A Novel Hybrid Method for SDD Pattern Grading and Selection

Ke Peng[1], Jason Thibodeau[1], Mahmut Yilmaz[2], Krishnendu Chakrabarty[3], Mohammad Tehranipoor[1]
[1]ECE Department, University of Connecticut
[2]Advanced Micro Devices, Inc., Sunnyvale, California
[3]ECE Department, Duke University

***Abstract*—Small-delay defects (SDDs) have become a major concern in nanometer technology designs. Traditional timing-unaware transition-delay fault (TDF) ATPGs are not efficient in detecting SDDs since they tend to detect delay faults via shorter paths. Timing-aware ATPG tools have been proven to result in significantly large CPU runtime and pattern count. In this paper, we present a hybrid procedure that grades patterns in terms of their effectiveness in detecting SDDs and selects the most effective ones. The grading procedure is performed on a large repository of patterns generated by n-detect TDF ATPG and takes advantage of n-detect capability in detecting a delay fault n times from different paths. 1-detect TDF ATPG is performed after pattern grading and selection to ensure same fault coverage as timing-aware ATPG's is obtained. Experimental results demonstrate that our proposed hybrid method is fast and efficient; it can sensitize a greater number of longer paths with much lower pattern count and CPU runtime compared to a commercial timing-aware ATPG tool.**

I. Introduction

Timing-related defects introduced by the imperfection of manufacturing processes have become major concerns in modern very large scale integration (VLSI) designs [1] [2]. Such defects can cause chip failures by introducing extra delay to the design. The small-delay defect (SDD) is one type of such timing defects. SDDs were not being seriously considered at the higher technology nodes since (i) they only introduced a small amount of extra delay to the design, which seldom resulted in failure of the design with comparable lower frequency and larger timing margins and (ii) they do not introduce many SDDs to the design. However, when technology scales to $65nm$ and below, resulting in an increase in design density and operating frequency, SDD introduction into fabricated designs necessitates consideration of their impact on test quality and in-field reliability [3]. Since a SDD only introduces a small amount of extra delay to the design, it may not necessarily cause a failure if the SDD is cited on a short path with large slack. However, if cited on a long path with small slack, the SDD may cause circuit failure. Therefore, it is commonly recommended to detect the SDDs via the long paths running through the fault sites.

In general, transition-delay fault (TDF) and path-delay fault (PDF) are two prevalent fault models widely used in industry. The traditional TDF automatic test pattern generation (ATPG) methods tend to detect timing faults via short paths making them inefficient for SDDs. The PDF model is superior to TDF in the modeling capacity by addressing distributed defects that affect the entire path. However, it requires a large test pattern set and may also be impossible to detect all the paths in the design under robust condition and the fact that the number of paths in the design increases exponentially with circuit size [2]. As a result, in practice, most timing defects are targeted by the TDF model.

Timing-aware ATPG tools [6] [7] were developed to deal with the deficiencies of traditional timing-unaware ATPGs. 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 [4] [5]. An n-detect ATPG generates patterns by detecting the target faults n times via different paths. In this way, setting a large n will result in an increased probability of sensitizing long paths running through the fault sites. However, its extremely large pattern count makes it impractical for large industry designs.

A. Related Prior Work

In recent years, several techniques have been proposed for screening SDDs among which most attempts have focused on developing algorithms to target a delay fault via longest path. For example, in [8], the authors proposed an as late as possible transition fault (ALAPTF) model to launch one or more transition faults at the fault site as late as possible to detect the faults through the path with least slack, which requires extended CPU runtime compared to traditional ATPGs. A method to generate K longest paths per gate for testing transition faults was proposed in [10]. However, the longest path through a gate may still be a short path for the design, and thus may not be efficient for SDD detection. Furthermore, this method also suffers from high complexity, extended CPU runtime, and pattern count. 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 and generating a slow-to-rise or slow-to-fall transition. However, the shortcomings of PDF model limit its usage in real applications. 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.

