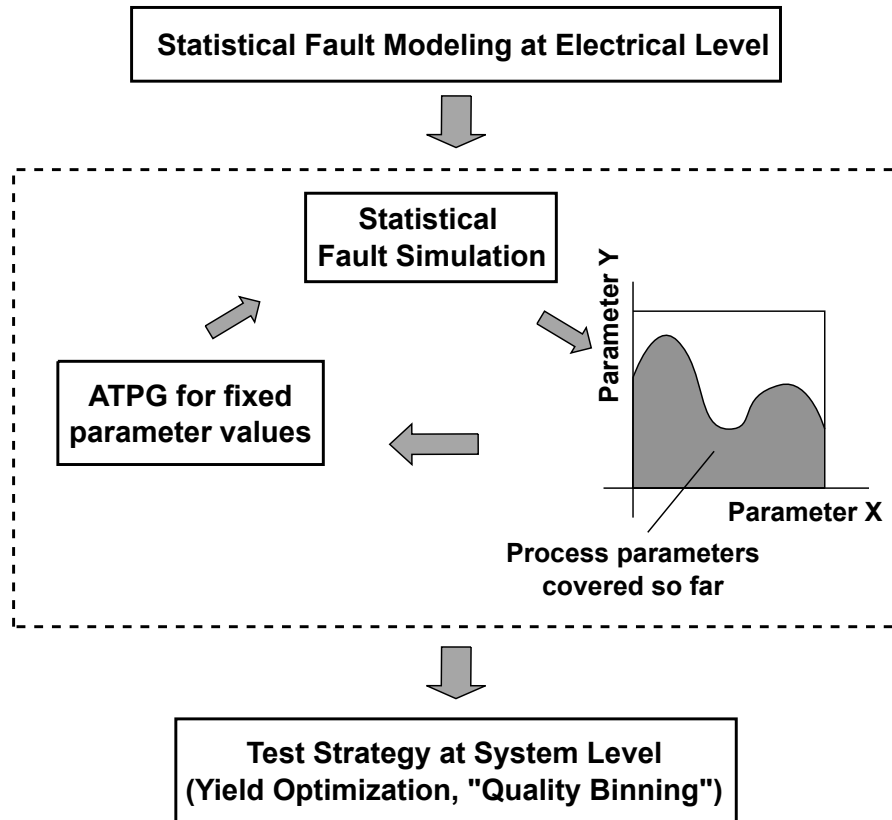


FIGURE 1.2: Overview of the statistical test flow [1]

The statistical fault modeling can be obtained by a systematic approach for a primitive cell library with injected defects [1]. In this approach electrical fault simulations with randomly changing circuit parameters were performed to achieve the delay distribution for injected defects, i.e., the statistical characterization of library cells in terms of the timing behavior for an injected defect approximated across all parameter variations.

Regarding the test flow at the gate level (bounded with dashed lines in Figure 1.2), due to the complexity of test generation, ATPG for fixed parameters values can be combined to the statistical fault simulation, to cover the whole parameter space incrementally. The fault coverage reported by the simulation should also be statistically defined, to evaluate the test quality under process variations and guide the iterative test flow.

1.2.2 Statistical Fault Coverage

Classically the fault coverage FC is defined as in Equation 1.1.

$$FC := \#detected\ faults / \#modeled\ faults \quad (1.1)$$

Under the impact of process variations, the new metric of fault coverage should be defined based on the configuration of process parameters and the corresponding statistical fault modeling.

Provided with manufacturing data, a finite list of parameters that could have impact on the timing-behavior of circuit elements and the range of possible values for each parameter are modeled by $\{P_1, P_2, \dots, P_N\}$ and Δ_i ($1 \leq i \leq N$), respectively. Due to the complexity of modeling possible correlations between process parameters, the parameters are assumed to be uncorrelated.

Given a circuit represented by a set of gates and interconnect lines between gates, a delay assignment for the circuit refers to the estimated propagation time for each gate, usually called gate delay. (The interconnect delay is not considered by this thesis work.) The gate delay can differ for different gate inputs and input edges. Thus, it can be modeled by the propagation time from a gate input to the output for a rising or falling transition. For each combination of parameter values $p := (p_1, p_2, \dots, p_N) \in P$ where P is the parameter space ($P := \Delta_1 \times \Delta_2 \times \dots \times \Delta_N$), the circuit exhibits the corresponding timing behavior, given by the delay assignment. A circuit instance that affected by a variation combination p is denoted by C_p . Moreover, assuming a normal or well-understood distribution of parameter variations, the probability density function of a possible combination of parameter values p can be derived, denoted by $\pi(p)$.

Then, given a delay fault $f := (g, s)$ where gate g is the fault location and s is the delay size (that could refer to a gate input and the transition edge depending on the delay model), f is detected in C_p by a test set T if at least one test leads to an incorrect response in C_p at time point t_{obs} , where the correct response is given by the fault-free nominal circuit (in absence of variations) at time t_{obs} . The observation time t_{obs} is commonly chosen such that all circuit outputs have stabilized after the test pattern is applied. Considering process variations, t_{obs} equals to the maximal delay of the circuit in absence of variations, multiplied with a safety margin greater than 1. The detectability of f by T in circuit instance C_p is denoted by $det_{C_p}(f, T)$ ($det_{C_p}(f, T)$ equals to 1 if f is detected and 0 otherwise). Finally, using π_p to denote the probability of an actual manufactured circuit having the parameter configuration p , the *circuit coverage* of f by T is determined by Equation 1.2. In contrast to the conventional fault coverage (Equation 1.1), i.e., the percentage of faults detected by a test set in a representative circuit with fixed parameter values, Equation 1.2 describes the percentage of the manufactured circuit instances with distributed parameter values in which the test set detects a given fault.

$$CCcov(f, T) = \int_{p \in P} det_{C_p}(f, T) \pi(p) dp \quad (1.2)$$

Moreover, according to the statistical fault modeling, the delay behavior of a circuit element is given by a probability density function of delay sizes, denoted by $\rho(s)$. Considering a continuous range of sizes S , the fault coverage of f with all possible delay sizes can be statistically defined by integration of the circuit coverage over the delay size s , as given in Equation 1.3.

$$FC(f, T) := \int_{s \in S} CCcov(f, T) \rho(s) ds \quad (1.3)$$

The statistical test flow must try to maximize the fault coverage and generate (compact) test sets identifying the fault in as many circuit instances as possible. For this purpose, statistical fault simulation is performed iteratively for each combination in the parameter space. For the sake of simplicity, only the space for two parameters X and Y is illustrated in Figure 1.2. The area marked by the shadow represents the parameter combinations covered so far. The variation-unaware ATPG is invoked with fixed parameter values to cover a further parameter combination in the space. This flow is iterated

until an acceptable coverage of the complete range is achieved and can be followed by a compaction of the obtained test set.

Using the finally obtained tests, which are of high quality and variation-aware, it is possible to explore test strategies at system level, such as to separate circuit instances into classes, or called “bins”, according to the frequency and voltage they can handle or according to the robustness of a system. Nevertheless, such systematic approaches are not focused by this thesis work. The test approaches at the gate level including ATPG for small delay defects and variation-aware fault simulation are more interested, which also motivate this thesis work.

1.2.3 SDD Tesing and KLPG Algorithm

The scaled technology tends to bring an increasing number of small delays defects, which puts more importance on SDD testing. An SDD is only tolerable if all functional paths (those paths could be exercised by normal circuit operations) that go through the defect site excite a delayed transition (slow-to-rise or slow-to-fall) at the site, and finish propagating within the clock period. Obviously, under assumption of static timing assignment, generating tests for the longest path through the target defect will allow detection of the defect with the minimum possible size; no other paths through the defect site need to be considered.

However, depending on the accuracy of the timing analysis, and mostly due to the impact of process variations, the selected path may not be the longest indeed. An alternative approach is to test all functional paths through the defect site, which is however impractical for large circuits when all possible delay defects are targeted. It has been observed that the contribution of within-die random variations not spatially correlated is increasing with continued process scaling [13], which implies that delay defects caused by random variations can appear anywhere in a circuit. However, even a circuit of a reasonable size can contain billions of paths [16]. It is not possible to test and simulate all of them due to the time and cost constraints. As a compromise, test for a number of longest paths that go through the SDD is of interest. Similar to the idea of “n-detection”, using tests generated for multiple longest paths raises the probability of detection.

Though it is easy to find the structurally longest paths by graph-traversal algorithms, many found paths are unsensitizable (or called false paths), i.e., no tests can be found to propagate desired transitions along these paths, and hence useless in practical applications. An efficient algorithm that identifies the longest *sensitizable* paths, which implies test generation for these paths (modeled as PDFs), is therefore required to tackle this problem. The test generation for PDF is very challenging because a large number of logic interdependency constraints have to be satisfied for the path sensitization. Such an algorithm that aims at identification of K (multiple) longest sensitizable paths through a given defect site, is called KLPG algorithm.

ATPG algorithms can be applied to verify the path sensitization by means of generating tests for the path modeled as PDF; the false path is identified if no tests can be found. Using traditional ATPG algorithms, tests are searched by means of assignments and justifications of signal values during the propagation of fault effect (D-Algorithm [39]); an implicit enumeration approach using path propagation constraints to limit the search space and backtrace (PODEM algorithm [40]); or approaches that improve the efficiency of the search by reducing the number of backtraces [41, 42]. Moreover, it has recently been shown that for several classes of faults a SAT-based ATPG [43–45], by which an ATPG problem is reduced to a Boolean **S**atisfiability Problem (**SAT**) instance and solved using a SAT solver [46], outperforms the structural approaches. Especially for the fault classes that could contain a large part of redundant faults (faults that are not testable), SAT-based ATPG is very efficient in contrast to the structural approach. The redundant faults can be quickly identified by SAT-based ATPG that takes techniques to prove the unsatisfiability of the Boolean formula. The performance of SAT-based ATPG can be significantly improved by utilizing learning strategies [47]. It is also possible to support non-standard fault models by a SAT-based ATPG tool, e. g., TIGUAN [44], by which tests are generated for conditional multiple stuck-at fault model that can be converted from non-standard faults. In [45], a SAT-based framework is proposed that can be integrated into the KLPG application by mapping the path search problem to SAT problem while the timing information is encoded directly into the SAT formula. As gate delays are encoded as integer numbers by the application, rounding errors of real-valued delays are inevitable; rounding strategies have to be applied to mitigate the error.

The conventional approach of KLPG [38] is to explore the structural search space containing sub-paths. As long as a sub-path is proved to be a false path, the sub-space containing all extensions of this sub-path is trimmed off immediately. Sub-paths in the search space are sorted by their largest possible length of a complete extension. Thus, the sub-path that can be potentially extended to the longest path will be tried first. The sensitizability of each extension from a sub-path to the next gate is verified. As long as the extension is proved to be incapable of propagating the desired transition without causing any conflict, the extended sub-path is eliminated from the search space; as a result all possible extensions from this path will not be considered. The next sub-path in order is taken into account. The verification can apply specific techniques that may efficiently identify the potential false paths but not all of these. To ultimately verify the sensitization of a completely extended path, ATPG algorithms can be used to generate tests that sensitize the path. A problem with the structural approach is that the number of sub-paths in the searching space can increase exponentially during the searching process. A size limitation has to be imposed to the searching space to ensure the efficiency of the application for testing large circuits. The arbitrary size can have an impact on the optimality of the solution since overflows beyond the limited space, among which the optimal solution can exist, are out of consideration. In presence of process variations, the possible sub-optimal solution of KLPG can cause a worse rate of defect escape. In this thesis work, an algorithm called Opt-KLPG is proposed. It aims at investigating the impact of the size of the search space on the optimality of the KLPG solutions. Moreover, it provides an iterative procedure that ensures all sub-paths will be considered and thus an optimal solution is guaranteed. Chapter 2 details the algorithm and presents experimental results for academic/industrial circuits.

1.2.4 Variation-aware Fault Simulation

Depending on the complexity of ATPG, the effectiveness of generated tests could be limited in terms of the ability of detecting target faults and the test application time proportional to the number of applied tests. High-quality and possibly compact test sets are preferred to minimize the cost of applying tests, which can be translated into the timing and memory requirements of test equipment, and to avoid the cost due to delivering defective or unreliable devices escaped from the tests. To obtain high-quality tests, fault simulation [2, p. 75] is required to efficiently evaluate the tests and report

the fault coverage, based on which the tests could be chosen. Moreover, test sets can be compacted without sacrificing the fault coverage before applied to actual devices.

Fault simulation commonly consists of two logic simulation runs that evaluate the logic functions of gates in their structural order, in the fault-free circuit and in the same circuit with the injected fault (by simulating the fault effect). The detection of the fault is determined if the difference in at least one output of both simulation runs is observed. In context of delay testing under process variations, the fault simulation needs to estimate the timing behavior of circuit elements affected by process variations with both static and dynamic sources (variations in physical parameters during the manufacturing and in electrical parameters during the operation). Dynamic variations depending on the applied tests, such as the fluctuations in supply voltage, could have a large impact on the gate delays. To accurately simulate these effects, the simulation should incorporate important sources of variations and estimate their impact on circuit timing based on a relatively accurate electrical model.

As technology scales into the VDSM regime, power supply noise, which explicitly refers to noise appearing on the supply and ground nets of the chip and in turn affecting the voltage supplied to gates, has become one of the major variation sources [48–50]. With the continuously scaled sizes of transistors, more transistors are allowed to be packed into an area while the improved transistor performance has resulted in a large increase in frequency. As a result, the increased density of switching devices and frequency can lead to the power density problem [51]. That is, the amount of current to be delivered increases significantly with a simultaneous reduction in power supply voltages, which could be localized according to instantaneous switching of gates, and results in excessive gate delays.

These voltage fluctuations on the power and ground distribution networks, referred by power supply noise, originate from the currents flowing through the parasitic elements of both the on-chip and package supply networks. The power supply noise can be further divided into two groups depending on whether they are caused by the rate of change of the instantaneous currents flowing through parasitic inductive elements of the network ($L \cdot di/dt$) or by the instantaneous currents flowing through the resistive elements of the network (IR) [52–54]. This thesis work focuses on the IR drop, which is the mainly

produced noise on the on-chip **P**ower **D**istribution **N**etwork (**PDN**) since it is predominantly resistive [55]. Targeting on an accurate yet efficient circuit simulation under the impact of the IR drop effect, an electrical model that estimates the gate delay depending on important electrical parameters has to be established. Moreover, a timing-aware event-driven logic simulation is proposed, which benefits from the time complexity of the simulation algorithm while keeping the accuracy by employing the electrical model. This application can be used to estimate IR drop effect given the electrical models, and further adapted to electrical models that incorporate more electrical parameters to raise the accuracy. By injecting faults, it is also possible to perform variation-aware fault simulations, explicitly considering the impact of IR drop. The developed simulator has been applied to large circuits (up to more than 70K gates) with reasonable performance. Chapter 3 introduces the electrical model and details the simulation algorithm as well as the experimental results.

Chapter 2

KLPG: K Longtest Path Generation

2.1 Motivation

Small manufacturing defects increasingly affect the timing behavior of the circuit as technology scales and in turn degrade the circuit performance by inducing extra delays [17]. PDF [2, 17] is a conventional fault model that can be used to test such manufacturing defects. The circuit is considered faulty by testing a PDF if the propagation of an input transition through the target path exceeds the maximum allowed timing period, which usually refers to the clock cycle. In context of SDD testing, any path through the defect site can be modeled as a PDF. A two-pattern test applied in two consecutive time frames for the PDF launches an input transition and propagates the transition along the path. The delay assignment, which refers to the assignment of a delay (a number or a range) to each connection and transition (rising or falling) [2], is often required to estimate the circuit timing. The total cumulated delay along the path can be compared to the specification time, at which the output will be observed and should stabilize at its correct value in the fault-free case, to determine if the circuit behaves faulty in presence of the target fault. If such a test exists, the path is said to be *sensitizable*, and *sensitized* by the test, otherwise the path is called a *false* path.

The main problem with the PDF model is the large amount of the paths in a real circuit; the exhaustive testing that tries to cover all paths is very expensive and almost infeasible

for many circuits. Instead, most approaches try to find the test that sensitizes a given path, and verify if any faulty behavior presents itself under this test for all possible delay sizes caused by manufacturing defects. The quality of the test can be evaluated by how close the minimum actually detected delay fault sizes are to the minimum possible detectable fault sizes [56]. Obviously, for tests sensitizing shorter paths, more delay sizes might not be detected since it takes less propagation time through these paths than longer ones. For instance, Figure 2.1 presents a circuit containing gates with different delays according to a delay assignment. Commonly the gate delay is given by a set of pin-to-pin (from an input to the output) propagation times depending on the input transition. In this example, all pin-to-pin delays for the same gate are assumed to be identical. The inverter G_0 has the delay 2, the AND gate G_1 and the NAND gate G_2 have the delay 3 while the NAND gate G_f where the defect lies has a larger delay 4. Assuming that the clock period is 8, the delay fault can only be detected by a test sensitizing the longest path $c \rightarrow c' \rightarrow e \rightarrow f$ since the propagation time cumulated along the path is 9, which exceeds the clock period. For tests sensitizing other paths, only delay sizes larger than 4 can be detected.

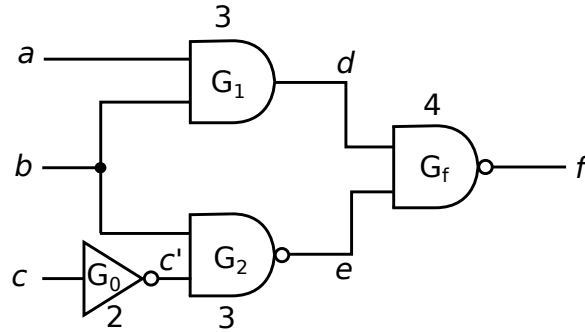


FIGURE 2.1: An example of the longest path

Thus, tests that sensitize the longest path are commonly preferred by small delay testing to achieve better fault coverages. Furthermore, instead of the global longest path, sensitization of the longest path through a given defect site is required by testing local defects, such as resistive opens or shorts.

Under process variations, different instances of a circuit design can exhibit various timing behaviors. Thus, the longest paths selected based on the nominal delay assignment, which can be obtained by timing analysis for the circuit design, might not be the actual longest path. Multiple longest paths through the fault site need to be identified and sensitized to raise the probability of detection. For this purpose, the KLPG algorithm is

proposed [38] that targets on searching K longest sensitized paths through each possible fault sites efficiently. The approach employs graph traversal search algorithm and a data structure called *path store* to store all sub-paths to be considered during the search process. Under the consideration of efficiency, an arbitrary size is exposed on the path store to limit the searching space that could be extraordinarily large in practice. The approach also applies a number of speed-up techniques that help to identify false sub-paths in advance and remove them from the path store immediately. Overflows from the path store will be discarded without consideration, which may lead to a sub-optimal solution with shorter or less paths found instead. To investigate the impact of the store size on the solution optimality, this thesis work proposes a new algorithm called Opt-KLPG that guarantees the optimality of the KLPG solution.

The remainder of the chapter is organized as follows: Section 2.2 gives preliminary knowledge related to the algorithm; the KLPG flow proposed by Walker et al. [38] is introduced in Section 2.3 briefly; Section 2.4 details the Opt-KLPG Algorithm [57]; Section 2.5 presents the application of Opt-KLPG in a statistical delay testing flow under process variations and Section 2.6 reports the experimental results.

2.2 Preliminaries

2.2.1 Classification of Path Delay Faults

A PDF can be classified according to its testability under various conditions. One of the most popular classifications of PDF is Cheng’s classification [2], given as follows: robustly testable, non-robustly testable, functionally sensitizable and functionally redundant PDFs. Figure 2.2 presents the relation between classified PDF sets. Subset relations can be observed that the functional sensitizable set contains the non-robustly testable set, which further contains the robustly testable set.

Every superset of PDF has a looser test condition—the condition required by a test to detect the PDF—than its subset. The following definitions are used to describe the test condition. A path in a circuit is a sequence of signals denoted by $s_0 s_1 \dots s_n$, where s_0 is a primary input, $s_1 \dots s_{n-1}$ are gate outputs and s_n is a primary output. An *on-input* is a connection between two gates along the path. An *off-path input* (or a *side-input*) is

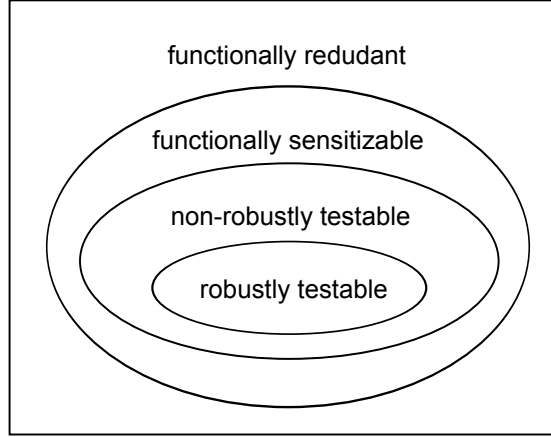


FIGURE 2.2: Testability classification of PDFs [2]

any input of a gate on the path other than its on-input. The test for a PDF consists of two patterns (P_1, P_2) applied to the circuit consecutively.

A PDF is *functionally sensitizable* if there exists an input vector P_2 such that all the side-inputs settle to non-controlling values¹ on P_2 when the on-inputs have the non-controlling values. Otherwise, it is called *functionally unsensitizable*. A PDF is *functionally redundant* if the corresponding path is a false path—the path cannot be sensitized by any test—under all delay assignments. A functional unsensitizable path is also functionally redundant. However, this is a sufficient condition but not a necessary one. The reason is that functional sensitization is only defined with respect to the second pattern P_2 while the test for a PDF requires two patterns; even if P_2 satisfying the functional sensitizable condition can be found, it does not mean the initialization vector P_1 can always be found such that the two-pattern test sensitizes the path.

A PDF is *non-robustly testable* if and only if there exists a two-pattern test (P_1, P_2) satisfying the following conditions:

- it launches the desired logic transition at the primary input of the target path;
- all side-inputs of the path settle to non-controlling values under P_2 .

A PDF is *robustly testable* if and only if there exists a two-pattern test (P_1, P_2) that satisfies the condition for the non-robust test, and, whenever the transition at an on-input is from the non-controlling value to the controlling value, a steady non-controlling value

¹An input of a gate is said to have a *controlling* value if it uniquely determines the gate output independent of other gate inputs. Otherwise, the value is said to be *non-controlling*.

is required at every side-input of the same gate. A robust test can detect the target PDF independent of the delays in the rest of the circuits, i.e., it guarantees the detection of the target fault in presence of other PDFs, whereas the detection is not guaranteed by a non-robust test in the same circumstance. The reason is that a non-robust test allows the output to change before the on-path transition is propagated, which can invalidate the test result. An example of such an invalidation is presented in Figure 2.3.

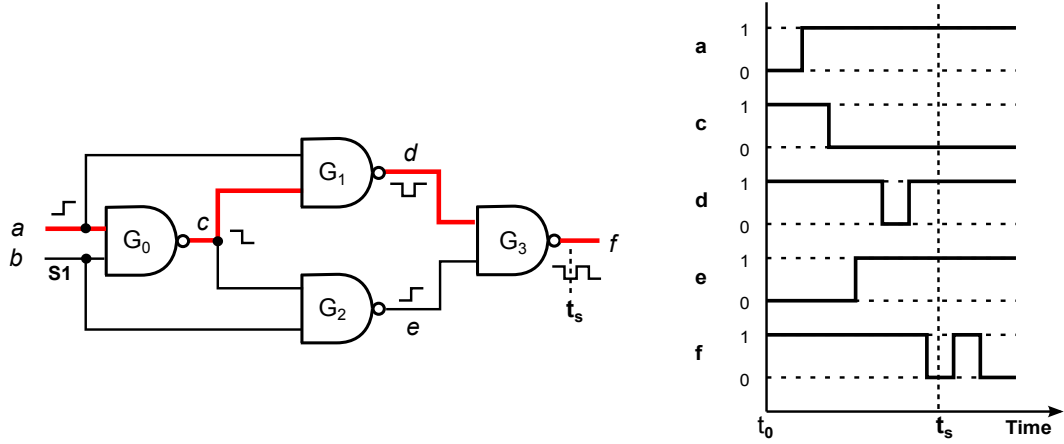


FIGURE 2.3: An example of the invalidation by a non-robust test

A two-pattern test $\{(0, 1), (1, 1)\}$ is applied to the circuit in the left under a given delay assignment. The path corresponding to the target PDF $a \uparrow c \downarrow d \uparrow f \downarrow$ (\uparrow and \downarrow denote the rising and falling transitions, respectively) is marked with bold lines. Signals in the circuit are initialized by the first vector $(0, 1)$, and stabilize before the time point t_0 . The signal transitions after t_0 are represented in the timing diagram right. After applying the second vector $(1, 1)$, a rising transition is launched on the primary input a while b remains a static value of 1, denoted by $S1$. The output of G_0 c falls from 1 to 0 after the gate delay of G_0 . A glitch $1 \rightarrow 0 \rightarrow 1$ at the output of G_1 d is caused by the input transitions at a and c with different arrival times. Assuming that the rising transition at the output of G_2 e is earlier than the first transition of d under the given gate delays, consequently the output f has a switching value of $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$. If the observation point is t_s , the target PDF will not be detected by the test since the correct value 0 is observed at the output. The test is invalidated though the PDF is present, i.e., the transition propagated along the corresponding path is delayed (the $1 \rightarrow 0$ transition after t_s at f). For the same circuit, an example of the robust test for PDF $a \uparrow d \downarrow f \uparrow$ is $\{(0, 0), (1, 0)\}$. It satisfies the non-robust condition, and the additional condition for

the robust test in that the side-input e steadily remains 1 when the on-input d switches from the non-controlling value 1 to the controlling value 0.

2.2.2 SAT-solver and SAT-based ATPG

SAT is the problem of determining whether there exists an interpretation that satisfies a given Boolean formula. In other words, the problem is defined by whether every variable in the given Boolean formula can be assigned with a logic value TRUE or FALSE so that the formula evaluates to TRUE. If such variable assignments exist, the formula is called *satisfiable*, otherwise the formula is *unsatisfiable*.

Solvers for SAT are powerful formal proof engines that determine an assignment to variables of a Boolean function $\Phi : \{0, 1\}^n \rightarrow \{0, 1\}$ such that Φ evaluates to 1 or prove that no such assignment exists [58]. As given by the function notation, the n variables of the function are assigned with either 1 or 0, which corresponds to the logic value TRUE or FALSE; the function evaluates to 1 or 0 indicating that the formula is satisfied by the assignment or not. The function is given in **Conjunctive Normal Form (CNF)**, which is a conjunction of clauses; a clause is a disjunction of literals and a literal is a Boolean variable in its positive form (x) or negative form (\bar{x}). SAT and extensions thereof are well investigated problems, for which efficient solving algorithms have been proposed in the past (e. g., [59–61]).

ATPG is a technology used to find a sequence of tests that can be applied to a digital circuit to distinguish between correct and faulty circuit behaviors caused by defects. A SAT-based ATPG requires a transformation of the problem description into a Boolean formula in CNF as defined above. Each signal in the circuit can be assigned with a Boolean variable representing its logic value 0 or 1. The functionality of each gate can be transformed into a set of clauses using Boolean variables to represent its inputs and output. Using truth tables or algebraic conversion a CNF can be easily derived for each gate type. Then the complete circuit is represented by a conjunction of all CNFs derived for gates, which is a CNF itself. Given a fault to be tested, additional constraints for fault excitation and propagation of the faulty behavior to some observation point can be formulated and given as a CNF. The conjunction of the CNF for the circuit and the CNF for the fault construct the SAT instance to be satisfied or proved to be not satisfiable by a SAT-solver. Each assignment of variables in the SAT instance corresponds to a test

for the given fault that excites the fault and propagates the fault effect for observation. If the SAT-solver proves that no such assignments exist, the fault is not testable.

The algorithm Opt-KLPG applies the SAT-based ATPG tool TIGUAN from the University of Freiburg [44] to verify the sensitizability of the found longest paths. The tool is originally designed for detection of conditional multiple stuck-at (CMS@) faults in multiple time frames. A CMS@ fault consists of a list of arbitrary assignments of logic values in a two consecutive time frames and a list of stuck-at faults, denoted as follows:

- a list of $s@0 \leftarrow v$ or $s@1 \leftarrow v$ where 0 or 1 behind @ denotes the first or second time frame, respectively; v denotes the logic value assigned to signal s in the corresponding time frame, and
- a list of stuck-at 0 or stuck-at 1 faults.

The algorithm that constructs the CMS@ fault for a path will be given in 2.4.4.2. If a test for the constructed fault can be found by TIGUAN, the corresponding path is proved to be testable, otherwise it is a false path and will be excluded from the solution—the K longest sensitizable paths.

2.3 Walker's KLPG Algorithm

The KLPG algorithm proposed in [38] is to find K longest sensitizable path through a target gate G_f . The delay assignment is given in advance for every gate in terms of a set of propagation time of the rising or falling input transition from each input to the output. The minimum-maximum delay from each gate to a primary output without considering logical constraints, which is called min-max PERT delay, is computed as well. Gates that are neither in input-cone nor in output-cone of G_f are first identified. (The input- and output-cones of a gate refer to the circuit blocks driving and driven by the gate, respectively.) They are not considered by computation of PERT delays since these gates can never be added to a path through G_f . The computation is performed iteratively in the reversed order of gate level. Obviously the PERT delay is zero for primary outputs. For signals driving gates, the PERT delay is given by the sum of gate delay and the PERT delay from the gate output. For signals driving a fan-out, PERT

delays for branches are computed at first, the minimum and maximum among them correspond to the max and min PERT delays of the stem, respectively; the associated branches are noted as well.

A data structure called *path store* is used to store sub-paths considered at the same time. It is initialized by sub-paths, each corresponds to a rising or falling primary input in input-cone of G_f . Assuming that input-cone of G_f contains n primary inputs, $2 \times n$ sub-paths are inserted into the path store. It is sorted by the max esperance of the sub-paths. The max (min) esperance of a sub-path is the sum of the path length and the max (min) PERT-delay of its last node. In other words, it corresponds to the upper (lower) bound of the path delay when it grows to a complete sensitizable path. These two bounds can be used by false path elimination techniques to speed up the searching.

During the search process, the sub-path with the largest max esperance is considered first since it will potentially grow to a complete path with the maximum delay. The sub-path will be extended to the next gate associated with the max PERT delay from its last node. If the sub-path before extension drives a fan-out, a copy of it is inserted into the path store. The min and max esperances for the copy is newly calculated without consideration of the branch already extended; the path store is sorted again. After each extension, constraints to propagate transitions through the added gate are applied. All side-inputs must settle to non-controlling values or remain non-controlling depending on certain sensitization condition required for the test. Direct implication is an operation that imply inputs or output values of a gate from known inputs or output of the same gate. It is used to propagate the constraints throughout the circuit. If any conflict is caused after the operation, the sub-path is dropped immediately and a new iteration starts by considering the next sub-path with the maximum esperance, otherwise the currently considered sub-path will be extended further. Since min esperance is only used by some other false path elimination techniques that will not be introduced in detail, in the remaining text of this thesis, max esperance is shortened to esperance for the simplicity of reading. Note that direct implication and false path elimination techniques may not exclude all sub-paths that are not testable. After a sub-path is extended completely, a justification process (a justification algorithm based on the FAN [41] style decision tree) is performed to get the test pattern and finally decide if the path is sensitizable. The search process terminates when K longest sensitizable paths are found or all paths in the path store are considered.

2.4 Opt-KLPG: K Longest Path Generation for Optimal Solutions Under Memory Constraints

Modern nanoscale circuits may contain tens of thousands or even millions of gates. As given in Section 2.3, if a sub-path drives a fan-out and is extended to one of its branches, a copy of the sub-path with updated esperance is inserted into the path store. Consequently, the search space for paths can grow exponentially during the KLPG flow in worst case. It might cost immense memory for saving the paths, and take a long time to find the longest sensitizable ones. Due to the memory and time constraints, the path store used by the KLPG algorithm is bounded in size. Sub-paths with the smallest esperance are removed from the search space immediately if the number of paths in the path store exceeds the size. In case that some of the K longest sensitizable paths are extensions of the removed sub-paths, they are missed by KLPG and a sub-optimal solution will be found instead.

In this section a new algorithm called Opt-KLPG aiming at the optimality of KLPG solution is proposed. The overflows from the path store, which are not investigated by KLPG, are stored in a buffer. The flow calls KLPG iteratively as long as there exist overflows. In each iteration the path store is emptied and filled with overflows. No solutions that belong to the K longest sensitizable paths will be possibly missed since all sub-paths are investigated by the flow. In contrast to the path store, the size of the buffer is not limited. In extreme cases that each extension of a sub-path in the path store results in an overflow, the number of overflows grows linearly in the number of paths in the path store during the current iteration. Furthermore, in case that a solution consisting of K longest sensitizable paths is already found by the current call of KLPG, sub-paths in the buffer that are not potentially better than the existing one can be dropped instantly. Therefore, an unlimited size of the buffer is acceptable. Experimental result shows that an extraordinary large buffer is not required for even a very small path store and large circuits.

The remainder of this section is organized as follows: an overview of the Opt-KLPG algorithm is given in 2.4.1; 2.4.2 presents the method of esperance computation; the KLPG module applied by the Opt-KLPG flow is detailed in 2.4.3; finally, 2.4.4 introduces the techniques used for the sensitization check.

2.4.1 Algorithm Overview

As illustrated in Figure 2.4, the algorithm consists of three functional modules: the *Init*, *Overflow* and *KLPG* modules.

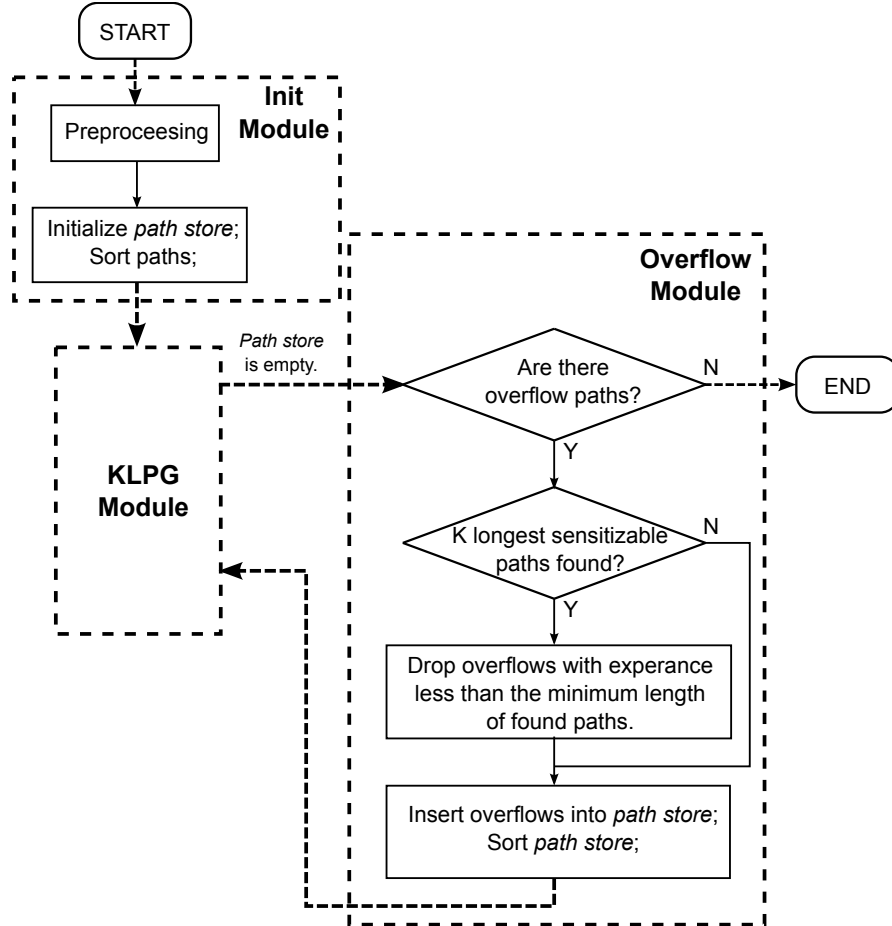


FIGURE 2.4: Opt-KLPG overview

The algorithm starts by running the *Init* module. In preprocessing first a user-defined size is assigned to the path store. The topological information of the circuit including delays for each gate instance is collected. Pin-to-pin delays, i.e., delays from an input pin to the output pin, are given for both rising and falling input transitions. For a given fault, which corresponds to a possible defect site, the PERT delays are computed for signals in input- and output-cones of the fault. A detailed introduction of the computation is given in Section 2.4.2. After that the path store is initialized by sub-paths containing solely primary inputs in input-cone of the target fault. Note that each sub-path is considered twice for both rising and falling transition launched on the primary input due to possible different timing behaviors. That means, for n primary inputs in the input-cone of the fault, $2 \times n$ sub-paths are inserted into the path store. The paths in path store are

sorted in descending order of the path esperance. Figure 2.5 shows an example of the initialization of the path store. The input-cone of a possible defect site G_f is marked in bold lines. In the case shown in Figure 2.5a, the path store is initialized by sub-paths $a \uparrow \rightarrow$, $a \downarrow \rightarrow$, $b \uparrow \rightarrow$, $b \downarrow \rightarrow$, $c \uparrow \rightarrow$ and $c \downarrow \rightarrow$, where \uparrow and \downarrow denote the rising and falling transitions launched at signal lines. In the case shown in Figure 2.5b, the gate driving e is the target fault. Then only sub-paths $b \uparrow \rightarrow$, $b \downarrow \rightarrow$, $c \uparrow \rightarrow$ and $c \downarrow \rightarrow$ need to be investigated and thus inserted into the path store; a sub-path from a is not considered since no extension from it to G_f is possible.

