Design and Test for Timing Uncertainty in VLSI Circuits

YUAN, Feng

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Doctor of Philosophy
in
Computer Science and Engineering

The Chinese University of Hong Kong

June 2012

Abstract

With technology scaling, integrated circuits (ICs) suffer from increasing process, voltage, and temperature (PVT) variations and aging effects. In most cases, these reliability threats manifest themselves as timing errors on speed-paths (i.e., critical or near-critical paths) of the circuit. Embedding a large design guard band to prevent timing errors to occur is not an attractive solution, since this conservative design methodology diminishes the benefit of technology scaling. This creates several challenges on build a reliable systems, and the key problems include (i) how to optimize circuit's timing performance with limited power budget for explosively increased potential speed-paths; (ii) how to generate high quality delay test pattern to capture ICs' accurate worst-case delay; (iii) to have better power and performance tradeoff, we have to accept some infrequent timing errors in circuit's the usage phase. Therefore, the question is how to achieve online timing error resilience.

To address the above issues, we first develop a novel technique to identify so-called false paths, which facilitate us to find much more false paths than conventional methods. By integrating our identified false paths into static timing analysis tool, we are able to achieve more accurate timing information and also save the cost used to optimize false paths. Then, due to the fact that existing delay automated test pattern generation (ATPG) methods may generate test patterns that are functionally-unreachable, and such patterns may incur excessive (or lim-

ited) power supply noise (PSN) on sensitized paths in test mode, thus leading to over-testing or under-testing of the circuits, we propose a novel pseudo-functional ATPG tool. By taking both circuit layout information and functional constrains into account, we use ATPG like algorithm to justify transitions that pose the maximized functional PSN effects on sensitized critical paths. Finally, we propose a novel in-situ correction technique to mask timing errors, namely InTimeFix, by introducing redundant approximation circuit with more timing slack for speed-paths into the design. The synthesis of the approximation circuit relies on simple structural analysis of the original circuit, which is easily scalable to large IC designs.

摘要

由於特徵尺寸不斷縮小，集成電路在生產過程中的工藝偏差在運行環境中溫度和電壓等參數的波動以及在使用過程中的老化等效應越來越嚴重，導致芯片的時序行為出現很大的不確定性。多數情況下，芯片的關鍵路徑會不時出現時序錯誤。加入更多的時序餘量不是一種很好的解決方案，因為這種保守的設計方法會抵消工藝進步帶來的性能上的好處。這就為設計一個時序可靠的系統提出了極大的挑戰，其中的一些關鍵問題包括：（一）如何有效地分配有限的功率預算去優化那些正爆炸式增加的關鍵路徑的時序性能；（二）如何產生能夠捕捉準確的最壞情況時延的高品質測試向量；（三）為了能夠取得更好的功耗和性能上的平衡，我們將不得不允許芯片在使用過程中出現一些頻率很低的時序錯誤。隨之而來的問題是如何做到在線的檢錯和糾錯。

為了解決上述問題，我們首先發明了一種新的技術用於識別所謂的虛假路徑，該方法使我們能夠發現比傳統方法更多的虛假路徑。當將所提取的虛假路徑集成到靜態時序分析工具里以後，我們可以得到更為準確的時序分析結果，同時也能節省本來用於優化這些路徑的成本。接著，考慮到現有的延時自動向量生成（ATPG）方法會產生功能模式下無法出現的測試向量，這種向量可能會造成測試過程中在被激活的路徑周圍出現過多（或過少）的電源噪聲（PSN），從而導致測試過度或者測試不足情況。為此，我們提出了一種新的偽功能ATPG工具。通過同時考慮功能約束以及電路的物理佈局信息，我們使用類似ATPG的算法產生狀態跳變使其能最大化已激活的路徑周圍的PSN影響。最後，基於近似電路的原理，我們提出了一種新的在線原位校正技術，即InTimeFix，用於糾正時序錯誤。由於實現近似電路的綜合僅需要簡單的電路結構分析，因此該技術能夠很容易的擴展到大型電路設計上去。

Acknowledgement

At the very beginning, I am deeply indebted to my supervisor, Professor Qiang Xu, who patiently motivated me to conceive and develop the main ideas in the thesis. I would like to express to him my sincere gratitude for his seasoned guidance from the very early stage of this research work as well as providing constructive advices throughout the entire study. In particular, I also would like to thank him and his wife for their concerns about my daily life.

Professor Yu-Liang (David) Wu and Professor Rung Tsong (Michael) Lyu, thank you for your constructive comments in the term presentations which gave me important suggestion to my research progress. I shall express my great thanks to Professor Kwang-Ting (Tim) Cheng from University of California, Santa Barbara for his insightful hints on my research works during several meetings, for serving as my external marker and reading my thesis.

My research partners Yubin Zhang, Lin Huang, Xiao Liu, Li Jiang, Rong Ye, Yuxi Liu, Jie Zhang, Chenfei Ma and Zelong Sun in the The CUhk REliable Computing Laboratory (CURE), thank you for your insightful comments on my research work. I am also grateful to all the colleagues who once stayed in 506 EDA office, Linfu Xiao, Minqi Jiang, Zaichen Qian, Xiaoqing Yang, Zigang Xiao, Yan Jiang, Xing Wei, Yi Diao and Xu He, it is you who bring me laugher and make my post-graduate study life colorful. Special thanks are also to my friends Qiang Ma and Lei Shi, who helped me to settle down when I just came to this campus.

Last but not the least, my father Guangfu Yuan, my mother Guangpin Gui, my fiancee Rong Huang and all my family members, without your love and support, I cannot achieve anything. I would like to give my greatest appreciation to you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the passed decades, semiconductor industry kept evolving the manufacture technology at the rate indicated by the famous Moore's law, which leaded to smaller transistors, higher packing densities, decreased supply voltages and increased clock frequencies, thereby contributing to the goals of higher performance and lower power consumption. However, with ultra deep sub-micron technologies, there is an increasing uncertainty for the timing behavior of today's integrated circuits (ICs), often manifesting themselves as infrequent timing errors on speed-paths [104, 103]. There are multiple factors that contribute to this effect: (i) inevitable static process variation caused by manufacturing imperfection leads to the mismatch of timing performance between the designed value and the actual one [116, 88]; (ii) dynamic environment fluctuation in supply voltage, temperature, and multiple-input switchings cause varying circuit delay at runtime [11, 105]; (iii) circuit aging mechanisms such as hot carrier injection (HCI) and negative-bias temperature instability (NBTI) lead to gradual increase of circuit delay over its lifetime.

1.1 Challenges to Solve Timing Uncertainty Problem

In IC design, timing error resilience is fulfilled by cooperating series of works through timing optimization, at-speed delay testing and online timing error protection. With ever-increasing timing uncertainty effect, we have to make a comprehensive improvement on every respects to achieve the goal.

To achieve ICs' timing closure, timing optimization techniques [14, 16, 5, 6] are developed at several phases of the backend design, from synthesis to physical design. Successfully applying the above techniques heavily relies on timing analysis [7] of logic circuits, by which the speed-paths are first identified and then are fed as the optimization targets. Unfortunately, traditional timing analysis always fails to predict the real speed-paths, thus resulting in missing the optimization focus or even leads to the irrevocable timing failure after product shipping. For one hand, the essential reason undoubtedly is the variation-induced timing uncertainty. To solve that, statistical timing analysis method [8] is invented, while one of side effects is that it exponentially increases number of critical paths. On the other hand, it is noted that the *false path* (also known as *unsensitizable path*), down which signal cannot propagate by any input pattern, is also a significant factor contributing to the timing analysis inaccurate. Since the false paths do not induce any timing error, by identifying those false paths [9] and integrating them in timing analysis tool [10], it not only helps us to generate better timing result and improves the circuit performance, but also alleviates the burden of timing optimization algorithms and saves the unnecessary cost used to reduce the false paths delay.

The false paths identification (FPI) is an important and relevant problem for IC designers. The path sensitization criteria used in most prior false path identification techniques are based on automatic test pattern generation (ATPG) like

techniques [101]. To be specific, for a given path, it is assumed to be sensitizable if there exists a test vector pair (v_1, v_2) that activates a transition at the launch point of a path and propagates to its ending point; if, however, we cannot find such a test vector pair in any circumstances, this path is deemed as a false path. Generally speaking, ATPG-based FPI techniques need to exhaustively search in an input space to prove that a targeted path to be false. To alleviate this problem, there have also been some implication-based false path identification techniques (e.g., [91, 94]). Essentially, these methods use the same criteria to identify false paths. The difference is that they try to prove the non-existence of test vectors using implication analysis instead of exhaustive search. The above techniques may miss to identify some false paths due to the fact: for a particular path, even if we are able to find a test vector pair that activates it, such a test may be *functionally-unreachable*. Consider a finite state machine encoded with one-hot code, the legal combinations of values in the circuit's state elements are only those with a single logic '1' and all the others logic '0'. Consequently, if a path can be activated in this circuit only with multiple logic '1's in these state elements (i.e., containing illegal states), this path is considered to be a true path based on the above criteria, but in fact it is functionally unsensitizable.

For the taped-out circuits, manufacturing test is applied to verify their timing correction and guarantee quality of shipped products. As mainstream design-for-test (DfT) technique, scan-based DfT makes automatic test pattern generation (ATPG) viable for large ICs, which, however, changes the circuit states in test mode, making them possibly different from that in functional mode, as mentioned earlier. Under ever-increasing runtime environment fluctuation, the discrepancy between functional mode and test mode has severe reverse effect on at-speed delay testing for ICs fabricated with latest technology, and become a serious concern for the industry [40, 45, 41]. Recent design evaluations have revealed that at-speed

scan patterns were up to 20% slower than any functional pattern [107]. Consequently, some good ICs that would work in application might fail at-speed delay tests, leading to unnecessary *test yield loss* (also known as *over-testing*) [43]. To reduce over-testing, several power-aware test generation methodologies were proposed to reduce test overkills, by reducing switching activities in scan capture mode to ensure the power integrity in manufacturing test [39, 47, 50]. This, however, leads to the concern for under-testing, i.e., if we over-restrict test power, some defective chips containing speed-related defects may pass manufacturing test, leading to *test escapes* [44] (also known as under-testing). Therefore, the real question is: *How do we make sure that circuits' activities in test mode correlate well with that in functional mode so that we can exercise the worst-case timing of the circuits under test (CUTs) in their functional mode during manufacturing test?*

With the continuous downscaling of transistor feature size, timing uncertainty poses significant challenge to IC. On one hand, embedding a large design guard band to prevent timing errors to occur is not an attractive solution, since this conservative design methodology diminishes the benefit of technology scaling [106]. On the other hand, it is increasingly difficult to rely on off-line delay testing to guarantee circuit timing correctness in functional mode [107]. Consequently, there is a growing research interest to achieve online timing error resilience.

Most of existing solutions for timing error resilience (e.g., the well-known *Razor* technique [31]) try to restore the state of the system to a known-good pre-error state. These techniques are very effective for timing error correction (TEC) in processors with microarchitectural support such as instruction replay, but they are very difficult, if not impossible, to be applied to general logic circuits, due to the high cost to checkpoint error-free states in such designs. *In-situ* timing error correction techniques that are able to mask errors without any rollback, are therefore very attractive. Among the few *in-situ* TEC techniques presented in the literature, most

of them [108, 109, 110] rely on time-borrowing technique to correct timing errors, by delaying the arrival time of the correct data to the next logic level. As these techniques reduce the timing slack for the logic level that follows speed-paths, they have difficulty to handle the case when speed-paths exist in consecutive logic levels, limiting the applicability of such solutions. In [111], the authors proposed to synthesize a redundant logic block that is activated only when the speed-paths of the circuit are sensitized, and use it to mask timing errors on targeted paths. While interesting, their proposed synthesis algorithm is time-consuming and the redundant logic block incurs large area overhead.

1.2 Contributions and Thesis Outline

In this thesis, we present advanced techniques targeting to solve several challenging issues caused by ever-increasing timing uncertainty of IC.

Chapter 3 is concerned with finding timing-independent false paths that cannot be sensitized under any signal arrival time condition in integrated circuits. Existing techniques regard a path as a true path as long as a vector pair can be found to sensitize it. This is rather pessimistic since such a path might be activated only with illegal states in the circuit and hence it is actually functionally-unsensitizable. In this part, we adapt the illegal state identification technique presented in [100] and integrate it into our FPI flow. For a given critical path, we present effective and efficient techniques to check whether it is a true path or a false path. We also present novel solutions to address the more general problem of finding as many false paths in a circuit as possible.

Motivated by the fact that current at-speed delay patterns cannot effectively activate the circuits' worst case functional delay under more and more severe run-time environment fluctuation, thus lead to either test yield loss or test escape. In Chapter 4, we present novel pseudo-functional ATPG techniques to simultane-

ously reduce both over-testing and under-testing in at-speed delay testing. Firstly, by taking the circuit layout information into account, functional constraints related to critical paths are extracted. Then, we generate functionally-reachable test cubes for delay faults in the circuit. Finally, we use ATPG-like algorithm to generate switching activities that pose the worst case power supply noises on sensitized critical paths under the consideration of functional constraints.

Despite that our proposed at-speed delay test methodology in Chatper 4 has significantly enhance test quality, it is inevitable that some timing error will escape to the usage phase and cause fatal timing error. Moreover, considering aging effect, the delay on circuits' critical paths gradually degrade and finally exhibit timing error. In Chapter 5, we propose a novel *in-situ* correction technique to mask timing errors, namely *InTimeFix*, by introducing redundant *approximation circuit* into the original design, we are able to create a logically-equivalent yet timing-improved circuit, and prove its correctness. The proposed low-cost and scalable synthesis technique timing error masking logic based on simple structural analysis.

In Chapter 6, we summarizes this thesis and points out the directions for future work.

□ **End of chapter.**

Chapter 2

Background

For better understanding the content this thesis, we present some background knowledge relevant research areas. In order to effectively address the timing uncertainty issues, in section 2.1, we detail the factors that contribute to timing uncertainty and to understand how do these factors work. Next, we first roughly introduce the technical flow to solve timing uncertainty problem with section 5.4. Finally, the main technical background targeting timing uncertainty will be detailed in section 2.3.2, 2.4.3 and 2.5.

2.1 Sources of Timing Uncertainty

2.1.1 Process Variation

Due to the imperfect manufacturing process, it is not possible to precisely control the design parameters for all the transistors on the silicon.

There are several sources of process variation, and the first one is the random dopant fluctuation. In CMOS technology, dopant atoms are doped into transistors' channel to control the threshold voltage. As transistor size shrinking, number of injected dopant atoms decreases significantly, which can be observed from Fig. 2.1.

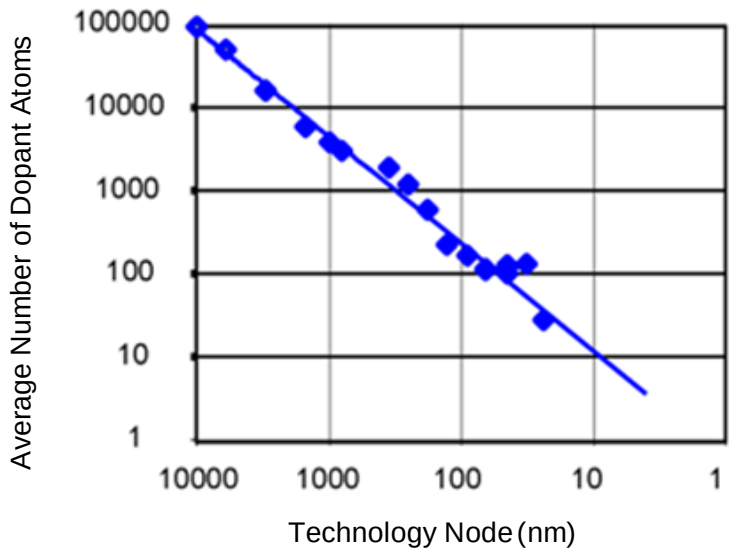


Figure 2.1: Average number of dopant atoms in the channel as a function of technology node from [28]

According to industrial data [103], there are tens of dopant atoms left for 32-nm generation. Therefore, the dopant distribution randomness increase dramatically, leading to significant variability. The second source is sub-wavelength lithography, which is originated since 0.25-μm technology node. It is noted from Fig. 2.2, 193 nm wavelength of light is used to pattern the transistors since 130 nm. The gap in light wavelength and transistor size will continue to widen until the appearance of extreme ultra-violet technology. It is the primary reason for line width roughness, and the source of transistor delay variation.

It should be noticed that process variation is fixed after fabrication and remains effective over the entire circuit lifetime. To address the randomness issue, circuit designer has to optimize numbers of paths which are potentially to be critical, and leads to severe power waste.

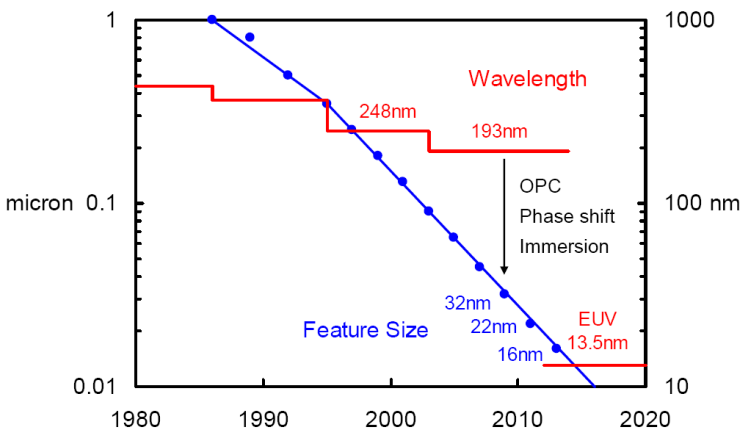


Figure 2.2: Lithograghy trend [29]

2.1.2 Runtime Environment Fluctuation

Circuit timing behavior is input vector dependent. For one hand, the critical path exhibits delay only if it is sensitized by input data sequence. On the other hand, even if a path is sensitized, its delay could vary because of different environment factors which are related to input vector.

Dynamic power consumption due to the state transition causes local temperature hot-spot, which in turn creates temperature variations across the die, affecting circuit performance. Power supply variation is another factor causing timing uncertainty. Inductive overshooting generated by sudden current increasing and resistive voltage drop due to high computational activity make supply voltage on power distribution network very sensitive to input vector. Furthermore, the propagation delay of on path transistor also suffers from some other coupling noise effects, for instance, interconnect cross talking.

Under runtime environment fluctuation, the traditional ATPG tool is increasingly difficult to generate high quality pattern to verify circuit timing correctness.

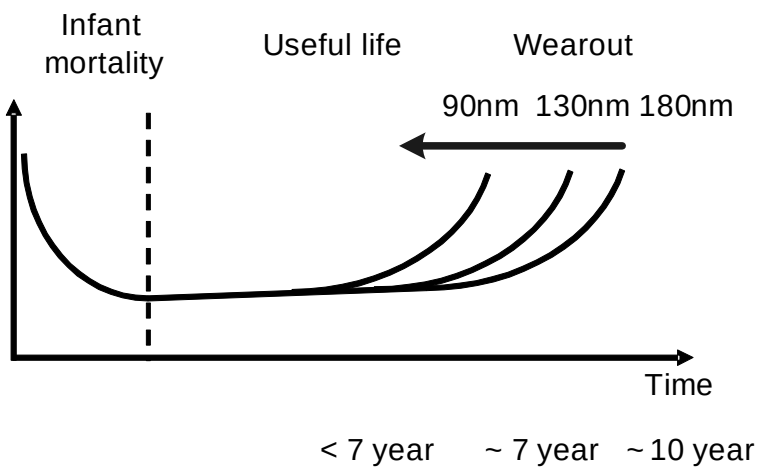


Figure 2.3: Aging trend

2.1.3 Aging Effect

In the usage phase, circuit performance degrades gradually by all kind of aging effects. Study in [103] releases that transistor's saturation current decreases over year due to the effects such as oxide wear out and hot carrier injection. To cope with this thread, traditional design relied on simple guardband to bypass the analysis and optimize these time-dependent effects. As demonstrated by famous bath tube curve Fig. 2.3, these reliability degradation is exacerbated as the aggressive scaling, and conservative design will lead to excessive over-margining.

2.2 Technical Flow to Solve Timing Uncertainty Problem

During circuit's life cycle, several techniques work cooperatively to guarantee its timing error resilience, which is demonstrated as Fig. 2.4.

For digital circuit, design engineers start to consider timing issue from the

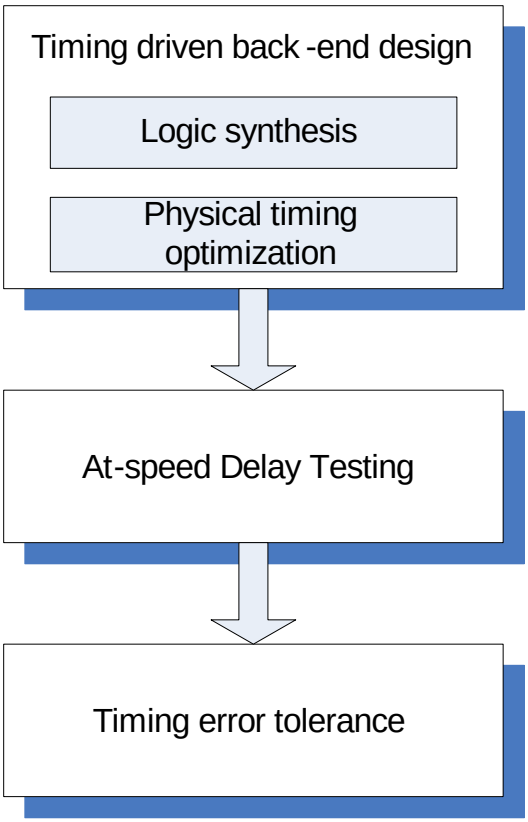


Figure 2.4: Flow to achieve timing closure

back-end design phase. With the RTL description, logic synthesis [13, 14, 15] is used to transfer circuit into multi-level logic network under timing constrain. In this step, timing analysis is conducted under some rough timing model, for example, constant gate delay. Typical physical design is composed of many important steps such as technology mapping [16, 17], floorplanning [18], placement [19, 20] and routing [21, 22, 23], and each of them puts significant effort on improving circuit timing performance. In the above procedures, timing analysis is extensively applied. Although more and more physical information is available, timing analysis result is still inaccurate due to timing uncertainty. To alleviate this problem, false path aware timing analysis is introduced, whose background is presented in

Section 2.3.2.