Several pattern evaluation and selection techniques have also been proposed for screening SDDs. In [9], the authors presented a method using 1-detect and timing-aware ATPG to detect SDDs with reduced pattern counts. Since 1-detect ATPG has very limited capability in detecting SDDs, the method may suffer from large pattern count generated using timing-aware ATPG. In [13], a static-timing-analysis based method was proposed to generate and select patterns that sensitize long paths. It finds long paths (LP), intermediate paths (IP),

[1]The work of K. Peng, J. Thibodeau and M. Tehranipoor was supported in part by SRC under Contract No. 1587, and by NSF under Grant No. ECCS-0823992. [2,3]The work of M. Yilmaz and K. Chakrabarty was supported in part by SRC under Contract No. 1588 and by NSF under Grant No. ECCS-0823835.

and short paths (SP) to each observation point using a static timing analysis tool. Then IP and SP observation points are masked in the pattern generation procedure to force the ATPG tool to generate patterns for LPs. Next, a pattern selection procedure is applied to ensure the pattern quality.

The output-deviation based method was proposed in [14]. This method defines gate-delay defect probabilities (DDPs) to model delay variations in a design. Gaussian distribution gate delay is assumed and a delay defect probability matrix (DDPM) is assigned to each gate. Then, the signal-transition probabilities are propagated to the outputs to obtain the output deviations, which are used for pattern evaluation and selection. However, in case of a large number of gates along the paths, the calculated output deviation metric can saturate and similar output deviations (close to 1) can be observed for both long and intermediate paths (relative to functional clock period). A similar method was developed in [15] to take into account the interconnect contribution to the total delay of sensitized paths. But it also suffers from the output-deviation saturation problem.

B. Contributions and Paper Organization

Contrary to our previously proposed techniques that is based on output deviation and evaluation and selection of effective SDD patterns from a large repository of patterns, in this paper, we propose a hybrid method based on standard delay format (SDF) timing information to grade and select the most effective patterns for screening SDDs, which has no saturation problem like [14] and [15]. Instead of evaluating the length of all the paths in the design, which is impossible for large industry designs, we only evaluate the sensitized paths of each pattern based on SDF timing, with which we evaluate/grade the pattern; this can save significant CPU runtime. Our contributions in the work include:

1. Developing a hybrid method based on pattern grading and selection from a large repository of patterns and using timing-unaware ATPG.
2. A procedure is developed to identify all the sensitized paths by each TDF pattern.
3. An procedure and a metric are developed to evaluate each TDF pattern in terms of its effectiveness for detecting SDDs.
4. A SDD pattern selection procedure is developed. Our patterns are selected from a large pattern repository, such as n-detect patterns generated using traditional timing-unaware TDF ATPG or randomly generated pattern set. This would make the pattern generation procedure much faster compared with timing-aware ATPG. Our pattern selection procedure will minimize the overlap of sensitized paths between patterns. This can ensure that only the most effective patterns with minimum sensitized path overlap can be selected and can further reduce the selected pattern count.

Our final pattern set ensures the same fault coverage as timing-aware ATPG by using 1-detect ATPG on top of selected patterns from n-detect pattern set. The proposed procedure is fast and effective in terms of CPU runtime, pattern count, and the number of sensitized long paths. Although, our primary objective is to implement our method on n-detect pattern set, implementation of our method on 1-detect and timing-aware pattern sets has shown significant improvement in pattern count and number of sensitized long paths.

The remainder of this paper is organized as follows. Section II will analyze the effectiveness of SDD detection of different pattern sets. Our pattern grading and selection procedure based on SDF is presented in Section III. To validate our procedure, we apply it to several benchmarks and the experimental results are presented in Section IV. Finally, we conclude our work in Section V.

