Case Study: Efficient SDD Test Generation for Very Large Integrated Circuits

Ke Peng[1], Fang Bao[1], Geoff Shofner[2], LeRoy Winemberg[2], Mohammad Tehranipoor[1]
[1]ECE Department, University of Connecticut
[2]Freescale Semiconductor

***Abstract*—Semiconductor industry has come to the era to rely heavily on detecting small-delay defects (SDDs) for high defect coverage of manufactured digital circuits and low defective parts per million (DPPM). Traditional timing-unaware transition-delay fault (TDF) ATPGs are proven to be inefficient in detecting SDDs. The commercial timing-aware ATPGs have been developed for screening SDDs, but they suffer from large pattern count and CPU runtime. The previously proposed methodologies are either inefficient or too complex in terms of memory and runtime to be applied to large industry designs (>few million gates). In this paper, we present a new SDD-based pattern grading and selection procedure to meet the SDD test challenges in practice. We propose techniques to reduce the runtime and memory complexity and make the procedure applicable and scalable to large industry designs. Experimental results on both academic and industry circuits demonstrate the efficiency of our procedure; it detects a greater number of SDDs with a much lower pattern count and CPU runtime.**

I. Introduction

Testing for delay defects, also referred to as timing-related defects, has become extremely vital for ensuring product quality and in-field reliability in the very deep-submicron (VDSM) regime. Such defects can fail the chip by introducing extra signal-propagation delay to produce an invalid response when the chip operates at its operating frequency [1] [2]. Small-delay defect (SDD) is one kind of such timing defects. SDDs were not being seriously considered when testing designs at higher technology nodes since they only introduced a small amount of extra delay to the design, which seldom results in failure of the design with comparable lower frequency and larger slack margins. However, when technology scales to 45nm and below, resulting in an increase in design density and operating frequency, SDDs requires a serious consideration [3]. Since a SDD only introduces a small amount of extra delay to the design, it is commonly recommended to detect them via the long paths, or least-slack paths running through the fault sites.

The transition-delay fault (TDF) model has been widely used in industry for testing delay defects. Experiments have demonstrated that TDF test patterns can achieve a defect coverage level that stuck-at patterns alone cannot [1]. Unfortunately, the TDF pattern set has a limited ability for detecting SDDs in the device and meeting the high SDD test coverage requirements. Per the demands from industry, timing-aware ATPG tools [4] [5] have been developed for SDD detection. However, its significantly increased CPU runtime and pattern count has limited its usage in real applications. n-detect TDF ATPG can be considered as an alternate solution for screening SDDs [6] [7]. However, its extremely large pattern count makes it impractical for large industry designs.

*The work was supported in part by NSF under Grants no. ECCS-0823992 and CCF-0811632. The work of Ke Peng for implementation on industry circuits was done at Freescale Semiconductor.

Figure 1 presents the normalized pattern count, number of detected SDDs, and CPU runtime of 1-detect, n-detect ($n = 5, 10, 20$) and timing-aware (ta) pattern sets for the IWLS benchmark ethernet (138,012 gates and 11,617 flip-flops) [8]. The detected SDD is defined as a detected TDF with slack equal or smaller than $0.3T$, where T is the clock period of the design. It can be seen that n-detect and timing-aware pattern sets can detect more SDDs than traditional 1-detect timing-unaware pattern set (1.5-1.9X). However, the penalty is large pattern count (3.6-12.2X) and CPU runtime (3.0-12.0X). It is obvious that as n increases, the increase in pattern count and CPU runtime of n-detect ATPG are approximately linear. For this design, the timing-aware ATPG results in a pattern count comparable to 5-detect ATPG. But its CPU runtime is even larger than 20-detect ATPG.

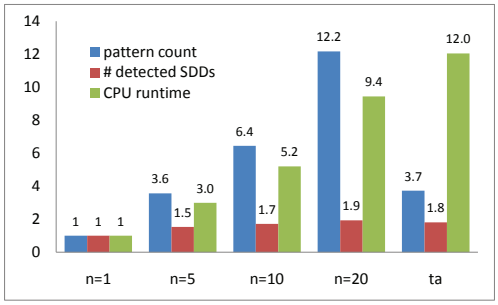

Fig. 1. Normalized pattern count, detected SDDs and CPU runtime of different pattern sets for ethernet. The reuslts are normalized with respect to 1-detect pattern set.