Process variation and delay defects introduced in manufacturing process may lead to timing constrain violation, at-speed delay testing is conducted to verify timing correctness of circuits. There are of scan based delay testing techniques [23, 35, 25] developed previously. Discrepancy between the circuit in functional mode and that in test mode may lead to over-test or under-test problem, pseudo-functional test method, therefore, is proposed to copy this problem recently, which will be detailed in section 2.4.3.

In usage phase, we rely on timing error tolerance mechanism to overcome timing violation due to the undetected timing error in testing phase or aging effect. The background overview will be presented in section 2.5.

2.3 False Path

2.3.1 Path Sensitization Criteria

A netlist is composed of simple gates (i.e. AND, NAND, OR or NOR) and connection between them. A logic path P_x is defined as $P = (G_0, f_0, G_1, f_1 ... f_{m-1}, G_m)$, which is an alternating sequence of gates and connections. Gate G_0 is a primary input and G_m is a primary output. Connection $f_i, 0 \leqslant i \leqslant m-1$, is on-input of P_x which connect G_i to G_{i+1}. A connect is called a side-input of P_x associated with f_i if it is connect to G_{i+1} but not originated from G_i. Delay assignment depicts the assignment of a delay to each connection in a circuit.

Under vector pair $< v_1, V_2 >$, the logic value stabilized at connection f and output of gate G is called stable values at f and G, respectively. For given delay assignment, the times when the end of f and output of G become stable under v_2 is the stable times at f and G under v_2, respectively.

For given $< v_1, v_2 >$, the stable value and stable time are known for v_2. Con-

nection f, which is connected to G, is considered to dominate G if the stable value and stable time of G is determined by f. A path is said to sensitized by v_1, v_2 if each on-input of the path dominates its connected gate. Under a delay assignment M, a path is defined as a sensitizable path if there is at lease one vector pair which can sensitize the targeted path.

According to [90] if there exists an input vector v such that all side-inputs along P_x are set to non-controlling values, then P_x is static sensitizable. if there exists an input vector v such that that all side-inputs along P_x are set to non-controlling values when the on-inputs of f_i has a non-controlling value (no requirement for the side-inputs of f_i when the on-input f_i is controlling value), the P_x is functional sensitizable.

It is notice that both static and functional sensitization are independent of the delay assignment. Furthermore, functional sensitization is more relaxed than static sensitization, and it is the necessary criteria to sensitize a path.

2.3.2 False Path Aware Timing Analysis

In static timing analysis, based on the connection information and the delay model of components, a weighted acyclic direct timing graph is constructed. We can then use a linear structural algorithm to find the longest path (or a number of long paths) in the timing graph, giving a fast feedback to the designers about the performance of the circuit. Due to the ignoring of operational conditions and functionalities of components in the design, however, the critical paths being reported by static timing verifiers may be not sensitizable, leading to pessimistic timing analysis results.

Effective removal of false paths from static timing analysis is a critical task. This is because, STA is used in the inner loop of many circuit optimization tools to resolve timing issues and the effectiveness of such optimization processes is

deteriorated with the presence of false paths, leading to sub-optimal solution or even failure to achieve timing closure.

In a sequential circuit, paths start from primary inputs of the circuit or primary outputs of sequential elements, namely *launch points*, and end with primary outputs of the circuit or primary inputs of sequential elements, namely *ending points*. A path is a *true path* if a *functional vector pair* (v_1, v_2) can satisfy the functional sensitization criteria. Otherwise, it is a *false path*.

2.4 Manufacturing Testing

Manufacturing test is typically conducted with the help of automatic test equipment (ATE). When testing a circuit, both test patterns and the expected test responses are stored in the ATE. During the manufacturing test process, test patterns are transported from ATE to the circuit, and then the actual responses captured by the circuit are sent back to ATE to compare against the expected responses. Those circuits that have different responses from the expected ones are marked as defective products.

2.4.1 Functional Testing vs. Structural Testing

Functional testing was historically used to test IC products, wherein a large amount of test patterns are required to completely excise the circuit's functionalities. Generally speaking, the number of input patterns for functional testing will be 2^n for a circuit with n inputs. Taking a 64-bit ripple-carry adder as example, 2^{129} patterns are needed to apply complete functional test, which would take 2.158×10^{22} years to finish such test on a 1 GHz ATE [1]. Due to such exhaustive nature of functional testing, it is impractical for any reasonable-sized circuits. In addition, due to the need of applying functional tests at speed, the functional tester is much more ex-

pensive. The semiconductor industry hence mainly resorts to structural testing for this duty, wherein test patterns are selected based on circuit structural information and a set of fault models. One of the greatest advantages of structural test is that it allows us to develop structural search algorithms to achieve efficient testing. For the same 64-bit ripple-carry adder, 1728 patterns are enough for structural testing based on stuck-at fault model (introduced later).

Defects in an electronic system is defined as the unintended differences between the implemented hardware and its intended design [1]. It is very hard to generate tests for every possible type of physical defects. Fault models, therefore, are proposed to abstract faulty behaviors induced by defects. To generate test patterns effectively, faults are always modeled at a certain level of design abstraction, such as behavioral level, logic/gate level or transistor level. Fault models at behavioral level usually have no clear correlation to manufacturing defects and hence are used more often in design verification rather than manufacturing test. Transistor level fault models are also known as technology-dependent faults and are mainly used in analog circuit testing. Fault models at logic level (i.e., circuit is modeled as an interconnection of boolean gates, called *netlist*) are technology-independent and over time have been proven to be quite efficient and effective for testing digital circuits [1].

2.4.2 Scan-Based DfT

With the ever increasing transistor-to-pin ratio in IC products, sequential ATPG is no longer applicable on today's complex sequential circuits. The main purpose for scan-based DfT is to increase the controllability (i.e., the ability to set a particular circuit node to logic '0' or logic '1') and the observability (i.e., the ability to observe the state of a logic signal within the circuit) of the circuit's internal node so that it is easier to generate test patterns for the circuit. In scan-based circuits,

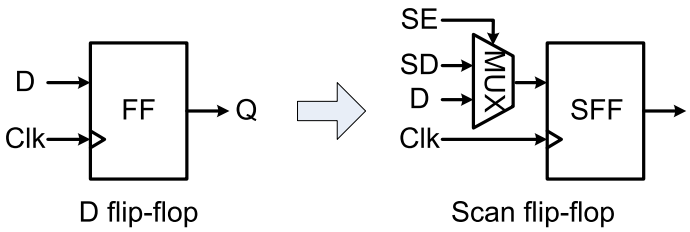


Figure 2.5: Transform the D Flip-Flop to Scan Flip-Flop

we substitute normal flip-flops (FFs) with scan FFs (SFFs), making them directly accessible in test mode. By doing so, from the test generation point of view, the circuit under test is a combinational circuit and hence the more tractable combinational ATPG can be used to generate test patterns.

SFFs can be implemented in various manners, e.g., mux-based SFFs, double latched SFFs, level sensitive scan latches SFFs [2, 1]. Fig. 2.5 depicts the transformation from a normal FF into a mux-based SFF. In the mux-based SFF, a multiplexor is inserted before the input of the FF with two inputs *D* and *SD*, which represent the original data input and the scan data input, respectively. Scan enable (*SE*) signal is used to select which channel as input of FF. By replacing normal FFs with SFFs, these state elements can be connected serially to form one or more long shift registers (called scan chain) through *SD* input, and the first and the last SFF of each scan chain are connected with an input pin and an output pins of the circuit. All the SFFs can be set as arbitrary states by shifting logic values into the scan chains. Similarly, the states of these SFFs can be observed by shifting out the contents of the shift registers.

The test procedure in scan-based testing can be divided into three phases.

- Scan in: *SE* signal is asserted to configure the circuit as scan mode. Test pattern is then shifted into scan chains for N_{sc} clock cycles, where N_{sc} is the length of longest scan chain;

- capture: *SE* signal is de-asserted, and the circuit applies the test pattern in functional mode and capture its responses into the same SFFs;
- Scan out: test responses are shifted out in the similar manner as the scan in process.

2.4.3 Pseudo-Functional Testing

When testing delay faults in scan-based designs, it is possible to activate *functionally infeasible paths* during test application [42]. If a chip fails a particular at-speed test that exercises such paths, this chip may be able to work in application but is considered as a bad chip. We solve this problem by the false path identification technique in Section 3. However, even if we are able to generate test patterns for those functionally-testable faults only, it is still possible that they incidentally detect some structural testable but functional untestable (ST-FU) faults and hence lead to test overkills [95], since non-functional patterns may generate excessive delay on the targeted path considering runtime environment fluctuation.

To cope with this problem, several power-aware test generation methodologies were proposed to reduce switching activities in scan capture mode to ensure timing safety in delay testing to avoid test overkills [39, 46, 47, 50]. However, if we over-restrict test power, under-testing might occur as some speed-related defects may not exhibit themselves, leading to *test escapes* [44]. Therefore, the real question is: *How can we exercise the worst-case timing of the circuits in their functional mode during manufacturing test?*

Pseudo-functional testing was proposed to tackle the above problem and has attracted lots of attention recently [51, 52, 53, 97, 99, 102]. In this technique, instead of identifying ST-FU delay faults in the CUT, functionally-unreachable states in the circuit are extracted and fed to a constrained ATPG tool, which backtracks immediately when illegal states are reached during test generation to obtain

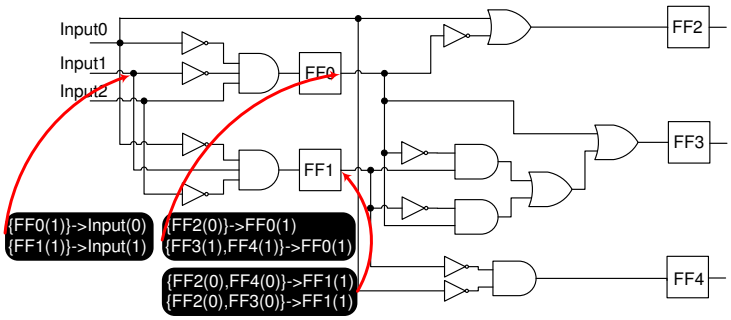


Figure 2.6: Illegal State Identification - An Example

pseudo-functional patterns [95].

Illegal state identification is one of the fundamental problems in pseudo-functional testing. Several approaches were proposed for illegal state identification in the literature, including SAT-based methods [51], implication-based strategies [97, 102], mining-based techniques [99], and a recent justification-based method [100]. As [100] is able to effectively and efficiently identify much more illegal states when compared to other techniques, it is utilized in this work and we briefly introduce how it works in the following paragraphs, using an example circuit as shown in Fig. 2.6.

In [100], the authors studied the structural root cause for illegal states and showed that they are mainly caused by multi-fanout nets in the circuit. That is, illegal states would imply logic violations at different branches of the same multi-fanout net, explicitly in the same time frame or implicitly across multiple time frames. Based on this observation, this work defined the so-called *justification scheme* at every circuit node in the format of $Cube0 \rightarrow 0$ and $Cube1 \rightarrow 1$, denoting that a state cube $Cube0/Cube1$ justifies logic '0/1' on this node. For example, a justification scheme $FF0(1) \rightarrow Input1(0)$ in Fig. 2.6 means that $FF0 = 1$ can justify logic '0' at circuit node $Input1$. All such justification schemes are systematically built for each net. Then, starting from a multi-fanout net, [100] derived illegal

states using contradictory justification schemes. In this example, for the multi-fanout at $Input1$, we have $\{FF0(1)\} \rightarrow Input1(0)$ and $\{FF1(1)\} \rightarrow Input1(1)$. We can therefore conclude that the state cube $\{FF0(1), FF1(1)\}$ is illegal as they imply logic conflicts on this net. The above procedure are conducted for every multi-fanout nets to find those illegal states that explicitly cause contradicting values on them.

Next, [100] expands the above-obtained illegal state cubes to the next sequential level, again, using the justification scheme information. For the example circuit in Fig. 2.6, since $\{FF0(1), FF1(1)\}$ is known to be illegal, obviously, those state cubes that can justify this state cube are also illegal. Consequently, by combining those compatible state cubes that lead to $FF0(1)$ and $FF1(1)$, we can obtain two new illegal state cubes, i.e., $\{FF2(0), FF4(0)\}$ and $\{FF2(0), FF3(0)\}$. In [100], the above expansion process is conducted for every illegal state cube until it cannot be expanded any more. Compared to prior work that explicitly unroll the circuit into a few time frames for illegal state identification, the above *illegal state expansion* procedure is able to implicitly walk through unlimited number of time frames of the circuit efficiently.

2.5 Timing Error Tolerance

In order to achieve timing error resilience, we can either predict the error occurrence or take proactive actions to avoid them or we need to be able to detect timing errors after they occur and recover from them.

2.5.1 Timing Error Detection

There are many timing error detector designs presented in the literature (e.g., [12, 26, 27, 30, 31, 32, 33]), and most of them are based on monitoring signal

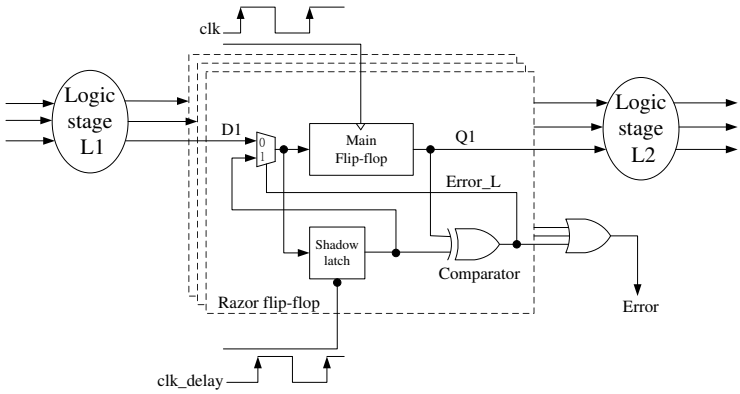


Figure 2.7: Razor Flip-Flop

transitions on speed-paths for a specified period after the clock edge. Let us use the well-known Razor flip-flop [31] as a representative technique to demonstrate how such error detectors work.

As shown in Fig. 2.7, Razor flip-flop includes a main flip-flop, an additional latch called shadow latch and other necessary components. The main flip-flop latches the output signal at the clock edge while the shadow latch, controlled by an inverted clock, latches the signal half of the clock period later, which is always correct. When timing error occurs, the main FF will latch an incorrect value, which is different from that in the shadow latch. Therefore, results from the comparator indicate whether there is timing error. Once there is timing error, the correct data in the shadow latch will be written back to the main flip-flop and the errant instruction will be flushed. Hence, each suspicious flip-flop, where timing errors most happen, needs to be replaced with a Razor flip-flop.

2.5.2 Timing Error Recover

One widely-used error recovery scheme is to restore the state of the system to a known-good pre-error state. Razor first implemented such a recovery scheme

for timing errors with microarchitectural support. That is, when a timing error is detected in a Razor flip-flop, the processor pipeline is flushed and the correct result from the shadow latch is inserted back into the pipeline. Then, by replaying instructions, the processor operates correctly with little performance penalty. By taking timing error rate into consideration, voltage-scaling is utilized to allow processor to run robustly at the edge of minimum power consumption, with occasional timing error recovery for heavyweight computations. Razor enables better than worst-case design by removing design guard band used to guarantee ”always correct” operations, and has inspired a large amount of later research work (e.g., [32, 33]).

While Razor-like techniques are very effective for timing error correction in microprocessors with the help of instruction replay, they are very difficult, if not impossible, to be applied to general logic circuits, due to the high cost to checkpoint error-free states in them. In-situ timing error correction techniques that are able to mask errors without any rollback, are therefore very attractive. Existing techniques in this area can be classified into two categories: logic error masking and temporal error masking.

Among the few in-situ TEC techniques presented in the literature, most of them [108, 109, 110] rely on time-borrowing technique to correct timing errors, by delaying the arrival time of the correct data to the next logic level. As these techniques reduce the timing slack for the logic level that follows speed-paths, they have difficulty to handle the case when speed-paths exist in consecutive logic levels, limiting the applicability of such solutions. In [111], the authors proposed to synthesize a redundant logic block that is activated only when the speed-paths of the circuit are sensitized, and use it to mask timing errors on targeted paths. While interesting, their proposed synthesis algorithm is time-consuming and the redundant logic block incurs large area overhead.

□ **End of chapter.**

Chapter 3

Timing-Independent False Path Identification

3.1 Introduction

Logic circuits typically contain a large amount of paths down which signals cannot propagate in functional mode, known as *unsensitizable paths*, or simply *false paths* [90, 9, 96]. These paths should not be considered during the design and test of integrated circuits (ICs). For example, static timing analysis (STA) is an integral part in the physical design optimization process (e.g., timing-driven placement) for today's IC designs, used extensively to achieve circuit timing closure. Optimizing false paths, however, does not help to improve the performance of the circuit and the associated cost of optimization and iteration is expensive [87]. Similarly, targeting false paths during manufacturing test is unnecessary and may lead to over-testing of the circuit [10]. Therefore, how to effectively and efficiently identify false paths is an important and relevant problem for IC designers.

False paths can be categorized into three types: (i). *timing-don't-care false paths* with asynchronous or varying time budgets, such as those paths in asyn-

chronous clock domain crossovers; (ii). *timing-independent false paths* that are *logically unsensitizable* in functional mode; and (iii). *delay-dependent false paths*, which are logically sensitizable, but cannot be activated since one or more *on-path signals* are dominated by *side-input signals* all the time. Here the on-path signals refers to those signals that lie on the path being considered, while the side-input signals are the other signals driving the logic cells on this path. Generally speaking, identifying timing-don't-care false paths requires the knowledge of the design and they are typically picked up by designers manually. Various automated false path identification (FPI) techniques have been presented to identify the other two types of false paths (e.g., [89, 9, 92, 10, 96, 101]).

When identifying delay-dependent false paths, we need to calculate the signal arrival times to determine whether the on-path signals are always dominated by side-input signals. Consequently, if the circuit timing model is not accurate enough, it is possible that certain true critical paths are claimed to be false and hence are excluded from optimization, leading to a more serious problem of false indication of timing closure and possible silicon failures [92]. This problem is exacerbated with the ever-increasing process variations in advanced semiconductor technology [88], as signal arrival times become statistical values. Therefore, in this work, we focus on identifying timing-independent false paths, which cannot be sensitized under *any* arrival time condition. These paths are guaranteed to be safely removable in circuit timing analysis and manufacturing test.

The path sensitization criteria used in most prior FPI techniques are based on automatic test pattern generation (ATPG) like techniques [101]. To be specific, for a given path, it is assumed to be sensitizable if there exists a test vector pair (v_1, v_2) that activates a transition at the launch point of a path and propagates to its ending point; if, however, we cannot find such a test vector pair in any circumstances, this path is deemed as a false path. Generally speaking, ATPG-based FPI techniques

need to exhaustively search in an input space to prove that a targeted path to be false. To alleviate this problem, there have also been some implication-based false path identification techniques (e.g., [91, 94]). Essentially, these methods use the same criteria to identify false paths. The difference is that they try to prove the non-existence of test vectors using implication analysis instead of exhaustive search.

For a particular path, even if we are able to find a test vector pair that activates it, such a test may be *functionally-unreachable*. Consider a finite state machine encoded with one-hot code, the legal combinations of values in the circuit's state elements are only those with a single logic '1' and all the others logic '0'. Consequently, if a path can be activated in this circuit only with multiple logic '1's in these state elements (i.e., containing illegal states), this path is considered to be a true path based on the above criteria, but in fact it is functionally unsensitizable.

Motivated by the above, we propose novel techniques to identify those timing-independent paths that imply illegal states or other logic conflicts when they are activated. To be specific, we adapt the illegal state identification technique presented in [100] and integrate it into our FPI flow. For a given critical path, we present effective and efficient techniques to check whether it is a true path or a false path. We also present novel solutions to address the more general problem of finding as many *false paths* in a circuit as possible.

In our experimental results on ISCAS'89 and IWLS'05 benchmark circuits, we show that a large amount of the false paths identified using the proposed technique are treated as true paths with conventional FPI methods. In addition, by injecting the false paths identified with our technique into a commercial static timing verifier, the critical path delay for certain circuits can be significantly reduced, indicating existing STA tools are often over-pessimistic.

The remainder of this Chapter is organized as follows. Section 2 presents the preliminaries of this work and motivates this paper. In Section 3, we describe our

proposed method to examine whether a given path is true or false by taking illegal states of the circuit into consideration. Section 4 details our proposed techniques to identify as many timing-independent false paths in a circuit as possible. Experimental results on several ISCAS'89 benchmark circuits are then presented in Section 5 to demonstrate the effectiveness of our technique. Finally, Section 6 concludes this paper.

