FAULT SIMULATION AND TEST GENERATION FOR SMALL DELAY FAULTS

A Dissertation

by

WANGQI QIU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2006

Major Subject: Computer Science

FAULT SIMULATION AND TEST GENERATION FOR SMALL DELAY FAULTS

A Dissertation

by

WANGQI QIU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Duncan M. Walker
Committee Members,	Jyh-Charn Liu
	Rabinarayan Mahapatra
	Weiping Shi
Head of Department,	Valerie E. Taylor

December 2006

Major Subject: Computer Science

ABSTRACT

Fault Simulation and Test Generation for Small Delay Faults.

(December 2006)

Wangqi Qiu, B.S., Fudan University, China

Chair of Advisory Committee: Dr. Duncan M. Walker

Delay faults are an increasingly important test challenge. Traditional delay fault models are incomplete in that they model only a subset of delay defect behaviors. To solve this problem, a more realistic delay fault model has been developed which models delay faults caused by the combination of spot defects and parametric process variation.

According to the new model, a realistic delay fault coverage metric has been developed. Traditional path delay fault coverage metrics result in unrealistically low fault coverage, and the real test quality is not reflected. The new metric uses a statistical approach and the simulation based fault coverage is consistent with silicon data. Fast simulation algorithms are also included in this dissertation.

The new metric suggests that testing the K longest paths per gate (KLPG) has high detection probability for small delay faults under process variation. In this dissertation, a novel automatic test pattern generation (ATPG) methodology to find the K longest testable paths through each gate for both combinational and sequential circuits is presented. Many techniques are used to reduce search space and CPU time significantly. Experimental results show that this methodology is efficient and able to handle circuits

with an exponential number of paths, such as ISCAS85 benchmark circuit c6288.

The ATPG methodology has been implemented on industrial designs. Speed binning has been done on many devices and silicon data has shown significant benefit of the KLPG test, compared to several traditional delay test approaches.

DEDICATION

To my wife, parents and grandmother:

without their support, this would not have been possible.

ACKNOWLEDGMENTS

My doctoral study at Texas A&M University has been a great experience, since I started my Ph.D. program in the Fall semester of 2000. The study and dissertation has been supported by many colleagues for more than five years. Texas A&M University provides its students with superb academic freedom, excellent coursework, and a well-developed system, to help her students maximize their achievements.

I would like to thank Dr. Duncan M. (Hank) Walker, my advisor and committee chair, for his advising and kind support from the first day I landed in the United States. He guided me into the electronic design automation field, helped me enjoy the pleasure of research, kept me always going in the right direction, gave me novel ideas, and finally helped me achieve my goals. This dissertation would never have been completed without his advice and encouragement. The treasure I got from him will benefit my whole life.

I would also like to thank Dr. Weiping Shi and his students Xiang Lu and Zhuo Li. We have had very smooth and productive cooperation for years. Dr. Shi is an expert in VLSI physical design and extremely knowledgeable in delay fault modeling. I enjoyed working with him on a variety of projects, from very theoretical to very practical problems. Under his guidance, Xiang and Zhuo developed accurate as well as easy-to-use models for new technology, which greatly simplified my work, with more accurate results.

I am grateful to my committee. Besides Dr. Walker and Dr. Shi, Drs. Jyn-Charn Liu and Rabinarayan Mahapatra gave me a lot of intuitive ideas, effective guidance, helpful suggestions, and personal encouragement. I feel like I am working in a big family and everybody in the family is trying their best to help me succeed.

I also greatly benefited from my industrial mentors and colleagues at Texas Instruments (TI) and Philips. Dr. Ananta Majhi at Philips Research did a huge amount of work to apply my research to a real industrial world. The results that he got for me gave me much useful feedback and were extremely important in helping determine our research direction. Hari Balachandran at TI did a lot of work to push my research into industrial production, and helped me get a great opportunity to work as a TIer. I had a wonderful experience at TI working with my supervisor Neil Simpson, colleagues Divya Reddy and Anthony Moore. They gave me more than two years of continuous support, and finally our work was recognized in both academia and industry.

Thanks to my team members: Jing Wang, Lei Wu, Ziding Yue, Hoki Kim, Zoran Stanojevic, Sagar S. Sabade, Ajaykumar A. Thadhlani and Bin Xue. I learned a lot from them during these years. Especially I would like to thank Ms. Jing Wang. She has contributed a huge amount of time and effort to the project, and solved several very critical problems.

My research was funded in part by SRC grant 000-TJ-844 and NSF grant CCR-0098329. I thank these sponsors for providing financial support.

Finally, I would like to thank my wife, Miao-Miao, my parents, and grandmother for their constant encouragement and support. Their love gave me endless confidence to go through hard times during my study. It is my honor to be loved in such a family.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

Page

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1 Delay Testing

Speed is crucial for high performance semiconductor products such as microprocessors. Due to many types of defects, integrated circuit (IC) fabrication processes produce defective ICs as well as good parts. Electronic testing screens out defective parts and guarantees the quality of shipped parts. Most defects are gross defects that cause errors at any speed (functional failure). However, some small manufacturing defects do not cause functional failure but affect speed. Delay testing detects such defects and ensures that the design meets the desired performance specifications. At smaller geometries, the number of timing-related defects is growing [1], and the effectiveness of functional and I_{DDQ} testing, which can detect some timing-related defects [2][3], is reduced. Functional testing is less effective because the complexity of functional test generation grows exponentially with increasing gate counts [4][5]. I_{DDQ} testing is less effective because a defective device current is difficult to distinguish from the normal quiescent current for 130 nm and smaller technologies [6]. Previous research has shown that 500 defective parts per million (DPPM) would not be achievable without delay testing [7]. Another study has shown that if at-speed tests were removed from the test program for a 180 nm microprocessor design, the escape rate went up nearly 3% [8]. Murphy and ELF35 experiments has also shown that 3 out of 116

This dissertation follows the style and format of *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*

defective parts escaped when tested at a slower than functional speed at normal operating voltage [9]. The International Technology Roadmap for Semiconductors (ITRS) [10] projects at-speed delay testing as an increasingly important but difficult problem. Rising clock frequencies and the increasing influence of interconnect on circuit delays are making traditional functional and delay test approaches inadequate.

Process variation can also cause timing-related faults (delay faults). Process variations are small natural variations in physical parameters from unit to unit [7][11]. These variations are caused by processing and mask imperfections and various wear-out mechanisms [12]. Process variation can be further divided into inter-die variations and intra-die variations. Inter-die variations are independent of the design implementation, and are considered globally in die-to-die, wafer-to-wafer and lot-to-lot areas [13]. Intra-die variations are dependent of design implementation and are considered locally [12].

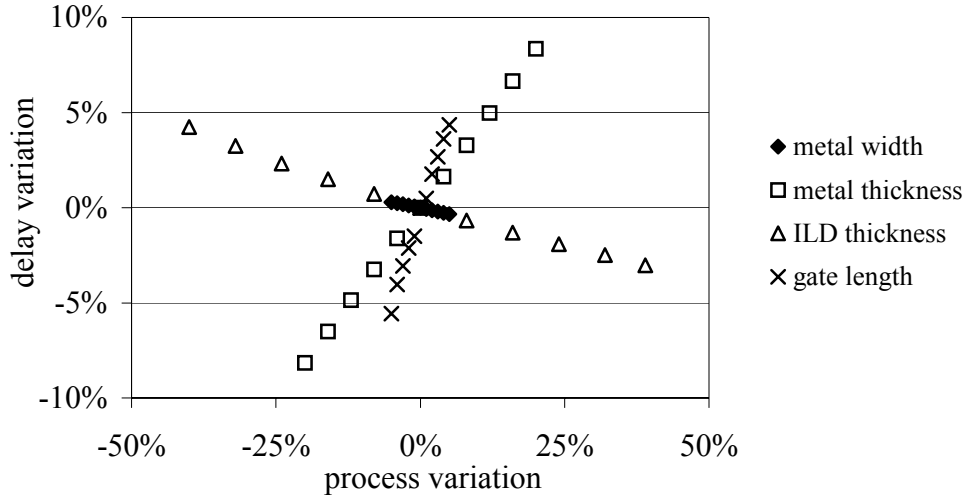


Figure 1. Delay variation vs. process variation.

Circuit delays are highly dependent on process variations because interconnect delays

are dependent on metal width, metal thickness, ILD thickness, via resistance, etc. and device delays are dependent on gate length, gate oxide thickness, etc. [14][15][16][17] Figure 1 shows the SPICE simulation results [12]. Due to process variation, some ICs without defects may also be slower than the specified speed. Delay testing must screen out these devices as well as the defective ones.

Typical delay testing is performed as follows. Each delay test pattern contains two test vectors. The first vector initializes the circuit under test (CUT) and the second vector tries to make transitions at the locations where delay faults are suspected, and propagate the slow transitions to observable primary outputs or memory cells. Logic values are measured in specified time and compared to correct values. A faulty circuit may pass a slow speed test but fail a high-speed test.

1.2 Traditional Delay Fault Models

A fault model is an abstraction of a type of defect behavior. A good fault model must be simple as well as accurate so that it can be efficiently used in simulation, test generation and diagnosis. Three classic delay fault models are commonly used in delay testing: transition fault model [18], gate delay fault model [19] and path delay fault model [20].

1.2.1 Transition Fault Model

The transition fault model assumes that the delay fault affects only one gate or line in the circuit under test, and the extra delay caused by the fault is large enough to prevent the transition from reaching any observable primary outputs or memory cells within the

specified time. In other words, a transition fault can be detected on any sensitized path through the fault site. The transition fault coverage is measured as the percentage of faults that are detected by a test set [21]. This model is simple and it can be applied without any accurate timing models. A transition fault test can be composed by pairing stuck-at-0 and stuck-at-1 test patterns [22] and test generation for complicated sequential circuits has been extensively investigated [23][24][25] for many years. The traditional transition fault test generation algorithm normally selects the easiest path, in terms of high controllability and observability, to propagate a transition, so that the automatic test pattern generation (ATPG) cost is close to minimum. As a result, in most cases the selected paths are short. Furthermore, short paths indicate fewer number of side input constraints, so that the average care bit density of the pattern that tests one fault is low, thus high test compaction rate and low test volume can be achieved [26][27][28][29].

However, recent research shows that resistive opens are one of the major defect types that cause delay faults [30], and that small delay faults cannot be neglected in new technology [31]. As the transition fault model targets relatively large delay faults, its test quality for small delay faults is a concern [32][33].

1.2.2 Gate Delay Fault Model

The gate delay fault model is more general than the transition fault model because it considers the amount of extra delay due to a defect. Therefore, a gate delay fault may only be detected through a long path. This model requires an accurate timing model. The quality of a test set is defined as how close the minimum actually detected delay fault sizes are to the minimum possibly detectable fault sizes [34]. However, the actual

minimum detectable fault size is always larger because of process variation, and it varies from device to device. The gate delay fault model always gives pessimistic fault coverage, and it gets worse when process variation increases with newer technology [11].

1.2.3 Path Delay Fault Model

The path delay fault model is the most conservative of any of the classical models for delay faults because a circuit is considered faulty if the delay of any of its paths exceeds the specified time. The path delay fault model is more realistic in modeling physical delay defects because it can also detect small distributed delay defects caused by process variation, or the combination of local and distributed delay. However, a major limitation of this model is that the number of paths in the circuit (and therefore the number of path delay faults) can be exponential in the number of gates. For example, ISCAS85 benchmark circuit c6288, a 16-bit multiplier, has close to 10^{20} paths.

Many techniques have been used to reduce the number of paths that must be tested in the path delay fault model. The simplest idea is to test the paths with maximum delays in a circuit. These paths are called the longest paths or critical paths. However, circuit optimization tends to compress the distribution of path delays in a circuit, so many paths are close to the maximum delay [35]. Because of manufacturing process variation, any of these paths can be the actual longest path. Therefore, a group of longest paths must be selected for testing. In practice, the path selection criteria can be based on if the nominal path delay is more than a certain threshold, e.g. 80% of the maximum specified delay of the circuit. It is assumed that the delays of the selected paths bound the maximum circuit delay with high confidence. The path selection is much more realistic if structural and

spatial correlations of path delays are used [36], and then the number of paths that must be tested can be significantly reduced. However, delay faults can be caused by both local small defects, such as a resistive short or via, and process variations during IC manufacturing [37], or their combination. The delay faults caused by local small defects are termed local delay faults [38] and those caused by process variations are termed global delay faults [36] or distributed path delay faults [39]. This path selection strategy assumes that delay faults are only caused by global delay faults. Thus if no path through a local delay fault site is selected for testing, the delay fault cannot be detected. To solve this problem, some path selection methods ensure that for every line in the circuit, the longest path through it must be selected [40][41][42][43].

However, the test coverage is unrealistically low using traditional path delay fault coverage metrics due to the fact that only a small subset of paths is tested using any realistic test sets, though some test sets obviously have much higher test quality than others.

1.3 False Path Problem

No matter which delay fault model is used, a delay fault is always tested by propagating a transition through a path or a set of paths and observing at the destination of the paths. However, due to logic dependence, not all paths can propagate transitions. These paths are called false paths or untestable paths. For example, in Figure 2, path *a*-*c*-*d* is a false path under the single-path sensitization criterion [44][45][46][47], because to propagate a transition through the AND gate requires line *b* to be logic 1 and to

propagate the transition through the OR gate requires line *b* to be logic 0. In this work, the terms “untestable” and “false” are used interchangeably. In fact, most paths are untestable. For example, more than 99% of the paths in ISCAS85 circuit c6288 have been proven untestable [48]. False path problem is the major problem in delay testing ATPG. Earlier approaches are inefficient in testable path generation because they get a list of candidate structural paths from timing analysis tools and check the testability for each path, and most of the paths are untestable. In this work, an efficient methodology is developed and the testability of each generated path is guaranteed.

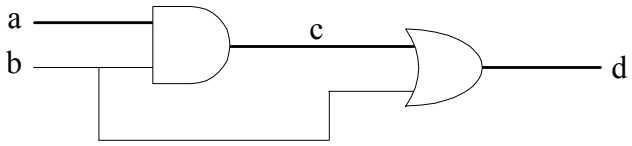


Figure 2. An example of a false path.

1.4 Scan-Based At-Speed Test Approaches

Test speed is another challenge of delay testing. To detect small defects, and sometimes the combination of small defects and slow paths due to process variation, high-test speed is required. Applying a test at the CUT functional speed is called at-speed test. Build-in-self-test (BIST) can be applied at full functional speed, but it has low probability to sensitize enough critical paths, though evidence has been shown that BIST can achieve very high fault coverage for stuck-at and transition faults [49]. On the other hand, functional tests running at full speed are becoming unattractive due to the high cost of development and application [50]. AC scan test using automatic test

equipment (ATE) can solve these problems, though the pin interface speed of a low-cost ATE is normally much lower than the functional speed of the CUT. Fortunately, as high-speed on-chip clock generation using phase-locked loops (PLL), is becoming common in industry, performing an at-speed delay test using low speed ATE channels will no longer be a bottleneck in the near future.

In this work, muxed scan design is assumed, with a scan enable signal selecting either serial scan data or functional logic data. The flip-flops are clocked with the system clock. Two scan-based at-speed test methodologies have found increasing usage in industry.

1.4.1 Launch-on-Shift (Skewed Load)

The procedure for the launch-on-shift (or skewed load [51][52]) test approach is:

1. The circuit is set to scan mode. The first test vector is scanned into scan chains using the slow scan clock, and the values are set on primary inputs (PIs).

2. The second test vector is obtained by shifting the scan chain by one bit. Usually the PIs do not change values due to the constraints from low-cost ATEs.

3. The circuit is set to the functional mode by flipping the scan-enable signals and pulsing the system clock to capture the circuit values in scan cells. The values on primary outputs (POs) are captured if necessary.

4. The circuit is set to scan mode and the values in the scan chains are scanned out using the slow scan clock. This step can be overlapped with step 1.

The advantage of this approach is that fast test generation methodologies for combinational circuits can be applied without many modifications. Scanned flip-flops are considered primary inputs in the ATPG for combinational circuits, and the adjacent

scan bit dependencies must be added to the existing ATPG. These constraints may result in some paths being untestable.

The disadvantage of this approach is that the scan enable signals must operate at full speed. In addition, many of the sensitizable paths under the launch-on-shift constraints are sequential false paths, i.e. these paths are not sensitizable in functional mode, so some redundant faults would be detected.

1.4.2 Launch-on-Capture (Functional Justification)

The procedure of the launch-on-capture (or functional justification, broadside [53]) test approach is:

1. Same as the launch-on-shift approach step 1.
2. The circuit is set to functional mode. A dummy cycle is inserted if the scan-enable signal cannot operate at full speed or the system clock frequency is very high, so that the launch clock pulse width is too large. Figure 3(a) shows the clock waveform. For comparison, Figure 3(b) shows the clock waveform if the time is sufficient for the scan enable signal to propagate. In this approach, the launch cycle is kept identical to the shift cycle with respect to period, rising edge, and pulse width.
3. The system clock is pulsed twice. At the first clock pulse, the second test vector is derived from the first vector. At the second clock pulse, the test is performed and the output values are captured in scan cells. The values on POs are captured if necessary.
4. Same as the launch-on-shift approach step 4.

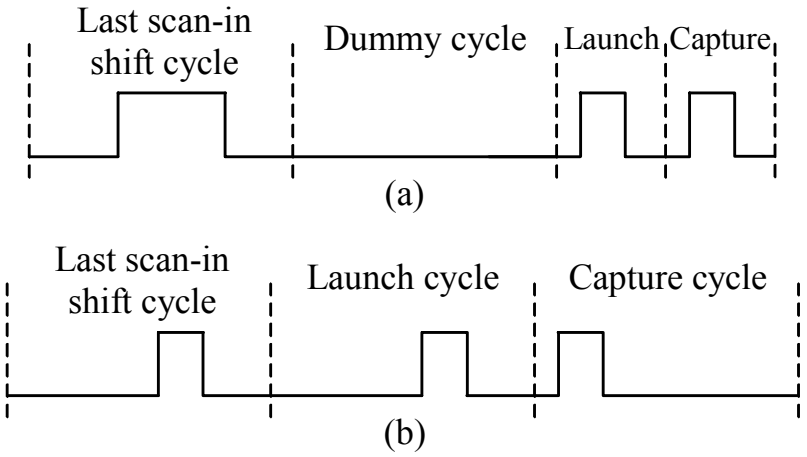


Figure 3. Launch-on-capture clock waveforms.

The advantage of this approach is that it does not require the scan enable signal to operate at full speed. The sensitizable paths under the launch-on-capture constraints are also sensitizable in functional mode, unless the first vector represents an illegal state.

1.5 Combined Delay Fault Model – A More Realistic Model

The traditional delay fault models do not completely describe all realistic fault behaviors, and the test quality of the test patterns based on these models is a concern. An experiment has shown that test quality varies from pattern to pattern even within a single transition fault test set, and it is reasonable to expect more variation among different test sets. In this experiment, the transition test patterns are divided evenly into 10 groups by their original order, and speed binning is run to get the maximum passing frequency (Fmax) for each group on the same device. Figure 4 shows the results. A 4% Fmax variation is observed. Higher variation is expected if the test set is divided into more groups, or more test sets are included into the experiment. In theory, a transition fault

test set results in close-to-upper-bound Fmax, and it is interesting to find a test that results in the lower bound. To find such a test, a more realistic delay fault model, which more accurately describes fault behaviors, is developed in this work.

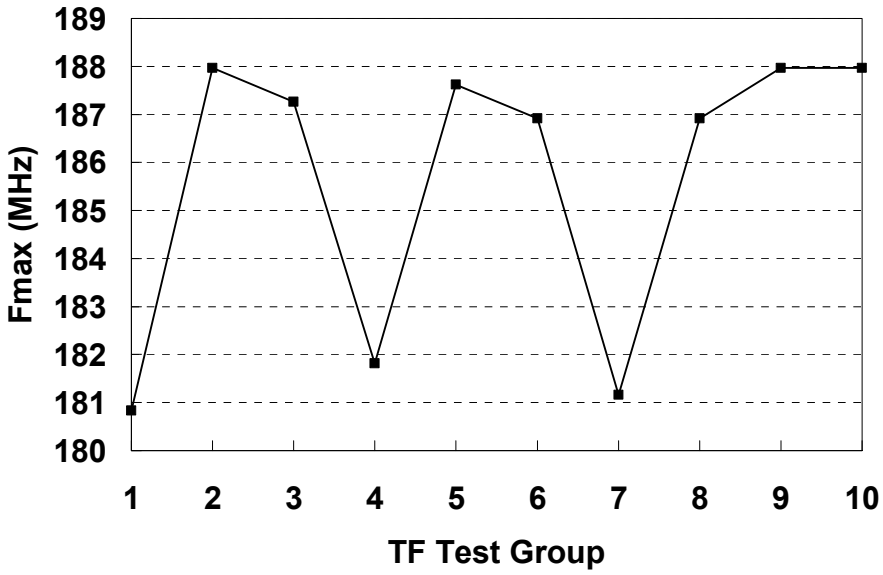


Figure 4. Fmax varies with delay test patterns.

Figure 5 shows a comprehensive fault mechanism. Spot defects and parametric process variation can cause functional failures, delay faults, or reliability hazards. A local delay fault is a local delay increase caused by a spot defect, such as a resistive open or short. The gate or transition fault model targets these faults. Global delay faults are slow paths due to process parameter variation, such as transistor gate length variation. The path delay fault model targets these faults. Combined delay faults (CDF) are delay faults caused by a combination of spot defect and process variation. By considering the entire range of spot defect parameters and process variation, the CDF model encompasses both local and global delay faults. At this time, delays due to capacitive

and inductive coupling [54][55][56][57] are not considered.

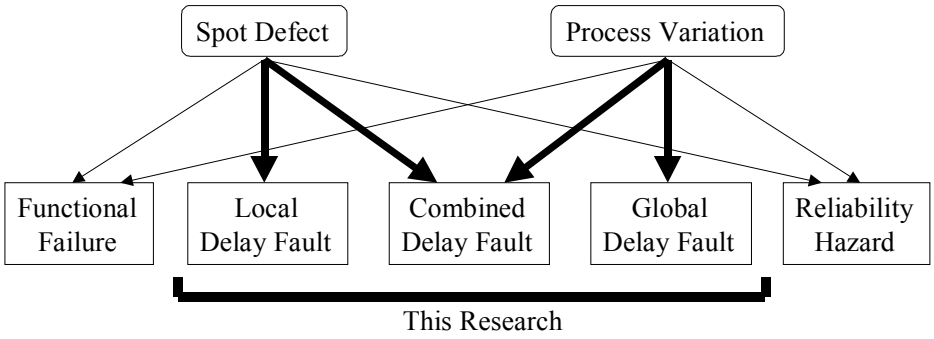


Figure 5. Fault types addressed in this work.

1.6 Organization

In section 2, statistical delay fault coverage metric is proposed for the CDF model, to more accurately evaluate the test quality of traditional and proposed delay test methodologies. The metric is based on fault simulation that is presented in section 3. To get higher quality tests based on the CDF model, in section 4 an ATPG algorithm is presented and applied to benchmark circuits and industrial designs. In section 5, silicon data shows the accuracy of the new simulation results, and significant test quality increase using the new test generation methodologies. Section 6 concludes the dissertation with future directions.

2. STATISTICAL FAULT COVERAGE METRIC

2.1 Background

The traditional metric of path delay fault coverage is the percentage of the paths which are tested under robust [20], non-robust [44], or functional [58] sensitization criteria, that is, coverage = number of tested paths / total number of structural paths. A structural path is a sequence of gates and nets without considering sensitization. Based on this metric, testing *p* long paths has the same fault coverage as testing *p* short paths, which does not reflect the real test quality. In addition, since the total number of structural paths is exponential in the number of gates, clearly this fault coverage metric results in very low fault coverage for any practical test set, which is far from the reality. Some research eliminated untestable paths [59][60], and then the coverage = number of tested paths / number of total testable paths. However, these methods are very expensive because the sensitization of all the paths must be checked and the coverage is still unrealistically low (around 20%). For example, as ISCAS85 benchmark circuit c6288 may have an exponential number of testable paths [48], the traditional path delay fault coverage of any practical test is close to zero.

2.2 Path Delay Correlation

Any two paths in a circuit have correlation in delays. Two paths have structural correlation when they share a common path segment. For example, in Figure 6 path *a*-*d*-*e* and *b*-*d*-*e* are structurally correlated because they share segment *d*-*e*. Two paths can

also have spatial correlation because the path delays are functions of the manufacturing process parameters, such as transistor gate length, which are spatially correlated. For two paths that are physically close to each other, the delay correlation is high because the paths have very similar process parameters.

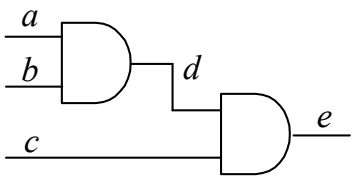


Figure 6. An example of structurally-correlated paths.