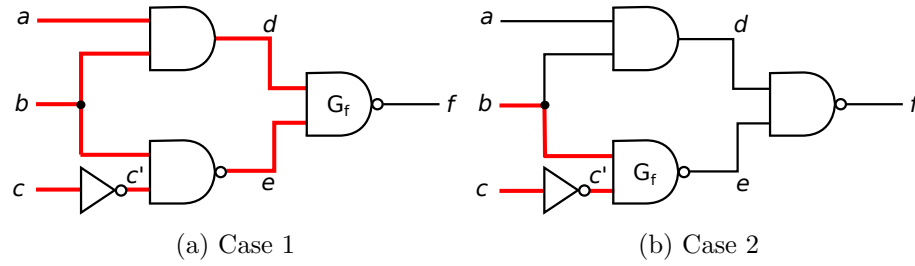


FIGURE 2.5: Path store initialization

After the initialization of the path store, KLPG module is called for the first iteration. Unlike Walker's algorithm, in which overflows from the path store are dropped instantly, the module stores the overflows in buffer for the next iteration. Section 2.4.3 details the implementation of the module. The first iteration of KLPG ends if the path store becomes empty, which means that sub-paths in the path store have been either considered for extension or dropped without consideration if their esperances are less than the minimum length of the currently found solution. The later case is under the assumption that K longest sensitizable paths, which may not construct the final optimal solution, are already found during the iteration. Due to the fact that the esperance of a sub-path corresponds to the upper bound of its possible length after a complete extension, these sub-paths dropped in the latter case will not affect the optimality of the final solution.

Overflows are dealt with by the *Overflow* module when the path store becomes empty. For the same reason given above, overflows with esperances less than the minimum length of an existing solution are dropped without consideration. The path store is filled with the rest overflows and sorted by their esperances in descending order. A new iteration of KLPG begins after that. During the iteration the solution will be updated if any longer sensitizable paths are found. New overflows are inserted into the buffer when the path store becomes full again. The *Overflow* module calls KLPG iteratively

as long as there exist overflows. Finally it gives an optimal solution if it exists, i.e. the K longest sensitizable paths indeed.

2.4.2 Esperance Computation

Like in Walker's flow [38], the term “esperance” is used to describe the maximum possible length of a sub-path after its full growth. As the max PERT delay of a signal indicates the maximum possible length of an extension from the signal, the esperance of a sub-path can be given by the sum of the path length and the max PERT delay from the last node on the path. For the simplicity of reading, the max PERT delay is shortened as PERT delay in the remaining text of this thesis.

The PERT delay of a signal s depends on the launched transition on s , and is always related to a successor, which is the next signal on the corresponding extension from s (with the length of the PERT delay).

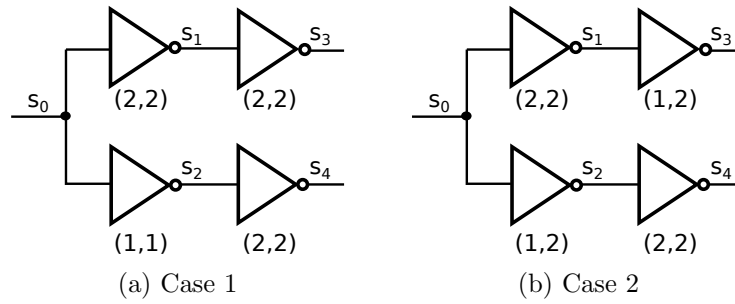


FIGURE 2.6: PERT delays under different delay assignments

Figure 2.6 presents an example of computing PERT delays under two delay assignments. For each gate the propagation delays from an input to the gate output for both rising and falling transitions are given in the form of (d_1, d_2) . In Figure 2.6a, d_1 and d_2 for all gates are assumed to be same. Thus PERT delay of s_0 is computed regardless of the launched transitions. Possible extensions from s_0 to primary outputs s_3 and s_4 are $s_0 \rightarrow s_1 \rightarrow s_3$ and $s_0 \rightarrow s_2 \rightarrow s_4$ that have the lengths of 4 and 3, respectively. The PERT delay of s_0 corresponds to the length of the longest extension 4, associated with the PERT successor s_1 . In Figure 2.6b different gate delays are assigned for rising and falling input transitions. Depending on the transition launched on s_0 , the PERT delay of s_0 could be different for both rising and falling transitions. For a rising transition launched on s_0 , the length of $s_0 \uparrow \rightarrow s_1 \downarrow \rightarrow s_3 \uparrow$ is given by adding up the gate delay of the first inverter on the extension when s_0 rises and the gate delay of the second inverter

when s_1 falls, which results in 4. Analogously the length of $s_0 \uparrow \rightarrow s_2 \downarrow \rightarrow s_4 \uparrow$ is given by 3. Thus the PERT delay of s_0 for a rising transition is 4 with the successor s_1 . For a falling transition on s_0 , the lengths of $s_0 \downarrow \rightarrow s_1 \uparrow \rightarrow s_3 \downarrow$ and $s_0 \downarrow \rightarrow s_2 \uparrow \rightarrow s_4 \downarrow$ are 3 and 4, respectively; the corresponding PERT delay is 4 with the successor s_2 .

During preprocessing the PERT delays are computed iteratively for signals in input- and output-cones of the target defect site in reverse topological order, which means, primary outputs in the output-cone are first considered. Obviously no extension is possible from a primary output, thus a zero PERT delay and a null object for the successor are given for every primary output (in the output-cone). For a signal driving downstream gates, assuming that PERT delays for all downstream gates are computed in advance, then the PERT delay of the signal corresponds to the largest sum of the delay of a downstream gate and the PERT delay from the same gate; the gate output is referred to as the PERT successor. A simple example is presented in Figure 2.7. Assuming that no difference of gate delays is given for rising and falling input transitions, signal s_0 drives two gates G_1 and G_2 with gate delays 2 and 3, respectively. Outputs of G_1 and G_2 are not primary outputs and have PERT delays 5 and 3, respectively. Adding up the PERT delay from the output of G_1 and its gate delay gives 7 while adding up the PERT delay from the output of G_2 and its gate delay gives 6. The PERT delay of s_0 corresponds to the larger sum 7 and the successor is s_1 .

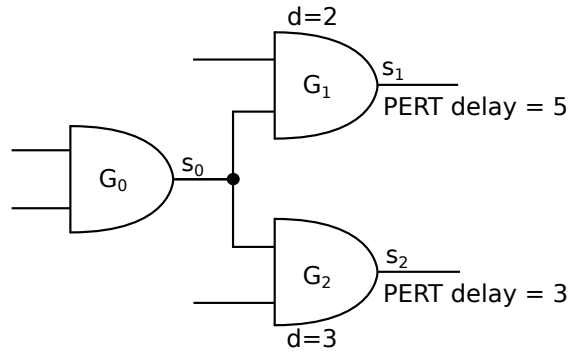


FIGURE 2.7: Computation of PERT delays for a signal driving a fan-out

In the example above, the gate delays for rising and falling input transitions are assumed to be same. In general gate delays could be different for the rising and falling input transitions. Thus PERT delays need to be computed for both transition edges launched on the same signal individually. For signals driving downstream gates, the inversion of the downstream gate needs to be considered by selecting the corresponding PERT delay from the downstream gate. The gate inversion is defined to 1 for gates whose output

is the inversion of the input when other inputs remain the non-controlling value, such as gates INV, NAND and NOR. For other gates, the inversion is defined to 0. Next, the algorithm that computers the PERT delay of the signal s with a rising or falling transition will be given.

For the sake of simplicity, only the pseudo code that computers for s with a rising transition is shown in Algorithm 1. The corresponding PERT delay and successor are denoted by $delay_s$ and $successor_s$, respectively. The initial value of $delay_s$ is zero while $successor_s$ is assigned with a null object. If s is a primary output, these initial values are returned, otherwise all downstream gates driven by s are considered iteratively. For each downstream gate G with output s' , the gate delay of G for the rising driving input s , denoted by d , is assumed to be known. Moreover, the PERT delay of s' with a falling or rising transition depending on whether the inversion of G is 1 is assumed to be already computed. A variable tmp is used to save the PERT delay of s' corresponding to the gate inversion. If the sum of tmp and d is larger than the present $delay_s$, $delay_s$ is updated by the sum and $successor_s$ newly refers to s' . Finally, $delay_s$ settles to the largest sum and $successor_s$ corresponds to the related successor. The PERT delay of s with a falling transition can be computed analogously.

Algorithm 1: Computation of the PERT delay of s with a rising transition

Input: signal s

Output: PERT delay of s with a rising transition

begin

```

     $delay_s = 0;$ 
     $successor_s = null;$ 
    if  $s$  is a primary output then
         $\perp$  return;
    else
        foreach gate  $G$  driven by  $s$  do
             $s' \leftarrow$  output signal of  $G$ ;
             $d \leftarrow$  gate delay of  $G$  for the rising transition on  $s$ ;
            if inversion of  $G$  is 1 then
                 $\perp$   $tmp \leftarrow$  PERT delay of  $s'$  with a falling transition;
            else
                 $\perp$   $tmp \leftarrow$  PERT delay of  $s'$  with a rising transition;
            if  $tmp + d > delay_s$  then
                 $\perp$   $delay_s = tmp + d;$ 
                 $\perp$   $successor_s = s';$ 

```

As given in Section 2.4.1, only the longest paths through the fault site, given by a gate output or a primary input, are targeted by the algorithm. It is unnecessary to compute PERT delays for signals outside the input- and output-cones of the fault site since they cannot be on any target path. Moreover, they need not to be considered when computing the PERT delays of signals driving them. For example, signal s is in the input-cone of the fault site, and s drives the gate g , which is neither in the input-cone nor in the output-cone. Then, by computation of the PERT delay of s , g should not be considered as any extension from s to g will not pass the fault site and grow into the target path. A modified version of Algorithm 1 is thus proposed that takes a list S as an extra input, as shown in Algorithm 2. The list contains all signals whose PERT delays should not be computed or considered by computation of the PERT delay of its upstream signal. That means, if the target s is in the list, the initial value of the PERT delay is returned. Otherwise, if the output of a downstream gate s' is in S , the downstream gate is ignored and the iteration continues for the next downstream gate.

In preprocessing of Opt-KLPG, signals are sorted in reverse topological order and their PERT delays are computed in the sorted order by applying Algorithm 2 iteratively. The iterative process starts for the primary outputs, which have the highest topological level and are assigned with the initial value of PERT delay. For a signal driving downstream gates, as the PERT delays for all downstream signals are computed beforehand during the iterative process, its PERT delay will always be returned by Algorithm 2.

Figure 2.8 presents an example of the iterative process. The gate delays are given in the form of (d_r, d_f) where d_r and d_f are delays for the rising and falling input transitions, respectively. Without loss of generality of the algorithm, different inputs of a gate are assumed to have the same delays. The signals are sorted by their topological level decreasingly: inputs a, b, c of level 1, d of level 2, e and f of level 3 and output g of level 4. The computation is performed for signals in queue: g, e, f, d, a, b and c . No particular order is given for signals of the same level.

First the primary output g is considered, its PERT delay is zero and its PERT successor refers to a null object by default. The next signal in queue is e . The PERT delay of e with a rising or falling transition equals to the gate delay of G_f for the same input transition. Signal g is the successor for both transitions. The same PERT delay and successor are assigned to the next signal in queue f . Signal d drives a NAND gate whose inversion is

Algorithm 2: Computation of the PERT delay of s with a rising transition**Input:** signal s , list S **Output:** PERT delay of s with a rising transition**begin** $delay_s = 0;$ $successor_s = null;$ **if** $s \in S$ **then**

return;

if s is a primary output **then**

return;

else **foreach** gate G driven by s **do** $s' \leftarrow$ output signal of G ; $d \leftarrow$ gate delay of G for a rising transition on s ; **if** $s' \in S$ **then**

continue;

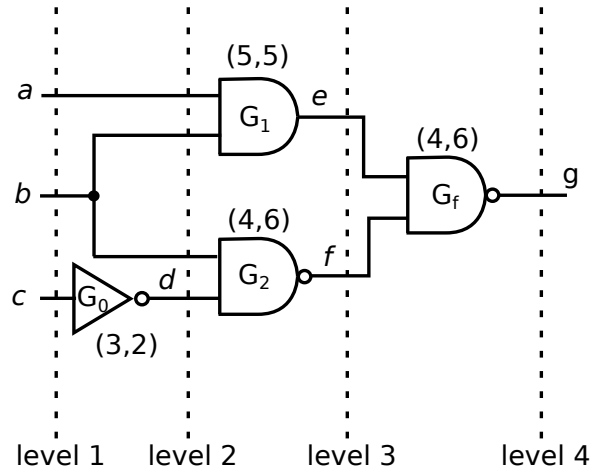
if inversion of G is 1 **then** $tmp \leftarrow$ PERT delay of s' with a falling transition; **else** $tmp \leftarrow$ PERT delay of s' with a rising transition; **if** $tmp + d > delay_s$ **then** $delay_s = tmp + d$; $successor_s = s'$;

FIGURE 2.8: Iterative computation of PERT delays

1, then the PERT delay of d with a rising transition is the sum of the rising gate delay 4 and the PERT delay of f with a falling transition 6, which is 10. Analogously the PERT delay for the falling transition of d is computed and given by 10. The PERT successor for both transitions is f . Signal a drives an AND gate, then the PERT delay of a with a rising transition is computed by adding up the delay of G_1 for the rising input 5 and

the PERT delay of e with a rising transition 4, which gives 9; e is the PERT successor. Analogously the PERT delay for the falling transition of a is computed and given by 11 with the successor e . The next signal in queue is b , which drives two downstream gates: AND G_1 and NAND G_2 . For the rising transition of b , adding up the gate delay of G_1 5 and the PERT delay of e with a rising transition 4 gives 9 while adding up the gate delay of G_2 4 and the PERT delay of f with a falling transition 6 gives 10. The PERT delay equals to the larger one, which is 10 associated with the successor f . Analogously for the falling transition of b , the PERT delay is computed and given by 11 with the successor e . The last signal in queue c drives an inverter G_0 whose inversion is 1, thus the PERT delay for the rising transition of c is computed by adding up the gate delay for the rising input 3 and the PERT delay of d with a falling transition 10, which gives 13 and the corresponding successor d . Analogously 12 and d are given for the PERT delay and successor of c with a falling transition.

2.4.3 KLPG Module

In this section, the *KLPG* module is introduced in detail. In Figure 2.9 the *Init* and *Overflow* modules are illustrated as boxes with dashed lines while the *KLPG* module is detailed by a flowchart that contains a sub-module dealing with new paths.

An iterative process is performed by the *KLPG* module as long as the path store is not empty. In each iteration the sub-path with the largest esperance is selected from the path store and considered first since it will potentially grow to the longest path. It is extended to the gate associated with the PERT delay of the last signal on the path. If the signal drives a fan-out, the *New Path* module is called to generate copy of the path with updated esperance. After the extension of the currently considered sub-path, direct implications are applied to check if any conflicts are caused by the extension. If the sub-path passes the check and is not complete yet, it is further extended and checked iteratively till it becomes complete. Anytime the check using direct implication fails, i.e., a conflict is caused by the extension, the sub-path is identified to be false and dropped instantly; the *KLPG* module starts a new iteration for the next sub-path with the largest esperance.

Note that the sensitizability of the path is not guaranteed by successful extensions checked only using direct implication. After the full growth of the currently considered

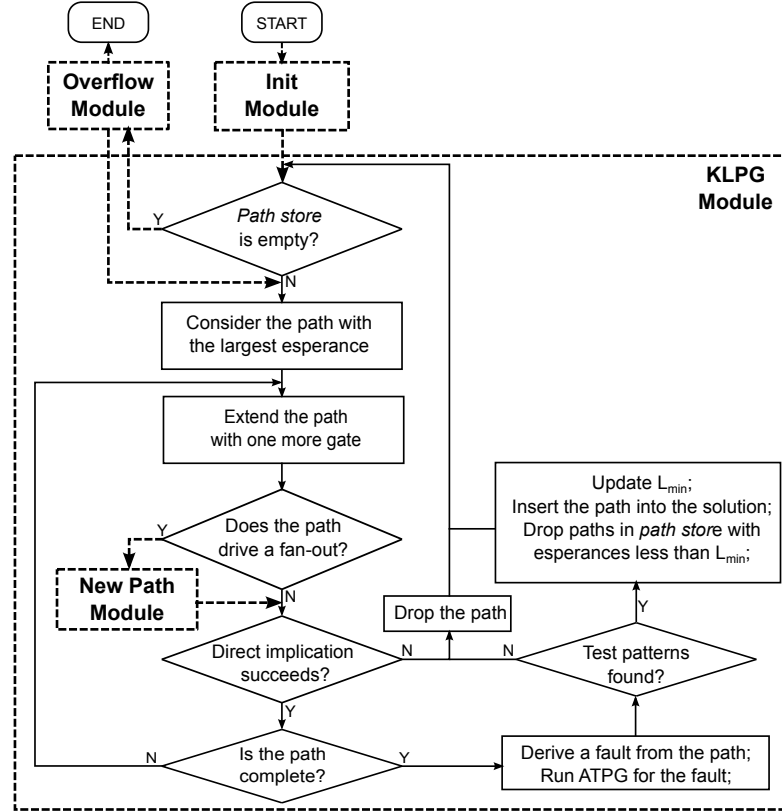


FIGURE 2.9: KLPG module

sub-path, ATPG is run to find test patterns sensitizing the path and finally verify the sensitizability. A SAT-based ATPG tool TIGUAN [44] is used for the test generation. If ATPG fails, the considered path is dropped as it is not sensitizable, otherwise it is marked as one of the K longest sensitizable paths. A variable L_{min} is used to denote the minimum path length in the existing solution. By default it is set to zero if not *all* K longest sensitizable paths are found, otherwise the value is updated by the minimum length of the K found paths. Whenever L_{min} is updated, all sub-paths with esperances less than L_{min} are removed from the path store and dropped instantly.

Figure 2.10 presents the function implemented in the *New Path* submodule. After a sub-path is extended to one of the branches in the fan-out, a copy of the original path with updated esperance needs to be generated. The new path can be extended to any other branches but the one already extended. That means, the PERT delay of the signal at the stem of the fan-out is newly computed considering the possibility of extension to branches that have not been tried yet. For the example shown in Figure 2.8, b drives a fan-out. For a falling transition of b , the sub-path $b \downarrow \rightarrow$ is extended to e , which is the PERT successor of b . A copy of $b \downarrow \rightarrow$ is generated and inserted into the path store. It

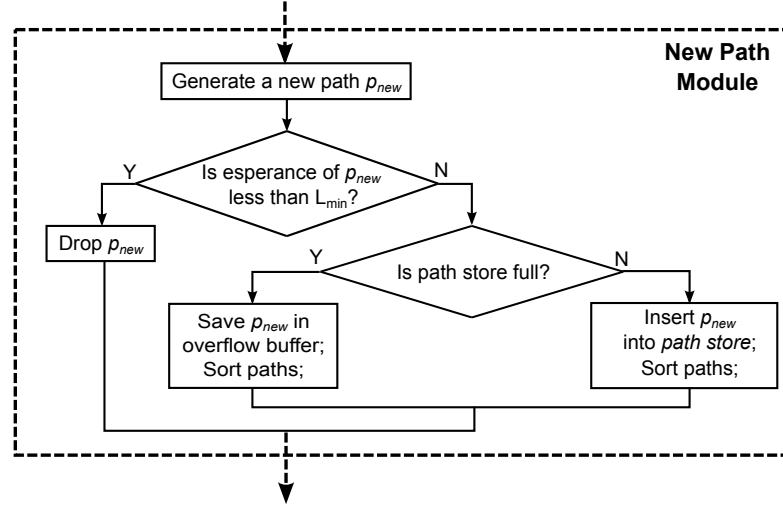


FIGURE 2.10: New Path submodule

can be further extended to other branches of b except the one leading to e . The PERT delay of b is updated by applying Algorithm 2. Since the extension to e exists already, e is included in list S and thus ignored by the computation. The computation gives the new PERT delay 10 and PERT successor f . Consequently, the esperance of the new path is updated by adding up the path length and the new PERT delay of the stem. If the esperance of the new path is less than L_{min} , the minimum length of the existing K -path solution, it can be dropped instantly. Otherwise, the new path is inserted into the path store or the overflow buffer depending on whether the path store is full.

As given in Section 2.4.1, the conventional KLPG algorithm [38] can miss the optimal solution due to the limited size of path store. To investigate the impact of store size on the solution optimality, an application based on the conventional KLPG algorithm was implemented and used in experiments. Since the purpose of the implementation is not to achieve computation speed that exceeds existing KLPG algorithms, a few false path elimination techniques applied by existing approaches were not integrated into the implementation. As shown in the flowchart illustrated in Figure 2.11, unlike Opt-KLPG, the KLPG tool does not take the overflows into account; it terminates when K longest sensitizable paths are found or all paths in the path store are investigated. Actually, it constitutes the first iteration of Opt-KLPG.

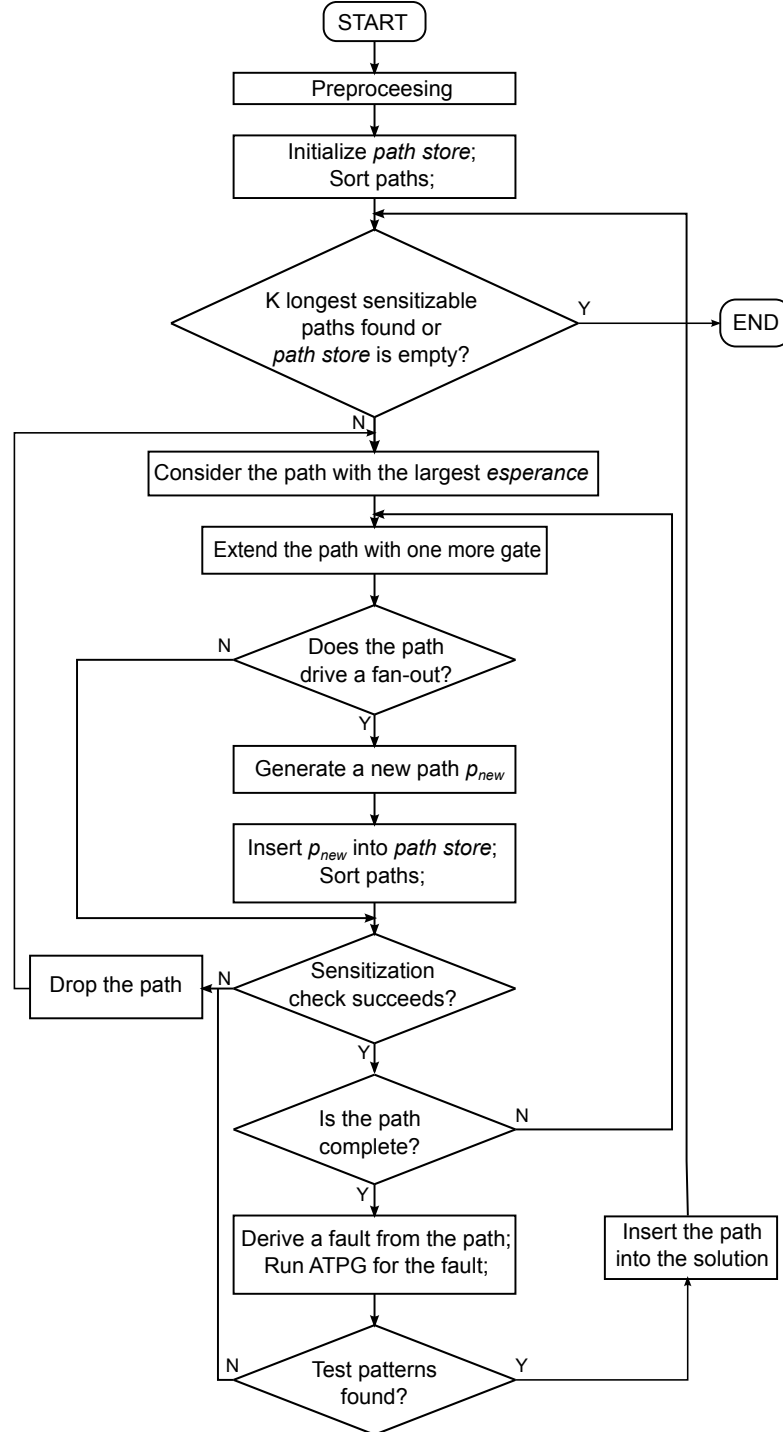


FIGURE 2.11: KLPG flowchart

2.4.4 Sensitization check

A path is said to be sensitizable if and only if there exist a two-pattern test that launches an input transition and allows the propagation of the transition through the path to its

output. During the flow the considered sub-path is extended iteratively. The sensitization of the path implies that each extension can propagate the transition of the on-input to the output of the added gate. If this requirement cannot be satisfied, the sub-path is false, which means that it can never grow to a sensitizable path, and thus can be dropped instantly. Methods that identify a false sub-path can be performed after each extension from the considered path. By eliminating these false sub-paths the search space can be trimmed off dramatically since all possible extensions from these paths, which are false either, are in turn not explored.

However, due to the complexity of sensitization check, it is not easy to implement methods that can guarantee the effectiveness and the efficiency at the same time. One variant is to execute direct implications based on value assignments required by the sensitization. If any conflict after the execution is observed, the considered sub-path is identified to be false. Using this method most false paths can be identified efficiently, as reported in [62]. But not all the false paths are guaranteed to be eliminated in this way, which means that a sub-path can be false while no conflict is caused by the execution of direct implications. Thus in the flow, ATPG is performed to finally verify the sensitization of the completely extended paths. If no tests can be found that sensitize the considered path, it is identified to be false and dropped by the flow. Both methods used for the sensitization check, by executing direct implications and by performing ATPG, are detailed in 2.4.4.1 and `refssec:atpg`, respectively.

2.4.4.1 Direct Implication

Direct implication is an operation that implies values at the inputs or outputs of a circuit element directly from the values of other inputs and outputs. The operation proceeds either forwardly in the fan-out of the element, or backwardly in its fan-in [2]. Values at lines are implied based on the logic behavior of the circuit elements. Examples of forward and backward direct implications for an AND gate are presented in Figure 2.12. To be distinguished from the values already known, the implied values are boxed by lines. As shown in Figure 2.12a, given a zero at one input and a *don't care* value, denoted by X , at the other input and the output (the figure above), a zero can be implied forwardly at the output according to the logic behavior of the AND gate (the figure below). Figure

2.12b presents a backward direct implication. The assignment of 1 to the output implies that both inputs of the AND must be 1.

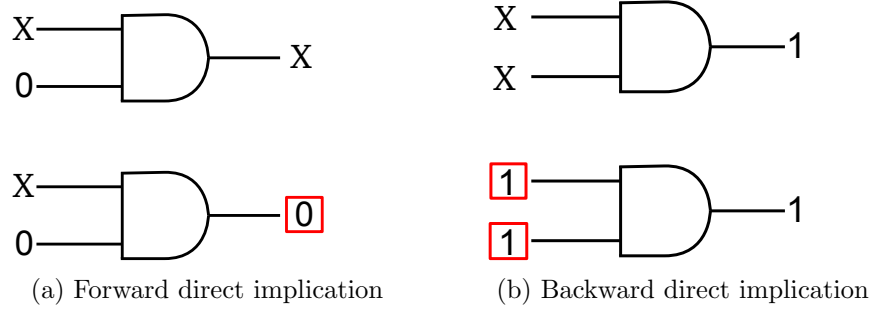


FIGURE 2.12: Examples of direct implication

The sensitizability of an extension from a sub-path requires that the transition on the path can be propagated through the extension. According to various sensitization criteria, different conditions are given to meet the requirement. For example, non-controlling values should be assigned to side-inputs of the extension by the second pattern in the two-pattern test according to the non-robust criterion. New assignments can be further derived from existing ones using direct implication. If any conflict is observed during the process, i.e., two opposite values are assigned to the same signal, the extended sub-path is false and can be dropped instantly since the conditions of propagation are violated. Figure 2.13 presents an example of the sensitization check using direct implication. Assume that a falling transition, marked by a down arrow \downarrow , is imposed on primary input a and is to be propagated through gate G_1 . According to non-robust sensitization criterion, the side input of the added gate G_1 b should settle to non-controlling value of AND, which is 1. Since b drives an inverter G_2 , a zero is assigned to its output d (given in the red box) by direct implication. No conflict is observed by now, the sub-path is further extended to e , which requires the assignment of the non-controlling value of NAND 1 to the side-input of G_3 d . A conflict is observed: the new value of d differs from the one assigned already. It indicates the failure of the sensitization check for the extension to e ; sub-path $a \downarrow \rightarrow c \downarrow \rightarrow e \uparrow \rightarrow$ can be dropped instantly.

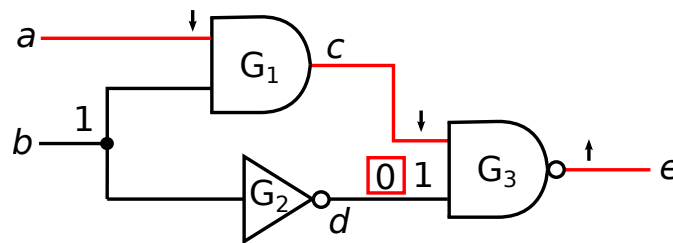


FIGURE 2.13: An example of sensitization check using direct implication

The method of executing direct implications for sensitization check is described in Algorithm 3. Assuming the non-robust sensitization criterion, for the currently considered sub-path, extension to one of its downstream gates requires that each side-input of the gate, denoted by s , must settle to the non-controlling value v_{new} . In preprocessing, all possible direct implication operations for each circuit element are saved in lookup tables, as functions that imply input or output values from known value assignments (using three-valued logic).

Algorithm 3: Sensitization check using direct implication *CheckForAssignment*

Input: signal s , logic value v_{new}
Output: TRUE or FALSE

begin

```

     $v_{old} \leftarrow$  the current value of  $s$ ;
    if  $v_{old} \neq \text{don't care AND } v_{old} \neq v_{new}$  then
        | return FALSE;
    if  $v_{old} = v_{new}$  then
        | return TRUE;
    Assign  $v_{new}$  to  $s$ ;
    if  $s$  is not the primary input then
        |  $G_{up} \leftarrow$  upstream gate of  $s$ ;
        | if  $\text{DirectImplicationOnGate}(G_{up}) == \text{FALSE}$  then
        | | /* conflict */
        | | return FALSE;
    if  $s$  is not the primary output then
        | foreach downstream gate  $G_{down}$  driven by  $s$  do
        | | if  $\text{DirectImplicationOnGate}(G_{down}) == \text{FALSE}$  then
        | | | /* conflict */
        | | | return FALSE;
    return TRUE;

```

Algorithm 4: Perform direct implications on a gate *DirectImplicationOnGate*

Input: gate G
Output: TRUE or FALSE

begin

```

    Imply values according to the implication function of  $G$ ;
    foreach signal  $s$  with the implied value  $v$  do
        | if  $\text{CheckForAssignment}(s, v) == \text{FALSE}$  then
        | | /* conflict */
        | | return FALSE;
    return TRUE;

```

For each new value assignment $s \leftarrow v_{new}$, sensitization check using direct implication will be executed recursively, by calling *CheckForAssignment*(s, v_{new}) (Algorithm 3). First, the current value of s (*don't care* by default if it is not assigned yet) is checked. If it is already assigned, depending on whether it differs to v_{new} , the Boolean value FALSE or TRUE will be returned. The former case indicates the failure of sensitization check while the latter case implies that the same assignment has been checked using direct implication before. Only for an unassigned s (with a *don't care* value), it is assigned with v_{new} . Then, for the upstream gate driving s (if s is not a primary input) and every downstream gate of s (if s is not a primary output), *DirectImplicationOnGate* (Algorithm 4) is called that performs the direct implication operation corresponding to the predefined function, and *CheckForAssignment*(s', v') for any newly implied value v' on s' . By any conflicts, the recursive process is aborted and returns FALSE. In that case, the considered path extension is identified to be not sensitizable; no further checks are required for other side-inputs. In the other case that the recursive process succeeds without conflicts and finally returns TRUE (when no more values can be directly implied), the side-input assignment is validated while other side-inputs need to be checked using the same method. The sub-path is successfully extended only when all assignments of side-inputs pass the sensitization check using direct implication; newly assigned/implied values due to the extension are considered validated. Note that for the new path copied from a sub-path driving a fan-out, all validated signal assignments for the original sub-path should not be violated either during further extensions of the new one.

2.4.4.2 ATPG using TIGUAN

Search for direct implications helps to identify most false paths in advance before they are extended completely. However, it is possible that some false paths do not have any conflicts after performing direct implications during the growth procedure. Figure 2.14 presents an example. The sensitization of the path marked with bold lines requires logic 0, the non-controlling value of OR, on both side inputs, which are outputs of two AND gates. Assuming that the algorithm does not know the inputs of either AND gate are tied together, i.e., they should have the same value, no direct implications can be performed since neither input must be 0 or 1. Indeed, the inputs of both AND gates have to be 0 to get the logic 0 on their outputs. It causes the conflict since the inputs

of the two AND gates are connected by an inverter. The path is false but it cannot be identified by the method using direct implication.

