Pseudo-Functional Testing for Small Delay Defects Considering Power Supply Noise Effects

Feng Yuan[†], Xiao Liu[†] and Qiang Xu[†‡]
[†]CUhk REliable Computing Laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
[‡]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences
Email: {fyuan,xliu,qxu}@cse.cuhk.edu.hk

ABSTRACT

Detecting small delay defects (SDDs) has become increasingly important to address the quality and reliability concerns of integrated circuits. Without considering functional constraints in the circuits under test, however, existing techniques may generate test patterns that are functionally-unreachable. Such SDD patterns may incur excessive (or limited) power supply noise (PSN) on sensitized paths in test mode, thus leading to over-testing or under-testing of the circuits. In this paper, we propose novel pseudo-functional testing techniques to tackle the above problem. 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 SDD faults in the circuit. Finally, we use ATPG-like algorithm to justify transitions that pose the maximized PSN effects on sensitized critical paths under the consideration of functional constraints. The effectiveness of the proposed methodology is verified with large ISCAS'89 and ILWS'05 benchmark circuits.

1. INTRODUCTION

Power supply noise (PSN) has an ever-increasing adverse impact on circuit timing with technology scaling. As demonstrated in [15], 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 [4, 12]. As shown in [14], however, at-speed scan patterns can be up to 20% slower than any functional patterns. 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 [7, 8, 16]. These test methodologies, unfortunately, lead to the concern for under-testing. That is, if we over-restrict the PSN effects, some defective chips that cannot meet circuit timing requirement may pass delay test, leading to test escapes [2]. 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 [10]. 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, [10] 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, [10] 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. By doing so, the proposed ATPG engine is able to cover much more critical paths when compared to [10]. The main contributions of this paper include:

- We present techniques to generate pseudo-functional SDD test cubes, wherein, by taking the circuit layout information into account, functional constraints related to critical paths are extracted and conformed during SDD test generation;
- We propose an efficient and effective 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;

The remainder of this paper is organized as follows. Section 2 reviews related work and motivates this paper. In Section 3 and Section 4, we detail our proposed pseudo-functional SDD test generation 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.

2. PRELIMINARIES AND MOTIVATION

2.1 Pseudo-Functional Testing

The discrepancy between circuits' activities in functional mode and that in structural test mode significantly affects test quality [1, 13]. Pseudo-functional testing was proposed to tackle this problem and has attracted lots of attention recently [9, 17, 18]. In this technique, functionally-unreachable states in the circuit are extracted [17] and then fed to a constrained ATPG tool to generate functional-like patterns [11].

2.1.1 Pseudo-Functional Path Delay Testing

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 [10], the authors proposed a pseudo-functional test pattern generation technique to maximize the PSN effects on selected critical paths, targeting path delay faults.

As [10] is well-related to this paper, we briefly review it here. In [10], 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 between the aggressors and the victims. Within this range, *PEW* is defined as follows.

$$PEW = 1 - |X_{agg} - X_{vic}|/EffectiveRange \quad (1)$$

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.

[10] 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.

2.2 Motivation

As shown in [10], 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 [10] 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 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.

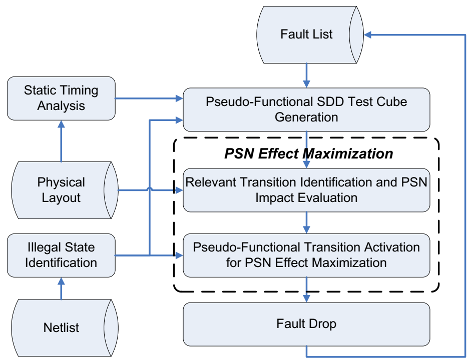
Due to the above, small delay defect testing has been widely accepted by the industry, wherein we try to detect transition faults by propagating their faulty effects through long paths whenever possible. Compared to path delay testing, the number of SDD test patterns increases almost linearly with the 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.

3. PROPOSED METHODOLOGY

Fig. 1 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 [17, 10]. As can be observed from Fig. 1, our proposed ATPG flow mainly contains two parts: (i). pseudo-functional SDD test cube generation; and (ii) PSN effect maximization.

[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.