Figure 7 shows the delay space [39] for two paths, assuming the path delay is a one-dimensional function of process parameters. The delay space is the bounded region in which the probable delay value combinations are represented. It is assumed that each path has min-max delays. If the two paths have no correlation, the delay value combination can be anywhere within the rectangle. If they are perfectly correlated, the delay space shrinks to a line, which means if path 1 has the max (min) delay under a combination of certain process parameters; path 2 also reaches its max (min) delay under the same combination of process parameters. In reality, the correlation is somewhere in between, and the realistic delay space is the shaded area. Using correlation information, the delays of untested paths can be predicted by the delays of tested paths [61].

An inter-die process variation model [62] is used in this work. In this model, the delay of a path is expressed as a linear function of process variables. If the delay of path p_1 is less than that of path p_2 under any process parameter combination, it is said that p_1

is covered by p_2. Thus, if p_2 is tested, p_1 does not have to be tested. One limitation of this model is that it does not consider intra-die process variation. However, by assuming 100% intra-die process correlation, the upper bound of delay fault coverage can be computed, as shown in the next section.

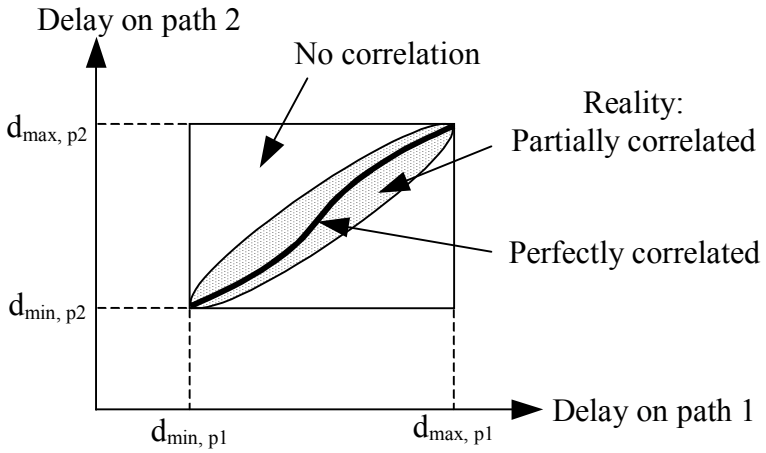


Figure 7. Delay spaces for different path correlations.

2.3 Delay Fault Coverage Metric

Realistic delay fault coverage can be computed as the percentage of faulty chips that can be detected as faulty by a test set [39][63]. If presented in a probability formula, the coverage for test set *t* is:

P(*t* detects delay fault | chip has a delay fault) (1)

Because this is a general metric and model-independent, to make it usable, it is necessary to map this abstract metric to CDF model, which considers both local and global delay faults.

In the CDF model it is assumed that there is at most one local delay fault (or no local delay fault) that can occur on any single line in the circuit. The position of the local delay fault is termed as fault site. The whole circuit is also subject to process variation, which may cause a timing failure by itself (global delay fault) or in combination with a local delay fault.

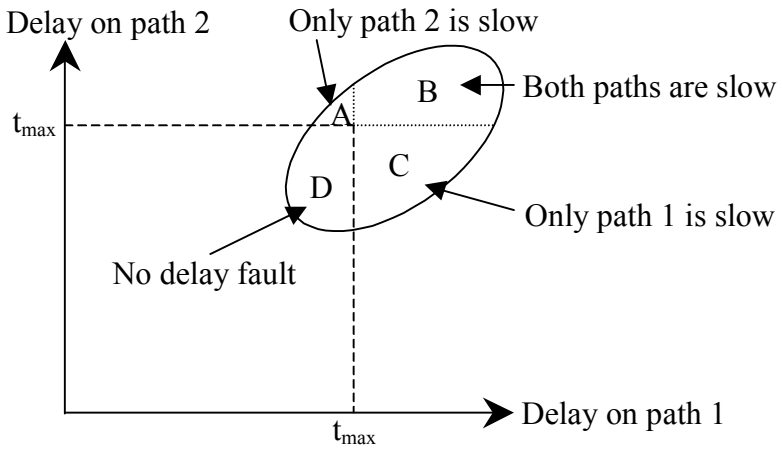


Figure 8. Delay space of a fault.

Under the CDF model, fault detection is probabilistic instead of deterministic. For example, suppose there are two paths, P_1 and P_2, through a resistive open fault site, and the local extra delay is not large enough for either path to be definitely slow. Figure 8 shows the delay space for this fault. t_{max} is the maximum specified delay of the circuit. The circuit has some probability that path 1 or 2 is slow (Delay < t_{min} is not considered in this work). Suppose test set t_1 tests path 1 only and test set t_2 tests path 2 only. Neither t_1 nor t_2 can guarantee the detection of the fault, e.g. t_1 cannot detect the delay fault in area *A*. Instead, each test set only has some probability of detection. Both test sets are required to guarantee detection. In this work the notion of detection probability (DP) [64]

for a single fault site is used.

Using this probability model, the general metric expressed in formula (1) can be translated into formula (2) to compute the DP for fault site i with local extra delay Δ (the size of the local delay fault):

$$\mathbf{DP_{i,\Delta}(t) = P(\geq 1 \text{ tested path through } i \text{ is slow} \mid \geq 1 \text{ path through } i \text{ is slow})} \quad (2)$$

In the example whose delay space is shown in Figure 8, according to formula (2), if test set t tests path 1 only, the DP is area(B∪C)/area(A∪B∪C); if t tests path 2 only, the DP is area(B∪A)/area(A∪B∪C); and if t tests both paths, the DP is 100%.

The above analysis is for a given local extra delay Δ. For fault site i with an arbitrary Δ, the DP for site i is computed as:

$$\mathrm{DP}_i(\boldsymbol{t}) = \int_{\Delta > \Delta_{0,i}} \mathrm{DP}_{i,\Delta}(\boldsymbol{t}) \cdot \mathrm{p}_i(\boldsymbol{\Delta}) \mathrm{d}\boldsymbol{\Delta} \quad (3)$$

where $\Delta_{0,i}$ is the value of local extra delay below which there is no delay fault. $p_i(\Delta)$ is the PDF of Δ at fault site i, and is computed using the PDF of delay caused by physical defects, such as resistive opens [31] or shorts [65][66][67].

The overall fault coverage for test set t is:

$$\mathrm{FC}(\boldsymbol{t}) = \sum_i \mathrm{DP}_i(\boldsymbol{t}) \cdot \boldsymbol{w_i} \times 100\% \quad (4)$$

where w_i is the weight for fault site i ($\sum_i w_i = 1$). w_i depends on the location of the fault. For example, the fault sites with many long paths through them are more likely to cause delay faults than the fault sites that have only short paths through them. Therefore, testing more paths through a high weighted fault site is an efficient way to increase the fault coverage. w_i is also sensitive to the ratio of local/global delay faults. If the ratio is

high, the weights are almost equal for all fault sites. If it is low, the fault sites with only short paths can have weights close to zero. In this work, equal weights are used for simplicity.

If no local delay fault is considered, only formula (2) is used in the computation, with Δ=0, and i is removed because the whole circuit, instead of a particular site, is considered.

According to formula (2), if the path delays are not independent (and in reality, they are not), the DP computation is dependent on the delay space. For example, in Figure 8, the areas of A, B, C change if the delay space changes, and then the DP changes accordingly. Therefore, if accurate correlation information is not known, the DP computation is not easy. To solve this problem, two extremities are assumed. If no correlation is assumed, path delays are independent variables. This assumption results in the lower bound of fault coverage. If 100% intra-die process correlation is assumed (only inter-die process variation is considered) [62], the upper bound of fault coverage is computed.

Applying this coverage metric (formulae 2-4) is inexpensive because only a small subset of paths must be considered. For example, Figure 9 shows the delays of four paths, each having a distribution due to process variation, through a certain fault site. Suppose path P_1 is tested by t, and the longest testable path P_0 is not tested. When $\Delta_0<\Delta<\Delta_1$, $DP_{i,\Delta}(t)$ is 0; when $\Delta>\Delta_2$, $DP_{i,\Delta}(t)$ is 100%, because the tested path P_1 is definitely slow; when $\Delta_1<\Delta<\Delta_2$, $DP_{i,\Delta}(t)$ increases from 0 to 100% as Δ increases. Thus, the fault coverage computation is required only in this interval. The main cost to compute the

fault coverage (or test efficiency, which is a more accurate fault coverage because it excludes undetectable faults), which is the number of tested faults over the number of testable faults, is on the sensitization check for all the paths whose length is within this interval. However, if all the structural paths are assumed testable, a lower bound coverage can be computed and experiments show the error on ISCAS85 circuits is <4%. The cost of enumerating structural paths is low because the total number of structural paths through a gate can be computed from the number of paths of its immediate fan-in and fan-out gates.

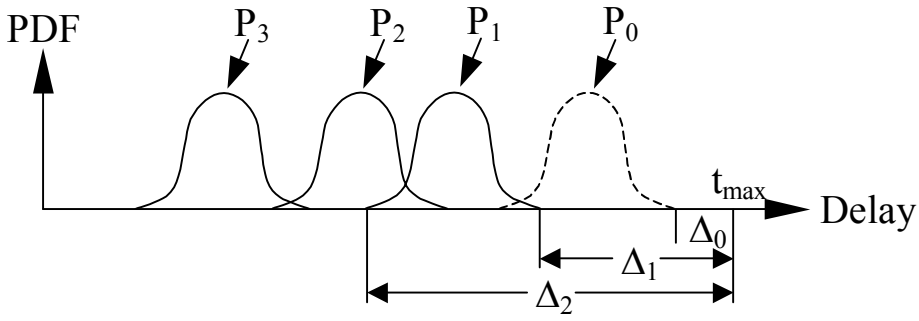


Figure 9. Fault coverage computation.

The fault coverage metric suggests a test strategy:

1. Apply transition fault tests to detect large local delay faults, and from industrial experience, most local delay faults are large.

2. Apply at-speed test to one of the longest paths, e.g. the path with maximum nominal delay, through each gate or line, to eliminate or reduce the 0-DP area between Δ_0 and Δ_1 in Figure 9, because this is the second largest coverage loss factor.

3. Test more possible longest paths (such as P_2 in Figure 9, if P_0 does not exist) to

increase the DP between Δ_1 and Δ_2.

Figure 10 shows the conceptual relation between fault coverage and the percentage of tested paths, using different fault coverage metrics. The paths are sorted by their nominal delays in descending order. If there is no local delay fault, the fault coverage increases quickly after the first several potentially critical paths are tested, and reaches 100% after all potentially critical paths are tested. If the percentage of local delay faults is high (in reality it is), the curves have some "jumps" because at these points the first path through some fault sites is tested. It is clear that the new fault coverage metric is closer to industrial experience and more realistic. The traditional fault coverage is computed as the number of tested paths over the total number of testable paths.

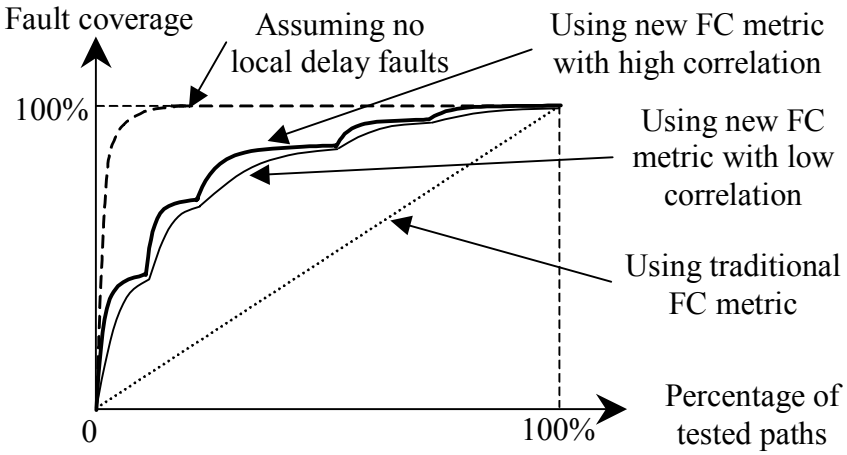


Figure 10. Fault coverage vs. percentage of tested paths.

3. FAULT SIMULATION

The goal of the CDF simulation is to compute the detection probability for each fault site, for test set *t*. The DP's for all the fault sites are used to compute the overall fault coverage, so that the quality of the test set is evaluated [68][69].

3.1 Fast Fault Simulation Algorithm

The fault simulation starts with spot-defect-free timing simulation for each test pattern. After this simulation, the initial and final logic values and the nominal transition time of the last event for each line are known. Figure 11 shows an example. The italic numbers next to the transition symbols indicate the transition time, assuming the unit gate delay model is used. S1 or S0 indicates a stable logic value 1 or 0 on the line.

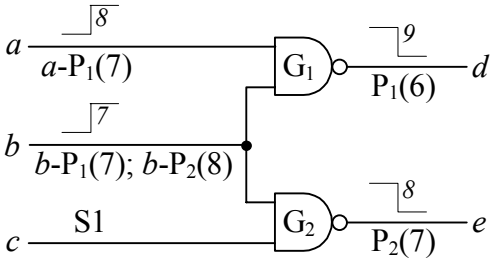


Figure 11. Robust propagation path identification.

Then the robust/non-robust propagation paths from each line to primary outputs or scan cells are identified. A line's robust propagation paths can be computed using its immediate fan-out lines' robust propagation paths. In Figure 11, suppose line *d* has a robust propagation path P_1 with length 6, and line *e* has path P_2 with length 7. The robust propagation paths for line *b* are computed by checking the final logic values on the side

inputs of gate G_1 and G_2. Then two paths are identified: b-P_1 with length 7 and b-P_2 with length 8. Because the propagation paths are robust, the slow signal is able to propagate through these paths independent of the delays on the side inputs to the paths. Therefore the extra delay Δ on a line must be detected if $t_{trans} + \Delta + l_{prop} > t_{max}$, where t_{trans} and l_{prop} are the transition time and propagation path length associated with that line, respectively. In the simulation, since t_{trans} and l_{prop} are statistical values (with PDFs), the computed Δ is a statistical value too. For resistive shorts, the sensitization condition, i.e. the opposite logic value on the other shorted line, must be checked.

The non-robust propagation paths can be identified in a similar way. The difference is that if there is no transition on a line, the non-robust propagation paths from that line must also be computed. Line g in Figure 12 is an example. The reason is that a spot defect on line d may generate a glitch on g, and the computation of the non-robust propagation paths from d uses g's propagation paths. The complexity of the spot-defect-free simulation is $O(V \cdot C)$, where V is the number of vectors and C is the circuit size (number of gates/lines in the circuit).

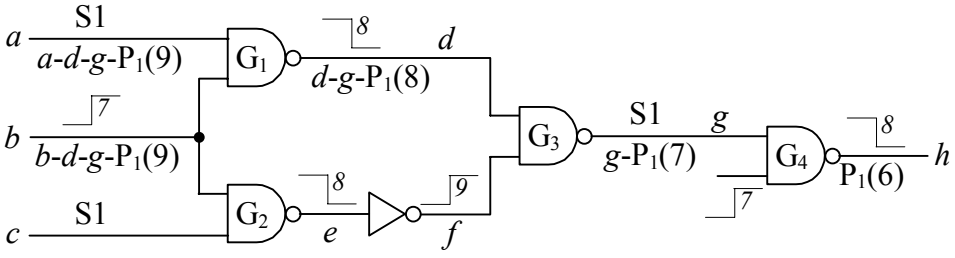


Figure 12. Non-robust propagation path identification.

A problem with the non-robust propagation paths is that the fault detection through

these paths is dependent on the delays on the side inputs to the path. In Figure 12, an extra delay on line *b* does not affect the transition time on line *h*, even though line *b* has a non-robust propagation path. Therefore, the validation of these paths must be checked. After spot-defect-free timing simulation, each line has a few non-robust propagation paths. The validation check can be performed by introducing an extra delay Δ on the line, where $\Delta = t_{max} - t_{trans} - l_{prop}$, and running fault simulation for the patterns which sensitize this path, to check if the slow signal can be detected at any primary output or scan cell. This procedure starts from the smallest Δ. If a small Δ can be detected through a non-robust propagation path, the validation check for the paths that can only detect large Δ is not necessary, because testing those paths, such as P_3 in Figure 9, does not increase the fault coverage. Experiments show that normally only a few paths must be checked for each line.

It is possible that some functional sensitizable paths are missed. However, because these paths always appear in pairs and the delay is determined by the shorter one, in most cases they do not contribute to the fault coverage. Thus, these paths are not checked unless there is no long robust or non-robust propagation path through the fault site.

3.2 Comparison of Simulated Fault Coverage

ISCAS85 benchmark circuits are used in the experiments. The TSMC 180 nm technology with five metal layers is used to generate the layouts. Delays for each gate and interconnect are extracted and assumed to have a Normal distribution with 3σ=10%

of the nominal value. The transistor gate length, the width and thickness of the five metal layers and the thickness of the five interlayer dielectrics are considered parameters in the process variation model [62]. The maximum specified delay t_{max} is set to be 8% longer than the nominal delay of the longest testable path.

In the first experiment, resistive opens are assumed on gate outputs. It is assumed that 80% of the opens have infinite resistance so that they can be detected by a transition fault test, and the remaining 20% are resistive, with log(R) uniformly distributed, where R is the open resistance [31]. This case is similar to the traditional gate delay fault model because the local delay fault size has a distribution. The delay fault coverage computed in this work is more accurate than the gate delay fault coverage, because process variation is also considered. The KLPG (K longest testable paths per gate) test generator [70] is used to generate K=500 testable paths through each gate (this process takes <10 minutes), for the fault coverage computation purpose. For >99% of the gates, K=500 covers all the possible longest testable paths through the gate, assuming a ±10% path delay variation. This increases fault coverage accuracy, because the fault sites with no transition fault test are not included in the computation, and the false paths are eliminated by the KLPG test generator.

Table 1 shows the fault coverage for the ISCAS85 circuits, using three test sets:

1. Transition fault test;
2. KLPG-1 test, which tests two longest paths through each gate, with one path having a rising transition and the other having a falling transition at the gate output;
3. (Critical) path delay test, which tests the C_{size} longest testable paths throughout

the circuit, where C_{size} is the circuit size (number of gates in the circuit), so that the number of patterns in this test set is about the same as the KLPG-1 test set.

The transition fault test is generated by a commercial ATPG tool. The other two test sets are generated by the KLPG test generator, and compressed by a simple greedy algorithm. All three test sets are applied at full functional speed.

Table 1. Fault coverage comparison for resistive opens and process variation.

Circuit	TF	KLPG-1		Critical	TF+C
	UB(%)	LB(%)	UB(%)	UB(%)	UB(%)
c432	98.85	99.64	99.98	57.78	99.51
c499	98.68	99.59	99.96	25.95	99.27
c880	98.82	99.61	99.95	24.38	99.25
c1355	97.23	99.53	99.89	18.80	98.01
c1908	98.51	99.44	99.92	31.99	99.11
c2670	98.69	99.49	99.78	15.35	98.94
c3540	97.97	99.51	99.90	25.66	98.60
c5315	99.03	99.50	99.83	10.68	99.18
c6288	97.51	99.12	99.65	12.83	98.14
c7552	98.96	99.61	99.93	28.50	99.35

Column 2 shows the fault coverage for the transition fault test. For most fault sites, the transition fault test does not test through the longest paths, but the fault coverage is still reasonably high. The reason is that, based on test structure data, 80% of the resistive opens cause large extra delay if the test is applied at full functional speed. These numbers reflect the reality that a transition fault test detects most of delay faults. It should be noted that these numbers are the upper bound because it is too expensive for

the KLPG test generator to generate all the paths whose length is close to the paths that the transition fault test sensitizes (since most of these paths are short, and the KLPG test generator generates long paths first). This also happens to the path delay test because for the faults that are not on critical paths, random short paths are sensitized by this test set. Therefore, a 100% path delay correlation is used to compute the upper bound.

Columns 3 and 4 show the lower and upper bounds of the fault coverage for the KLPG-1 test. The lower bound is computed by assuming no path delay correlation (even no structural correlation). The upper bound is computed by using an inter-die process variation model [62] and assuming 100% intra-die process correlation. The bounds are close because the majority of delay faults caused by resistive opens can be modeled as transition faults. It can be seen that the upper bound fault coverage is almost 100% for most circuits. The reason is that for most fault sites, only 2-3 paths can be the longest paths with process correlation. Thus, most fault sites have 100% DP. The fault coverage upper bound for circuits c2670, c5315 and c6288 is lower than the other circuits because the number of longest paths per fault site for these circuits is relatively large even with process correlation. Column 5 shows the upper bound of the fault coverage for the path delay test. The coverage loss mainly comes from the fact that many local delay fault sites have no test, because they are not on a long path. Column 6 shows the upper bound of the fault coverage for applying both transition fault and path delay tests, which are used in industry. The higher fault coverage of this test reflects industrial experience [71].

Figure 13 shows the fault coverage for circuit c7552, assuming the K value in the KLPG test increases from 1 to 5. As can be seen, only a small number of paths are

needed through each fault site to achieve high fault coverage. The benefit of testing one of the longest paths (the fault coverage increase from the transition fault test to the KLPG-1 test) is more significant than that of testing more long paths (the increase from the KLPG-1 test to the KLPG-5 test).

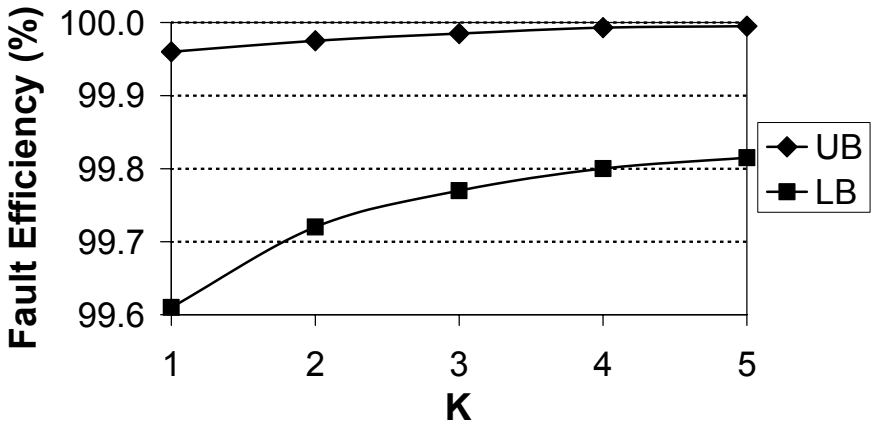


Figure 13. Fault coverage vs. K (circuit c7552).

In the second experiment, random non-feedback resistive shorts are assumed. Table 2 shows the results. The number of shorts is approximately twice the number of lines in the circuits. Shorts between lines feeding the same gate are not included. Shorts between signal lines and power/ground grid are not considered because they are more likely to behave as stuck-at or transition faults [72][73][74][75]. The bridge resistance is assumed to be uniformly distributed between 0 Ω and 40 kΩ [76]. It can be seen that for transition fault and path delay tests, there is a large coverage loss but KLPG-1 test keeps high fault coverage. It is straightforward that the path delay test has low coverage because some bridging faults on non-critical paths are not sensitized randomly. However, this should not happen for the transition fault test. Although the transition fault test does not target

resistive shorts explicitly, a bridging fault can be detected by either shorted line in most cases. In addition, as the transition fault test generates at least one observable transition through each gate/line, and the probability that an opposite logic value is set on the other shorted line is high, most bridging faults can be sensitized. However, there is still a significant difference of coverage between the KLPG-1 and transition fault tests.

Table 2. Fault coverage comparison for resistive shorts and process variation.

Circuit	# Shorts	TF	KLPG-1		Critical	TF+C
		UB(%)	LB(%)	UB(%)	UB(%)	UB(%)
c432	821	85.3	98.2	98.8	82.4	97.7
c499	1 102	91.1	99.1	99.3	88.0	98.5
c880	1 421	90.5	98.5	99.0	85.1	94.6
c1355	2 488	88.9	98.6	99.0	84.1	96.0
c1908	4 007	93.0	96.6	97.5	90.7	95.8
c3540	8 919	84.1	94.6	95.9	85.3	92.1
c5315	12 168	93.9	97.1	98.2	94.6	96.4
c6288	14 170	80.6	95.0	99.1	90.1	94.2
c7552	12 156	85.5	93.3	97.4	89.6	93.7

Figure 14 shows the delay caused by most resistive shorts and explains why testing long paths is important. For most shorts, the slack of the longest sensitizable path through either shorted line is not very tight (Figure 14a), therefore the coverage loss, which is the range between the two dotted vertical lines, is small. However, for some shorts with one shorted line on a critical path, a large coverage loss occurs if the longest sensitizable path through that line is not tested even if the tested path is slightly shorter (Figure 14b). This also explains the results that applying both transition fault and path

delay tests can get high fault coverage.

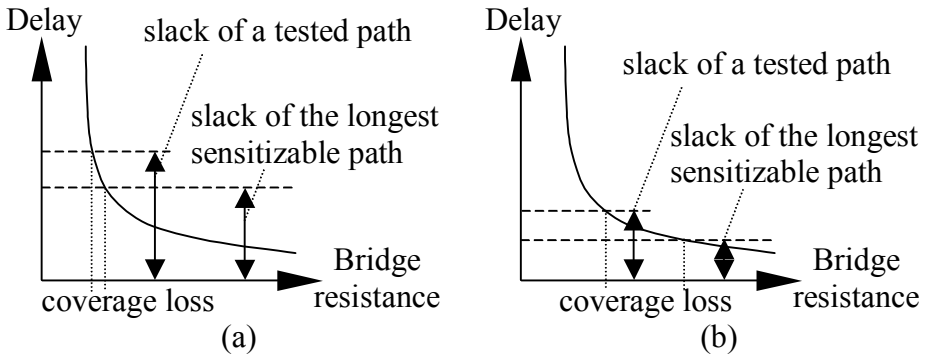


Figure 14. Delay vs. bridge resistance.