II. SDD Detection Analysis for n-Detect and Timing-Aware ATPGs

The effectiveness of traditional TDF ATPGs for detecting SDDs is questioned since (i) a timing defect can only be detected if the introduced delay is larger than the slack of the affected path and (ii) traditional ATPG tools tend to detect the TDFs through short paths while it is efficient to detect SDDs through the long paths running through the fault sites.

Timing-aware ATPGs [6] [7] use the delay information obtained from static timing analysis (STA) tools to select possible long paths for TDF detection. However, they have been proven to result in a much larger pattern count compared to traditional timing-unaware TDF pattern sets, and require significant CPU runtime, especially for large industry designs. n-detect ATPG can be run for screening SDDs as well. Such ATPG generates patterns detecting the target faults n times, through different paths. With the increase of n, we may have a greater possibility to detect delay defects via the long path running through the fault site. Therefore, n-detect ATPG could be an efficient method for screening SDDs. Furthermore, the n-detect ATPG has proven to require much less CPU runtime when compared with timing-aware ATPG. The major drawback, however, is the very large pattern count, which limits its usage in real applications.

Figure 1 presents the normalized CPU runtime and pattern count of 1-detect, n-detect ($n = 3, 5, 8, 10$) and timing-aware TDF ATPGs for the IWLS benchmark tv80 [16]. The CPU runtime and pattern count are normalized with respect to 1-detect ATPG. It can be seen from the figure that as n increases, the CPU runtime of n-detect timing-unaware ATPG increases. Timing-aware ATPG consumes a much larger CPU runtime compared with timing-unaware ATPGs (over $200X$ of 1-detect ATPG). The pattern count of n-detect ATPG increases almost linearly with n for this benchmark. The timing-aware ATPG results in an even larger pattern count than 10-detect ATPG for this benchmark.

Note that although we use n-detect pattern sets in this work, other pattern sets such as random patterns and timing-aware patterns can also be used as our pattern repository.

III. Pattern Grading and Selection

A. Path Classification and Pattern Grading

Since it is desirable to detect SDDs via long paths running through the fault sites, a technique is needed to target SDDs via long paths and gross delay defects via paths of any length. An SDD is defined as a TDF on a long path. Defining the long path will greatly impact the pattern grading and selection.

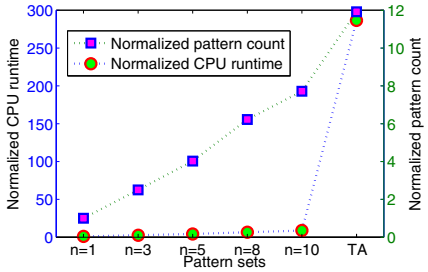

Fig. 1. Normalized CPU runtime and pattern count of different pattern sets for tv80.

Whether a path is long or short is determined by comparing its length to the functional clock period. In general, when the length of a path is close to the clock period, which means it has a small slack, we consider it as a long path. Otherwise, when the path length is short compared with clock period, we consider it as a short path with large slack. Thus, it is necessary to define a threshold (named long path threshold LP_{thr} in this paper) according to the clock period to differentiate the paths to be long paths or short paths.

In this paper, we consider a path to be long if its length is equal or greater than $0.7T$, where T is the clock period. The effectiveness of a TDF pattern is determined by its ability in sensitizing long paths. Note that any other long path threshold can be used in practice as well. For example $LP_{thr} = 0$ means every fault must be targeted via its longest sensitizable path while $LP_{thr} = 0.95T$ means that only critical paths are targeted. The greater number of sensitized long paths, the more effective the pattern is considered. Due to the large pattern count in the original pattern repository, a procedure is needed to select the most effective ones. Our pattern grading and selection procedure is based on this observation.