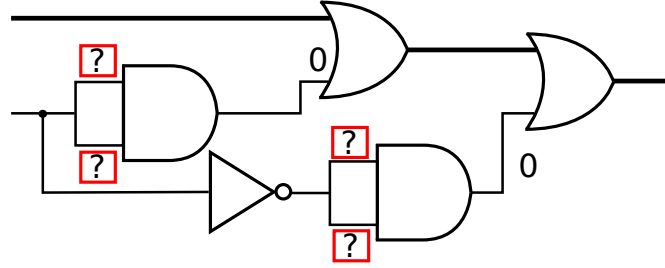


FIGURE 2.14: A false path that cannot be identified using direct implication

To eliminate all false paths and generate tests for sensitizable paths, the ATPG tool TIGUAN [44] is used to sensitize the completely extended paths. CMS@ faults—the dedicated fault model used by the tool—need to be derived for the considered paths, as presented in Algorithm 5. The path P is given by a sorted list of signals s_0, s_1, \dots, s_n , each with the corresponding rising or falling transition. As defined in Section 2.2.2, the CMS@ fault to be derived comprises a list of value assignments A , denoted by $s@n \leftarrow v$ where $n \in \{0, 1\}$, and a list of stuck-at faults F . First, the launched input transition on the path is translated into two corresponding assignments, which are inserted into A . Then, depending on the sensitization criteria [2, p. 386], assignments of non-controlling values on the side-inputs are inserted into A as well. In case of the non-robust criterion, the assignment of non-controlling value to each side input in the second timing frame is inserted into A . The stuck-at faults included by the CMS@ definition need not to be specified, which leaves the list F empty.

Note that the CMS@ fault has a limitation that the value assignments in a time frame corresponds to the finally stabilized value, which means glitches before stabilization or a stable value (no glitches) can not be modeled by the fault. For that reason, the ATPG does not target on robust tests since the robust criterion requires a stable non-controlling value of side-inputs in both time frames whenever an on-input has the transition from a non-controlling value to a controlling value.

An example of deriving a CMS@ fault from a PDF is given in Figure 2.15. The target path $c \uparrow \rightarrow d \downarrow \rightarrow f \uparrow \rightarrow g \downarrow$ is illustrated by bold lines. The up arrow \uparrow and down arrow \downarrow denote the rising and falling transitions on the lines, respectively. According to non-robust test criterion, a two-pattern test (P_1, P_2) sensitizing the target path launches the desired logic transition at the input of the path, and all side-inputs on the path settle

Algorithm 5: Derivation of the CMS@ fault according to non-robust sensitization criterion

Input: Path P : s_0, s_1, \dots, s_n with corresponding transitions

Output: CMS@: list A and list F

begin

if s_0 has a rising transition **then**

 Insert $s_0@0 \leftarrow 0$ and $s_0@1 \leftarrow 1$ into A ;

else

 /* s_0 has a falling transition */

 Insert $s_0@0 \leftarrow 1$ and $s_0@1 \leftarrow 0$ into A ;

foreach signal s on P **do**

 Gate $G \leftarrow$ the gate driven by s ;

$v \leftarrow$ non-controlling value of G ;

foreach input i of G **do**

if $i \neq s$ **then**

 /* i is a side-input */

 Insert $i@1 \leftarrow v$ into A ;

*/

to their non-controlling values under P_2 . The conditions can be interpreted by a list of assignments of logic values defined in a CMS@. Launching the rising transition on primary input c is interpreted by two assignments $c@0 \leftarrow 0$ and $c@1 \leftarrow 1$, which indicate that c is assigned with logic 0 and logic 1 in the first and second time frames when P_1 and P_2 are applied, respectively. Requirements of non-controlling values settled on side inputs in the second time frame are interpreted by assignments $b@1 \leftarrow 1$ and $e@1 \leftarrow 1$, which indicate that the side inputs b and e settle to the non-controlling value of G_2 and G_f in the second time frame, respectively. A CMS@ fault is constructed by these assignments, and used to generate the two-pattern test by the ATPG tool.

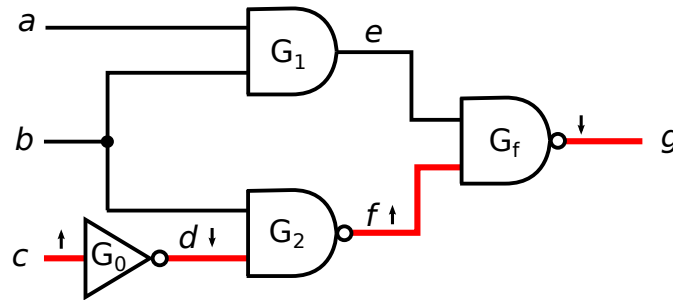


FIGURE 2.15: An example of CMS@ derivation

2.5 Application of Opt-KLPG by Variation-aware Fault Grading

The work described in this section [63] is a collaboration work with the partner from Universities of Freiburg and Stuttgart within the project RealTest [64]². The contribution of this thesis work in the collaboration work is to provide test patterns for testing circuit instances affected by variations using Opt-KLPG. In the remainder of this section, 2.5.1 presents the problem modeling for testing delay faults under process variations and 2.5.2 introduces briefly the variation-aware fault-grading procedure that generates tests maximizing the modeled fault coverage

2.5.1 Modeling for Variation-aware Delay Testing

As introduced in 1.2.1, a statistical test flow targeting variation-aware tests combines the ATPG for fixed process parameter values with the statistical fault simulation. In the context of delay testing, the relevance of parameter variations lies in their effect on the delays of individual circuit elements, e. g., gate delays. Let N be the number of gate delays that can be affected by variation. A *parameter configuration* $p = (p_1, p_2, \dots, p_N) \in P$ is a list of actual values of every modeled gate delay, and P is the space of all possible parameter configurations. A circuit instance corresponding to the parameter configuration p is denominated by C_p .

The infinite set P of parameter configurations is approximated by a finite set I of circuit instances. Every circuit instance $i \in I$ has fixed gate delays that can be chosen randomly according to a gate delay distribution (or statistically based on actual measurements). Instance 0, called the nominal circuit instance, is defined to have no variations (each gate delay is the mean of the gate delay distribution). All the other instances ($i > 0$) are affected by variations with randomly or statistically determined gate delays.

Assuming the equiprobable distribution of circuit instance, *circuit coverage* $CCcov(f, T)$ of a delay fault f by a test set T (Equation 1.2) can be adapted for the circuit instances $i \in I$, and defined by Equation 2.1 where $det_i(f, T)$ equals to 1 if f is detected in i and 0 otherwise.

²Parts of the collaboration work were supported by the German Research Foundation (DFG) (under grants BE 1176/14-2, PO 1220/2-2, GRK 1103, WU 245/5-1 and WU 245/5-2).

$$CCcov(f, T) = \frac{\sum_{i \in I} det_i(f, T)}{|I|} \quad (2.1)$$

Furthermore, considering a fault list F , the *statistical fault coverage* can be defined by aggregation of circuit coverages for all faults in F , as given in Equation 2.2. The statistical fault coverage corresponds to the percentage of circuit instances in which the target faults F are detected by the test set T .

$$SFC(F, T) = \frac{\sum_{f \in F} CCcov(f, T)}{|F|} = \frac{\sum_{f \in F} \sum_{i \in I} det_i(f, T)}{|F| \cdot |T|} \quad (2.2)$$

The statistical test generation aims at maximizing the circuit coverage for the faults and the overall statistical fault coverage SFC , that means, as many faults from F in as many circuit instances from I should be detected by the test set T .

2.5.2 Variation-aware Fault-grading Procedure

To tackle the problem modeled above, a variation-aware fault-grading procedure is proposed that aims at maximizing SFC by performing ATPG combined with Monte-Carlo fault simulation [63]. An initial set of test pairs is generated for the nominal circuit instance. Then delay fault simulation is performed using a new highly parallel algorithm for General-Purpose Graphics Processing Units (GPGPU), explicitly considering all faults from F and all circuit instances from I . The simulation determines the faults affected by variation configurations in terms of fault-instance-pairs that are not covered by the initial test set and thus lower the SFC. These faults will be targeted by timing-aware ATPG, e. g., the KLPG algorithm, to generate tests that are more likely to detect the fault under variations. The newly generated tests will be verified by the fault simulations and improve SFC by detecting by far uncovered fault-instance combinations. The GPGPU-based parallel algorithm can efficiently handle an extremely large number of fault-instance-pairs by assigning different circuit instances to different GPGPU cores. The flow outlined above could result in a very large amount of tests. A test compaction technique, called T_{min} -algorithm, was used to bring an aggressive compaction while preserving the SFC achieved by the original test set.

2.5.3 Path-oriented Delay ATPG

The timing-aware ATPG needs to pin-pointedly generate tests for a delay fault $f = (g, s)$ that excite the desired transition on the fault site g and propagate the transition along the path with the minimum slack less than or equal to the delay size s . The KLPG algorithm can be used to generate such tests that sensitize multiple longest paths through the fault site and thus raise the probability of detecting the fault under variations.

The path-oriented delay ATPG described above can employ two methods as follows:

- *KLPG* approach that works directly on the circuit structure, and extends sub-paths that could be potentially longest and sensitizable while pruning false paths from the searching space at the same time [38, 57]; and
- *SAT-based* approach that transforms the ATPG problem into Boolean formula and generates tests in terms of solutions satisfying the formula [65].

The KLPG flow considers the search space of sub-paths that could be extraordinarily large for modern circuits; the number of considered paths has to be arbitrarily limited due to the efficiency requirement. The optimality of the KLPG solution could be affected by the limitation since overflow paths are out of consideration. Opt-KLPG guarantees the optimality by applying an extra data structure that saves the overflows and iteratively performing KLPG processes until all paths including overflows are considered. The purely SAT-based ATPG benefits from the efficient search optimization techniques incorporated into modern SAT-solvers. However, it represents delays by integer numbers and has to employ sophisticated rounding strategies to compensate for the inaccuracies induced by the representation.

Moreover, a combination of several sensitization conditions (as described in 2.2.1) could be helpful in construction of test sets during the fault grading. Robust tests are particularly powerful because of their capability of detecting faults independently of delays elsewhere in the circuit. However, due to the tightened sensitization constrains, the number of robust tests is usually very limited and thus could result in a low coverage. Other sensitization conditions, such as non-robust and functional sensitization, use looser constrains for the side-inputs. On one hand, these weaker constrains could lead to invalidation of fault detection due to delays in other parts of the circuit. On

the other hand, they often allow sensitization of longer paths than robust tests, which might improve the SFC. Therefore, no sensitization condition is universally optimal; a combination of sensitization conditions can be applied by the fault grading procedure aiming at maximizing SFC. For example, in cases of uncovered faults by robust tests, tests will be generated using weaker conditions in the next iteration.

The contribution of this thesis work in the fault-grading procedure is to generate tests for all considered fault-instance-pairs using Opt-KLPG, according to non-robust and functional sensitization conditions, which are not supported by the SAT-based approach. In the following section, experimental results from the investigation on the KLPG optimality (2.6.1), as well as the fault-grading procedure (2.6.2) will be reported.

2.6 Experimental Results

2.6.1 On the Optimality of KLPG Algorithms

As given in 2.4.1, using the existing KLPG algorithm [38] sub-optimal solutions, less or shorter sensitizable paths instead of the longest ones, may be found due to the limited size of the path store. For the first time, the effects of missing some of the longest sensitizable paths on the defect coverage were investigated. Experimental results from both KLPG and Opt-KLPG flows performed in the same experimental environment were compared.

Both implementations of KLPG and Opt-KLPG were applied to ISCAS'85 benchmark circuits [66], which are known to be small but have a large amount of reconvergencies thus are challenging for path-selections. The experiments were also run for combinational cores of ITC'99 benchmark circuits [67] and industrial circuits provided by NXP with up to 70K gates. The gate-level net-lists and the corresponding real-valued delays were obtained by mapping original structural specifications (Verilog) to the NanGate 45nm Open Cell Library [68] using Synopsys Design Compiler. The real-valued delays for circuit instances were saved in SDF files, which contain pin-to-pin gate delays given by three values for each gate and both rising and falling transitions on every gate input: the minimum, nominal and maximum delays. The nominal delays were used by the experiments. For circuit c6288, a multiplier with a large number of long non-sensitizable

paths, the results were omitted due to the enormous requirement for memory and runtime. In [38], a specific technique called Smart-PERT was developed in order to handle circuit c6288 (a multiplier with a large number of false paths); this technique is not included in this implementation.

For each circuit, KLPG and Opt-KLPG were run for a number of possible defect sites and different path store sizes between 10 and 30,000. Each gate is considered as a possible defect site. For ISCAS'85 circuits all gates were investigated whereas for ITC and NXP circuits randomly selected 1,000 gates were considered since it is very time-consuming to investigate all gates due to the enormous circuit size. A fault list containing all target gates was given for each circuit. K was set to 5 for both flows, i.e., (up to) 5 longest sensitizable paths are to be found for each target gate. Non-robust and functional sensitization criteria were used by the ATPG process in both KLPG and Opt-KLPG flows.

Table 2.1 summarizes the results of both flows for functional sensitization criterion. Column 1 gives the name of each circuit and its gate count in parentheses. The size of the path store π_{max} is set to values between 10 and 30,000 as presented in column 2. Experimental results returned by KLPG and by Opt-KLPG are contained in columns 3 through 6 and columns 7 through 11, respectively. The results of both flows comprise of the total number of overflows encountered during the flow (columns 3 and 7), the total length of the (up to) 5 paths identified by both flows (columns 4 and 9), the total CPU runtime (columns 5 and 10) as well as the time excluding the calls of the SAT-ATPG engine ("Excl SAT", columns 6 and 11); "Excl SAT" describes the actual efforts spent in searching of paths. As given in Section 2.4.1, Opt-KLPG calls the *KLPG* module iteratively. The average number of iterations per SDD, which corresponds to a gate in the fault list, is given in column 8.

From the KLPG results, it can be seen that very small sizes of the path store typically result in large amounts of overflows. As shown in column 3, the number of overflows tends to decrease as π_{max} increases. However, it does not always hold. For example, 143,619 overflows were encountered by running KLPG for c1908 when π_{max} was set to 10 while 215,713 overflows were encountered when π_{max} was set to 50. The reason is that a larger path store may contain a sub-path which in turn results in generation of more sub-paths; a smaller path store may simply not contain this sub-path. In case that

the number of overflows is zero, the result of KLPG is optimal, which means the total length of found paths are the same for both flows (columns 4 and 9). The minimum value of π_{max} by which no overflows were encountered varies among different circuits and does not always grow with circuit sizes. For most ITC and NXP circuits a path store with size limit 30,000 satisfies the requirement of optimality. Moreover, if there exist overflows, KLPG may fail to find the optimal solution but an optimal solution is still possible to be found. For example, for c7552 and $\pi_{max} = 500$ an optimal solution was found though there exist 509 overflows. It means that none of these overflows can grow to one of longest sensitizable paths. Note that the path lengths are real numbers rounded to the nearest integer. For instance, KLPG fails to find the optimal solution for p45k with $\pi_{max} = 30,000$, but the reported length of 4659 corresponds to the optimal length due to rounding artifacts. As for the runtime, it rises with growing π_{max} in general. An exception happened to p35k, for which the runtime by $\pi_{max} = 30,000$ is more than 50% faster than by $\pi_{max} = 3,000$ while for p45k and p78k the situation is rather the reverse. The reason for the exception is that for $\pi_{max} = 3,000$ many sub-paths in the path store could be false whereas for $\pi_{max} = 30,000$ sub-paths which are more likely sensitizable are included and investigated first after sorting newly; the runtime is thus shortened.

Columns 7 through 11 present experimental results from Opt-KLPG. Column 7 contains the number of overflows encountered during the complete run of all the iterations. This number is always higher than the number of overflows encountered by KLPG (Column 3) since KLPG actually constitutes the first iteration of Opt-KLPG. Column 8 gives the average number of re-iterations of KLPG module per fault, that is, iterations required by the flow after the initial run of KLPG. The number is always zero if no overflows are encountered. Note that some reported average numbers are zero due to rounding artifacts though there exist overflows. The sum of lengths of the found paths is reported in column 9. This number is independent from the size limit of the path store since Opt-KLPG always gives an optimal solution. Columns 10 and 11 contain the runtimes. The trend observed from the result of KLPG, that the runtime tends to be longer for a larger π_{max} , is less pronounced here. For the same π_{max} , Opt-KLPG requires more time than KLPG due to the possible re-iterations. However, for the same target on an optimal solution, less time can be spent by Opt-KLPG than KLPG. For instance, c1908 requires $\pi_{max} = 30,000$ in order to avoid overflows altogether; this necessitates a total runtime of 815 CPU-seconds. However, Opt-KLPG achieve the same result for

$\pi_{max} = 500$ using 306 CPU-seconds. Furthermore, even though running Opt-KLPG with $\pi_{max} = 10$ results in a larger number of re-iterations per gate, the total runtime of 346 CPU-seconds is still less than for KLPG with $\pi_{max} = 30,000$. It shows that an appropriate value of π_{max} , by which Opt-KLPG yields the optimal solution most efficiently (with the shortest run time), varies for different circuits. The value can be given according to the circuit size or empirically. For KLPG, it may be the minimal value of π_{max} by which the optimality is maintained. For Opt-KLPG, since the optimality is always guaranteed, the runtime is the only metrics of the selection. For example, $\pi_{max} = 30,000$ for p35k and $\pi_{max} = 1,500$ for p78k may be good choices for Opt-KLPG aiming at the optimal solution with the best performance.

Moreover, in contrast to the FAN algorithm which can be seamlessly integrated into the KLPG framework in [38], the SAT-based ATPG tool TIGUAN [44] is used by the path sensitization that requires a lengthy process: a path is mapped onto a CMS@ fault for which an ATPG-instance is constructed, which in turn is mapped onto a SAT-instance and solved. Even though the employed SAT-ATPG engine can perform these tasks in a fraction of a second, the large number of paths to be evaluated results in a large amount of SAT-ATPG instances and in turn high runtime requirements. In addition to the total runtime (column 5 and 10), the runtime excluding the time spent in solving the path sensitization is reported (column 6 and 11), labeled by “Excl SAT”.

The last three columns evaluate the coverage problems due to sub-optimal path selecting by the KLPG algorithm. Column 12 shows the ratio between the total length of paths found by KLPG (column 4) and by Opt-KLPG (column 9). This ratio is rather large for very small values of π_{max} ; it tends to be within 1% for $\pi_{max} \geq 500$. This is because for many gates the solution are not affected by overflows or the identified paths are not much shorter than the optimal ones. The second-to-last column contains the number of SDD locations (gates) through which KLPG found less paths than are in existence. This is a severe situation with a significant impact on testability under process variations. It rarely occurs for $\pi_{max} > 10$. The final column shows the number of gates for which the total length of paths identified by KLPG was shorter than the optimum. This corresponds to a coverage impact for the SDD associated with this particular gate. In some of the manufactured instances of the circuit, such SDDs will not be adequately tested. A significant number of gates turn out to be vulnerable to this problem even for large values of π_{max} . Figure 2.16 shows, in graph form, the number of such gates

in circuit p35k, accompanied by runtimes for KLPG and Opt-KLPG (which are not affected by this problem).

TABLE 2.1: Experimental results under functional sensitization criterion

Circuit (gate cnt)	π_{max}	KLPG				Opt-KLPG				Comparison			
		#Overflows	Path length	CPU time[s]		#Overflows	Ave. #re-iterations	Path length	CPU time[s]		Δ Path length [%]	Gates with less paths	Gates with shorter paths
				Total	Excl SAT				Total	Excl SAT			
c1908 (795)	10	143619	75569	36	4	332249	42.25	86185	346	26	14.05	15	671
	50	215713	82819	76	8	280743	7.56	86185	337	22	4.06	1	352
	100	125803	85717	68	8	163067	2.46	86185	309	21	0.55	0	118
	500	13356	86155	82	13	29950	0.08	86185	306	42	0.03	0	4
	1000	14932	86181	122	30	23696	0.03	86185	343	77	0.00	0	2
	1500	16278	86181	177	56	20164	0.02	86185	386	122	0.00	0	2
	3000	12902	86181	375	186	12902	0.01	86185	526	252	0.00	0	2
c5315 (2228)	30000	0	86185	815	510	0	0.00	86185	774	490	0.00	0	0
	10	117690	201540	60	7	170043	8.02	215813	147	9	7.08	8	1306
	50	68354	215615	94	8	74667	0.97	215813	126	8	0.09	0	96
	100	33548	215770	111	9	35998	0.26	215813	131	8	0.02	0	21
	500	1402	215812	132	11	1402	0.01	215813	134	11	0.00	0	1
	1000	0	215813	134	11	0	0.00	215813	134	11	0.00	0	0
	1500	0	215813	134	12	0	0.00	215813	134	11	0.00	0	0
c7552 (2952)	3000	0	215813	134	11	0	0.00	215813	133	11	0.00	0	0
	30000	0	215813	134	12	0	0.00	215813	135	12	0.00	0	0
	10	210254	249957	141	17	258557	9.19	260443	324	22	4.20	25	1858
	50	102766	259465	211	17	110967	1.06	260443	277	20	0.38	0	382
	100	49843	260108	237	20	52150	0.30	260443	268	20	0.13	0	125
	500	509	260443	265	23	509	0.00	260443	264	22	0.00	0	0
	1000	0	260443	264	22	0	0.00	260443	265	22	0.00	0	0
b14 (6763)	1500	0	260443	264	23	0	0.00	260443	265	22	0.00	0	0
	3000	0	260443	269	23	0	0.00	260443	264	23	0.00	0	0
	30000	0	260443	264	23	0	0.00	260443	270	23	0.00	0	0
	10	214621	97653	127	15	1417923	21.03	126305	5688	336	29.34	19	931
	50	475719	113633	367	19	1360685	4.09	126305	5792	147	11.15	2	662
	100	573137	121862	490	23	1201388	1.84	126305	5255	114	3.65	0	550
	500	485778	125788	1745	48	761236	0.26	126305	5214	131	0.41	0	90
b15 (8931)	1000	378266	126103	2614	108	560882	0.11	126305	5559	246	0.16	0	38
	1500	309834	126169	3084	175	414736	0.05	126305	5378	320	0.11	0	32
	3000	221434	126252	3825	360	267347	0.02	126305	4984	511	0.04	0	14
	30000	0	126305	6959	2717	0	0	126305	7928	3566	0	0	0
	10	185349	50203	138	58	879949	9.9	69250	5268	944	37.94	61	700
	50	231739	61128	270	78	823864	1.88	69250	5249	903	13.29	15	530
	100	250348	62495	398	97	799232	0.93	69250	5306	941	10.81	15	484
p35k (23267)	500	252861	69093	1357	341	373079	0.09	69250	4406	850	0.23	0	51
	1000	179548	69188	2239	578	250773	0.03	69250	4727	1002	0.09	0	20
	1500	150366	69219	2948	803	198749	0.02	69250	5003	1215	0.04	0	12
	3000	78511	69223	4279	1404	97662	0.01	69250	5872	1910	0.04	0	10
	30000	0	69250	7151	3242	0	0	69250	8545	4244	0	0	0
	10	293796	7395	542	40	360685	1.83	8223	1218	55	11.19	0	852
	50	270197	7813	589	40	342143	0.36	8223	1329	51	5.25	0	219
p45k (25679)	100	263782	7869	701	39	359731	0.19	8223	2178	63	4.49	0	160
	500	242401	7988	1095	48	368222	0.04	8223	2959	95	2.94	0	116
	1000	268786	8091	1749	69	285228	0.02	8223	2644	92	1.62	0	87
	1500	216978	8150	1451	74	226846	0.01	8223	1710	92	0.90	0	51
	3000	159280	8196	1594	116	160445	0.00	8223	1887	150	0.33	0	40
	30000	0	8223	506	57	0	0.00	8223	489	55	0.00	0	0
	10	80796	3824	142	58	609588	2.68	4569	8048	647	19.47	17	699
p78k (70475)	50	122172	4376	261	68	480945	0.43	4569	7715	265	4.41	0	342
	100	115619	4494	470	82	425049	0.19	4569	6717	240	1.66	0	169
	500	111723	4555	1105	123	326295	0.03	4569	9029	549	0.30	0	50
	1000	100109	4562	1580	216	267548	0.01	4569	7105	947	0.15	0	32
	1500	102996	4563	1703	312	239735	0.01	4569	6102	1153	0.12	0	24
	3000	98656	4568	2898	849	195398	0.00	4569	7082	2276	0.03	0	12
	30000	7513	4569	20185	15750	7513	0.00	4569	20211	15756	0.00	0	1
p78k (70475)	10	166206	6342	240	101	689810	1.20	7149	7139	418	12.72	8	901
	50	177257	7039	391	112	501469	0.18	7149	5451	198	1.56	0	722
	100	99069	7147	335	108	373759	0.07	7149	5483	174	0.03	0	96
	500	135763	7148	703	125	321940	0.01	7149	5240	245	0.01	0	5
	1000	150340	7148	1168	180	299284	0.01	7149	7062	413	0.01	0	6
	1500	151226	7149	1344	208	283626	0.00	7149	5099	498	0.00	0	2
	3000	148676	7149	2257	461	243633	0.00	7149	5545	927	0.00	0	2
30000	0	7149	10590	5859	0	0.00	7149	10560	5840	0.00	0	0	

Tables 2.2 and 2.3 contain the comparison between KLPG and Opt-KLPG (as in the final three columns of Table 2.1) under non-robust sensitization conditions (as opposed to functional sensitization in Table 2.1) for two alternative delay models: gate delays extracted from the technology library and unit-delay model which assumes that each gate has a delay of 1. The influence of the model assumptions tends to be limited. Similar observation were made for further benchmarks which are not reported here.

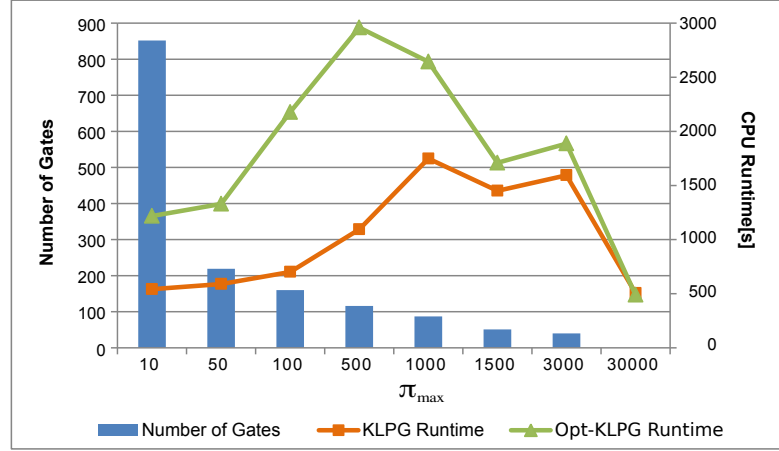


FIGURE 2.16: Experimental results for circuit p35k. Runtimes are shown by lines; number of gates with shorter paths are shown by bars.

TABLE 2.2: Comparison of KLPG and Opt-KLPG under different model assumptions for circuit c7552

π_{max}	Non-robust sensitization			Unit delay, functional sens.			Unit delay, non-robust sens.		
	$\Delta path$ length[%]	#Gates w. less paths	#Gates w. short paths	$\Delta path$ length[%]	#Gates w. less paths	#Gates w. short paths	$\Delta path$ length[%]	#Gates w. less paths	#Gates w. short paths
10	3.95	19	1603	4.22	32	1638	4.60	33	1755
50	0.24	0	197	0.40	0	407	0.37	0	354
100	0.10	0	81	0.12	0	118	0.12	0	106
300	0.00	0	2	0.00	0	7	0.00	0	2
500	0.00	0	0	0.00	0	0	0.00	0	0
1000	0.00	0	0	0.00	0	0	0.00	0	0
1500	0.00	0	0	0.00	0	0	0.00	0	0
3000	0.00	0	0	0.00	0	0	0.00	0	0
30000	0.00	0	0	0.00	0	0	0.00	0	0

TABLE 2.3: Comparison of KLPG and Opt-KLPG under different model assumptions for industrial circuit p45k

π_{max}	Non-robust sensitization			Unit delay, functional sens.			Unit delay, non-robust sens.		
	$\Delta path$ length[%]	#Gates w. less paths	#Gates w. short paths	$\Delta path$ length[%]	#Gates w. less paths	#Gates w. short paths	$\Delta path$ length[%]	#Gates w. less paths	#Gates w. short paths
10	22.17	25	698	17.12	19	673	21.02	27	676
50	5.59	1	358	4.36	0	310	6.11	2	338
100	2.01	1	180	2.08	0	156	2.79	1	178
300	0.51	1	62	0.67	0	57	0.76	0	58
500	0.33	1	44	0.47	0	47	0.50	0	44
1000	0.13	0	23	0.27	0	32	0.25	0	28
1500	0.08	0	14	0.21	0	27	0.12	0	19
3000	0.01	0	5	0.08	0	13	0.02	0	5
30000	0.00	0	0	0.00	0	0	0.00	0	0

2.6.2 Fault-grading Procedure using Opt-KLPG

The variation-aware fault-grading procedure was performed for the combinational cores of industrial NXP circuits synthesized using the Nangate 45nm open cell library [68]. To obtain delays appropriate for state-of-the-art 22nm technology, the observation time point t_{obs} and the nominal rising and falling delays of every gate were scaled by 0.75 [69].

The delay of each gate was modeled by a Gaussian distribution with the mean equal to the scaled nominal delay and the variance of 20%. Instance 0 was assumed to have no variations thus every gate in the instance has the nominal delay. In other instances, a fixed delay was randomly chosen from the distribution for every gate. In total, one nominal circuit and 100 circuit instances affected by variations comprise the set of instances, referred by I_{100} . A fault list was used that consists of 100 randomly chosen fault locations and 9 fixed fault sizes $(0.1 \cdot t_{obs}, \dots, 0.9 \cdot t_{obs})$. In total, $(1 + 100) \cdot 100 \cdot 9 = 90900$ simulation runs were performed for a two-pattern test.

The initial test set T_1 was generated by a commercial ATPG tool targeting on TFs at the fault locations in the nominal circuit instance. To compare the effectiveness of the traditional n -detection ATPG [70]³ to the ATPG applied by the procedure, 5-detect test set T_5 was generated using the same tool. Moreover, a synthesis tool was used to determine t_{obs} under pessimistic safety margins implicitly accounting for process variations while the generation of T_1 and T_5 does not take process variations into account.

Using the timing-aware path-oriented ATPG (purely SAT-based ATPG and Opt-KLPG), the test set T_{ATPG} was generated for faults uncovered by the initial test set T_1 . The number of tests in each test set (T_1 , T_5 and the union of T_1 and T_{ATPG}) and the corresponding SFC obtained by fault simulations for several NXP circuits are listed in Table 2.4. Column 1 and 2 present the circuit name and the total number of gates in the circuit, respectively.

Columns 3 through 6 give the number of tests and the obtained SFC by fault simulations in circuit instances I_{100} for T_1 and T_5 , which are generated for the TFs. It can be seen that the SFC achieved by these variation-unaware test sets is rather low. Furthermore, using T_5 can only achieve a marginally better SFC than using T_1 for all circuits except p35k, which has very low SFC values for both test sets anyway. The very limited improvement in SFC by using n -detection technique shows that traditional variation-unaware ATPG methods might be very insufficient in testing for nanoscale technologies affected by massive variations.

The last three columns report the performance of the initial set T_1 augmented with T_{ATPG} generated by the variation-aware test flow for faults not covered by T_1 , in terms

³The n -detection TF ATPG aims at generating the test set that detects each TF at least n times (if existing).

of the total number of tests (Column 7) and the SFC values obtained by the fault-grading procedure (Column 8 and 9). A significant improvement of SFC can be observed that the SFC is almost doubled compared to the value for T_5 (generated using n -detection technique). To verify these results, the generated test pairs were applied to a new collection of 200 random circuit instances I_{200} . The SFC measured for this new set of circuit instances (Column 9) has only minimal difference to the results for I_{100} (Column 8), which underscores the validity of the variation-aware tests. Note that the absolute numbers of SFC are rather low. The reason is that no redundancy proof engine was integrated into the flow to rule out the undetectable faults, which can constitute a large portion of the undetected faults and thus lower the SFC significantly.

TABLE 2.4: Statistical fault coverages for 101 instances, 100 random faults and 9 fault sizes

Circuit	#Gates	Test sets for TFs				$T_1 \cup T_{ATPG}$		
		T_1		T_5		I_{100}		I_{200}
		#Tests	SFC	#Tests	SFC	#Tests	SFC	SFC
p35k	23267	216	0.50	838	5.17	16997	29.23	29.01
p45k	25679	53	17.12	195	18.84	4305	29.16	28.86
p78k	70479	31	34.06	69	35.50	6333	54.37	54.16
p89k	58638	44	14.69	178	16.80	5650	27.65	27.35
p100k	61006	38	16.22	133	17.41	8080	29.02	28.85

It is reported in [63] that using the test compaction technique the variation-aware tests can be reduced to a size that is comparable to the size of the n -detection sets without lowering the detection capability. In other words, using a variation-aware test-generation flow can significantly improve SFC with pattern counts similar to n -detection test sets. The results are not reported here as the compaction technique is irrelevant to the contribution of this thesis work.

In summary, experimental results show that Opt-KLPG, as an enabler technology, helps in detecting SDDs under process variations with much higher probability, compared to conventional ATPG technologies such as N-detection.

Chapter 3

MIRID: Mixed-mode IR Drop Induced Delay Simulator

3.1 Motivation

Deep-submicron CMOS technology tends to increase transistor density significantly due to scaling of feature sizes and growing demands for transistors. Furthermore, the requirement for high performance of VLSI integrated circuits implies a raised clock frequency. Both factors, the ultra-high transistor density and the raised frequency, lead to a power density problem that requires a large amount of current from the PDN. As a result, power supply noise rises that leads to reduction of the actual voltage levels supplied to the gates. Excessive power supply noise can degrade circuit performance by inducing additional signal delay, or even lead to functional failure of logic gates [49]. Therefore, power supply noise has become a critical concern in chip manufacture for reliability and high performance.

Compared to package interconnect, PDN includes predominantly resistive elements, such as metal wires and vias to deliver power throughout the chip, but as well as capacitive and inductive parasitic elements. Two fundamental sources of power supply noise are inductive voltage noise and resistive voltage noise, commonly denoted by Ldi/dt and IR , respectively. The former depends on the rate of change of the instantaneous current flowing through parasitic inductive elements of PDN while the latter depends on the instantaneous currents flowing through the resistive elements of the network [71]. As

PDN is predominantly resistive, the voltage drop due to IR noise, called IR drop, is especially important for the supply network [72]. In this thesis work, IR drop in on-chip PDN is focused on.

IR drop has been studied in both design and test domains. In context of PDN design optimization, the purpose is to estimate the IR voltage noise and reduce overall IR drop effect as much as possible at the chip level. Different supply network and circuit models have been proposed [52, 73–75] to estimate power supply noise due to IR drop. These models can be used to identify critical areas of the chip and provide information that helps the designer by improving supply network design. Most of these works targeted on predicting the spatial effects of IR drop using statistic models and performed analysis based on SPICE-like electrical models. However, such analysis is almost infeasible for pattern-dependent approaches due to the prohibitive simulation cost in terms of anticipating all possible operational conditions on the chip to establish an electrical model required by an accurate estimation at the chip level.

Approaches in test domain concern not only the spatial effect but also the temporal effect induced by IR drop. Simulations applying test patterns are necessary to verify whether the chip present any functional failure due to excessive delays induced by IR drop. Furthermore, in context of SDD testing under process variation, estimation of IR drop induced delays is important for evaluating test patterns that sensitize paths through targeted SDDs. These paths are commonly selected based on nominal gate delays whereas extra delays due to supply noise can cause variations in circuit timing and thus affect the fault coverage. By estimation of the extra delay for generated test patterns, these exacerbating IR drop effect are preferred for a higher fault coverage. Moreover, power-aware test is an important concern that requires reduction of power dissipation in circuits during the test mode. For example, minimization of IR drop effect is essential for scan testing. It is well known that switching activities tend to increase during the test cycle and result in supply noise that affects the fault coverage. Thus peak power consumption during test cycle of scan testing has to be controlled to avoid noise phenomena [76]. For all the purposes given above, an accurate pattern-dependent simulation of IR drop induced delay is required. However, due to the high costs of simulation and memory constrains, grossly simplified electrical models were used by most of the approaches in test domain. The challenge here is to develop more accurate and yet efficient pattern-dependent simulation of IR drop induced delays.