In the third experiment, no spot defects are assumed. The circuits are only subject to process variation. This case is equivalent to the traditional path delay fault model, and the new metric computes a more accurate and reasonable fault coverage. The KLPG test generator is used to generate all the potentially critical paths to compute more accurate fault coverage (the CPU time is <5 minutes per circuit).

Table 3 shows the fault coverage for this case, using the KLPG-1 and path delay tests. The fault coverage for the transition fault test is almost zero for all these circuits, since few critical paths are tested by luck because transition fault test normally sensitizes short paths to keep the test generation cost low. It can be seen that the KLPG-1 test results in high fault coverage. This is because many paths in this test set are also critical paths. After the process correlation is applied, most paths are trimmed and for most circuits there are <10 “must be tested” paths remaining (circuit c2670 has 14 paths remaining, which is the maximum number). The experiments show that the KLPG-1 test set covers all the “must be tested” paths for most circuits (KLPG-1 for c499 misses one path and

KLPG-1 for c2670 misses two). It is not surprising that the path delay test achieves higher fault coverage, and the number of critical paths in all the circuits is less than C_{size}, except for c6288.

Table 3. Fault coverage comparison for process variation only.

Circuit	KLPG-1		Critical	
	LB(%)	UB(%)	LB(%)	UB(%)
c432	100	100	100	100
c499	99.81	99.94	100	100
c880	99.83	100	100	100
c1355	99.72	100	100	100
c1908	99.77	100	100	100
c2670	99.25	99.75	100	100
c3540	100	100	100	100
c5315	99.17	100	100	100
c6288	96.44	100	98.89	100
c7552	100	100	100	100

For comparison, Table 4 shows the fault coverage for the three test sets using the traditional transition and path delay fault coverage metrics. It can be seen that the transition fault coverage does not reflect the real test quality since both transition fault and KLPG-1 tests have 100% fault coverage after removing undetectable faults. The traditional path delay fault coverage for the transition fault test is not included because the cost is too high to identify all the sensitized paths. The total number of testable paths, which is used in the traditional path delay fault coverage computation, is from the RESIST test generator [48]. As some aborted paths may also be testable, the numbers in

columns 5 and 6 are the upper bound. For example, in circuit c6288, 12 592 testable paths are found by RESIST and about 10^{18} paths are aborted. The traditional path delay fault coverage is much lower than the real test quality.

Table 4. Fault coverage comparison using traditional delay fault coverage metrics.

Circuit	Transition FC (%)			Path Delay FC (%)	
	TF	KLPG-1	Critical	KLPG-1	Critical
c432	100	100	57.81	3.01	4.30
c499	100	100	25.99	0.28	0.15
c880	100	100	24.41	2.19	2.38
c1355	100	100	18.86	1.94	2.41
c1908	100	100	32.04	0.37	0.90
c2670	100	100	15.48	4.44	8.34
c3540	100	100	25.73	1.01	1.89
c5315	100	100	10.79	1.71	2.83
c6288	100	100	13.08	11.69	19.19
c7552	100	100	28.55	1.82	4.07

4. TEST GENERATION

4.1 Background

The problem of finding the longest path through each gate or line in a circuit has been extensively studied [41][42][77][78]. However, these methodologies are either inefficient or fail to guarantee the testability of the generated paths. The inefficiency comes from the fact that most prior work lists many long structural paths first, and then checks their testability. If there are subcircuits with a large number of paths, all the long paths listed are similar and it is possible that none of them is testable.

Many ATPGs for the global longest path generation were studied to see if they could be extended to solve the problem of finding the K longest testable paths through each gate (KLPG). A fast ATPG tool NEST [79] generates paths in a nonenumerative way, which can handle a large number paths simultaneously, but it is only effective in highly testable circuits, where large numbers of path delay faults are testable. DYNAMITE [80] is very efficient in poorly testable circuits, but in highly testable circuits many faults are treated separately, which results in huge memory consumption and so is not practical for large circuits. RESIST [48] exploits the fact that many paths in a circuit have common subpaths and sensitizes those subpaths only once, which reduces repeated work and identifies large sets of untestable paths. Moreover, for the first time this research identified 99.4% of all the path delay faults as either testable or untestable in ISCAS benchmark circuit c6288, which is known for having an exponential number of paths. However, the test generation for c6288 is still slow. RESIST took 1 122 CPU hours to

find 12 592 paths on a SPARC IPX 28 MIPS machine. Following research [43] presented an efficient method to extend the RESIST algorithm to the problem of finding a set of longest testable paths that cover every gate. This method takes advantage of the relations of the longest paths through different gates, and guarantees their testability. However, partly because this work assumes a unit delay model, there is no obvious way to extend it to handle the problem of finding the K longest testable paths through each gate, and this method fails when applied to c6288, indicating that the advantages in the RESIST algorithm may not be easily applied to generating paths through a certain gate.

A timing analysis tool [81] presents another method to efficiently identify the global longest testable paths in a combinational circuit. Instead of generating many long structural paths and checking their testability, this tool grows paths from the primary inputs. In each iteration, a new gate is added, and the constraints are applied to that gate. Then instead of assigning logic values on one or more primary inputs to satisfy the constraints on the newly added gate, as done in VIPER [47], direct implications, which are more efficient, are applied to find local conflicts. If conflicts exist, the whole search space that contains the already-grown series of gates is trimmed off. This technique is called implicit false path elimination [45][82]. Some other false path elimination techniques, such as forward trimming and dynamic dominators, are also applied in this tool to identify false paths earlier. This tool is efficient and able to handle c6288.

The algorithm in this work inherits the framework of the efficient timing analysis tool but aims at particular gates one by one, and it has been extended to handle scan-based sequential circuits in industry. Many heuristics are implemented in the algorithm to

reduce the search space and avoid repeated work. Experiments have shown that this algorithm can achieve a linear CPU time and memory consumption in circuit size.

4.2 Test Generation Flow

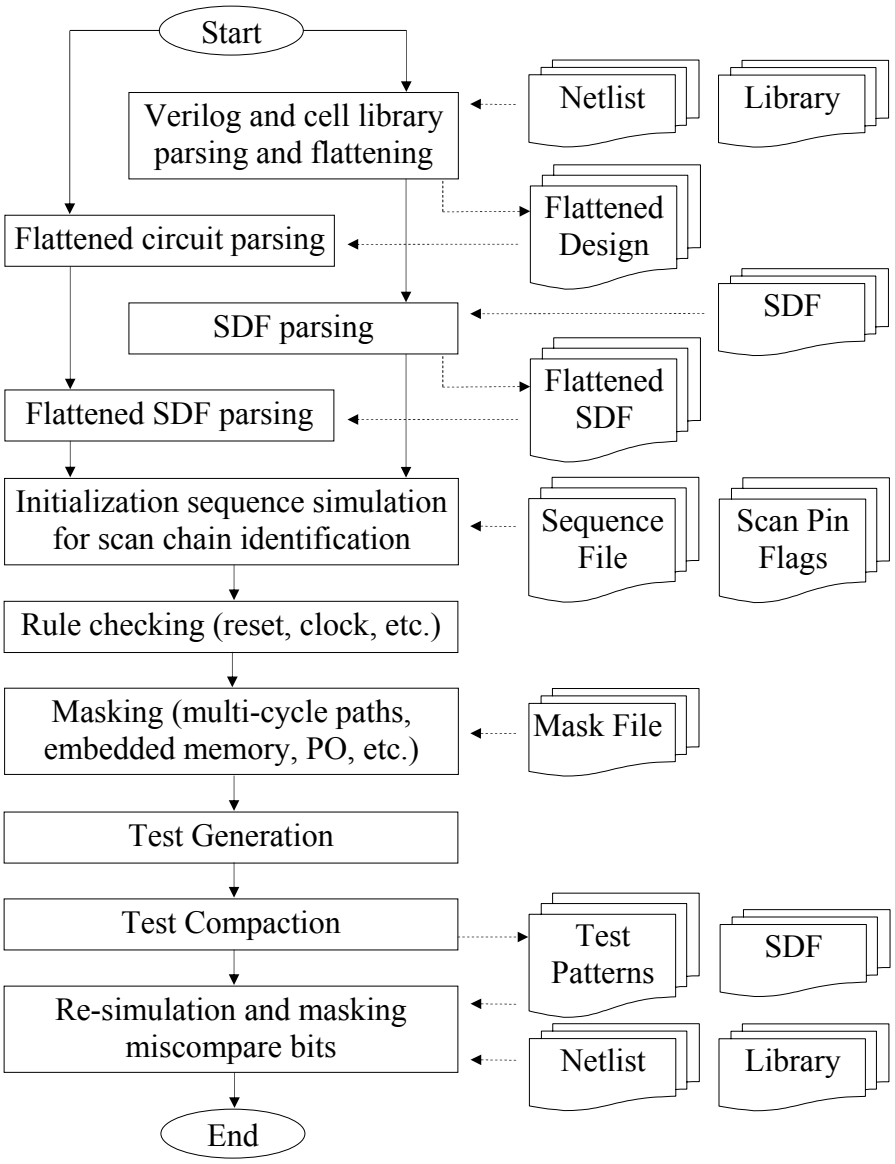


Figure 15. KLPG test generation flow.

Figure 15 shows the KLPG test generation flow. Files required in each step are listed on the right side. The first step is to parse the hierarchical netlist and library. In this work Verilog format is supported. When the hierarchical design is being flattened, each pin, cell, gate, and primitive is assigned an index. An index-to-hierarchical-name mapping table and hierarchical-name-to-index hash table are established and written to files. Since KLPG is not a design-for-testability (DFT) tool, it does not have to remember all the hierarchical names for future debugging (designs are assumed to be bug-free, e.g. a taped-out design). After this step, the long hierarchical names can be removed from memory to reduce memory consumption. Flattened design is also written to a file in Verilog format with simple names for pins, cells, gates, and primitives (using indices, such as N100, G200, etc.), so that the flattening does not have to be run for multiple times if test generation needs to be rerun, as long as the design is not modified.

The second step is to parse the standard delay format (SDF) file of the design. The SDF file specifies pin-to-pin path delays for cells and interconnects. In most cases, both maximum and minimum delays are included. The SDF file can be parsed cell by cell, so the memory consumption is insignificant. However, the whole hierarchical-name-to-index hash table must be read into memory to process hierarchical names in the SDF file. After the SDF file parsing is complete, the hash table can be removed from memory. Data structures that are used in the following steps can be set up at this point so that the peak memory usage can also be kept at a low level. In this work, the peak memory usage is about 1 KB per gate. An equivalent SDF file for the flattened design is saved. Because the new file does not contain any long hierarchical names, it can be processed faster in

future ATPG runs, without the hash table.

The following three steps (Test sequence simulation, rule checking, and masking) are similar to the standard flow used in commercial tools.

The sequence file (or procedure file) specifies test pin values under test mode, test setup clocking, pattern load/unload clocking, and other test configuration details. Scan pin flag file, which can also be included in the sequence file, specifies the functionality of each test pin, such as scan-in, scan-out, scan-enable, functional clock, test clock, etc. During test sequence simulation latches/flip-flops for clocking or testing are set to proper values, and every scan cell is identified (the scan chain it belongs to, position in the chain, inversion from scan-in and scan-out, etc.).

The purpose of rule checking (or test structure verification) is to verify that scan cells work properly during shift and test launch/capture, according to the given test sequence. Because commercial tools can do the same job, tool only provides basic rule checking, to roughly verify that the flattened design and test sequence do not have errors.

During the masking step, primary outputs and some scan cells are masked. For most designs, the destination scan cells of multi-cycle paths are masked. However, it has been observed that the transition fault coverage can be increased from 30.04% to 76.45% if the sources of multi-cycle paths are forced to X during ATPG, instead of masking the destinations. Masking is a tricky job for some designs. Circuits selected in this work do not have such problems.

In the test generation phase, K longest paths through each potentially faulty line are generated, with K paths having rising transitions at the fault site and the other K having

falling transitions. The corresponding test patterns are also generated, with one pattern per path before compaction. Then the patterns are statically compacted and saved.

The saved patterns need to go through re-simulation as a standard step in industry. During re-simulation, the hierarchical version of the design and SDF files are loaded into the simulator and any mismatches in scan-out data need to be masked. Then the patterns are rewritten in an industrial standard format such as the standard test interface language (STIL) format, and ready to be shipped to testers.

4.3 Path Generation

4.3.1 Preprocessing

Before path generation, some topology information is collected to help guide the path generation process and trim the search space. In this work, a launch point (of a path) is an unmasked primary input or scan cell, and a capture point is an unmasked primary output or a scan cell. First, the min-max delay from each gate to any capture point is computed without considering any logic constraint (PERT delay or STA delay). A gate's min-max PERT delay can be simply computed using its fan-out gates' min-max PERT delays and the rising/falling buffer-to-buffer delays between gates.

In addition to the PERT delays, the earliest and latest possible rising/falling transition times on the input and output lines for each gate are computed, assuming that a transition at any launch point can only occur at time zero. This procedure is similar to the PERT delay computation, with complexity linear in the circuit size. This information is useful under some sensitization criteria because transitions can occur only within the

earliest/latest range.

4.3.2 Path Store

To find the K longest testable paths through gate g_i, a path store is established for path generation. In the path store, many partial paths, which may become the K longest testable paths through gate g_i, are stored. A partial path is a path that originates from a launch point but has not reached a capture point. Figure 16 shows an example. The partial path starts from launch point g_0, and ends at gate g_i. At the beginning, the path store contains $2n_{LP}$ partial paths, where n_{LP} is the number of launch points. There are two partial paths from each launch point, representing a rising or falling transition at that launch point. Each partial path has only one node (a launch point) initially. A partial path grows when one more gate is added to it. When a partial path reaches a capture point, it becomes a complete path.

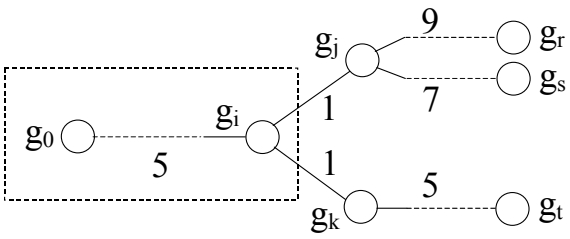


Figure 16. A partial path and its *esperance*.

A value called *esperance* [45] is associated with a partial path. The min-max esperance is the sum of the length of the partial path and the min-max PERT delay from its last node to a capture point. In other words, the max esperance of a partial path is the upper bound of its delay when it grows to a complete path, and the min esperance is the

lower bound if the complete path is testable. In Figure 16, suppose the length of the partial path $g_0 \ldots g_i$ is 5, and the PERT delays between gate g_i and capture points g_r, g_s, g_t are 10, 8, 6, respectively. The min-max esperance of partial path $g_0 \ldots g_i$ is 11/15.

The partial paths are sorted by max esperance. Every time the path generator selects the partial path with the largest max esperance. Potentially this partial path will grow to a complete path with maximum delay.

4.3.3 Algorithm

Before the path generation for gate g_i, all gates that are not in gate g_i's fan-in or fan-out cone are identified because when a partial path grows, it is impossible for any of these gates to be added (otherwise the partial path has no chance to pass through gate g_i). These gates are still useful because they are related to constraints, such as side input constraints of a gate on the path that is being searched. Figure 17 shows the search space.

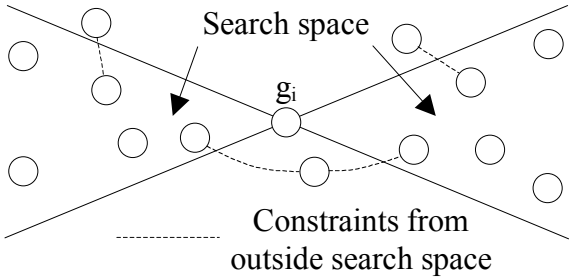


Figure 17. Search space for a path through a particular gate.

Figure 18 is the algorithm of finding the K longest testable paths through gate g_i. Each iteration of the path generation begins by popping the first partial path from the path store, which has the largest max esperance. The partial path is extended by adding a fan-out gate that contributes to the largest max esperance. For example, in Figure 19, the

partial path $g_0 \ldots g_i$ is extended by adding gate g_j because extending to g_j could potentially keep the max esperance. If the partial path has more than one extendable fan-out, it must be saved in another copy, and in the copy, the already tried fan-out must be marked "blocked" or "tried." The esperance of the copy is then updated and the copy is pushed into the path store. For example (Figure 19), since gate g_i has 2 fan-outs, and extending the partial path to g_j may result in false paths later, the partial path $g_0 \ldots g_i$ must be saved because extending it to gate g_k may get a longer testable path. Because fan-out g_j has been tried, in the copy the min-max esperance becomes 11/11.

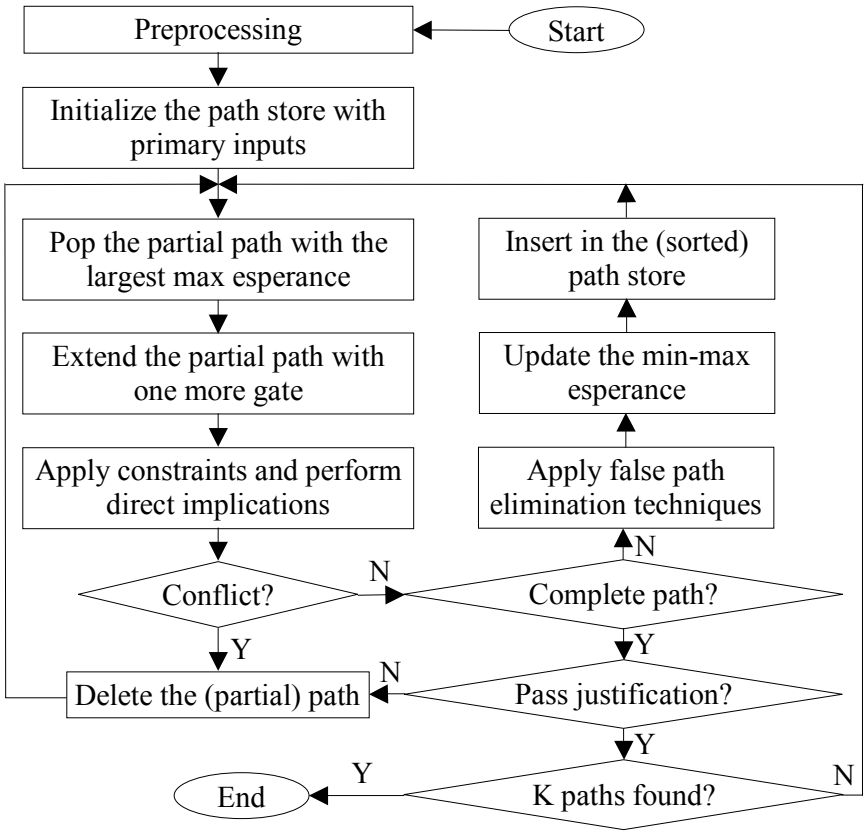
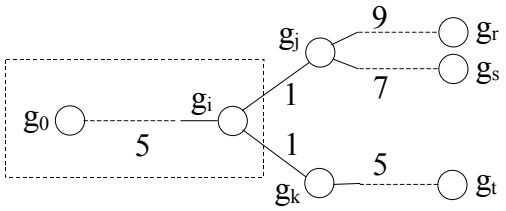


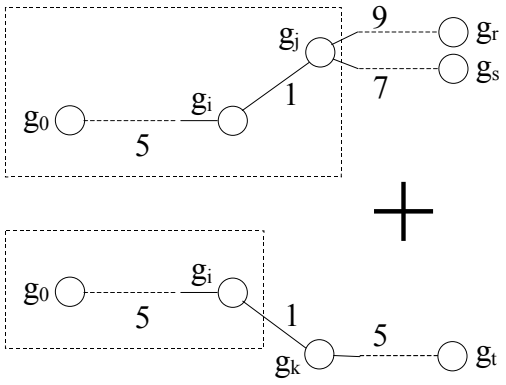
Figure 18. Path generation algorithm.

After the partial path is extended (path $g_0 \ldots g_i g_j$ in Figure 19) the constraints to

propagate the transition on the added gate (g_j) are applied. Under the non-robust sensitization criterion [20], non-controlling final values on the side inputs are required. Under the robust sensitization criterion [44], in addition to non-controlling final values, the side inputs must remain non-controlling if the on-path input has a transition to the controlling value. Then direct implications are used to propagate the constraints throughout the circuit to prune non-solution search space. When a partial path reaches a capture point, it becomes complete path. Final justification is performed on it to find a vector pair. Direct implications and final justification will be discussed in detail in the following sections.



(a) Before extension



(b) After extension

Figure 19. Partial path extension.

If the extended partial path is not a complete path, some false path elimination

techniques are applied to it, to more efficiently prevent the new partial path from becoming a false path. Then the min-max esperance of the partial path is updated and it is inserted into the path store. Since its max esperance may decrease and min esperance may increase after extension, it may not be inserted at the top of the path store. If this happens, in the next iteration another partial path will be selected for extension. For example (Figure 19), after extending partial path $g_0 \ldots g_i$ to gate g_j, the min-max esperance changes from 11/15 to 13/15. If path $g_j \ldots g_r$ is blocked, which means path $g_0 \ldots g_r$ is a false path, after applying false path elimination techniques, the min-max esperance is 13/13.

Because each partial path consumes memory, and the path store cannot have an infinite size, when the number of partial paths exceeds the path store size limit, some partial paths with low max esperance are removed from the path store. The maximum esperance of the removed partial paths is recorded. In the future, any partial path with max esperance below that value must be removed from the path store because it may not truly be one of the K longest testable paths. Therefore, it may happen that K paths have not been found when the path store is empty. However, since a partial path is represented as a sequence of gates, usually it consumes less than 1 KB memory. Thus, the path store can have a large number of partial paths so that in most cases the algorithm does not abort unless none of the structural paths through the gate is testable.

The path generation iteration does not stop until the K longest testable paths through gate g_i are found or the path store is empty. Since the K longest testable paths through different gates may overlap, every time a new path is generated, it must be checked to

see if it has already been generated during the path generation for another gate.

4.3.4 Direct Implication

A direct implication on a gate is one where an input or output of that gate can be directly determined from the other values assigned to that gate. Figure 20 shows some examples of direct implications on an AND gate. The values in boxes are implied from the existing values, which are not in boxes. Figure 20(a) is an example of forward implication, and (b)(c) are examples of backward implications. If a conflict happens during direct implications, the partial path is false. In other words, any path including this partial path is a false path. For example (Figure 19), if extending partial path $g_0 \ldots g_i$ to gate g_j results in a conflict (Figure 21 shows an example), both path $g_0 \ldots g_r$ and $g_0 \ldots g_s$ are determined to be false. Therefore, the partial path is deleted from the path store so that the whole search space that contains this partial path is trimmed off. Previous research has shown that most false paths can be eliminated by direct implications [45], and this is also observed in this work.

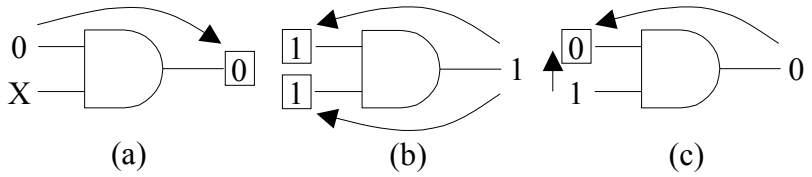


Figure 20. Examples of direct implications.

Direct implications can be performed on scan cells as well as regular gates to detect most local conflicts and eliminate sequential false paths.

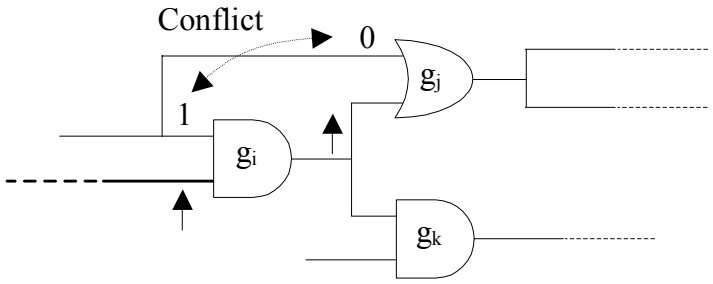


Figure 21. Conflict after applying direct implications.

If the launch-on-shift approach is used, the logic values on neighboring scan cells are dependent on each other. For example, in Figure 22, the logic value of cell *A* in the first vector is the same as that of cell *B* in the second vector. The relation between cell *B* and *C* is the same. Therefore, if there is a rising transition assigned to cell *B*, direct implications try to assign a logic 1 to cell *A* in the first vector and a logic 0 to cell *C* in the second vector, and propagate the new assignments throughout the circuit. If there are any conflicts, the partial path is a sequential false path under the launch-on-shift constraints. It is assumed that the scan chain design cannot be modified to reduce the dependence, such as inserting dummy cells between the scan cells.