3.2 Preliminaries and Motivation

Most prior works determine whether a path is a true path by checking whether a test vector pair that sensitizes the path can be found according to the following criteria: for each logic element on a true path,

- When the on-path signal is a controlling value, there are no side-input signals with controlling values that arrived earlier;
- When the on-path signal is a noncontrolling value, all side-input signals are also noncontrolling values and they arrive no later than the on-path signal;

Various FPI techniques have been presented in the literature to find a sensitization vector pair for a path, typically using ATPG-like techniques [89, 9, 92, 10, 101]. When we consider the signal arrival time constraint in the above criteria, it is likely that a path is sensitizable under one delay model but false under another delay model [90]. Because of this, a critical path of the circuit might be mistakenly regarded as a delay-dependent false path due to inaccurate delay model, leading to false indication of timing closure and silicon failures. This problem is exacerbated with technology scaling, because it is increasingly difficult to construct sufficiently accurate timing models such that we are highly confident that delay-dependent false paths are guaranteed to be unsensitizable in silicon, especially considering the ever-increasing process variations in latest semiconductor

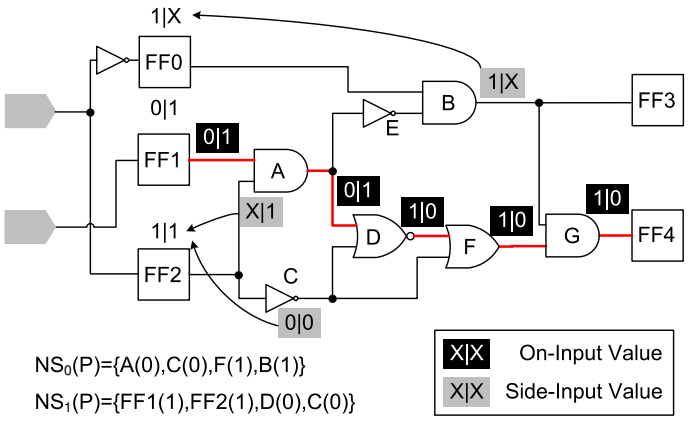


Figure 3.1: False Path Caused by Illegal State - An Example

technology. To safely remove false paths from STA [92], this paper is concerned about identifying those timing-independent false paths in the circuit, which are guaranteed to be false under *any* arrival time condition.

3.2.1 Motivation

As discussed earlier, existing FPI techniques regard a path to be a true path as long as a test vector pair can be found to sensitize it. However, none of them explicitly considers whether this found vector pair is functionally reachable or not. This is rather pessimistic because: *if a path is activated only with illegal states in the circuit, this path is a false path.*

Let us use the circuit shown in Fig. 3.1 as an example. Consider the path $P = \{FF1, A, D, F, G, FF4\}$, where a rising transition occurs at the launch point $FF1$ and propagates to the ending point $FF4$. To activate this transition, we need to have an input vector to drive logical values for the on-path signals $\{FF1, A, D, F, G\}$ to be $\{0,0,1,1,1\}$ in the first clock cycle and $\{1,1,0,0,0\}$ in the second cycle, as shown in the figure. One vector pair on $\{FF0, FF1, FF2\}$, $< 1,0,1;X,1,1 >$, can be found to sensitize this path. Consequently, traditional FPI technique (either

ATPG-based or implication-based method) would treat this path as a true path. A closer examination of the vector pair, however, tells us that it is functionally-unreachable. This is because, $FF0$ has to be the inverted value of $FF2$ in functional mode according to the circuit structure, and hence $\{FF0(1), FF2(1)\}$ is an illegal state cube. At this moment, however, we still cannot claim path P is a false path as we are not certain whether there exist other vectors being able to sensitize it without violating functional constraints. Nevertheless, we can find out that $\{FF0(1), FF2(1)\}$ is implied by the transitions occurring on path P through circuit structural analysis, as shown in Fig. 3.1 (denoted by the arrowed line). In other words, $\{FF0(1), FF2(1)\}$ is a necessary condition to activate path P, and hence we can conclude it is a false path.

The above example motivates us to take illegal states into consideration in false path identification and use implication-based techniques to efficiently determine whether a path is a false path or not.

3.3 False Path Examination Considering Illegal States

In this section, we consider the problem of evaluating whether a given path is a timing-independent false path. This procedure can be integrated into the inner loop of existing circuit optimization tools. That is, before we try to optimize the critical path reported by STA tools, we first quickly evaluate whether this path is a false path to avoid unnecessary optimization efforts.

3.3.1 Path Sensitization Criterion

For identifying timing-independent false paths, we have the following theorem:

Theorem 1 *A path is a timing-independent false path if and only if there exists at least one on-path signal such that when it is a non-controlling value, one or more*

of its corresponding side-input signals are with controlling values in functional mode.

proof 1 *The sufficiency of this theorem is obvious. That is, when an on-path signal is a non-controlling value while some side-input signals are with controlling values, this path cannot be activated and hence is a false path. As for the necessity of the theorem: (i). when the on-path signal is a controlling value, since we are considering timing-independent false paths where the side-input signals can arrive at any time, the path is sensitizable even if some side-input signals are with controlling values since they can arrive later; (ii). when the on-path signal and its corresponding side-input signals are all non-controlling values, similarly, the side-input signals are likely to arrive earlier so that the path can be activated to be a true path. Therefore, only if when the on-path signal is a non-controlling value while one or more of its side-input signals are with controlling values in functional mode, we can deem this path as a timing-independent false path.*

Apparently, with the above theorem, we can derive the following lemma:

Lemma 1 *A path is not a timing-independent false path if and only if, for any on-path signal, when it is a non-controlling value, all its corresponding side-input signals are also with non-controlling values in functional mode.*

To sensitize a path with either a rising transition or a falling transition at its launch point, all the on-path signals need to have transitions and hence they are applied with both logic '0' and '1' in two consecutive clock cycles. According to the above lemma, for a given path P, to determine whether it is a timing-independent false path, we only need to propagate logic '0' and logic '1' at its launch point separately and examine whether those on-path signals with non-controlling values (e.g., logic '1' for AND gate) and their corresponding side-input signals also with non-controlling values can co-exist during propagation in functional mode. They

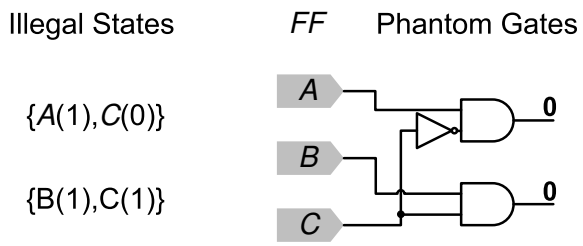


Figure 3.2: Representation of Illegal States as Phantom Gates

are called *the necessary set-up values to propagate logic '0'* and *the necessary set-up values to propagate logic '1'* for the path, denoted as $NS_0(P)$ and $NS_1(P)$, respectively. To sensitize the path, both $NS_0(P)$ and $NS_1(P)$ must be satisfiable.

Again, let us use the the example path $P = \{FF1, A, D, F, G, FF4\}$ shown in Fig. 3.1. When propagating logic '0' at launch point $FF1$, we need to justify $NS_0(P) = \{A(0), C(0), F(1), B(1)\}$ in functional mode; when propagating logic '1' at launch point $FF1$, we need to justify $NS_1(P) = \{FF1(1), FF2(1), D(0), C(0)\}$ in functional mode. Since $C(0) \rightarrow FF2(1)$ and $B(1) \rightarrow FF0(1)$, satisfying $NS_0(P)$ implies the existence of illegal state cube $\{FF0(1), FF2(1)\}$. Therefore, P is a timing-independent false path.

3.3.2 Path-Aware Illegal State Identification

In [100], the authors try to systematically identify all the illegal states in a circuit, which takes non-trivial runtime. For a particular path, however, only those illegal states that are within its fan-in logic cone need to be considered when determining whether it is functionally-sensitizable or not, denoted as *path-relevant illegal states*. Considering the fact that we are mainly interested in critical paths in timing analysis, we propose to adapt [100] and integrate it into our FYP flow as follows.

We first conduct STA on the targeted circuits to find critical paths and record their ending points. Next, for each ending point, we perform structural analy-

sis to trace the relevant flip-flops within its logic fan-in cone. Different from the method in [100], when building the justification schemes, we only consider those state cubes composed of the traced relevant flip-flops. This strategy not only facilitates to save the effort to construct a large number of useless justification schemes, but also automatically avoids to target those non-relevant illegal states. Hence, it improves the efficiency of both illegal state generation and the later false path identification procedures.

3.3.3 Proposed Examination Procedure

In the proposed method, we first extract illegal states of the circuit according to [100]. Next, we insert *phantom logic AND gates* into the circuit to represent them. As shown in Fig. 4.3, each phantom gate corresponds to an illegal state by linking its corresponding flip-flop (directly or through an inverter) with a *AND* gate, e.g., illegal state $\{A(1), C(0)\}$ in Fig. 4.3. This representation has the following advantages:

- we do not need to store illegal states as a separate list (e.g., a large number of independent conjunctive normal form [95]) by naturally integrating them into circuit structure;
- more importantly, by assigning the outputs of the phantom gates to be logic '0's, if a path is sensitizable only with illegal states, it would result in logic conflicts on the phantom gates and we automatically know it is a false path without necessarily generating the vector pair first and checking whether it includes any illegal state cube;

For a given critical path, according to the path sensitization criterion discussed earlier, we conduct two-pass processing to determine whether the path is a timing-independent false path, for justifying $NS_0(P)$ and $NS_1(P)$, respectively. For each

pass, we first assign logic values for the circuit nodes according to $NS_0(P)$/$NS_1(P)$ and logic '0' at all phantom logic gates, while leaving the other circuit nodes to be unknown values. Then, we conduct forward propagation and backward justification to obtain more logic values on those nodes that are initially unknown. If logic conflicts arise during the above logic reasoning process (either at the phantom gates representing illegal states with logic '1' or at a multi-fanout net with contradicting values at its branches), we can conclude the path is a timing-independent false path; otherwise, it is not.

3.4 False Path Identification

In this section, we consider the more general problem of identifying as many false paths in the circuit as possible, which is especially important for delay testing [91, 94]. Since the number of the paths is exponential to the circuit size, it is apparently impossible to use the technique shown in Section 4 to check path by path. Fortunately, as discussed earlier, timing-independent false paths would imply logic conflicts in the circuit at the phantom gates representing illegal states and/or multi-fanout nets in the combinational logic network. Based on this observation, we propose to identify false paths by targeting at their root causes structures, since the number of inserted phantom gates and multi-fanout nets are much less than the total number of paths in the circuit.

If a *path segment* cannot be activated in functional mode, all paths going through this segment are false paths. In particular, if any section of a false path segment is sensitizable, the false path segment is called a *prime false path segment*. For example, the path segment $\{D,F,G\}$ shown in Fig. 3.4 is a prime false path segment since sensitizing it implies illegal state cube $\{FF0(1),FF2(1)\}$ while both $\{D,F\}$ and $\{F,G\}$ can be activated in functional mode. Therefore, instead of considering a whole path to be false or not, we target at a path segment in each run

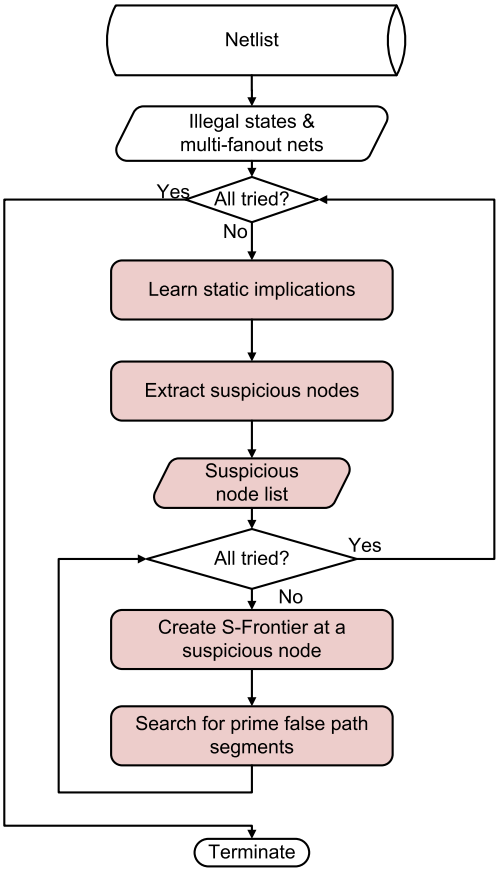


Figure 3.3: Flowchart for the Proposed Prime False Path Segment Identification Technique

because such representation is more efficient.

Based on the above, we propose novel algorithms to systematically identify prime false path segments in a circuit. The basic idea of our approach is to find the minimum path segment P_s whose necessary set-up logic values $NS_1(P_s)$ or $NS_0(P_s)$ imply logic conflicts in the circuit.

index	A(1)	B(1)	C(0)	D(1)	F(0)	G(1)
context	FF2(1)	FF0(1)	FF2(1)	FF2(1)	FF2(1)	FF0(1)

Table 3.1: Implication Lookup Table.

3.4.1 Overall Flow

Fig. 3.3 presents the overall flow for our systematic prime false path segment identification algorithm.

With given circuit netlist, we first extract illegal states and multi-fanout nets in the circuit. Next, we iteratively target one illegal state or one multi-fanout net in each run. Static implication learning is used to build implications for any possible internal circuit node that can imply values on the targeted illegal state elements or multi-fanout net (detailed in Section 3.4.2). These implication schemes are then stored in a lookup table as shown in Table 3.1, e.g., the entry $B(1)$ represents a implication $\{B(1) \rightarrow FF0(1)\}$ for the circuit shown in Fig. 3.4. For a given path segment, we are now ready to determine whether it is false by quickly checking whether the implications stored in the lookup table lead to any logic conflicts. For example, for path segment $P_s = \{D, F, G\}$, it is false since $D(1)$ and $G(1)$ imply the illegal state cube $\{FF0(1), FF2(1)\}$ according to the implications stored in Table 3.1.

However, our objective is to identify as many prime false path segments as possible, instead of checking a specific path segment is false or not. Apparently, we cannot afford to consider every circuit node to be the starting point of a false path segment. Fortunately, based on the implication lookup table built earlier, we can extract a set of suspicious nodes as the possible starting point of prime false path segments (detailed in Section 3.4.3). Then, for each suspicious node, we create a so-called *S-Frontier* to record the path segment we have visited, which contains the following items:

- *segment* is used to record all the circuit nodes on the current path segment;
- *launch value* with 1/0 represents that we propagate logic '1/0' at the launch point of the segment stored in *S-Frontier*;
- *implied cube* records all the corresponding implication schemes of *segment* in the implication lookup table;

Then, false segment identification is conducted by propagating *S-Frontiers*, as detailed in Section 3.4.4. Our program terminates when all the illegal states and multi-fanout nets have been considered.

Again, let us use the example circuit shown in Fig. 3.4 to illustrate our identification procedure. For illegal state cube $\{FF0(1), FF2(1)\}$, we build its corresponding implication lookup table as shown in Table 3.1. Consider the suspicious node D with $D(1) \rightarrow FF2(1)$, we create a *S-Frontier* with launch value '$1'$ at node D and then propagate it along the path. Accordingly, newly-implied values are continuously added into *implied cube*, and once we reach node G, we obtain {FF0(1),FF2(1)} in the *implied cube* of the updated *S-Frontier* and hence we find a false path segment $\{D, F, G\}$.

3.4.2 Static Implication Learning

For each targeted illegal state or multi-fanout net, single node implication is utilized to learn which circuit nodes can justify the expected values on them.

Consider the illegal state $\{FF0(1), FF2(1)\}$ for the circuit shown in Fig. 3.4, we first conduct implication for their inverse values $FF0(0)$ and $FF2(0)$ independently. For example, we can obtain $\{FF0(0) \rightarrow B(0)\}$ since $FF0(0)$ is a controlling value for the *AND* gate. Similarly, we have $\{B(0) \rightarrow G(0)\}$, and hence $\{FF0(0) \rightarrow G(0)\}$ according to the transitivity of implication. By applying counter-positive law, the following implications are stored in the implication

lookup table: $\{B(1) \rightarrow FF0(1)\}$ and $\{G(1) \rightarrow FF0(1)\}$. When conducting static implication learning on multi-fanout net, we imply both logic '0' and logic '1' from the targeted multi-fanout net, apply counter-positive law and store the learned information in the same format.

3.4.3 Suspicious Node Extraction

The starting point of a *S-Frontier* determines whether a false segment can be found and which false segment can be found. Therefore, we need to carefully extract the set of suspicious starting points to create *S-Frontiers*. The selection should satisfy the following requirements: (i). all the possible false segment can be detected; (ii). the selected starting points should be as less as possible.

After static implication learning, we define those nodes that have implications as the *affected nodes*, e.g., nodes A, B, C, D, F and G in Fig. 3.4. Only affected nodes can serve as the starting point of a prime false segment because the other nodes do not contribute any implications to justify values on the targeted illegal states or multi-fanout nets, and hence they cannot be part of the prime false

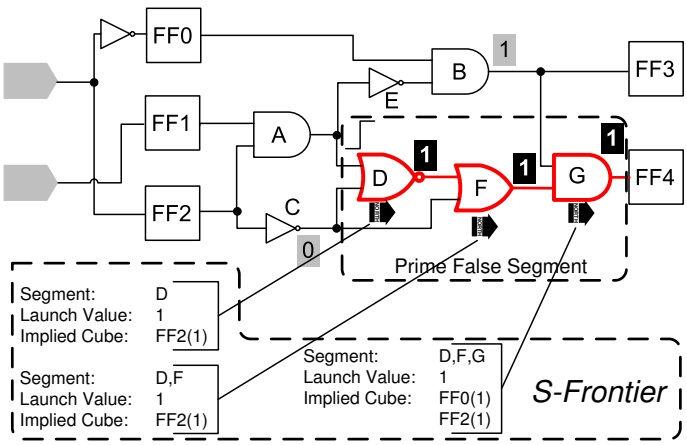


Figure 3.4: Example to Demonstrate False Path Identification

segment. At the same time, not all affected nodes need to be considered as the starting point of prime false path segments. Take node C as example, which has one implication $\{C(0) \rightarrow FF2(1)\}$. It has two following logic elements D and F. To propagate C(0), we can only generate $D(1)$ and $F(0)$. As can be easily observed in Table 3.1, however, both $D(1)$ and $F(0)$ imply $FF2(1)$, therefore making $\{C(0) \rightarrow FF2(1)\}$ in fact a redundant implication scheme.

Based on the above observation, we conduct pre-processing for the affected nodes and we remove those nodes that only contain redundant implication schemes. The rest of the affected nodes are defined as the *suspicious nodes* (e.g. nodes D, F and G), and *S-Frontier* can only be created and propagated from them.

3.4.4 S-Frontier Propagation

The *S-Frontier* propagation is essentially a breadth-first search process. Firstly, an *S-Frontier* is created at each suspicious node with launch value 0(1) and is propagated along the path. Once an *S-Frontier* reaches a multi-fanout net, it will be split into several copies and delivered to all branches of the multi-fanout net.

After propagating *S-Frontier* to a new node, we first add the new node at the end of *segment* stored in *S-Frontier*, and we add new implications to its *implied cube*, if any. If the on-input of the current node is with non-controlling value, non-controlling value is assigned on the side-inputs and implications for the side-input signals will be also added into *implied cube* of this *S-Frontier*. As shown in Fig. 3.4, let us consider the propagation of *S-Frontier* from node $F(1)$ to node $G(1)$. Since the on-input of node G is non-controlling value, we also need to assign B with logic '1' and add the implication $\{B(1) \rightarrow FF0(1)\}$ into the *implied cube*. Also, we should check whether the implication schemes with the new node include all the implications with the first node kept in *segment* of the *S-Frontier*. If so, the implication schemes of the first node are redundant and hence we update

Benchmark	Flip-flop #	Illegal states (#)[100]	Path-Related Illegal states (#)	Longest paths (#)	PrimeTime WCD	[91] False path (#)	[91] WCD	Proposed False path (#)	Proposed WCD	Proposed Runtime (s)
s1196	18	138	9	1000	**4.05**	78	**4.05**	82	**4.05**	1.67
s1238	18	126	5	1000	**4.19**	112	**4.19**	116	**4.19**	1.15
s5378	179	8516	392	1000	**3.89**	696	**3.89**	742	**3.89**	1.38
s9234	211	1109	271	1000	9.01	798	**8.99**	798	**8.99**	1.85
s13207	638	82651	1185	5000	13.06	4692	12.89	4986	**12.15**	37.62
s38417	1636	90983	1220	5000	11.42	3582	9.89	4440	**9.76**	8.9
s38584	1426	63558	791	5000	15.07	4380	15.04	4624	**15.03**	25.62
wb_conmax	3316	2083	122	5000	**4.72**	89	**4.72**	102	**4.72**	16.9
DMA	3131	2097	97	5000	5.99	2330	**5.98**	2375	**5.98**	49.05
pci	3720	2442	141	5000	5.09	943	5.09	1120	**5.08**	12.083
usb	1960	30676	388	5000	4.34	2102	4.34	2322	**4.32**	64.3
ethernet	10752	4205	195	5000	9.78	355	9.33	431	**9.02**	236.267
vga_lcd	17265	3096	90	5000	**10.36**	539	**10.36**	793	**10.36**	660.883

Table 3.2: Experimental Results for Static Timing Analysis.

S-Frontier by removing this node to guarantee that the obtained false path segment is a prime one.