In brief, simulation at electrical level is prohibitively expensive for large circuits and pattern counts thus a vector-based simulation is almost infeasible by electrical simulations. However, to estimate IR drop induced delay accurately for applied test vectors, the electrical effects have to be considered by the time-aware simulation of logic blocks. As a trade-off between the accuracy and efficiency, a mixed-mode simulator can be considered that takes advantage of the efficiency of the logic simulation and the accuracy benefited from integration of electrical models.

3.1.1 Mixed-mode Simulation

In previous works, a number of mixed-mode simulation technologies that integrate electrical models into simulations at high levels have been proposed. In [77], a simulation method for CMOS bridging faults is proposed, which employs a set of tables derived by using electrical simulation tool (SPICE [78]) to exactly characterize bridging faults. The **I**nductive **F**ault **A**alysis (**IFA**) approach is described in [79] that characterizes faults by drawing conclusions based on analyzing the particulars of low-level fault-inducing mechanisms and simulates the extracted faults. An extension of IFA with a new simulation-based fault modeling methodology is presented in [80] that uses the three-dimensional contamination-defect-fault simulator CODEF to directly relate effects of process-induced contamination to circuit-level malfunctions.

In context of simulating IR drop induced delay, a mixed-mode simulation that combines a timing-aware logic simulation with electrical models of the power supply network, named **M**ixed-mode **I**R-drop **I**nduced **D**elay Simulator (**MIRID**) [81], is developed by this thesis work. For the first time, the PDN configuration is integrated into the mixed-mode simulation, which is necessary for simulating electrical phenomena related to the supply networks. This is a joint work partially funded by a French-German collaboration project¹. The electrical models employed by MIRID are proposed by the project partner, the team led by Prof. Renovell [55]. They also provide the pre-characterization library, based on which the electrical models are established. The contribution of this these work includes the implementation of the mixed-mode simulation flow, along with appropriate interfaces to electrical models, and the integration of electrical models into

¹BFHZ project (FK 39-10) of University of Passau and LIRMM (Laboratory of Informatics, Robotics and Microelectronics, Montpellier, France)

the simulation flow based on the provided pre-characterization library. Moreover, experiments were performed that help in validating the accuracy of employed electrical models and evaluating the performance of the simulator for large circuits.

The remainder of this chapter is organized as follows: 3.2 introduces basic concepts about PDN and IR drop effect as well as the state-of-the-art approaches of power-aware test; 3.3 gives a brief overview of the simulation algorithm; the employed electrical models are presented in 3.4; implementation of the simulator is detailed in 3.5; 3.6 introduces the application of MIRID for mitigating the short-path problem in a self-adaptive design; finally, 3.7 presents the experimental results.

3.2 Preliminaries

3.2.1 Power Distribution Network (PDN)

The on-chip supply nets, consisting of the *power* (Vdd) and *ground* (Gnd) nets, connect each cell (the logical or functional unit built from various components in the chip design) to the supply source. As each cell must have both Vdd and Gnd connections, the supply nets are large enough to span across the entire chip, and are routed first before any routing of signals that connect between components or to the external environment. In many applications one power net and one ground net are sufficient while some ICs, such as mixed-signal or low-power designs, can have multiple power and ground nets [82].

Typically power and ground nets have multiple dedicated metal layers to avoid consuming signal routing resources. Between the layers sufficient *vias*, connections of routing structures on different layers, are used to carry current while avoiding reliability issues like *electromigration* [83].

A commonly used structure of the PDN in integrated circuits of high complexity and high performance is the grid structure, which is also used by the electrical model in this thesis work. Each layer of the grid consists of many equidistantly placed power and ground lines of equal width. The direction of lines within each layer is orthogonal to the direction of lines in the adjacent layers. Each power and ground line is connected by vias to other power and ground lines in the adjacent layers at the overlap sites,

respectively. Typically, the lower the layer is, smaller are the width and pitch of lines [82]. An example of a two-layer power distribution grid is illustrated in Figure 3.1.

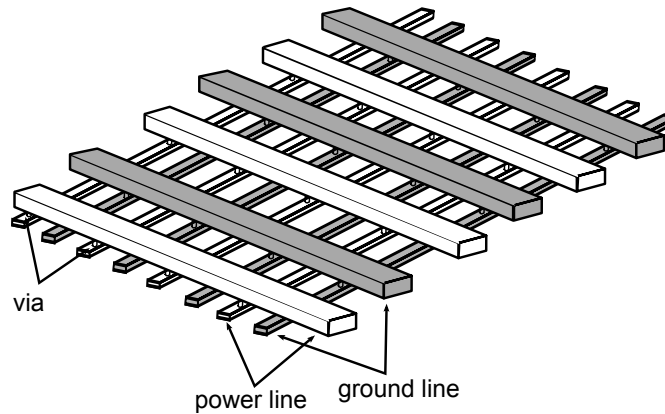


FIGURE 3.1: A two-layer power distribution grid

3.2.2 IR Drop Effect

A chip can contain various resistive elements, such as resistance of the on-chip wires and vias, resistances of the bond wires or solder bumps to the package, resistances of the package planes or traces, and resistances of the printed circuit board planes. Because the package planes and printed circuit board typically have much lower resistance than on-chip wires, the on-chip network dominates the resistive drop [71]. IR drop is commonly associated with the resistive element in power supply network. It refers to the voltage drop across the resistance R of a power supply grid between the supply pin and the block drawing current I from the networks; the voltage drop is given by $I \times R$ according to the Ohm's law.

Static and dynamic IR drops can occur on a chip. The former is the average voltage drop for the design whereas the latter depends on the instantaneous current caused by switching activities in the logic block [53]. The instantaneous current flowing through resistive elements in the power supply network can be much higher than the average current demand. This is because spikes of current draw are likely to occur locally near the clock edge when many gates switch simultaneously. Due to raised transistor density and functional frequency with the scaled technology, more gates tend to switch simultaneously and in turn demand a large amount of instantaneous current draws from the supply networks. Because of the high instantaneous current draws, fluctuations of voltage levels across inherent parasitic elements of PDNs are produced. Gates can be

powered with a lower-than-normal Vdd and/or higher-than-normal Gnd, which results in an increased gate delay. Moreover, as technology scales, gate delay becomes more sensitive to power supply noise. It was reported that in 180nm technology voltage fluctuations of 10% increases gate delay by 8% [84], while in 130nm technology fluctuations of 10% can cause up to 30% increase in gate delay [85] and even 1% fluctuation of power supply adds nearly 4% of gate delay in 90nm technology [51].

Bypass capacitances near the switching gates² can supply much of the instantaneous current, and thus only low enough resistance are needed to deliver the average current demand. This is not necessarily true for the peak current. Concerning SDD testing, when many gates in a logic block switch simultaneously, current draws that are not compensated by the bypass capacitances can lower the voltage supplied to gates thus cause extra delays, which in turn affect the fault coverage for the applied test patterns.

3.2.3 Power-aware Test

Modern CMOS circuits are often required to have high performance and transistor density, however, this comes at the cost of increased power dissipation. Potential consequences are higher system costs (packaging/cooling) and, for battery operated embedded systems, shorter battery lifetimes. This trend forces the designer to be power-aware, i.e., to develop various power managements and low-power-design techniques.

Furthermore, power consumption is not only an important design dimension, but also a major manufacturing test consideration. In fact, concerns about power consumption stand out during various test modes of circuit operations, not just restricted to the functional mode [86]. Generally, a circuit consumes more power in test mode than in functional mode, mainly due to the higher switching activity in test mode and parallel testing used to reduce test application time [87]. (It has been shown that the power consumed during test could be twice as high as the power consumed during normal operations [88].) Excessive test power could induce problems related to higher temperature and current density during test, such as extra cost for removal of excessive heat and electromigration [83]—an electrical effect that decreases the CUT reliability.

²Bypass capacitances are often placed between supply connections of circuit elements to supply transient current and thus mitigate the supply noise [82].

This phenomenon is even more severe for circuits with built-in self-test (BIST) architecture, which might be tested frequently, especially for battery-powered portable systems. Moreover, the recent trend to implement high-performance, low-power devices that operate within specified power consumption budgets makes the power management a critical parameter in test development, e. g., the Razor processor proposed by Ernst et al. [5]. Reduction of test power is therefore required to save product cost, reduce difficulties in performance verification and enhance the circuit reliability.

Addressing the problem of power-aware tests (tests that can ensure reduced power consumption), test techniques in various categories have been proposed. The first category includes the power-aware ATPG techniques that generate test patterns able to reduce the test power in addition to meeting classic ATPG objectives. Wang and Gupta [89] propose a new version of the conventional ATPG algorithm PODEM [40] where the clever assignment of *don't-care* bits minimizes the number of transitions between two consecutive vectors; the power dissipation during test application is therefore decreased. Another ATPG technique proposed in [89] reduces power dissipation during the test of sequential circuits. The proposed approach exploits some redundancy introduced during the test pattern generation and selects a subset of sequences that reduces the consumed power without lowering the fault coverage. The other category consists of ordering techniques that reduce the switching activity, by modifying the order in which test vectors are applied [90–92] and, for scan based circuits (circuits with a scan path structure), the order in which scan flip-flops are chained [93]. Power dissipation during scan test can also be reduced by modifying the conventional test vector compaction technique, e. g., selecting the merging order of test cube pairs during static compaction [94], and transforming the conventional scan architecture [95]. Moreover, as demonstrated in [96], the clock tree is a major contributor of the test power. Techniques that modify the clocking scheme are proposed to minimize the average and peak power dissipation, by disabling clocks of some scan chains for portions of the test set [97] and using a gated-clock scheme for the scan path and the clock tree feeding the path [98]. In addition to the external testing techniques mentioned above, various BIST techniques have been proposed to cope with the power problems during BIST, such as by scheduling the execution of every BIST element [88], by modifying test pattern generators [99, 100] and by partitioning the circuit for parallel BIST [101].

Estimation of test power consumption is important to evaluate the reduction of test

power by power-aware tests. Methodologies for the evaluation haven been developed and widely employed by commercial tools, such as PrimeRail [102] and RedHawk [103]. With respect to IR drop effect, the RedHawk tool provides a dynamic simulator which uses an internal statistical approach towards dynamic IR drop analysis. However, the approach is vectorless, which means, it aims at an efficient computation of the upper bound on the worst-case drop, but not input-vector dependent and thus cannot be used to identify chips with functional failures caused by excessive delay due to IR drop.

3.3 Simulation Overview

The aim of the simulator is to validate test patterns applied to the logic block on a chip with consideration of IR drop effect as accurately as possible while within a reasonable simulation time. As it is almost infeasible to perform electrical simulation of a larger logic block due to the very high simulation cost, the simulation core is developed using logic simulation algorithm [2, p. 64], which is usually performed at the gate level and much faster than the electrical simulation.

3.3.1 Event-driven Logic Simulation

In logic simulation, the logic behavior as well as delays of each logic element are considered, to imply the logic values at circuit lines and/or timing of each logic value transition and hazard caused due to the applied input pattern. Traditionally, two algorithms have been used in logic simulation: “compiled-code” and “event-driven” algorithms [104]. The former evaluates every logic element at each time step while the latter evaluates a logic element only if one of its input has changed, that is, an event has occurred. It has been reported that the event activities in a large circuit is extremely low [105], and the percentage activity decreases with the increase in circuit size [106]; thus, the event-driven simulation has the advantage of saving much of the simulation execution time in large circuits.

Moreover, the increase in complexity of modern VLSI design could raise the simulation execution time enormously. Thus, parallelism has become an important consideration by developing event-driven logic simulation algorithms. Two approaches have been pursued on that purpose. One is based on a parallelization of the “synchronous” simulation

algorithm; the other is a class of “asynchronous” (or known as “distributed”) algorithms [104]. The synchronous simulation algorithm handles all events in the order of their times; the simulation time does not advance until all (simultaneous) events in the current simulation time are processed. The simultaneous events can be processed in parallel, which requires that the circuit (data structures representing the logic elements) and the event list can be divided among different processors. In asynchronous simulation there is no global simulation time control, instead each value assignment carries its own time information known as a time stamp. The evaluation of a circuit element occurs when all the input tokens (a data value and its time stamp are referred to as a token) are available. The lack of global time control allows asynchronous algorithm to concurrently process events that belong to different instances of time. Asynchronous logic simulation can bring more performance benefits compared to synchronous simulation. For example, two independent gates can be evaluated concurrently using asynchronous simulations while they have to be evaluated sequentially using synchronous simulations. Despite the potential benefits of the asynchronous algorithm, it has its limitation, e. g., deadlocks [104]—none of the elements are able to evaluate because all of them have at least one missing token on an input—could occur in circuits with feedbacks.

In context of simulating IR drop induced delay, a global time control is essential because the event processing includes not only the evaluation of logic elements, but also the current distribution throughout the supply networks due to the switching, which further affects the voltage supplied to the next switching gate and its delay. In that sense, there are no two switching events totally independent. Thus a synchronous event-driven simulation algorithm is preferred by the logic simulation. Furthermore, parallelization in processing simultaneous events by the synchronous simulation is possible but might not help much in improving the performance. In that case, the current distributed over PDN need to be updated additively for all changes due to the simultaneous events before the global simulation time is advanced. Moreover, the probability of simultaneous switching events is very low for the timing resolution of 1ps used by MIRID; among such a small timing slot not many switching activities could take place and thus be said to be “simultaneous”. Consequently, considering the limited effectiveness of parallelism (in improving the simulation performance) and its complexity (in partitioning the circuit optimally onto available processors to achieve the maximized runtime concurrency and

the minimized interprocessor communication), MIRID was implemented without applying parallelization techniques. However, there is still possibility of extending MIRID, which currently targets on simulation in a logic block, to parallel simulation of different logic blocks on the chip. This possibility is discussed in 3.4.2.

3.3.2 Interfaces to Electrical Models

As given in 3.2.2, IR drop is an electrical phenomenon related to the PDN structure and current distribution in resistive elements in the networks. The impact of IR drop on the time-aware simulation is reflected in extra gate delays induced by power fluctuations in PDNs due to instantaneous current, drawn by switching activities in the logic block. Thus the simulation requires an electrical model of PDNs that defines the network topologically and estimates the current demanded by the switching gates and distributed throughout the networks. Furthermore, delays of switching gates depend on the supplied voltage levels and other electrical parameters; an electrical model that approximates the gate delays is required as well. As illustrated in Figure 3.2, the simulation implementation consists of the logic simulation engine and two electrical models for PDNs (PDN model) and the circuit timing (timing model).

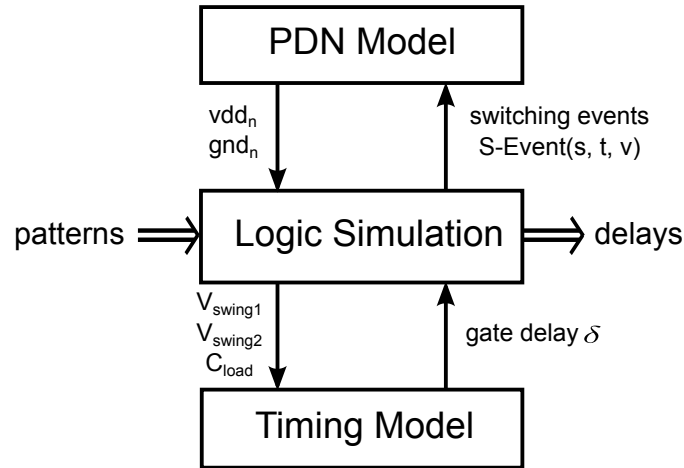


FIGURE 3.2: Schematic illustration of the simulator

A set of two-pattern tests *patterns* is applied to the logic block containing gate instances and interconnects between them. The simulation is time-aware and reports the propagation delays of transitions *delays* induced by the applied two-pattern test. Moreover, currents flowing through resistive elements in PDNs and voltage drops resulted after each switching activity can be optionally recorded. Two time modes are implemented by the

simulation algorithm, that is, the gate delays are either given by the nominal timing assignment, or determined dynamically according to the active simulation environment considering IR drop effect. For the former, the implementation functions like a traditional time-aware event-driven simulator without integration of the electrical models. For the latter, delays induced by IR drop are computed according to electrical models, and used to update the execution time for next events. For each switching event emitted by the logic simulation $S\text{-event}(s, t, v)$, where s is the switching input of a gate and v is the logic value newly assigned at time t , voltage levels on supply nodes connected to the gate, denoted by vdd_n and gnd_n , are returned by the PDN model. The timing model is established to determine gate delay according to the active simulation environment. The gate delay δ might differ from its nominal value due to the IR drop effect. It is obtained by the interface function between the logic simulation and the timing model, which takes parameter values from the active simulation environment, such as the voltage swing of the upstream gate V_{swing1} , the voltage swing of the switching gate V_{swing2} and the load capacitance of the switching gate C_{load} . New switching event $S\text{-event}(s_{out}, t + \delta, v')$ will be generated after the delay δ if the output of the switching gate s_{out} is assigned with a new value v' . Events generated during the simulation are inserted into an event queue, sorted by the time and processed in this order. The simulation flow terminates after processing all events in queue iteratively, which implies that the queue becomes empty.

The simulation design has a two-fold goal: the accuracy in delay prediction and the efficiency in simulation runtime. On one hand, the simulation accuracy highly depends on the employed electrical models and can be improved by incorporating more electrical parameters into the models. Thus a versatile interface design is important to make sure that least effort will be required by the adaption to more complex and accurate electrical models. The principle of the interface design is to define interface functions that only return necessary values required by the logic simulation while the function bodies can be modified without/least affecting the simulation core. On the other hand, as the simulation runtime mostly depends on the complexity of the applied electrical models, possibly simplified electrical models without much loss of accuracy in the gate delay prediction are preferred. Two aspects are especially considered by establishing the electrical models:

- In the simulated logic block, SPICE-like simulation at transistor level is complex

and unnecessary in context of estimation of IR drop induced delay. Not all electrical parameters are related closely to the IR drop phenomenon and thus can be neglected by the estimation. Only two parameter are of interest here: the current draw caused by the switching activity and the gate delay.

- In PDNs, the voltage fluctuations need to be estimated accurately since the gate delay depends directly on the supplied voltage level. Hence an accurate electrical model is required for PDNs. The most relevant parasitic elements should be included in the model.

Furthermore, currents flowing through the resistive elements, which determine the voltage fluctuation, should be monitored during the simulation. That means, the current flowing through each resistive element as a function of time has to be known for the computation of voltage levels supplied to a switching gate. Initially no current is assumed to flow throughout the PDN grids when each signal in the circuit stabilizes at a logic value after the first test pattern is applied. As a gate switches due to the second applied pattern, it draws currents from the PDN grids; the current waveforms in grid resistors need to be updated correspondingly. In addition to the current waveforms in PDN grids, an accurate computation of voltage requires the grid resistor information to be extracted as accurately as possible, and presented by resistances in different areas according to the actual configuration. The mapping between power supply nodes and gates is also important as it determines the current draw caused by switchings combined with the topological structure of the PDNs. As given in 3.2.1, two independent but very similar PDNs, Vdd and Gnd networks that distribute power and ground voltages to chip cells, respectively, are designed for ICs. Then, each gate can be mapped to Vdd and Gnd supply nodes; the voltage levels supplied to the gate at the corresponding nodes are considered by the estimation of gate delays.

In experiments, based on a pre-characterization library containing results of SPICE simulation given a PDN configuration, the employed electrical models are established. Due to the time constrains, the capacitive and inductive elements are not considered by the configuration. Interface functions between the logic simulation and electrical models are derived according to the PDN configuration and the pre-characterization library. The generic interfaces can be adapted for more accurate electrical models, for example, the PDN configuration that includes capacitive elements.

3.4 Electrical Models

As given above, the logic simulation requires interface functions of mapping the logic gates to power supply nodes, the voltage levels on power supply nodes, and the gate delay induced by IR drop. The implementation of these functions highly depends on the PDN configuration including the structural assumption and the distribution of parasitic elements, and the used CMOS technology. An accurate simulation of the voltage drops is possible only when the currents drawn by switching gates and distributed in PDN grids are estimated accurately. Aiming at simulating IR drop induced delays for applied input patterns, a pre-characterization of standard gates in terms of the current draw and the delay due to switching activities, as well as the pre-characterization of the current distribution in the PDN are required to derive the functions. The electrical models introduced in this section are proposed by our collaboration partners from LIRMM in Montpellier, France [55].

In the remainder of this section, PDN configuration used by the electrical model is introduced in 3.4.1; the pre-characterization of current distribution factors throughout the PDNs is detailed in 3.4.2; electrical models at the gate level evaluating current draw and gate delay for switching gates are presented in 3.4.3.

3.4.1 PDN Configuration

3.4.1.1 Structural Assumption

PDN is a complex system that delivers power to the whole integrated circuit containing many layers. Usually it is organized as a set of parallel large wires located in the upper metal layers covering the whole circuit surface [54]. The topological assumption of the PDN model highly depends on the physical structure. In this thesis work, the PDN arrangement is considered as a simplified structure of three levels, which are given as follows:

- At the top level, two sets of parallel metal lines are placed orthogonally in two layers one upon the other. An example is illustrated in Figure 3.3 from the top view. Every two parallel lines dedicate to a Vdd line (the white stripe) and a

Gnd line (the grey stripe). Vdd and Gnd lines in the layer above are connected to the Vdd and Gnd lines below by vias (the small circles), respectively. In this way, Vdd and Gnd distribution networks can be considered as two independent two-dimensional networks³. This structure corresponds to the high metal levels of the chip that consists of long metal lines spreading over the whole chip size.

- Intermediate metal layer connects the layers at the top/bottom level. Parallel metal lines are placed regularly over the chip.
- In the bottom metal levels of the chip, Vdd and Gnd lines typically have smaller lengths and form irregular shapes corresponding to logic cells they feed. Parallel multiple vias are often used to connect the intermediate layer.

Among the parasitic elements implied by the PDN structure, resistive elements are predominant compared to the capacitive and inductive ones. Therefore, an accurate representation of resistive elements is important for the PDN modeling. As given above, the top level of the structure is comprised of long parallel wires and vias spreading over the whole chip. Compared to the reduced resistive behavior of metal lines in the bottom level due to much smaller lengths and multiple parallel vias, the resistances at the top level are determinant for the current distribution throughout the network, which in turn mainly contributes to the IR drop phenomenon. For this reason, the top level can be modeled as two-dimensional resistive grids while the low level is neglected. The intermediate level has similar structure to the top level and thus can be represented by the model too.

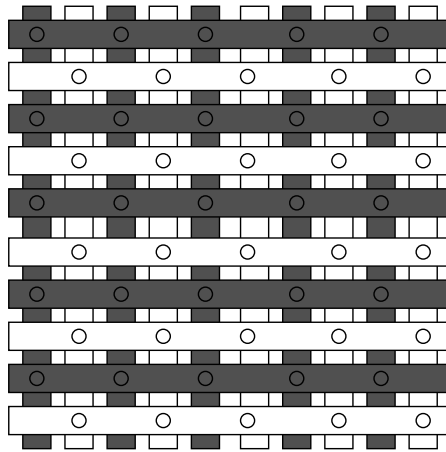


FIGURE 3.3: Two sets of Vdd and Gnd lines

³In this example, each PDN is represented by a 4×4 grid.

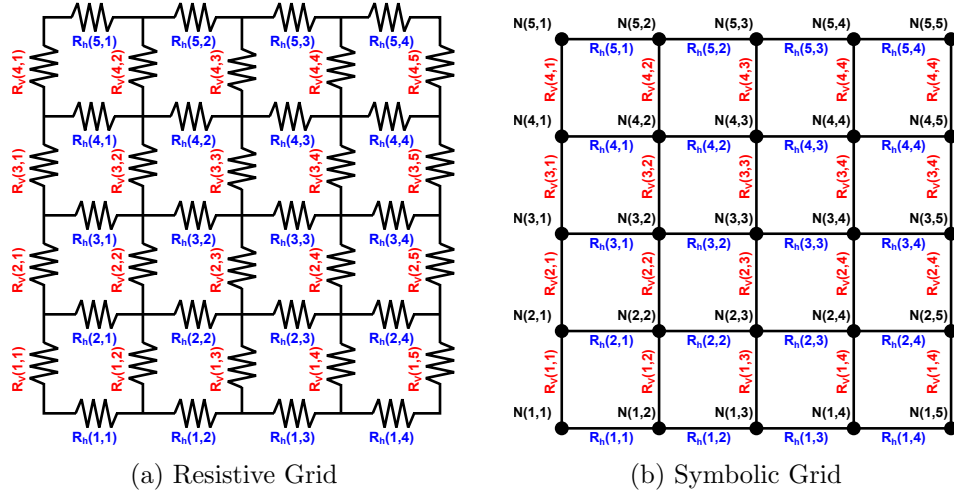


FIGURE 3.4: PDN grids

In brief, the electrical model of PDNs contains two independent power supply networks, which have symmetrical structures of grids with resistive elements. The capacitive and inductive elements are not included in the first version of the PDN model applied by the experiments. A future version can consider the capacitive elements to raise the precision of the IR drop estimation while the inductive elements can be neglected since their impacts on the IR drop are typically very small.

3.4.1.2 Parasitic Elements in PDN

The representation of resistive elements in the model depends on the PDN topology, the technology and the types of metal in the PDN. Based on the extracted resistance values from the PDN metal levels by commercial design tools, the value of each grid resistor can be determined. Commonly it can be assumed for the grid structure that all grid resistances have the same value.

A 4×4 resistive grid is presented in Figure 3.4a. It contains 5×4 and 4×5 resistances parallel placed in horizontal and orthogonal directions, denoted by $R_h(x, y)$ ($1 \leq x \leq 5$, $1 \leq y \leq 4$) and $R_v(x, y)$ ($1 \leq x \leq 4$, $1 \leq y \leq 5$), respectively. To simplify the presentation, a symbolic grid, where each segment implicitly represents a resistance at the corresponding position, is used in the remaining text of this thesis. Figure 3.4b illustrates an example of the 4×4 grid. The nodes at the crossings of lines represent vias in PDNs, through which the logic block is connected to the supply networks. Resistance of vias are neglected by the model due to their small values. Each segment between nodes

$N(x, y)$ and $N(x, y + 1)$ represents the horizontal resistance $R_h(x, y)$ while each segment between nodes $N(x, y)$ and $N(x + 1, y)$ represents the vertical resistance $R_v(x, y)$.

3.4.1.3 Mapping to the Logic Block

The **Block Under Test (BUT)** on the chip is represented by a set of gates and the interconnects. Each gate is mapped to certain power supply nodes in Vdd and Gnd networks depending on the chip design. As PDNs are modeled as two independent resistive grids, the mapping is given by $G_i \rightarrow Vdd(x, y)$ and $G_i \rightarrow Gnd(x', y')$ where G_i is a gate with index i , $Vdd(x, y)$ and $Gnd(x', y')$ are nodes located at coordinates (x, y) in Vdd network and (x', y') in Gnd network, respectively. Note that the assigned nodes to a gate in Vdd and Gnd grids are not necessarily at the same coordinates.

Mapping of supply nodes to gates can have significant impact on the IR drop effect. When many gates connected closely in the logic block switch almost simultaneously, the voltage drops can be high due to a large sum of currents flowing through adjacent nodes in PDNs. Considering the example illustrated in Figure 3.5, gate G_1 is connected to nodes $Vdd(x, y)$ and $Gnd(x', y')$ while its downstream gate G_2 is connected to nodes $Vdd(n, m)$ and $Gnd(n', m')$. When the upstream gate G_1 switches, instantaneous current draws appear at its supply nodes $Vdd(x, y)$ and $Gnd(x', y')$. The current flowing through each grid resistor is updated by adding the distribution of the instantaneous current. The closer the nodes are to the location where the instantaneous current appears, the higher the corresponding distribution factors for the nodes. Gates located nearby tend to be connected to adjacent supply nodes or even to the same node due to space constraints of the chip design. Then assuming that $Vdd(x, y)$ and $Vdd(n, m)$ are adjacent in Vdd grid, and so are the nodes $Gnd(x', y')$ and $Gnd(n', m')$ in Gnd grid, the instantaneous currents due to switching of G_1 have a high distribution to the raised currents at $Vdd(n, m)$ and $Gnd(n', m')$, which might hardly decline for a little while. G_2 can switch very shortly and result in even higher currents flowing throughout the neighboring area, followed by exacerbated voltage drops.

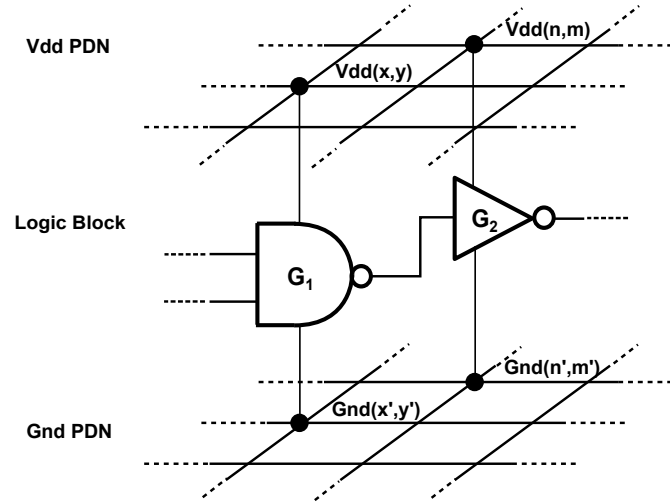


FIGURE 3.5: Gates connected to Vdd and Gnd power supply nodes

3.4.2 Current Distribution in PDN Grids

The instantaneous current caused by a switching event dissipates throughout the resistive grid, and affects the voltage drops in the neighboring area. In area far from where the instantaneous current appears, the current distributed in resistances is very small thus its impact on voltage drops can be neglected. To accurately simulate the impacts of the current draw on voltage drops, the current distribution in PDN grids (or only in neighboring area for the sake of efficiency) needs to be estimated as precisely as possible. A predefined area called current window is given by establishing the electrical models. In this area, the current-distribution factors for all grid segments are calculated. Though a closed-form mathematical calculation would be ideal, it is difficult to derive equations due to the non-trivial grid topology. Instead the fractions of current distribution are determined by pre-characterization of the grid using the simulation tool SPICE.

The SPICE simulation is performed for a resistive grid with nominal supply voltage Vdd at the edges. The approach can be easily adapted to the Gnd PDN by setting the supply voltage to the nominal ground value. Figure 3.6a presents an example to explain the principle of the approach. A 4×4 PDN grid is supplied by nominal power voltage Vdd at the surrounding edges. The current window has the same size as the grid. Considering a switching inverter connected to the grid node $N(i,j)$, the current draw $I(i,j)$ due to the switching activity spreads over the whole grid. The distribution of the current in horizontal and vertical resistances in the current window, denoted by $I_h(x,y)$ ($1 \leq x \leq 4, 1 \leq y \leq 3$) and $I_v(m,n)$ ($1 \leq m \leq 3, 1 \leq n \leq 4$), respectively, need to be

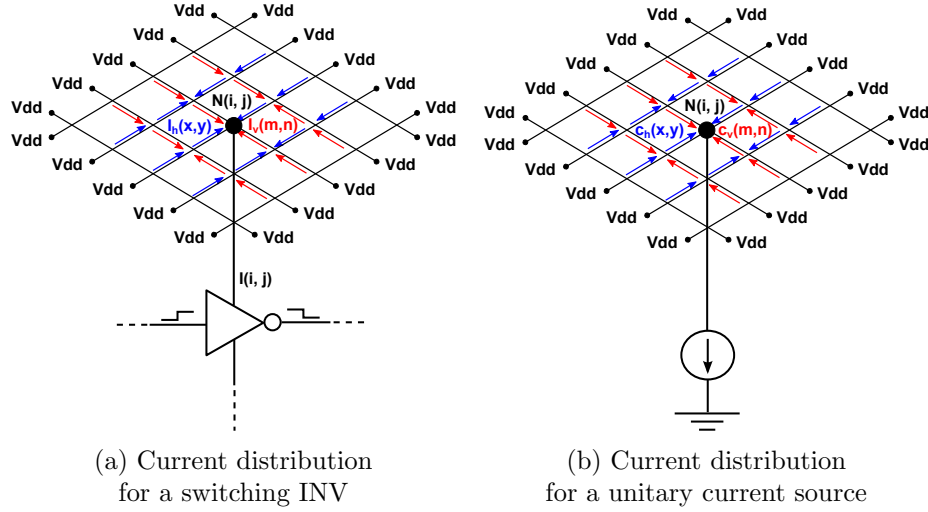


FIGURE 3.6: Current distribution in Vdd PDN

calculated. For the same configuration of PDNs, the current distributed in each resistor is proportional to a constant distribution factor, which varies for different resistors but is independent on the current draw. Based on this property, the distributing factors can be obtained by connecting a unitary current source to $N(i, j)$ and simulating the current distribution in SPICE. As illustrated in Figure 3.6b, the currents distributed in 4×3 horizontal resistances and 3×4 vertical resistances correspond to the desired distributing factors, denoted by $c_h(x, y)$ and $c_v(m, n)$, respectively. Then, given the estimated amount of current draw $I(i, j)$ due to the switching activity at supply node $N(i, j)$, the current distributed in the horizontal resistor $I_h(x, y)$ or in the vertical resistor $I_v(m, n)$ equals to $I(i, j)$ multiplied with the corresponding factor at the same position ($c_h(x, y)$ or $c_v(m, n)$).

In general, the distribution factor matrix, consisting of distribution factors in horizontal or vertical grid resistors, varies for the location where the instantaneous current appears in PDN. Since the goal of the pre-characterization is to approximate current distribution not only accurately but also efficiently, a slight error is allowed for the approximation, by using the same distribution matrix for instantaneous currents located nearby. The electrical model employed by the experiments achieves an error inferior to 1% in the approximation, by using dedicated distribution factor matrices for instantaneous currents in different PDN areas. The error is estimated by comparing the approximated current with the current given by SPICE simulation. The former is obtained by computing distributed currents in all horizontal and vertical resistors using the dedicated distribution factor matrix assuming that a unit of current draw appears at the concerned node. The

latter is obtained by performing a SPICE simulation that connects a unitary current source to the concerned node and reports the current distributed in all resistors. Meanwhile, the selected current window must be large enough to contain all current fractions higher than 1%. In other words, all distribution factors outside the current window are less than 1% and their impact on the voltage drops are disregarded by the simulation.

In experiments, three sets of distribution factors were used to approximate current distribution in central, corner and band areas in a 100×100 PDN, which is symbolically illustrated in Figure 3.7. The central area corresponds to a 40×40 sub-grid in the middle of the grid while each corner area corresponds to a 30×30 sub-grid in the corner; the rest of the grid consists of band areas.

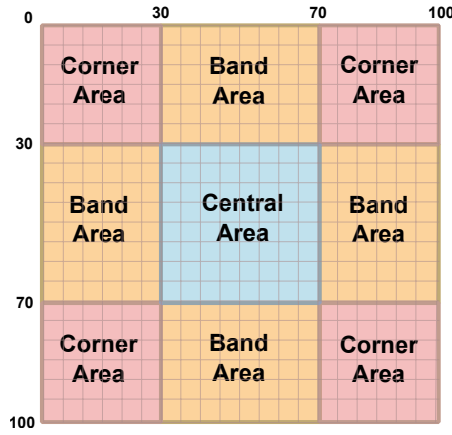


FIGURE 3.7: Central, corner and band areas in a 100×100 grid

The dedicated distribution factor matrix for the central area is obtained by SPICE simulation that connects the unitary current source at the middle node $N(50, 50)$ in PDN. For nodes outside the central area, the same distribution factor matrix for the central node is not applicable due to the edge effect caused by the proximity of the power supply at the grid edges. Dedicated factor matrices for nodes in corner and band areas are obtained by SPICE simulation in which a unitary current source is connected to nodes at coordinates $(15, 15)$ and $(15, 50)$, respectively. Note that the obtained matrices are appropriate for nodes in left-top corner area and top band area, however, nodes in other corner and band area can use transformed matrices due to the symmetrical structure of the grid. As given above, the approximated current distribution using the dedicated matrices in different PDN areas has an error less than 1%.