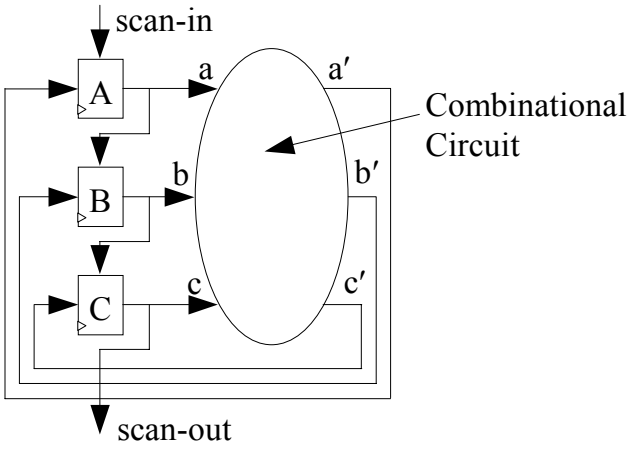


Figure 22. Direct implications on scan cells.

If the launch-on-capture approach is used, dependence exists between the two vectors. Even if the circuit has a pipeline structure, in which the two vectors are independent, the structure can also be seen as the general structure shown in Figure 22. The conversion is shown in Figure 23. Thus, the second vector is the output of the combinational circuit, derived from the first vector, excluding the primary input and output bits. In other words, $V_2=C(V_1)$, where V_1 and V_2 are the two vectors and C is the logic of the combinational circuit. For example, if it is assumed that a testable path has a rising transition launching from cell A and a rising transition captured on cell B, in Figure 22, then for the first vector, output a' must be a logic 1 (then it becomes the value for input a in the second vector); and for the second vector, input b must be a logic 0 because it is derived from the first vector. Then more direct implications can be performed from a' and b.

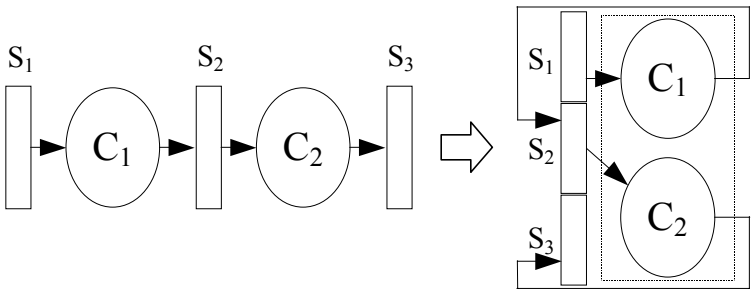


Figure 23. A pipeline structure.

4.3.5 Final Justification

If an extended partial path reaches a capture point, it becomes a complete path. In this case, a final justification process (a PODEM [83] style decision tree based justification algorithm) is performed on the path. One reason to do final justification is to find a test pattern that sensitizes this path. The other reason is to exclude the false paths that do not

have any direct implication conflict during the path growth procedure. Figure 24 shows an example [45]. Suppose both AND gates need their side inputs to be logic 1, and direct implications stop at the two OR gates because neither input must be logic 0 or 1, assuming the algorithm does not know the two inputs are tied together. This path does not fail direct implications but it is a false path. Since most false paths cannot pass the direct implications, it is not often that a complete path fails the final justification.

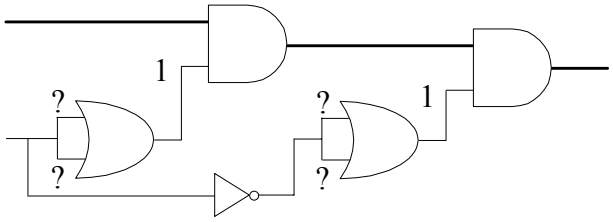


Figure 24. A path that passes direct implications but fails final justification.

For sequential circuits, since the two vectors are dependent, whenever a decision (a logic value on any bit in either vector) is made at a primary input or scan cell, direct implications have to be performed to trim the search space. For the launch-on-shift approach, both vectors can be justified in this way.

For the launch-on-capture approach, because the second vector is derived as the circuit response to the first vector, one time frame expansion is used. In Figure 25, both the circuit and scan chains are duplicated. The first vector V_1 can be generated within one time frame, but since the second vector $V_2=C(V_1')$, the goal is to find a satisfying V_1'. Because V_1 and V_1' are identical excluding the "don't care" bits, in the justification process there must be no conflicts between V_1 and V_1', i.e. a bit is logic 1 in V_1 but 0 in V_1' (it is consistent if one of them is a "don't care"). Similarly, whenever a decision is

made on any bit in either vector, direct implications must be performed to keep the logic assignments on any line in the two identical circuits consistent.

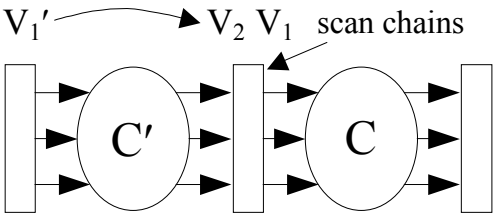


Figure 25. Time frame expansion for final justification using launch-on-capture.

4.4 Heuristics for False Path Elimination

4.4.1 Forward Trimming

When a partial path grows, in some cases after the constraints are applied and the direct implications are performed, the possibility to continue extending the partial path to capture points may be reduced. In the extreme case, there is no way to continue extending the partial path (completely blocked).

Figure 26 shows an example where the path through the logic block cannot propagate to capture points [81]. In this example, the partial path has grown to the NAND gate g_i, and the side input must be logic 1 if the single-path sensitization constraints [45] are considered. This value propagates forward through the inverter and becomes a controlling value on one of the inputs of the NAND gate g_j. This condition prevents propagation from gate g_i to gate g_j through any paths within the logic block. With forward trimming, the entire logic block is trimmed off and the search process is guided

toward an unblocked path (the upper inverter) earlier. Without forward trimming, the search process might be much less efficient. Since the paths through the logic block are not blocked until the final gate g_j, the search process may attempt to traverse through the logic block until it reaches gate g_j and learn that the path is blocked. This may be done for each possible path through the logic block.

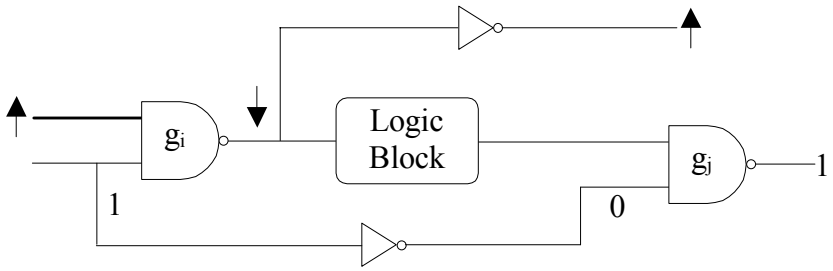


Figure 26. Application of forward trimming.

During the circuit preprocessing the PERT delays are computed assuming no path is blocked. With the growth of a partial path, more and more information is known because more constraints are applied. Forward trimming recomputes the min-max PERT delay from the end of the partial path, based on the structure and the current value assignments. In this case, blocked extension choices are not considered in the computation and false paths can be eliminated earlier so that the search can be guided more accurately toward a testable path. If the search space is partially trimmed off, the partial path still has a chance to become a testable complete path, but its esperance may be reduced according to its PERT delay reduction. In the next iteration of path generation, a more promising partial path may be selected.

4.4.2 Smart-PERT Delay

If the PERT delays are used, a local conflict in the unexplored search space is not detected until a partial path grows to that site, because the PERT delays are computed without considering any logic constraint. ISCAS85 circuit c6288 is a good example to show the inefficiency of this methodology. Path generation for this circuit has been known as a hard problem before the heuristic is developed [84].

ISCAS85 circuit c6288 is a 16×16 multiplier. Figure 27 [85] shows its structure. The circuit contains 240 adders, among which 16 are half adders, which are shaded in Figure 27. $P_{31}...P_0$ are the 32-bit outputs. Each floating line, including P_0, is fed by an AND gate, whose inputs are connected to two primary inputs.

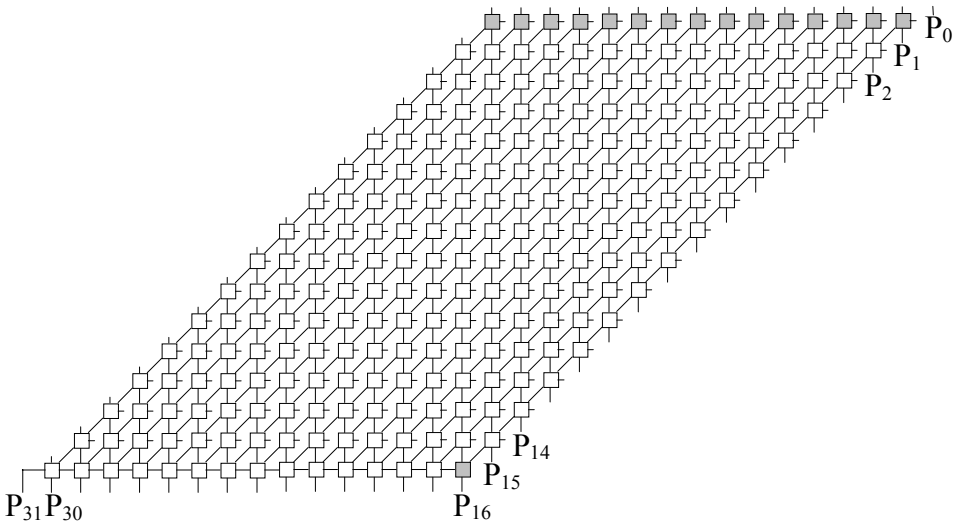
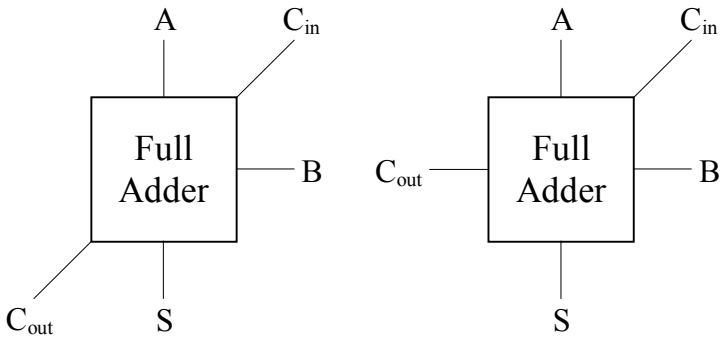


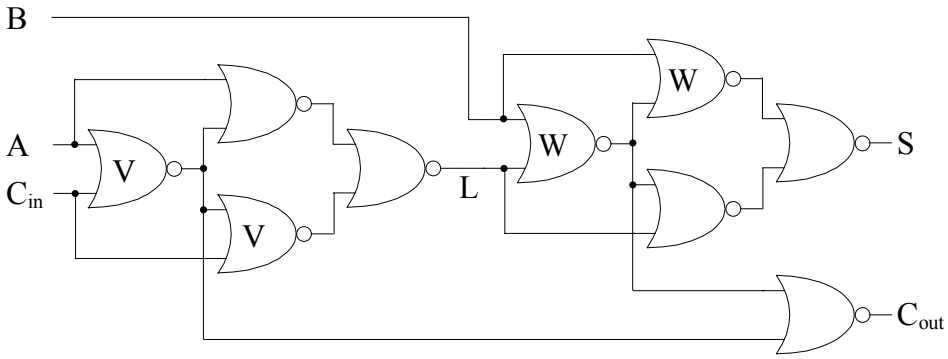
Figure 27. Structure of ISCAS85 circuit c6288 16×16 multiplier.

Figure 28(a)/(b) [85] shows the symbolic and schematic view of a full adder in c6288, respectively. The 15 top-row half adders in Figure 27 lack the C_{in} input. Each of them has two inverters at locations *V* in Figure 28(b). The single half adder in the bottom row

lacks the *B* input, and it has two inverters at locations *W*.



(a) Symbolic view



(b) Schematic view

Figure 28. Full adder module in c6288.

Table 5 shows the delays between all the input-output pairs in the full adder, assuming every gate has one unit delay and no sensitization constraint is considered. In other words, the numbers are the structural length between the input-output pairs.

Table 5. Maximum delays between input-output pairs in a full adder.

	A	*B*	C_{in}
S	6	3	6
C_{out}	5	2	5

The paths highlighted in Figure 29 are the longest structural paths in the circuit. These paths are all from primary input A_{15} or B_0, to output P_{30}. Each of these paths contains 124 gates, and must include the longest structural path from input A or C_{in} to output C_{out} in at least one adder.

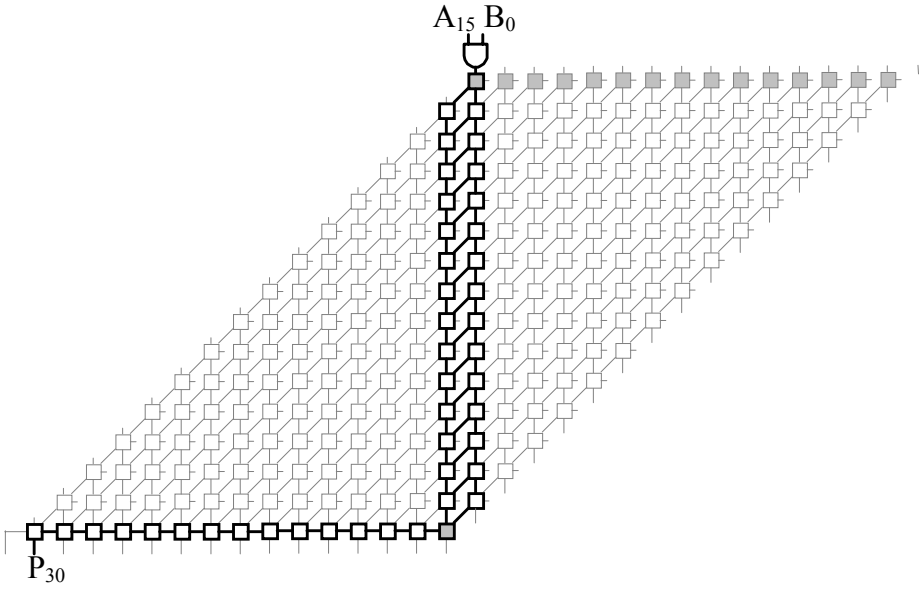


Figure 29. Longest structural paths in c6288.

However, there is no robust test for the longest structural paths from input C_{in} to output C_{out} in a full adder. Figure 31 shows the signal propagation and necessary assignments if a rising or falling transition is applied to input C_{in}. Similarly, the longest structural paths from input A to output C_{out} are not robustly testable either. Therefore, the 5-gate paths from input A or C_{in} to output C_{out} are not robustly testable. However, the 4-gate paths from input A or C_{in} to output C_{out} are robustly testable, if the first gate is bypassed. Figure 30 shows an example.

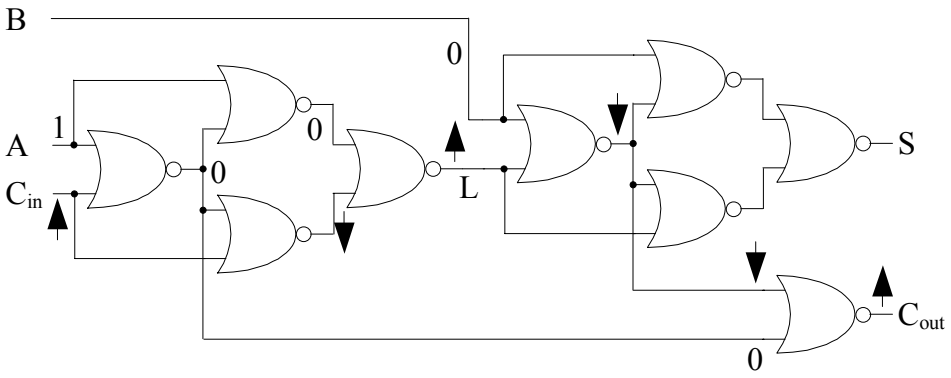
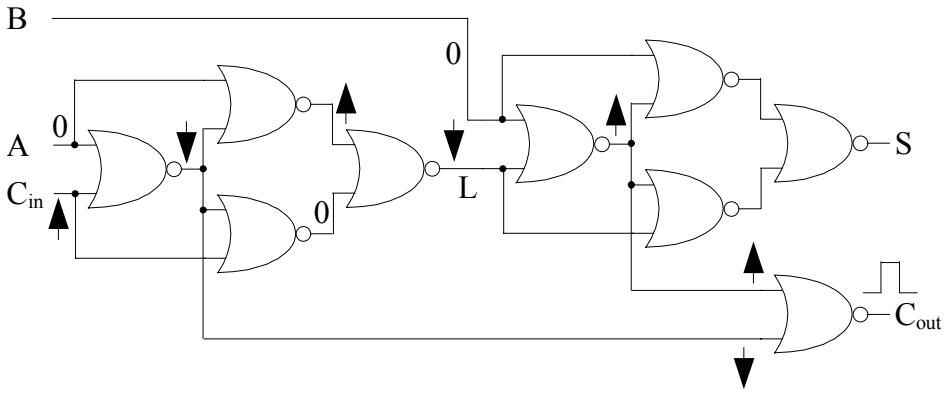
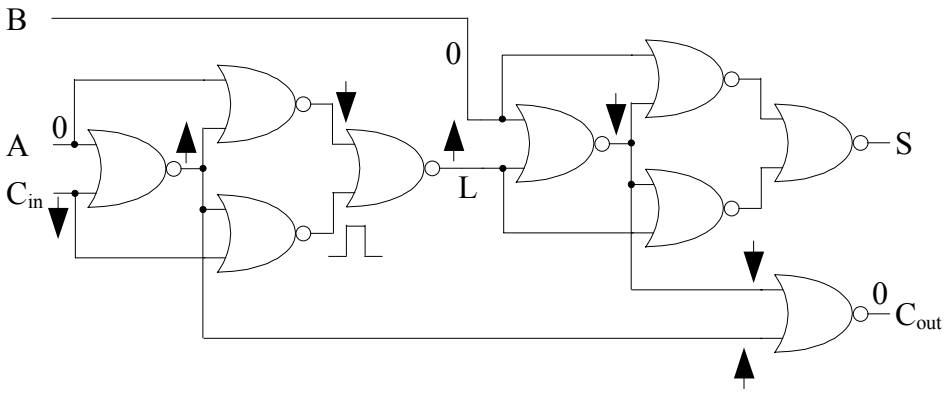


Figure 30. A longest robustly testable path through output *Cout* in a full adder.

Thus, when a robustly testable path goes from input A or C_{in} to output C_{out} through a full adder or the half adder in the bottom row, its delay must be at least one gate less than that of a longest structural path. It can be seen from Figure 27 that this case must happen in one full adder and the half adder in the bottom row. Therefore, the delay of a longest robustly testable path is at least two gates less than that of a longest structural path, which contains 124 gates. With similar analysis, the delay of a longest non-robustly testable path is at least one gate less than that of a longest structural path. The reason is, if the path goes through a longest structural path from A or C_{in} to C_{out} in a full adder, there must be a glitch at C_{out}, according to Figure 31. This glitch cannot propagate through the longest structural paths from input A or C_{in} to output C_{out} in the half adder in the bottom row. Therefore, the non-robustly testable path must bypass one gate before it reaches the half adder, or bypass one gate in the half adder. Then a glitch can be generated at the output C_{out} of the half adder and propagated to the primary output P_{30}, without bypassing any more gates.



(a) A rising transition at C_{in}



(b) A falling transition at C_{in}

Figure 31. Longest structural paths through output *Cout* in a full adder.

The reason why traditional path generators fail on c6288 is that all the longest structural paths, which have 124 gates, are not testable. Because of the reconvergence within the adder modules (local reconvergence) and in the adder network (global reconvergence), the number of longest structural paths is exponential in the number of gates. For example, in an adder network shown in Figure 32, there are n full adders and the output S of the ith adder is connected to the input A of the (i+1)th adder ($0<i<n$). The dotted curves represent the four reconverged longest structural paths between input A

and output C_{out} within an adder. In this network, there are 2^{2n} longest structural paths, which have equal length, from the input A_1 to the output S_n. If they are all false paths, to prove so many longest structural paths are untestable consumes too much CPU time for traditional path generators.

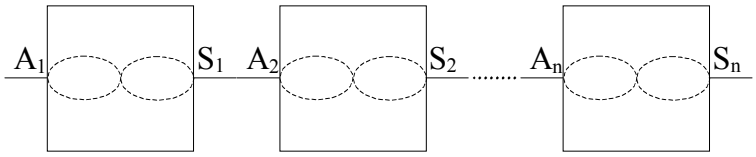


Figure 32. An adder network example.

The path generator used in previous research [81] is able to find the global longest testable path in c6288. The reason is that it uses a dynamic dominator heuristic [86]. This tool grows paths from primary inputs. It can be seen in Figure 33 that when a path reaches line *H* it must go through line *L*, and when it reaches line *P* it must go through line *S*, assuming the path attempts to extend to the farthest primary output from that point. It is said that line *L* dynamically dominates line *H* and line *S* dynamically dominates line *P*. Therefore, if the path is blocked somewhere and the conflict is irrelevant to the decision of going through the upper NOR gate or the lower NOR gate, all the paths through the series of dynamic dominators can be proven untestable.

However, the limitation of this heuristic is that it only solves the problem of exponential number of paths due to the local reconvergence within an adder. It does not solve the problem due to the global reconvergence in the adder network. Fortunately, as shown in Figure 29, there is not much global reconvergence which causes backtracks, before a global longest testable path is generated. This assumes the local reconvergence

problem is solved, so that there are few backtracks within an adder.

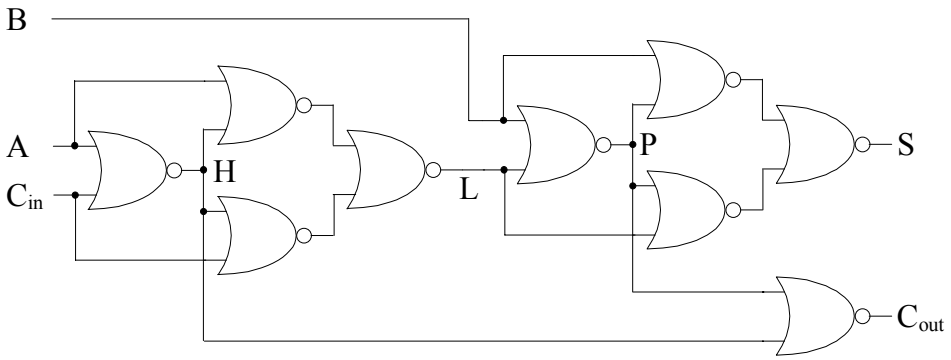


Figure 33. Dynamic dominators in a full adder.

Without the smart-PERT heuristic, the global reconvergence problem is much worse if longest testable paths through some gates/adders in c6288, e.g. the black one in Figure 34, are required. The longest structural paths are highlighted in Figure 34. As discussed above, whenever a robustly testable path passes from the input A or C_{in} to the output C_{out} in an adder, it must bypass at least one gate. So the delay of a longest robustly testable path through a particular adder is several gates less than that of a longest structural path through this adder, depending on which column the adder is located in. For example, a longest robustly testable path through the black adder in Figure 34 should bypass at least six gates. This results in more false paths that are longer than the longest testable paths.

The smart-PERT heuristic is able to exclude untestable subpaths due to local conflicts caused by both local and global reconvergence when computing the PERT delay for a gate. The new values are called Smart-PERT delays, or S-PERT. For simplicity, only maximum PERT and S-PERT delays are discussed. Because some untestable subpaths are not included in the S-PERT computation, a gate's S-PERT delay is always less than

or equal to its PERT delay. Moreover, compared to the PERT delay, the S-PERT delay is closer to the delay of the longest testable path from that gate to a capture point.

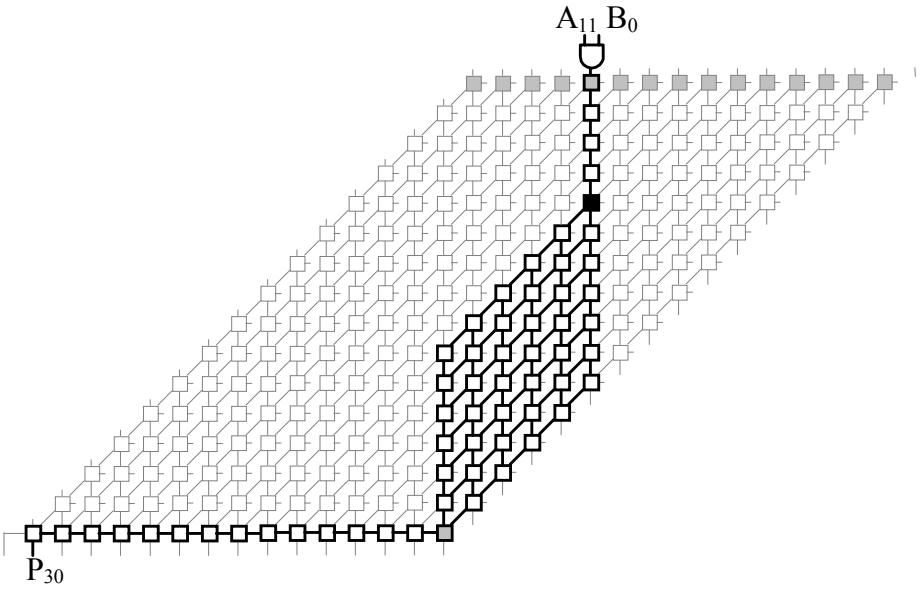


Figure 34. Longest structural paths through a particular adder.