Finally, since we may obtain the same prime false path segments starting from different suspicious nodes. Before propagating an *S-Frontier*, we first check whether its starting point is the first node of an existing false path segment. If so, we simply delete this *S-Frontier* as it must have been propagated earlier.

3.5 Experimental Results

To evaluate the effectiveness of the proposed solution, we perform three sets of experiments on several ISCAS'89 and IWLS 2005 benchmark circuits. Our experiments are performed on a 2GHz PC with 1GB memory.

In our first experiment, we extract a number of critical paths using Synopsys's

static timing analyzer PrimeTime [98]. As shown in Table 2, we use the proposed method and the implication-based method presented in [91] to examine them. As can be observed, both techniques determine that a large percentage of the reported critical paths are actually false paths. For example, for s13207, by applying our method, only 14 paths out of the the 5000 longest paths are deemed as true paths; while 308 paths are reported to be true with [91]. Consequently, the proposed method is more effective for false path identification.

In terms of timing analysis result, Columns 6, 8 and 10 present the worst case delay (WCD) reported by PrimeTime[1], [91] and the proposed method, respectively. It can be observed that the three methods report the same WCD for several circuits (e.g., s1196, s5378 and vga_lcd). This is because the longest paths of these circuits are in fact true paths. For other circuits, the true critical path delay can be reduced after considering the false paths. Due to that our method can detect more false paths, our WCD results is shown to be less pessimistic when compared to [91]. In particular, for s13207, the worst case timing delay reported by us is $12.15ns$ while [91] returns $12.89ns$. Our proposed FPI technique is very efficient as indicated by the short runtime, which is an extremely important feature since it can thus be tightly integrated into the inner loop of circuit optimization tools. It should be highlighted that the above runtime includes the time used for path-aware illegal state identification, and as can be seen in Columns 3 and 4, the number of illegal states related to long paths is quite small.

In the second experiment, we present results for our systematic prime false path segment identification algorithm, as shown in Table 5.2. Column 2 is the number of prime false path segments acquired with the proposed method. We then feed these segments into an ATPG engine built on top of an academic tool *Atalanta* [93] to check whether we can find a vector pair to sensitize them. Column 3 gives the

[1]PrimeTime can report true critical path delay with its own FPI feature.

Benchmark	False seg. $\mathcal{F}$ (#)	Justified seg. $\mathcal{J}$ (#)	Unjustified seg. $\mathcal{U}$ (#)	Improvement (%) $I = \frac{\mathcal{J}}{\mathcal{U}} \times 100\%$	Runtime (s)
s1196	108	18	90	20.00	56.35
s1238	202	16	186	8.60	60.59
s5378	755	316	439	71.98	85.9
s9234	929	106	823	12.88	93.85
s13207	80510	11826	68684	17.22	356.32
s38417	68864	10647	58217	18.29	290.52
s38584	37168	5056	32112	15.74	593.65
wb_conmax	941	42	899	4.67	127.77
DMA	414	36	378	9.52	923.26
pci	5052	4644	408	1138.24	1561.48
usb	2351	1433	918	156.10	1738.60
ethernet	979	153	826	18.52	665.15
vga_lcd	1105	97	1008	9.62	464.58
Average				107.28	

Table 3.3: Experimental Results for Prime False Path Segments.

number of segments that the ATPG engine can find a solution to activate them. Conventional method therefore would regard them as true path segments. On the other hand, we also present, in Column 4, the number of segments that the ATPG cannot find a vector pair to activate them. As can be seen in Column 5, in average we can find 107% more false path segments using our proposed method, each of which may correspond to multiple false paths. In particular, for some extreme cases such as *pci*, most of the false segments are treated as true path segments with ATPG-based FPI technique. It should be emphasized that, if we are not able to find a sensitization vector pair for a path segment because ATPG engine aborts computation, we cannot conclude such a path segment is false. The improvement results reported in Table 5.2 is hence rather conservative. It is also important to note that ATPG-like FPI techniques operate on a 'path-by-path' basis and cannot identify prime false path segments efficiently.

Benchmark	Implication-based [91] $\mathcal{B}$ (#)	Proposed $\mathcal{O}$ (#)	Improvement (%) $I = \frac{\mathcal{O}-\mathcal{B}}{\mathcal{B}} \times 100\%$
s1196	843	1509	79.01
s1238	1809	2687	48.54
s5378	4908	6389	30.18
s9234	282567	322827	14.25
s13207	2927528	20526546	601.16
s38417	253691	297198	17.15
s38584	1198836	1745552	45.60
wb_conmax	653128	680372	4.17
DMA	9945	13261	33.34
pci	1569350	13076785	733.26
usb	79851	130086	62.91
ethernet	50417	67023	32.94
vga_lcd	119472	149083	24.78
Average			132.87

Table 3.4: Experimental Results for False Paths Compared against Implication-Based Method.

Finally, we compare the number of false paths identified with [91] and that with our method, as shown in Column 2 and Column 3 in Table 3.4. Since we only identify prime path segment, for fair comparison, the path counting algorithm in [91] is implemented to count the corresponding false paths for our identified segments. As can be observed, our proposed method are able to find much more false paths than [91], around 132% more on average. In particular, for s13207 and *pci*, we identify six times and 7 times more false path when compared to [91], respectively. We attribute the reason for our improvement to the fact that a large number of illegal states are considered in false path identification.

3.6 Conclusion and Future Work

Effective removal of false paths from static timing analysis is a critical task to achieve timing closure for state-of-the-art IC designs. It also facilitates to identify untestable path delay faults. Traditionally, a path is regarded as a true path as long as a vector pair can be found to sensitize it. In this work, we show that the above criteria is rather pessimistic since certain paths are activated only with illegal states in the circuit and hence they are functionally-unsensitizable. Based on this observation, we develop efficient and effective FPI techniques to identify those timing-independent false paths that cannot be sensitized under any signal arrival time condition in integrated circuits. Experimental results on ISCAS'89 benchmark circuits show that our proposed technique are able to find much more timing-independent false paths than existing implication-based and ATPG-based FPI techniques.

In our future work, we plan to integrate the proposed efficient FPI technique into existing circuit optimization tools (e.g., the open-source logic synthesis tool *ABC* [86]) and conduct experiments on large industrial circuits.

□ **End of chapter.**

Chapter 4

PSN Aware Pseudo-Functional Delay Testing

4.1 Introduction

Power supply noise (PSN) has an ever-increasing adverse impact on circuit timing with technology scaling. As demonstrated in [34], a 1% voltage change can cause approximately a 4% change in gate delay in 90-nm, 0.9-V technology. Consequently, it is essential to take PSN effects into consideration in at-speed delay testing to guarantee that integrated circuits (ICs) fully meet customer performance expectations.

Some prior works advocated to generate test patterns that induce maximum PSN effects in delay testing to ensure the timing correctness of the shipped IC products even in the worst-case scenario [35, 36, 37]. As shown in [107], however, at-speed scan patterns can be up to 20% slower than any functional patterns due to the discrepancy between functional mode and test mode in scan-based testing. Consequently, such methodologies may lead to over-testing and induce significant test yield loss. To resolve this issue, on the other hand, various low capture-power

and low IR-drop testing techniques were presented to reduce the PSN effects in at-speed testing [46, 47, 48, 50]. These test methodologies, unfortunately, lead to the concern for under-testing. That is, if we over-restrict the PSN effects during delay testing, some defective chips that cannot meet circuit timing requirement may pass manufacturing test, leading to test escapes [38]. Therefore, to avoid both over-testing and under-testing, the real question is: *How can we exercise the worst-case timing of the circuits under test (CUTs) in their functional mode during manufacturing test?*

To tackle the above problem, a layout-aware pseudo-functional testing technique targeting path delay faults was presented in [49]. By extracting functionally-unreachable states (also known as illegal states or functional constraints) in the circuit and feeding them into automatic test pattern generation (ATPG) tools, [49] first generates functionally-reachable test cubes for every true critical path in the circuit. Then, they used a heuristic to fill the don't-care bits in the test cubes to maximize power supply noises on critical paths under the consideration of functional constraints. As pseudo-functional testing naturally minimizes the possibility of over-testing while their proposed X-filling strategy is able to maximize PSN effects, [49] is able to simultaneously reduce both test overkills and test escapes.

Although the above pseudo-functional path delay testing technique is quite effective, it is inherently non-scalable due to the exponential number of paths in the circuit and hence can only be used to generate a few top-up patterns for selected critical paths. Today, timing-aware ATPG for transition faults has gained wide acceptance in the industry to detect those small delay defects (SDDs) that cause quality and reliability concerns for high-performance ICs. In this work, we present novel pseudo-functional ATPG techniques to simultaneously reduce both test overkills and test escapes in SDD testing. Firstly, by taking the circuit layout information into account, functional constraints related to critical paths are ex-

tracted. Then, we generate functionally-reachable test cubes for SDD faults in the circuit. Finally, we use ATPG-like algorithm to generate switching activities that pose the worst-case power supply noises on sensitized critical paths under the consideration of functional constraints. Experimental results on benchmark circuits demonstrate the effectiveness of the proposed technique.

The remainder of this chapter is organized as follows. Section 2 reviews related work and motivates this paper. In Section 3 and Section 4, we detail our proposed methodology. Experimental results on several large ISCAS'89 and IWLS'05 benchmark circuits are then presented in Section 5 to show the effectiveness of the proposed solution. Finally, Section 6 concludes this paper.

4.2 Preliminaries and Motivation

Layout-Aware Pseudo-Functional Testing for Critical Paths

With a large set of identified illegal states, applying pseudo-functional patterns naturally minimizes the possibility of over-testing, but under-testing may occur without taking PSN effects into consideration during the ATPG process. To address this issue, in [49], the authors proposed a pseudo-functional test pattern generation technique to maximize the PSN effects on selected critical paths, targeting path delay faults.

As [49] is well-related to this paper, we briefly review it here. In [49], a so-called *PSN effect weight* (PEW) was proposed to evaluate the PSN effect caused by transitions on aggressors[1]. As the location of the aggressors should be close enough to that of the victim so that they are competing for power supply, the authors defined a so-called $EffectiveRange$ as a pre-defined maximum distance

[1]For a critical path under test, the on-path logic cells and the cells that induce power supply noise on them are denoted as victims and aggressors, respectively.

between the aggressors and the victims. Within this range, *PEW* is defined as follows.

$$PEW = 1 - |X_{agg} - X_{vic}|/EffectiveRange \tag{4.1}$$

where X_{agg} and X_{vic} denote the row-coordinate of the aggressor and the victim, respectively, which represents the closer an aggressor is to a victim cell, the higher PSN it induces on it.

At the same time, as pointed out in [37], the transition type of aggressor cells (e.g., rising or falling) also plays an important role for PSN effects on a victim cell. Consider an on-path victim cell in Fig. 4.1, to maximize power supply noise on it, for those aggressor cells that are in the same row, they are desired to have the same transition type as the victim cell; for those aggressor cells that are in different row but share a common power wire with it, they are desired to have a rising transition; while for the remaining aggressor cells that are in different row but share a common ground wire with it, they are desired to have a falling transition. Based on the above, [49] also defined a *probability-based transition PSN metric* to evaluate the impact of X-bits on the PSN of targeted path from transitions of relevant gates. Based on above, a novel X-filling heuristic is proposed to assign logic values for X-bits in the test cube to maximize the PSN effects on selected critical paths under functional constraints.

4.2.1 Motivation

As shown in [49],simply maximizing or minimizing PSN effects in at-speed delay testing is not a good strategy since such one-sided solutions are inevitable to result in the concern of the other side. The work in [49] made a good attempt to tackle this problem considering path delay faults. However, since the number of paths in a circuit increases exponentially as the circuit size grows, it is infeasible

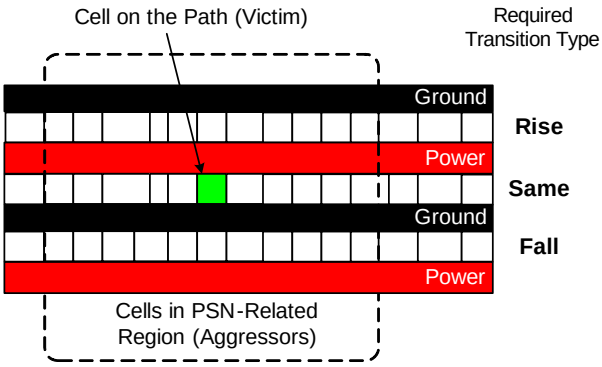


Figure 4.1: Example of PSN-Related Cells.

to consider every path in the circuit explicitly. Instead, only those critical paths identified by timing analysis tools can be considered during test generation. Unfortunately, the ever-increasing process variation makes circuits' timing behavior unpredictable, and hence there might be a large number of paths being critical. Consequently, only a subset of critical paths can be tested based on path delay fault model, which cannot guarantee test quality and can only be used to generate some top-up patterns.

Due to the above, small delay defect testing has been widely accepted by the industry, wherein we try to detect transition faults by propagating their faulty effects through long paths whenever possible. Compared to path delay testing, the number of SDD test patterns increases almost linearly with the circuit size and we can achieve good transition fault coverage by being able to flexibly choosing the sensitization paths.

The above motivates us to take the circuit layout into consideration and maximize power supply noise effects for SDD testing under the consideration of functional constraints. By doing so, we are able to achieve high quality delay testing by simultaneously reducing both test escapes and test overkills of the CUTs.

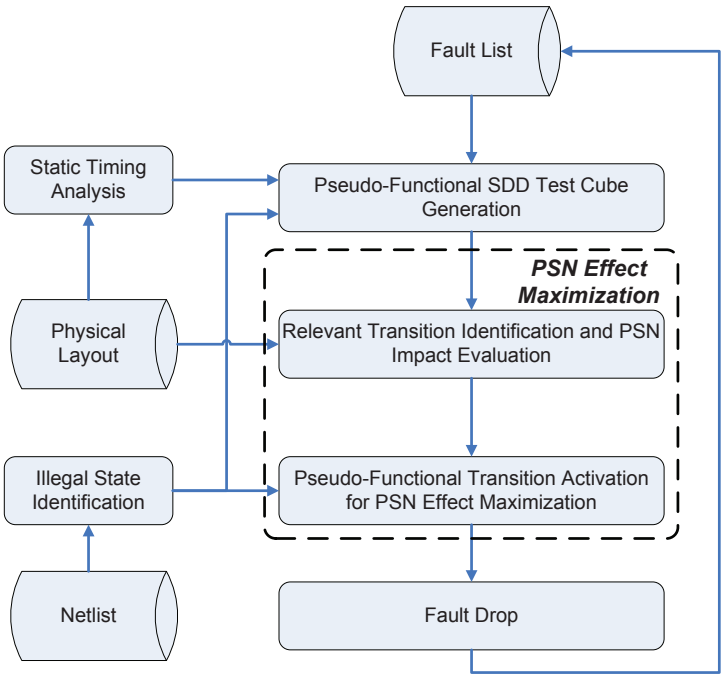


Figure 4.2: Main flow of our proposed pseudo-functional SDD test generation methodology

4.3 Proposed Methodology

Fig. 5.4 presents the overall framework for our proposed layout-aware pseudo-functional SDD test pattern generation procedure. Given the layout and netlist information of circuit, we first obtain critical paths with commercial timing analysis tool, and then extract illegal states related to these critical paths based on the method presented in [100, 49]. As can be observed from Fig. 5.4, our proposed ATPG flow mainly contains two parts: (i). pseudo-functional SDD test cube generation; and (ii) PSN effect maximization.

To generate pseudo-functional SDD test cube, we extend the conventional SDD test generation method presented in [54] by integrating the functional constraints checking and breaking mechanism, which is to sensitize SDD through critical

paths whenever possible, meanwhile, it guarantees that no illegal states is included in the test cube. In order to satisfy functional constraints, as in [49], we represent functional constraints as *phantom gates* and virtually inserts these gates in the CUT as shown in Fig. 4.3. By doing so, we can selectively activate a subset of functional constraints by assigning logic '1' at the output of the corresponding phantom *AND* gates during the test pattern generation process. For example, in order to avoid any one of $\{A(0), C(1)\}$ and $\{A(1), B(1)\}$ to appear in test cube, we will assign logic '1' on both $F1$ and $F2$. Then, suppose that input B has been set as logic '1' during pattern generation, implication function automatically implies logic '0' at A to break illegal state $\{A(1), B(1)\}$. Similarly, C will be then assigned as logic '0' to break illegal state $\{A(0), C(1)\}$.

Next, in terms of PSN effect maximization, for the critical paths sensitized by the test cube, we first parse the circuit layout to identify those relevant transitions that may induce power supply noise on it, and estimate the delay impact caused by each transition. Then, several algorithms are introduced to justify as many relevant transitions as possible by judiciously filling the X-bits of the test cubes without violating functional constraints, so that the PSN effects incurred by the final test pattern is nearly the worst-case scenario that exists in functional mode (detailed in Section 4). After obtaining each pattern as above, we drop those transition faults that are located on the same sensitized critical path. To note, we do not conduct fault simulation and drop the other detected transition faults since PSN effects are not considered for their sensitized paths yet.

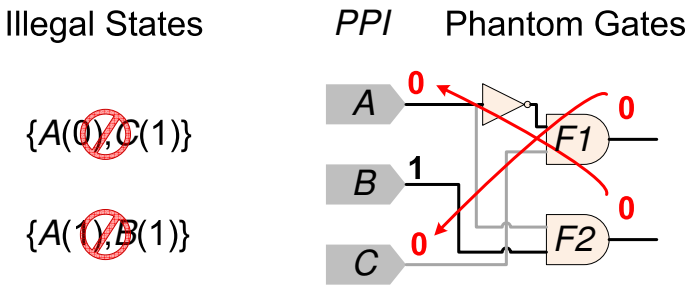


Figure 4.3: Insertion and Activation of Functional Constraints as Phantom Gates

4.4 Maximizing PSN Effects under Functional Constraints

After obtaining pseudo-functional test cubes, our objective is to fill the X-bits to maximize PSN effects on the sensitized critical path under functional constraints of each SDD test pattern. We do not simply reuse the probability-based X-filling technique presented in [49] due to its computational complexity[2]. Instead, we propose an effective ATPG-like technique to fill X-bits, which is able to directly justify the targeted transitions on relevant aggressors to required values by filling X-bits in the test cubes. As we cannot justify all the relevant transitions simultaneously, the main challenge is how to effectively justify those *highly-relevant* transitions as many as possible without violating functional constraints.

To tackle the above problem, we first parse the circuit layout to identify those relevant aggressors that may induce power supply noise affecting the targeted fault's behavior. Then based on the distance and required transition type of aggressors, the $PEW_{agg-vic}$ is calculated according to Eq. (1) for each pair of the

[2]As there is no direct correlation between X-bits within test cube and the relevant signals with required transitions, time-consuming probability-based simulation is conducted in [49] to guide the filling procedure for every test pattern.

on-path victim cells and their respective aggressors. The PSN impact for relevant transition on a specific aggressor is then calculated by summing up all the $PEW_{agg-vic}$ between this transition and all the on-path victims and we denote it as *transition weight* (TW). *By formulating the problem as above, our objective becomes to maximize the total TW by justify relevant transitions as many as possible using X-bits in the test cube, without violating functional constraints.*

4.4.1 Pseudo-Functional Relevant Transitions Generation

The task to justify the maximum number of compatible transitions that lead to PSN effects is quite challenging, as certain relevant transitions cannot be justified simultaneously due to logic conflicts in the circuit. Since it is obviously unacceptable to enumerate all the possible combinations, we propose to justify the set of relevant transitions in an *incremental* manner.

The flowchart of our proposed algorithm is shown in Fig. 4.4. It is composed of four main parts. Firstly, we conduct a fast pre-processing step on the relevant transition set, and try to form a *compatible transition graph* (*CTG*) in such a way that some possible concurrently-justifiable transitions are identified by logic implication. Next, we conduct *incremental transition extraction*, in which we extract the maximum clique on the CTG. The subset of transitions on this clique is our focus in the next *transition justification* step, wherein several heuristics are used to justify as many as transitions in this subset as possible[3]. To avoid being trapped into local optimal solution, we also equip our algorithm with the flexibility to search within certain range (denoted as *backtracking mechanism*), so as to find better solution in the end. After processing each clique, we update *CTG*, find the maximum clique and conduct transition justification again. The above procedure

[3]Not all transitions can be concurrently justified even for this subset in the clique as logic implication is usually incomplete.