A. Related Prior Work

Several techniques have been presented for screening SDDs. A method to generate K longest paths per gate for testing transition faults was proposed in [9]. In [10], a static-timing-analysis based method was proposed to generate and select patterns that sensitize long paths by masking the observation points of short paths and intermediate paths. The authors in [11] proposed a delay fault coverage metric to detect the longest sensitized path affecting a TDF fault site. It is based on the robust path delay test and attempts to find the longest sensitizable path passing through the target fault site. The authors in [12] proposed path-based and cone-based metrics for estimating the path delay under test, which can be used for path length analysis. All these long path-based methods are suffering from high complexity, extended CPU runtime, and limited by the shortcomings of path delay fault model.

Two hybrid methods were proposed in [13] using 1-detect and timing-aware ATPGs to detect SDDs based on fault classification. Another circuit topology-based fault classification method was proposed in [14]. It differentiates faults to be critical and non-critical for timing-aware and timing-unaware ATPGs, respectively. The efficiency of these fault classification-

978-1-61284-656-9/11/$26.00 ©2011 IEEE

based methods is questioned since it still results in a pattern count much larger than 1-detect TDF ATPG.

The output-deviation based method was proposed in [15] [16]. This method uses the delay defect probability matrix (DDPM) of each gate to calculate the output deviations, which are used for pattern selection. However, in case of a large number of gates along the paths, the output deviation metric can saturate and make the procedure inaccurate.

In [17], a SDF-based hybrid method was proposed to generate patterns with minimized pattern count and large long path sensitization. This method was enhanced in [18], by taking the impacts of power supply noise and crosstalk into consideration when calculating the path delay. A similar pattern grading and selection procedure was presented in [19], which is based on statistical timing analysis and takes process variations and crosstalk effects into consideration. All these methods use sensitized long paths as a criteria to grade and select patterns, which makes them difficult to scale for large industry designs with millions of gates.

B. Contributions and Paper Organization

In this paper, we proposed a new pattern grading and selection procedure for screening SDDs, which is based on detected SDDs and their actual slack, rather than sensitized long paths like [17], [18], [19]. This significantly saves the CPU runtime of path tracing and hardware for storing sensitized paths. Several techniques are proposed in this paper to enable the procedure scalable to very large industry circuits. Before generating the original pattern repository, static timing analysis (STA)-based timing-critical fault selection is performed to save ATPG runtime and hardware resources. After pattern generation, fault simulation is performed on each individual test pattern for the pattern evaluation and selection. Parallel fault simulation is used to further reduce the CPU runtime. A new fault merging technique is also proposed to reduce the hardware resources and data processing runtime in the pattern selection procedure.

The pattern selection procedure will minimize the overlap of detected SDDs between patterns. This can ensure that only the most effective patterns with minimum overlap between detected SDDs can be selected and can further reduce the selected pattern count. 1-detect top-off ATPG is performed after pattern selection to ensure that our final pattern set can detect all the testable TDFs in the design. Several new metrics are used to evaluate the efficiency of our final pattern set for screening SDDs on large designs.

The remainder of this paper is organized as follows. Section II presents our proposed techniques used for reducing CPU runtime and memory. Our pattern grading and selection procedure is presented in Section III. The proposed procedure is applied to several academic and industrial circuits and the experimental results are presented in Section IV. Finally, we conclude our work in Section V.

II. Techniques for Reducing Runtime and Memory

A. Critical Faults Identification

We use n-detect pattern set as the original pattern repository since it has demonstrated its efficiency for screening SDDs. However, the CPU runtime of n-detect ATPG may still be significant when n is large. Furthermore, the n-detect ATPG on the entire fault list results in a significantly large pattern count, which requires large hardware resources and CPU runtime for the following fault simulation step. In fact, there is a large portion of faults in a design that may never be timing-critical, and it is not necessary to run n-detect ATPG on them. Therefore, we identify and select the timing critical faults before running n-detect ATPG to avoid unproductive consumption on CPU runtime and hardware resources. In practice, the proposed procedure can be applied to any kind of pattern set.

TABLE I
Comparison between TF- and CF-based methods for 10-detect ATPG on two academic circuits and one industry circuit.

Circuit		TF-based	CF-based	Percent
wb_ conmax	# faults	347,300	28,981	8.34%
	# patterns	3,302	771	23.35%
	CPU	2m11s	38s	29.01%
ether- net	# faults	868,248	28,209	3.25%
	# patterns	17,582	5,906	33.59%
	CPU	8m37s	2m26s	28.24%
Circuit_ A	# faults	3,396,938	377,857	11.12%
	# patterns	>500K	56,784	<11.36%
	CPU	>5days	17h30m03s	<14.58%

We use the static timing analysis (STA) tool for critical fault selection. Note that the STA tool reports the minimum fault slack by calculating the length of the longest path running through the fault site. In reality, the actual fault slack after pattern generation may not necessarily be equal to this minimum value since (i) the longest path running through the fault may not be testable, and (ii) the ATPG tool does not generate patterns to detect the target fault via the longest path. A critical fault selection slack threshold is needed for the timing critical fault selection. All the TDFs with minimum slack equal or smaller than the pre-defined slack threshold will be selected as timing-critical faults.