Fault List
Static Timing Analysis
Pseudo-Functional SDD Test Cube Generation
PSN Effect Maximization
Physical Layout
Relevant Transition Identification and PSN Impact Evaluation
Illegal State Identification
Pseudo-Functional Transition Activation for PSN Effect Maximization
Netlist
Fault Drop

Figure 1: Proposed Pseudo-Functional SDD Test Generation Flow

To generate pseudo-functional SDD test cube, we extend the conventional SDD test generation method presented in [6] 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 by setting them as functional constraint as [10].

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.

4. MAXIMIZING PSN EFFECTS UNDER FUNCTIONAL CONSTRAINTS

4.1 Relevant Transition Identification

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 [10] 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 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.*

[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 [10] to guide the filling procedure for every test pattern.

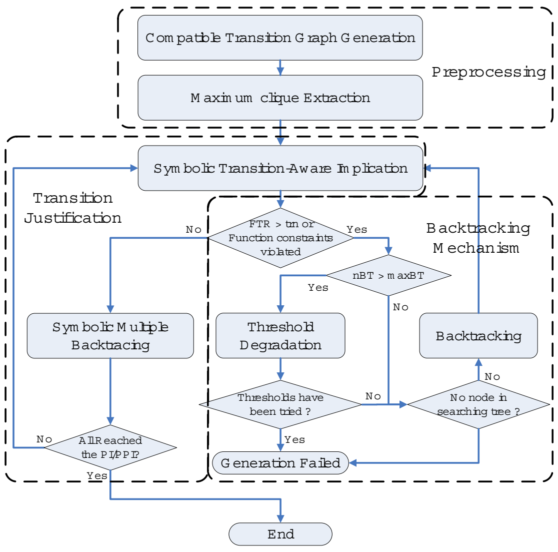

Compatible Transition Graph Generation
Maximum clique Extraction
Preprocessing
Symbolic Transition-Aware Implication
Transition Justification
Backtracking Mechanism
FTR > tm or Function constraints violated
No
Yes
nBT > maxBT
Yes
No
Symbolic Multiple Backtracing
Threshold Degradation
Backtracking
Thresholds have been tried ?
No
No node in searching tree ?
No
Yes
All Reached the PI/PPI?
No
Yes
Generation Failed
End

Figure 2: Flowchart for Our Pseudo-Functional Relevant Transition Generation ALgorithm

4.2 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.

Our proposed algorithm is composed of three main parts as shown in the flowchart in Fig. 2. 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, and then 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.

4.2.1 Preprocessing

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

To resolve the above 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. 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 quite efficient because logic implication can be conducted efficiently and *CTG* graph construction is a one-time effort only. Before transition justification, we first extract the maximum clique of CTG and target the transitions within it in the following steps.

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

4.2.2 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. 3, 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.

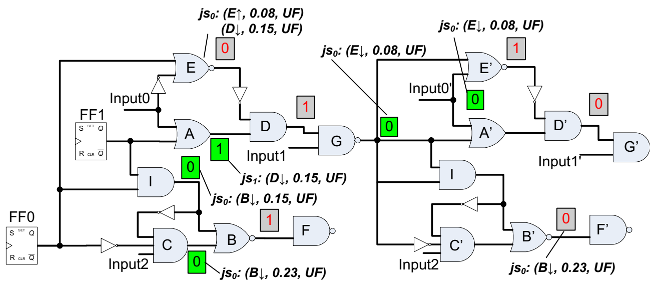


Figure 3: Symbolic Justification Mechanism: An Example

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 [3], 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 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. 3, we use Fig. 4 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*.

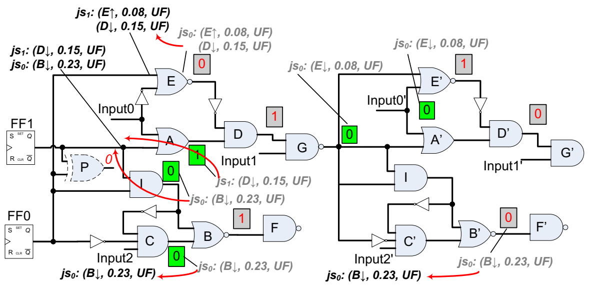


Figure 4: Symbolic Multiple Backtracing: An Example

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 described in [10]. As the example shown in Fig. 5, 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. 5, 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.