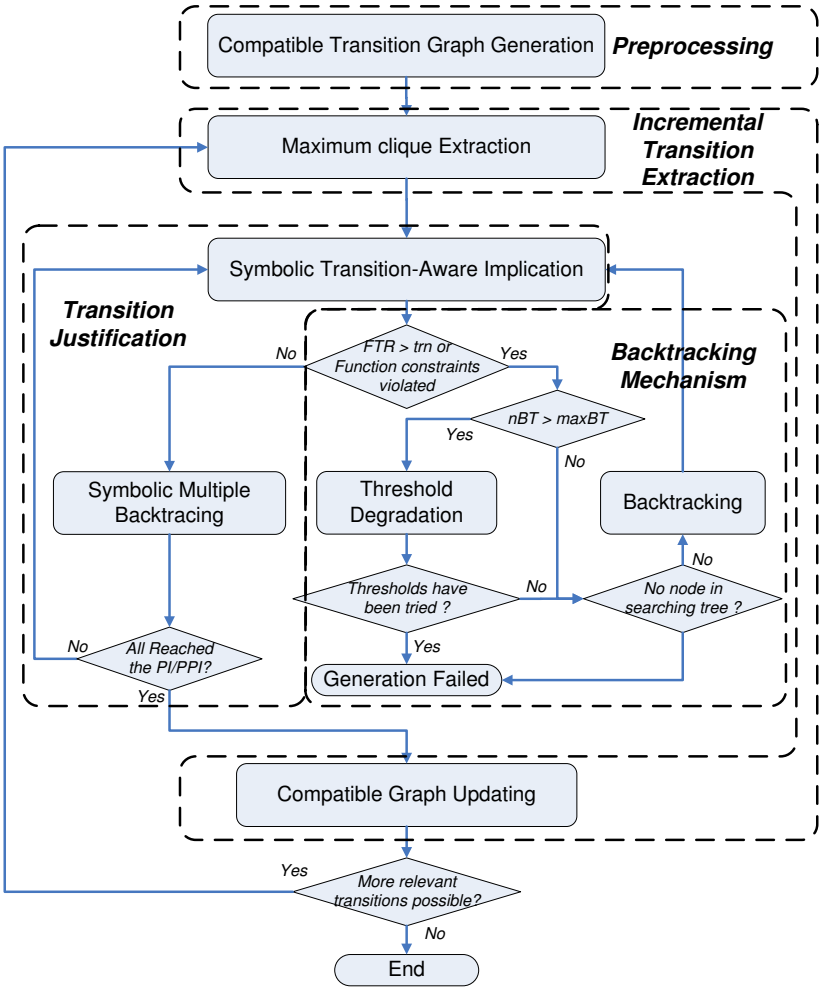


Figure 4.4: Flowchart for the pseudo-functional relevant transition generation algorithm

iterates until no more relevant transitions can be achieved under the consideration of functional constraints.

Compatible Transition Graph Generation

The relevant transitions identified from the layout information may logically-conflicting with each other. Although we are able to identify these conflicts during the logic

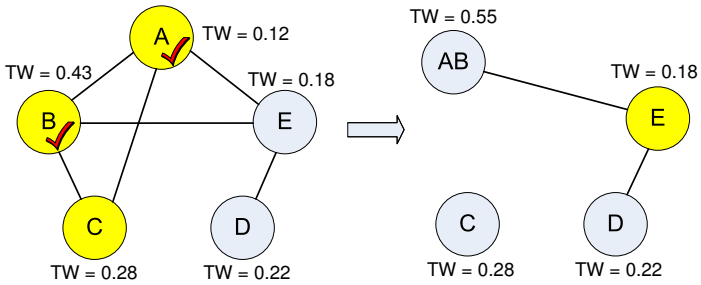


Figure 4.5: Example to show CTG updating

value justification process, too many conflicting transitions will dramatically increase the processing burden and hence severely impact the runtime of our solution.

To resolve this problem, we conduct a pre-processing step to reduce the problem complexity, by using logic implication to build the so-called *compatible transition graph* (*CTG*) as follows. Given a test cube, two relevant transitions are treated as compatible if there is no conflict after applying logic implication for the two transitions. As shown in Fig. 4.5, every node on the *CTG* denotes a relevant transition which is weighted by TW value, and two transitions are connected with a edge if they are compatible. It is worth noting that building such *CTG* graph is very efficient since the implication can be conducted very fast and *CTG* graph construction is a one-time effort.

Incremental Transition Extraction

Before transition justification, we first extract the maximum clique of CTG in a greedy manner and target the transitions within it.

Updating Compatible Transition Graph

After each round of *transition justification* , the CTG graph is to be updated. Take the *CTG* depicted in Fig. 4.5 as an example. The initial clique composes of

transitions A, B and C. Then, suppose A and B are justified while C fails. since it is still possible that E is justifiable together with A and B, we update the CTG by merging the justified transitions into a super-node in the graph and then cutting off the links between the super-node and the unjustified transitions. Next, we fix those transitions that have been justified and only put the untried transitions that have links with the super-node into the next round of justification process. This mechanism guarantees that our algorithm is performed on the complete relevant transition set, i.e., not restricted by the maximum clique that we have selected at the very beginning.

Transition Justification

We introduce two techniques to justify the required transitions as many as possible. As both of them are based on the so-called *symbolic justification mechanism*, we discuss it first.

To justify a transition, it is necessary to concurrently justify two values for the same node in the circuit in two consecutive timeframes. As the example shown in Fig. 4.6, wherein the circuit has been unrolled and we want to justify three transitions at B, D and E. Taking transition at D as an example, there are two objects located at B and B' in the two timeframes, respectively. For justifying $B = 1$, we need to justify both $C = 0$ and $I = 1$ on different branches since logic '1' is the non-controlled value for NOR gate. Initially, three values need to be justified for one transition, and this number keeps increasing as justification proceeds. Suppose that a particular transition is failed because it cannot be justified at any branch, it is meaningless to justify the rest of the branches. Hence, we need to hold such information at the unjustified gates to indicate which transitions it is related to. Moreover, the to-be-justified transitions may be treated differently as they may have different impact on PSN effects.

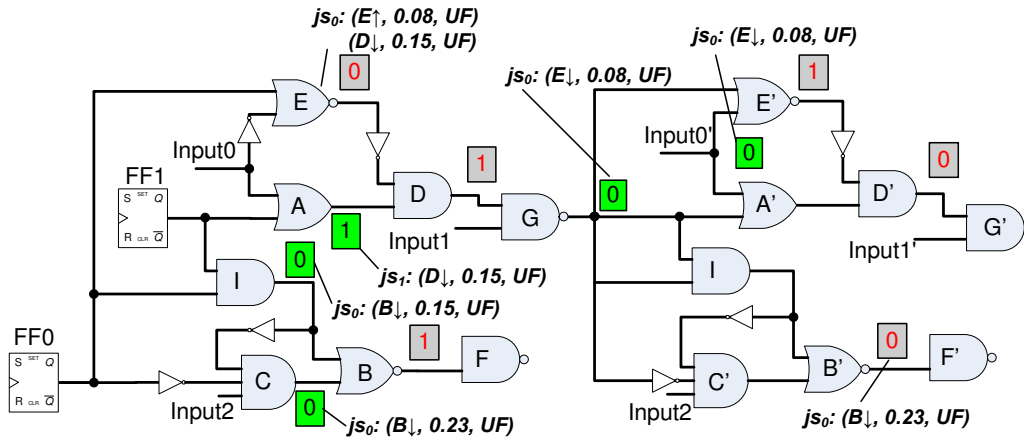


Figure 4.6: Example to show symbolic justification mechanism

To represent all of above, we introduce two sets of the so-called *justifying symbols* (js_0 and js_1) at the unjustified gates. Each *justifying symbol* is composed of a three-tuple element including the correlated transition, the weight of the transition and the state of the transition (i.e., the un-failed transition is labeled as UF while the failed transition is labeled as F). js_0 and js_1 list the set of transitions that require the gate to be '0' and '1', respectively.

Symbolic Multiple Backtracing

Starting from several unjustified values, we employ multiple backtracing technique to trace them concurrently. In conventional ATPG, the technique propagates n_0 and n_1 to indicate how many times that the signal is required to be logic '0' and logic '1', and it simply treats every unjustified value equally important. For our problem, however, different transitions have non-equal weight as indicated by their TW values, and we need to have higher priority to justify those transitions with larger TW. At the same time, we also need to remove the state of some relevant transitions, since it does not make sense to consider those failed transitions. Therefore, we propose a symbolic multiple backtracing technique that propagates

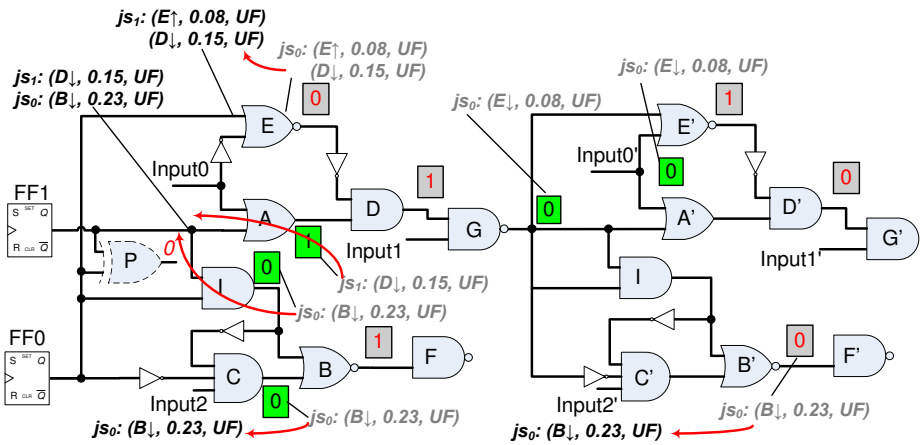


Figure 4.7: Example to show symbolic multiple backtracing

the justifying symbol list js_0 and js_1, based on the following rules:

- For NOT gate, duplicate the js_0/js_1 to js_1/js_0 of its fan-in gate;
- For the other kinds of gates, if vo at output and vi at input are the non-controlled and non-controlling values for the gate respectively, we duplicate the js_{vo} to the js_{vi} of all the fan-in gates; otherwise when they are the controlled and controlling values for the gate respectively, we duplicate the js_{vo} to the js_{vi} of the easiest justifiable fan-in gate, which is defined as the gate closest to the primary/pseudo-primary input.

Following the example shown in Fig. 4.6, we use Fig. 4.7 to illustrate the procedure for justifying symbol propagation, which is depicted by the arrowed lines. All the backward propagations stop at multi-fanout nets or at the inputs. For example, for $FF1$, some transitions require it to be logic '1' while others require it to be logic '0', hence we need to make value decision on such multi-fanout nets, which is detailed in the following *symbolic transition-aware implication*.

Symbolic Transition-Aware Implication

There are two major tasks in the symbolic transition-aware implication procedure. The first one is to guarantee no functional constraints is violated during justification process. This is achieved by representing functional constraints in the same way as depicted in Fig. 4.3. As the example shows in Fig. 4.8, suppose flip-flops $FF0$ and $FF1$ should have the same value, we insert a phantom *XOR* gate P into the circuit and assign logic '0' to it. By doing so, we can detect the logic conflict on P if and only if these two flip-flops are assigned with the same logic value. Once any functional constraint is violated, we stop implication for backtracking by inverting the logic value that is last assigned at the multi-fanout net.

The second task is to make value assignment decision when multi-fanouts are reached during the multiple backtracing process. We can observe that different implication orders result in failures of different relevant transitions. As shown in Fig. 4.8, starting from multi-fanout $FF0$ first, it is assigned with logic '1' since the js_0 set is empty on it. Next, when making decision on $FF1$, we specify it as logic '0' because the TW on B is larger than that on D. Functional constraint violation is then detected on gate P, and backtracking is conducted to invert the logic value on the last multi-fanout $FF1$. Consequently, transition on B fails.

However, suppose the decision making order is first to assign '0' on $FF1$ and then specify '1' on $FF0$, the functional constraint violation results in backtracking to invert the logic value on $FF0$. In this case, the justifications for $E(0)$ and $A(1)$ are both dependent on $Input0$. According to *justifying symbols* on these two nodes, we specify $Input0$ as logic '0'. This assignment inverts the logic value on A, and correspondingly, the logic values on D, G and E' are all flipped, making transitions on both E and D failed.

Based on the above observation, we propose a symbolic transition-aware implication procedure that heuristically reduces the total amount of weighted failed transitions as follows. For each reached multi-fanout during the multiple back-

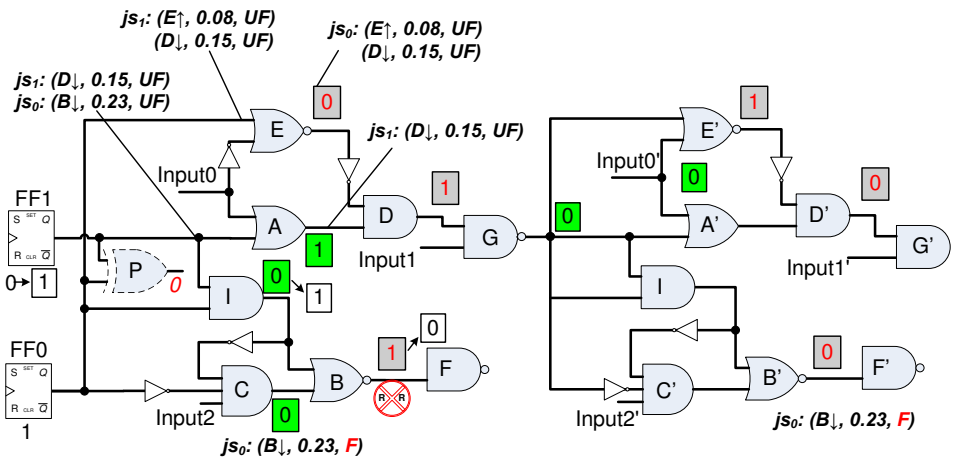


Figure 4.8: Example to show impact of implication order

tracing process, we check its js_0 and js_1 lists and calculate the *weighted sum of the un-failed justifying symbols* on js_0 and js_1, which are defined as $WSUJB_0$ and $WSUJB_1$, respectively. Next, $WSUJB$ is used to store the larger value between $WSUJB_0$ and $WSUJB_1$ on every reached multi-fanout. We then sort the set of reached multi-fanouts in a non-decreasing order based on their $WSUJB$ values. Finally, we make value decision on multi-fanouts and perform logic implication one by one. To be specific, starting from the first gate in sorted multi-fanout list, we assign the corresponding logic value according to $WSUJB$. Then we conduct logic implication process and some transitions may fail, hence we update the states of the *justifying symbols* on each reached multi-fanout or input and then check the multi-fanout list again. The above procedure iterates until all the reached multi-fanouts and inputs have specified values. By doing so, the multi-fanout with higher weighted sum of un-failed justifying symbols is processed with higher priority, and hence we can effectively reduce the total amount of failed transitions.

Multi-Level Backtracking Mechanism

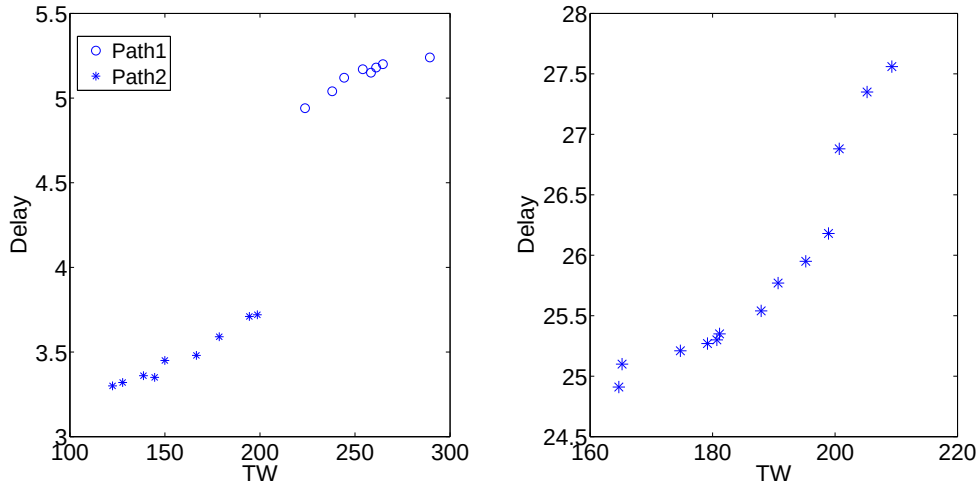
In conventional ATPG process, backtracking is conducted as soon as a logic conflict is detected during implication. While for our problem, we can accept certain amount of temporary logic conflicts that reduce the number of desired transitions and resolve them in later stage. Consequently, we need to design a new backtracking mechanism for maximizing PSN effects, as shown in the following.

During transition justification, we denote the TTW_f/TTW as the failed transition ratio (FTR), where TTW_f and TTW are the total TW of the failed transitions and that of total relevant transitions. Our initial thinking is to set one threshold ratio value TR, and backtracks once the FTR is larger than TR. However, this strategy is not quite effective because the optimal TR values for different set of to-be-justified transitions vary significantly, and hence a universal threshold value is not preferred. In order to overcome this problem, we set a series of threshold $TR = \{tr_0, tr_1, tr_2...\}$ arranged in an increasing order and there is some interval between any neighboring pair. For example, $TR = \{10\%, 20\%, 30\%...\}$. For a given set of transitions to be justified, we try the threshold values one by one in the TR vector, and record the number of backtrackings (denoted as nBT). Clearly, later trials are easier with smaller nBT. In each trial, if nBT is larger than a a pre-defined constant value $maxBT$, we use the next more relaxed threshold. This procedure terminates when either a solution is found with $nBT \leq maxBT$ or all the threshold levels have been tried.

4.5 Experimental Results

4.5.1 Experimental Setup

We implement our layout-aware pseudo-functional SDD pattern generation framework on top of an academic ATPG tool *Atalanta* [93], which originally targets



(a) s38417 (b) des

Figure 4.9: TW-Delay Correlation Plot

stuck-at faults using FAN algorithm [55]. Experiments are conducted on the largest ISCAS'89 and IWLS'05 benchmark circuits that are available to us. We synthesize them using UMC's *130nm* CMOS technology with *1.08V* power supply voltage, and layout them using commercial tools.

It is important to note, while the benchmark circuits used in our experiments are still small when compared to industrial designs, we believe they are sufficient to prove the effectiveness of the proposed technique. *This is because, power supply noises are rather "local" effects and the number of aggressors for a particular critical path is usually not related to the overall circuit size.*

4.5.2 Results and Discussion

As we mentioned in the previous section, our proposed technique does not directly optimize the delay caused by PSN effects since we can only obtain relatively accurate delay value by applying both PSN simulation and timing analysis and it is not

Benchmark	TFC	Path A.R.	Random-fill under F. C.		PSN Max. without F. C.		Our proposed			
	(%)	(%)	TW_r	Runtime(s)	TW_w	Runtime(s)	TW_o	$OR_{TW}(\%)$	$OW_{TW}(\%)$	Runtime(s)
s5378	80.57	77.82	19.15	217.68	29.36	481.39	26.56	38.69	-9.54	591.18
s9234	80.29	81.68	24.31	120.36	38.17	892.1	30.85	26.90	-19.18	982.33
s13207	83.82	83.27	35.98	608.96	61.37	1584.29	48.65	35.21	-20.73	1985.38
s15850	82.28	64.13	28.79	267.32	50.07	983.57	38.58	34.00	-22.95	1289.65
s38417	92.76	79.32	57.8	1025.15	105.25	3215.61	91.83	58.88	-12.75	3606.9
s38584	86.29	72.56	73.45	1329.28	132.89	3504.27	115.68	57.49	-12.95	4138.5
Average	84.33	76.46						41.86	-16.35	

Table 4.1: Comparison among Different SDD Patterns.

affordable to integrate such time-consuming process into our algorithm. Instead of doing so, we employ the *transition weight* metric TW to evaluate the PSN effects caused by certain transition to a sensitized path and then try to maximize the overall effective TW by generating relevant transitions on top of the original SDD pattern. To demonstrate the effectiveness of our method, it is important to observe the correlation between TW and the real circuit delay.

In our first experiment, we randomly select two original SDD test patterns for s38417 and one pattern for des, and then we randomly fill the X-bits in test pattern and calculate the TW sum of the activated relevant transitions for several rounds. To obtain the delay information under PSN effects, we first perform IR-Drop analysis on the layout with the commercial tool to extract the exact voltage on each node of the sensitized path, and then feed this information into static timing analysis tool to obtain the delay for the targeted path. After acquiring the delays on corresponding pathes for these patterns, we plot TW-delay figure as shown in Fig.4.9. It can be observed that they are not perfectly correlated. However, the trend is quite similar and the delay increases as the growth of the activated TW in most cases. Hence, it is with sufficient accuracy to use TW as the optimization target in our algorithm.

Table 5.1 shows our main experimental results, in which we generate pseudo-functional SDD test cubes first using the techniques presented in Section 3, but fill the X-bits in test cubes differently to obtain three kinds of SDD patterns: (1) our proposed pseudo-functional patterns that try to maximize the PSN effects under functional constraints; (2) pseudo-functional patterns with randomly-filled X-bits; (3) test patterns that are generated with maximum PSN effects without considering functional constraints.

Column 2-3 in Table 5.1 present the quality for small delay defect detection. Transition fault coverage (TFC) is shown in Column 2 and it can be observed that at least 80% transition faults can be covered by our pseudo-functional SDD patterns for all the benchmark circuits. Column 3 (i.e., *Path A.R.*) represents the *critical path activation ratio*, which is calculated as follows. We first extract those critical paths which have at most 10% slack from the longest path in the circuit according to static timing analysis results. Next, we remove those false paths that are not sensitizable in functional mode and denote the remaining paths as *sensitizable paths*. We then count those paths that are sensitized by our SDD test pattern, denoted as *sensitized paths*. The *Path A.R.* is the ratio between the sensitized paths and the sensitizable pathes. On average, there are 76% sensitizable paths activated by our SDD patterns. This result shows that our SDD test patterns can effectively sensitize most sensitizable critical paths in the circuit.