Table I shows the efficiency of the proposed critical fault selection method on two academic circuits (ethernet and wb_conmax) and one industry circuit (shown as "Circuit_A"). The slack threshold for critical fault selection is $0.3T$, where T is the clock period. 10-detect ATPG is performed on both total fault (TF) list and selected critical fault (CF) list of the circuits. It is clearly seen that the number of faults for 10-detect ATPG is significantly reduced after critical fault selection for all these three circuits. It can also be seen that the CPU runtime and pattern count is significantly reduced after critical fault selection. This is because the selected critical fault list is significantly shrunk. Note that the fault coverage between the TF-based and CF-based pattern sets is different. Since top-off ATPG will be performed after our pattern selection procedure for compensation, it does not impact the fault coverage of our final pattern set.

B. Parallel Fault Simulation

After pattern generation, fault simulation is performed on each of the test patterns in the pattern repository. The individual fault simulation is based on the entire fault list of the design. With the fault simulation, the detected fault list of each pattern and the actual slack of each detected fault can be obtained, based on which we grade and select patterns. The individual fault simulation may consume a large CPU runtime, especially for the modern industry circuits with a large pattern count. For instance, fault simulation on a single test pattern of "Circuit_A" may take 50-65 seconds. Therefore, for the 10-detect pattern set with 56,784 patterns, it may need 39 days for the fault

simulation if running one by one. To reduce the CPU runtime, parallel fault simulation is proposed in this paper.

Since the individual fault simulation is independent of each other, we divide the pattern repository into groups and run fault simulation on each group simultaneously as shown in Figure 2. In this example, i ($i \in [0, pat.count)$) represents the pattern ID. Therefore, each group runs fault simulation on k test patterns according to the pattern IDs, except for the last fault simulation group, which runs fault simulation on the remaining test patterns in the pattern repository. The fault simulation on all these groups can be run simultaneously. In this way, if the pattern repository is equally divided into $k+1$ different groups, the fault simulation can be sped up approximately $(k+1)$X. For example, if we divide the above 10-detect pattern set of "Circuit_A" into 100 fault simulation groups and run them simultaneously, it can be finished in less than 10 hours.

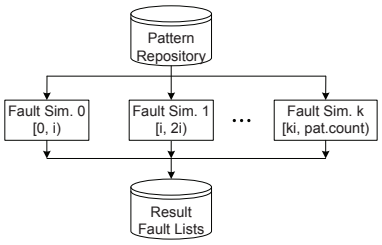

Fig. 2. Parallel fault simulation.

C. Fault Merging

After fault simulation, each test pattern will result in an fault list. The fault lists of all the test patterns may require a large hardware resources, especially for large industry circuits. Again, take the above "Circuit_A" as an example, the size of the fault list from a single test pattern may reach to 2M bytes on average, even after file compression. Therefore, for its 10-detect pattern set with 56,784 patterns, over 100G bytes hardware resources is needed to store the fault lists. We propose a fault merging technique in the paper to meet this challenge and save hardware resources.

Assuming that there are n test patterns in a pattern repository, and they detect m TDF faults in total. Then we can merge the fault lists of these n test patterns to be a pattern-fault matrix like Figure 3. In the matrix, the n patterns are named as $P_{1,2\cdots n}$ according to their pattern IDs and the m faults are named as $F_{1,2\cdots m}$ according to their assigned IDs. If fault F_i is detected by pattern P_j, then the element a_{ij} represents the actual slack of fault F_i when detected by pattern P_j. Otherwise, if it is not detected by pattern P_j, a_{ij} goes to 0 or some other value that the fault slack may never reach to so that it can be differentiated from the actual slack of the fault. This pattern-fault matrix can be compressed by matrix compression methods [20]. Note that a fault may be detected by one or several test patterns, therefore the pattern-fault matrix may come to a sparse matrix, which can be compressed with a considerable ratio.

$$\begin{array}{c} \\ F_1 \\ F_2 \\ \vdots \\ F_i \\ \vdots \\ F_m \end{array} \begin{array}{c} \begin{array}{cccccc} P_1 & P_2 & \cdots & P_j & \cdots & P_n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix} \end{array}$$

Fig. 3. Pattern-fault matrix used for fault merging and compression.