For each TDF pattern, we run a fault simulation to identify all its detected TDF faults. Bearing such information, we search the topology of the design for the sensitized paths of this pattern. Then we add timing information for each of the sensitized paths based on the corresponding SDF file. With the timing information, we can differentiate the sensitized long paths (LPs) from short paths (SPs). In our work, the paths with length equal or greater than LP_{thr} are referred to as long paths, the remaining paths are defined as short paths. Fig. 2 presents the flow of sensitized path identification and classification. The procedure will provide a complete list of all sensitized paths and their respective length.

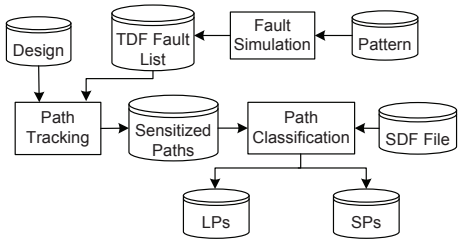

Fig. 2. Sensitized path classification of a pattern.

With the definition of long path threshold, LP_{thr}, we can grade a path by assigning a weight to it according to its path length, and further evaluate a TDF pattern by calculating the weight of all its sensitized long paths. Assume that W_{Pi} is the weight of pattern Pi, N is the total number of sensitized paths of pattern Pi, and W_{pathi} is the weight of i-th sensitized path. The weight of the pattern Pi is calculated by $W_{Pi} = \sum_{i=1}^{N} W_{pathi}$. Here, we assign a weight of 1.0 for each long path ($W_{pathi} = 1$), and a weight of 0.0 for each short path ($W_{pathi} = 0$). Therefore, in this work, the weight of a pattern is equal to the number of its sensitized long paths.

We acknowledge that the definition of pattern weight is open ended, according to the requirements of the application. For example, one can select patterns that target very small delay defects in the design by setting $LP_{thr} = 0.8T$ or $0.9T$. One can also select patterns that will target the SDDs on long paths with first priority and the SDDs on the intermediate paths with second priority. Then, the sensitized paths can be differentiated to long paths (LPs, i.e., the paths that are longer than LP_{thr}), intermediate paths (IPs, i.e., the paths that are between a defined SP_{thr} and LP_{thr}), and short paths (SPs, i.e., the paths that are shorter than SP_{thr}). Then different weights can be applied to LPs, IPs, and SPs. Also, the weight of a pattern is defined as the sum of all its sensitized paths' weights. In this way, the patterns are evaluated according to their sensitized LPs, IPs, and SPs. If the absolute length of a path is a concern, we can define the path weight to be the ratio between the path length and the clock period for pattern evaluation. In any case, the pattern is evaluated based on its sensitized paths, and each path is weighted by its length. Furthermore, we use typical delay values in the SDF file for path evaluation. According to the requirements of the application, we can also use max/min delay values in the SDF file for worst/best case path evaluation. Defining and analyzing thresholds for long paths, intermediate paths, and short paths to further quantify patterns will be considered in our future work.

B. Pattern Selection

From the pattern evaluation procedure in the previous subsection, if a pattern sensitizes a large number of long paths (i.e. it is efficient in detecting SDDs) it will have a large weight. Thus, in our pattern selection procedure, we will select patterns with the largest weights, which ensures that they are most effective in detecting SDDs. It must be noted that many paths in the design can be sensitized by multiple patterns. In this case, if a path has been sensitized by a previously selected pattern, it is not necessary to use it for evaluating the remaining patterns, if we do not want to target the faults on the path multiple times. Our pattern selection procedure will check the overlap of sensitized paths between patterns, and ensures that only the unique sensitized paths of a pattern are involved in the pattern evaluation procedure.