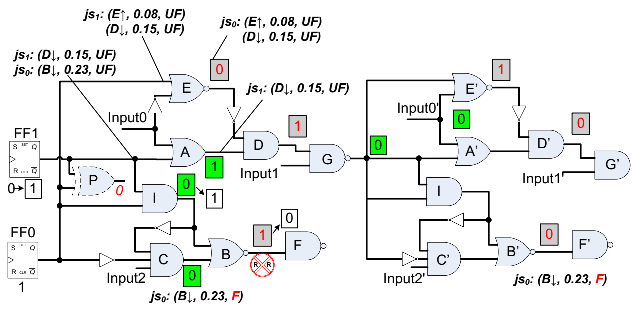


Figure 5: Impact of Implication Order: An Example

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 backtracing 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.

Benchmark	Number of Gate (#)	Fault Coverage (%)	Path Activation Ratio (%)
s5378	2958	80.57	77.82
s9234	5808	80.29	81.68
s13207	8589	83.82	83.27
s15850	10303	82.28	64.13
s38417	23815	92.76	79.32
s38584	20679	86.29	72.56
des	154323	85.64	98.7
Average		84.52	79.64

Table 1: SDD Fault Detection Quality.

4.2.3 *Multi-Level Backtracking Mechanism*

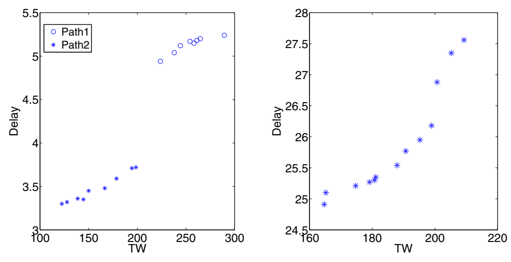
In conventional ATPG process [3], backtracking is conducted as soon as a logic conflict is detected during implication. While for our problem, we can accept certain amount of temporary logic conflicts that reduce the number of desired transitions and resolve them in later stage. Consequently, we need 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 $nBT \le maxBT$ or all the threshold levels have been tried.

5. EXPERIMENTAL RESULTS

5.1 Experimental Setup

We implement our layout-aware pseudo-functional SDD pattern generation framework on top of an academic ATPG tool *Atalanta* [5], which originally targets stuck-at faults using FAN algorithm [3]. Experiments are conducted on several large ISCAS'89 and IWLS'05 benchmark circuits that are available to us. Among them, the largest circuit *des* contains about 150k logic gates. We synthesize them using UMC's *130nm* CMOS technology with *1.08V* power supply voltage, and layout them using commercial EDA tools.



(a) s38417 (b) des

Figure 6: *TW*-Delay Correlation Plot

Benchmark	SDD_1	SDD_2	SDD_3	SDD_4			
	TW_r	TW_w	TW_f	TW_o	$OR_{TW}(\%)$	$OW_{TW}(\%)$	OF_{TW}
s5378	19.15	29.36	23.62	26.56	38.69	-9.54	12.45
s9234	24.31	38.17	28.88	30.85	26.90	-19.18	6.82
s13207	35.98	61.37	41.51	48.65	35.21	-20.73	17.20
s15850	28.79	50.07	31.36	38.58	34.00	-22.95	23.02
s38417	57.8	105.25	72.91	91.83	58.88	-12.75	25.95
s38584	73.45	132.89	85.2	115.68	57.49	-12.95	35.77
des	306.89	431.25	346.87	419.68	36.75	-2.68	20.99
Average					41.13	-14.40	20.32

$OR_{TW} = \frac{TW_o - TW_r}{TW_r} \times 100\%$ $OW_{TW} = \frac{TW_o - TW_w}{TW_w} \times 100\%$ $OF_{TW} = \frac{TW_o - TW_f}{TW_f} \times 100\%$

Table 2: Comparison among Different SDD Patterns.

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. This is because, we can only obtain relatively accurate delay value by applying both PSN simulation and timing analysis and it is not affordable to integrate such time-consuming process into our ATPG flow. Instead of doing so, we employ the *transition weight* metric TW to evaluate the PSN effects caused by certain transitions on a sensitized path and then try to maximize the overall effective TW by justifying relevant transitions in the original SDD test cube. 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 SDD test cubes for *s38417* and one pattern for *des*, and then we randomly fill the X-bits in them and calculate the sum of the activated relevant transitions TW for several rounds. To obtain the delay information under PSN effects, we first perform IR-Drop analysis on the layout with 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 paths for these patterns, we plot TW-delay figure as shown in Fig.6. It can be observed that, although they are not perfectly correlated, 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.