Since only detected SDDs are considered for pattern grading and selection, we can only include SDDs in the pattern-fault matrix to further reduce its size. With the pattern-fault matrix and sparse matrix compression technique, we can merge the fault list and reach a 60-100X reduction in the data volume for both academic and industry circuits. This fault merging and compression technique can be easily integrated into current commercial EDA tools.

III. Pattern Evaluation and Selection

A. Patttern Evaluation

After pattern generation and fault simulation, we can evaluate each test pattern for our pattern selection procedure. Since only SDDs are used for pattern evaluation, a slack threshold (SL_{thr}) is needed to differentiate the detected TDFs and SDDs. If the actual slack of a detected TDF is equal or smaller than SL_{thr}, it will be considered as an SDD and will be counted for pattern evaluation. Otherwise, it will be considered as an gross TDF and will be ignored in the pattern evaluation procedure. In this paper, we use $0.3T$ as the slack threshold to differentiate the gross TDFs and SDDs, where T is the clock period. In practice, some other slack thresholds can be used as well. It is obvious that the greater number of detected SDDs, the more effective the pattern is considered. Our pattern evaluation and selection procedure is based on this observation.

During pattern evaluation, we assign a weight to each detected SDD based on its slack, and then calculate the weight of the target test pattern using Equation (1).

$$W_{Pi} = \sum_{i=1}^{N} W_{sddi} \tag{1}$$

Where W_{Pi} is the weight of pattern Pi, N is the total number of detected SDDs of pattern Pi, and W_{sddi} represents the weight of i-th detected SDD. In this paper, we assign a weight of 1.0 for each detected SDD. The gross TDFs are ignored in the pattern evaluation procedure. Therefore, in this work, the weight of a pattern represents the number of its detected SDDs. Similar to the long path-based method in [17], the definition of pattern weight in this paper is also open ended. For example, given the available slack for each SDD, a metric with actual slack representing W_{sddi} can also be developed.

B. Pattern Selection

The pattern selection procedure is shown in Figure 4. Before pattern selection, the inefficient patterns without SDD detection are removed since they will never be selected. This can save the following pattern selection runtime. During pattern selection, the pattern with the largest weight will be selected first. Then the newly detected SDDs by the selected pattern will be removed from the remaining patterns for re-evaluation. After that, the pattern with the largest weight in the remaining pattern set is secondly selected. This procedure is repeated to select patterns from the original pattern repository until the pattern weight threshold (shown as "W_{thr}") is met (i.e., $WP_{max} < W_{thr}$). Therefore, only patterns with weight equal or larger than this threshold can be selected. The pattern selection procedure will check the overlap of detected SDDs between selected patterns and ensure that only the unique detected SDDs of a pattern are

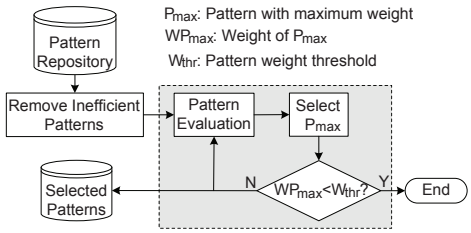

Fig. 4. Selection and sorting procedure for TDF patterns based on their unique weight.

used for the pattern evaluation and selection, which can further reduce the selected pattern count.

Assume that there are N patterns in the original pattern repository, and a maximum of M SDDs are detected by a pattern. The worst-case time complexity of the pattern selection algorithm is $O(N^2M)$ where $N >> M$ for large designs. In fact, this is the worst-case scenario which may never be met in real applications since several new techniques are added to the procedure to speed it up: (i) The inefficient patterns are removed before performing pattern selection; (ii) Once a pattern is selected, all its detected SDDs will be removed from the detected SDD lists of the remaining patterns. This will reduce the SDD list of each pattern significantly after several patterns are selected; (iii) After re-evaluation, the new inefficient patterns in the remaining pattern set will be removed since they will never be selected; and (iv) the pattern selection will be terminated if $WP_{max} < W_{thr}$, no matter how many patterns are left in the repository. According to our experiment on the 10-detect pattern set of "Circuit_A" (56,784 patterns), the CPU runtime for pattern selection is just *1 hour and 6 minutes*.

IV. Experimental Results

In this section, we will present experimental results on both academic and industry circuits. The characteristics of these circuits are shown in Table II in a form of total number of cells (gates+FFs). For the academic circuits, 180nm Cadence Generic Standard Cell Library was used for physical design. 90nm technology library was used for the industry circuits. All patterns are generated using TDF launch-off-capture (LOC) method. After pattern selection, top-off ATPG is run on the undetected faults to ensure that all the testable faults in the design are detected by our final pattern set. The entire flow for our pattern selection procedure is shown in Figure 5.