The patterns are first sorted according to their unique weights before selecting the most effective ones. The pattern selection and sorting procedure is shown in Fig. 3. The pattern with the largest weight will be first selected to put on the top of the sorted list. Then the weights of the remaining patterns are re-calculated by excluding paths that have been sensitized by patterns in the sorted list. Then, the pattern with the largest weight in the remaining pattern set is selected to put in the second position of the sorted list. This procedure is repeated to sort the patterns in the n-detect pattern set, and ensure that

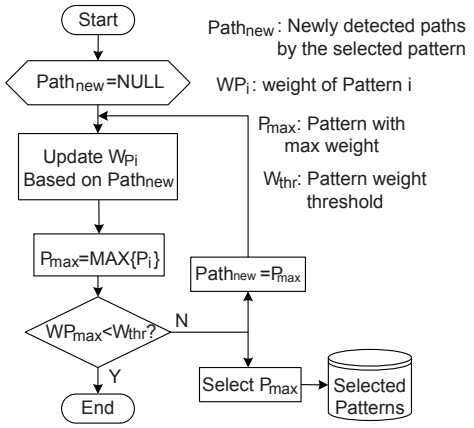

Fig. 3. Selection and sorting procedure for TDF patterns based on their unique weight.

every pattern is evaluated by its sensitized unique paths, which are not sensitized by the previous patterns in the sorted pattern list.

Our unique-path-evaluation method will ensure that there is as little overlap as possible between patterns in terms of sensitized paths. Therefore, it can further reduce the selected pattern count. The pattern-sorting iteration will be stopped when weights of the remaining patterns are 0 (considered as W_{thr} in Fig. 3). According to our simulations, only less than 20% of the patterns in the top of the sorted pattern list can sensitize all the long paths that are sensitized by the n-detect pattern set. This pattern sorting algorithm will return a pattern list with decreasing order according to the test efficiency of the patterns. By just selecting patterns with weights larger than the pattern selection threshold (W_{thr}), the pattern-sorting iteration can be stopped when the largest weight in the remaining patterns is equal to, or smaller than the pattern selection threshold. This can further save additional CPU runtime.

IV. Experimental Results

In this section, we will present experimental results for several IWLS [16] benchmarks. The experiments were performed on a Linux x86 server with 8 processors and 24 GB of available memory. Four IWLS benchmarks were used. The characteristics of these benchmarks are shown in Table I. Note that the data in Table I is obtained from original Verilog RTL code of the benchmark and they may be slightly different after synthesizing the circuit, since the tool may optimize the design according to specified optimization options. Commercial tools were used for circuit synthesis, physical design and pattern generation. In order to get accurate timing information, we extract the post-layout SDF file of the design for pattern evaluation. 180 nm Cadence Generic Standard Cell Library was used for physical design. All patterns are generated using TDF launch-off-capture (LOC) method. After pattern selection, top-off ATPG is run for the undetected faults to ensure that all the detectable faults in the design are detected by our final pattern set and meet the fault coverage requirements. The entire flow for hybrid method is shown in Fig. 4.

Figure 5 presents the relation between the number of selected patterns and unique sensitized long paths by the

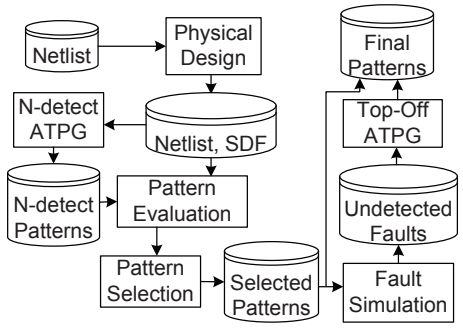

Fig. 4. The entire flow of hybrid method.

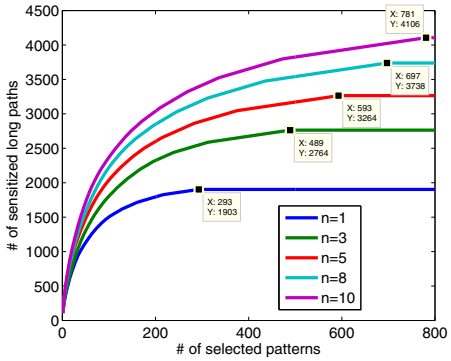

Fig. 5. Relation between number of selected patterns and unique sensitized long paths.