In our second experiment, we randomly select six true critical paths sensitized by the SDD test patterns generated by [6] and then record the maximum TW and minimum TW when these paths are under test, as shown by "Orig. MIN" and "Orig. MAX" in Fig. 7. The corresponding result with the proposed method is shown by "Pro.". We utilize the method in [10] to generate corresponding patterns on each selected path and the TW result is denoted by "[10]" in Fig. 7. We can observe that some paths (e.g., Path 6 in Fig. 7) can never be sufficiently tested with the original test set, as the maximum TW is smaller than the one obtained by both the proposed one and [10], causing possible test escapes. We also observe that, in most cases, a path can be sensitized with much higher TW value than the one obtained under functional constraints (e.g., Path 1 in Fig. 7), and hence leading to possible test overkill. When comparing the induced PSN effects from the proposed method against the one in [10] for the same path, we can observe that the proposed one achieves higher TW than [10] on every path. This demonstrates the proposed ATPG-like X-filling technique is able to induce higher PSN effects under functional constraints when compared with existing solution, thus improving test quality.

Then, Table .1 presents the SDD detection quality of our proposed SDD patterns. Transition fault coverage is shown in Column 3 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 4 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 patterns, denoted as *sensitized paths*. The *Path Activation Ratio* is the ratio between these two sets of paths. On average, there are 80% 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.

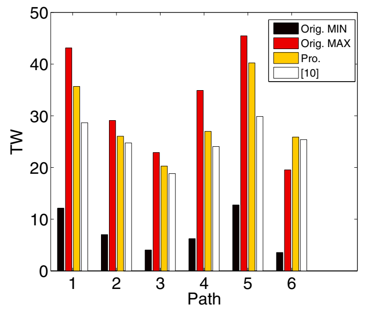


Figure 7: TW comparison for 6 Paths of *des*

Table 2 presents the comparison between different X-filling mechanisms for small delay defects, in which we generate pseudo-functional SDD test cubes first using the techniques presented in Section 3, and fill the X-bits in test cubes differently to obtain four kinds of SDD patterns: (1) pseudo-functional patterns with randomly-filled X-bits, denoted as SDD_1; (2) test patterns to maximize PSN effects without considering functional constraints, represented as SDD_2; (3) test patterns to fast generate PSN effects under functional constraint, where we speed-up relevant transition generation without applying multi-level backtracking, shown as SDD_3; (4)our proposed pseudo-functional patterns to maximize the PSN effects under functional constraints, depicted as SDD_4;

Columns 2-8 list the comparison among the four kinds of SDD patterns. Columns TW_r, TW_w, TW_f and TW_o represent the average activated transition weight for different kinds of patterns. OR_{TW}, OW_{TW} and OF_{TW} are calculated as $OR_{TW} = \frac{TW_o - TW_r}{TW_r} \times 100\%$, $OW_{TW} = \frac{TW_o - TW_w}{TW_w} \times 100\%$, and $OF_{TW} = \frac{TW_o - TW_f}{TW_f} \times 100\%$, respectively. By comparing test patterns generated using the proposed solution against conventional pseudo-functional patterns with randomly-filled X-bits, it can be observed from Column 6 that our method can achieve up to 59% more PSN effects for benchmark s38417, and for all 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 14% less PSN effects on average for all benchmark circuits, and scan patterns for benchmark s15850 can result in up to 23% 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 pat-

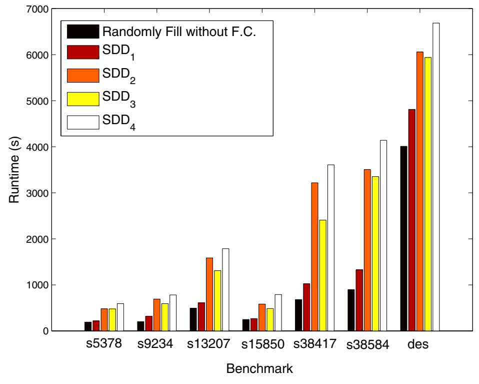


Figure 8: Runtime Comparison

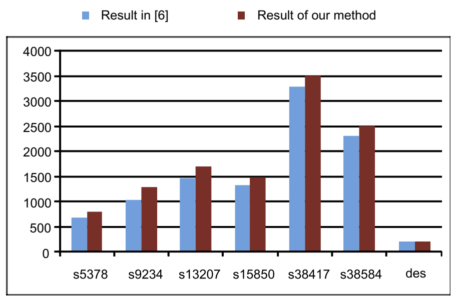


Figure 9: Pattern Count Comparison

terns. Otherwise, circuits can be over-tested, leading to significant test yield loss. Then, we compare our proposed results fast maximization ones to demonstrate the effectiveness of multi-level backtracking mechanism. Without applying such technique, we lose to generate 20% PSN effect. In particular, we improve 35% PSN effect for benchmark s38584.