In context of the power supply at the chip level, supply grids with nominal supply voltages at the edges are used for the whole chip. However, it is impossible to simulate the

whole chip (using SPICE) for the pre-characterization because of the prohibitive simulation time. Only a sub-grid is therefore simulated where the power supply at the borders of the sub-grid can vary from the nominal voltage value due to the activities in neighboring blocks. Though the SPICE simulation for the whole chip is not feasible, an overall static effect resulting from the average activity of the neighboring blocks can be estimated and taken into account by the pre-characterization. The average power consumption of the neighboring blocks can be estimated by statistical approximations [52, 54]. In addition, commercial tools such as RedHawk [103] allow estimation of the power consumption using statistical analysis of the power supply noise. A pre-characterization of the sub-grid considers the effective voltage levels around the sub-grid, and gives the corresponding distribution factors. Figure 3.8 illustrates a high-level view of the power grid over a chip with 9 blocks. Nominal voltages are supplied surrounding the whole chip. Extreme nodes at the borders of BUT can have different voltage levels considering the static influence of the average activities in neighboring blocks. Assuming that the effective voltage levels around sub-grids over different blocks on the chip and the corresponding pre-characterizations are given, parallel simulation technologies can be considered to enhance the performance of MIRID at the chip level, by simulating sub-grids concurrently for different logic blocks.

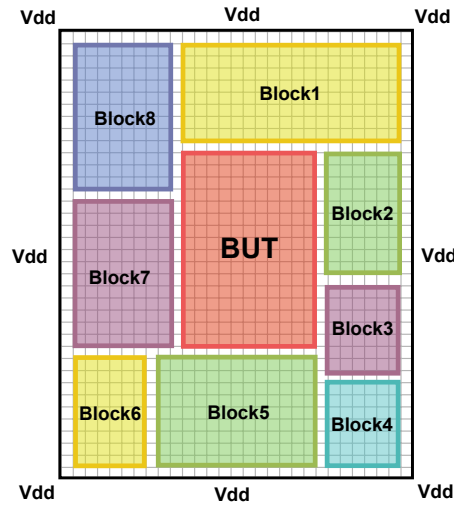


FIGURE 3.8: A high-level view of the power grid over a chip with 9 blocks

3.4.3 Electrical Models at the Gate Level

As given in Section 3.3, two electrical phenomena at the gate level are essential with respect to the IR drop: the current draw and the delay induced by the switching activities in the logic block. Both of them highly depend on the active simulation environment. For an accurate yet efficient simulation, electrical parameters that are most relevant to these two phenomena need to be derived and considered by establishing corresponding electrical models. Important parameters regarding a switching gate could be given as follows:

- **Edge:** rising or falling transition of the gate input;
- **Input voltage swing:** voltage swing of the upstream gate V_{swing1} defined in equation 3.1 where $Vdd_1(t)$ and $Gnd_1(t)$ are the power supply and ground levels of the upstream gate;

$$V_{swing1}(t) = Vdd_1(t) - Gnd_1(t) \quad (3.1)$$

- **Supply voltage swing:** voltage swing of the considered gate V_{swing2} defined in equation 3.2 where $Vdd_2(t)$ and $Gnd_2(t)$ are the power supply and ground levels of the considered gate;

$$V_{swing2}(t) = Vdd_2(t) - Gnd_2(t) \quad (3.2)$$

- **Load capacitance:** equivalent capacitance C_{load} of the downstream gates (fan-out) connected to the gate output.

Due to the complexity of deriving equations for modeling these two electrical phenomena, SPICE simulations were performed to pre-characterize the current draw and the gate delay for every standard gate, for a given technology and under all conditions likely to occur in practice; the electrical models are established based on the simulation results. For the experiments, the pre-characterization library is constructed

- for 45nm CMOS technology, and under conditions including

- all combinations of input and supply voltage swings varying from 80% to 100% of the nominal voltage swing,
- load capacitance varying from one to five times the elementary equivalent capacitance, and
- both (rising and falling) input transition edges for all standard gates.

Pre-characterizations of the current draw due to switching activity and the gate delay are introduced in [3.4.3.1](#) and [3.4.3.2](#), respectively.

3.4.3.1 Dynamic Current Model

Intrinsic currents due to electrical behavior of a logic gate are traditionally classified into two classes: static and dynamic currents. The former is the current that flows regardless of the switching activity. The latter appears when the gate output is switching. According to results from SPICE simulations for the pre-characterization, static currents are in the order of magnitude of 10^{-9} A, which is much smaller than the maximum current draw in the order of magnitude of 10^{-4} A when a gate is switching. Since the dynamic current mainly contributes to the IR drop effect, it is focused by the electrical model.

The dynamic current is created as a gate switches, and dissipates throughout the PDN grid as the time flows—it is a function of time. During the simulation, the current draw due to the switching activity is evaluated; the distribution of the current draw is computed using the distribution factors. Consequently, the current flowing through each resistor in PDN is updated by adding the distribution of the active current draw. It is necessary to store and update the current waveform in each resistor because the voltage drops are estimated based on the values of currents flowing through the resistive grid. Instead of representing the current waveform by a closed-form time-dependent function, an array of current values is used for the sake of simplicity in updating current values during a time interval. The n -th element in the array represents a discrete current value after n units of time. The time unit of the array corresponds to the precision of the simulation, i.e., the time step of the simulation. Using a higher precision raises the accuracy of the estimated voltage drops while the simulation cost grows enormously. In experiments, picosecond resolution is used by the simulation, and the current array has a size of 100 since all possible instantaneous currents in experiments will surely

settle down to zero within the time interval of 100 ps. As Vdd and Gnd PDNs can be considered independently, only amplitude values of currents are used though the currents from the two networks have opposite directions. In summary, the current waveform in each resistor of Vdd or Gnd PDN is stored as an array of 100 amplitude values with picosecond resolution.

SPICE simulations were run for all standard gates in NanGate 45nm Open Cell Library [68] with all possible input transitions that bring to the output switching. Constant voltage supply levels were applied to the gate to be characterized and the upstream gate; an inverter was used as the upstream gate. The output of the gate to be characterized was connected to a load capacitance representing the fan-out of the gate. Moreover, combinations of variable parameters that likely occur in a realistic environment (V_{swing1} and V_{swing2} between 80% and 100% of the nominal voltage swing, C_{load} between 1 and 5 times elementary equivalent capacitances) were taken into account. SPICE simulations were performed individually for each variable parameter varying in the range while the other two were kept to their nominal values (100% of nominal voltage swing for V_{swing1} and V_{swing2} , one elementary equivalent capacitance for load capacitance). Transient current draws from the power and ground supplies were reported as results.

The current model is established analytically based on the results from SPICE simulations. The current curves for a variable V_{swing2} (and the other two parameters with nominal values) can be derived from the reference curve, the one with all three parameters of nominal values, by reduction of the amplitudes based on the analytic results. Thus using a single multiplying factor is sufficient to derive the current curves from the reference one for different values of V_{swing2} within its realistic range. Compared to the reference curve, a similar reduction of the amplitude can be observed when V_{swing1} varies while the other two are kept to their nominal values. In addition, a shift in time presents in this case. Thus it is possible to approximate the impact of V_{swing1} on the reference current by applying shifting and multiplying operations. Corresponding factors for the shifting and multiplying operations are computed by **Multiple-Linear Regression (MLR)** given in Equations 3.3 and 3.4, respectively.

Both equations contain terms of V_{swing1} , V_{swing2} and the product of them, which reflects the multiplicative interaction between the two parameters.

$$sh = e_0 + e_1 \cdot V_{swing1} + e_2 \cdot V_{swing2} + e_3 \cdot V_{swing1} \cdot V_{swing2} \quad (3.3)$$

$$mul = f_0 + f_1 \cdot V_{swing1} + f_2 \cdot V_{swing2} + f_3 \cdot V_{swing1} \cdot V_{swing2} \quad (3.4)$$

The impact of the load capacitance seems to be more complicated and cannot be approximated using only shifting and multiplying factors. As only five values of the load capacitance (one to five times elementary equivalent capacitances) are considered, all five current arrays ($V_{swing1} = 100\%$, $V_{swing2} = 100\%$, $C_{load} =$ one to five times elementary equivalent capacitances) are used as reference currents. That means, for a given gate with a given switching input, and a value combination of the three variable parameters, a reference current curve is selected according to the load capacitance⁴, from which the dedicated current curve is derived by applying corresponding shifting and multiplying operations.

Consequently, the pre-characterization library contains sets of a reference current array and polynomial coefficients (e_0 – e_3 in Equation 3.3 and f_0 – f_3 in Equation 3.4) to compute the corresponding shifting and multiplying factors applied on the reference array. Each set refers to a standard gate, a switching input of the gate, the rising or falling transition edge, the load capacitance, the Vdd or Gnd PDN from where the current is drawn. That means, according to the active simulation environment, the current draws from both PDNs caused by a switching activity can be approximated by applying appropriate transformation factors to a dedicated reference array. The approximation is evaluated by the average **N**ormalized **R**oot **M**ean **S**quare **E**rror (**NRMSE**) between the approximated values and the current value from SPICE simulation in the same environment. Simulations performed in Montpellier [55] showed that in average the estimated error, given by the average NRMSE, is as low as 1.15%. Compared to the simple triangular function [50, 107] and the trapezoidal function [49, 108], this current model seems to be more accurate.

⁴The value of load capacitance corresponds to the number of downstream gates in this electrical model.

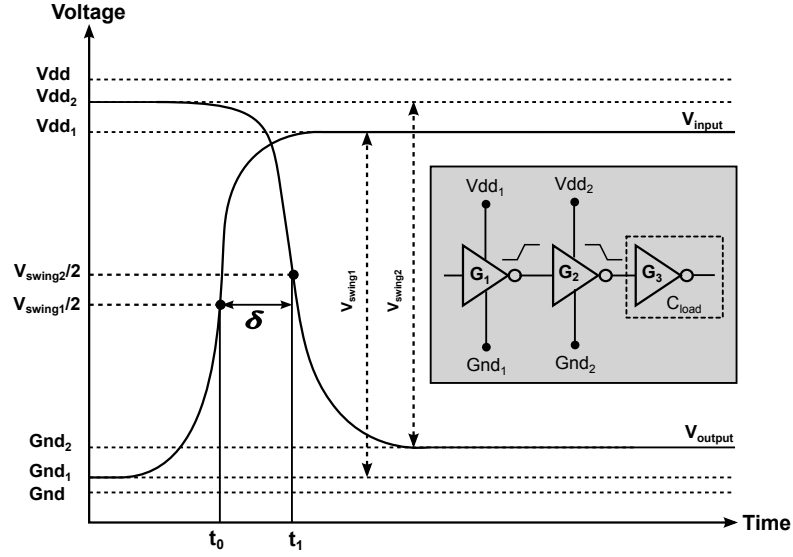


FIGURE 3.9: Supply and input voltage swings and gate delay

3.4.3.2 Gate Delay Model

The propagation delay of a gate is conventionally defined as the duration between the time when the input signal of the gate crosses half of its excursion (the change in voltage on the input in a short period, which also corresponds to the nominal voltage swing in an ideal case without IR drop), and the time when the output signal crosses half of its excursion. Concerning the IR drop effect, different supply and input swings need to be simulated by the pre-characterization to derive an accurate model of gate delay. Thus, as illustrated in Figure 3.9, for the characterized inverter G_2 with a rising input transition that is connected to an upstream gate G_1 and the load capacitance C_{load} representing the fan-out of G_2 , the propagation delay δ is defined as the duration between the time t_0 when the input voltage level V_{input} crosses half of its specific excursion ($V_{swing1}/2$) and the time t_1 when the output voltage level V_{output} crosses half of its specific excursion ($V_{swing2}/2$).

SPICE simulations were performed for all standard gates with all possible input transitions when the three variable parameter were set to the most possible values in a realistic environment (V_{swing1} and V_{swing2} between 80% and 100% of the nominal voltage swing, C_{load} between 1 and 5 times elementary equivalent capacitances). A linear approximation of the gate delay was derived individually for each parameter that varies when the other two hold to their nominal values. The simulation result shows that the gate delay increases almost linearly when V_{swing2} falls; the same yet slightly less linear dependence

of the gate delay on V_{swing1} is observed. The latter can also be linearly approximated with an average error inferior to 0.5%, which is quite acceptable. Though the relation between the delay and the load capacitance fits an exponential curve generally, in a small range from one to five times elementary equivalent capacitances a linear approximation of the delay can be derived.

As the three variable parameters are dependent, the effect of one parameter on the delay is affected by values of the other two. Thus, the gate delay is modeled by an MLR equation with multiplicative interactions containing terms of the three parameters and all possible products of the parameters, i.e., a polynomial of the first order for each parameter and for all combinations of the parameters. For the rising or falling edge on each input of a standard gate, the coefficients of an MLR equation is given by the model correspondingly. As shown in Equation 3.5, the delay function is defined by an MLR equation where coefficients c_0 – c_7 depend on the concerned gate, its switching input and the transition edge.

$$\begin{aligned} \delta = & c_0 + c_1 \cdot V_{swing1} + c_2 \cdot V_{swing2} + c_3 \cdot C_{load} \\ & + c_4 \cdot (V_{swing1} \cdot V_{swing2}) + c_5 \cdot (V_{swing1} \cdot C_{load}) + c_6 \cdot (V_{swing2} \cdot C_{load}) \\ & + c_7 \cdot (V_{swing1} \cdot V_{swing2} \cdot C_{load}) \end{aligned} \quad (3.5)$$

To evaluate the accuracy of the established delay model, the worst relative error and the average relative error are computed by comparing the gate delay approximated according to the model and the results from SPICE simulation. The average relative error is as low as 0.35% and appears to be satisfactory. The worst relative error was up to 5%, but it was observed only in corner cases when both input and voltage swings are very close to 80% of the nominal swing.

3.5 Simulator

As described in Section 3.3, MIRID consists of three parts, a logic simulation engine driven by switching events of gates, electrical models used to estimate the voltage drops

and the induced gate delays, and generic interfaces that takes active simulation environment as inputs and return values of electrical parameter according to the electrical models. Furthermore, a library containing objects of circuit elements needs to be established for an object-oriented implementation. Provided with the pre-characterization library containing look-up tables, electrical models are integrated into the simulation by building interface functions and converting the pre-characterization library into corresponding objects that can be accessed by the interface functions efficiently, e. g., to obtain the required electrical parameters values in constant time. The challenge of the implementation is to develop the logic simulation with flexibility of adapting to various electrical models. As introduced in Section 3.4, electrical models applied by the experiments still have the potential for raising the accuracy of estimated electrical effects, such as distribution of the current throughout the PDN grid where capacitive elements are also present. As the principle of the logic simulation does not depend on the applied electrical models, the part of the logic simulation and the part of the electrical models should be implemented individually, between which the interactions are realized by interface functions. Thus, the overall implementation can be easily adapted to the update of electrical models by only modifying the corresponding interface functions.

In the remainder of this section, 3.5.1 introduces the logic library containing objects of circuit elements used by both the logic simulation and the interface functions; pre-processing of the simulation is detailed in 3.5.2; the principle of the event-driven logic simulation is presented in 3.5.3; finally, 3.5.4 gives a brief introduction of interfaces functions that are adapted to the current version of electrical models and can be further extended.

3.5.1 Logic Library

The logic block under test is represented by a set of logic gates and interconnects, which can be converted into corresponding C++ (an object-oriented programming language) [109] objects. Pseudo codes that loosely follow C++ syntax are presented here to avoid bulky parts for the sake of readability.

A synthesized netlist in Verilog [110] is given for establishing the logic library of the circuit. Information on the gates and interconnects is extracted from the Verilog file and converted into C++ objects. The definition of objects for circuit elements must

represent the logic structure of the circuit. In addition, the object definition should include references to other related objects for an efficient logic simulation. For example, after evaluating the function of a gate G_0 , its downstream gates to be simulated next can be accessed in constant time through references to the corresponding object.

```
object gate{
    string      name;
    GateType_t  type;
    unsigned int id;
    unsigned int level;
    unsigned int fanin;
    set<pin&>    input_pins;
    pin&        output_pin;
};
```

FIGURE 3.10: Definition of the *gate* object

Three basic objects of *gate*, *signal* and *pin* are defined in the implementation. As given in Figure 3.10, the structure of a *gate* object includes seven members listed in the curly bracket. Each member is given by a pair of a data type and an identifier. The member *name* is the name of the corresponding gate instance defined in the Verilog file. Its data type *string* indicates that the member is saved as a sequence of characters. The member *type* identifies the type of the gate in a standard library of the used technology. Its data type *GateType_t* is not a built-in type in C++ but a user-defined enumeration of identifiers for standard gates used by the 45nm technology. The members *id*, *level* and *fanin* have the type of *unsigned int*, which is a positive integer. They indicate the unique identifier of the gate object, the level of the gate in the circuit and the number of its inputs, respectively. The member *input_pins* is a set of references of input pins given as *pin* objects. (The reference of an object is denoted by the ampersand (&) after the object type in the declaration of the object or before the object identifier when it is referred to.) The reference of the output pin of the gate is saved in the member *output_pin*.

The interconnects between gates are represented by a collection of *signal* objects. As given in Figure 3.11, the members *name*, *id* and *level* for the *signal* object are similarly defined as for the *gate* object. The *type* of a *signal* object is given by a named value from the enumeration *SignalType_t*, which corresponds to the classification of signals

```

object signal{
    string      name;
    SignalType_t type;
    unsigned int id;
    unsigned int level;
    unsigned int fanout;
    set<pin&>    input_pins;
    pin&         output_pin;
};

```

FIGURE 3.11: Definition of the *signal* object

based on their roles in the circuit, such as primary inputs, primary outputs and wires connecting two gates. The member *fanout* is the number of the gates fed by the signal, i.e., the fan-out of the upstream gate driving the signal. Moreover, references to input and output pins connected to the signal are saved in members *input_pins* and *output_pin*, respectively.

```

object pin{
    gate&         the_gate;
    signal&       the_signal;
    array<Time_t> delays;
};

```

FIGURE 3.12: Definition of the *pin* object

As given above, the *gate* and *signal* objects refer to a set of input pins and an output pin to which they are connected. Correspondingly, a *pin* object referring back to the gate and the signal is defined as shown in Figure 3.12. The members *the_gate* and *the_signal* refer to the corresponding gate and signal objects. In addition, since gate delays can vary for different inputs of each gate instance, delay information related to a single input pin are saved in the member *delays*. The member has the data type of an array of the user-defined type *Time_t* for possible maximum, minimum and nominal delays with presence of a rising or falling input transition; *Time_t* can be defined as an appropriate primitive number type according to the timing resolution used by the simulator. In the scenario of IR drop simulation, gate delays are computed during the simulation process according to the active environment. The nominal delays are used by the timing-aware simulation if IR drop is not taken into account. The simulation result using nominal

delays can be compared to the estimated IR drop induced delays, for evaluating the impact of the IR supply noise.

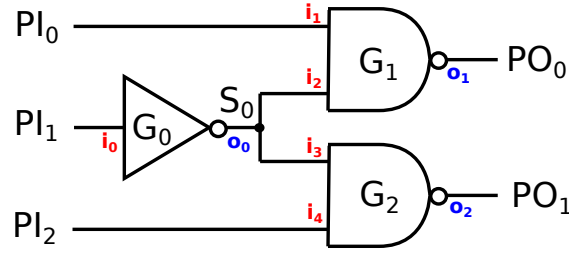


FIGURE 3.13: A circuit containing an inverter and two NAND gates

$G_0\{$ name G0; type INV; id 0; level 0; fanin 1; input_pins {&i ₀ }; output_pin &o ₀ ; };	$G_1\{$ name G1; type NAND2; id 1; level 1; fanin 2; input_pins {&i ₁ , &i ₂ }; output_pin &o ₁ ; };	$G_2\{$ name G2; type NAND2; id 2; level 1; fanin 2; input_pins {&i ₃ , &i ₄ }; output_pin &o ₂ ; };
--	--	--

FIGURE 3.14: Examples of constructed gate objects

Figure 3.13 presents a circuit containing an inverter G_0 and two NAND gates G_1 and G_2 . Examples of constructed gate objects for G_0 , G_1 and G_2 are presented in Figure 3.14. For instance, the *gate* object G_1 has the name of G1; its type is NAND2 indicating a NAND gate with two inputs; the member *id* is assigned with 1 which is its unique identifier; its level in the circuit is 1; *fanin* of 2 indicates that it has two inputs; the references of two *pin* objects i_1 and i_2 are saved in the member *input_pins* while the member *output_pin* refers to the output pin o_1 . Objects G_0 and G_2 are analogously constructed.

Corresponding *signal* objects are also constructed for primary inputs PI_0 , PI_1 and PI_2 , primary outputs PO_0 and PO_1 , and the wire connecting gates S_0 . Input pins i_0 – i_4 and output pins o_0 – o_2 are given as *pin* objects as well. Note that members for which no values are given or required by the simulation can be set to null. For instance, primary inputs are connected to input pins exclusively thus the member *output_pin* is null in corresponding objects. Examples of the *signal* objects PI_0 , S_0 and the *pin* object i_2 are presented in Figure 3.15. For object PI_0 , the member *type* is PI, a value from the

<pre> PI0{ string PIO; type PI; id 0; level 0; fanout 1; input_pins {&i1}; output_pin null; }; </pre>	<pre> S0{ string S0; type WIRE; id 3; level 1; fanout 2; input_pins {&i2, &i3}; output_pin &o0; }; </pre>	<pre> i2{ the_gate &G1; the_signal &S0; delays [1, 2, 3, 1, 2, 3]; }; </pre>
---	---	--

FIGURE 3.15: Examples of constructed signal and pin objects

enumeration *SignalType_t* indicating that the signal is a primary input. It is connected to the input pin i_1 of G_1 , hence its *fanout* is 1 and *input_pins* includes the reference of i_1 while the *output_pin* is null. Object S_0 has the type of WIRE indicating that it is an internal signal. The signal connects the upstream gate G_0 to downstream gates G_1 and G_2 individually. Correspondingly, its *fanout* is 2; the member *input_pins* contains the reference of i_2 and i_3 , which are the input pins connected by the signal; the member *output_pin* refers to o_0 , the output pin of G_0 . The *pin* object i_2 consists of the references to G_1 and S_0 , and a timing assignment given as a sequence of delays.

The logic library including the three types of objects can be established based on the netlist given in Verilog. Each *gate* object and each *signal* object is assigned with a unique identifier by a simple numeration. The levels of gates and signals are decided based on the circuit topology. A gate's output has the level $i + 1$ if the maximum level of the gate inputs is i . Assuming that primary inputs have the level of 0, then levels of all signals in the circuit can be computed iteratively. The level of the gate is defined to be the same as the maximum level of its inputs. Note that for a fan-out system with a stem and branches, such as S_0 is Figure 3.13, a common approach of levelization considers the stem and branches separately and assigns the level of branches with the level of the stem incremented by one. However, in the gate library established for the simulation, the fan-out system is considered as a single signal since no different effects will be assumed on the stem and branches in the scenario of IR drop simulation. As a result, no difference of level is assumed for the stem and branches in a fan-out. Finally, the *signal* and *gate* objects can be organized as lists sorted by the levels. The library also provides methods to access an object by its *id* within a constant time, which allows

a configuration for indexed objects that can be applied by the simulator efficiently.

3.5.2 Simulation Preprocessing

Electrical models are established based on a pre-characterization library containing look-up tables for current waveforms, distribution factors of current in PDNs and coefficients for computing gate delays, provided by the collaboration partner in Montpellier. These models have to be integrated into the implementation and compatible to the interfaces provided by the simulator. In preprocessing, the look-up tables are parsed and converted into C++ containers organizing data extracted from the look-up tables. These containers are categorized by combinations of different parameter values used by the pre-characterization, corresponding to different simulation environments of SPICE.

As given in Section 3.4.2, the current distribution factors in PDNs were obtained by SPICE simulations that connect a unitary current source in one of the three areas of the Vdd or Gnd PDN, the center, the left-top corner and the top band. Assuming that the current window is an $X \times X$ grid, then for each simulation, two matrices of the distribution factors were returned: a $(X - 1) \times X$ matrix for the horizontal resistors and an $X \times (X - 1)$ matrix for the vertical resistors. Since Vdd and Gnd PDNs are assumed to have identical structures, distribution of currents should have the same behavior in both PDNs. Consequently, in total six matrices of distribution factors in horizontal and vertical resistors for nodes in these three areas are saved in look-up tables. In preprocessing, the look-up tables are parsed and converted into two-dimensional arrays with corresponding sizes. Note that the matrices for nodes in corner and band areas are dedicated to the left-top corner and top band where the current draw appears. These matrices can be converted into matrices for nodes in other corner and band areas by mirroring rows and columns and transpositions according to various edge effects. For instance, assuming the current window is a 4×4 grid, the matrices of horizontal distribution factors for the nodes in left-top corner A_h and the distribution factors for nodes in the right-top corner obtained by mirroring columns A'_h are given as follows:

$$A_h = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \end{pmatrix} \quad A'_h = \begin{pmatrix} a_{1,4} & a_{1,3} & a_{1,2} & a_{1,1} \\ a_{2,4} & a_{2,3} & a_{2,2} & a_{2,1} \\ a_{3,4} & a_{3,3} & a_{3,2} & a_{3,1} \end{pmatrix}$$

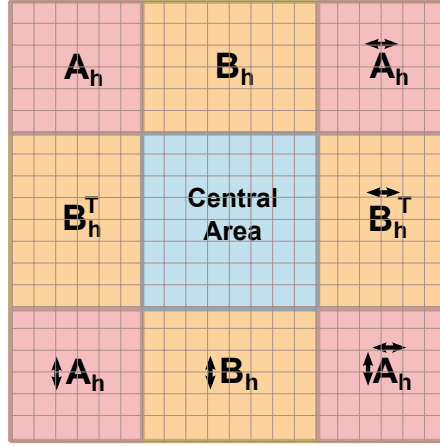


FIGURE 3.16: Symbolic presentation of operations for matrices in different areas in PDN

For nodes in other corner and band areas, operations that must be applied to obtain corresponding matrices are illustrated in Figure 3.16. Assume that A_h and B_h are $X \times Y$ distribution factors in horizontal resistors for nodes in the left-top corner and the top band, respectively. Then the matrix for nodes in the right-top area can be obtained by mirroring the column vectors in A_h , i.e., switching the first column and the Y -th column, the second column and the $(Y - 1)$ -th column, and so on. The operation is denoted by the \leftrightarrow symbol. The matrix for nodes in the left-bottom corner is obtained by mirroring rows, denoted by the \updownarrow symbol. Applying both operations to A_h results in the matrix for nodes in right-bottom corner. For nodes in the bottom band, operation of mirroring rows in B_h is applied to obtain the distribution factors in horizontal resistors. For nodes in left and right bands, the situation is somewhat different in that the same edge effect reflected by factors in B_h occurs in vertical resistors instead of horizontal resistors. Distribution factors in vertical resistors for nodes in left band corresponds to the transposed matrix B_h^T . For nodes in right band, mirroring columns in B_h^T results in the desired distribution factors in vertical resistors. Provided with distribution factors in vertical resistors for nodes in the left-top corner A_v and in the top band B_v , matrices for nodes in other corner or band areas can be obtained by applying appropriate operations analogously.

In preprocessing, provided with the six matrices by the pre-characterization, other matrices can be transformed as given above and saved in C++ containers. However, it may cost excessive memory in case of a large current window. Instead, functions mapping the

index in a transformed matrix to the index in the original are defined for the three transformation operations. Assuming the provided original matrix has the size of $X \times Y$, the mapping functions for mirroring rows, mirroring columns and the transpose are given in Equations 3.6, 3.7 and 3.8, respectively, where (x, y) are coordinates in the transformed matrix given as a vector. Then the required distribution factor can be obtained from the original at coordinates mapped using corresponding functions. For example, for nodes in right-bottom corner, the distribution factor at coordinates (x, y) corresponds to the value in the matrix for nodes in top-left corner at coordinates $(F_2(F_1(x, y)))$.

$$F_1((x, y)) = (X - x + 1, y) \quad (3.6)$$

$$F_2((x, y)) = (x, Y - y + 1) \quad (3.7)$$

$$F_3((x, y)) = (y, x) \quad (3.8)$$

For electrical models at the gate level, SPICE simulations were run for standard gates with different inputs. By analyzing the simulation results, information needed to model the current draw induced by a switching activity and the gate delay is extracted and saved in look-up tables, which are categorized by the following parameters:

1. standard gates such as buffer, inverter, NAND or NOR with inputs up to 4;
2. the index of the switching input;
3. the rising or falling input edge of the switching input.

As given in 3.4.3.2, the gate delay is modeled by a polynomial function of all combinations of three variables: the input voltage swing V_{swing1} , the supply voltage swing V_{swing2} and the load capacitance C_{load} (Equation 3.5). Each look-up table for the delay model contains eight polynomial coefficients corresponding to the delay function for a gate with a rising or falling input. During the course of preprocessing, sets of coefficients are saved in arrays with three dimensions of the categorizing parameters enumerated above.

For the current model, reference currents that obtained by SPICE simulations with nominal values of variable parameters are provided by the pre-characterization library. The shifting and multiplying factors are defined as well to derive approximated current waveforms from the reference currents for all combinations of parameter values that most possibly occur in a realistic environment. Due to the complexity of deriving approximated current waveform for different load capacitances, five current waveforms for each combination of the categorizing parameters have been determined by SPICE simulations for load capacitance varying from one to five elementary equivalent capacitances when the other two variables V_{swing1} and V_{swing2} are kept to the nominal value. Furthermore, currents drawn from Vdd and Gnd PDNs due to a switching activity are considered independently. Thus two extra parameters should be added to the enumeration above to categorize the corresponding look-up tables:

4. load capacitance varying from one to five elementary equivalent capacitances;
5. Vdd or Gnd PDN;

All look-up tables saving the reference currents, represented by 100 discrete current values for each, are converted into arrays with dimensions of the five categorizing parameters. Corresponding shifting and multiplying factors to derive currents for various values of V_{swing1} and V_{swing2} are taken from the pre-characterization library and saved in arrays with the same dimensions of the reference arrays.

3.5.3 Logic Simulation

As given in Section 3.3, MIRID combines a time-aware event-driven simulation performed at the gate level with the IR drop effect reflected by extra gate delays induced by the reduced voltage supplies. In other words, the approach modifies the conventional time-aware event-driven logic simulation by evaluating gate delays using provided electrical models. One of the challenges encountered by the implementation is to develop the core of the simulator that is most possibly independent from the electrical models, i.e., the simulator needs to be modified at least cost by updating of electrical models, which is likely to occur due to the requirement for raising the accuracy by integration of more electrical parameters. For this purpose, the simulator consists of two parts: the

simulation core that runs an event-driven simulation with modified timing configuration, and interface functions that return parameters required by the logic simulation, e. g., the gate delay, to the simulation core, and compute/update IR drop related electrical parameters according to the electrical models. In this section, the algorithm of the simulation core is introduced in detail.

Conventionally, an accurate logic simulation implies the evaluation of logic values at lines for applied test patterns, as well as the timing of each logic value transition. It also determines hazards, that is, the output of a logic system momentarily changes to a new value after the input changes, and settles back to the old value. For logic simulations that focus on the steady-state logic values implied at circuit lines, the temporal circuit behavior, including the timing of transition occurrences and the hazards, are often out of consideration. A zero-delay simulation is commonly performed in this scenario, by which the logic value at each gate output is evaluated by the gate function assuming that a line is assigned with the newly evaluated value without any delay. In this thesis work, as the target of the simulation is to evaluate IR drop induced delay due to instantaneous current requirement of switching activities, the timing analysis considering the temporal circuit behavior is necessary for the simulation. The simulation flow consists of two time frames for a two-pattern test. In the first time frame, the zero-delay simulation is performed since only the steady-state logic values at lines need to be evaluated. These values correspond to the initial state of lines in the timing-aware simulation performed in the next time frame.

In the remaining part of this section, algorithms for the zero-delay logic simulation (3.5.3.1) and the timing-aware simulation (3.5.3.2), which is a modified version of the zero-delay simulation, will be presented.

3.5.3.1 Zero-delay Logic Simulation

As shown in Algorithm 6, for a circuit C applied with a test pattern P , the zero-delay logic simulation assigns the test pattern to the primary inputs and evaluates logic values at other lines in order of their levels. The evaluation of the logic value at a line which is not a primary input considers two situations that the line is a gate output or it is a branch in a fan-out. In the former situation, the output value is evaluated by computing the logic function implemented by the gate. As lines are evaluated in order of level, it

is guaranteed that all gate inputs are already assigned by logic values when the gate output is evaluated according to the gate function. In the latter situation, the logic value of a branch is the same as the value of the stem.

Algorithm 6: Zero-delay Logic Simulation

Input: Circuit C , Test Pattern $P = \{p_1, p_2, \dots, p_n\}$

Preprocessing:

begin

 Levelize lines in the circuit;
 Save lines in the list L sorted by their levels;

begin

foreach primary input PI_i ($i \in \{1, 2, \dots, n\}$) **do**
 $PI_i \leftarrow p_i$;
 foreach line $l \in L$ **do**
 if l is not a primary input **then**
 Evaluate the logic value at l ;

As given in Section 3.5.1, a logic library is built to represent the circuit. It contains signal objects corresponding to lines in the circuit. The signals are saved in a list sorted by their levels, thus the implementation of the simulation simply evaluate logic values of signals in order of appearance in the list. In addition, a fan-out system is considered as a single signal. Thus a signal is either assigned with the corresponding logic value in the test pattern as a primary input, or evaluated by the computation of the related gate function as a gate output.

Commonly, the gate function can be given as a three-valued truth table, which describes the fault-free behavior of the gate. The three-valued logic contains components of logic 0, logic 1 and an unspecified value, usually denoted by symbol X called *don't care*. An example of the three-valued truth table for the NAND gate with two inputs i_1 and i_2 is given as follows:

		i_1		
		0	1	X
i_2	0	1	1	1
	1	1	0	X
	X	1	X	X

Truth tables for all standard gates are given as functions in the implementation; each takes the logic values of the input signals as inputs, and returns the logic value of the

output. The three-valued logic is defined by a named enumeration *LogicValue_t* that contains the values: *Logic_1*, *Logic_0* and *Logic_x*. Then the declaration of the function for a two-input gate can be given as follows:

```
LogicValue_t GateFunction(LogicValue_t i1, LogicValue_t i2){...};
```

For an n -input gate ($n > 2$), which can be considered as a hierarchical system of two-input gates, it is not necessary to save the truth table with a size of 3^n . Its function can be defined as an iterative call of the gate function of the corresponding two-input gate. For instance, the output of a three-input NAND gate can be evaluated by calling *FunctionNAND2(FunctionNAND2(i₁, i₂), i₃)* where i_1 , i_2 and i_3 are input signals and *FunctionNAND2* is the gate function of a two-input NAND gate.

3.5.3.2 Timing-aware Event-driven Simulation

For the purpose of simulating IR drop induced by switching activities in the logic block, two-pattern tests need to be applied to the circuit. For each two-test pattern, simulation is performed in two consecutive time frames. In the first time frame, the zero-delay logic simulation regardless of the timing behavior of circuit elements is performed. Each signal in the circuit is assigned with a steady logic value after the simulation. In the next time frame, the fault simulation commonly simulates the circuit with the faulty behavior modeled for physical defects. However, in context of simulating IR drop induced delays, no specific faulty behavior is given but the timing behavior affected by the IR drop has to be simulated.