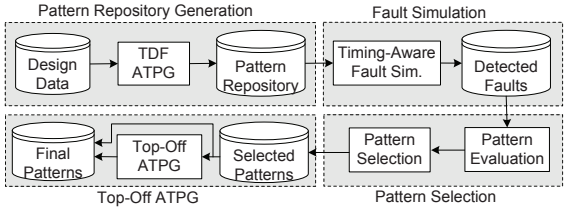

Fig. 5. The entire SDD-based pattern selection flow.

TABLE II
Characteristics of the experimental circuits.

Circuit	wb_conmax	ethernet	Circuit_A	Circuit_B
# total cells	47,757	149,638	1.7 million	3.7 million

A. Pattern Set Analysis

In this subsection, we compare our pattern set, which is selected patterns plus top-off patterns, with 1-detect and timing-aware pattern sets generated using a commercial ATPG tool. In this experiment, 1-detect and timing-aware (ta) pattern sets are generated based on the entire fault list. After applying our pattern selection procedure to $5, 10, 20$-detect pattern sets of ethernet, 1-detect top-off ATPG is run to ensure that all these pattern sets have the same fault coverage. The pattern count and number of detected SDDs of each pattern set is shown in Figure 6.

TABLE III
Comparison of seleced patterns from SDD-based and LP-based methods in [17] on wb_conmax and ethernet.

Pat. Set		# ori. Pat.	SDD-based		LP-based [17]	
			# Sel. Pat.	CPU	# Sel. Pat.	CPU
wb_ conmax	n=5	433	274	3s	404	1m21s
	n=10	771	351	6s	674	2m22s
	n=20	1,446	423	11s	1,122	4m29s
ether- net	n=5	3,048	350	4s	1,094	8m50s
	n=10	5,906	428	7s	1,429	14m16s
	n=20	11,558	532	15s	1,739	26m21s

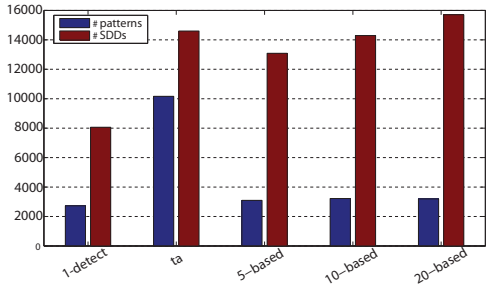

Fig. 6. Pattern set comparison in terms of pattern count and detected SDDs for ethernet. Here "5-based" represents selected patterns from 5-detect pattern set plus its corresponding top-off patterns.

It can be seen from Figure 6 that timing-aware ATPG can detect a lot more SDDs compared with 1-detect ATPG. However, the penalty is that it also results in a much larger pattern count. In contrast, the pattern set generated with our proposed method can detect SDDs comparable to timing-aware pattern set but with a much lower pattern count, which is very close to 1-detect pattern set. In fact, our pattern set generated from 20-detect patterns (shown as "20-based") can even detect more SDDs than timing-aware pattern set. This is because 20-detect pattern set generated on critical faults can detect more SDDs and our pattern set generated from it maintains its SDD detection efficiency. Since only high-quality patterns are selected, our pattern count is significantly reduced.

B. Comparison with LP-Based Method

In this subsection, we compare the proposed SDD-based method with the previous long path-based (LP-based) method [17] in terms of pattern selection efficiency and CPU runtime. To make a fair comparison, we apply both methods to the same n-detect pattern set, and ensure that (i) the slack threshold SL_{thr} in the SDD-based pattern selection is equivalent to the long path threshold LP_{thr} in the LP-based method, i.e., $SL_{thr} + LP_{thr} = T$, where T is the clock period; (ii) the SDD-based method selects patterns that can detect all the SDDs detected by the original n-detect pattern set; and (iii) the LP-based method selects patterns that can sensitize all the long paths sensitized by the original n-detect pattern set, which ensures the selected patterns to detect all the SDDs detected by the original pattern set. Therefore, the selected patterns from both methods have the same SDD coverage.

Table III presents the pattern selection results from both SDD-based and LP-based methods. In this experiment, both methods are applied to 5-detect, 10-detect, and 20-detect pattern sets of IWLS benchmarks ethernet and wb_conmax, respectively. It is clearly seen that the selected pattern set (shown as "# Sel. Pat.") from SDD-based method is much smaller than LP-based method. This is due to the fact that sensitizing an extra unique long path may not necessarily contribute to detect more new SDDs, if these SDDs have been detected by some other already-selected patterns. The LP-based method did not take this issue into consideration and consequently results in a comparable large pattern count. When comes to CPU runtime, the SDD-based method is significantly better than the LP-based method. This is because (i) the pattern selection procedure is much faster after introducing several new techniques as discussed in Section III-B; and (ii) the SDD-based method saves the runtime for path tracing, which is dominant in the LP-based method.