A gate's PERT delay can be computed using its fan-out gates' PERT delays. If the unit delay model is used, PERT(g_i) = max {PERT(g_j) | g_j is a fan-out gate of g_i} + 1. Figure 35(a) shows an example, assuming PERT(g_3) = 8 and PERT(g_4) = 6 are known. In this example, PERT(g_0) = 10 is computed using PERT(g_1) and PERT(g_2).

When S-PERT(g_i) is computed, a user-defined variable S-PERT depth is used. If the S-PERT depth is set to d, then S-PERT(g_i) is computed using S-PERT(g_j) where g_j is d gates from g_i in g_i's fan-out tree. For example, in Figure 35(b), if d is set to 2, then S-PERT(g_0) is computed using S-PERT(g_3) and S-PERT(g_4).

The heuristic works as follows. Suppose S-PERT(g_i) is being computed. G = {g_j | g_j is d gates from g_i in g_i's fan-out tree}, and G is sorted by S-PERT(g_j) in decreasing order. The heuristic pops the first gate g_j in G and attempts to propagate a transition from gate

g_i to gate g_j. If there is no conflict (the transition successfully reaches gate g_j, with all the constraints applied), S-PERT(g_i) is set to S-PERT(g_j) + d. Otherwise, it pops the second gate in G and repeats the same procedure. In Figure 35(b), for example, at first the heuristic tries to propagate a transition from gate g_0 to gate g_3, but finds it is impossible to set the side inputs of gates g_1 and g_3 both to non-controlling values. Then it tries gate g_4 and does not meet any conflict. So S-PERT(g_0) is 8. It is obvious that increasing the S-PERT depth can make the S-PERT delays closer to the delay of the longest testable path from that gate to a capture point, but its cost increases exponentially. Therefore, there must be some trade-off.

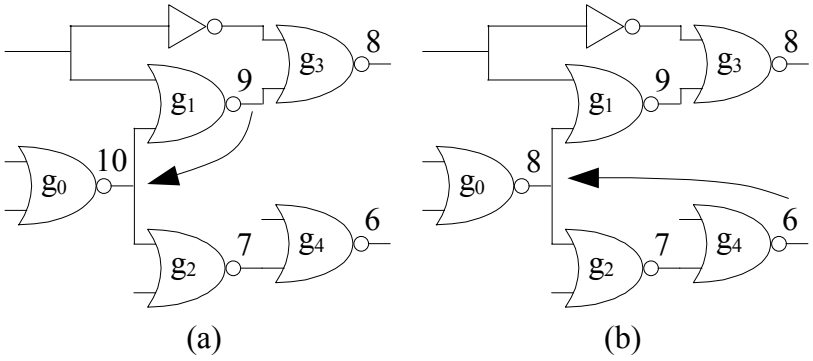


Figure 35. Computation of PERT delay and S-PERT delay.

Since most conflicts are local, with S-PERT delays, the path generation is well guided to a testable path, with many fewer conflicts, because most of the non-solution space is trimmed off during the preprocessing phase.

The usefulness of this technique is highly dependent on the structure of the circuit. The most benefit would be derived from a path with d gates, each of which has fan-out f and each fan-out reconverges at a later gate (Figure 36), such as circuit c6288. This

results in f^d possible paths that must be traversed. The worst case is that all of them are false paths but conflicts do not occur until the partial path grows very long. With the use of S-PERT delays, the path generation extends to a shorter structural path P_2 (Figure 36) because it has larger esperance, and the traversal of all the false paths with equal length from gate g_1 to gate g_4 is avoided. As the experimental results show, this technique helps exclude most false paths in circuit c6288.

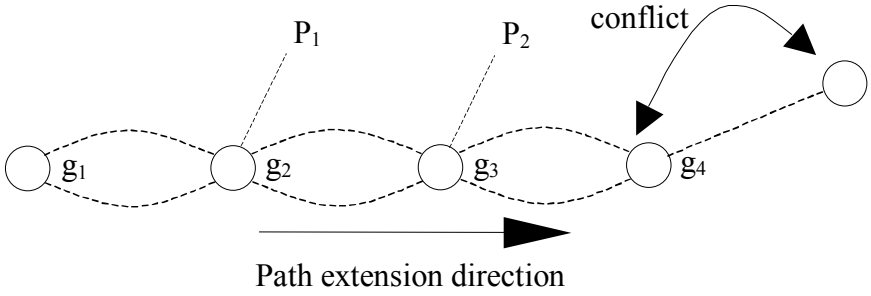


Figure 36. A circuit with an exponential number of false paths.

4.4.3 Seven-Value Algebra

If the circuit is not full scan, the non-scanned cells may or may not be initialized after the first vector is scanned in. Logic simulation of the test setup sequence and scan procedure is done before the ATPG is performed, to identify such cells, as shown in Figure 15. Industrial designs also contain embedded memories, whose values cannot be easily initialized during the test for the logic. Extensive research [87][88] has been done to solve the initialization problem. However, there may still be many non-scanned cells that cannot be initialized. These bits are considered “uncontrollable” in the test generation. In commercial tools, embedded memories are usually considered “black-boxes” as well.

The algebra used in this work includes seven values: logic 0/1, *x* (unknown/unassigned), *u* (uncontrollable), 0/*u* (0 or uncontrollable), 1/*u* (1 or uncontrollable) and *x*/*u* (unknown or uncontrollable). At the beginning of test generation, the lines from the non-scanned cells are *u* and all the other lines are *x*. Both *u* and *x* have "don't know" values but *x* may be assigned a value in the test generation process (assuming controllable). Figure 37 shows the truth table of a 2-input AND gate. For example, if one input is *x* and the other is *u*, the output is 0/*u* because if the input with *x* is assigned a logic 0 the output becomes 0, but if this input is assigned a logic 1 the output becomes uncontrollable. Before the test generation, logic simulation is performed throughout the circuit to reduce the number of *x*'s. Figure 38 shows two examples, assuming M_1 is a non-scanned cell and M_2 is a scan cell. The logic values assigned by simulation are shown. If the conventional 3-value algebra is used, all the lines are assigned *x*'s.

	0	1	*x*	*u*	0/*u*	1/*u*	*x*/*u*
0	0	0	0	0	0	0	0
1	0	1	*x*	*u*	0/*u*	1/*u*	*u*
x	0	*x*	*x*	0/*u*	0/*u*	*x*/*u*	*x*/*u*
u	0	*u*	0/*u*	*u*	0/*u*	*u*	0/*u*
0/*u*	0	0/*u*	0/*u*	0/*u*	0/*u*	0/*u*	0/*u*
1/*u*	0	1/*u*	*x*/*u*	*u*	0/*u*	1/*u*	*x*/*u*
x/*u*	0	*u*	*x*/*u*	0/*u*	0/*u*	*x*/*u*	*x*/*u*

Figure 37. Truth table of an AND gate using 7-value algebra.

Using this 7-value algebra significantly speeds up the test generation, because it

divides unknown values into controllable and uncontrollable categories. In the example shown in Figure 38(a), since the logic value on line n_3 can never be a logic 1, all the paths through line n_4 are false. Thus, the test generation stops growing partial paths at line n_4 and all the gates in the fan-out cone of line n_4 are pruned. If the conventional 3-value algebra is used, the test generation may have to generate all the paths through line n_4 and find there is no test for any of them. Moreover, by looking at the logic values on line n_5, it can be learned that it is impossible to intentionally make a transition on this line, because logic 1 is not achievable, therefore both slow-to-rise and slow-to-fall faults on this line are untestable. Since all the paths through line n_4 must contain line n_5, it can also be known that both delay faults on line n_4 are untestable. In summary, many faults can be proven untestable by simple analysis, before the test generation is performed, and non-solution search space is more efficiently pruned during the test generation.

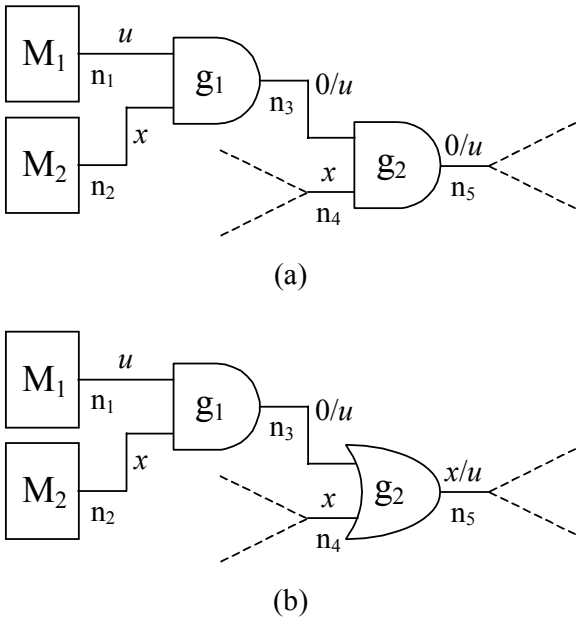


Figure 38. Application of 7-value algebra.

Figure 38(b) shows another example. When a partial path reaches gate g_2, the value of the second vector on side input n_3 is set to logic 0 (non-controlling value, for both robust and non-robust test requirement). Then direct implications are performed backward. The value of the second vector on line n_2 is set to logic 0 and further direct implications for the first vector can be performed from M_2. Thus, conflicts can be found earlier. If the conventional 3-value logic is used, direct implications stop at gate g_1. Some conflicts may be hidden.

4.4.4 Relations of Gates

There are tight relations among long testable paths through different gates because a long testable path through a gate is possibly a long testable path through another gate. Some rules in previous research [43] are extended in this work and they are very easy to apply to the path generation algorithm.

Each gate in the circuit has two arrays: L_{ub}[1…K] and L_{lb}[1…K], which indicate the upper and lower bounds of the lengths of the K longest testable paths through this gate. The two arrays are sorted. Initially the values in L_{ub}[1…K] are all set to the length of the longest structural path through the gate, and the values in L_{lb}[1…K] are all 0.

When the K longest testable paths are found for gate g_i, L_{ub}[1…K] and L_{lb}[1…K] for gate g_i are updated and both of them are set to the actual lengths of the K longest testable paths. Suppose a newly found path for gate g_i also contains gate g_j, which means this path passes through both gates g_i and g_j. If the length of this path is greater than that of a previously found path for gate g_j, L_{lb}[1…K] for gate g_j is updated by inserting a link to the newly found path and deleting the link to the shortest path found. This process may

increase the values in L_{lb}[1…K] for all the gates contained in the newly found path. Figure 39 shows an example. Assuming K=3, at some point L_{lb}[1…3] for gate g_j is {22,18,15}, which means the lengths of the 3 longest paths through g_j found by the path generator are at least 22, 18 and 15. Suppose the length of a newly found path for gate g_i is 20. Then L_{lb}[1…3] for gate g_j is updated to {22,20,18}.

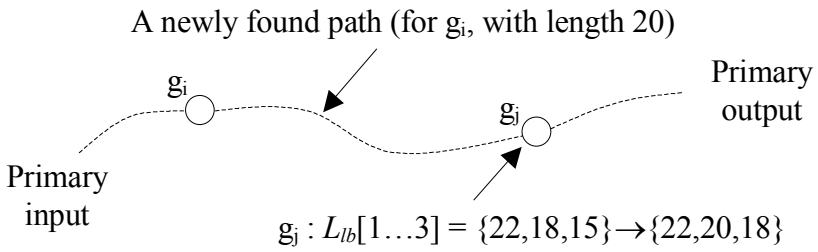


Figure 39. Updating the lower bound of path lengths.

On the other hand, the values in L_{ub}[1…K] for some gates decrease when a new path is found. Suppose gate g_i has f fan-in gates, and $\cup_{\text{fanin}} L_{ub}$[1…K] indicates the union of the L_{ub}[1…K] arrays of its fan-in gates and it is sorted in decreasing order. The upper bound of the lengths of the K longest testable paths through gate g_i cannot exceed the first K values in $\cup_{\text{fanin}} L_{ub}$[1…K], because all the paths through gate g_i must also pass through one of its fan-in gates. Figure 40 shows an example. Assuming K=3, and gate g_i has two fan-in gates, with L_{ub}[1…3]={17,16,11} and {20,18,12}. Then the values in L_{ub}[1…3] for g_i must be no more than {20,18,17}. The same analysis can be performed using the fan-out gates or absolute dominators [81] of gate g_i.

As more paths are found, the values in L_{ub}[1…K] and L_{lb}[1…K] for gate g_i, for which the path generation has not been performed, get closer. If they are close enough, say the

difference is less than 1%, it can be assumed that the K longest testable paths for gate g_i have been found so that the path generation for it can be skipped. Many gates can be skipped if a unit delay model is used [43].

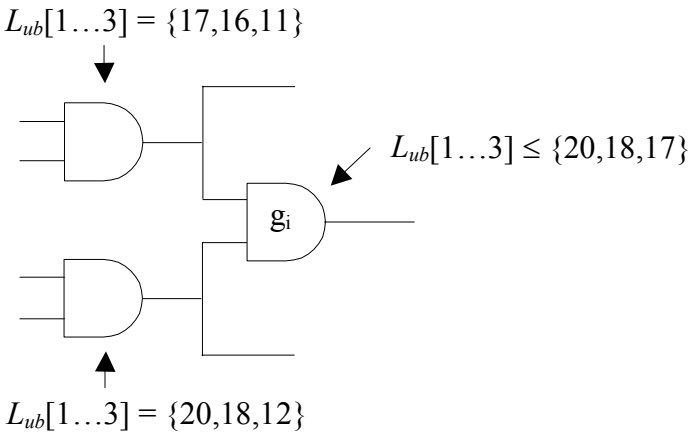


Figure 40. Updating the upper bound of path lengths.

If gate g_i cannot avoid path generation, during its path generation process, if the max esperance of the partial path being processed is less than its $L_{lb}[v]$ ($1<v\leq K$) value, then the first v paths already found for gate g_i are proved to be the v longest paths through g_i, because the partial paths in the path store have no chance to grow to a complete path with larger length. So $L_{ub}[u]$ (u=1,…,v) are updated accordingly (set to $L_{lb}[u]$). On the other hand, when the vth longest path through gate g_i is being searched, and the min esperance of a partial path is greater than $L_{ub}[v]$, the partial path can be immediately deleted, because when it becomes a complete path, it either is a false path or has already been found.

The $L_{ub}[1]$ values of other gates can also be taken advantage of during the path generation process for gate g_i. Suppose a partial path grows to gate g_j. If the max

esperance of the partial path is greater than the L_{ub}[1] value for gate g_j, it must be reduced to L_{ub}[1] for gate g_j, because it is impossible for this partial path to grow to a complete path with length greater than L_{ub}[1] for gate g_j, otherwise this path would also be a testable path through gate g_j, which invalidates L_{ub}[1] for gate g_j. When the partial path continues to grow, its min esperance may increase, and if it becomes larger than its max esperance, the partial path is deleted because it must eventually grow to a false path.

4.4.5 Global Longest Path Generation

Global longest paths (or critical paths) are the longest paths throughout the circuit, regardless of which gates they pass through. The global longest path generation algorithm is a slight modification to the path generation algorithm for a particular gate (Figure 18). If no gate is eliminated from being added to a partial path (gates that are not in gate g_i's fan-in and fan-out cones are eliminated for the path generation for gate g_i), the complete paths generated from the path generation are the global longest paths.

The advantage of finding the global longest paths is that if there are p global longest paths covering gate g_i, these paths must be the p longest paths through g_i. Therefore, at the beginning, the global longest path generation can cover many gates. For comparison, if no global longest path generation were performed, gate g_i would also get some potential longest path through it during the path generation process for other gates, but in most cases, they would not be "verified" until the path generation for gate g_i is performed.

However, as the global longest path generation finds more paths, the possibility that more gates are covered falls. The worst case is that almost all the global long paths only

pass through a very small subset of the gates. Therefore, at the beginning, the global longest path generation is useful, but after a certain number of paths, it is necessary to apply the path generation aiming at individual gates.

In this work, a 2-phase strategy is used: Run the global longest path generation until no more gates benefit. Then run the path generation for individual gates that are not fully covered during the global path generation.

The benefits from the global longest path generation are not only that it drops some gates from the individual path generation; it also speeds up the individual path generation for the gates that are not dropped. Suppose the length of the last generated global longest path is *L*. During the individual path generation, all the partial paths with min esperance greater than *L* can be removed because all the testable paths whose length is greater than *L* have already been generated. This technique is especially useful when a circuit contains many long false paths.

4.5 Experimental Results

The KLPG tool has been implemented in Visual C++ and run on Windows 2000 with a 2.6 GHz Pentium 4 processor and 2 GB memory. Buffer-to-buffer nominal delays are extracted from circuit layouts and used in the experiments.

4.5.1 Effectiveness of Heuristics

To quantify the effectiveness of the false path elimination heuristics, some of them are turned off during experiments on ISCAS85 and combinational ISCAS89 circuits. Table 6 shows the CPU time, assuming K=3 and the robust sensitization criterion is used.

Table 6. Effectiveness of false path elimination techniques (K=3, robust).

Circuit	CPU Time (m:s)			
	No FT	No SP	No R&G	All Used
c1908	2:25	0:23	0:36	0:25
c2670	0:21	1:07	0:43	0:13
c3540	3:49	0:57	1:45	0:58
c5315	0:37	0:26	0:46	0:26
c6288	58:13	×	62:38	38:10
c7552	1:21	0:55	1:42	0:59
c9234	1:48	1:11	5:04	1:17
c13207	2:24	2:49	4:13	1:52
c15850	3:12	2:33	6:25	2:39
c38417	8:20	6:35	15:26	6:43
c38584	14:31	10:42	20:09	10:49

"×" means the path generation did not finish within 12 hours.

The results assuming the forward trimming (FT) or smart-PERT (SP) technique is not used are listed in columns 2 and 3. Column 4 lists the CPU time if neither the relations of gates nor the global longest path generation (R&G) is used. Column 5 is the CPU time if all the techniques are used. It can be seen that the forward trimming technique significantly reduces the CPU time; the smart-PERT technique is not very useful for most circuits, and using it sometimes results in slightly longer CPU time, but it significantly helps circuits c2670 and c6288. The reason is because in most cases, the local conflicts do not cause an exponential number of false paths and these local conflicts can be efficiently removed by forward trimming. Moreover, for most circuits the cost of using the smart-PERT technique is greater than the benefits, while for circuits

with many reconvergences and long untestable paths, such as circuits c2670 and c6288, applying this technique results in huge benefits. Without it, circuit c6288 cannot even finish path generation in a reasonable time (12 hours).

If neither the relations of gates technique nor the global longest path generation (R&G) is used, the path generation slows down significantly for some circuits. However, if only one of the techniques is not used, only a 5-10% slowdown is observed. This phenomenon indicates that the benefits from the two techniques greatly overlap for most circuits. It has been observed that in circuits c2670 and c6288 most global long structural paths are untestable. If the global longest path generation is applied, these untestable paths are recognized, and during the path generation for a particular gate, none of these paths need to be recognized again. Similarly, if the relations of gates are applied, these long untestable paths are also recognized only once. For a short untestable path p, the effects are limited because this path may be a long path through gate g_i, assuming all the paths through gate g_i are short paths, but it is not likely to be a long path through another gate g_j. Thus, during the path generation for gate g_j, it is very possible that the K longest testable paths through it have already been found before considering path p. Therefore, the more untestable global long paths, the more benefits from applying the two techniques.

Another interesting phenomenon is that the forward trimming technique is more efficient for the ISCAS85 circuits while the relations of gates technique and the global longest path generation (R&G) are more efficient for the ISCAS89 circuits. The reason is that in most ISCAS89 circuits more than half of the gates are inverters and buffers.

Since these gates have only one input, no path is blocked if a logic value is assigned to their inputs. Thus, the efficiency of the forward trimming technique decreases. However, the efficiency of the R&G increases because the inverters and buffers have only one fan-in gate. If the K longest testable paths have been found for the only fan-in gate and all these paths go through the inverter or buffer, which is very likely, then the path generation for this inverter or buffer can be skipped.

From the analysis, it can be concluded that the forward trimming and smart-PERT techniques reduce the search space and guide the path generation more accurately to generate a testable path, while the relations of gates and the global longest path generation mainly help avoid repeated work.

The effectiveness of the 7-value algebra has been observed on industrial designs with partial scan. As these circuits are large (500 K – 1 M gates), test generation cannot be done within a reasonable time (e.g. 1 week) without the 7-value algebra. It has been observed that for some gates in these circuits, CPU time increases exponentially by turning off this heuristic. However, the 7-value algebra does not help ISCAS85 and ISCAS89 circuits because there are no uncontrollable lines in these circuits.

4.5.2 CPU Time vs. K

Table 7 shows the CPU time when K increases. It can be seen that the CPU time is approximately linear in K, and it increases much more slowly than K. For circuit c7552, for example, there are 2 319 paths generated when K=1 and 19 888 paths when K=10. The number of generated paths when K=10 is more than 8 times that of K=1, but the CPU time increases less than 72%. The reason is that when the first longest path through

a gate is generated, in the path store there are many partial paths almost reaching a capture point. Therefore, the cost of generating more paths is small. Large K values, such as 500, have been tried and it is observed that the CPU time is still linear in K. However, in practice K would not be that large because the test set would be too large.

Table 7. CPU time vs. K (robust).

Circuit	CPU Time (m:s)		
	K=1	K=5	K=10
c1908	0:23	0:26	0:28
c2670	0:12	0:14	0:16
c3540	0:49	1:06	1:28
c5315	0:23	0:29	0:36
c6288	31:33	40:33	45:04
c7552	0:49	1:06	1:24
c9234	1:09	1:29	1:45
c13207	1:36	2:21	3:19
c15850	2:15	3:17	4:15
c38417	5:52	7:48	9:47
c38584	8:01	13:16	16:44

4.5.3 Robust Test for Sequential Circuits

Table 8 shows the information of the sequential circuits used in the experiments. It is assumed that at each fault site there are slow-to-rise and slow-to-fall delay faults. The number of faults is twice the number of lines in a circuit, and the same as the number of transition faults. Column 3 shows the upper bound of detectable faults. This number is less than the total number of faults, because it is also assumed that the primary inputs

cannot change their logic values, and the primary outputs are masked (not observed), due to the constraints from low-cost ATEs. Therefore, no transition can happen at some of the fault sites and some transitions are not observable. Columns 4 and 5 show the number of primary inputs and scan cells for each circuit. Control signals, such as clock and scan enable, are added into the standard ISCAS89 circuits, as primary inputs. It is assumed that the ISCAS89 circuits are full scan and there is only one scan chain for each circuit, in random order. The industrial design *controller1* contains 4 scan chains and *controller2* contains 16 scan chains. Both designs are partial scan. There are 38 non-scanned cells in *controller1* and 5 557 in *controller2*.

Table 8. Summary of sequential circuits used in ATPG experiments.

Circuit	# Lines	UB # Detectable Faults	# Primary Inputs	# Scan Cells
s1423	1 423	2 420	20	74
s1488	1 488	1 310	11	6
s1494	1 494	1 324	11	6
s5378	5 378	7 564	38	179
s9234	9 234	16 166	39	211
s13207	13 207	22 886	65	638
s15850	15 850	24 338	80	534
s35932	35 932	59 246	38	1 728
s38417	38 417	74 926	31	1 636
s38584	38 584	59 454	41	1 426
controller1	86 612	130 692	38	3 503
controller2	1 966 204	1 815 222	201	57 352

Table 9. Summary of robust test generation for sequential circuits.

Circuit	Launch-on-Capture			Launch-on-Shift		
	# Paths Generated	# Test Patterns	CPU Time (m:s)	# Paths Generated	# Test Patterns	CPU Time (m:s)
s1423	395	215	0:13	666	191	0:07
s1488	192	87	0:01	206	81	0:01
s1494	193	85	0:02	204	79	0:01
s5378	1 799	406	0:07	1 110	94	0:04
s9234	2 376	790	3:59	3 608	681	2:52
s13207	3 220	909	2:25	6 469	1 635	1:03
s15850	2 637	472	2:35	5 828	645	1:08
s35932	9 762	36	14:31	12 194	44	8:15
s38417	14 905	949	14:21	17 554	655	2:46
s38584	9 723	526	11:20	21 047	679	4:28
controller1	12 275	2 275	130:10	19 626	657	102:41
controller2	493 779	70 670	132 hrs*	714 116	43 289	57 hrs

*A commercial ATPG tool took >48 hours for transition fault test generation using launch-on-capture for controller2.

Table 9 shows the results for generating the longest robustly-testable path for each fault, under the launch-on-capture and launch-on-shift constraints. Columns 2-4 show the results for the launch-on-capture approach and columns 5-7 for the launch-on-shift approach. Columns 2 and 5 show the number of paths generated by the ATPG. Before test compaction, each generated path has a test pattern. The number of patterns after compaction is shown in columns 3 and 6. The test patterns are compacted by a simple greedy static compaction algorithm, in which each new pattern is combined with the first

compatible existing pattern. Columns 4 and 7 show the CPU time. It can be seen that dealing with uncontrollable signals from non-scanned cells and embedded memories significantly increases the CPU time.