The simulation is accelerated by employing the event-driven technology [2, p. 69]. That means, only the propagation of an input switching to the gate output will be considered as an event and processed during the simulation. The intrinsic static currents drawn by gates with only switching inputs or even without switching terminals have much less impact on the IR drop effect, and are ignored by the simulation for the sake of efficiency. Moreover, the situation that different inputs of the same gate switch simultaneously or successively during a small slice of time (typically smaller than the gate delay) is not taken into special consideration. That means, these input switchings are processed as independent events, which implies a simple combination of independently estimated

electrical values (for the current draw and for the gate delay) during processing these events. However, the actual resulted current draw and gate delay in this situation are not the same as the simple combinations. This impreciseness is caused by the lack of corresponding pre-characterization in electrical models. An exhaustive characterization for all possible combinations of (nearly) simultaneous input switchings is not feasible [111]. Fortunately, the probability of these simultaneous switchings is small and in addition their impact is limited [111].

The transitions of logic values at signals are described by switching events, denoted by $S\text{-event}(s, t, v)$, where s is the switching signal, t is the simulation time in picoseconds when the switching activity occurs and v is the logic value newly assigned to s . During the simulation, switching events are generated and inserted into a queue sorted by the time of events. Meanwhile, the simulation processes switching events in the sorted queue iteratively. An example of the event simulation applying an event queue is illustrated in Figure 3.17. Figure 3.17a presents three NAND gates G_1 , G_2 and G_3 , each line is assigned with a steady logic value by the zero-delay simulation for the first pattern. Assuming that a falling transition is imposed at the input signal a at time t_1 after applying the second pattern, a switching event $S\text{-event}(a, t_1, 0)$, denoted by e_1 , is generated and inserted into the event queue. The event queue is sorted by the time of events; the first event in the queue will be processed by the simulator. As shown in Figure 3.17b, assuming that e_1 is the first event, the event is processed by updating the active simulation time to t_1 and evaluating the gate function of G_1 . It implies a rising transition on the output c since the other input b has the non-controlling value of NAND. Moreover, using the provided interfaces to electrical models, the instantaneous current caused by the switching event is estimated and distributed throughout the supply networks; the gate delay of G_1 , denoted by d_1 , is estimated as well and returned to the logic simulation by the interface function. Accordingly, a new switching event $e_{n+1} : S\text{-event}(c, t_1 + d_1, 1)$ is generated, which indicates that the value of signal c is changed to 1 at the time $t_1 + d_1$. After the removal of e_1 and insertion of e_{n+1} , the event queue is newly sorted. Assuming that e_{n+1} has the earliest time in all events, it will be processed at the next. Analogously the electrical parameters are updated by the interface functions; delays of gates driven by c , d_2 and d_3 , are returned to the logic simulation. As the transition on c is propagated to outputs of the downstream gates f and g , two new switching events, $e_{n+2} : S\text{-event}(f, t_1 + d_1 + d_2, 0)$ and $e_{n+3} : S\text{-event}(g, t_1 + d_1 + d_3, 0)$, are generated

and inserted into the queue. The event queue is newly sorted and the first event will be processed in the next iteration.

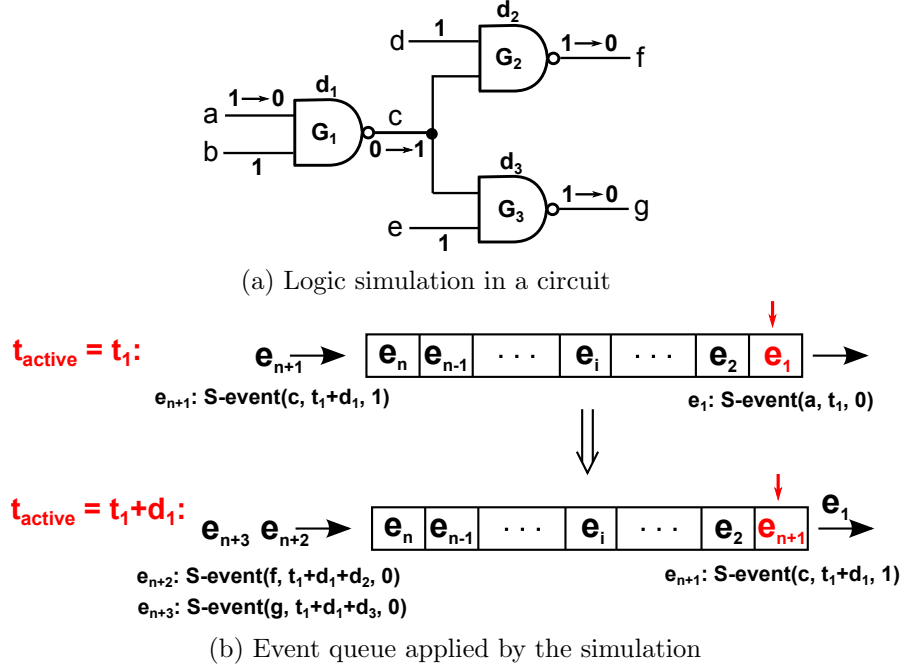


FIGURE 3.17: Event-driven simulation applying an event queue

The iterative simulation flow is presented in Algorithm 7. The simulator is run for a circuit C represented by a logic library as described in 3.5.1, and a sequence of two-pattern tests P . Each pattern in P contains n logic values where n is the number of primary inputs in combinational circuits, or the total number of primary inputs and flip-flops in sequential circuits. For sequential circuits, the values assigned at the state lines during the first time frame are saved in the flip-flops, and are used to initialize the corresponding lines in the next time frame, i.e., the test patterns are constructed by the values newly assigned to the primary inputs and the values saved in flip-flops during the previous simulation run. Besides that, no difference is assumed for the simulation algorithm regarding combinational circuits and sequential circuits.

The preprocessing of the algorithm consists of building a library of electrical models compatible to the interfaces functions (Section 3.5.2) and mapping each gate object in the logic library to Vdd and Gnd supply nodes. In practice, data from circuit layout should be used for determining parts of the PDN, to which a specific logic gate is connected. If the actual topological information is lacking, random connection between the gates and supply nodes can be assumed, or even very rare mappings can be considered for special

Algorithm 7: IR Drop Simulation**Input:** Circuit C , Patterns P **Preprocessing:****begin**

Build a library of electrical models compatible to the interfaces functions;

Assign each gate to Vdd and Gnd supply nodes;

begin **foreach** *two-pattern test* $(p_1, p_2) \in P$ **do** Run zero-delay simulation for p_1 ; **foreach** *primary input* PI_i ($1 \leq i \leq m$) **do** **if** $val(PI_i) \neq p_2[i]$ **then** Insert $S\text{-event}(PI_i, 0, p_2[i])$ into event queue EQ ; **while** EQ not empty **do** $S\text{-event}(s, t, v) \leftarrow$ first event in EQ ; $t_{sim} \leftarrow t$; $UpdateCurrent()$; **foreach** *gate* g *driven by* s **do** $s_{out} \leftarrow$ gate output of g ; $v_{old} \leftarrow$ logic value currently assigned to s_{out} ; Evaluate s_{out} according to the gate function of g ; $v_{new} \leftarrow$ logic value newly assigned to s_{out} ; **if** $v_{new} \neq v_{old}$ **then** $\delta \leftarrow GetGateDelay()$; Insert $S\text{-event}(s_{out}, t + \delta, v_{new})$ into EQ ; Remove $S\text{-event}(s, t, v)$ from EQ ; Sort EQ by the time of events;

investigation purposes, such as to provoke maximal IR drop by connecting all gates to a single supply node.

After the preprocessing, all two-pattern tests (p_1, p_2) in P are applied to the simulator iteratively. During each iteration, a zero-delay simulation is first performed for p_1 . All signals are assigned with logic values by evaluating gate functions during the simulation. Then the second pattern p_2 is applied; the event queue EQ is initialized by switching events of primary inputs, denoted by $S\text{-event}(PI_i, 0, p_2[i])$ indicating that the i -th primary input PI_i is assigned with $p_2[i]$ that is different to its currently assigned value $val(PI_i)$. The time of these events are initialized to an arbitrary value, e. g., the time zero. Events inserted in EQ are sorted by the time. No particular order is given for events with the same time in EQ .

As long as EQ is not empty, events in EQ will be processed iteratively in the sorted

order. During each iteration, the first event in EQ , denoted by $S\text{-event}(s, t, v)$, is to be considered. A global variable t_{sim} , which denotes the active simulation time, is updated by the event time t . The global time control facilitates synchronization of updating currents flowing throughout PDN resistors, realized by the interface function $UpdateCurrent()$. Note that the notation of functions in the algorithm does not contain any input argument for the sake of readability. Detailed declaration of interface functions including the input parameters will be given in 3.5.4. Electrical parameters affected by the switching event and mainly contributing to the IR drop effect are updated by the function. For example, the instantaneous current induced by the switching activity is evaluated and distributed throughout the PDN grids. After the estimation of electrical parameters according to active simulation environment, the logic simulation is performed further. For each gate g driven by s , the gate output s_{out} is evaluated by the corresponding gate function, as described in 3.5.3.1. If the newly evaluated value v_{new} differs from the current value of the gate output v_{old} , the gate delay δ is determined using the interface function $GetGateDelay()$ ⁵. Meanwhile, a new switching event $S\text{-event}(s_{out}, t + \delta, v_{new})$ indicating that the gate output s_{out} switches to the value v_{new} after the delay δ is generated and inserted into EQ .

After all downstream gates are considered, the processed event $S\text{-event}(s, t, v)$ is removed from EQ . If there exist newly inserted events, EQ is sorted by the time of events. A new iteration starts for the first encountered event in EQ . This iterative process continues until EQ becomes empty, i.e., all events in queue are processed and removed from EQ . No new events will be generated during the simulation when signal transitions cannot be propagated to the primary outputs or all events regarding switching primary outputs are processed.

For the test patterns generated for SDDs, the simulator propagates transitions along the sensitized paths through the possible defect sites. Delays of the propagation are compared to the propagation time under nominal delay assignment of gates. The extra delay given by the comparison indicates the impact of IR drop induced by applying these test patterns. Tests can be evaluated more accurately by the simulation when variations due to IR drop are considered. For example, tests generated by KLPG may not turn out as most effective after the evaluation. The tests that sensitize relatively shorter paths under nominal timing assignment but launch a large amount of switching activities and

⁵The input parameters of the function are omitted for the sake of readability

in turn induce longer delays in presence of IR drop should be used for SDD testing. Moreover, signal hazards due to different arrival times of input transitions are simulated by the time-aware simulation. For example, events $S\text{-event}(s, t, 1)$, $S\text{-event}(s, t + \delta_1, 0)$ and $S\text{-event}(s, t + \delta_1 + \delta_2, 1)$ represent the hazard $0 \rightarrow 1 \rightarrow 0$ at signal s . A signal hazard may invalidate nonrobust test for a given PDF. Furthermore, in presence of IR drop, a robust test generated under the nominal timing assignment may become invalid due to the extra induced delay. Thus, MIRID can also be used to evaluate pairs of patterns with respect to their testability. In context of power-aware test, IR drop has become an important concern with the trend of low-power high-performance design. The increased power consumption during test, e. g., during scan shift operation in scan based testing, can lead to a significant IR drop and result in timing failures. By accurately estimating the IR drop effect, it is possible for MIRID to evaluate various power-aware ATPG approaches.

3.5.4 Interface Functions For Electrical Models

As given in Algorithm 7, the interfaces functions, denoted by *UpdateCurrent* and *GetGateDelay*, are called by processing switching events. Actually, the implementation of *UpdateCurrent* includes two tasks: estimation of the instantaneous current drawn by a switching gate and distribution of the current throughout PDN grids. Corresponding functions *CurrentEstimation* and *CurrentDistribution* are presented in 3.5.4.1 and 3.5.4.2, respectively; the function *GetGateDelay* that determines gate delays is declared in 3.5.4.3.

3.5.4.1 Estimation of Dynamic Currents

The function *CurrentEstimation* approximates the dynamic current induced by a switching activity during the simulation process. It is defined as an interface function that takes active environmental parameters from the logic simulation, and estimates the induced current according to the electrical model. The declaration of the function is given as follows:

```

Array<Current_t> CurrentEstimation(GateType_t gate, Index_t idx,
    Edge_t edge, PDN_t pdn, Capacitance_t Cload,
    Voltage_t Vswing1, Voltage_t Vswing2)
{...};

```

The function returns an array of current values of the type *Current_t* corresponding to a floating-point number with user-defined precision. As given in 3.4.3.1, the array contains 100 distinct current values within the time interval of 100 picoseconds; the beginning of the interval refers to the time of switching event. The function is called by the logic simulation when the current drawn by a switching gate needs to be estimated. The input values are given according to the active simulation environment, based on which dedicated reference current array and operation factors are returned by the electrical model for the current approximation. According to the employed electrical model for dynamic current, the required input parameters are the type of the switching gate *gate*, the index of the switching input *idx*, the rising or falling input edge *edge*, the Vdd or Gnd PDN, the load capacitance *C_{load}*, the input voltage swing *V_{swing1}* and the supply voltage swing *V_{swing2}*. Each parameter has a user-defined data type specifying possible parameter values in a named enumeration, e. g., *PDN_t* corresponds to a named enumeration of $\{vdd, gnd\}$. For a modified electrical model, the implementation of the function needs to be modified correspondingly; more input parameters are possibly required.

As given above, the approximated current array depends on the input and supply voltage swings computed using equations 3.1 and 3.2. The computations require the voltage levels at supply nodes in Vdd and Gnd PDNs for both the upstream gate and the switching gate, which can be determined based on the active current distributions in PDN grids and the grid resistances.

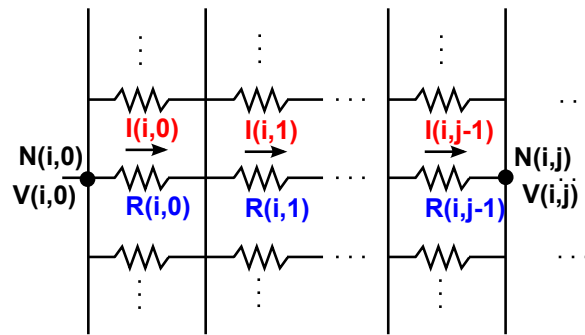


FIGURE 3.18: Calculation of voltage level at PDN node $N(i, j)$

Algorithm 8: CalculateVoltage**Input:** gate g , time t , current arrays in grid resistors**Output:** voltage $V(i, j)$ **begin** $N(i, j) \leftarrow$ supply node connected to g ; $V(i, 0) \leftarrow$ left border voltage of i -th line; $V(i, j) \leftarrow V(i, 0)$; **for** y from 0 to $j - 1$ **do** $R(i, y) \leftarrow$ grid resistance with coordinates (i, y) ; $I(i, y) \leftarrow$ current value in the corresponding resistor at time t ; $V(i, j) \leftarrow V(i, j) - I(i, y) \times R(i, y)$;

Algorithm 8 presents the computation of the voltage level supplied to gate g at time t , provided with current values distributed in grid resistors. Assume that g is connected to the supply node $N(i, j)$ in a PDN, which refers to the grid node at i -th line and j -th column in the PDN as illustrated in Figure 3.18. The voltage at $N(i, j)$, denoted by $V(i, j)$, can be calculated by applying the Ohm's law along the path from an extreme node on the border to $N(i, j)$. For example, the left extreme node of i -th line $N(i, 0)$ is supplied with a given voltage $V(i, 0)$. Then the voltage at $N(i, j)$ is computed by the equation: $V(i, j) = V(i, 0) - \sum_{y=0}^{j-1} (I(i, y) \times R(i, y))$, assuming that the horizontal resistances $R(i, 0), \dots, R(i, j - 1)$ and the active current values flowing through the resistors, denoted by $I(i, 0), \dots, I(i, j - 1)$, are known. In the electrical model applied by the experiments, the Vdd and Gnd PDNs are supplied with nominal power and ground voltage values at the borders, respectively. All horizontal and vertical grids have the same resistance 0.4Ω . Given the currents flowing through resistors, the voltage levels on supplied nodes connected to the considered gate in both PDNs can be calculated using Algorithm 8. Note that the algorithm presents only a possible way of the computation. Actually the Ohm's law can be applied along a path from any extreme node to the supply node of g to get the voltage level, which will always lead to the same result.

3.5.4.2 Distribution of Currents in PDNs

After the estimation of the instantaneous current drawn by the active switching event, currents flowing through neighboring resistors are updated by adding distributions of the current. The declaration of the corresponding function is given as follows:


```
void CurrentDistribution(Node_t n, Array<Current_t> current)
{...};
```

It takes the supply node connected to the switching gate and the current array returned by *CurrentEstimation* as inputs. No returned value is required by the logic simulation through the interface, which is indicated by the *void* before the function name. The supply node is defined as an object *Node_t* that contains information on the coordinates of the node in PDN grid, the connected gate, etc.

Algorithm 9: CurrentDistribution

Input: PDN node $N(i, j)$, Current array *current* appears at $N(i, j)$

begin

```

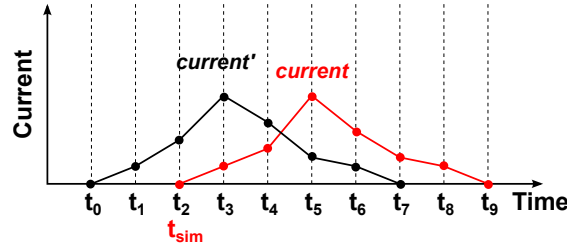
 $F_{X \times Y} \leftarrow X \times Y$  horizontal distribution factors for  $N(i, j)$ ;
 $R_{X \times Y} \leftarrow X \times Y$  neighboring horizontal resistors of  $N(i, j)$ ;
for  $x = 1 \rightarrow X$  do
    for  $y = 1 \rightarrow Y$  do
         $current' \leftarrow$  saved current array in resistor  $R(x, y)$ ;
        Update discrete current values in  $current'$  using current and  $F(x, y)$ 
        (Equation 3.9);
```

As given in 3.5.2, two sets of factors regarding the current distribution in horizontal and vertical resistors are given for nodes in different areas. Actually, it is not necessary to update the current distributed in both the horizontal and vertical resistors. The reason is that the voltage level at a supply node can be computed by applying the Ohm's law along a horizontal or vertical path, as given in 3.5.4.1. Consequently, only the currents in either horizontal or vertical resistors are used by the computation and thus need to be updated. Without loss of generality, Algorithm 9 presents the update of currents saved in horizontal resistors in the neighboring area of $N(i, j)$. The current window is assumed to have a size of $X \times Y$. The horizontal resistors in the neighboring area of $N(i, j)$, denoted by $R_{X \times Y}$, is defined to have the same size of the current window. As given in 3.4.3.1, the pre-characterization library saves matrices of current distributing factors in the sub-grid of PDN with the same size of the current window. According to the position of $N(i, j)$ in PDN, an appropriate matrix of current distribution factors in horizontal resistors is selected, which is denoted by $F_{X \times Y}$. Each resistor $R(x, y)$ refers to a distributing factor $F(x, y)$ ($1 \leq x \leq X$, $1 \leq y \leq Y$). The current array flowing through $R(x, y)$, denoted by $current'$, is updated by adding the distribution of

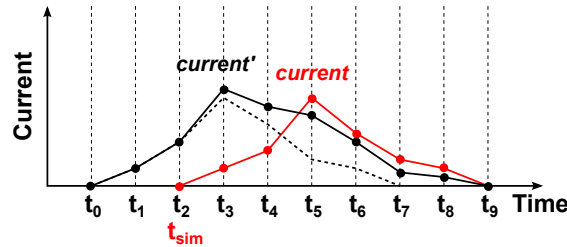
the newly induced current, i.e., discrete values in *current* multiplied with $F(x, y)$, to corresponding values in *current'*. Assuming that the first value in *current* corresponds to the active simulation execution time t_{sim} , then only values after t_{sim} in *current'* need to be updated, as given in Equation 3.9.

$$current'[m] = current'[m] + current[m] \times F(x, y) \quad (m \geq t_{sim}) \quad (3.9)$$

Figure 3.19 presents an example of updating the values in a current array *current'*, given as eight discrete values in the time interval t_0 – t_7 in Figure 3.19a, using the estimated current array *current* and a distribution factor. As *current* appears due to the active switching event, the first value in the array corresponds to the active simulation time t_{sim} . Assuming that t_{sim} has the value of t_2 , then only the values in *current'* after t_2 need to be updated. As shown in Figure 3.19b, the updated *current'* contains ten discrete values. Values at t_0 and t_1 are not changed while values for t_2 – t_9 are incremented by the corresponding distributions of *current*.



(a) Current array before the update



(b) Current array after the update

FIGURE 3.19: Update current array *current'*

3.5.4.3 Estimation of Gate Delays

During the simulation gate delays are estimated by the function *GetGateDelay*, which is declared as follows:

```
Time_t GetGateDelay(GateType_t gate, Index_t idx, Edge_t edge,
    Capacitance_t Cload, Voltage_t Vswing1, Voltage_t Vswing2)
{...};
```

As given in 3.4.3.2, the gate delay is computed by the polynomial (Equation 3.5) of the input voltage V_{swing1} , the supply voltage V_{swing2} , the load capacitance C_{load} and all possible combinations between them. Hence, the active values of the three parameters must be given as function inputs when the function is called by the logic simulation. Moreover, the function requires the information on the target gate $gate$, the index of its switching input idx and the edge of the input transition $edge$ to get dedicated coefficients of the polynomial.

3.6 Application to Self-adaptive Better-than-worst-case Design

As the core of the simulator is a time-aware logic simulation developed independently on the electrical model, modifications adapted to various research purposes can be easily integrated into the core without affecting the original design. In this section, the application of the simulator to self-adaptive better-than-worst-case design will be introduced. Better-than-worst design [5, 32, 34] is a popular approach addressing the large variability in state-of-the-art technology. The aim of the design is to overcome worst-case clock timing requirements by employing a shorter clock period and allowing occasional errors to occur. Low-cost error detection and correction techniques are often used to detect and correct these errors. In contrast to the traditional, conservative design style, better-than-worst-case circuits are run at an aggressive voltage-frequency operating point, which improves the circuit performance but does not guarantee the correct timing behavior of the circuit; the reliability of operations is achieved by detection and correction of errors, and the improvement in performance and/or energy consumption can outweigh the overhead of error correction. Thus, better-than-worst-case architectures are often self-adaptive: they may keep raising the frequency or lowering the voltage if the error rate is under a limit; stop the process if the limit is exceeded.

To detect timing-related errors in self-adaptive better-than-worst-case circuits, low-cost solutions are well-studied. One extensively used technique [5, 34] equips flip-flops with duplicated shadow memory elements triggered by a delayed clock. The delay is chosen such that the extended clock window ensures the timing even in worst case, i.e., the propagation delay through the longest path. Hence, the timing errors can be detected by comparing the values in the main and the shadow memory elements. Such an error detection approach has been proposed for a better-than-worst-case design called *Razor* [5], and also for several experimental prototype designs in the industry [112, 113]. However, this scheme does not work reliably when the flip-flop is driven by a path that is shorter than the timing skew. Two solutions are known to mitigate the short-path problem: padding the short paths with buffers until they become sufficiently long, or placing additional latches on to the short paths. Aiming at reliable self-adaptive circuit operations, possible invalidation scenarios that could prevent error detection in Razor-like designs are investigated in [114]. The study also derives conditions to avoid such invalidations and performs theoretical analysis of both solutions for the short-path problem based on a simple model. A modified version of MIRID performs simulations for benchmark circuits with buffers or latches placed according to the derived conditions.

The remaining text of this section is organized as follows: the short-path invalidations and mitigation techniques by inserting buffers and latches are first reviewed in 3.6.1; a brief introduction to the detection conditions for timing errors based on a simplified model is given in 3.6.2; finally, the modified simulation flow of MIRID for mitigating the short-path problem is presented in 3.6.3.

3.6.1 Short-path Invalidation

Razor memory elements consist of a main latch and a second “shadow” latch that is driven by the same input but clocked with a certain delay δ after the main latch. The value δ is chosen to ensure that the correct value will always be captured during the delayed clock window while the incorrect value due to a delay fault is captured by the main latch. The delay fault is detected in cases that values saved in both latches are different. However, for short paths driving such a memory element, a *false positive* detection is possible when the transition propagates too fast through the short path.

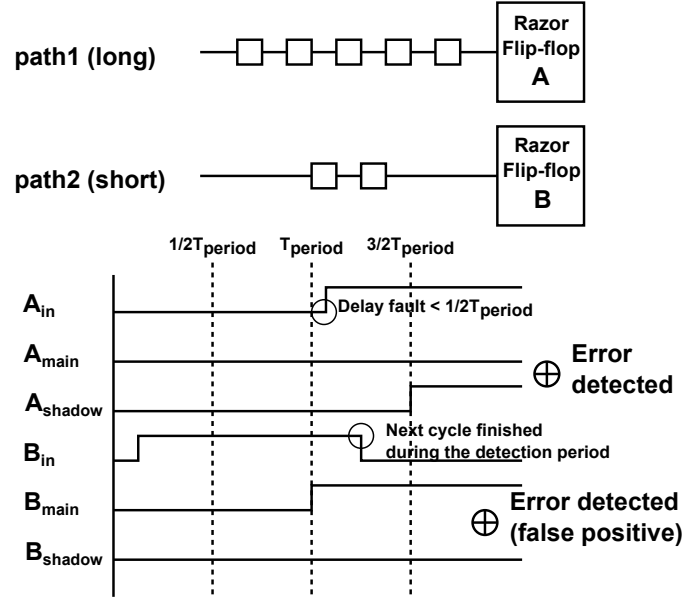


FIGURE 3.20: Short-path invalidation in Razor design

Figure 3.20 illustrates an example of the short-path invalidation. Razor memory elements A and B are driven by a long path (path1) and a short path (path2), respectively. Given the clock period T_{period} of the main latch, the shadow latch is clocked with a delay of $1/2 T_{period}$. The timing diagram represents values of the latch inputs A_{in} and B_{in} , the value in the main latches A_{main} and B_{main} , and the values in the shadow latches A_{shadow} and B_{shadow} of A and B , respectively. Assuming that the long path propagates the transition with a delay shorter than $1/2 T_{period}$, the main latch captures the wrong value 0 at the clock edge T_{period} whereas the shadow captures the correct value 1 at the clock edge $3/2 T_{period}$, as illustrated in the timing diagram. Comparing these two values using a XOR will flag an error and trigger a possible recovery, e. g., re-execution of an instruction in a microprocessor. However, the error detection is invalidated when the propagation time is shorter than $1/2 T_{period}$ along the short path. In this case, the transition of B_{in} in the next cycle is finished within the detection period between one and $3/2 T_{period}$. The main latch captures the correct value 1 whereas the shadow latch captures the value 0, which is the correct value of B_{in} after the transition in the next cycle. An error is reported because of the difference of the two values. The detection is false positive since no delay fault is actually present. The incorrect detection can cause a repeated execution for the recovery, which leads to the same error again and results in a deadlock.

Two ways are considered to alleviate this short-path invalidation: adding buffers to

a short path such as to guarantee a minimal delay of $1/2 T_{period}$ [115], or placing a latch onto the short path [116]. For the latter, the inserted latch is opaque in the first half and transparent in the second half of the clock cycle. That means, the latch delays the transition on the path beyond $1/2 T_{period}$ and thus excludes the possibility of the invalidation. An example of the latch placement mitigating the short-path problem is illustrated in Figure 3.21. A latch L is inserted into the short path so that the input transition of the latch input is delayed beyond the opaque period of the latch in both clock periods. Consequently, both the main and the shadow latches in the razor memory element B captures the correct value; no error will be false-positively detected.

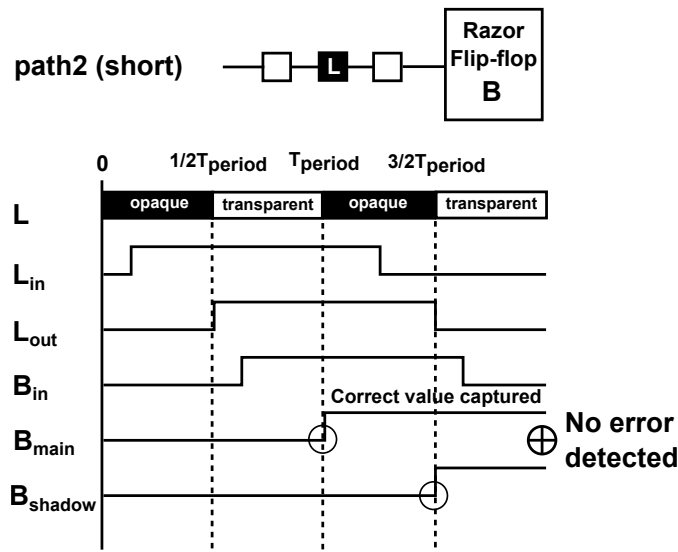


FIGURE 3.21: Mitigation by latch placement

However, both mitigation techniques, buffer insertion and latch placement, are prone to invalidations under extreme variations. A path that has been padded by buffers to guarantee delay of $1/2T_{period}$ under maximal variation-induced speed-up may actually become potentially too long under maximal variation-induced slow-down. In the case of latch-based mitigation, sub-paths from a circuit input to the placed latch or from the latch to a circuit output may have delays that exceed T_{period} under maximal variation-induced slow-down. If such a path is sensitized, the detection scheme will not work and in turn worsen the robustness of the system operation. In the following section, all these invalidations due to variations are investigated and analyzed based on a simple model; closed-form conditions for a reliable detection are derived from the analysis.

3.6.2 Detection Conditions

The purpose of the analysis in this section is to identify conditions for reliable detection of errors in self-adaptive designs.⁶ The analysis yields two clock periods durations: the *safe clock period* T_{safe} for which all paths are guaranteed to complete switching and no error can occur even under maximal slow-down; and the *minimal clock period* T_{min} at which errors can occur but are guaranteed to be detected under all variations (between maximal slow-down and maximal speed-up). Operating the circuit at a frequency higher than $1/T_{min}$ may lead to unrecognized errors and thereby result in erroneous operations. An adaptive frequency scaling strategy may choose a clock period $T \in [T_{min}, T_{safe}]$ and adaptively increase (decrease) it if the number of detected errors is over (under) a user-specified threshold.

The statistical model is typically used by analysis of timing under variations. For example, one could calculate the probability of a detected error and the overhead of correction, and determine the clock period that lead to the minimal expected execution time. However, the requirements on the detection mechanism are much stricter, as even one undetected error may lead to a deadlock or erroneous operation. For this reason, the analysis focuses on identification of corner cases, even though their likelihood of occurrence might be very low.

3.6.2.1 Simplified Variability Model

The analysis is performed based on a simplified model avoiding details that can be easily incorporated into the model but would make the analysis bulky. The model assumes the unit-delay timing of gates under maximal speed-up. Two sources of variations are considered: process-related variability p and pattern-dependent variability d . Process-related variability incorporates factors such as line-edge roughness, grain alignment in the gate, and atomic-scale dopant fluctuations. Parameter p may already assume values of 1.4 ($\pm 20\%$) in today's technologies displaying moderate variability, and can be expected to assume values of 2 ($2X$) or more in future technologies, particularly in low power circuits operated at near-threshold or subthreshold supply voltages. If only process-related

⁶It is assumed that designs have no distinct voltage or power domains. The techniques discussed here are applicable on individual domains of more complex designs.

variability is considered, a given gate may assume the delay between 1 and p , and a path of m gates may have a delay between m and mp time units for extreme corner cases.

Pattern-dependent variability refers to the influence of the patterns applied on the inputs of a switching gate on the delay. For example, the NAND3 gate may have three times larger delay for the $111 \rightarrow 011$ at the inputs compared with the $111 \rightarrow 000$, though both input transitions induce the same $0 \rightarrow 1$ at the output. It is possible to construct an entire path composed of alternating NAND and NOR gates such that all gates have minimal or maximal delay at the same time depending on their side-inputs. The simplified model does not consider the side input explicitly and assumes that a gate affected by pattern-dependent variability has a delay between 1 and d . Note that d can be rather large even in conventional technologies with low process variability. Then, if both sources of variations are present at the same time, the delay of a gate varies between 1 and n where $n := pd$ is the maximal cumulative variability.

3.6.2.2 Mitigation by Buffer Padding

As given in 3.6.1, a minimal delay of $1/2 T_{period}$ needs to be guaranteed for every path when mitigating the short-path problem by buffer padding. Assuming that each gate has the maximal delay n and the structurally longest path has m gates, the safe clock period for which no errors can occur even under maximal slow-down is $T_{safe} = mn$. Then, the reliable error-detection is ensured for a raised clock frequency (or, equivalently, a reduced clock period) if all paths have the minimal delay $1/2 T_{safe} = 1/2 mn$. The condition $n \leq 2$ is derived in [114] for buffer padding applied for all short paths with less than $m/2$ gates without violating $T_{safe} = mn$.

Moreover, it is assumed that buffers used for padding are prone to variability but possibly to a lesser extent than regular gates, which means a buffer delay between 1 and b , where $b \leq n$. The reason is that a buffer has one input and is not affected by pattern-dependent variability. (Variability-control techniques can also be employed to reduce its maximal slow-down.) Then under variation $n > 2$, it is proved by inductive analysis ([114]) that the buffer padding is impossible if $b > n/(n - 1)$. However, some paths may not be padded by buffers even if the buffers exhibit no variability at all ($b = 1$). Such an invalidation for $b = 1$ can be addressed by extending T_{safe} . Details of the performed analysis are given in [114].

The minimal clock period T_{min} is a relaxed clock period ($T_{min} < T_{safe}$) in which the circuit is not guaranteed to finish switching but any errors are detected. To determine the value of T_{min} for which buffer padding is possible, first it must be ensured that the circuit still meet the clock period T_{safe} under the maximal slow-down after buffers have been inserted. Then, to be detected, errors need to be visible up to half clock cycle after the end of the clock period, otherwise the shadow latch will not catch the correct value. That implies all paths will finish switching before $3/2 T_{period}$. As it is known that all paths will finish switching before T_{safe} , the condition is fulfilled for T_{period} satisfying $3/2 T_{period} \geq T_{safe}$. Therefore, $T_{min} = 2/3 T_{safe}$.

In summary, buffer padding is impossible if the variability n of gates exceeds $2\times$ and the variability b of the inserted buffers exceeds $n/(n - 1)$. It requires a larger T_{period} for certain path lengths even if the buffers are not affected by the variability at all. If padding is possible, frequency scaling is reliable between $[T_{min}, T_{safe}] = [2/3 mn, mn]$.

3.6.2.3 Mitigation by Latch Placement

Latch insertion can give rise to three invalidation scenarios: input-to-latch invalidation, strong latch-to-output invalidation and weak latch-to-output invalidation. They are discussed next based on the same model used for buffer padding.

Figure 3.22 presents the *input-to-latch invalidation* scenario: a latch L is inserted into a path where the length of sub-path from the input to L exceeds T_{period} under maximal slow-down. As the latch captures the wrong value of the unfinished transition and delays it beyond the operating period of the error-detection circuitry, both the main and the shadow latches capture the same value; the error is not detected. The reason why a latch is inserted on such a long path may be the existence of a short path to which the latch belongs. Thus, the algorithm that places latches on short paths must avoid locations which also belong to long paths and could cause the input-to-latch invalidation.

As shown in Figure 3.23, the *strong latch-to-output invalidation* occurs if a sub-path between the latch and an output has a delay of more than T_{period} under the maximal slow-down. The cumulative delay of the path exceeds $3/2 T_{period}$, and both the main and the shadow latches capture the wrong value, such that no error is detected.