C. Multiple Detection Analysis for Critical Faults

In presence of uncertainties such as process variations, on-chip temperature, power supply and crosstalk noises, the length of a path can vary to a large extent. As a result, it is desirable to detect each critical fault via various long paths. If a pattern set can detect the critical faults via a large number of different long paths, it is superior to keep the high test quality in presence of uncertainties mentioned above. Figure 7 presents a comparison between our n-detect-based pattern sets ($n = 5, 10, 20$) and timing-aware pattern set of ethernet benchmark. In this figure, we compare the average number of sensitized long paths running through each detected critical fault (shown as "NLP_{DCF}"), which is calculated using Equation (2), and the average number of sensitized long paths running through each critical fault (shown as "NLP_{TCF}"), which is calculated using Equation (3).

$$NLP_{DCF} = \frac{\sum_i NLP_{CFi}}{N_{DCF}} \tag{2}$$

$$NLP_{TCF} = \frac{\sum_i NLP_{CFi}}{N_{TCF}} \tag{3}$$

where NLP_{CFi} represents the number of sensitized long paths running through detected critical fault i, N_{DCF} represents the number of detected critical faults, N_{TCF} represents the total number of critical faults in the design.

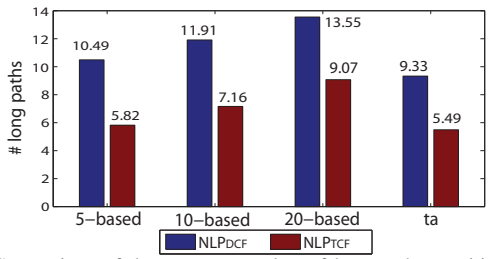

Fig. 7. Comparison of the average number of long paths sensitized through each critical fault for ethernet.

It can be seen from the figure that the number of unique sensitized long paths running through each detected critical fault or each critical fault of our pattern set increases as n increase from 5 to 20. In any case, our pattern set is superior to timing-aware pattern set in detecting critical faults via various long paths, although they detect comparable number of SDDs as shown in Figure 6. Therefore, our pattern set can ensure a more reliable SDD detection capability.

TABLE IV
ATPG AND PATTERN SELECTION RESULTS ON INDUSTRY CIRCUITS.

Circuit		1-detect	10-detect	EMD_3-20	EMD_10-20	ta
Circuit_A	# ori. Pat.	5,456	56,784	14,125	49,137	16,940
	CPU(ATPG)	2.4h	17.5h	8.0h	26.2h	26.3h
	# Sel. Pat.	631	1,542	1,050	1,348	1,099
	CPU(Sel.)	4m19s	1h5m41s	15m33s	58m53s	20m43s
	# SDDs	63,097	80,950	74,755	80,110	73,650
Circuit_B	# ori. Pat.	2,113	15,249	4,456	11,749	5,370
	CPU(ATPG)	1.1h	5.6h	3.6h	6.5h	7.0h
	# Sel. Pat.	593	1,013	737	1,066	1,066
	CPU(Sel.)	22m58s	34m51s	51m1s	28m1s	4m51s
	# SDDs	57,278	67059	61,783	69,125	76,211

D. Experiments on Industry Circuits

In this subsection, we present the experimental results on industry circuits. 1-detect, 10-detect, timing-aware, and embeded multi-detection (EMD) pattern sets are generated as the original pattern repositories for our pattern grading and selection procedure. The EMD technique tries to increase the number of detection of each testable fault without pattern count increase. But it does increase the ATPG runtime [21]. To save CPU runtime, the original pattern sets are generated based on selected timing-critical faults, which were selected with $0.3T$ slack threshold. Table IV shows the pattern selection results on "Circuit_A" and "Circuit_B". The "EMD_3-20" and "EMD_10-20" pattern sets are generated using EMD technique. For example, the "EMD_3-20" is generated to guarantee each testable fault to be detected at least 3 times, but desired to be detected 20 times or more. The pattern selection procedure ensures that the selected patterns can detect all the SDDs detected by the original pattern sets. For instance, both 1-detect pattern set of Circuit_A (5,456 patterns) and its selected patterns (631 patterns) detect the same 63,097 SDDs. It can be seen from the table that, our pattern selection procedure can reduce the pattern count significantly while keeping the original SDD detection efficiency. For example, only 11.6% patterns (631 out of 5,456) selected from the 1-detect pattern set of Circuit_A can detect all the SDDs detected by the original pattern set. For its 10-detect pattern set, only 2.7% patterns (1,542 out of 56,784) are selected. This is because with larger pattern repository, 10-detect has a good chance to include patterns with higher SDD test quality. It can also be seen that 10-detect and EMD pattern sets detect more SDDs than 1-detect pattern set, and sometimes even more than timing-aware pattern set (Circuit_A). For Circuit_B, timing-aware pattern set detects more SDDs than both 10-detect and EMD pattern sets, and consequently performed better than our selected patterns. This is because the SDD detection capability of our selected patterns is bounded by its original pattern repository. However, when comparing the pattern count between our selected patterns and the original timing-aware patterns ("# ori. Pat." in "ta" column), the benefit of our procedure is obvious. When applying our pattern selection procedure to timing-aware pattern set, it can also reduce the pattern count significantly for both circuits. Also, the CPU runtime of our pattern selection procedure is much smaller compared with the ATPG runtime.