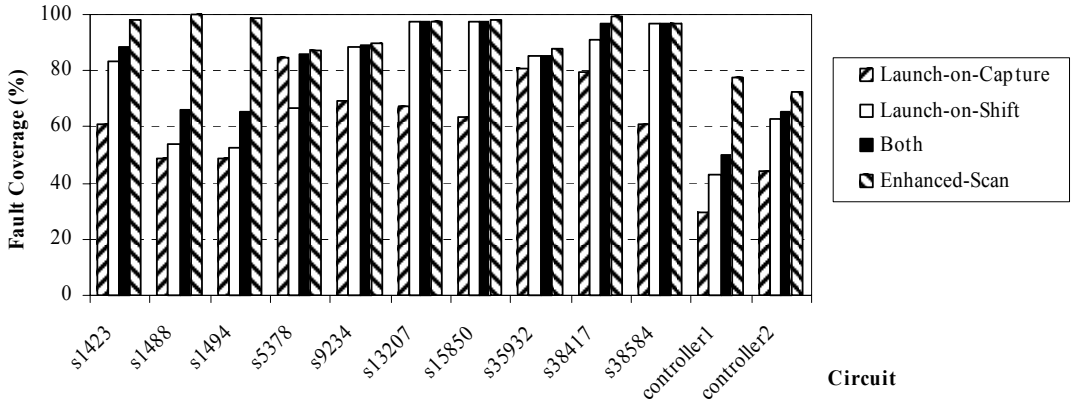


Figure 41. Comparison of robust test fault coverage.

Figure 41 shows the robust test fault coverage (percentage of faults which have at least one path robustly tested) using the launch-on-capture or launch-on-shift approach only, or both. The data in column 3 in Table 8 is used as the total number of detectable faults. For most circuits, the launch-on-shift approach can detect more faults robustly than the launch-on-capture approach, except for circuit s5378, in which the launch-on-capture approach does better. The fault coverage assuming combinational enhanced-scan is shown for comparison. In combinational enhanced-scan, two independent vectors can be stored in scan chains, so the fault coverage for this method is an upper bound. Again, in this mode it is assumed that the primary inputs hold their logic values from the first vector to the second vector, and the primary outputs are masked. Thus, the coverage loss is purely due to the launch-on-capture and launch-on-shift constraints and uncontrollable

values. Although the faults that the launch-on-capture approach cannot detect must be sequentially redundant in functional mode, the test patterns are still useful because these sequentially redundant faults may cause reliability problems.

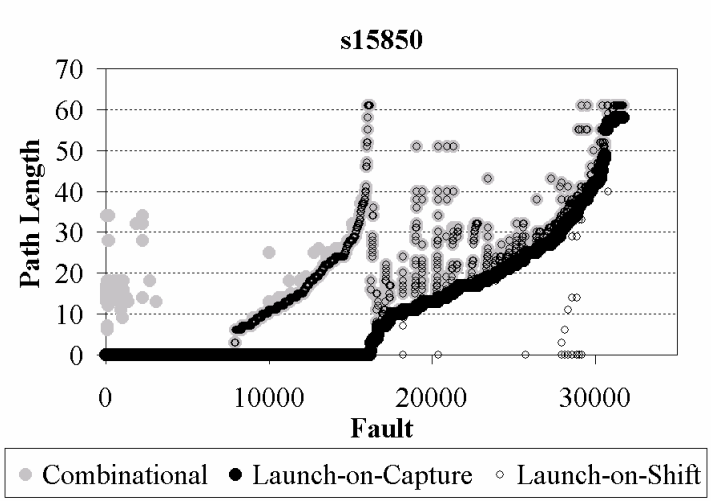


Figure 42. Path length comparison using LOC and LOS.

Figure 42 is the comparison for robustly testable path length using the launch-on-capture and launch-on-shift approaches, for circuit s15850. The faults are indexed so that the length of the longest testable path for each fault, under the launch-on-capture constraints, is in increasing order. The longest robustly testable path for each fault assuming combinational enhanced-scan is also generated for comparison. Because the primary inputs hold and the primary outputs are masked, some faults have no coverage even if the circuit uses combinational enhanced-scan. For most faults, the maximum path length using the launch-on-shift approach is close to the upper bound, but this is not true for the launch-on-capture approach. All the other circuits have similar plots except for circuit s5378. This phenomenon indicates that the constraints from the launch-on-capture approach are stronger than the constraints from the launch-on-shift approach for most

circuits.

4.5.4 Comparison to Transition Fault Tests

As not all transition faults can be tested through a robust or non-robust path, to make KLPG and transition fault test comparable, a test set is constructed as follows:

If a fault (slow-to-rise or slow-to-fall) has robustly testable paths, the longest one is selected, because the delay fault can always be detected regardless of the delay of the other gates or interconnects in the circuit, though this may not be the longest sensitizable path. The results for robust tests are shown in the previous section.

If a fault does not have a robust test, the longest restricted non-robustly testable path is selected, if it exists. The path selection has the following restrictions:

1. The path must be non-robustly testable [20];
2. It must have the required transition at the fault site;
3. The local delay fault must be detected at some capture points, if there is no other delay fault.

In short, if a test is a restricted non-robust test for a fault, it must also be a transition fault test.

For example, path n_1-n_2-n_3-n_5 is a non-robustly testable path in Figure 43. It is a valid non-robust test for line n_2 and n_3. However, it is not a valid non-robust test for line n_5 because the glitch (or transition) may not happen if the delay of path n_1-n_4 is greater than the delay of path n_1-n_2-n_3 (violation of restriction 2). Similarly, it is not a valid non-robust test for line n_1 because the slow-to-rise fault may not be detected even if there are no other delay faults (violation of restriction 3).

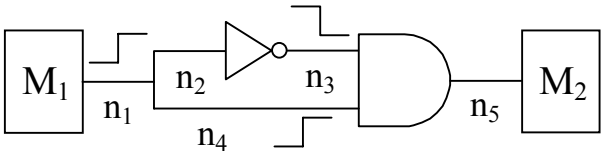


Figure 43. Restricted non-robust test in KLPG.

If a fault does not have a non-robust test, a transition fault test that can detect a small local delay fault is generated. In other words, this test has higher quality than the traditional transition fault test because the traditional one assumes large local delay and propagates the fault through any path (usually a short path). In test generation, this case usually happens when the local delay fault can only be activated or propagated through multiple paths, such as the slow-to-fall fault on line n_2 in Figure 44. The test quality is determined by the length of the shortest paths in the activating or propagating path set. The longer the shortest path is, the smaller the local delay fault that can be detected. The tool does not guarantee that the best transition fault test is generated, in terms of the detected local delay fault size, but the test should be better than the traditional transition fault test. If no high quality test can be found, a transition fault test is generated. This happens for less than 0.05% of the faults in the ISCAS89 test cases.

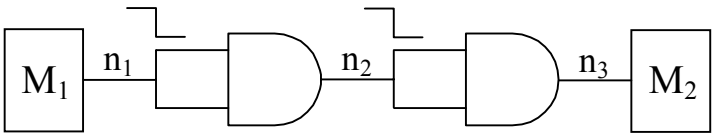


Figure 44. Top-off transition fault test in KLPG.

To keep the test size comparable, K is set to one in this experiment, and the resulting test set is termed a KLPG-1 test set. The KLPG-1 test set is composed of these three

types of path delay fault tests, as shown in Figure 45. It has the same transition fault coverage as the commercial transition fault test set, but has higher quality, in terms of the detected delay fault size.

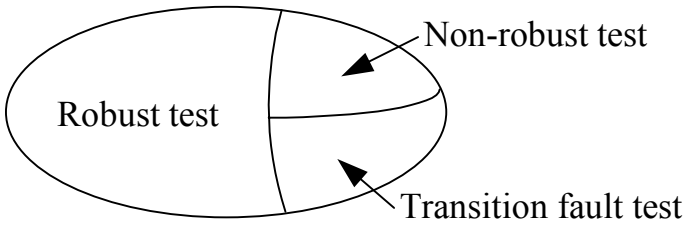


Figure 45. Test composition of KLPG.

Table 10 shows the comparison of test set size, or test volume (number of test patterns) between KLPG-1 test sets and the commercial transition fault test sets, using the launch-on-capture approach. Columns 2-4 show the number of statically compacted robust/non-robust/ transition fault test patterns in KLPG-1 test sets. Column 5 shows the total number of compacted KLPG-1 patterns. If longest paths are not required, some KLPG-1 patterns can be dropped but the transition fault coverage remains the same. Column 6 shows the number of transition fault effective patterns, which detect unique transition faults, out of the KLPG-1 test sets. The number of transition fault test patterns generated and dynamically compacted by the commercial tool is listed in column 7.

For most circuits, the KLPG-1 test sets are 2-3x larger than the commercial transition fault test sets. The KLPG-1 test set for circuit s35932 is smaller, but the KLPG-1 test set for *controller2* is significantly larger. On the other hand, for *controller2*, the number of patterns that have unique transition fault detection is not much larger than the commercial transition fault test set. This phenomenon indicates that many transition

faults in *controller2* are easy-to-detect but testing them through the longest paths results in many more necessary assignments and lower compaction rate. This is likely because the average path contains 40 gates.

Table 10. Comparison of test size KLPG-1 vs. TF (LOC).

Circuit	Robust	Non-Robust	Tran-sition	Total	Effect-ive	Comm-ercial
s1423	215	35	12	262	208	95
s1488	87	40	2	129	119	102
s1494	85	41	2	128	116	101
s5378	406	2	2	410	341	194
s9234	790	69	6	865	697	465
s13207	909	32	117	1 058	612	382
s15850	472	31	4	507	397	231
s35932	36	3	1	39	37	68
s38417	949	51	1	1 001	724	365
s38584	526	443	50	1 019	945	528
controller1	2 275	825	311	3 411	2 600	1 900
controller2	70 670	1 856	4 025	76 551	16 284	11 702

Figure 46 shows the robust/non-robust/transition fault coverage of the KLPG-1 test sets, using launch-on-capture. For example, for circuit s1423, 60.66% of the faults are robustly tested, and 19.01% of the faults are non-robustly only tested. If the launch-on-shift or combinational enhanced-scan approach is used, adding a few transition fault test patterns to the robust test sets results in the same transition fault coverage as the commercial transition fault test sets, for most ISCAS89 circuits. However, this is not true for the launch-on-capture approach. The strong constraints from this approach

prevent many faults from having robustly testable paths. For the industrial designs *controller1* and *controller2*, robustly detected faults are even fewer due to the constraints from non-scanned cells. Figure 46 shows that many more faults are only non-robustly detected or can only be detected through multiple paths (detected by the transition fault test only). For example, in circuit s5378, 354 faults are non-robustly detected but not robustly detected. The number of non-robust test patterns included in the test set is two. This does not indicate that these two test patterns detect the 354 faults, because the 460 robust test patterns also non-robustly detect some other faults by luck. Thus, the two non-robust test patterns for this circuit can be seen as a cleanup phase.

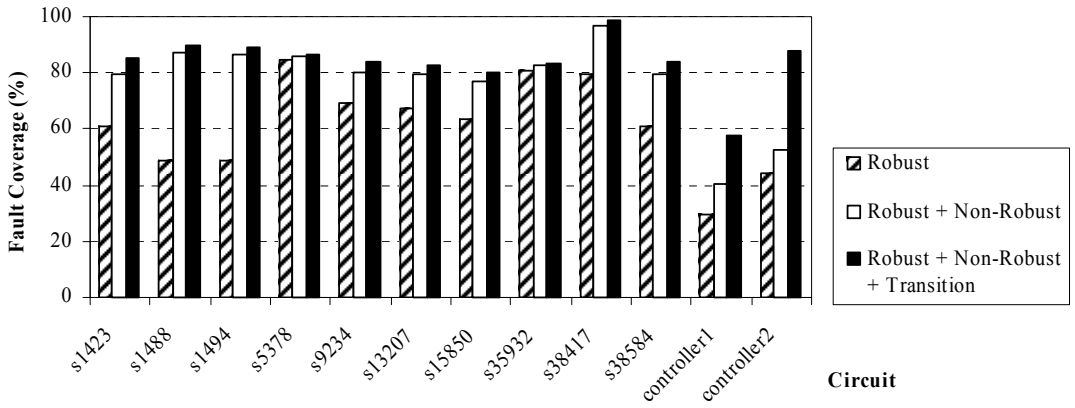


Figure 46. Robust/non-robust/transition fault coverage of KLPG-1 test (LOC).

Although KLPG-1 test has higher quality, in some cases a larger test set is not affordable. Then trade-offs between test quality and test size are necessary in compaction.

Two factors affect the test quality. One is the number of transition faults that are covered by the test set. The other is the path delay through the transition fault sites. For a

fault that has long sensitizable paths, testing a short path is not able to detect a small delay fault. Therefore, a long path whose delay is close to the longest sensitizable path through the fault must be tested. The patterns that test long paths are "must keep" patterns. For a fault that has only short paths, relative to the critical paths, the coverage loss can be neglected if one path, not necessarily the longest one, is tested. The reason is that the probability of a delay fault large enough to cause the longest short path to fail but small enough to cause the short path to pass is small. Thus, for these faults, transition fault tests are good enough if a small test size is required. Given a maximum test set size, first the "must keep" patterns, which test long paths, are selected. Then the remaining patterns are selected based on the number of faults they detect. Most of the remaining transition faults can be detected by randomly filling the don't care bits in the selected patterns.

If the test size is fixed at 6 000 for circuit *controller2*, and all the paths whose delay exceeds 85% of the longest sensitizable path are kept, 3.02% of the non-redundant transition faults are not detected, assuming the commercial tool detects all non-redundant transition faults. This indicates that the transition fault coverage curve must have a long tail. The average care bit density of the compacted test set is 10.2% before randomly filling. Experiments show that if more long paths (with delay >75% of the longest sensitizable path) are kept, 3.05% of the non-redundant transition faults become undetected (an additional 0.03%). Trade-offs between transition fault coverage and path delays can be made accordingly.

4.5.5 Test Generation for Resistive Shorts

Simulation results have shown that the detection probability of resistive shorts using KLPG-1 test is high, though KLPG-1 does not explicitly target resistive shorts. The reason can be found in the following analysis. There are two key factors in detecting a delay fault caused by a resistive short.

1. Sensitization: There must be a transition on one of the shorted lines (victim) and an opposite logic value on the other shorted line (aggressor) to slow down the transition.

2. Observation: The slow transition must propagate through one of the longest paths of the victim line to have high fault coverage [69].

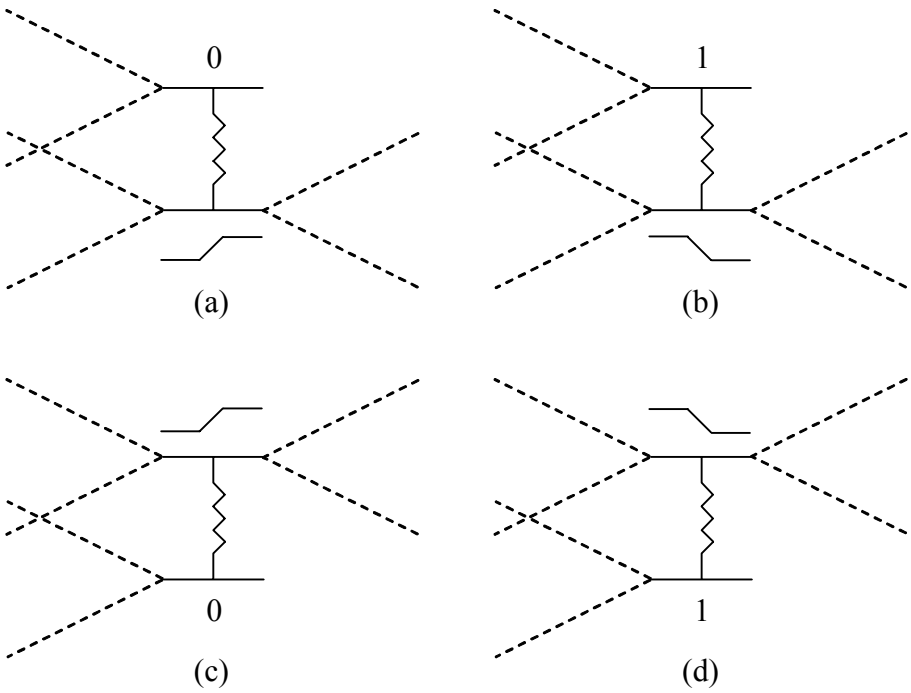


Figure 47. Test generation for resistive shorts.

Fault sensitization is relatively easy. There are four ways to activate a bridging fault, which are shown in Figure 47. KLPG tests ensure the transitions in all the four cases

unless impossible. By randomly filling the don't care bits in the test patterns, opposite logic values may be set by luck. Suppose the probability of a line to be set to logic 1 or 0 is 50%. Then the probability that a bridging fault is activated is $1-(1-0.5)^4$, which is 93.75%. In reality, the probability must be much higher because a line has more than 50% of chance to have a transition or glitch if random test patterns are applied. However, out of the four cases, one of them must have the maximum local delay increase. This case happens when the driver of the victim line, which has a transition, is weak and that of the aggressor line is strong, e.g. a weak rising transition and a strong 0 in Figure 47 case (a). Combined with the path length through the victim line that is also important, one of the four cases must have the highest fault coverage for the resistive short.

However, to determine which case has the highest fault coverage can be expensive because it depends on both local delay increase and the delay of the longest path through the victim line, and the two factors are dependent on each other. Simulation results have shown that to get high fault coverage, generating long paths is more important and much harder than setting a logic value on the aggressor [89]. Therefore, the strategy is to generate longest paths first, and then try to activate the aggressor, for all the four cases, so that the best test is guaranteed.

The algorithm works as follows: during the KLPG test generation, when a longest path is generated for either shorted lines, an opposite value is set on the other line and the final justification is run again. It has been observed that in most cases only don't care bits are involved. Because KLPG test patterns are required to detect resistive opens anyway, there is no additional cost to test resistive shorts by filling don't care bits. In

some cases, setting don't care bits reduces the compaction rate so that the test volume goes up, but the increase is very small. In few cases that the aggressor cannot be set because it conflicts against the requirements to test the longest path, path generation starts over with a preset aggressor value, to get a constrained longest path. These patterns can be seen as top-off patterns but they may not necessarily increase the volume after compaction.

Table 11. Test generation summary for resistive shorts.

Circuit	# Lines	# Shorts	Robust KLPG-1 with ATPG for Shorts		Robust KLPG-1 w/o ATPG for Shorts	
			# Test Patterns	CPU Time (m:s)	# Test Patterns	CPU Time (m:s)
s1423	1 423	2 846	233	0:14	215	0:13
s1488	1 488	2 976	99	0:01	87	0:01
s1494	1 494	2 988	89	0:02	85	0:02
s5378	5 378	10 756	441	0:07	406	0:07
s9234	9 234	18 468	815	4:05	790	3:59
s13207	13 207	26 414	1 006	2:30	909	2:25
s15850	15 850	31 700	520	2:45	472	2:35
s35932	35 932	71 864	43	15:51	36	14:31
s38417	38 417	76 834	1 061	15:03	949	14:21
s38584	38 584	77 168	589	12:00	526	11:20

Table 11 shows the test generation results for resistive shorts. ISCAS89 circuits are used and robust test generation using launch-on-capture is assumed. Random non-feedback shorts are used. The number of shorts is twice the number of lines in the circuits and it equals the number of transition faults. Again, as the experiments for fault

simulation, shorts between lines feeding the same gate are not included, and shorts between signal lines and power/ground grid are not included. The results of robust KLPG-1 test without ATPG explicitly for resistive shorts are listed in columns 6-7 for comparison. It can be seen that after resistive shorts are included into the fault space the test volume does not increase significantly (min: 4.7% max: 19.4%) though the fault population doubles. The CPU time does not increase significantly either (max: 9.2%). The reason is that to put an opposite value on an aggressor line is much easier than to sensitize a long path.

5. EXPERIMENTS ON SILICON

5.1 Philips Experiments

The design selected for the experiments is a digital signal processor (DSP) core. The DSP core is a 180 nm technology design and contains 39.713 gates. This is a full scan design, with 2 290 muxed scanned flip-flops in 13 scan chains. This design is for research purpose only and it has no specified speed. Speed binning has been performed to evaluate the test quality of the KLPG test, compared to the traditional transition fault test.

A KLPG-1 test is generated by our KLPG tool by setting K=1 [90][91], which tests two longest non-robustly testable paths through each potentially faulty line, with one path having a rising transition at the fault site and the other having a falling transition. If there is no non-robustly testable path through a potentially faulty line, the tool generates a transition fault test that tries to propagate the transition through a set of long paths. Therefore, the transition fault coverage of KLPG-1 is the same as that of the traditional transition fault test. CPU time for test generation is 9 hours 14 minutes, and 34 621 testable paths are generated.

Table 12. Test volume comparison in Philips experiments.

Test	# of patterns
TF	1 820
KLPG-1	13 650

Table 12 shows the test volume of the two test sets. KLPG-1 has a significantly larger

test volume. This is partly because our KLPG tool uses static compaction [92][93], while the transition fault tests use very sophisticated dynamic compaction algorithms [94][95][96][97]. The second reason is that fault dropping in KLPG test generation is very rare, because a fault is dropped only if a path is proved the longest sensitizable one for this fault. However, in transition fault test generation, fault dropping is common by re-simulating generated patterns.

Table 13. Fmax comparison of KLPG-1 and transition fault tests.

Device	Fmax (MHz)	
	TF	KLPG-1
FDL-2	88.26	77.58
FDL-3	81.23	74.57
FDL-4	77.58	72.41
FDL-5	69.88	62.23
FDL-6	72.89	69.20
FDL-7	80.13	78.00
FDL-9	94.43	84.75
GDL-3	32.54	30.06
GDL-4	42.57	40.88
GDL-6	32.24	29.52
GDL-7	58.07	57.27

There are 20 devices selected in the experiments. Out of these devices, 10 have comparatively large delay faults ($\geq$ 5 ns more than nominal path delay) and the other 10 are suspected to have small delay faults ($\leq$ 5 ns more than nominal path delay). Small delay faults are also termed fine delay faults in Philips. Table 13 shows the speed

binning results. Devices FDL-x are the ones that are suspected to have small delay faults (or fine delay faults) and devices GDL-x are the ones with large delay faults, or gross delay faults. Significant Fmax difference between the transition fault and KLPG tests has been observed (max: 13.1%).

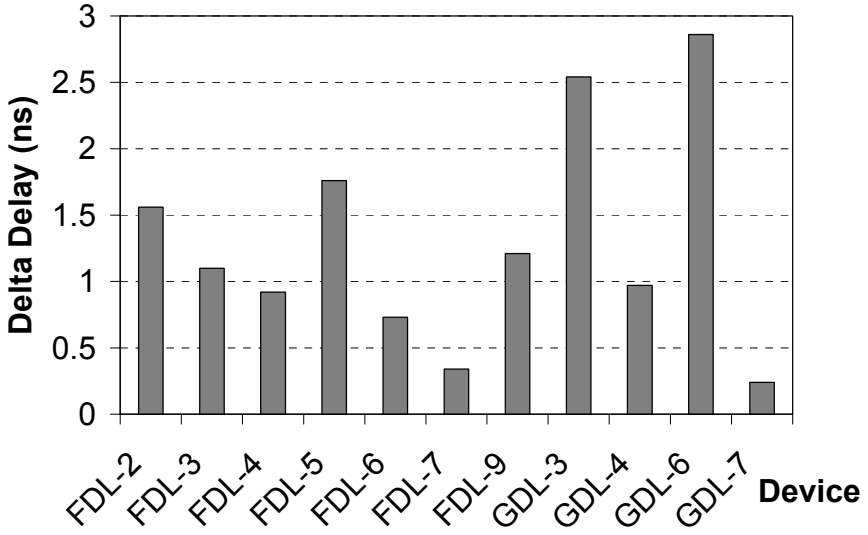


Figure 48. KLPG tests sensitize longer paths than transition fault tests.

Figure 48 shows the difference of delays using KLPG-1 and transition fault tests. Delta delay = (Delay using KLPG-1 test – Delay using transition fault test). For example, the delay for device GDL-6 is 31.02 ns using the transition fault test, and it is 33.88 ns using the KLPG-1 test. The delta delay is 2.86 ns. If the nominal delay of the longest path is 10 ns, then device GDL-6 may contain a defect that causes 23.88–31.02 ns extra delay, assuming both transition fault and KLPG-1 tests sensitize the fault and cause equal extra delays. Under these assumptions, the transition fault test sensitizes a 0-7.14 ns path while the KLPG-1 test sensitizes a 2.86-10 ns path, and the difference is always

2.86 ns, which is an increase of more than 39.5%. It is also possible that both tests sensitize a bridging fault and the KLPG-1 test introduces larger extra delay than the transition fault test, which is consistent with the simulation results.