selected pattern sets (selected from 1-detect, 3-detect, 5-detect, 8-detect, and 10-detect pattern sets) for IWLS benchmark usb_funct [16] considering $LP_{thr} = 0.7T$. It can be seen that only 781 (6.3% of the total pattern count 12,345) patterns are needed to detect all the long paths sensitized by the 10-detect pattern set. 7.0% (697 out of 9958), 9.3% (593 out of 6397), 12.2% (489 out of 4006), and 18.7% (293 out of 1569) patterns are needed to detect all the long paths sensitized by 8-detect, 5-detect, 3-detect, and 1-detect pattern sets, respectively.

After setting a long path threshold LP_{thr}, we can evaluate each sensitized path and further each pattern in the pattern set. A pattern weight threshold is needed in our pattern selection procedure to terminate the procedure and ensure only the most effective patterns can be selected. In our experiments, we set the pattern weight threshold $W_{thr} = 1$. Note that we assigned a weight 1.0 for each long path and a weight 0.0 for each short path; this pattern weight threshold will ensure that only the patterns contributing to unique long paths sensitization can be selected. Changing this threshold may slightly impact the number of selected patterns. Setting a very small W_{thr} will select more patterns from the n-detect pattern set while setting a high W_{thr} will select fewer patterns and more top-off patterns will be generated.

In these experiments, we evaluate and select patterns from different pattern sets (i.e., different n-detect and timing-aware pattern sets). Table II presents the number of patterns for these pattern sets (shown at "original"), and the number of selected patterns (shown at "selected") as well as top-off patterns (shown at "topoff.") based on these pattern sets. It is clear that as n increases, the number of original patterns increases for each benchmark. Furthermore, the pattern count of timing-aware ATPG is larger and is sometimes even larger than 10-detect ATPG pattern set, as is the case for tv80 and

TABLE I
BENCHMARKS CHARACTERISTICS.

Benchmark	# of gates	# of FFs	# of Total standard cells
tv80	6,802	359	7,161
mem_ctrl	10,357	1,083	11,440
systemcaes	7,289	670	7,959
usb_funct	11,262	1,746	12,808

TABLE II
NUMBER OF PATTERNS FOR DIFFERENT PATTERN SETS.

Benchmark		n=1	n=3	n=5	n=8	n=10	ta
tv80	original	1,435	3,589	5,775	8,931	11,072	12,931
	selected	50	108	133	196	193	193
	topoff	1,332	1,260	1,213	1,142	1,174	1,173
	total	1,382	1,368	1,346	1,338	1,367	1,366
mem_ ctrl	original	1,595	4,187	6,863	10,740	13,142	8,769
	selected	27	96	170	269	309	156
	topoff	1,404	1,309	1,212	1,147	1,109	1,252
	total	1,431	1,405	1,382	1,416	1,418	1,408
syste- mcaes	original	591	1,318	2,005	3,028	3,686	6,166
	selected	304	612	885	1,206	1,430	1,918
	topoff	234	78	26	8	4	25
	total	538	690	911	1,214	1,434	1,943
usb_ funct	original	1,569	4,006	6,397	9,958	12,345	11,530
	selected	293	489	593	697	781	788
	topoff	1,180	1,016	929	823	767	1,006
	total	1,473	1,505	1,522	1,520	1,548	1,794

TABLE IV
NUMBER OF DETECTED SDDS FOR DIFFERENT PATTERN SETS.