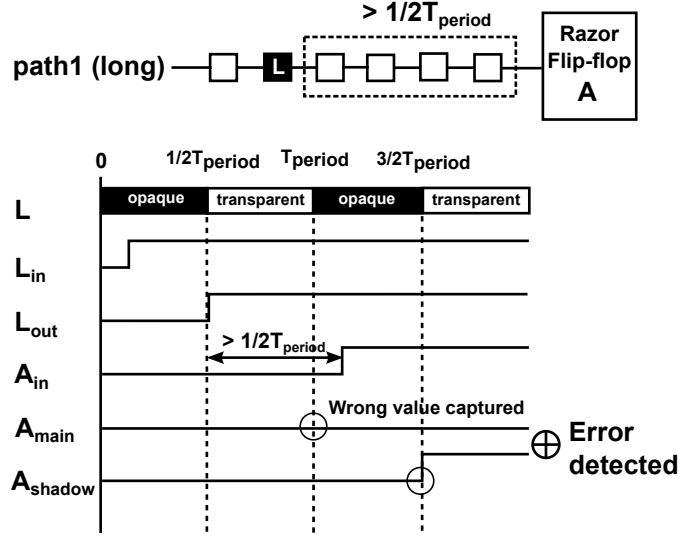


FIGURE 3.24: Weak latch-to-output invalidation

on the structural longest path and n is the maximal gate delay, is investigated. Unlike buffer padding, by which this choice is problematic under certain circumstances, placing a latch anywhere in a circuit will not yield any sub-path with more than m gates. That means, no sub-path can have a delay exceeding T_{safe} and thus the strong invalidations are not possible in this case. Error-free operation is guaranteed if latches are placed such that their maximal structural distance to an output does not exceed $m/2$ gates⁷.

It is possible to place latches such that the detection of all variation-induced errors is guaranteed for the clock period $mn/2$, i.e., $T_{min} = mn/2$. Undetected errors do not occur for this clock period if the distance of all placed latches from the inputs and to the output is less than or equal to $m/2$ gates; the strong invalidations are in turn avoided. Furthermore, it takes advantages to place the latch as close to the end of the path as possible without risking a strong invalidation (that is, at position $m/2$). The reason is that placing a latch far from the end of the path raises the probability of a weak latch-to-output invalidation: the transition is delayed at the latch until time $T_{min}/2$; the sub-path to the output takes longer than $T_{min}/2$ to switch; and the total switching time exceeds the clock period which results in a detected error, though the total switching time of gates on the path could be less than T_{min} .

In summary, reliable mitigation of the short-path problem by latches under extreme variability is always possible. Error-free operations avoiding all three described invalidations

⁷Under this condition, the transition of the sub-path from an input to the latch is delayed till $mn/2$ and the sub-path from the latch to an output cannot exceed $mn/2$. As a result, the total switching time of the path is less than the clock period mn .

is achieved at clock period $T_{safe} = mn$, whereas reliable operation with guaranteed error detection is possible at $T_{min} = 1/2 mn$. It gives the reliable frequency scaling in range $[1/2 mn, mn]$, which is larger than its counterpart $[2/3 mn, mn]$ for mitigation by buffer padding.

3.6.3 Strategies of Mitigation

Based on the application conditions derived above for buffer padding and latch placement, strategies of the mitigation are considered for the two techniques individually. For buffer padding, the following procedure can be applied to insert buffers into short paths while avoiding unnecessary delay increase of the long paths. Given a circuit C and variabilities n and b , let S be the set of short paths that can include either all short paths or only the sensitizable ones. First the condition $n > 2$ and $b > n/(n - 1)$ is checked; if it applies, padding is impossible as explained in 3.6.2.2. Then, if the condition does not apply, for each gate g the slack is calculated, which is defined as $slack(g) := m - (\text{number of gates on the structurally longest path through } g)$.

As long as there are still short paths in S , a gate g_{max} with $slack(g_{max}) > 0$ that is on a maximum number of short paths in S is determined. A buffer is inserted at the output of g_{max} (at the fanout branches if the gate fans out). The short-path condition (the number of gates on the path is less than $1/2 T_{period}$) is evaluated for the paths from S going through g_{max} , and if it no longer holds, the paths are removed from S . Moreover, the slacks of gates in the input- and output-cones of g_{max} are updated. The procedure terminates as soon as S becomes empty. The identified buffer placement guarantees reliable operation for $T_{min} = 2/3 mn$.

For mitigation by latch placement, the strategy is to place latches within every short paths at the (unique) location with maximal structural distance $m/2$ from any input (or at the end of a path if its distance from any input is less than $m/2$). This placement eliminates the strong invalidations and minimizes the likelihood of weak invalidation as explained in 3.6.2.3. Detailed experimental results of the both mitigation flows for actual circuits are presented in 3.7.4.

3.7 Experimental Results

Applying the electrical models introduced in 3.4, experiments were performed with various targets. The accuracy of the applied electrical models are validated by experiments as given in 3.7.1; experiments presented in 3.7.2 aim at investigating the impact of switching activity on IR drop effect; 3.7.3 shows the performance of MIRID in experiments for large circuits; finally, solutions for mitigating short-path problem in Razor-like self-adaptive designs are reported in 3.7.4.

3.7.1 Accuracy Validation

The first experiment aims at validating the accuracy of the electrical models by comparing the results reported by MIRID to results of SPICE simulation performed for the same circuit. The results of SPICE simulations are provided by the collaboration partner from LIRMM. As it is almost infeasible for SPICE to simulate large circuits due to the excessive runtime, a simple schematic is used by the validation. As shown in Figure 3.25, the schematic includes a chain of 14 inverters, each inverter is connected to supply nodes in 100×100 resistive PDN grid. For the sake of simplicity, only two lines in a PDN where the connected nodes lie and the inverter chain are drawn in the Figure. (The connection to PDN nodes is marked with dashed line while the connection between inverters is marked with straight line.) The supply nodes are denoted by $N(i, j)$ ($i \in \{40, 60\}, j = \{47, \dots, 53\}$) and the resistors on these two horizontal lines are denoted by $R(i, j)$ ($i \in \{40, 60\}, j = \{46, \dots, 53\}$). The inverters are connections to supply nodes in the other PDN in exactly the same way. A two-pattern test (1, 0) is applied to the single primary input in both simulation flows. Note that half of the inverters that are odd-numbered with a falling transition under the test are connected to nodes on line 60. The other half of the inverters with a rising input transition are connected to nodes on line 40.

As given in 3.4.3.1, additional currents will be drawn from the Vdd and Gnd PDNs by a switching activity. Depending on the transition edge, one of the two current draws is predominant on the other. The main benefit of using this schematic is that all inverters connected to the same line have the same input transition and thus all their predominant current draws appear in the same PDN. The IR drop effect caused by the switching of

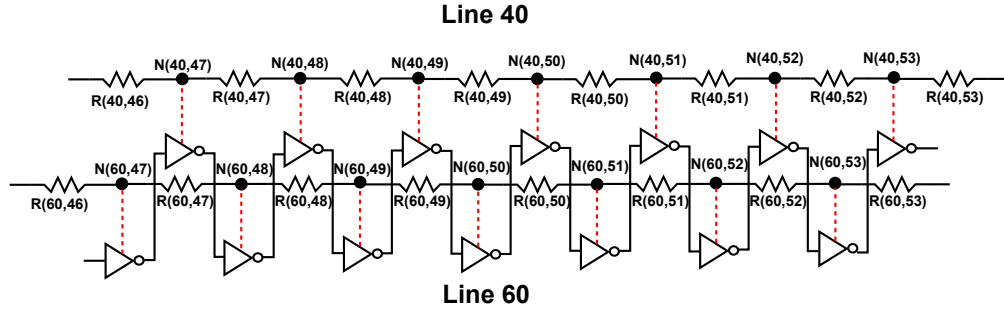


FIGURE 3.25: 14-inverters model for validation

gates connected to the same line is in turn enlarged by the additive effect of these predominant currents. Moreover, the weak current draws at connected nodes on the same line also appear in the same PDN. The accuracy of the current model can be relatively independently analyzed for the predominant current and the week current since they appear at nodes on two different lines in a PDN with a distance of 20 nodes; the impact on current distribution due to switching activities at such a distance is limited. Figure 3.26 presents the current flowing through $R(40, 50)$ in Vdd network (3.26a) and in Gnd network (3.26b). In each figure, two waveforms are illustrated representing the current values simulated by MIRID and SPICE. It can be observed that these two waveforms have similar shapes and close amplitudes. As a rising transition is applied to the inverter which is connected to $N(40, 50)$, the current draw due to the input switching is predominant in Gnd network. Correspondingly, the peak-to-peak amplitude of the current waveform for $R(40, 50)$ in Gnd network is about $4.5 \times 10^{-5} \text{ A}$, which is much larger than the amplitude of current waveform for $R(40, 50)$ in Vdd network. Moreover, the voltages at $N(40, 50)$ in Vdd and Gnd networks are presented in Figure 3.27a and Figure 3.27b, respectively. The voltage at $N(40, 50)$ in Vdd PDN varies around the nominal voltage 1.0V with a peak-to-peak amplitude up to $1.5 \times 10^{-5} \text{ V}$. The amplitude of voltage waveform at the Gnd supply node is about $3.2 \times 10^{-5} \text{ V}$, which is much higher due to the predominant current draw induced by the rising input transition of the connected gate.

The difference between both current (voltage) waveforms for each supply node is measured by NRMSE, computed using Equation 3.10. Two sets of 100 discrete current (voltage) values on the waveforms given by MIRID and SPICE, are denoted by y_t and \hat{y}_t ($t = 1 \dots 100$), respectively. The root mean square error is computed to measure the difference of the two sets of values, and normalized by the range of \hat{y}_t , given by $\hat{y}_{max} - \hat{y}_{min}$ where \hat{y}_{max} and \hat{y}_{min} are the maximum and minimum values in the set.

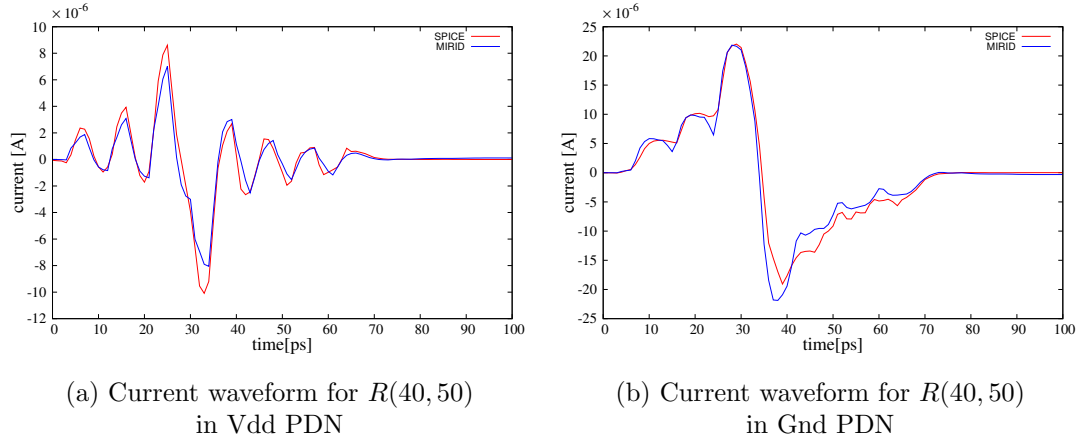


FIGURE 3.26: Current validation for 14-inverters chain

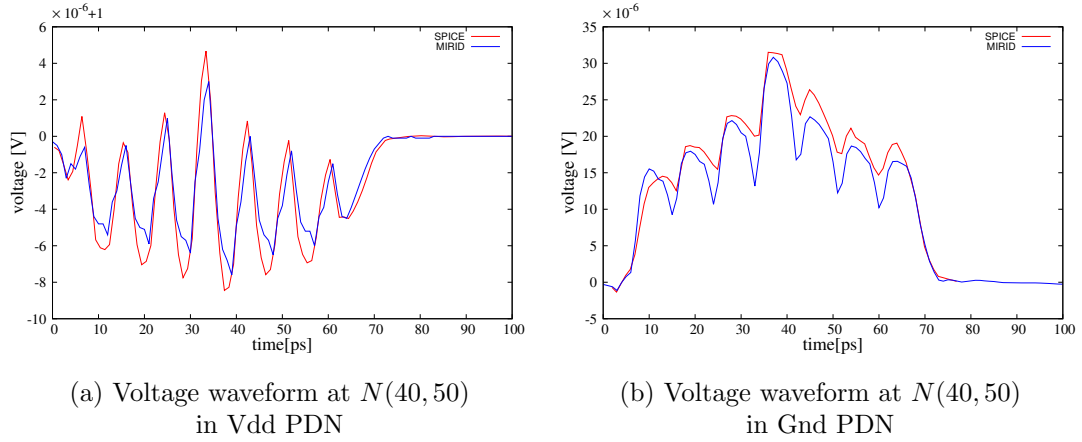


FIGURE 3.27: Voltage validation for 14-inverters chain

Due to space constraints, only the values of NRMSE computed for connected supply nodes and corresponding resistors are presented in Tables 3.1 and 3.2, respectively. In the PhD thesis by M. A. Rodriguez [111], who has performed the SPICE simulations for the experiments presented here, NRMSE is computed for an equivalent experiment using a different range of samples and thus have different values.

$$NRMSE = \frac{\sqrt{\sum_{t=1}^{100} (y_t - \hat{y}_t)^2 / 100}}{\hat{y}_{max} - \hat{y}_{min}} \quad (3.10)$$

The first column in Table 3.1 contains the coordinates of resistors on Line 40 in Vdd and Gnd networks. Corresponding values of NRMSE by approximating currents flowing through these resistors in Vdd and Gnd networks are presented in the second and third columns, respectively. Columns 4 through 6 present the NRMSE for resistors on Line 60 in the same way. The average value of NRMSE is given for each set of resistors, grouped

depending on whether their positions are on Line 40 or 60 and in Vdd or Gnd PDN. It turns out that the set of resistors on Line 60 in Gnd PDN (Column 6) has the largest average error of current estimation 7.07%. As gates connected to corresponding supply nodes (next to the resistors in this set) have falling input transitions, weak current draws appear in Gnd PDN due to the falling switching activities. The largest average error for this set of resistors implies an inferior accuracy in the weak current draw model. Note that the impact of weak current model on the accuracy in the global current model is limited since it has a smaller amplitude than the predominant current.

TABLE 3.1: Difference of currents in 14-inverter chain reported by MIRID and SPICE

R(40,y)	NRMSE[%] (Vdd)	NRMSE[%] (Gnd)	R(60,y)	NRMSE[%] (Vdd)	NRMSE[%] (Gnd)
(40,47)	4.81	7.01	(60,47)	5.22	4.65
(40,48)	4.30	5.91	(60,48)	3.20	4.81
(40,49)	3.92	5.06	(60,49)	2.97	6.90
(40,50)	3.82	4.53	(60,50)	2.71	7.53
(40,51)	3.96	4.03	(60,51)	2.58	8.40
(40,52)	4.02	3.65	(60,52)	2.62	8.77
(40,53)	3.91	2.81	(60,53)	2.80	8.43
Average	4.10	4.71	Average	3.15	7.07

Table 3.2 presents the estimation of difference between voltages at the considered supply nodes from both simulation flows. Column 1 and 4 contain the coordinates of the considered supply nodes at Line 40 and 60 in PDNs, respectively. Values of NRMSE are given in columns 2–3 and columns 5–6 correspondingly. For each set of NRMSE values, the average value is computed. It can be seen that for supply nodes on Line 40, the average NRMSE is higher in Vdd network than in Gnd network, while for nodes on Line 60, the situation is in reverse. In both cases when a higher average NRMSE is observed for nodes on Line 40 or on Line 60, a weak current is induced due to the corresponding rising or falling transition of connected gates. This result also indicates a probably lower accuracy in the modeling of the weak current compared to the modeling of the predominant one. The reason could be that the weak current is more complex and more sensitive to the power voltage supply [111].

Validation of the additional delay induced by IR drop is not possible for this small circuit since the additional delay has the order of magnitude 10^{-6} ps, which is much smaller than the time resolution of MIRID (1ps); the impact of voltage drop on the delay is almost imperceivable.

TABLE 3.2: Difference of voltages in 14-inverter chain reported by MIRID and SPICE

N(40,y)	NRMSE[%] (Vdd)	NRMSE[%] (Gnd)	N(60,y)	NRMSE[%] (Vdd)	NRMSE[%] (Gnd)
(40,47)	25.46	19.49	(60,47)	12.30	26.73
(40,48)	26.72	17.27	(60,48)	15.19	27.26
(40,49)	27.69	17.38	(60,49)	16.47	30.16
(40,50)	28.35	17.79	(60,50)	17.13	30.33
(40,51)	28.14	18.26	(60,51)	17.56	29.38
(40,52)	26.98	19.08	(60,52)	18.01	29.14
(40,53)	26.89	20.60	(60,53)	18.83	27.57
Average	27.17	18.55	Average	16.49	28.65

Another experiment is performed to the ISCAS'85 circuit c17, which contains 2 INV, 3 NOR2 and 4 NAND2 gates, to validate the current model with respect to different gates (and not only INV). To maximize the induced current, all gates are connected to a single supply node $N(50, 50)$ in both supply networks. A two-pattern test (01111, 01011) was applied to the circuit. The current and voltage waveforms at the node given by MIRID and SPICE are compared in Figure 3.28 and Figure 3.29, respectively. It can be seen that the current (voltage) waveforms according to both simulations have the similar shape with the difference of amplitude in order of magnitude 10^{-5} A (V).

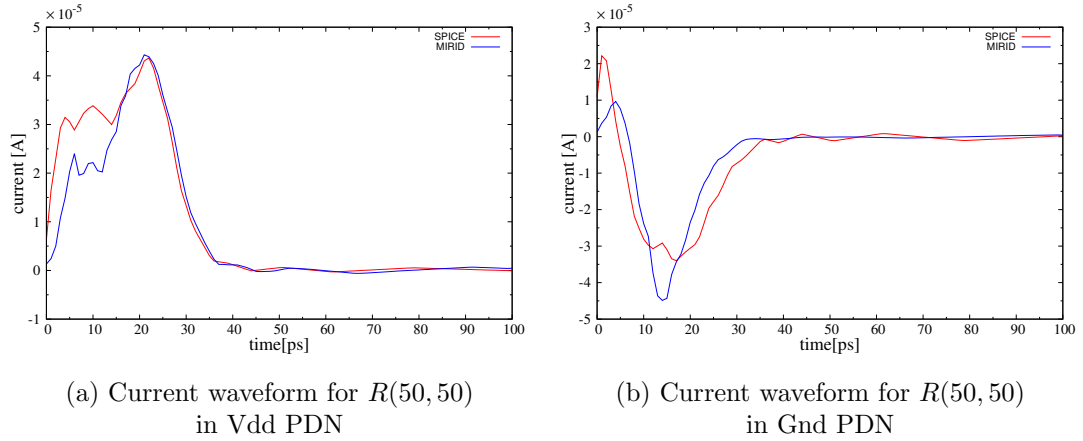


FIGURE 3.28: Current validation for c17

The difference is also estimated using NRMSE as given in Table 3.3. It seems that the error of current and voltage approximated by MIRID is limited (lower than 13%) with the simulation resolution of 1ps. The impact of the error on the induced delay has the order of magnitude 10^{-6} ps, which is very slight and much smaller than the simulation resolution. Though the difference of induced delays in both simulations is almost neglectable in this case, for larger benchmark circuits a noticeable difference in

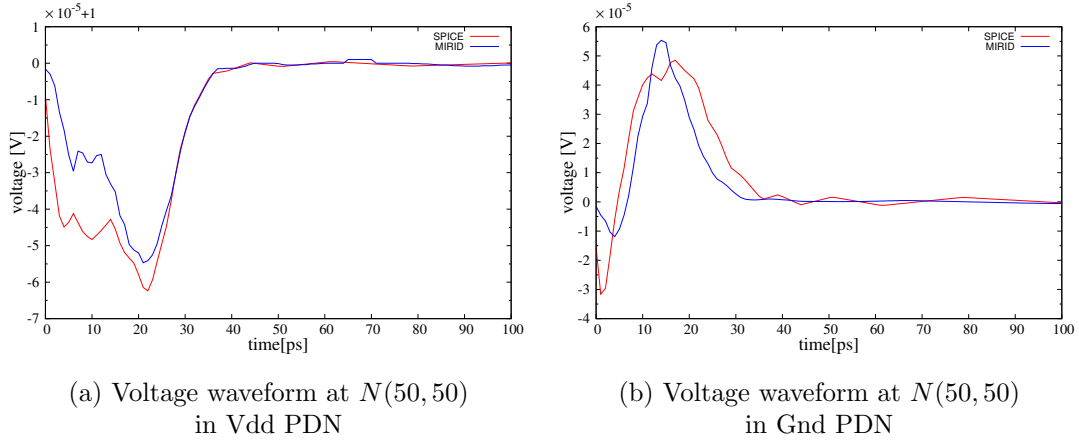
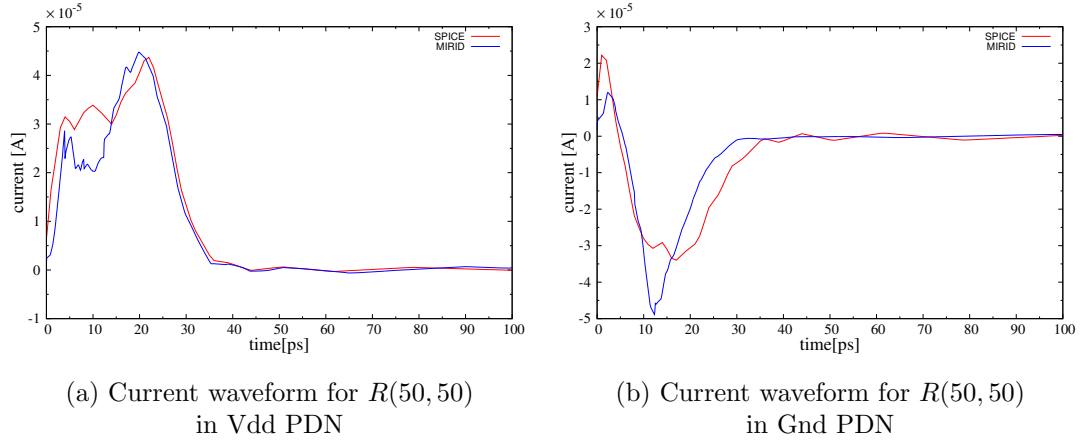
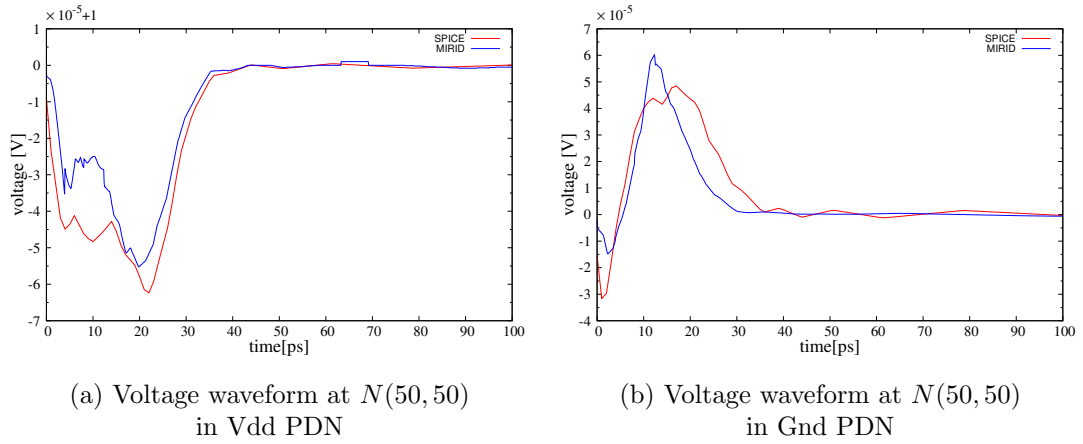


FIGURE 3.29: Voltage validation for c17

delay due to the simplified approximation by MIRID is still possible. Therefore, it is desired to raise the accuracy of the delay prediction, however, at a comparatively low cost of simulation as a trade-off between accuracy and runtime. To improve the accuracy, which highly depends on the current approximation, a question first arises what is the main cause of the difference between currents from both simulations. At the first glance, it was suspected that the difference mainly originates from the rounding error of current computed by MIRID in picosecond resolution. The same experiment was performed by MIRID with a smaller time step of 10^{-1} ps, in which the current was rounded with the corresponding precision and in turn the total computation time was raised enormously. The comparison of current and voltage waveforms from both simulations are given in Figure 3.30 and 3.31, respectively. The difference for each comparison is computed and also presented in Table 3.3. It turns out that the approximation using 10^{-1} ps precision does not improve the accuracy significantly. For approximations of current and voltage in Gnd network, even a slightly larger error is observed. The reason may be that due to the lack of SPICE simulation result using 10^{-1} ps resolution, only 100 samples (the current or voltage value at every picosecond on the waveforms) in each dataset are used for the computation of NRMSE. Nevertheless, it can be predicted that the result would not be much better if a larger dataset is used since the variation between samples is very limited.

As shown above, the time precision is not the key point to explain the difference between results from MIRID and SPICE. As proposed by the collaboration partner from LIRMM, who established the electrical model, a fundamental reason for the difference could be the underestimated input capacitance of gates. For the sake of simplicity, the

FIGURE 3.30: Current validation for c17 with 10^{-1} ps precisionFIGURE 3.31: Voltage validation for c17 with 10^{-1} ps precisionTABLE 3.3: Difference of current and voltage at $N(50,50)$ in c17 reported by MIRID and SPICE

Node	Precision	NRMSE[%]			
		Current		Voltage	
		Vdd	Gnd	Vdd	Gnd
N(50,50)	1ps	10.84	9.68	12.94	9.82
N(50,50)	0.1ps	8.70	9.86	11.37	9.87

electrical model assumes that each standard gate has the equivalent input capacitance of an inverter C_{min} , which is lower than or equal to the real value of its input capacitance. The load capacitance is given by $n \times C_{min}$ where n is the fanout of the gate output, i.e., the number of its downstream gates. Thus, as far as the input capacitance of each downstream gate is likely to be underestimated, so is the total load capacitance. As given in 3.4.3.1, the current approximation depends on various electrical parameters, among which the load capacitance is very important; the amplitude and time duration of the

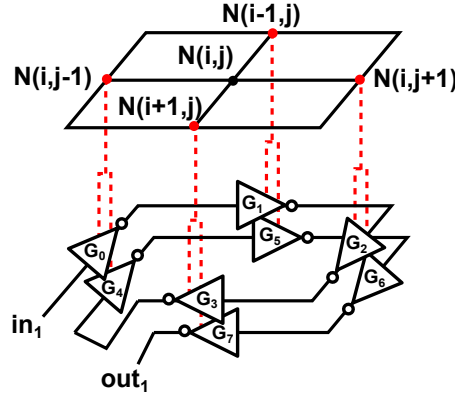
approximated current curve vary for different load capacitances. The underestimated load capacitance could mainly affect the accuracy of the current approximation and in turn cause the up to 13% error in Table 3.3. The error is expected to be reduced in the future version of electrical models that take different input capacitances in function of the type of gate rather than the fixed value C_{min} .

3.7.2 Investigation on IR Drop induced Delay

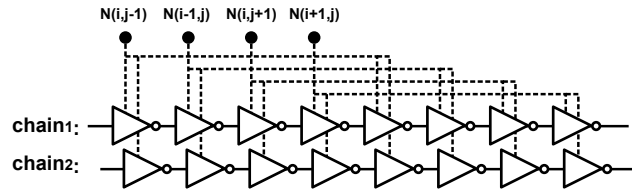
The experiments above aim at the validation of the applied electrical models, especially the model related to current approximation, which mainly determines the IR drop in a resistive PDN. However, the impact on the additional delay induced by IR drop cannot be validated by these experiments since the amount of additional delay is so tiny that the difference between delays reported by both simulations is neglectable. To validate the additional delay, a schematic is required that enables a relative high IR drop in PDN which in turn produces a noticeable additional delay. This requirement implies that the current draw caused by switching gates should be maximized, however, not too many gates should be involved since SPICE is not capable of simulating large circuits. Two aspects are considered by the design that satisfies the requirement: the dissipation of the induced current in time and the very located current distribution in PDN. Regarding the former, gates need to switch simultaneously to draw a totally significant amount of current at the switching time. Due to the latter, which also means a dissipation of voltage drop in a narrow neighborhood, a high density of gates connected to the small area is required to enhance the IR drop. In summary, voltage drop increases in a PDN area with a higher density of connected gates that switch simultaneously.

A dedicated design is therefore proposed for validating IR drop induced delay on a sufficiently small circuit for SPICE simulations. The circuit consists of a basic structure of an 8-inverter chain connected to a 2×2 PDN grid. As illustrated in Figure 3.32a, eight inverters G_0, \dots, G_7 are connected in a chain with a primary input in_1 and a primary output out_1 . The inverters are connected to supply nodes in the following way: G_0 and G_4 connected to node $N(i, j - 1)$, G_1 and G_5 connected to node $N(i - 1, j)$, G_2 and G_6 connected to node $N(i, j + 1)$, and G_3 and G_7 connected to node $N(i + 1, j)$. To increase the number of gates that switch simultaneously, a number of identical inverter chains can be parallel connected to supply nodes in exactly the same way. An example

of the parallel connection of two inverter chains is presented in Figure 3.32b where only the connected supply nodes and the inverter chains are drawn. The number of parallel connected chains can be configured, which enables regulation of the level of the induced voltage drop.



(a) Connection of an 8-inverter chain



(b) Parallel connection of two 8-inverter chains

FIGURE 3.32: 8-inverter chains connected to a 2×2 grid

As illustrated in Figure 3.33a, a simplified schematic uses a diamond shape with dashed lines to represent the 8-inverter chain connected to four adjacent nodes of $N(i,j)$ as given in Figure 3.32a. The experiment applies a design consisting of nine copies of such a schematic as presented in Figure 3.33b. Each diamond shape has the center at $N(50,50)$ or one of its neighboring node in a 100×100 PDN grid. Such a design raises the density of switching gates connected to the neighboring area of $N(50,50)$; the voltage drop at $N(50,50)$ is in turn enhanced. Moreover, by raising the number of parallel connected inverter chains at each position of the nine diamond shapes, the voltage drop at the center node can be further enhanced.

Simulations were performed by MIRID for the schematic illustrated in Figure 3.33b by setting the number of parallel connected inverter chains X from 1 to 100. For each inverter chain, a falling transition $1 \rightarrow 0$ was applied on its primary input. SPICE simulations were also performed by the collaboration partner from LIRMM using the same schematic and the same test. Both results show that the delay increases linearly

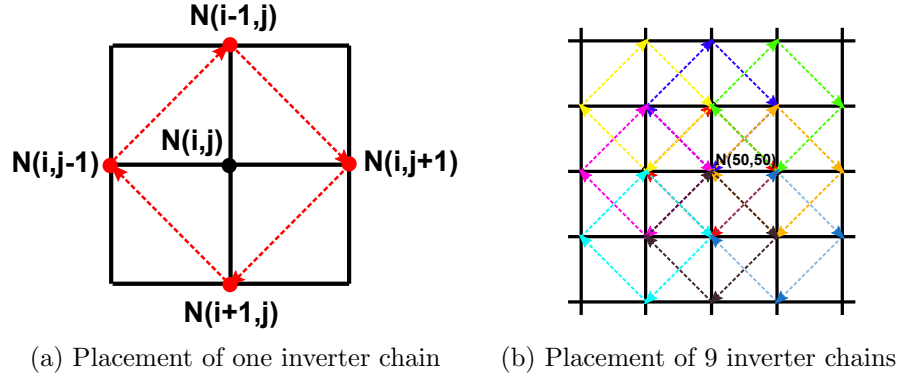
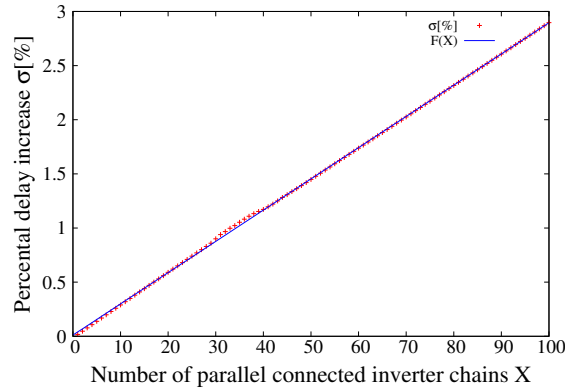


FIGURE 3.33: Model of 9 inverter chains

with the number of chains in parallel. Using the delay reported by SPICE as a reference, the relative error of the estimated delay by MIRID is up to 1.17% according to [111]⁸. The result seems satisfactory especially when taking into account that the error is accumulated by the replicated computation of delay for all parallel connected gates.

It is also interesting to know the correlation between the additional delay induced by IR drop and the number of switching events, which is proportional to the number of inverter chains in parallel. Compared to the nominal delay of an 8-inverter chain when IR drop is not considered, which is given by 35.64ps, the percental delay increase due to IR drop σ is computed for X in $\{1 \dots 100\}$. Figure 3.34 illustrates the result and a linear function $F(X)$ that best fits the result. It can be seen that the deviation of the linear approximation is very slight, which implies that the delay induced by IR drop increases almost linearly with the value of X . As X is proportional to the number of inverters, a linear relationship between the addition delay and the number of switching activities can be derived in this case.

FIGURE 3.34: Correlation of the IR drop induced delay and X

⁸The SPICE simulation was performed for X from 1 to 80, thus the error was computed based on the delays simulated by MIRID and SPICE in the same range of X .

3.7.3 Performance for Large Circuits

MIRID is designed to estimate the IR drop induced delay as a compromise between accuracy and efficiency. Thus, not only the validation of its accuracy, but also the evaluation of its performance for larger circuits is very important.

The simulator was run for several ISCAS and NXP circuits. The in-house tool Opt-KLPG was used to generate tests that sensitize the longest path through each gate under the unit-delay assumption. By sensitizing the longest path, as many as possible gates propagate the input transition along the path, which might result in the most switching events. The diverse locations of target gates throughout the circuit also enables a simulation of switching events evenly distributed in all parts of the circuit. For relatively small circuit (ISCAS'85), tests (if existing) are generated for all gates considered as possible defect sites; for larger NXP circuits, randomly selected 1000 gates are considered by the test generation.

Rather than using full-custom layouts of the circuits, each gate is connected to the central nodes in both 100×100 PDN grids. For each applied test, the IR drop induced delay is estimated by comparing the transition propagation time under two scenarios, with or without consideration of IR drop effect. In the latter case, the nominal gate delays are used by a conventional timing-aware logic simulation. The transition propagation time corresponds to the accumulated gate delay along the longest sensitized path.

The results are presented in Table 3.4. Column 2 contains the size of gates in each circuit. Note that MIRID has used a re-synthesized netlist due to the constraints of the electrical model⁹: each AND (OR) gate is replaced with a NAND (NOR) gate connected to a NOT gate; for each gate feeding more than 5 downstream gates, buffers are inserted to drive every 5 branches recursively. Thus, the actual number of gates in each simulated circuit instance is higher than the number presented in column 2, however, in the same order of magnitude. The number of applied tests is given in column 3. Column 4 and 5 present the average transition propagation time per test along the longest path without or with consideration of IR drop effect, denoted by case 1 and 2, respectively. The percental delay increase σ due to IR drop per test is given in column 6. The last

⁹Electrical models are established for standard gates excluding AND and OR gates, and for load capacitance up to 5 elementary equivalent capacitance.

two columns present the average number of switching activities and the average CPU runtime per test, respectively.