5.2 Texas Instruments Experiments

The design selected for the experiments is a controller with 5 clock domains. The fastest clock domain is 250 MHz. Delay tests are applied to the 250 MHz domain only. The controller is a 130 nm technology ASIC design and contains 738 K gates. The logic within the 250 MHz domain contains 597 K gates, not including embedded memories. According to the test configuration, the controller is a partial scan design and contains 8 scan chains for the 250 MHz domain, with 14 963 muxed scanned flip-flops in total, and these flip-flops are clocked with the system clock under both shift and functional mode. At-speed test can only be performed using launch-on-capture.

There are 24 devices included in the experiments. These devices marginally pass the transition fault test of the 250 MHz domain in production. Four delay test sets are used in the experiments:

1. (Critical) path delay test (generated by the KLPG tool, by removing the constraints that every gate/line must be covered), which tests 2 137 critical paths (longest non-robustly sensitizable paths) in the 250 MHz domain;
2. Regular transition fault test (generated by a commercial tool), which is the delay test in production;
3. Randomized transition fault test (generated by the same commercial tool), which

makes random decisions and does not necessarily propagate transition faults through the easiest paths;

4. KLPG-1 test (generated by the KLPG tool).

Table 14 shows the test volume of each test. The volume of the randomized transition fault test is slightly higher than that of the regular one. Again, KLPG-1 has a significantly larger test volume.

Table 14. Test volume comparison in Texas Instruments experiments.

Test	# of patterns
Path delay test	774
Regular TF	1 445
Randomized TF	1 471
KLPG-1	12 579

Figure 49 shows the speed binning results of the four delay test methodologies. It can be seen that for most devices, KLPG has the lowest Fmax, which means testing longer paths does result in slower speed. KLPG has two unique detects: devices 6 and 14. They pass the other tests but fail the KLPG test, given the 250 MHz specification. Devices 3, 4 and 24 do not fail any test, but KLPG is significantly slower. For these devices, the path delay test does not result in low Fmax, which indicates that these devices are likely to contain small defects that have no critical paths going through and KLPG detects them through longer paths than the transition fault tests do. These paths may still be much shorter than any critical paths. The results prove the benefit of testing the longest paths through each gate/line.

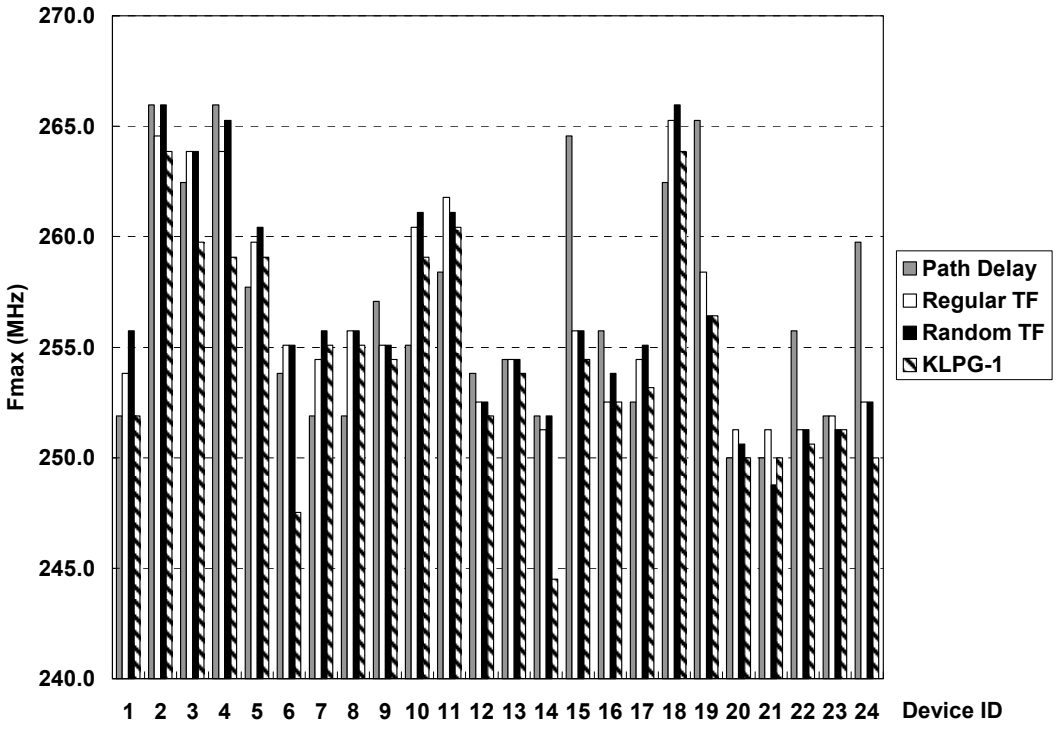


Figure 49. Fmax comparison using four delay test methodologies.

Randomized transition fault test does not seem to have significant benefits. Compared to the regular transition fault test, it results in slightly lower Fmax for five devices but slightly higher Fmax for 10 devices. Interestingly, it gets a unique detect (device 21) though the difference from KLPG or path delay test is very small. The test may sensitize a path or a set of paths that is longer than the paths generated by KLPG, and it can be more likely if a delay fault can only be detected through a set of paths, because KLPG deterministically propagates transitions through single long paths. Alternatively, it can be a fault that is not modeled in KLPG, such as a resistive bridging fault, which is fortuitously detected by the randomized transition fault test. It can also be a power supply noise issue, because transition fault tests are highly compacted so that more

switches are activated by a single test pattern, which may result in higher power supply noise and slower paths.

It is interesting to look at the path delay test results. For ~1/3 of the devices, the path delay test results in the lowest Fmax. This is straightforward because if a device is defect-free, or the device contains only undetectable faults, e.g. all paths through a small defect are very short so that no path is slow (this fault can be detected using higher-than-functional-speed test), the actual longest path is the one that determine the speed of the circuit. Because of process variation, many critical paths can be the actual longest path. KLPG-1 tests the longest path (one that has the highest probability to be the actual longest path) through each potential fault site, but the number of globally longest paths may not be enough. The path delay patterns test 2 137 critical paths so that it is much more likely to hit the actual longest one. Devices 5, 7, 8, 10, 11, 17, 18 and 20 are likely to be in this category. Theoretically, if more critical paths are tested, almost all defect free devices should drop into this category.

However, Fmax of the path delay test for devices 15, 19, 22 and 24 is significantly higher than that of the other tests. Possible reasons can be:

1. Huge process variation;
2. Small defects that the path delay test misses;
3. Fortuitous detects by other tests;
4. Lower power supply noise than other tests.

Huge process variation can cause the path delay test to miss the actual longest path, which may be sensitized by other tests by luck simply because they have more test

patterns. However, this is not very likely by looking at the behavior of other devices in our experiments.

Small defects are the most likely reason because both transition and KLPG tests cover the whole circuit while the path delay test does not. All the critical paths that are sensitized by the path delay patterns can be within a small area of the circuit. If a small defect exists out of this area, the path delay test may miss it while it can be easily detected by KLPG, and sometimes by the transition fault test either.

If the small defect behave as a bridging fault that only causes extra delay, it is also a fortuitous detect by the other three tests as none of them targets bridging faults explicitly. In fact, KLPG is good at catching resistive bridging faults fortuitously, as the simulation results show.

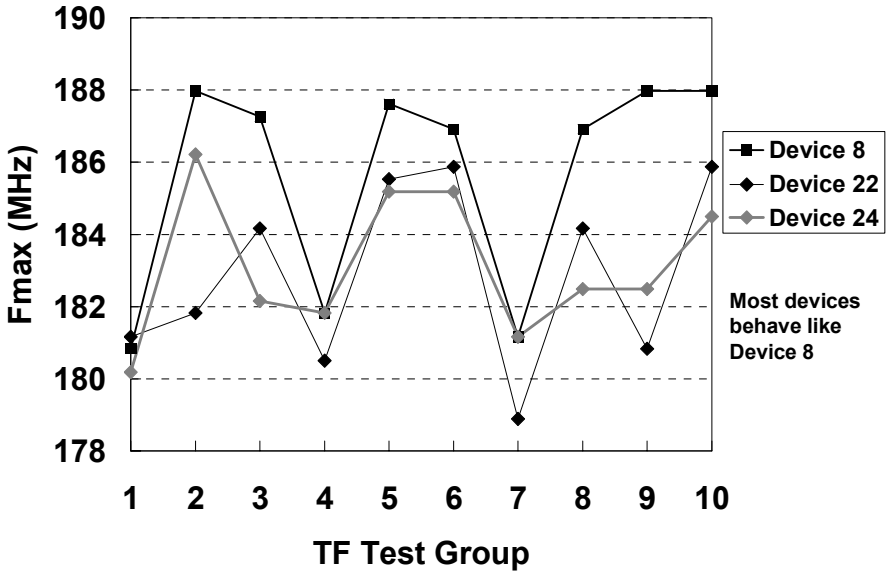


Figure 50. "Signatures" of devices 22 and 24.

An experiment of "Fmax signature" is performed on devices 15, 19, 22 and 24. The

transition test patterns are divided evenly into 10 groups by their original order, and speed binning is run for each group, at a lower VDD (1.2V, compared to the nominal 1.5V) to exaggerate the difference. Figure 50 shows Fmax curves for devices 22 and 24, and the curves can be considered "signatures". More than 50% of the devices behave like device 8, with an "M-V" shaped signature. However, devices 22 and 24 behave differently. Device 22 is slower than expected for pattern groups 2 and 9. It is likely to contain a small defect that is only sensitized by some patterns in these two groups, but not by the path delay test. Similarly, device 24 seems to have a small defect too because it is slow for groups 3, 8 and 9.

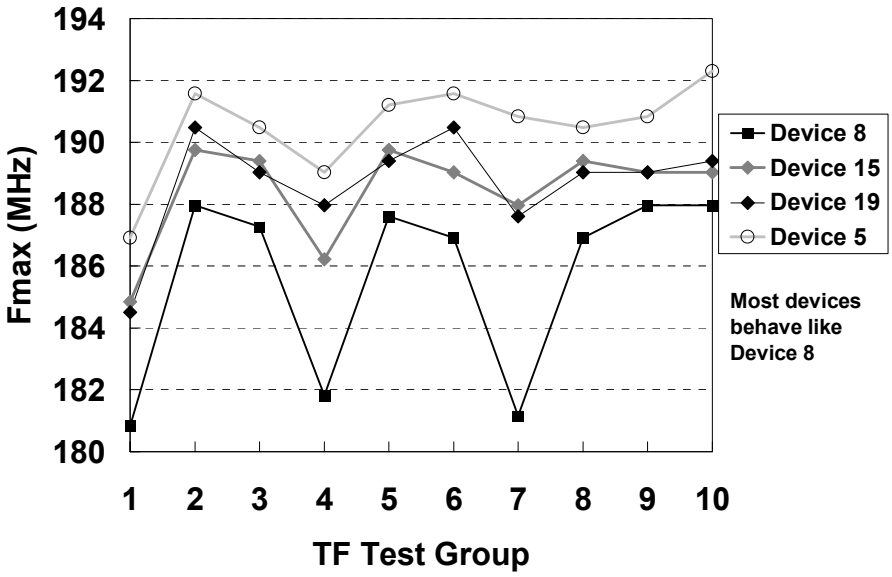


Figure 51. "Signatures" of devices 15 and 19.

However, devices 15 and 19 do not have obviously abnormal signatures, as shown in Figure 51. They behave slightly differently from device 8, which represents more than 50% of the devices but similarly to device 5, which seems to be a good device according

to the results shown in Figure 49. The signatures of devices 15 and 19 are completely bounded by the signatures of devices 5 and 8. It is hard to say whether they contain small defects or not from their signatures.

Power supply noise may be another reason [98][99]. Due to this reason, paths under the transition fault tests may be slower than paths under the path delay test. This is because: care bit density of the transition fault test patterns is significantly higher than that of the path delay test patterns ➔ a transition fault test pattern may cause more switching activities in the circuit ➔ it is likely to have higher power supply noise ➔ paths sensitized by a transition fault test pattern become significantly slower than their nominal delays due to the power supply noise [100]. This can be more sensitive as power supply voltage decreases in deep sub-micron (DSM) designs [101].

Figure 52 is the care bit density comparison of the path delay test and transition fault test without random fill. Every point represents the average number for 50 patterns. Because the care bit density is unknown with regular setting (with random fill) in the commercial tool, the random fill switch is turned off in this experiment. However, this introduces a 1.6x test volume increase (1 445 patterns with random fill turned on vs. 2 331 patterns with random fill turned off). Average care bit density of the transition fault test without random fill is 4.59%, and it can be 1.6x higher, which is ~7.4%, for the transition fault test with random fill. In reality, this number can be slightly lower than 7.4% because some care bits overlap during compaction (it is not always a 1 or 0 compacted with an X). However, this number is close to the production transition fault test value, which really matters.

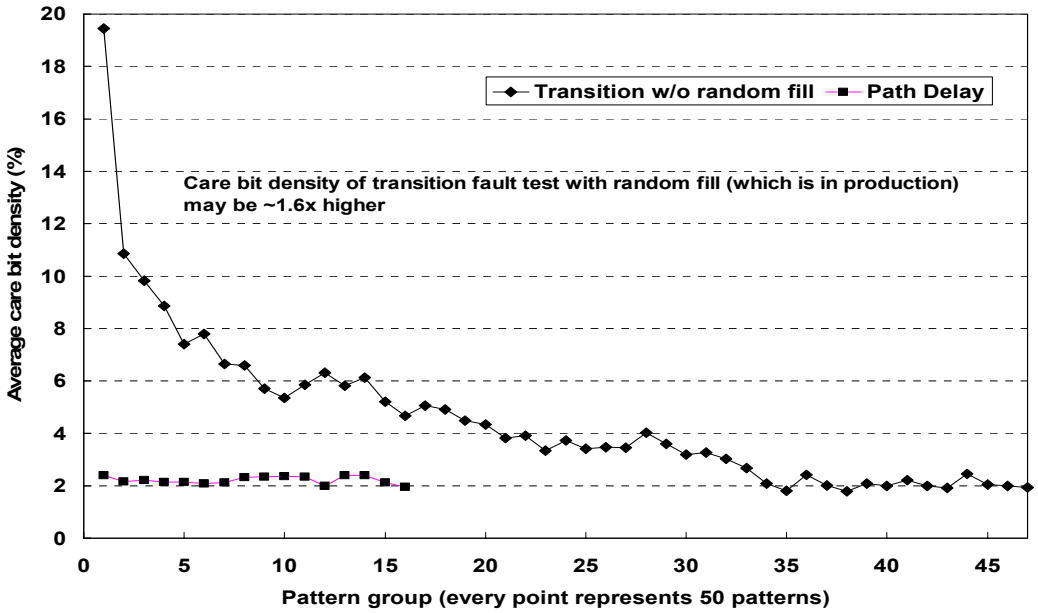


Figure 52. Comparison of care bit density (TF vs. path delay).

Compared to the high care bit density of the transition fault test, it is significantly lower for the path delay test (2.23%). Furthermore, the maximum care bit density of the transition fault test pattern (without random fill) is 38.66% (pattern #3), which is much higher than that of the path delay test patterns (3.91%). On average, each path delay test pattern tests only 2.74 paths. It is reasonable to expect a higher level of power supply noise introduced by a transition fault test pattern.

Figure 53 shows the comparison between the transition fault test (without random fill) and the KLPG-1 test. The average care bit density of the KLPG-1 test is 2.60% and the maximum is 12.09% (pattern #1). Like the path delay test, the care bit density of the KLPG-1 test does not drop significantly from the first pattern to the last, which happens to most structural tests like stuck-at and transition fault tests. From this data, it seems that the KLPG-1 test does not have high power supply noise either, if all of our

assumptions are valid. However, the KLPG-1 test does well in the Fmax comparison. Therefore, power supply noise may not be the main reason that Fmax of the path delay test is significantly higher for four devices, or our assumptions that the transition fault test cause much more transition activities are invalid for this device.

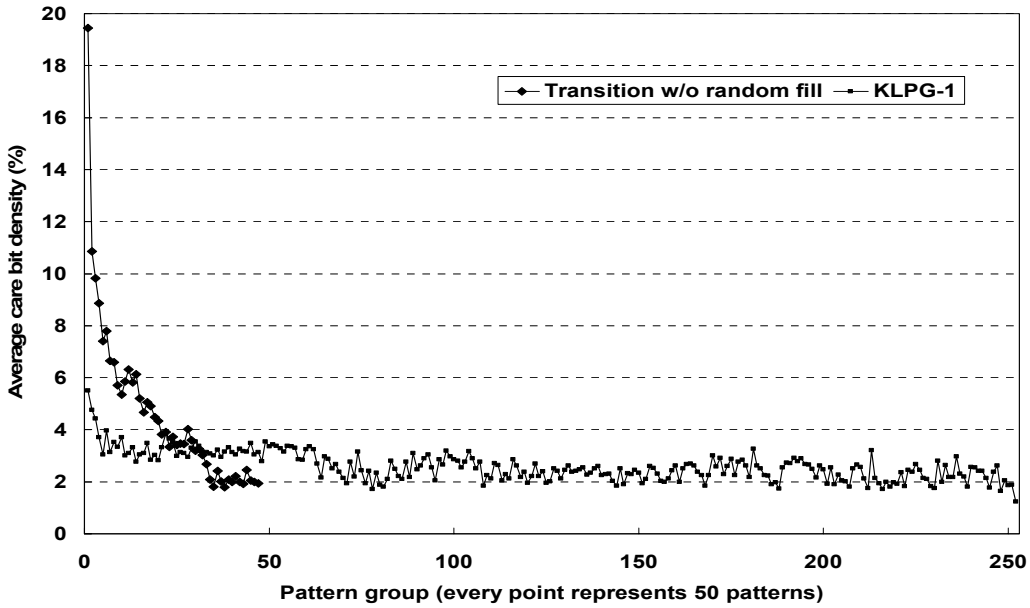


Figure 53. Comparison of care bit density (TF vs. KLPG-1).

To get additional data, the following experiment was performed. The first 200 transition fault test patterns in production are selected as "noise" source, because these patterns are assumed to have the highest care bit density so that they are likely to be the "noisiest". The original path delay test patterns are considered "quiet" due to the large amount of don't care bits (min: 96.08% max 98.90%). For each "quiet" path delay test pattern, one of the "noisy" transition fault test patterns that has fewest conflicting bits is selected as the noise source for that "quiet" path delay test pattern. The two patterns are then "compacted" together. Figure 54 shows an example. If two patterns contain

conflicting bits, such as input bit #4 in this example, the path delay test pattern bit is kept, so that the target paths of the path delay test pattern are still sensitized. In this way, "noise" is introduced into the "quiet" pattern because more switching activities may happen in the circuit. The expected output of the "noisy" path delay test pattern keeps the same as that of the "quiet" pattern. Bits that are not the destinations of the target paths are all masked. This is because if other paths are fortuitously sensitized, it may result in an unfair comparison between "noisy" and "quiet" path delay test patterns.

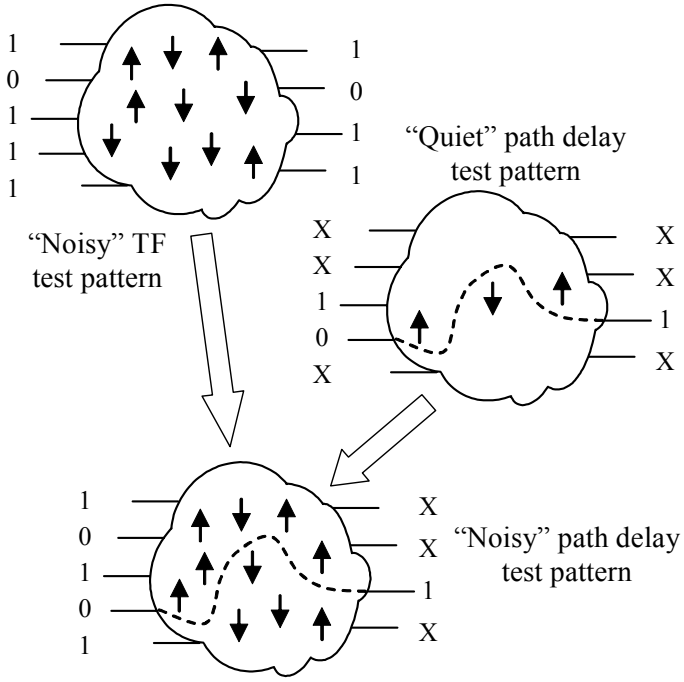


Figure 54. Making path delay tests more "noisy".

Figure 55 shows the Fmax comparison of the "noisy" path delay test and variety of tests that have different noise level: tests that fill the don't care bits with random values, all 1's-fill and all 0's. The shift frequency is 25 MHz so that the voltage drop during launch and capture caused by shift can be neglected. The first four devices (devices 15,

19, 22, 24) are the ones that have significantly higher Fmax using randomly filled path delay test. This experiment uses a low VDD (1.2V, compared to the nominal 1.5V) to exaggerate the difference. The other devices (devices 101-107) are slow devices only under low VDD (these devices are separate from the 24 devices that are used in previous experiments).

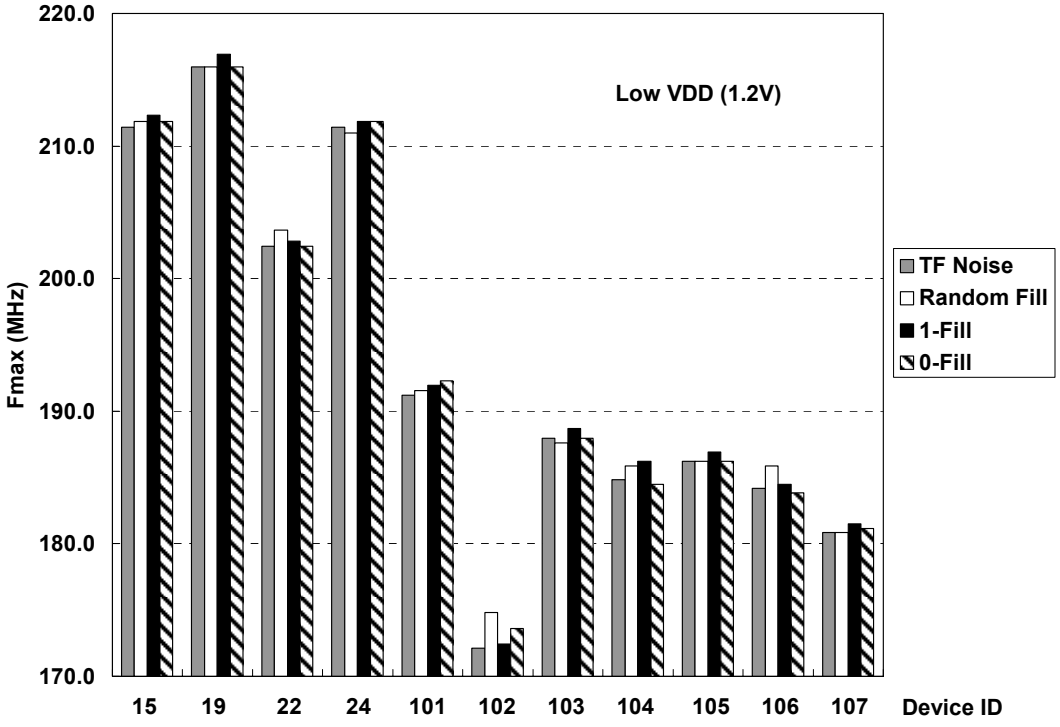


Figure 55. Fmax comparison of "noisy" and "quiet" path delay tests.

It can be seen that for most devices "noisy" pattern do result in lower Fmax, but the difference is small. The most possible reason is that our assumption that the transition fault patterns with high care bit density make more switching activities may be wrong. For 7 out of 11 devices, Fmax with 0-fill is equal or lower than Fmax with random fill. This breaks our assumption that 1-fill or 0-fill makes fewer switches in the circuit than

random fill. The number of switching activities in these devices may not vary significantly with different fill methods. As a result, the path delay test compared in Figure 49 may not be as “quiet” as assumed. From the data, the conclusion is that these four devices are more likely to contain small delay defects that the path delay test misses.

In summary, Figure 49 shows that Fmax of applying KLPG-1 alone is close to the lower bound. It can be even closer if KLPG-1 is combined with the path delay test. However, test volume is a concern at this moment. Before powerful compaction methodologies are developed for KLPG, combining the transition fault and path delay tests may be a good choice. It does better than applying the transition fault test alone and the test volume is still kept at a low level. Alternatively, the transition fault test can be the top-off test for the path delay test; to get a lower combined test volume.

6. SUMMARY AND FUTURE WORK

6.1 Summary

In this work a new delay test methodology has been developed. This includes a new delay fault model, a statistical delay fault coverage metric, and fault simulation and test generation algorithms.

Traditional delay fault models are incomplete in that they only model a subset of delay defect behaviors. To overcome this problem, a physically realistic combined delay fault model (CDF) incorporating both spot defects and parametric process variation is presented in this dissertation. Spot defects are modeled as both resistive opens and shorts.

A simulation-based statistical delay fault coverage metric was developed according to the new model. Traditional delay fault coverage metrics are either incomplete (transition fault coverage metric) or too pessimistic (gate and path delay fault coverage metrics) so that the realistic test quality cannot be accurately evaluated. The new fault coverage metric considers the combined effects of spot defects and process variation, and takes advantage of inter-die process correlation so that the coverage is much closer to the real test quality. A fast simulation algorithm, which is used in the fault coverage computation, is also presented in this dissertation. The simulation results have suggested a strategy of high quality and low cost delay fault test development, which is the ultimate goal of this work.