After pattern selection, 1-detect top-off ATPG is run on the entire undetected faults in the target clock domain to ensure

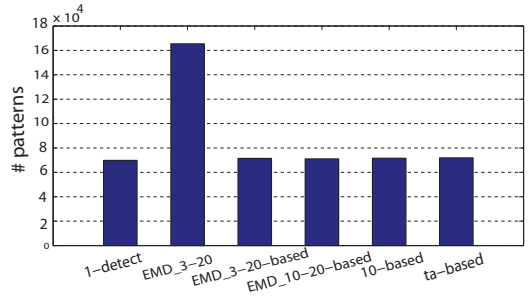

Fig. 8. Pattern count comparison between different pattern sets for Circuit_A. Here "EMD_3-20-based" represents selected patterns from CF-based EMD_3-20 pattern set plus its corresponding top-off patterns.

all the testable faults in this domain can be detected by our final pattern set. Figure 8 shows the pattern count comparison between our final pattern sets (selected + top-off patterns) and 1-detect and EMD_3-20 pattern sets (based on the entire fault list) of Circuit_A. All the pattern sets shown in this figure have the same test coverage (i.e., detect all the testable TDFs). Due to the complexity of runtime and memory, 10-detect, EMD_10-20 and timing-aware ATPG on the entire fault list cannot be finished in a reasonable time. But we would expect them to provide a much larger pattern count than EMD_3-20 pattern set. It can be seen from the figure that our final pattern count, regardless of the pattern repository, is very close to 1-detect pattern set, which is much smaller than EMD_3-20 pattern set. However, with different original pattern repository, the SDD detection efficiency of our final pattern set is varied, as can be seen from Table IV. Please note that our final pattern set may detect a slightly more SDDs than the corresponding selected patterns shown in Table IV, since the top-off patterns may detect some extra SDDs fortuitously.

We also run our procedure on faults crossing clock domains, and the results are shown in Table V. In this experiment, we run EMD_3-20 and timing-aware ATPG on faults crossing domain CLK_1 and CLK_2 of Circuit_B, and apply our pattern selection procedure on both pattern sets. Similar to the experiments in Table IV, the pattern selection procedure ensures the selected patterns can detect all the SDDs detected by the original pattern repository. It can be seen that for this case, EMD_3-20 detects more SDDs than timing-aware pattern set (53,133 vs. 52,824) with slightly smaller pattern count (10,339 vs. 10,479). This is because there are fewer paths running through the crossing-domain faults, and therefore, EMD ATPG has a good chance to detect the faults via long paths runing through them. Note that the performance of both EMD and timing-aware ATPG are all design-dependent. However, in any case, our pattern selection can reduce the pattern count significantly while keeping the SDD detection efficiency of the original pattern repository. The proposed procedure can also be used to evaluate the effectiveness of a pattern set for screening SDDs.

TABLE V
EXPERIMENTAL RESULTS ON CLK_1 TO CLK_2 FOR CIRCUIT_B.

Pat. Set	# Pat.	CPU (ATPG)	# Sel. Pat.	CPU (Sel.)	# SDDs
EMD_3-20	10,339	15.7h	1,248	48m0s	53,133
ta	10,479	56.3h	1,196	45m0s	52,824

V. CONCLUSIONS

In this paper, we have presented an efficient SDD-based pattern grading and selection procedure for screening SDDs. n-detect pattern sets were used as the original pattern repository to take its advantage for SDD detection. Techniques were proposed to reduce the runtime and memory complexity significantly and therefore make the procedure applicable to very large industry circuits. Compared with previous LP-based procedure, this SDD-based method also reduces the number of selected patterns significantly. Experimental results on both academic and industry circuits demonstrate the efficiency of the proposed procedure.