Benchmark		n=1	n=3	n=5	n=8	n=10	ta
tv80	original	1,190	2,012	2,005	2,591	2,558	3,052
	selected	1,190	2,012	2,005	2,591	2,558	3,052
	topoff	362	170	164	17	52	166
	total	1,552	2,182	2,169	2,608	2,610	3,218
mem_ ctrl	original	1,148	1,918	2,102	2,358	2,856	2,692
	selected	1,148	1,918	2,102	2,358	2,856	2,692
	topoff	561	139	263	90	32	122
	total	1,709	2,057	2,365	2,448	2,888	2,814
syste- mcaes	original	4,073	4,782	5,687	5,909	6,321	6,672
	selected	4,073	4,782	5,687	5,909	6,321	6,672
	topoff	537	24	5	0	0	0
	total	4,610	4,806	5,692	5,909	6,321	6,672
usb_ funct	original	5,362	6,317	6,515	6,934	7,105	7,570
	selected	5,362	6,317	6,515	6,934	7,105	7,570
	topoff	590	185	86	10	6	6
	total	5,952	6,502	6,601	6,944	7,111	7,576

TABLE V
CPU RUNTIME OF DIFFERENT PATTERN SETS FOR USB_FUNCT BENCHMARK.

pat. sets	1-detect	3-detect	5-detect	8-detect	10-detect	ta
# pat.	1,569	4,006	6,397	9,958	12,345	11,530
CPU(ATPG)	15s	37s	57s	1m28s	1m49s	21m08s
CPU(HB+I/O)	4m19s	11m25s	18m30s	29m40s	38m30s	35m2s
CPU(HB)	56s	2m24s	3m51s	6m5s	7m49s	7m4s

systemcaes. For tv80 and mem_ctrl benchmarks, our final pattern set (shown at "total"), i.e., the selected patterns plus top-off patterns, is independent of n and similar in terms of pattern count. The pattern count, in fact, is even smaller than 1-detect pattern set, except for systemcaes. For systemcaes and usb_funct benchmarks, the final pattern set varies more for different pattern repositories, especially for the results obtained from timing-aware pattern sets. Other notable observations are: (i) The "total" number of patterns when $n = 1$ is smaller than 1-detect pattern set. Therefore, we can use this procedure to reduce the pattern count of 1-detect pattern set without any fault coverage penalty. (ii) The "total" number of patterns is considerable smaller than n-detect (shown at "original") or timing-aware (shown as "ta") ATPG pattern sets. Therefore, our procedure can reduce the pattern count of these pattern sets significantly while maintaining or even increasing their SDD-detection effectiveness.

The number of sensitized unique long paths and detected unique SDDs of these pattern sets are shown in Table III and Table IV, respectively, when $LP_{thr} = 0.7T$. In order to make a fair comparison, we set the slack to be $0.3T$ when running timing-aware ATPG. From these two tables, it can be seen that for each benchmark, as n increases, the number of sensitized unique long paths and detected unique SDDs increase, except for tv80 where the 8-detect pattern set performs slightly better than 10-detect pattern set.

Since our selected pattern set is a subset of the original pattern repository, the number of its sensitized long paths is upper bounded by the original pattern repository. It cannot sensitize more long paths than the original pattern repository. However, due to our W_{thr} definition, it sensitizes all the long paths of the original pattern repository. It is also interesting to note that: (i) Our hybrid method is able to increase the number of sensitized long paths for 1-detect pattern set (e.g., 218 in "original" and 314 in "total" for tv80 in Table III) with a smaller pattern count (1,435 vs. 1,382). (ii) It also improved the results obtained from timing-aware pattern set (e.g., 695 in "original" and 772 in "total" for tv80 as in Table III) with much smaller pattern count (12,931 vs. 1,366). This is because the top-off pattern set is helpful in incidental detection of some long paths. Therefore, our final pattern set can sensitize more long paths than the original n-detect and in some cases larger than timing-aware pattern sets.