As shown in the table, the increase of delay due to IR drop ($\sigma[\%]$) is correlated with the number of switching activities. A counterexample for the positive correlation is given for p45 and p78. Compared to the result for p45, a high or average switching activity occurs by simulating p78, however, a lower increase in delay is observed in this case. The reason is that not only the number of switching gates but also other parameters incorporate into the total increase in delay. For example, the circuit containing more gates with longer IR-drop induced delays tends to have a larger total delay increase. A positive relation to the number of switching gates can also be observed for the runtime, which is not surprising as every switching event is processed in nearly constant time.

The circuit size has less impact on the delay increase, especially for the circuits of large sizes that are structurally more complex. For the group of ISCAS'85 circuits that have less than 3000 gates, a slight increase in σ is observed for the larger circuit. However, for NXP circuits which are much larger, the positive relation of the circuit size to the delay increase does not hold. For example, p35 has less gates than p45 and p78 but with a more complex structure which implies possibly longer paths and more switching events. The propagation time through the paths and the simulation runtime are in turn longer. The largest percental delay increase due to IR drop is 2.24% for p35, which is close to the value 3% reported previously in the literature [117]. Moreover, it can be observed that for most simulated circuits, the average runtime is less than 10s, which is quite acceptable.

TABLE 3.4: Simulation for ISCAS and NXP circuits

circuit	#gates	#patterns	prop. time[ps]		$\sigma[\%]$	#switching	sim. time[s] (CPU)
			case 1	case 2			
c432	322	322	130.89	131.04	0.11	300	<1
c1355	737	736	137.46	137.74	0.20	403	<1
c1908	795	795	188.83	189.24	0.22	951	2
c3540	1581	1581	249.18	249.89	0.29	1265	2
c7552	2952	2951	193.71	195.12	0.73	2554	5
p35	23267	1000	562.50	575.11	2.24	19488	50
p45	25679	1000	371.97	375.17	0.86	3052	6
p78	70475	1000	332.21	334.47	0.68	3213	7

3.7.4 Mitigation of the Short-path Problem using Buffers and Latches

The modified version of MIRID is used to empirically investigate the required number of buffers and latches to mitigate the short-path problem (Section 3.6) in actual circuits. Both procedures described in 3.6.3 are implemented accordingly. The core of the simulator, the timing-aware logic simulation, is basically not affected by the modification except the decision of timing of switching events that needs to match the behavior of inserted buffers and latches. The experiments were performed on re-synthesized versions of several ISCAS'85 and ITC'99 circuits and a 4-bit carry-ripple adder.

Table 3.5 summarizes the experimental results from both mitigation procedures. Column 1 contains the circuit name and the value of m , which is the number of gates on the structurally longest path; the variability of gates n and the calculated safe clock period $T_{safe} = mn$ is given in column 2 and column 3, respectively; column 4 presents the number of short paths in S .

For buffer padding, the variabilities n and b (column 5) and the number of buffers (column 6) sufficient for padding all short paths are included. For $n = 3$, the numbers of invalidated paths that cannot be padded because they have over $3/4 m$ gates are reported in column 8 “inv”; note that such solutions are inherently unreliable.

For latch placement two experiments were performed. In *Experiment 1*, a short path is equipped by a latch on position pos , where pos is the closest gate to the primary output for which all sub-paths from a primary input to pos are not longer than $m/2$. This placement guarantees the reliable operation for $T_{min} = 1/2 mn$, which is better than $T_{min} = 2/3 mn$ obtained by buffer padding. To objectively compare latch placement with buffer padding, *Experiment 2* was conducted for $T_{min} = 2/3 mn$, which is the same value of T_{min} that guarantees the reliable operation by buffer padding. Latches can be safely placed on locations that have maximal distance to all primary inputs and outputs less or equal to $2/3 m$. From these locations the one through which most short paths go is selected; a latch is placed on the location and the corresponding short paths are removed from S ; this procedure continues until S becomes empty. The number of latches determined in *Experiment 1* and *Experiment 2* are reported in columns “#1, exp1” (column 8) and “#1, exp2” (column 9), respectively.

TABLE 3.5: Experimental results for buffer padding and latch placement

circuit	n	T_{safe}	$ S $	buffer padding			latch placement	
				b	#buf	inv.	#l, exp1	#l, exp2
c432 (m=24)	1.2	28.8	1305	1.0	246	-	47	19
				1.2	264	-		
	1.5	36.0	4739	1.5	436	-		
	2.0	48.0	23006	1.5	989	-		
				2.0	985	-		
c1908 (m=32)	1.2	38.4	54700	1.0	1373	-	97	44
				1.2	1386	-		
	1.5	48.0	191331	1.5	1810	-		
b07 (m=15)	1.2	18.0	994	1.0	411	-	87	35
				1.2	431	-		
	1.5	22.5	3715	1.5	771	-		
	2.0	30.0	8711	1.5	1144	-		
				2.0	1169	-		
b12 (m=17)	1.2	20.4	3819	1.0	1501	-	217	146
				1.2	1510	-		
	1.5	25.5	8694	1.5	2120	-		
	2.0	34.0	13279	1.5	3458	-		
				2.0	3469	-		
4-bit carry ripple adder (m=12)	1.2	14.4	159	1.0	46	-	10	7
				1.2	47	-		
	1.5	18.0	195	1.5	58	-		
	2.0	24.0	253	1.5	103	-		
				2.0	105	-		
	3.0	36.0	261	1.0	203	40		

It can be seen that the number of required buffers is large for moderate variability and excessive for extreme variability. The number of latches is constant and much smaller than the number of buffers. This is the case even for *Experiment 1*, where latches are placed such as to guarantee $T_{min} = 1/2 mn$, which outperforms $T_{min} = 2/3 mn$ obtained by buffer padding. If the same T_{min} as for buffer padding is acceptable, the solutions become even more compact.

The latch behavior can be obtained by transformation of the preceding gate at the latch position¹⁰. The area costs of such a transformation of a NAND gate amounts to two transistors, which is comparable with the costs of one buffer. However, the latches

¹⁰Instead of inserting a physical memory element, it is possible to tri-state the preceding gate for the required behavior (the gate output enters high-impedance state during the first half of the clock period).

introduce further overhead for routing the clock signals to the latches and controlling their skews.

In summary, MIRID was shown to be useful beyond its originally intended use case. Due to its clearly defined interfaces between the logic simulation engine and electrical models, it can be adapt to various timing-aware simulations by specifying interface functions to dedicated timing models.

Chapter 4

Conclusion

The impact of process variations on the performance and reliability of VLSI circuits has been increased significantly as technology scales into the nanometer regime. Due to intrinsic limitations of fabrication process, variations in physical parameters are not avoidable and in turn cause variations in electrical parameters that affect the circuit timing. This negative impact of process variations has become a critical concern from both the design and testing perspectives. Existing design approaches combating process variations usually target on reduction of the impact by tuning certain operating parameters or by compensating and correcting the performance deviations induced by process variations adaptively. In context of testing under process variations, the target is to rule out the devices with poor timing performance or even have functional failures induced by extreme variations. Due to the random nature of parameter variations (that could follow normal or more complicated distributions), each circuit instance can present individual timing behavior. The challenge in this area is to develop a testing flow that is “statistic”-oriented in various testing stages, including fault modeling, test generation and fault simulation. However, due to the complexity of ATPG, it is impractical to generate tests for all possible parameter combinations. A flow that combines the ATPG for fixed parameter values with the statistical fault simulation over the whole parameter space is therefor of interest and proposed in [63].

One of the contributions of this thesis work is the ATPG tool called “Opt-KLPG” that can be integrated into the statistical testing flow and generate variation-aware tests pin-pointedly for SDDs. Opt-KLPG is developed based on the traditional KLPG

algorithm that aims at tests sensitizing multiple longest paths through the fault site given a fixed timing assignment. One problem related to the KLPG algorithm is that the search space could be extraordinarily large for benchmark circuits. Traditional KLPG algorithms usually tackle this problem by exposing a limited size on the data structure, called *path store*, that contains all paths to be considered. However, the limited size could lead to a sub-optimal solution when the optimal one (the sensitizable longest paths indeed) exists beyond the path store. For the first time, the optimality of the KLPG algorithm has been studied systematically by this thesis work. The proposed Opt-KLPG algorithm targets on the optimal solution by iteratively performing the KLPG flow for all sub-paths without discarding overflows from the path store.

Opt-KLPG was validated by experiments performed on academic/industrial circuits with up to 70K gates by showing the solution optimality under various configurations of path store size, denoted by π_{max} . Moreover, by comparing both the KLPG and Opt-KLPG solutions, it can be observed that conservative setting of π_{max} beyond 1000 does yield optimal result for most of (but not all) the considered circuits. In contract, setting π_{max} to very small values appears to have grave impact on the test quality, which means, the SDDs could be insufficiently tested by sensitizing less or shorter paths through them. Intuitively, a relatively large path store may be preferred for large circuits to obtain the (nearly) optimal solutions while the runtime is proportional to the store size. However, experimental results show that the minimum value of π_{max} used by KLPG for the optimal solution does not always grow with circuit sizes. It suggests that an appropriate π_{max} that results in the (nearly) optimal solution might be determined for circuits individually and empirically used, for example, for the slightly changed timing assignments due to variations.

The other interesting observation about runtime is that it rises with growing π_{max} in general but with exceptions for certain circuits, in which the runtime can be more than 50% shorter for larger π_{max} . In these circuits the potentially optimal solution are contained in the larger store while excluded from the smaller one so that many false paths need to be identified first as long as the solution is not found. It indicates that the runtime depends on the effort in identifying the false paths to a large extent. In most cases Opt-KLPG has similar or even better timing performance than KLPG targeting on the optimal solution. The reason is that Opt-KLPG benefits from the reduction of runtime by utilizing a smaller store, to find and update the solution iteratively while

eliminating sub-paths that are not potentially better than the existing solution; however, KLPG has to use a rather large store to find the optimal solution.

Metrics used to evaluate the optimality of KLPG solutions show that the optimal or nearly optimal solutions are given for many faults by KLPG when $\pi_{max} \geq 500$. Moreover, missed paths by KLPG for target faults could lead to escape of SDDs on these fault sites from the test in certain circuit instances, which corresponds to a severe situation with a significant impact on the testability under process variations. However, this situation rarely occurs when $\pi_{max} \geq 10$. Faults with shorter paths found by KLPG could be not adequately tested in circuit instances affected by variations. A significant number of gates turn out to be vulnerable to this problem even for large values of π_{max} .

Integrated into a statistical delay testing flow that aims at maximizing fault coverage under process variations, Opt-KLPG targets uncovered faults by the initial test set and generates tests that are likely to detect these faults. Experimental results show that the flow succeeds in generating tests that substantially outperform test sets obtained by conventional tools.

In brief, the experimental result shows that both KLPG and Opt-KLPG can generate tests for SDDs in large circuits with acceptable timing performance. An appropriate value of π_{max} can be determined empirically for each circuit that leads to a (nearly) optimal solution and a reasonable runtime. Interestingly, when both KLPG and Opt-KLPG target on the optimal solution, Opt-KLPG can have a similar or even better performance than KLPG by using a smaller path store.

The other contribution of this thesis work is a timing-aware simulation tool called MIRID for estimating the impact of IR drop, a major source of power supply noise in PDN, on circuit timing. An accurate estimation of the pattern-dependent impact is important for power-aware tests. As a compromise between simulation efficiency and accuracy, the mixed-mode simulator combines an event-driven logic simulation with electrical models incorporating electrical parameters that mainly contribute to the IR-drop effect, provided with interface functions between these two parts. The logic simulation is developed independent of the electrical models, which means, it can be adapted to modified electrical models with minor effort in editing the interfaces only. For the first time, the PDN configuration is integrated into the mixed-mode simulation.

Experiments were performed for various purposes. The accuracy of electrical models applied by MIRID is evaluated by comparing results from both SPICE and MIRID simulations for the same circuit design. The difference in results is represented by NRMSE, which corresponds to the model accuracy. The impact of the difference in the induced delay has the order of magnitude 10^{-6} ps, which is slight and much smaller than the simulation resolution (and thus almost neglectable). However, as only very simple designs were used due to the complexity of SPICE, a noticeable difference in delay due to the simplified approximation of electrical models is still possible for large benchmark circuits. A fundamental reason for the difference could be the underestimated input capacitance of gates, which is assumed by the electrical model to be the equivalent input capacitance of an inverter for every standard gate. The accuracy of MIRID is expected to be significantly improved by applying the electrical model with more realistic assumptions.

The impact of IR drop on the circuit timing is estimated for a dedicated schematic that tends to produce a noticeable additional delay. The relative error of the delay estimated by MIRID (with reference to the SPICE result) is up to 1.17%, which seems to be quite satisfactory. To evaluate the IR drop induced delay, MIRID simulations were performed with or without consideration of IR drop effect, for the dedicated schematic and academic/industrial circuits. Based on the delays reported in both scenarios, the percental delay increase due to IR drop is calculated. A positive relation between the percental delay increase and the rate of switching activity can be observed for most of the considered circuits. The percental delay increase due to IR drop is up to 2.24%. However, it should be noted that delay increase could be mitigated by the presence of capacitive elements in PDN that compensate the voltage drop effect but are not considered by the simplified electrical model. Moreover, it seems that the size of considered circuits has less impact on the delay increase, especially for circuits of large sizes and are structurally more complex.

The other important objective of the experiments is to evaluate the performance of MIRID for large circuits, measured by the runtime. It can be observed that the runtime rises as more gates switch, which is not surprising since every switching event is processed in nearly constant time. For most simulated circuits, the average runtime is less than 10s, which suggests that the simulator can handle large academic/industrial circuits.

In general, the simulator can be adapt to other applications that require a timing-aware logic simulation, not restricted to simulating the IR drop induced delay. A first application of the simulator reported in this thesis is to tackle the short-path problem of a self-adaptive better-than-worst-case design [5], by buffer padding or latch placement in the circuit. Short paths can cause the error-detection mechanism in the self-adaptive design to be unreliable. To mitigate this problem, application conditions for buffer padding and latch placement are derived based on a simplified model; corresponding strategies are developed and integrated into the simulation flow.

To verify the effectiveness of MIRID in mitigation of the short-path problem, experiments were performed for several ISCAS'85 and ITC'99 circuits and a 4-bit carry-ripple adder using both strategies of buffer padding and latch placements. It can be seen that the number of required buffers is large for moderate variability and excessive for extreme variability while the number of latches is constant and much smaller. Moreover, T_{min} (the minimum clock frequency for reliable operations) is guaranteed to be shorter by the latch placement than by buffer padding. If the same T_{min} as for buffer padding is acceptable, the latch solutions become even more compact. However, it should be noted that the latches introduce further overhead for routing the clock signals to the latches and controlling their skews.

In future work, the ATPG tool Opt-KLPG can be accelerated by applying speed-up techniques, such as using learning strategy during the path-searching process to trim off the search space incrementally, or by investigating correlations between gates to identify gates whose solutions can only belong to existing ones, and thus need not to be considered. The accuracy of the MIRID simulator can be raised by applying more accurate and complex electrical models. The adaption of the simulator for such electrical models is associated with low effort due to a clear and versatile interface function design. Moreover, algorithmic improvements that enhance the application performance, e. g., integration of the concurrent simulation method into the application, could be a part of the future work.

In summary, the contribution of this thesis work consists of two approaches Opt-KLPG and MIRID that generates high-quality tests targeting small delay faults and simulates IR drop induced delay, respectively. Using the proposed ATPG tool Opt-KLPG, the

impact of the path store size utilized by conventional KLPG algorithms on the solution optimality has been systematically investigated; the solution optimality is guaranteed by Opt-KLPG. And, for the first time, the PDN configuration is integrated into the mixed-mode simulator MIRID to raise the simulation accuracy. These two approaches can be incorporated into a statistical delay testing flow, by means of generating variation-aware tests and efficiently yet accurately simulating the impact of power supply noises on circuit timing.

Bibliography

- [1] F. Hopsch, B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H. Wunderlich. Variation-aware fault modeling. *IEEE Asian Test Symp.*, pages 87–93, 2010.
- [2] N. Jha and S. Gupta. *Testing of Digital Systems*. Cambridge University Press, 2003.
- [3] S. Borkar. Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Micro*, 25:10–16, 2006.
- [4] 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 Trans. on VLSI Systems*, 11:888–899, 2003.
- [5] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. *IEEE/ACM Int’l Symp. on Microarchitecture*, 36:7–18, 2003.
- [6] X. Lu, Z. Li, W. Qiu, D. M. H. Walker, and W. Shi. Longest-path selection for delay test under process variation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(12):1924–1929, 2005.
- [7] S. Tani, M. Teramoto, T. Fukazawa, and K. Matsuhiro. Efficient path selection for delay testing based on partial path evaluation. *IEEE VLSI Test Symp.*, pages 188–193, 1998.
- [8] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor. Test-pattern grading and pattern selection for small-delay defects. *IEEE VLSI Test Symp.*, pages 233–239, 2008.

- [9] E. Park, M. Mercer, and T. Williams. Statistical delay fault coverage and defect level for delay faults. *IEEE Int'l. Test Conf.*, pages 492–499, 1988.
- [10] J. Liou, A. Krstic, Y. Jiang, and K. Cheng. Modeling, testing, and analysis for delay defects and noise effects in deep submicron devices. *IEEE Trans. on CAD*, 22(6):756–769, 2003.
- [11] M. Shintani, T. Uezono, T. Takahashi, H. Ueyama, T. Sato, K. Hatayama, T. Aikyo, and K. Masu. An adaptive test for parametric faults based on statistical timing information. *IEEE Asian Test Symp.*, pages 151–156, 2009.
- [12] U. Ingelsson, B. Al-Hashimi, S. Khursheed, S. M. Reddy, and P. Harrod. Process variation-aware test for resistive bridges. *IEEE Trans. on CAD*, 28(8):1269–1274, 2009.
- [13] D. Blaauw, K. Chopra, A. Srivastava, and L. Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(4):589–607, 2008.
- [14] L. Scheffer. Physical CAD changes to incorporate design for lithography and manufacturability. *IEEE Design Automation Conf.*, page 768–773, 2004.
- [15] S. R. Nassif. Modeling and analysis of manufacturing variations. *IEEE Conf. on Custom Integrated Circuits*, pages 223–228, 2001.
- [16] K. S. Kim, S. Mitra, and P. G. Ryan. Delay defect characteristics and testing strategies. *IEEE Design & Test of Computers*, 20:8–16, 2003.
- [17] S. K. Goel and K. Chakrabarty. *Testing for Small-Delay Defects in Nanoscale CMOS Integrated Circuits*. CRC Press, 2001.
- [18] S. Mitra, E. Volkerink, E. J. McCluskey, and S. Eichenberger. Delay defect screening using process monitor structures. *IEEE VLSI Test Symp.*, pages 43–48, 2004.
- [19] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor. Test-pattern selection for screening small-delay defects in very-deep submicrometer integrated circuits. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29:760–773, 2010.

- [20] N. Ahmed, M. Tehranipoor, and V. Jayaram. Timing-based delay test for screening small delay defects. *IEEE/ACM Design Autom. Conf.*, page 320–325, 2006.
- [21] X. Lin, T. Kun-Han, W. Chen, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada, and T. Aikyo. Timing-aware ATPG for high quality at-speed testing of small delay defects. *IEEE Asian Test Symp.*, pages 139–146, 2006.
- [22] P. Nigh and A. Gattiker. Test method evaluation experiments and data. *IEEE Int’l Test Conf.*, pages 454–463, 2000.
- [23] E. S. Park, M. R. Mercer, and T. W. Williams. Statistical delay fault coverage and defect level for delay faults. *IEEE Int’l Test Conf.*, pages 492–499, 1988.
- [24] A. K. Pramanick and S. M. Reddy. On the detection of delay faults. *IEEE Int’l Test Conf.*, pages 845–856, 1988.
- [25] K. T. Cheng and H. C. Chen. Delay testing for non-robust untestable circuits. *IEEE Int’l Test Conf.*, pages 954–961, 1993.
- [26] J. Jung and T. Kim. Variation-aware false path analysis based on statistical dynamic timing analysis. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 31(11):1684–1697, 2012.
- [27] A. Agarwal, D. Blaauw, V. Zolotov, S. Sundareswaran, M. Zhao, K. Gala, and R. Panda. Statistical delay computation considering spatial correlations. *IEEE Design Automation Conf.*, pages 271–276, 2003.
- [28] J. Jess, K. Kalafala, S. Naidu, R. Otten, and C. Visweswariah. Statistical timing for parametric yield prediction of digital integrated circuits. *IEEE Design Automation Conf.*, page 932–937, 2003.
- [29] 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. *IEEE Design Automation Conf.*, pages 535–540, 2005.
- [30] Y. Xie and Y. Chen. Statistical high-level synthesis under process variability. *IEEE Design & Test of Computers*, 26(4):78–87, 2009.

- [31] K. Kuhn, C. Kenyon, A. Kornfeld, M. Liu, A. Maheshwari, S. Wei-kai, S. Sivakumar, G. Taylor, P. VanDerVoorn, and K. Zawadzki. Managing process variation in Intel's 45nm CMOS technology. *Intel Technology Journal*, 12(2), 2008.
- [32] B. Stefano, D. Bertozzi, L. Benini, and E. Macii. Process variation tolerant pipeline design through a placement-aware multiple voltage island design style. *IEEE Design, Automation and Test in Europe Conf.*, pages 967–972, 2008.
- [33] J. W. Tschanz, J. Kao, S. Narendra, R. Nair, D. Antoniadis, A. Chandrakasan, and V. De. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *IEEE Journal of Solid-State Circuits*, 37:1396–1402, 2002.
- [34] M. Nicolaidis. GRAAL: a new fault tolerant design paradigm for mitigating the flaws of deep nanometric technologies. *IEEE Int'l Test Conf.*, pages 1–10, 2007.
- [35] B. Doyle, P. Mahoney E. Fetzner, and S. Naffziger. Clock distribution on a dual-core, multi-threaded Itanium[®] family microprocessor. *IEEE Int'l Conf. on Integrated Circuit Design and Technology*, pages 1–6, 2005.
- [36] D. Sylvester, D. Blaauw, and E. Karl. ElastIC: An adaptive self-healing architecture for unpredictable silicon. *IEEE Design & Test of Computers*, 23:484–490, 2006.
- [37] M. Simone, M. Lajolo, and D. Bertozzi. Variation tolerant NoC design by means of self-calibrating links. *IEEE Design, Automation and Test in Europe*, 23:1402–1407, 2008.
- [38] W. Qiu and D. M. H. Walker. An efficient algorithm for finding the K longest testable paths through each gate in a combinational circuit. *IEEE Int'l Test Conf.*, pages 592–601, 2003.
- [39] J. P. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10(4):278–291, 1966.
- [40] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. on Computers*, 30(3):215–222, 1981.
- [41] H. Fujiwara and T. Shinomo. On the acceleration of test generation algorithms. *IEEE Trans. on Computers*, 32(12):1137–1144, 1983.

- [42] I. Hamzaoglu and J. H. Patel. New techniques for deterministic test pattern generation. *IEEE VLSI Test Symp.*, pages 446–452, 1998.
- [43] R. Drechsler, S. Eggersgluss, G. Fey, A. Glowatz, F. Hapke, J. Schloeffel, and D. Tille. On acceleration of SAT-based ATPG for industrial designs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1329–1333, 2008.
- [44] A. Czutro, I. Polian, M. Lewis, P. Engelke, S. M. Reddy, and B. Becker. Thread-parallel integrated test pattern generator utilizing satisfiability analysis. *Int’l Journal of Parallel Programming*, 38:185–202, 2010.
- [45] M. Sauer, B. Becker, and I. Polian. PHAETON: A SAT-based framework for timing-aware path sensitization. *IEEE Trans. on Computers*, pages 1–14, 2015.
- [46] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.
- [47] G. Fey, T. Warode, and R. Drechsler. Reusing learned information in SAT-based ATPG. *IEEE Int’l Conf. on VLSI Design*, pages 69–76, 2007.
- [48] K. L. Shepard and V. Narayanan. Noise in deep submicron digital design. *IEEE Int’l Conf. on Computer-Aided Design*, pages 524–531, 1996.
- [49] H. H. Chen and D. D. Ling. Power supply noise analysis methodology for deep submicron VLSI design. *IEEE Design Automation Conf.*, pages 638–643, 1997.
- [50] Y. M. Jiang and K. T. Cheng. Analysis of performance impact caused by power supply noise in deep submicron devices. *IEEE Design Automation Conf.*, pages 760–765, 1999.
- [51] C. Tirumurti, S. Kundu, S. K. Susmita, and Y. S. Change. A modeling approach for addressing power supply switching noise related failures of integrated circuits. *IEEE Design, Automation and Test in Europe Conf.*, pages 1078–1083, 2004.
- [52] K. Shakeri, R. Savari, and J. D. Meindl. Compact physical IR-drop models for GSI power distribution networks. *IEEE Int’l Interconnect Technology Conf.*, pages 54–56, 2003.

- [53] S. K. Nithin, G. Shanmugam, and S. Chandrasekar. Dynamic voltage (IR) drop analysis and design closure: Issues and challenges. *IEEE Int'l Symp. on Quality Electronic Design (ISQED)*, pages 611–617, 2010.
- [54] J. Rius. IR-drop in on-chip Power Distribution Networks of ICs with nonuniform power consumption. *IEEE VLSI Test Symp.*, 21:512–522, 2013.
- [55] M. Aparicio-Rodriguez, M. Comte, F. Azaïs, M. Renovell, J. Jiang, I. Polian, and B. Becker. Pre-characterization procedure for a mixed mode simulation of IR-drop induced delays. *IEEE Latin-American Test Workshop*, pages 1–6, 2013.
- [56] V. Iyengar, B. K. Rosen, and I. Spillinger. Delay test generation. I. concepts and coverage metrics. *IEEE Int'l Test Conf.*, pages 857–866, 1988.
- [57] J. Jiang, M. Sauer, A. Czutro, B. Becker, and I. Polian. On the optimality of K longest path generation algorithm under memory constraints. *IEEE Conf. on Design, Automation and Test in Europe*, pages 418–423, 2012.
- [58] B. Becker, R. Drechsler, S. Eggersgluss, and M. Sauer. Recent advances in SAT-based ATPG: Non-standard fault models, multi constraints and optimization. *IEEE Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*, pages 1–10, 2014.
- [59] N. Eén and N. Sörensson. An extensible SAT solver. *Int'l Conf. on Theory and Applications of Satisfiability Testing, ser. Lecture Notes in Computer Science*, 2919: 502–518, 2004.
- [60] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. *Int'l Joint Conf. on Artificial Intelligence*, page 386–392, 2007.
- [61] T. Schubert, M. Lewis, and B. Becker. Antom–solver description. *SAT Race*, 2010.
- [62] J. Benkoski, E. V. Meersch, L. J. M. Claesen, and H. D. Man. Timing verification using statically sensitizable paths. *IEEE Trans. on Computer-Aided Design*, 9(10): 1073–1084, 1990.
- [63] A. Czutro, M. E. Imhof, J. Jiang, A. Mumtaz, M. Sauer, B. Becker, I. Polian, and H. Wunderlich. Variation-aware fault grading. *IEEE Asian Test Symp.*, pages 344–349, 2012.

- [64] <http://realtest.date.uni-paderborn.de/home.html>.
- [65] M. Sauer, J. Jiang, A. Czutro, I. Polian, and B. Becker. Efficient SAT-based search for longest sensitisable paths. *IEEE Asian Test Symp.*, pages 108–113, 2011.
- [66] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN. *IEEE Int’l Symp. on Circuits and Systems*, 1985.
- [67] S. Davidson. ITC’99 benchmark circuits — preliminary results. *IEEE Int’l Test Conf.*, 1999.
- [68] Nangate 45nm open cell library, <http://www.nangate.com>.
- [69] Int’l. technology roadmap for semiconductors (ITRS), <http://www.itrs.net>. 2011.
- [70] S. C. Ma, P. Franco, and E. J. McCluskey. An experimental chip to evaluate test techniques experiment results. *IEEE Int’l Test Conf.*, pages 663–672, 1995.
- [71] N. H. E. Weste and D. M. Harris. *CMOS VLSI Design*. Addison-Wesley, 2011.
- [72] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits*. Pearson Education, Inc., 2003.
- [73] S. R. Nassif and O. Fakhouri. Technology trends in power-grid-induced noise. *Int’l Workshop on System-level Interconnect Prediction*, pages 55–59, 2002.
- [74] T. Rahal-Arabi, G. Taylor, M. Ma, and C. Webb. Design and validation of the Pentium III and Pentium 4 processors power delivery. *Symp. on VLSI Circuits, Digest of Technical Papers*, pages 220–223, 2002.
- [75] K. Shakeri and J. D. Meindl. Compact physical IR-drop models for chip/package co-design of Gigascale Integration (GSI). *IEEE Trans. on Electron Devices*, 52(6): 1087–1096, 2005.
- [76] N. Badereddine, P. Girard, S. Pravossoudovitch, and H. Wunderlich. Structural-based power-aware assignment of don’t cares for peak power reduction during scan testing. *IEEE IFIP Int’l Conf.*, pages 403–408, 2006.
- [77] C. Di and J. A. G. Jess. An efficient CMOS bridging fault simulator: with spice accuracy. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 1071–1080, 1996.

- [78] L. W. Nagel and D. O. Pederson. SPICE (Simulation Program with Integrated Circuit Emphasis). *EECS Department, University of California, Berkeley, Technical Report No. UCB/ERL M382*, 1973.
- [79] F. J. Ferguson and J. P. Shen. Extraction and simulation of realistic CMOS faults using inductive fault analysis. *IEEE Int'l Test Conf.*, pages 475–484, 1988.
- [80] J. Khare and W. Maly. Inductive contamination analysis (ICA) with SRAM application. *IEEE Int'l Test Conf.*, pages 552–560, 1995.
- [81] J. Jiang, M. Aparicio-Rodriguez, M. Comte, F. Azaïs, M. Renovell, and I. Polian. MIRID: Mixed-mode IR-drop induced delay simulator. *IEEE Asian Test Symp.*, pages 177–182, 2013.
- [82] A. B. Kahng, J. Lienig, I. L. Markow, and J. Hu. *VLSI Physical Design: From Graph Partitioning to Timing Closure*. Springer Science & Business Media, 2011.
- [83] J. R. Black. Electromigration—a brief survey and some recent results. *IEEE Trans. on Electron Devices*, 16(4):338–347, 1969.
- [84] R. Saleh, S. Z. Hussain, S. Rochel, and D. Overhauser. Clock skew verification in the presence of IR-drop in the power distribution network. *IEEE Trans. on Computer-Aided Design*, 19(6):635–644, 2000.
- [85] S. Pant, D. Blaauw, V. Zolotov, S. Sundareswaran, and R. Panda. Vectorless analysis of supply noise induced delay variation. *IEEE Int'l Conf. on Computer-Aided Design*, pages 184–191, 2003.
- [86] S. Ravi. Power-aware test: Challenges and solutions. *IEEE Int'l Test Conf.*, pages 1–10, 2007.
- [87] P. Girard. Survey of low-power testing of VLSI circuits. *IEEE Design & Test of Computers*, 19(3):80–90, 2002.
- [88] Y. Zorian. A distributed BIST control scheme for complex VLSI devices. *IEEE VLSI Test Symp.*, pages 4–9, 1993.
- [89] S. Wang and S. K. Gupta. ATPG for heat dissipation minimization during test application. *IEEE Int'l Test Conf.*, pages 250–258, 1994.

- [90] S. Chakravarty and V. P. Dabholkar. Two techniques for minimizing power dissipation in scan circuits during test application. *IEEE Asian Test Symp.*, pages 324–329, 1994.
- [91] P. Girard, C. Landrault, S. Pravossoudovitch, and D. Severac. Reducing power consumption during test application by test vector ordering. *IEEE Circuits and Systems*, 2:296–299, 1998.
- [92] P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch. A test vector ordering technique for switching activity reduction during test operation. *IEEE Ninth Great Lakes Symp. on VLSI*, 9:24–27, 1999.
- [93] W. D. Tseng. Scan chain ordering technique for switching activity reduction during scan test. *IEEE Computers and Digital Techniques*, 152(5):609–617, 2005.
- [94] K. Sankaralingam, R. R. Oruganti, and N. A. Toubia. Static compaction techniques to control scan vector power dissipation. *IEEE VLSI Test Symp.*, pages 35–40, 2000.
- [95] L. Whetsel. Adapting scan architectures for low power operation. *IEEE Int’l Test Conf.*, pages 863–872, 2000.
- [96] B. Pouya and A. L. Crouch. Optimization trade-offs for vector volume and test power. *IEEE Int’l Test Conf.*, pages 873–881, 2000.
- [97] K. Sankaralingam, B. Pouya, and N. A. Toubia. Reducing power dissipation during test using scan chain disable. *IEEE VLSI Test Symp.*, pages 319–324, 2001.
- [98] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch. A gated clock scheme for low power scan testing of logic ICs or embedded cores. *IEEE Asia Test Symp.*, pages 253–258, 2001.
- [99] S. Wang and S. K. Gupta. DS-LFSR: a new BIST TPG for low heat dissipation. *IEEE Int’l Test Conf.*, pages 848–857, 1997.
- [100] P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch, and H. Wunderlich. A modified clock scheme for a low power BIST test pattern generator. *IEEE VLSI Test Symp.*, pages 306–311, 2001.

- [101] P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch. Low power BIST design by hypergraph partitioning: methodology and architectures. *IEEE Int'l Test Conf.*, pages 652–661, 2000.
- [102] <http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeRail.aspx>.
- [103] <https://www.apache-da.com/products/redhawk>.
- [104] W. I. Baker, A. Mahmood, and B. S. Carlson. Parallel event-driven logic simulation algorithms: tutorial and comparative evaluation. *IEEE Proceedings-Circuits, Devices and Systems*, 143(4):177–185, 1996.
- [105] L. Soule and T. Blank. Statistics for parallelism and abstraction level in digital simulation. *IEEE Design Automation Conf.*, pages 588–591, 1987.
- [106] M. L. Bailey. How circuit size affects parallelism. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(2):208–215, 1992.
- [107] H. Kriplani, F. N. Najm, and I. N. Hajj. Pattern independent maximum current estimation in power and ground buses of CMOS VLSI circuits: Algorithms, signal correlations, and their resolution. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, pages 998–1012, 1995.
- [108] L.-C. Wang, W. Qiu, S. Fancier, and D. M. H. Walker. Static compaction of delay tests considering power supply noise. *IEEE VLSI Test Symp.*, pages 235–240, 2005.
- [109] <http://www.open-std.org/jtc1/sc22/wg21/>.
- [110] IEEE Standard Verilog Hardware Description Language. <http://ieeexplore.ieee.org/servlet/opac?punumber=7578>, 2001.
- [111] M. Aparicio-Rodriguez. *Modeling and simulation of the IR-Drop phenomenon in integrated circuits*. PhD thesis, University of Montpellier II, 2013.
- [112] D. Bull, S. Das, K. Shivashankar, G. S. Dasika, K. Flautner, and D. Blaauw. A power-efficient 32 bit ARM processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation. *IEEE Journal of Solid-State Circuits*, 46:18–31, 2010.

- [113] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. L. Lu, T. Karnik, and V. De. Energy-efficient and metastability-immune timing-error detection and recovery circuits for dynamic variation tolerance. *IEEE Journal of Solid-State Circuits*, 44:49–63, 2009.
- [114] I. Polian, J. Jiang, and A. Singh. Detection conditions for errors in self-adaptive better-than-worst-case designs. *IEEE European Test Symp.*, pages 1–6, 2014.
- [115] N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constraints. *IEEE Int’l Conf. On Computer Aided Design*, pages 156–161, 1993.
- [116] D. Bull and S. Das. Latch to block short path violation. *US Patent 2008/0086624 A1*, 2008.
- [117] Z. Jiang, Z. Wang, J. Wang, and D. M. H. Walker. Levelized low cost delay test compaction considering IR-drop induced noise. *IEEE VLSI Test Symp.*, pages 52–57, 2011.