Columns 4-11 list the comparison among the three kinds of SDD patterns. Columns TW_r, TW_w and TW_o represent the average activated transition weight for the three different kinds of patterns. OR_{TW} and OW_{TW} are calculated as $OR_{TW} = \frac{TW_o - TW_r}{TW_r} \times 100\%$ and $OW_{TW} = \frac{TW_o - TW_w}{TW_w} \times 100\%$, respectively. Let us first compare patterns generated using the proposed solution against pseudo-functional patterns with randomly-filled X-bits first. It can be observed from Column 3 that our method can achieve up to 59% improvement for benchmark s38417, and for all

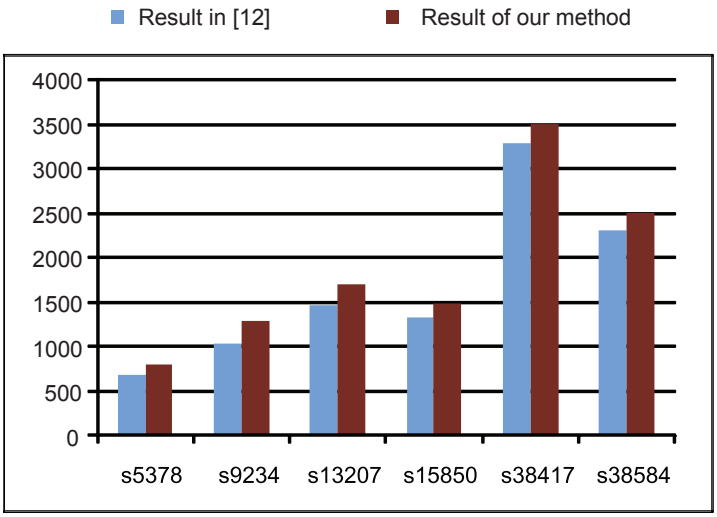


Figure 4.10: Pattern count comparison

the benchmark circuits the average improvement is around 40%. As stated earlier, while pseudo-functional testing inherently minimizes over-testing problem, it may suffer from serious under-testing problem. The above results demonstrate the effectiveness of our proposed algorithm by explicitly taking PSN effects into consideration. When comparing against patterns with maximum PSN effects without considering functional constraints, we can observe more than 16% less PSN effects on average for all benchmark circuits, and scan patterns for benchmark s15850 can result in up to 22% more power supply noises than our patterns that try to maximize PSN effects under functional constraints. This comparison indicates that it is crucial to take functional constraints into consideration when generating SDD test patterns. Otherwise, circuits can be over-tested, leading to significant test yield loss. In terms of computational time, our proposed method is 10-20% longer than that of [54], which is acceptable considering the test quality improvement provided by our solution.

In [54], once a SDD pattern is generated, fault simulation is conducted and

all the faults that are propagated through long paths are dropped. In our method, however, we drop those undetected faults if and only if they are located on the path targeted in relevant transition justification, since we need to guarantee the dropped faults have been affected by sufficient PSN effects. Consequently, our solution generates more test patterns than [54]. We compare the pattern count between the two methods, as shown in the Fig.4.10. It can be observed that, the pattern count increase is moderate and we attribute this to the fact that long paths are difficult to be sensitized if we do not target on them during the test pattern generation process.

4.6 Conclusion

The ever-increasing sensitivity of circuits' timing behavior to PSN effects is a serious challenge for at-speed delay testing. Without considering functional constraints, conventional ATPG may incur either excessive or limited PSN effects on critical paths, leading to over-testing or under-testing of the CUT. In this work, we present a novel pseudo-functional ATPG technique to simultaneously reduce both test overkills and test escapes in SDD testing. Experimental results on large benchmark circuits demonstrate the benefits of the proposed solution.

□ **End of chapter.**

Chapter 5

In-Situ Timing Error Masking in Logic Circuits

5.1 Introduction

With the continuous downscaling of transistor feature size, there is an increasing uncertainty for the timing behavior of today's integrated circuits (ICs). On one hand, embedding a large design guard band to prevent timing errors to occur is not an attractive solution, since this conservative design methodology diminishes the benefit of technology scaling [106]. On the other hand, it is increasingly difficult to rely on off-line delay testing to guarantee circuit timing correctness in functional mode [107]. Consequently, there is a growing research interest to achieve online timing error resilience.

Most of existing solutions for timing error resilience (e.g., the well-known *Razor* technique [31]) try to restore the state of the system to a known-good pre-error state. These techniques are very effective for timing error correction (TEC) in processors with microarchitectural support such as instruction replay, but they are very difficult, if not impossible, to be applied to general logic circuits, due to the

high cost to checkpoint error-free states in such designs.

In-situ timing error correction techniques that are able to mask errors without any rollback, are therefore very attractive. Among the few *in-situ* TEC techniques presented in the literature, most of them [108, 109, 110] rely on time-borrowing technique to correct timing errors, by delaying the arrival time of the correct data to the next logic level. As these techniques reduce the timing slack for the logic level that follows speed-paths, they have difficulty to handle the case when speed-paths exist in consecutive logic levels, limiting the applicability of such solutions. In [111], the authors proposed to synthesize a redundant logic block that is activated only when the speed-paths of the circuit are sensitized, and use it to mask timing errors on targeted paths. While interesting, their proposed synthesis algorithm is time-consuming and the redundant logic block incurs large area overhead.

In this paper, we propose a novel *in-situ* timing error correction technique, namely *InTimeFix*. Similar to [111], we introduce redundant logic into the original circuit to mask timing errors on speed-paths when they are sensitized. Unlike [111] that tries to synthesize the Boolean function that activates speed-paths in a ”brute-force” manner, the redundant TEC circuit in *InTimeFix* is generated based on the concept of *approximation circuit* [112, 113] and can be obtained by simple structural analysis of the original circuit, which is of low cost and is easily scalable to large IC designs. Since the corresponding approximation circuits for speed-paths is with simpler logic structure, a large timing slack is guaranteed for those flip-flops driven by speed-paths (denoted by *suspicious FFs*) and hence is able to mask timing errors occurring on them. The main contributions of this paper include:

- we present a novel technique to add redundant approximation circuit into the original design to create a logically-equivalent yet timing-improved circuit, and prove its correctness;

- we propose a low-cost and scalable technique to synthesize timing error masking logic based on simple structural analysis, without necessarily acquiring the characteristic function for the set of *all* speed-path activation patterns;

From another perspective, *InTimeFix* can be also regarded as a timing optimization technique, since it facilitates to improve circuit timing slack with low hardware cost, as demonstrated in our experimental results. It is important to emphasize that, as a redundancy scheme, *InTimeFix* is compatible with other timing/power optimization techniques such as gate sizing [114] and dual V_{th} allocation [115], and in fact, these techniques can be combined to further improve circuit performance under variation.

The remainder of this chapter is organized as follows. Section 5.2 surveys prior work for online timing error resilience and motivates this work. In Section 5.3 and Section 5.4, we detail the proposed *InTimeFix* technique for *in-situ* correction of timing errors on speed-paths. Experimental results on various benchmark circuits are then presented in Section 5.5. Finally, Section 5.6 concludes this paper.

5.2 Prior Work and Motivation

In order to achieve timing error resilience, we can either predict the error occurrence and take proactive actions to avoid them or detect and correct timing errors (or their effects) when they occur. Generally speaking, timing error prediction techniques (e.g., [116]) are applicable to detect gradual increase of circuit delay resulting from aging effects only. While *Razor*-like techniques are very effective for timing error correction in microprocessors with the help of instruction replay, they are very difficult, if not impossible, to be applied to general logic circuits, due to the high cost to checkpoint error-free states in them. It is therefore imperative

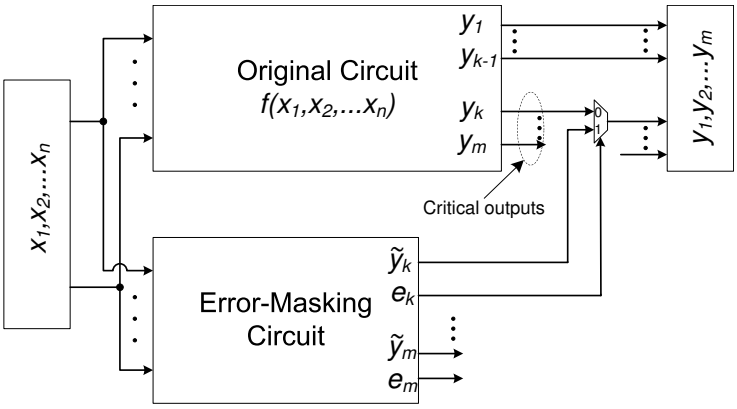


Figure 5.1: Timing Error Masking Scheme in [111].

to develop *in-situ* timing error correction techniques that are able to mask errors without any rollback. There are a few such techniques presented in the literature and they can be classified into two categories: *temporal error masking* and *logic error masking*. Time-borrowing techniques for timing error correction have the inherent weakness of error effect propagation. That is, the timing slack of the successive logic level driven by suspicious FFs is reduced and hence some initially non-suspicious flip-flops in this level may become suspicious ones and need to be replaced by sequential elements with time-borrowing capability again. Due to this timing error propagation effect, the hardware cost for such temporal error masking techniques can be quite high.

In [111], Choudhury and Mohanram proposed to add a redundant logic block to predict the outputs of the circuit upon application of inputs that sensitize speed-paths. With this exact sensitization constraint, the error-masking circuit tends to have more timing slack when compared to the original circuit, and hence is immune to timing errors. As shown in Fig. 5.1, targeting those timing-critical outputs $y_k, \ldots, y_m$, error masking circuit generates two outputs for each of them, e.g., $\tilde{y}_k$ and e_k for y_k with potential timing error. To be specific, when a speed-path driving y_k is sensitized, e_k is set correspondingly and the original circuit's output y_k is sub-

stituted with the fast predicted value $\tilde{y}_k$ to achieve timing error resilience. While the idea is interesting, construction of the proposed redundant logic block incurs quite large area/power overhead, as demonstrated in their experimental results. In addition, to synthesize such error-masking circuits is quite complex, requiring to obtain the characteristic function for the set of *all* speed-path activation patterns, which is only practical for small circuit blocks.

To sum up, process, voltage, and temperature (PVT) variations have an ever-increasing adverse impact on the timing behavior of integrated circuits with technology scaling. While there are a few techniques shown in the literature, they either rely on certain assumptions about the circuit structure and hence limit their applicability, or suffer from scalability issues and cannot be easily applied in large IC designs. This motivates the proposed *InTimeFix* technique to achieve low-cost and scalable timing error resilience in logic circuits.

5.3 In-Situ Timing Error Masking with Approximate Logic

The concept of *approximation circuit* was proposed in [112], which tries to increase a microprocessor's clock frequency by replacing a complete logic function with a simplified circuit that mimics the function and uses rough calculations to speculate and predict results. In [113], the authors defined *approximate logic* in digital circuit as: Given two Boolean functions F and G, $G0$ is a *0-approximate logic* of F if $G0 = 0 \Rightarrow F = 0$. Similarly, $G1$ is a *1-approximate logic* of F if $G1 = 1 \Rightarrow F = 1$. Consider a Boolean function $F = a + b + \bar{a}cd$, there are 13 out of total 16 minterms in its truth table that output logic 1 while the other 3 minterms output logic 0. A 1-approximate logic function of F is $G1 = a + b$. This approximate function covers 12 out of 13 minterms for $F = 1$ minterms and can

be implemented with one logic gate. Similarly, a 0-approximate logic function of F is $G0 = a + b + c$, which covers 2 out of 3 minterms for $F = 0$.

Generally speaking, since the approximate logic is with much simpler logic structure when compared to the original circuit, the computation latency is smaller. The basic idea of the proposed *InTimeFix* technique is to generate approximate logic for the original logic function of suspicious FFs in such manner that it covers all the logic minterms that sensitize speed-paths.

5.3.1 Equivalent Circuit Construction with Approximate Logic

Given a logic circuit that implements Boolean function F, suppose $G0$ is a 0-approximate logic for F and $G1$ is a 1-approximate logic for F and we denote by P, $P0$ and $P1$ all the minterms in F's truth table, the minterms covered by $G0$ and the minterms covered by $G1$, respectively. Now, let us construct a circuit $F' = F \cdot G0 + G1$ as shown in Fig. 5.2. We define $d_{G0}(P0)$, $d_{G1}(P0)$ and $d_F(P0)$ as the worst-case delay among all minterms in P0 through circuit $G0$, $G1$ and F, respectively. Similarly, $d_{G0}(P1)$, $d_{G1}(P1)$, $d_F(P1)$ and $d_F(P - P0 - P1)$ are defined. d_{AB} is the total propagation delay of *AND* gate A and *OR* gate B. Assuming approximate logic $G0$ and $G1$ are implemented with simpler logic structures when compared to original circuit F and hence has less computation latency (e.g., $d_{G0}(P0)$¡$d_F(P0)$, $d_{G1}(P0)$¡$d_F(P0)$ and $d_{G1}(P1)$¡$d_F(P1)$), we have the following theorem:

Theorem 2 *The circuit shown in Fig. 5.2, $F' = F \cdot G0 + G1$, is logically-equivalent to the original circuit F, and its worst-case timing delay is* $\max\{d_F(P - P0 - P1), d_{G0}(P0), d_{G1}(P0), d_{G1}(P1)\} + d_{AB}$.

proof 2 *When the original circuit F outputs 1, by applying counter-positive law,*

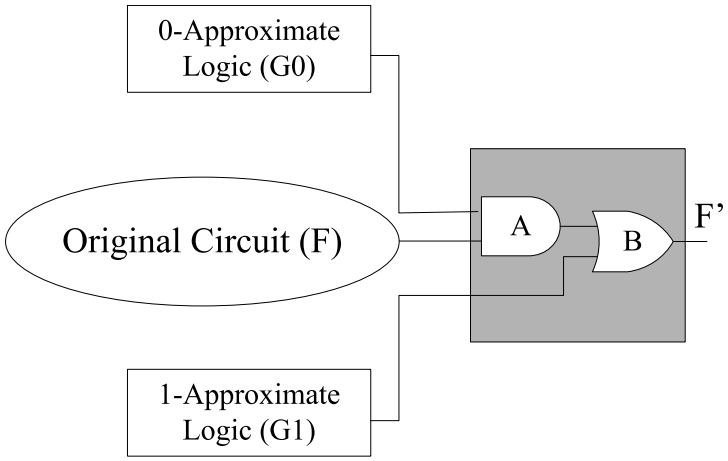


Figure 5.2: Equivalent Circuit with Approximate Logic.

its 0-approximate logic must also output 1 (i.e., $G0 = 1$), and hence $F' = 1$. Similarly, when the original circuit F outputs 0, by applying counter-positive law, its 1-approximate logic must also output 0 (e.g., $G1 = 0$), and hence $F' = 0$. Consequently, F and F' are logically-equivalent.

To obtain the worst-case delay for this equivalent circuit F', let us consider the circuit delay before the shaded logic block in Fig. 5.2 for the following three cases, corresponding to application of inputs belonging to different set of minterms of the truth table of F/F'.

- *When the inputs applied to the circuit belong to $P0$, $G0$ outputs controlling value 0 for* AND *gate A after $d_{G0}(P0)$ and dominates the path through the original circuit F with longer delays. The worst-case delay would be* $\max\{d_{G0}(P0), d_{G1}(P0)\}$. *Note that $d_{G1}(P0)$ is the time spent to settle $G1$ to be non-controlling value 0 for* OR *gate B.*

- *When the inputs applied to the circuit belong to $P1$, $G1$ outputs controlling value 1 for* OR *gate B after $d_{G1}(P1)$ and it dominates the path through the original circuit F and $G0$. The worst-case delay in this case is therefore simply $d_{G1}(P1)$.*

- *When the inputs applied to the circuit belong to* $P - P0 - P1$*, we have to wait for the original circuit F to settle down, and hence the worst-case delay would be* $d_F(P - P0 - P1)$.

The worst-case timing delay for circuit F' is therefore $\max\{d_F(P - P0 - P1)$, $d_{G0}(P0)$, $d_{G1}(P0)$, $d_{G1}(P1)\} + d_{AB}$*, after considering the time spent on gates A and B.*

From manufacturing test perspective, adding redundancy into the circuit may introduce untestable faults, and there is no exception for the above design. For example, the stuck-at-1 fault at the output of $G0$ is untestable, because to activate this fault, we need to set it as logic '0', but when $G0 = 0$, the original circuit F will also output logic '0', preventing the propagation of its faulty effect to outputs. Similarly, the stuck-at-0 fault at the output of $G1$ is untestable. It is important to note that, the presence of such faults would not affect the functional correctness of the circuit. At the same time, they do render the corresponding timing error masking logic to be ineffective, but the likelihood for such faults to exist is quite low due to their small sizes.

One way to make these faults testable is to add some design-for-testability (DfT) circuit, e.g., adding additional flip-flop to be driven by G0/G1 directly. This, however, increases the hardware cost and also prolongs the propagation delay on approximate logic paths due to extra load. Alternatively, we can rely on delay testing to guarantee the timing correctness of the corresponding speed-paths. In other words, as long as the path can pass at-speed test, its timing correctness is guaranteed and we do not need to care whether these untestable faults exit or not.

5.3.2 Timing Error Masking with Approximate Logic

Generally speaking, a suspicious FF is driven by multiple paths, and timing errors may occur only when speed-paths are sensitized. In other words, timing errors

may be activated by only a few minterms of the truth table for a suspicious FF, denoted as *critical minterms*. Motivated by this observation and the performance upper bound theorem shown earlier, if all the critical minterms are covered with approximate logic, we can achieve large timing slack and mask potential timing errors. *The question now becomes how to efficiently construct redundant approximate logic for speed-paths?*

Consider an example circuit shown in Fig. 5.3, wherein path P $\{Input1, A, D, H, F, G, I, J\}$ is a speed-path. When logic '1' is applied at $Input1$ and propagates along this path to generate logic '0' at the receiving end, we have to assign logic '1' at the side-input[1] of gate A. This is because, logic '1' is a non-controlling value of *AND* gate, and the output of A will be dominated by the side-input if it is assigned with controlling value. Similarly, side-inputs of gate F and gate I have to be assigned as non-controlling value (see Fig. 5.3).

Let us define the side-inputs on the path that need to have deterministic non-controlling values (marked in shade) as *essential side-inputs*. Based on the above discussion, we have the following lemma:

Lemma 2 *To cover all the critical minterms that sensitize a particular speed-path is equivalent to approximate its essential side-inputs.*

With the above, we can construct redundant approximate logic for each speed-path by simple structural analysis. Again, take path P in Fig. 5.3 as an example. Suppose we would like to construct 0-approximate logic for this path, we first duplicate the entire path and then gradually remove those gates without essential side-inputs.

To be specific, the removing process is conducted structurally by analyzing the targeted speed-path P reversely from the ending gate (i.e., gate G) to the sending

[1]Given a path P = $\{G_0, G_1, ...G_m\}$, for a specific gate G_i, G_{i-1} is the *on-input* signal of G_i, while other input signals of G_i are *side-inputs*.

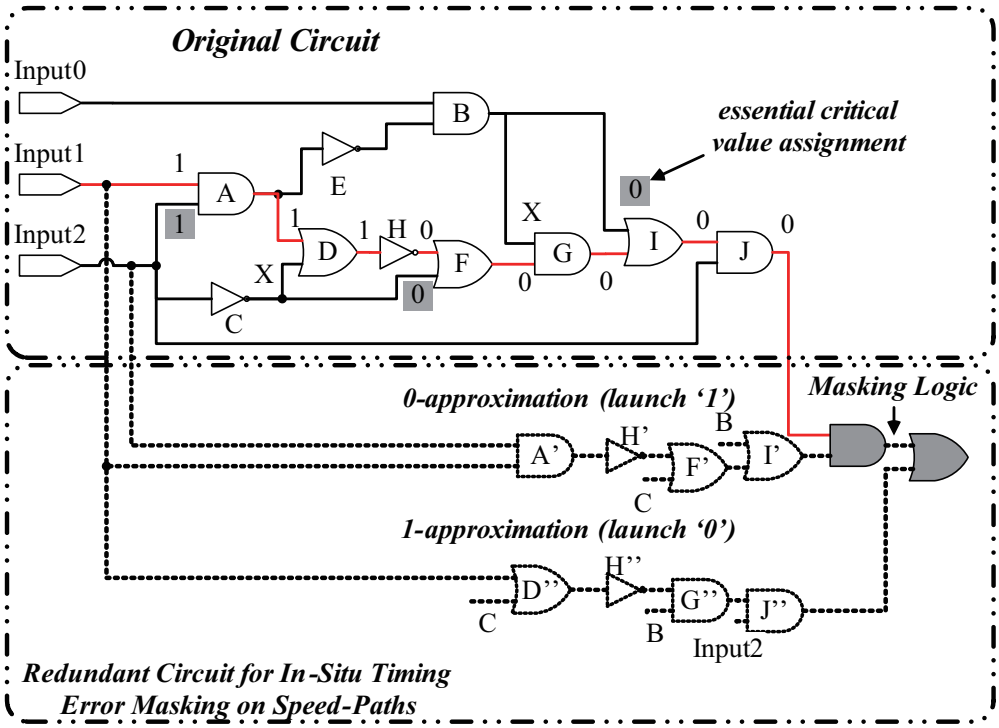


Figure 5.3: Speed-Path Approximation.

gate (i.e., gate A). Consider gate J, since it is dominated by its on-input with controlling value 0, we can remove it from the approximate logic. Similarly, gate G and gate D are not needed. While the outputs of gates F and A are determined by both on-input and side-input signals, two gates need to be duplicated in the 0-approximate logic, and the side-inputs are connected to the same net as path P in the original circuit. As shown in Fig. 5.3, the 0-approximate logic constructed as above will output logic '0' if and only if the speed-path P in the original circuit is sensitized with launching value logic '1'. The 1-approximate logic for speed-path P can be constructed similarly (see Fig. 5.3). Since the approximate logic is with much simpler logic structure and the delay of the masking logic is usually insignificant (without necessarily sizing it up), we can achieve large timing slack and mask potential timing errors on speed-paths.

Note that, not all kinds of logic cells have controlling values for their inputs,

e.g., XOR/XNOR gate. If a speed-path contains such kind of logic cells, their side-inputs will be treated as essential side-inputs to have deterministic values to approximate and the proposed methodology is applicable to such designs.