Next, we study the impact on ATPG runtime with the proposed pseudo-functional SDD testing technique. The result is shown in Figure. 8, and we also plot the runtime for conventional non-pseudo-function SDD ATPG with randomly filling for comparison. First of all, we notice that functional constraints can be processed very efficiently in both test cube generation procedure and relevant transition generation procedure. This conclusion is drawn by observing the close runtime between pseudo-functional random filling ATPG (Denoted by "R.F. under F.C.") and non-pseudo-functional random filling ATPG (Denoted by "Non-Functional pattern R.F. without F.C."), and between our proposed ATPG (Denoted by "Our proposed") and the ATPG that maximize PSN without functional constraint (Denoted by "PSN MAX without F.C."). Secondly, on average, we need to pay more than twice runtime of proposed solution than the pseudo-functional random filling ATPG, and for some extreme cases (e.g. s38417) this ratio may reach as many as 3 times. The main overhead here is caused by relevant transition extraction, process and generation. However, as we can see, the extra runtime does not grow exponentially with the increase of circuit size, which indicates the good scalability of our proposed method. Thirdly, multi-level backtracking is a very timing-consuming part in relevant transition generation as we need to try different fail ratio thresholds and it may introduce some meaningless search effort. To extract runtime used on such mechanism, we compare our proposed ATPG with a fast PSN maximization ATPG (Denoted by "PSN FMAX with F.C."). It can be observed that we significantly improve PSN effect by taking only up to 30% extra runtime, which should be worthwhile considering the associated improvement of test quality.

In [6], 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 [6]. We compare the pattern count between the two methods, as shown in the Fig.9. 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.

6. CONCLUSION

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.

7. ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation of China (NSFC) under grant No. 60876029, in part by the General Research Fund CUHK417807 and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

8. REFERENCES

[1] E. Alpaslan, J. Dworak, B. Kruseman, A.K. Majhi, W.M. Heuvelman, and P. van de Wiel. NIM- A Noise Index Model to Estimate Delay Discrepancies between Silicon and Simulation. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 1373–1376, 2010.

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

[3] H. Fujiwara and T. Shimono. On the Accelaration of Test Generation Algorithms. C-32(12):1137–1144, Dec. 1983.

[4] A. Krstic, Y.-M. Jiang, and K. T. Cheng. Pattern Generation for Delay Testing and Dynamic Timing Analysis Considering Power-Supply Noise Effects. *IEEE Transactions on Computer-Aided Design*, 20(3):416–425, March 2001.

[5] 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.

[6] 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 *Proc. IEEE Asian Test Symposium (ATS)*, pp. 139–146, 2006.

[7] 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 *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 1184–1189, 2008.

[8] J. Li, X. Liu, Y. Zhang, Y. Hu, X. Li, and Q.Xu. On Capture Power-Aware Test Data Compression for Scan-Based Testing. In *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 67–72, 2008.

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

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

[11] X. Liu and M. S. Hsiao. A Novel Transition Fault ATPG that Reduces Yield Loss. *IEEE Design & Test of Computers*, pp. 576–584, November 2005.

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

[13] P. Pant and J. Zelman. Understanding Power Supply Droop during At-Speed Scan Testing. In *Proc. IEEE VLSI Test Symposium (VTS)*, pp. 227–232, 2009.

[14] 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.

[15] C. Tirumurti, S. Kundu, S. Sur-Kolay, and Y.-S. Chang. A Modeling Approach for Addressing Power Supply Switching Noise Related Failures of Integrated Circuits. In *Proc. IEEE/ACM Design, Automation, and Test in Europe (DATE)*, pp. 1078–1083, 2004.

[16] J. Wang and D. M. Walker. Modeling Power Supply Noise in Delay Testing. *IEEE Transactions on Computers*, 24(3):226–233, June 2007.

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

[18] F. Yuan and Q. Xu. Compression-Aware Pseudo-Functional Testing. In *Proc. IEEE International Test Conference (ITC)*, paper 9.1, 2009.