VI. ACKNOWLEDGMENT

The authors would like to thank Dr. Mahmut Yilmaz from Advanced Micro Devices, Inc. for his assistance in developing the tool for path tracing, and Prof. Krishnendu Chakrabarty from Duke University for his valuable suggestions and discussions on the pattern selection procedure.

REFERENCES

[1] J. Savir and S. Patil, "On Broad-Side Delay Test," in *Proc. VLSI 2 Symp. (VTS'94)*, pp. 284-290, 1994.
[2] M. Bushnell and V. Agrawal, "*Essentials of Electronics Testing*", Kluwer Publishers, 2000.
[3] R. Mattiuzzo, D. Appello C. Allsup, "Small Delay Defect Testing," *http://www.tmworld.com/article/CA6660051.html* Test & Measurement World, 2009.
[4] Synopsys Inc., "TetraMAX ATPG, Automatic Test Pattern Generation," Synopsys Datasheet, 2010.
[5] Mentor Graphics, "Tessent FastScan, Advanced Automatic Test Pattern Generation," Silicon Test and Yield Analysis Datasheet, 2009.
[6] M. E. Amyeen, S. Venkataraman, A. Ojha, S. Lee, "Evaluation of the Quality of N-Detect Scan ATPG Patterns on a Processor", *IEEE International Test Conference (ITC'04)*, pp. 669-678, 2004.
[7] Y. Huang, "On N-Detect Pattern Set Optimization," in *Proc. IEEE the 7th Int. Symp. on Quality Electronic Design (ISQED'06)*, 2006.
[8] IWLS 2005 Benchmarks, "http://iwls.org/iwls2005/benchmarks.html".
[9] W. Qiu, J. Wang, D. Walker, D. Reddy, L. Xiang, L. Zhou, W. Shi, and H. Balachandran, "K Longest Paths Per Gate (KLPG) Test Generation for Scan Scan-Based Sequential Circuits," in *Proc. IEEE ITC*, 2004.
[10] N. Ahmed, M. Tehranipoor and V. Jayaram, "Timing-Based Delay Test for Screening Small Delay Defects," *IEEE Design Automation Conf.*, pp. 320-325, 2006.
[11] A. K. Majhi, V. D. Agrawal, J. Jacob, L. M. Patnaik, "Line coverage of path delay faults," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 5, pp. 610-614, 2000.
[12] H. Lee, S. Natarajan, S. Patil, I. Pomeranz, "Selecting High-Quality Delay Tests for Manufacturing Test and Debug," in *Proc. IEEE Int. Symp. on Defect and Fault-Tolerance in VLSI Systems (DFT'06)*, 2006.
[13] S. Goel, N. Devta-Prasanna and R. Turakhia, "Effective and Efficient Test pattern Generation for Small Delay Defects," *IEEE VLSI Test Symposium (VTS'09)*, 2009.
[14] S. Goel, K. Chakrabarty, M. Yilmaz, K. Peng, M. Tehranipoor, "Circuit Topology-Based Test Pattern Generation for Small-Delay Defects," to appear in *IEEE Asian Test Symposium (ATS'10)*, 2010.
[15] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-Pattern Grading and Pattern Selection for Small-Delay Defects," in *Proc. IEEE VLSI Test Symposium (VTS'08)*, 2008.
[16] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Interconnect-Aware and Layout-Oriented Test-Pattern Selection for Small-Delay Defects," in *Proc. Int. Test Conference (ITC'08)*, 2008.
[17] K. Peng, J. Thibodeau, M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "A Novel Hybrid Method for SDD Pattern Grading and Selection," in *Proc. IEEE VLSI Test Symposium (VTS'10)*, 2010.
[18] K. Peng, M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "A Noise-Aware Hybrid Method for SDD Pattern Grading and Selection," to appear in *Proc. IEEE Asian Test Symposium (ATS'10)*, 2010.
[19] K. Peng, M. Yilmaz, M. Tehranipoor, and K. Chakrabarty, "High-Quality Pattern Selection for Screening Small-Delay Defects Considering Process Variations and Crosstalk," in *Proc. IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE'10)*, 2010.
[20] David Salomon, "*Data Compression, The Complete Reference, Forth Edition*", Springer Publishers, 2007.
[21] J. Geuzebroek, E.J. Marinissen, A. Majhi, A. Glowatz, F. Hapke, "*Embedded multi-detect ATPG and Its Effect on the Detection of Unmodeled Defects*", in *Proc. Int. Test Conference (ITC'07)*, pp. 1-10, 2007.