5.4 Cost-Efficient Synthesis for InTimeFix

Adding redundant approximate logic facilitates to achieve timing error resilience on speed-paths. As the construction of the approximate logic needs to take side-input signals from the original circuit, however, two potential problems arise: (i) the latencies for side-inputs may become a concern for the propagation delay of the approximate logic and such side-inputs are denoted as *critical side-inputs* (formally defined later); (ii) the increased loading capacitance on side-inputs can prolong the delay of those paths going through them. The above observation motivate us to propose a cost-efficient and scalable synthesis framework to resolve these issues, as illustrated in Fig. 5.4.

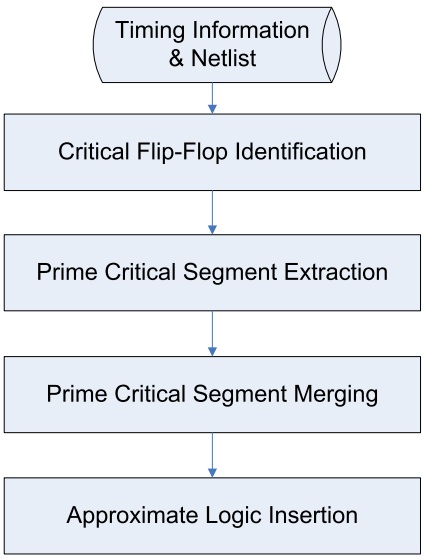


Figure 5.4: *InTimeFix* Synthesis Framework.

5.4.1 Overall Flow

Fig. 5.4 describes the synthesis overflow for InTimeFix. With the optimized circuit netlist and the corresponding timing information in standard delay format (SDF), we firstly identify those suspicious FFs that are driven by speed-paths and thus need to be considered. Speed-paths are defined as those paths whose propagation delays exceed a threshold value, e.g., 80% of the maximum path delay. Note that, although static timing analysis is not able to output accurate timing values, its accuracy is sufficient for suspicious FF identification because we only need a comparative relationship among paths and we can always tune the threshold to tradeoff between the hardware cost and protection strength.

Next, a set of so-called *prime critical segments* is extracted from speed-paths, defined as the segments of speed-paths that do not include any critical side-inputs, with which approximate logic can be safely generated with more timing slack. Then, a heuristic method is used to merge prime critical segments to minimize the hardware cost of the approximate logic and the extra loading capacitance of side-inputs. Finally, approximate logic is generated for the merged prime critical segments and inserted into the original circuit as redundant resources, as shown in Section 3.2. In the following, we present the algorithms for prime critical segment extraction and prime critical segment merging in detail.

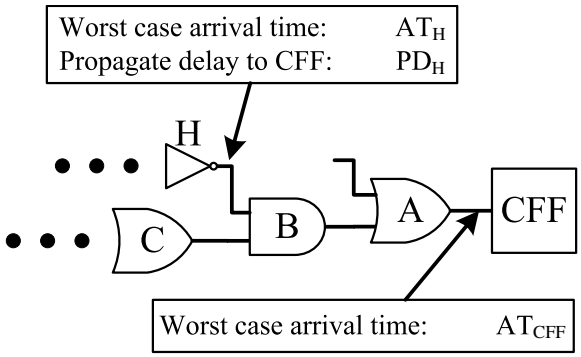


Figure 5.5: Example to Illustrate the Critical Side-Input.

5.4.2 Prime Critical Segment Extraction

Before introducing the details of our proposed algorithm, let us first formally define the criteria used to classify critical side-input, as depicted in Fig. 5.5. Taking side-input H as an example, when considering the critical flip-flop CFF, we have the worst case arrival time AT_H on side-input H, the propagation delay PD_H from H to CFF, and the worst case arrival time AT_{CFF} on CFF. Suppose RD is a pre-defined reduced delay (i.e., extra slack) that we want to achieve and MD is the delay of the masking logic, then H is a critical side-input if $AT_{CFF} - (AT_H + PD_H + MD) > RD$. The basic idea behind this definition is that, if this criteria is not satisfied, there is sufficient timing slack on this side-input and it does not affect the timing of the approximate logic at all. Based on the above definition, we denote a gate on speed-path to be prime critical gate if all its side-inputs are non-critical. Furthermore, a segment of a speed-path is a prime critical segment if it only consists of prime critical gates.

Our proposed prime critical segment extraction algorithm (denoted as *ExPriSeg*) is shown in Algorithm 1, where, *Gate* is one gate on the speed-path and *Segment* is the parameter to store the targeted segment; *SegmentSet* denotes the set of extracted prime critical segments; while *Ogate* and *Cside* represent the on-input and critical side-input of *Gate*, respectively.

To relax the timing slack to a critical flip-flop by RD, we need to reduce the delay of all the speed-paths connected to it. Here we regard an extracted prime critical segment as a *legal* one if the approximate logic for it is able to reduce the delay of the targeted speed-path for at least RD. Starting from a specific critical flip-flop, our algorithm extracts the prime critical segments by recursively tracing its fan-in cone in a depth-first manner. Initially, *Segment* is empty and *Gate* is the input gate of the targeted critical flip-flop. During the tracing procedure, once a primary input or a flip-flop, denoted as *PI* or *FF*, is reached (Line 2), function

Algorithm 1: Extract Prime Critical Segment(ExPriSeg)

```
   input: Gate,Segment
1  begin
2      if PI or FF is reached then
3          if Segment is not illegal then
4              return FailToExtract;
5          else
6              add Segment into SegmentSet;
7              return SucceedToExtract;
8      if Gate is prime critical gate then
9          add Gate into Segment;
10         find the on-input Ogate;
11         return ExPriSeg(Ogate,Segment);
12     else
13         if Segment is legal then
14             add Segment into SegmentSet;
15             return SucceedToExtract;
16         else
17             foreach Cside do
18                 new Segment;
19                 if ExPriSeg(Cside,Segment) ==FailToExtract then
20                     return FailToExtract;
21             return SucceedToExtract;
22 end
```

returns *FailToExtract* if there is no legal prime critical segment found, otherwise we keep the *Segment* and return *SucceedToExtract*. Suppose *Gate* is detected to be a prime critical gate, we add *Gate* into *Segment* and keep on tracing its on-input gate. On the other hand, if *Gate* is not a prime critical gate (Line 12), we first check whether the current *Segment* is legal or not. The searching process is stopped by storing *Segment* and return *SucceedToExtract* if *Segment* is legal prime critical segment, otherwise, we empty the current *Segment* and start to trace each critical side-input separately. Clearly, the extracted prime critical segments can

cover all the speed-paths ending at the targeted flip-flop if the final returned value is *SucceedToExtract*. Suppose it returns *FailToextract*, we will try to reduce the value of *RD* by a pre-defined ratio, and the same search is conducted again. One thing to note is that our method tends to extract legal prime critical segments that are close to the targeted flip-flops, and they are able to cover more speed-paths, if any.

5.4.3 Prime Critical Segment Merging

The prime critical segments extracted from different critical flip-flops are likely to be merged to further reduce hardware cost. As the example shown in Fig. 5.6, two prime critical segments $\{E,D,...C,A\}$ and $\{E,D,...C,B\}$ can share a merged prime critical segment $\{E,D,...C\}$. Furthermore, we notice that our extracted prime critical segments are not in the most compact format. Taking segment $\{E,D,...C,A\}$ as an example, it is not necessary to approximate the entire segment, instead, only by approximating $\{E,D,...C\}$ is enough to achieve *RD* delay reduction. We denote the length (i.e., the number of logic gates) of the shortest legal subpart of prime critical segment as the *essential length*, and represent the length of the remaining part as the *redundant length*. For the sake of simplicity, we regard a subpart of a prime critical segment still legal if the length of the subpart is larger than the essential length. Therefore, two prime critical segments can be merged if the length of the shared part is larger than both of their essential lengths.

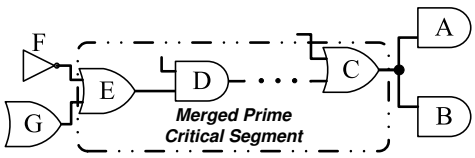


Figure 5.6: Prime Critical Segment Merging: An Example.

The main flow of our proposed prime critical segment merging algorithm is

shown in Fig. 5.7, comprising the following steps:

1. Starting from the extracted prime critical segment set, we first sort them in non-decreasing order in terms of their redundant length. The basic idea behind this step is that we need to first fix those prime critical segment with less flexibility. Then, all the segments are merged by iteratively applying the following steps.

2. We always select the top un-processed segment and employ a heuristic method to find the optimal subpart. First of all, the closest-to-output gate on the selected segment shared by most remaining un-processed segments is identified. Taking the circuit shown in Fig. 5.6 as example, gate C is picked first if the selected segment is $\{E,D,C,A\}$ since it has higher probability to replace the most remaining un-processed segments by backwardly tracing the selected segment from this gate, say, segment $\{E,D,C\}$. Suppose that the length of $\{E,D,C\}$ is less than the essential length, we extend this sub-segment to $\{E,D,C,A\}$ and such extension is conducted until the length requirement is satisfied. Suppose the length of $\{E,D,C\}$ is larger than the essential length, we enumerate all the possible legal subparts to identify the one that includes minimal number of side-inputs in the hope that increased loading capacitance is minimized when inserting approximate logic. Finally, the optimal segment is fixed.

3. The remaining un-processed segments are checked to determine whether they can be replaced by newly fixed segment. The replaceable segments are labeled as processed.

4. The procedure terminates if all the segments have been processed, otherwise it goes back to step 2.

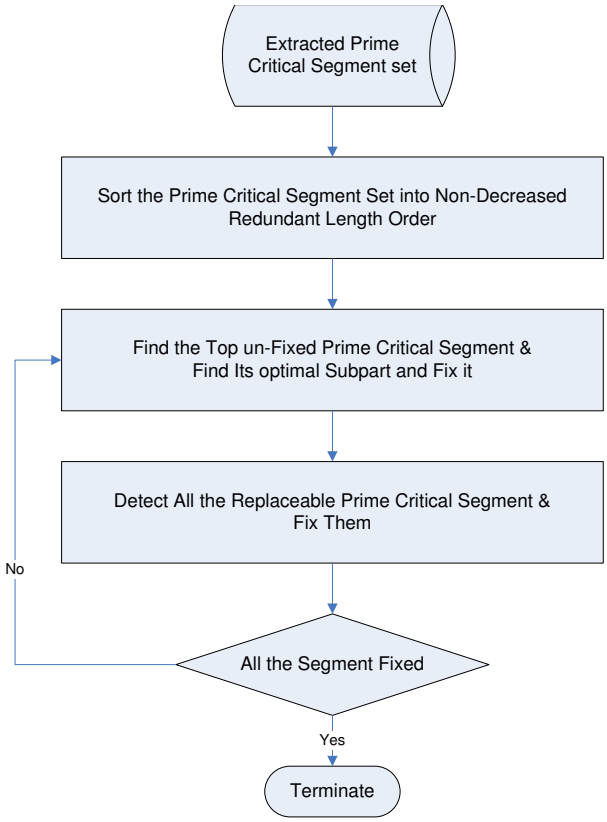


Figure 5.7: Flowchart of Prime Critical Segment Merging.

Benchmark	Circuit Size (# of gates)	FF (#)	Cri. FF (#)	Cost (# of gates)	Inc. Ratio (%)	Ori. WCD (ns)	Our WCD (ns)	Relaxed Slack (ns)	Imp. Ratio (%)	Runtime (s)
s38417	24370	1636	78	570	2.34	35.34	31.93	3.41	9.64	0.09
s38584	21066	1426	12	98	0.47	20.50	18.49	2.01	9.80	0.1
des_perf	154323	9105	89	592	0.38	7.80	6.68	1.12	14.36	0.95
wb_conmax	75352	3316	277	1160	1.54	8.47	7.74	0.73	8.59	0.4
ethernet	157841	10752	28	386	0.24	8.21	7.11	1.10	13.38	1.083
Ave.					0.89				11.15	

Table 5.1: Experimental Results on Improved Timing Slack and Hardware Cost.

5.5 Experimental Results

5.5.1 Experimental Setup

To evaluate the effectiveness of our proposed *InTimeFix* technique, we conduct experiments on two large ISCAS'89 benchmark circuits, *s38417* and *s38584*, as

Benchmark	Gate Sizing with Area Constraint vs. *InTimeFix*				*InTimeFix* on top of Gate Sizing				
	Area Constraint (# of gates)	Gate Sizing WCD (ns)	*InTimeFix* WCD (ns)	Improvement (%)	Gate Sizing WCD (ns)	Area Cost (# of gates)	*InTimeFix* WCD (ns)	Area Cost (# of gates)	Further Improvement (%)
s38417	570	34.70	31.93	7.99	31.93	1050	27.90	508	12.63
s38584	98	19.96	18.49	7.36	15.18	1217	13.85	71	8.81
des_perf	592	6.95	6.68	3.88	6.43	5937	6.06	1309	5.85
wb_conmax	1160	7.47	7.74	-3.61	5.38	4008	4.88	561	9.42
ethernet	386	7.23	7.11	1.73	6.15	8257	5.25	314	14.61
Ave.				3.47					10.26

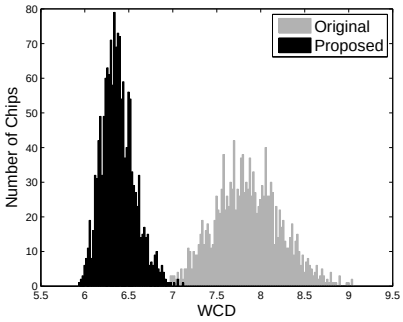
Table 5.2: Comparison on Timing Slack Improvement: Gate Sizing vs. *InTimeFix*.

well as three large IWLS benchmark circuits, *wb_conmax*, *ethernet* and *des_perf*, which are the largest benchmark circuits available to the public domain.

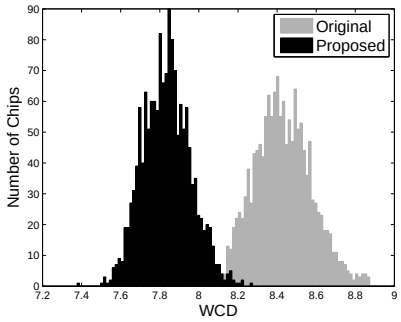
In the experimental flow, we first synthesize the benchmark circuits with a commercial tool to obtain the optimized circuit netlist and its SDF timing information, under 0.13μm CMOS technology. They are then fed to *InTimeFix* to generate redundant approximate logic to mask potential timing errors, targeting speed-paths within 20% of the longest path delay. Finally, commercial timing analyzer is applied to evaluate the solution, and the quality of the solution is demonstrated by the extra timing slack achieved with the new circuit when compared with the original one.

5.5.2 Results and Discussion

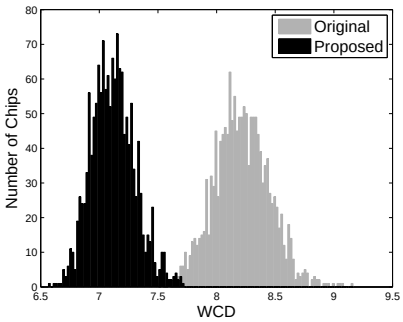
Table 5.1 present our experimental results. When comparing the worst case delay (WCD) between the original circuit and the one equipped with redundant approximate logic (Columns 6-9), it can be observed that the proposed solution is able to achieve 11.15% timing slack relaxation on average. On the other hand, as shown in Column 3, the hardware cost (the unit of cost is the area of smallest 2 input *AND* gates) introduced in the proposed *InTimeFix* technique to achieve the above timing



(a) des_perf



(b) wb_conmax



(c) ethernet

Figure 5.8: Circuit Timing under Process Variation

slack is extremely low, less than 0.89% on average. As can be seen from Column 10, the runtime to process the largest benchmark circuit *ethernet* takes less than one second. Consequently, we believe the proposed methodology can be easily scalable to large industrial designs.

A close examination of the experimental results show that benchmark circuits *s*38417 and *wb_conmax* consume the largest percentage of hardware overhead. The reason is that the speed-paths in these two benchmarks are quite evenly distributed and the prime critical segments identified from different critical flip-flops are more likely to be independent to each other. Therefore, the hardware cost is higher than other benchmark circuits. To have a better design tradeoff, we try to reduce the

hardware cost for these two benchmarks by gradually removing the approximation logic on the shortest speed-paths. Even when the hardware cost is reduced to less than 1%, we can still achieve 7.1% and 11.6% slack relaxation for *wb_conmax* and *s*38417, respectively.

With the above experimental results, we can observe clear advantages of the proposed *InTimeFix* technique over [111], without performing direct comparisons. In [111], the authors conducted experiments on a number of benchmark circuits whose sizes are less than 2000 gates. Their experimental results show that, on average 18% area overhead is required to mask timing errors on speed-paths within 10% of the longest path delay (the minimum overhead is 4%).

While not targeting timing error resilience, gate sizing is an effective technique to improve circuit timing. In our next experiment, we compare our *InTimeFix* solution against a greedy gate sizing technique [114]. Firstly, when we constrain the area overhead for gate sizing solution to be the same as the one with *InTimeFix* in earlier experiment, it can be seen that *InTimeFix* outperforms gate sizing in most cases (except *wb_conmax*), and the average improvement is 3.47%, which proves the cost-efficiency of the proposed solution. Next, since *InTimeFix* is compatible with gate sizing technique, we combine the two solutions in such manner that we first employ gate sizing to improve circuit timing until no further benefits can be achieved and then apply *InTimeFix* on top of it. We can observe that, without area constraints, gate sizing can significantly improve circuit performance, but at considerable area and power cost. *InTimeFix* is able to provide additional 10.26% timing slack on average, and the area cost is still quite small.

Finally, we evaluate the impact of process variation on the proposed InTimeFix architecture, using Monte Carlo simulation. According to [119], we assume there is 10% variation on each standard cell. The results are depicted in Fig. 5.8, where we plot and compare the WCD distributions of two sets of circuits (i.e. the black

pile represents set of processed circuits and the gray one denotes the original set of circuits) for the three large IWLS circuits. As can be seen from the figure, even for circuit with InTimeFix approximate logic under the worst case process variation corner, it has smaller or similar WCD when comparing with that of the original circuit under the best case scenario. Moreover, it can be observed that the number of the closer-to-mean chips increases and the standard deviation of WCD distribution shrinks with InTimeFix. In particular, as can be observed in Fig. 5.8(a), benchmark circuit *des_perf* with InTimeFix has only one third of the distribution width when compared to that of the original circuit.

The above phenomenon demonstrates that the proposed InTimeFix technique facilitates to tolerate process variation effects. The reason is that our proposed method effectively reduces the number of logic elements on the critical paths and also shrinks the variation, behind which the mathematic principle can be explained by the example given in [120]: assuming inverters have independent gaussian delay distribution (μ, σ), the delay of a path including n inverters obey the gaussian distribution $(n\mu, \sqrt{n}\sigma)$. Clearly, less n leads to smaller deviation.

5.6 Conclusion

In this paper, we propose a novel *in-situ* timing error correction technique, namely *InTimeFix*, by introducing redundant *approximate logic* with more timing slack for speed-paths in the circuit. The proposed synthesis methodology for the redundant circuit only relies on simple structural analysis of the original circuit, and hence it can be easily scalable to large IC designs. Experimental results demonstrate that the proposed solution can effectively increase circuit timing slack with very low cost.

□ **End of chapter.**

Chapter 6

Conclusion and Future Work

Timing uncertainty issue caused by aggressive technology scaling has significantly threatened ICs' reliability, which creates several challenges on building a reliable system with unreliable devices.

To address the above issues, we develop efficient and effective FPI techniques to identify those timing-independent false paths that cannot be sensitized under any signal arrival time condition in integrated circuits, which facilitate us to find much more false paths than conventional methods. The identified false paths not only helps us to generate better timing result and improves the circuit performance, but also alleviates the burden of timing optimization algorithms and saves the unnecessary cost used to reduce the false paths delay. Then, we present a novel pseudo-functional ATPG technique to simultaneously reduce both test overkills and test escapes in SDD testing. The proposed method enhance test quality by capturing accurate worst-case delay. Finally, we propose a novel in-situ timing error correction technique, by introducing redundant approximate logic with more timing slack for speed-paths in the circuit. The proposed synthesis methodology for the redundant circuit only relies on simple structural analysis of the original circuit, and hence it can be easily scalable to large IC designs.

There are several important topics yet to explore for future work. To continue the technology scaling, we have to accept some infrequent timing errors in circuit's the usage phase in the near future. As mentioned in this thesis, several works have been done to tolerate timing error. In order to obtain better power and performance tradeoff, it is possible to construct cross layer timing error resilience mechanism, which is able to tolerate errors at different abstract layer with different cost. To achieve this, several fundamental issues need be solved: (i) how to evaluate error propagation impact at different layers? (ii) how to design the error information interface to pass the error status from circuit layer to high layer; (iii) how to estimate recover or masking cost for different layers? (iv) how to verify the correctness and the error coverage of proposed error tolerant mechanism? Secondly, motivated by the factor that error recover or masking cost is related to timing error rate, it is essentially to take into consideration the error rate information when designing a error tolerant system, so that the power consumption and performance benefits can be enhanced. Moreover, since the timing error rate is dependent on input vector sequence, which can only be extracted at online stage. Therefore, we can explore the online tuning techniques such as voltage frequency scaling to achieve application-aware timing error tolerance. Finally, the traditional at-speed delay test is performed in a deterministic manner, yet the timing error tolerant circuit is inherently timing error resilience. Therefore, how to conduct delay testing on such a circuit is questionable.