To achieve this, an efficient algorithm that generates the K longest testable paths through each gate/line (KLPG) has been developed. Both launch-on-shift and launch-on-

capture at-speed test approaches for sequential circuits are supported. Many false path elimination techniques are used in the test generation. Experimental results have shown that these techniques efficiently trim non-solution search space, which is the main problem of any test generation algorithms. To our knowledge, this tool is faster than any existing tools solving similar problems, and it is the first tool that efficiently generates the longest testable path through each gate in ISCAS85 circuit c6288, which is known as a hard problem due to the fact that this circuit contains an exponential number of paths and more than 99% of the paths are untestable.

A complete test generation flow has been implemented on industrial designs. KLPG tests have been compared to a variety of traditional delay fault tests on silicon. The silicon data match the theoretical prediction and simulation results developed in this work, which indicates the accuracy of the new delay fault model and fault coverage metric. The silicon data has also shown significant test quality increase of the KLPG tests, as predicted in the simulation.

6.2 Future Work

Although KLPG significantly increase delay fault test quality, its test size is a concern of industrial designs. One of the reasons is that KLPG uses static compaction while the traditional transition fault test uses sophisticated dynamic compaction. Experiments have shown that the care bit density in KLPG test patterns is significantly lower than that in transition fault test patterns, and this indicates a potentially much higher compaction rate for KLPG tests.

Noise can have increasing impact on circuit delays as transistor dimensions shrink [102]. Noise can be induced by capacitive or inductive coupling [55][57], heat [103], or power supply noise [98][99]. Some algorithms have been developed to generate the worst-case noise [54][56]. In fact, the future test generation and compaction algorithms should keep noise at a reasonably high level rather than maximize it, which may result in yield loss.

REFERENCES

[1] G. Aldrich and B. Cory, "Improving Test Quality and Reducing Escapes," *Proceedings of Fabless Forum*, Fabless Semiconductor Association, vol. 10 no. 1, 2003, pp. 34-35.

[2] P. Nigh, D. Vallett, A. Patel, J. Wright, F. Motika, D. Forlenza, R. Kurtulik and W. Chong, "Failure Analysis of Timing and I_{DDQ}-Only Failures from the SEMATECH Test Methods Experiment," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1998, pp. 43-52.

[3] C. F. Hawkins, A. Keshavarzi, and J. M. Soden, "Reliability, Test and I_{DDQ} Measurements," in *Proc. IEEE International Workshop on I_{DDQ} Testing*, Washington, DC, Nov. 1997, pp. 96-102.

[4] X. Lin, R. Press, J. Rajski, P. Reuter, T. Rinderknecht, B. Swanson and N. Tamarapalli, "High-Frequency, At-Speed Scan Testing," *IEEE Design & Test of Computers*, vol. 20, no. 5, Sept.-Oct. 2003, pp. 17-25.

[5] A. Carbine and D. Feltham, "Pentium Pro Processor Design for Test and Debug," in *Proc. IEEE International Test Conference*, Washington, DC, Nov. 1997, pp. 294-303.

[6] T. W. Williams, R. Kapur, M. R. Mercer, R. H. Dennard and W. Maly, "I_{DDQ} Test: Sensitivity Analysis of Scaling," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1996, pp. 786-792.

[7] K. S. Kim, S. Mitra and P. G. Ryan, "Delay Defect Characteristics and Testing Strategies," *IEEE Design & Test of Computers*, vol. 20, no. 5, Sept.-Oct. 2003, pp. 8-16.

[8] J. Gatej, L. Song, C. Pyron, R. Raina, T. Munns, "Evaluating ATE Features in Terms of Test Escape Rates and Other Cost of Test Culprits," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 1040-1048.

[9] J. C. M. Li and E. J. McCluskey, "Diagnosis of Sequence Dependent Chips," in *Proc. IEEE VLSI Test Symposium*, Monterey, CA, Apr. 2002, pp. 187-192.

[10] *International Technology Roadmap for Semiconductors (ITRS)*, San Jose, CA, Semiconductor Industries Association, 2003.

[11] S. R. Nassif, "Modeling and Analysis of Manufacturing Variations," in *Proc. IEEE Custom Integrated Circuits Conference*, San Diego, CA, May 2001, pp. 223-228.

[12] X. Lu, Z. Li, W. Qiu, D. M. H. Walker and W. Shi, "PARADE: PARAmetric Delay Evaluation Under Process Variation," in *Proc. International Symposium on Quality Electronic Design*, San Jose, CA, Mar. 2004, pp. 276-280.

[13] S. R. Nassif, "Design for Variability in DSM Technologies," in *Proc. International Symposium on Quality Electronic Design*, San Jose, CA, Mar. 2000, pp. 451-454.

[14] V. Mehrotra, S. L. Sam, D. Boning, A. Chandrakasan, R. Vallishayee and S. Nassif, "A Methodology for Modeling the Effects of Systematic Within-Die Interconnect and Device Variation on Circuit Performance," in *Proc. ACM/IEEE Design Automation Conference*, Los Angeles, CA, Jun. 2000, pp. 172-175.

[15] R. B. Brawhear, N. Menezes, C. Oh, L. T. Pillage and M. R. Mercer, "Predicting Circuit Performance Using Circuit-level Statistical Timing Analysis," in *Proc. ACM/IEEE European Conference on Design Automation*, Paris, France, Mar. 1994, pp. 332-337.

[16] E. Acar, S. N. Nassif, L. Ying and L. T. Pileggi, "Assessment of True Worst Case Circuit Performance Under Interconnect Parameter Variations," in *Proc. International Symposium on Quality Electronic Design*, San Jose, CA, Mar. 2001, pp. 431-436.

[17] A. Gattiker, S. Nassif, R. Dinakar and C. Long, "Timing Yield Estimation from Static Timing Analysis," in *Proc. International Symposium on Quality Electronic Design*, San Jose, CA, Mar. 2001, pp. 437-442.

[18] Z. Barzilai and B. K. Rosen, "Comparison of AC Self-Testing Procedures," in *Proc. IEEE International Test Conference*, Philadelphia, PA, Oct. 1983, pp. 89-94.

[19] J. L. Carter, V. S. Iyengar and B. K. Rosen, "Efficient Test Coverage Determination for Delay Faults," in *Proc. IEEE International Test Conference*, Washington, DC, Sept. 1987, pp. 418-427.

[20] G. L. Smith, "Model for Delay Faults Based Upon Paths," in *Proc. IEEE International Test Conference*, Philadelphia, PA, Oct. 1985, pp. 342-349.

[21] K. Heragu, V. D. Agrawal and M. L. Bushnell, "Statistical Methods for Delay Fault Coverage Analysis," in *Proc. VLSI Design Conference*, New Delhi, India, Jan. 1995, pp. 166-170.

[22] J. Waicukauski, E. Lindbloom, B. K. Rosen and V. S. Iyengar, "Transition Fault Simulation," *IEEE Design & Test of Computers*, vol. 4, no. 5, Apr. 1987, pp. 32-38.

[23] K. T. Cheng, S. Devadas and K. Keutzer, "Delay-Fault Test Generation and Synthesis for Testability Under a Standard Scan Design Methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 8, Aug. 1993, pp. 1217-1231.

[24] K. T. Cheng, "Transition Fault Testing for Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 12, Dec. 1993, pp. 1971-1983.

[25] K. T. Cheng, "Test Generation for Delay Faults in Non-Scan and Partial Scan Sequential Circuits," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, Santa Clara, CA, Nov. 1992, pp. 554-559.

[26] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth and D. Wheater, "A SmartBIST Variant with Guaranteed Encoding," in *Proc. IEEE Asian Test Symposium*, Kyoto, Japan, Nov. 2001, pp. 325-330.

[27] S. Mitra and K. S. Kim, "X-Compact: An Efficient Response Compaction Technique for Test Cost Reduction," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 311-320.

[28] S. Mitra and K. S. Kim, "XPAND: Deterministic Test Generation with Very Few Scan Pins and Lots of Scan Chains," *Technical Report*, Santa Clara, CA, Intel Corporation, 2003.

[29] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K. H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eide and J. Qian, "Embedded Deterministic Test for Low Cost Manufacturing Test," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 301-310.

[30] B. R. Benware, R. Madge, C. Lu and R. Daasch, "Effectiveness Comparisons of Outlier Screening Methods for Frequency Dependent Defects on Complex ASICs," in *Proc. IEEE VLSI Test Symposium*, Apr. 2003, pp. 39-46.

[31] R. R. Montanes and J. P. Gyvez, "Resistance Characterization for Weak Open Defects," *IEEE Design & Test of Computers*, vol. 19, no. 5, Sept. 2002, pp. 18-25.

[32] E. S. Park, M. R. Mercer and T. W. Williams, "Statistical Delay Fault Coverage and Defect Level for Delay Faults," in *Proc. IEEE International Test Conference*, Washington, DC, Sept. 1988, pp. 492-499.

[33] C. W. Tseng and E. J. McCluskey, "Multiple-Output Propagation Transition Fault Test," in Proc. in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2001, pp. 358-366.

[34] V. Iyengar, B. K. Rosen and I. Spillinger, "Delay Test Generation 1 – Concepts and Coverage Metrics," in *Proc. IEEE International Test Conference*, Washington, DC, Sept. 1988, pp. 857-866.

[35] T. W. Williams, B. Underwood and M. R. Mercer, "The Interdependence Between Delay-Optimization of Synthesized Networks and Testing," in *Proc. ACM/IEEE Design Automation Conference*, San Francisco, CA, June 1991, pp. 87-92.

[36] G. M. Luong and D. M. H. Walker, "Test Generation for Global Delay Faults," in *Proc. IEEE International Test Conference*, Washington, DC, Oct. 1996, pp. 433-442.

[37] W. Maly, "Computer-Aided Design for VLSI Circuit Manufacturability," *Proceedings of the IEEE*, vol. 78, no. 2, Feb. 1990, pp. 356-392.

[38] D. M. H. Walker, "Tolerance of Delay Faults," in *Proc. IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, Dallas, TX, Nov. 1992, pp. 207-216.

[39] M. Sivaraman and A. J. Strojwas, "Delay Fault Coverage: A Realistic Metric and an Estimation Technique for Distributed Path Delay Faults," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, Nov. 1996, pp. 494-501.

[40] J. J. Liou, L. C. Wang and K. T. Cheng, "On Theoretical and Practical Considerations of Path Selection for Delay Fault Testing," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, Nov. 2002, pp. 94-100.

[41] A. K. Majhi, V. D. Agrawal, J. Jacob and L. M. Patnaik, "Line Coverage of Path Delay Faults," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 5, Oct. 2000, pp. 610-613.

[42] A. Murakami, S. Kajihara, T. Sasao, I. Pomeranz and S. M. Reddy, "Selection of Potentially Testable Path Delay Faults for Test Generation," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 376-384.

[43] M. Sharma and J. H. Patel, "Finding a Small Set of Longest Testable Paths that Cover Every Gate," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 974-982.

[44] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 9, Sept. 1987, pp. 694-701.

[45] J. Benkoski, E. V. Meersch, L. J. M. Claesen and H. D. Man, "Timing Verification Using Statically Sensitizable Paths," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 10, Oct. 1990, pp. 1073-1084.

[46] P. McGeer and R. K. Brayton, "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network," in *Proc. ACM/IEEE Design Automation Conference*, Las Vegas, NV, Jun. 1989, pp. 561-567.

[47] H. Chang and J. A. Abraham, "VIPER: An Efficient Vigorously Sensitizable Path Extractor," in *Proc. ACM/IEEE Design Automation Conference*, Dallas, TX, June 1993, pp. 112-117.

[48] K. Fuchs, M. Pabst and T. Rossel, "RESIST: A Recursive Test Pattern Generation Algorithm for Path Delay Faults Considering Various Test Classes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 12, Dec. 1994, pp. 1550-1562.

[49] M. P. Kusko, B. J. Robbins, T. J. Koprowski and W. V. Huott, "99% AC Test Coverage Using Only LBIST on the 1GHz IBM S/390 zSeries 900 Microprocessor," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2001, pp. 586-592.

[50] J. Saxena, K. M. Butler, J. Gatt, R. Raghuraman, S. P. Kumar, S. Basu, D. J. Campbell and J. Berech, "Scan-Based Transition Testing – Implementation and Low Cost Test Challenges," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 1120-1129.

[51] J. Savir, "Skewed-Load Transition Test: Part I, Calculus," in *Proc. IEEE International Test Conference*, Baltimore, MD, Sept. 1992, pp. 705-713.

[52] S. Patel and J. Savir, "Skewed-Load Transition Test: Part II, Coverage," in *Proc. IEEE International Test Conference*, Baltimore, MD, Sept. 1992, pp. 714-722.

[53] J. Savir and S. Patel, "Broad-Side Delay Test," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, Aug. 1994, pp. 1057-1064.

[54] W. Y. Chen, S. K. Gupta and M. A. Breuer, "Test Generation for Crosstalk-Induced Delay in Integrated Circuits," in *Proc. International Test Conference*, Atlantic City, NJ, Sept. 1999, pp. 191-200.

[55] B. Choi and D. M. H. Walker, "Timing Analysis of Combinational Circuits Including Capacitive Coupling and Statistical Process Variation," in *Proc. VLSI Test Symposium*, Montreal, Canada, Apr. 2000, pp. 49-54.

[56] A. Krstic, J. J. Liou, Y. M. Jiang and K. T. Cheng, "Delay Testing Considering Crosstalk-Induced Effects," in *Proc. International Test Conference*, Baltimore, MD, Oct. 2001, pp. 558-567.

[57] W. Moore, G. Gronthoud, K. Baker and M. Lousberg, "Delay-Fault Testing and Defects in Sub-Micron ICs – Does Critical Resistance Really Mean Anything?" in *Proc. International Test Conference*, Atlantic City, NJ, Oct. 2000, pp. 95-104.

[58] K. T. Cheng and H. C. Chen, "Classification and Identification of Non-Robust Untestable Path Delay Faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 8, Aug. 1996, pp. 845-853.

[59] K. T. Cheng and H. C. Chen, "Delay Testing for Non-Robust Untestable Circuits," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 1993, pp. 954-961.

[60] W. K. C. Lam, A. Saldanha, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Delay Fault Coverage and Performance Trade-Offs," in *Proc. ACM/IEEE Design Automation Conference*, Dallas, TX, Jun. 1993, pp. 446-452.

[61] J. B. Brockman and S. W. Director, "Predictive Subset Testing: Optimizing IC Parametric Performance Testing for Quality, Cost, and Yield," *IEEE Transactions on Semiconductor Manufacturing*, vol. 2, no.3, Aug. 1989, pp. 104-113.

[62] X. Lu, Z. Li, W. Qiu, W. Shi and D. M. H. Walker, "Longest Path Selection for Delay Test Under Process Variation," in *Proc. Asian and South Pacific Design Automation Conference*, Yokohama, Japan, Jan. 2004, pp. 98-103.

[63] W. Qiu, X. Lu, J. Wang, Z. Li, D. M. H. Walker and W. Shi, "A Statistical Fault Coverage Metric for Realistic Path Delay Faults," in *Proc. IEEE VLSI Test Symposium*, Napa Valley, CA, Apr. 2004, pp. 37-42.

[64] V. R. Sar-Dessai and D. M. H. Walker, "Resistive Bridge Fault Modeling, Simulation and Test Generation," in *Proc. IEEE International Test Conference*, Atlantic City, NJ, Sept. 1999, pp. 596-605.

[65] Z. Li, X. Lu, W. Qiu, W. Shi and D. M. H. Walker, "A Circuit Level Fault Model for Resistive Opens and Bridges," in *Proc. IEEE VLSI Test Symposium*, Napa Valley, CA, Apr. 2003, pp. 379-384.

[66] Z. Li, X. Lu, W. Qiu, W. Shi and D. M. H. Walker, "A Circuit Level Fault Model for Resistive Bridges," *ACM Transactions on Design Automation of Electronic Systems*, vol. 8, no. 4, Oct. 2003, pp. 546-559.

[67] X. Lu, Z. Li, W. Qiu, D. M. H. Walker and W. Shi, "A Circuit Level Fault Model for Resistive Shorts of MOS Gate Oxide," in *Proc. IEEE International Workshop on Microprocessor Test and Verification*, Austin, TX, Sept. 2004, pp. 97-102.

[68] W. Qiu, X. Lu, Z. Li, D. M. H. Walker and W. Shi, "CodSim – A Combined Delay Fault Simulator," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Boston, MA, Nov. 2003, pp. 79-86.

[69] W. Qiu, Z. Li, X. Lu, W. Shi and D. M. H. Walker, "Combined Delay Fault Modeling and Simulation," in *Proc. Semiconductor Research Corporation Technical Conference*, Dallas, TX, Aug. 2003, No. 1.7.

[70] W. Qiu and D. M. H. Walker, "An Efficient Algorithm for Finding the K Longest Testable Paths Through Each Gate in a Combinational Circuit," in *Proc. IEEE International Test Conference*, Charlotte, NC, Sept. 2003, pp. 592-601.

[71] D. Belete, A. Razdan, W. Schwarz, R. Raina, C. Hawkins and J. Morehead, "Use of DFT Techniques in Speed Grading a 1GHz+ Microprocessor," in *Proc. IEEE International Test Conference*, Baltimore, MD, Oct. 2002, pp. 1111-1119.

[72] S. D. Millman and E. J. McCluskey, "Detecting Bridging Faults with Stuck-At Test Sets," in *Proc. IEEE International Test Conference*, Washington, DC, Sept. 1988, pp. 773-783.

[73] N. A. Touba and E. J. McCluskey, "Pseudo-Random Pattern Testing of Bridging Faults," in *Proc. IEEE International Conference on Computer Design*, Austin, TX, Oct. 1997, pp. 54-60.

[74] B. Chess and T. Larrabee, "Logic Testing of Bridging Faults in CMOS Integrated Circuits," *IEEE Transactions on Computers*, vol. 47, no. 3, Mar. 1998, pp. 338-345.

[75] J. P. Cusey and J. H. Patel, "BART: A Bridging Fault Test Generator for Sequential Circuits," in *Proc. IEEE International Test Conference*, Washington, DC, Nov. 1997, pp. 838-847.

[76] M. Spica, M. Tripp and R. Roeder, "A New Understanding of Bridge Defect Resistances and Process Interactions from Correlating Inductive Fault Analysis Predictions to Empirical Test Results," in *Proc. IEEE Defect Based Testing*, Marina Del Rey, CA, Apr. 2001, pp. 11-16.

[77] W. N. Li, S. M. Reddy and S. K. Sahni, "On Path Selection in Combinational Logic Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 1, Jan. 1989, pp. 56-63.

[78] Y. Shao, S. M. Reddy, I. Pomeranz and S. Kajihara, "On Selecting Testable Paths in Scan Designs," in *Proc. IEEE European Test Workshop*, Corfu, Greece, May 2002, pp. 53-58.

[79] I. Pomeranz, S. M. Reddy and P. Uppaluri, "NEST: A Nonenumerative Test Generation Method for Path Delay Faults in Combinational Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 12, Dec. 1995, pp. 1505-1515.

[80] K. Fuchs, F. Fink and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 10, Oct. 1991, pp. 1323-1355.

[81] J. A. Bell, "Timing Analysis of Logic-Level Digital Circuits Using Uncertainty Intervals," M. S. *Thesis*, Department of Computer Science, Texas A&M University, 1996.

[82] R. Stewart and J. Benkoski, "Static Timing Analysis Using Interval Constraints," in *Proc. IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, June 1991, pp. 308-311.

[83] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C-30, no. 3, Mar. 1981, pp.215-222.

[84] W. Qiu and D. M. H. Walker, "Testing the Path Delay Faults of ISCAS85 Circuit c6288," in *Proc. IEEE International Workshop on Microprocessor Test and Verification*, Austin, TX, May 2003, pp. 19-24.

[85] M. Hansen, H. Yalcin and J. P. Hayes, "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, July-Sept. 1999, pp. 72-80.

[86] T. Kirkland and M. R. Mercer, "A Topological Search Algorithm for ATPG," in *Proc. ACM/IEEE Design Automation Conference*, Jun. 1987, Miami, FL, pp. 502-508.

[87] W. T. Cheng and T. J. Chakraborty, "Gentest: An Automatic Test Generation System for Sequential Circuits," *IEEE Computer*, vol. 22, no.4, Apr. 1989, pp. 43-49.

[88] T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," in *Proc. ACM/IEEE European Design Automation Conference*, Amsterdam, Netherlands, Feb. 1991, pp. 214-218.

[89] A. A. Thadhlani, "Realistic Fault Modeling and Quality Test Generation of Combined Delay Faults," M. S. *Thesis*, Department of Computer Science, Texas A&M University, 2001.

[90] W. Qiu, J. Wang, D. M. H. Walker, D. Reddy, X. Lu, Z. Li, W. Shi and H. Balachandran, "K Longest Paths Per Gate (KLPG) Test Generation for Scan-Based Sequential Circuits," in *Proc. IEEE International Test Conference*, Charlotte, NC, Oct. 2004, pp. 223-231.

[91] W. Qiu, J. Wang, X. Lu, Z. Li, D. M. H. Walker and W. Shi, "At-Speed Test for Path Delay Faults Using Practical Techniques," in *Proc. IEEE International Workshop on Current and Defect Based Testing*, Napa Valley, CA, Apr. 2004, pp. 61-66.

[92] J. Wang, Z. Yue, X. Lu, W. Qiu, W. Shi and D. M. H. Walker, "A Vector-Based Approach for Power Supply Noise Analysis in Test Compaction," in *Proc. IEEE International Test Conference*, Austin, TX, Nov. 2005, pp. 517-526.

[93] J. Wang, X. Lu, W. Qiu, Z. Yue, S. Fancler, W. Shi and D. M. H. Walker, "Static Compaction of Delay Tests Considering Power Supply Noise," in *Proc. IEEE VLSI Test Symposium*, Palm Springs, CA, May 2005, pp. 235-240.

[94] E. M. Rudnick and J. H. Patel, "Efficient Techniques for Dynamic Test Sequence Compaction," *IEEE Transactions on Computers*, vol. 48, no. 3, Mar. 1999, pp. 323-330.

[95] I. Pomeranz and S. M. Reddy, "COREL: A Dynamic Compaction Procedure for Synchronous Sequential Circuits with Repetition and Local Static Compaction," in *Proc. IEEE International Conference on Computer Design*, Austin, TX, Sept. 2001, pp. 142-147.

[96] S. Y. Lee, B. Cobb, J. Dworak, M. R. Grimaila and M. R. Mercer, "A New ATPG Algorithm to Limit Test Set Size and Achieve Multiple Detections of All Faults," in *Proc. Design Automation and Test in Europe Conference and Exhibition*, Paris, France, Mar. 2002, pp. 94-99.

[97] J. Wingfield, J. Dworak and M. R. Mercer, "Function-Based Dynamic Compaction and Its Impact on Test Set Sizes," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Boston, MA, Nov. 2003, pp. 167-174.

[98] K. L. Shepard and V. Narayanan, "Noise in Deep Submicron Digital Design," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, Nov. 1996, pp. 524-531.

[99] Y. M. Jiang and K. T. Cheng, "Analysis of Performance Impact Caused by Power Supply Noise in Deep Submicron Devices," in *Proc. ACM/IEEE Design Automation Conference*, New Orleans, LA, Jun. 1999, pp. 760-765.

[100] R. Ahmadi and F. N. Najm, "Timing Analysis in Presence of Power Supply and Ground Voltage Variations," in *Proc. IEEE/ACM International Conference on Computer Aided Design*, San Jose, CA, Nov. 2003, pp. 176-183.

[101] C. Tirumurti, S. Kundu, S. Sur-Kolay and Y. S. Chang, "A Modeling Approach for Addressing Power Supply Switching Noise Related Failures of Integrated Circuits," in *Proc. Design Automation and Test in Europe Conference and Exhibition*, Paris, France, Feb. 2004, pp. 1078-1083.

[102] T. M. Mak, A. Krstic, K. T. Cheng and L. C. Wang, "New Challenges in Delay Testing of Nanometer, Multigigahertz Designs," *IEEE Design & Test of Computers*, vol. 21, no. 3, May-Jun. 2004, pp. 241-248.

[103] S. Rusu, S. Seidel, G. Woods, D. Grannes, H. Muljono, J. Rowlette and K. Petrosky, "Backside Infrared Probing for Static Voltage Drop and Dynamic Timing Measurements," in *Proc. IEEE International Solid-State Circuits Conference*, San Francisco, Feb. 2001, pp. 276-277.

VITA

Wangqi Qiu

1269-1-501 N. Zhongshan Rd.

Shanghai 200065

People's Republic of China

E-mail: wqqiu2000@hotmail.com

Wangqi Qiu was born in Shanghai, China on February 12, 1978. He obtained his B.S. in Computer Science from Fudan University, Shanghai, China in July 2000, and Ph.D. in Computer Science from Texas A&M University, College Station, TX in December 2006. His research interests are delay fault testing, automatic test pattern generation, timing analysis and optimization.