□ **End of chapter.**

Bibliography

[1] M. Bushnell and V. Agrawal. Essentials of Electronic Testing. Kluwer Academic Publishers, 2000.

[2] M. Abramovici, M. Breuer, and A. Friedman. Digital Systems Testing and Testable Desige. IEEE Press, 1990.

[3] G. D. Micheli. Synchronous logic synthesis: algorithms for cycle-time minimization. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 63–73, Jan 1991.

[4] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 492–498, 1992.

[5] A. B. Kahng, S. Mantik and I. L. Markov. Min-max placement for large-scale timing optimization. In *Proceedings international symposium on Physical design (ISPD)*, pages 143–148, 2002.

[6] M. Cho, D. Z. Pan, H. Xiang and R. Puri. Wire density driven global routing for CMP variation and timing. In *Proceedings IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 487–492, 2006.

[7] L.T. Pillage. Asymptotic waveform evaluation for timing analysis. In *Proceedings IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 352–366, 1990.

[8] K. Ravindran, K. Kalafala, S.G. Walker, S. Narayan, D.K. Beece, J. Piaget, N. Venkateswaran and J.G. Hemmett. First-Order Incremental Block-

Based Statistical Timing Analysis. In *Proceedings IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 2170–2180, October 2006.

[9] D. H. Du, S. H. Yen, S. Ghanta. On the general false path problem in timing analysis. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 555–560, October 1989.

[10] J. J. Liou, A. Krstic, L. C. Wang, K.-T. Cheng. False-path-aware statistical timing analysis and efficient path selection for delay testing and timing validation. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 566–569, 2002.

[11] T. Sato, Y. Kunitake. A simple flip-flop circuit for typical case design for DFM. In *Proceedings Intl Symposium on Quality Electronic Design*, pages 539–544, 2007.

[12] C. Metra, M. Favalli and B. Ricco. Online detection of logic errors due to crosstalk, delay and transient faults. In *Proceedings Intl.Test Conference (ITC)*, pages 524–533, 1998.

[13] J. A. Darringer, D. Brand, J. V. Gerbi, W. H. Joyner and L. Trevillyan. LSS: A system for production logic synthesis. In *IBM Journal of Research and Development*, pages 537–545, 1984.

[14] G. D. Micheli. Synchronous logic synthesis: algorithms for cycle-time minimization. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 63–73, 1991.

[15] K. Bartlett, W. Cohen, A. D. Geus and G. Hachtel. Synthesis and Optimization of Multilevel Logic under Timing Constraints. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 582–596, October 1986.

[16] K. Chaudhary and M. Pedram. A near optimal algorithm for technology mapping minimizing area under delay constraints. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 492–498, 1992.

[17] Y. Kukimoto, R. K. Brayton and P. Sawkar. Delay-optimal technology mapping by DAG covering. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 348–351, 1998.

[18] W. P. Lee, H. Y. Liu and Y. W. Chang. Voltage Island Aware Floorplanning for Power and Timing Optimization. In *Proceedings of IEEE/ACM International Conference on Computer-Design Automation*, pages 389–394, 2006.

[19] M. Marek-Sadowska and S. P. Lin. Timing driven placement. In *Proceedings of IEEE/ACM International Conference on Computer-Design Automation*, pages 94–97, 1989.

[20] W. Swartz and C. Sechen. Timing Driven Placement for Large Standard Cell Circuits. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 211–215, 1995.

[21] J. Hu and S. S. Sapatnekar. A timing-constrained algorithm for simultaneous global routing of multiple nets. In *Proceedings of ACM/IEEE International Conference on Computer-Design Automation*, pages 99–103, 2000.

[22] H. P. Tseng, L. Scheffer and C. Sechen. Timing- and crosstalk-driven area routing. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 528–544, Apr., 2001.

[23] C. f. Chieh, Y. C. Hsu and F. S. Tsai. Timing optimization on routed designs with incremental placement and routing characterization. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 188–196, Feb., 2000.

[24] A. Krstic, Y.-M. Jiang and K.-T. Cheng. Delay testing for non-robust untestable circuits. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 416–425, Mar., 2001.

[25] V. Iyengar, T. Yokota, K. Yamada, T. Anemikos, B. Bassett, M. Degregorio, R. Farmer, G. Grise, M. Johnson, D. Milton, M. Taylor and F. Woytowich. At-Speed Structural Test For High-Performance ASICs. In *Proceedings of International Test Conference (ITC)*, pages 1–10, 2006.

[26] M. Favalli and C. Metra. Sensing circuit for on-line detection of delay faults. In *IEEE Trans.VLSI Systems*, pages 130–133, 1996.

[27] Y. Tsiatouhas, S. Matakias, A. Arapoyanni and T. Haniotakis. A sense amplifier based circuit for concurrent detection of soft and timing errors in CMOS ICs. In *Proceedings Intl.On-line Testing Symposium*, pages 12–16, 2003.

[28] Intel's 45nm CMOS Technology. In *Intel Technology Journal*, Vol. 12, June, 2008.

[29] Mark Bohr. The New Era of Scaling in an SoC World. In *Proceedings IEEE International Solid-State Circuits Conference*, pages 23–28, 2009.

[30] M. Nicolaidis. Time redundancy based soft error tolerance to rescue nanometer technologies. In *Proceedings VLSI Test Symposium*, pages 86–94, 1999.

[31] D. Ernst *et al*. Razor: a low-power pipeline based on circuit level timing speculation. In *Proceedings Intl. Symposium on Microarchitechture*, pages 7–18, 2003.

[32] G. Brian *et al*. Blueshift: Designing processors for timing speculation from the ground up. In *Proceedings High Performance Computer Architecture*, pages 213–224, 2009.

[33] L. Wan *et al*. DynaTune: Circuit-level optimization for timing speculation considering dynamic path behavior. In *Proceedings Intl. Conference Computer-aided Design (ICCAD)*, pages 172–179, 2009.

[34] 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 *Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pages 1078–1083, 2004.

[35] A. Krstic, Y.-M. Jiang, and K.-T. Cheng. Pattern Generation for Delay Testing and Dynamic Timing Analysis Considering Power-Supply Noise Effects. In *IEEE Transactions on Computer-Aided Design*, pages 416–425, March 2001.

[36] J.-J. Liou, A. Krstic, Y.-M. Jiang, and K.-T. Cheng. Path Selection and Pattern Generation for Dynamic Timing Analysis Considering Power Supply Noise Effects. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 493–496, 2000.

[37] J. Ma, J. Lee, and M. Tehranipoor. Layout-Aware Pattern Generation for Maximizing Supply Noise Effects on Critical Paths. In *IEEE VLSI Test Symposium (VTS)*, pages 221–226, 2009.

[38] M. K. Butler and O. N. Mukherjee. Power-Aware DFT-Do We Really Need it?. In *Proceedings IEEE International Test Conference (ITC)*, 2007.

[39] D. Czysz, M. Kassab, X. Lin, G. Mrugalski, J. Rajski, and J. Tyszer. Low Power Scan Shift and Capture in the EDT Environment. In *Proceedings IEEE International Test Conference (ITC)*, paper 13.2, 2008.

[40] J. Rearick. Too Much Delay Fault Coverage Is A Bad Thing. In *Proceedings IEEE International Test Conference (ITC)*, pages 624–633, Nov. 2001.

[41] C. Shi and R. Kapur. How Power Aware Test Improves Reliability and Yield. EE Times, Sept. 15, 2004.

[42] L.-T.Wang, C. E. Stroud, and N. A. Touba. System-on-Chip Test Architectures: Nanometer Design for Testability. Morgan Kaufmann Pub., 2007.

[43] P. Maxwell, I. Hartanto, and L. Bentz. Comparing Functional and Structural Tests. In *Proceedings IEEE International Test Conference (ITC)*, pages 400–407, 2000.

[44] Moderator: K. Butler, Organizer: N. Mukherjee. Power-Aware DFT - Do We Really Need it? Panel, International Test Conference, 2008.

[45] J. Saxena, K. Butler, V. Jayaram, and S. Kundu. A Case Study of IR-Drop in Structured At-Speed Testing. In *Proceedings IEEE International Test Conference (ITC)*, 2003.

[46] J. Li, Q. Xu, Y. Hu, and X. Li. iFill: An Impact-Oriented X-Filling Method for Shift- and Capture-Power Reduction in At-Speed Scan-Based Testing. In *Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pages 1184–1189, 2008.

[47] S. Remersaro, I. Pomeranz, X. J. Lin, J. Rajski, and S. M. Reddy. Scan-Based Tests with Low Switching Activity. In *IEEE Design & Test of Computers*, pages 268–275, June 2007.

[48] J. Wang and D. M. Walker. Modeling Power Supply Noise in Delay Testing. In *IIEEE Design & Test of Computers*, pages 226–233, June 2007.

[49] X. Liu, Y. Zhang, F. Yuan, and Q. Xu. Layout-Aware Pseudo-Functional Testing for Critical Paths Considering Power Supply Noise Effects. In *Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, 2010.

[50] X.Wen, K. Miyase, T. Suzuki, S. Kajihara, Y. Ohsumi, and K. K. Saluja. Critical-Path-Aware X-Filling for Effective IR-Drop Reduction in At-Speed Scan Testing. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 527–533, 2007.

[51] Y.-C. Lin, F. Lu, and K.-T. Cheng. Pseudofunctional Testing. In *IEEE Transactions on Computer-Aided Design*, pages 25(8):1535–1546, 2006.

[52] Y.-C. Lin and K.-T. Cheng. A Unified Approach to Test Generation and Test Data Volume Reduction. In *Proceedings IEEE International Test Conference (ITC)*, pages 18.2, 2006.

[53] I. Polian and H. Fujiwara. Functional Constraints vs. Test Compression in Scan-Based Delay Testing. In *In Proceedings IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pages 1039–1044, 2006.

[54] X. Lin, K. Tsai, C. Wang, M. Kassab, J. Rajski, T. Kobayashi, R. Klingenberg, Y. Sato, S. Hamada, and T. Aikyo Timing-Aware ATPG for High Quality At-speed Testing of Small Delay Defects. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 139–146, 2006.

[55] H. Fujiwara and T. Shimono. On the Accelaration of Test Generation Algorithms. In *Proceedings IEEE Transactions on Computers*, Pages 1137–1144 , 1983.

[56] A. B. T. Hopkins and K. D. McDonald-Maier. Trace Algorithm for Deeply Integrated Complex and Hybrid SoCs. In *Proceedings Adaptive Hardware and Systems (AHS)*, pages 1–6, 2007.

[57] H. F. Ko and N. Nicolici. Automated Trace Signals Identification and State Restoration for Improving Observability in Post-Silicon Validation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, pages 1–6, April 2008.

[58] Y. C. Hsu, F. Tsai, W. Jong, and Y. T. Chang. Visibility Enhancement for Silicon Debug. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 13C18, July 2006.

[59] Brglez. On Testability Analysis of Combinational Networks. In *Proceedings IEEE Symposium on Circuits and Systems*, pages 221–225, 1984.

[60] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller. A Reconfigurable Design-for-Debug Infrastructure for SoCs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 7–12, July 2006.

[61] M. Abramovici. Experience and Opinon (Design for Debug). In *Proceedings IEEE International Workshop on Silicon Debug and Diagnosis*, 2006.

[62] D. D. Josephson. The Mannic Depression of Microprocessor Debug. In *Proceedings IEEE International Test Conference (ITC)*, pages 657–663, October 2002.

[63] D. D. Josephson and B. Gottlieb. The Crazy Mixed Up World of Silicon Debug. In *Proceedings IEEE Custom Integrated Circuits Conference*, pages 665–670, October 2004.

[64] Altera Inc. Design Debugging Using the SignalTap II Embedded Logic Analyzer. http://www.altera.com.

[65] E. Anis and N. Nicolici. On Using Lossless Compresstion of Debug Data in Embedded Logic Analysis. In *Proceedings IEEE International Test Conference (ITC)*, pages 1–10, October 2007.

[66] E. Anis and N. Nicolici. Low Cost Debug Architecture using Lossy Compression for Silicon Debug. In *Proceedings Design, Automation, and Test in Europe (DATE)*, April 2007.

[67] ARM Ltd. Embedded Trace Macrocell Architecture Specification. http://www.arm.com/.

[68] A. B. Hopkins and K. D. McDonald-Maier. Debug Support for Complex Systems On-chip: A Review. In *IEE Proceedings, Computers and Digital Techniques*, pages 197–207, July 2006.

[69] A. B. T. Hopkins and K. D. McDonald-Maier. Trace Algorithms for Deeply Integrated Complex and Hybrid SoCs. In *NASA/ESA Conference on Adaptive Hardware and Systems*, pages 641–646, 2007.

[70] G. Rootselaar and B. Vermeulen. Silicon Debug: Scan Chains Alone Are Not Enough. In *Proceedings IEEE International Test Conference (ITC)*, pages 892–902, September 1999.

[71] MIPS Technologies Inc. EJTAG Trace Control Block Specification. http://www.mips.com.

[72] Semiconductor Industry Association (SIA). *The International Technology Roadmap for Semiconductors (ITRS): 2003 Edition*. http://public.itrs.net/Files/2003ITRS/Home2003.htm, 2003.

[73] N. Stollon, R. Leatherman, B. Ableidinger, and E. Edgar. Multi-Core Embedded Debug for Structured ASIC Systems. http://www.fs2.com/.

[74] B. Vermeulen, T. Waayers, and S. Bakker. IEEE 1149.1-Compliant Access Architecture for Multiple Core Debug on Digital System Chips. In *Proceedings IEEE International Test Conference (ITC)*, pages 55–63, Baltimore, MD, Oct. 2002.

[75] B. Vermeulen, T. Waayers, and S. K. Goel. Core-Based Scan Architecture for Silicon Debug. In *Proceedings IEEE International Test Conference (ITC)*, pages 638– 647, October 2002.

[76] D. Josephson and B. Gottlieb. Debug Methodology for the McKinley Processor. In *Proceedings IEEE International Test Conference (ITC)*, pages 451–460, October 2001.

[77] Xilinx Inc. Chipscope Pro Software and Cores User Guide. http://www.xilinx.com.

[78] B. Vermeulen and S. K. Goel. Design for Debug: Catching Design Errors in Digital Chips. *IEEE Design & Test of Computers*, 19(3):37–45, May 2002.

[79] B. R. Quinton and S. J. E. Wilton. Concentrator Access Networks for Programmable Logic Cores on SoCs. *IEEE International Symposium on Circuits and Systems*, pages 45– 48, May 2005.

[80] M. J. Narasimha. A Recursive Concentrator Structure with Applications to Self-Routing Switching Networks. *IEEE Trans. on Communication*, 42(2):896–897, April 1994.

[81] S. Nakamura and G. M. Masson. Lower bounds on crosspoints in concentrators. *IEEE Trans. on Computers*, C(31):1173–1178, 1982.

[82] X. Liu and Q. Xu. Trace Signal Selection for Visibility Enhancement in Post-Silicon Validation. In *Proceedings Design, Automation, and Test in Europe (DATE)*, Apr 2009.

[83] M. Li, P. Ramachandran, S. Adve, V. Adve and Y. Zhou. Understanding the propagation of hard faults to software and its implications on Resilient Systems Design. In *Proceedings Architecture Support for Programming Languages and Operating Systems (ASPLOS)*, March 2008.

[84] F. K. Hwang. The Mathematical Theory of Nonblocking Switching Networks. World Scientific, New Jersey, 1998.

[85] M. Boulé, J. S. Chenard and Z. Zilic. Adding debug enhancements to assertion checkers for hardware emulation and silicon debug. In *Proceedings EEE Int. Conf. on Computer Design (ICCD)*, pages 294– 299, 2006.

[86] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/ alanmi/abc/.

[87] D. Blaauw, R. Panda, and A. Das. Removing User Specified False Paths from Timing Graphs. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 270–273, 2000.

[88] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 338–342, 2003.

[89] H.-C. Chen and D.-C. Du. Path Sensitization in Critical Path Problem. *IEEE Transactions on Computer-Aided Design*, 12(2):196–207, Feb. 1993.

[90] K.-T. Cheng and H.-C. Chen. Classification and Identification of Nonrobust Untestable Path Delay Faults. *IEEE Transactions on Computer-Aided Design*, 15(8):845–853, August 1996.

[91] K. Heragu, J. H. Patel, and V. D. Agrawal. Fast Identification of Untestable Delay Faults Using Implications. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 642–647, 1997.

[92] Y. Kukimoto and R. K. Brayton. Timing-Safe False Path Removal for Combinational Modules. In *Proceedings International Conference on Computer-Aided Design (ICCAD)*, pages 544–550, 1999.

[93] H. K. Lee and D. S. Ha. On the Generation of Test Patterns for Combinational Circuits. Technical Report 12-93, Dept. of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[94] Z. Li, Y. Min, and R. K. Brayton. Efficient Identification of Non-Robustly Untestable Path Delay Faults. In *Proceedings IEEE International Test Conference (ITC)*, pages 992–997, 1997.

[95] X. Liu and M. S. Hsiao. A Novel Transition Fault ATPG that Reduces Yield Loss. *IEEE Design & Test of Computers*, 22(6):576–584, Nov.-Dec. 2005.

[96] E. McCluskey. *Integrating Functional and Temporal Domains in Logic Design: The False Path Problem and Its Implications*. Kluwer Academic Publishers, Norwell, 1991.

[97] M. Syal, K. Chandrasekar, V. Vimjam, M. S. Hsiao, Y.-S. Chang, and S. Chakravarty. A Study of Implication Based Pseudo Functional Testing. In *Proceedings IEEE International Test Conference (ITC)*, page paper 24.3, 2006.

[98] Synopsys Inc. User Manuals for SYNOPSYS Toolset Version 2007.12.

[99] W. Wu and M. S. Hsiao. Mining Sequential Constraints for Pseudo-Functional Testing. In *Proceedings IEEE Asian Test Symposium (ATS)*, pages 19–24, 2007.

[100] F. Yuan and Q. Xu. On Systematic Illegal State Identification for Pseudo-Functional Testing. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, pages 702–707, 2009.

[101] J. Zeng, M. Abadir, and J. Abraham. False Timing Path Identification Using ATPG Techniques and Delay-Based Information. In *Proceedings ACM/IEEE Design Automation Conference (DAC)*, 2002.

[102] Z. Zhang, S. Reddy, and I. Pomeranz. On Generate Pseudo-Functional Delay Fault Tests for Scan Designs. In *Proceedings IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT)*, pages 215–226, 2005.

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

[104] D. Frank, R. Puri, and D. Toma. Design and CAD Challenges in 45nm CMOS and beyond. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pp. 329–333, 2006.

[105] K. Bowman, et al. Circuit techniques for dynamic variation tolerance. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 4–7, 2009.

[106] T. Austin and V. Bertacco. Deployment of better than worst-case design: solutions and needs. In *Proc. International Conference on Computer Design (ICCD)*, pp. 550–555, 2005.

[107] S. Sde-Paz and E. Salomon. Frequency and Power Correlation between At-Speed Scan and Functional Tests. In *Proc. IEEE International Test Conference (ITC)*, paper 13.3, 2008.

[108] M. R. Choudhury and K. Mohanram. TIMBER: Time borrowing and error relaying for online timing error resilience. In *Proc. Design, Automation, and Test in Europe (DATE)*, pp. 1554–1559, 2010.

[109] M. Kurimoto, et al. Phase-adjustable error detection flip-flops with 2-stage hold driven optimization and slack based grouping scheme for dynamic voltage scaling. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 884–889, 2008.

[110] K. Hirose, et al. Delay-compensation flip-flop with in-situ error monitoring for low-power and timing-error-tolerant circuit design. *Japan Journal of Applied Physics*, 47(4):2779–2787, April 2008.

[111] M. R. Choudhury and K. Mohanram. Masking timing errors on speed-paths in logic circuits. In *Proc. Design, Automation, and Test in Europe (DATE)*, pp. 87–92, 2009.

[112] S.-L. Lu. Speeding up processing with approximation circuits. *Computer*, 37(3):67–73, Mar. 2004.

[113] M. R. Choudhury and K. Mohanram. Approximate logic circuits for low overhead, non-intrusive concurrent error detection. In *Proc. Design, Automation, and Test in Europe (DATE)*, pp. 903–908, 2008.

[114] O. Coudert, R. Haddad and S. Manne. New Algorithm for Gate Sizing: A Comparative Study. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 734-739, 1996.

[115] M. Mani, A. Devgan, and M. Orshansky. An Efficient Algorithm for Statistical Minimization of Total Power under Timing Yield Constraints. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pp. 309-314, 2005.

[116] M. Agarwal, B. C. Paul, M. Zhang, and S. Mitra. Circuit Failure Prediction and Its Application to Transistor Aging. In *Proc. IEEE VLSI Test Symposium (VTS)*, pp. 277–286, 2007.

[117] S. Das, et al. RazorII: In Situ Error Detection and Correction for PVT and SER Tolerance. *IEEE Journal of Solid-State Circuits*, 44(1):32–48, 2009.

[118] B. Greskamp, et al. Blueshift: Designing processors for timing speculation from the ground up. In *IEEE International Symposium on High Performance Computer Architecture*, pp. 213–224, 2009.

[119] G. Yu, et al. Statistical Static Timing Analysis Considering Process Variation Model Uncertainty. In *Proc. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1880-1890, 2008.

[120] J. L. Tsai, et al. A Yield Improvement Methodology Using Pre- and Post-Silicon Statistical Clock Scheduling. In *Proc. International Conference on Computer-Aided Design (ICCAD)*, pp. 611-618, 2004.