Table V presents the CPU runtime for implementing our hybrid method (shown as "CPU (HB)") on different pattern sets for usb_funct benchmark. Row 2 presents the number of patterns generated using n-detect and timing-aware ATPGs, and Row 3 presents their respective CPU runtime. From the table, it can be seen that the CPU runtime of n-detect timing-unaware ATPG is small and negligible even though as n increases. However, timing-aware ATPG needs much more CPU runtime compared with timing-unaware ATPGs (Row 3). Row 4 and Row 5 present the CPU runtime of our hybrid method with and without file input/output (I/O) operation (shown as "CPU(HB+I/O)" and "CPU(HB)"), respectively. When n is small (n=1, 3, 5, and 8), the total CPU runtime of our procedure (Row 4) is lower CPU compared with the timing-aware ATPG for this benchmark. For a larger n (n=10), our procedure consumes a larger CPU runtime compared with the timing-aware ATPG. However, running our procedure on

TABLE III
NUMBER OF SENSITIZED LONG PATHS FOR DIFFERENT PATTERN SETS.

Benchmark		n=1	n=3	n=5	n=8	n=10	ta
tv80	original	218	428	431	724	633	695
	selected	218	428	431	724	633	695
	topoff	96	69	114	37	74	77
	total	314	497	545	761	707	772
mem_ ctrl	original	208	601	1,117	1,825	2,053	1,082
	selected	208	601	1,117	1,825	2,053	1,082
	topoff	285	137	210	91	125	241
	total	493	738	1,327	1,916	2,178	1,323
syste- mcaes	original	945	2,049	2,793	3,775	4,520	6,135
	selected	945	2,049	2,793	3,775	4,520	6,135
	topoff	379	72	24	11	1	26
	total	1,324	2,121	2,817	3,786	4,521	6,161
usb_ funct	original	1,903	2,764	3,264	3,738	4,106	4,338
	selected	1,903	2,764	3,264	3,738	4,106	4,338
	topoff	593	206	154	73	79	98
	total	2,496	2,970	3,418	3,811	4,185	4,436

TABLE VI
NUMBER OF UNIQUE SENSITIZED LONG PATHS, NUMBER OF PATTERNS FOR TV80 WITH DIFFERENT LONG PATH THRESHOLD LP_{thr}.

tv80	$LP_{thr} = 0.5T$		$LP_{thr} = 0.6T$		$LP_{thr} = 0.7T$		$LP_{thr} = 0.8T$		$LP_{thr} = 0.9T$	
10-detect	# LPs	# patterns	# LPs	# patterns	# LPs	# patterns	# LPs	# patterns	# LPs	# patterns
original	12,368	11,072	4,086	11,072	633	11,072	28	11,072	13	11,072
selected	12,368	1,761	4,086	845	633	193	28	8	13	2
topoff	62	306	65	654	74	1,174	8	1,374	0	1,370
total	12,430	2,067	4,151	1,499	707	1,367	36	1,382	13	1,372

larger n will result in a larger number of sensitized long paths sensitization with a small increase in the final pattern count. However, by comparing Row 4 and Row 5, it can be seen that most of the CPU time are consumed by file input/and output operation. This is because our procedure has to deal with the fault list of each pattern. Therefore, if the proposed method is integrated into an ATPG tool, it will be much faster than timing-aware ATPG even based on 10-detect pattern set, since there will be no need for file I/O operation. Furthermore, it does not seem fair to compare CPU runtime between our hybrid method and the commercial timing-aware ATPG tool, since it is a comparison between an experimental non-optimized codes and the highly-optimized commercial tool. We believe that our method's CPU runtime can be further reduced by better programming, optimizing the data structures and algorithms.

As discussed earlier, the long path threshold LP_{thr} is an important parameter for our procedure. If the long path threshold changes, the number of sensitized long paths of will change as well. This will impact the number of selected patterns, and 1 the number of top-off patterns. Table VI presents the experimental results with different path threshold ($LP_{thr} = 0.5T$, $0.6T$, $0.7T$, $0.8T$, and $0.9T$) for 10-detect pattern set of tv80 benchmark. It is clear that as LP_{thr} increases from $0.5T$ to $0.9T$, (i) the number of sensitized unique long paths decreases significantly (from 12,368 to 13) even though the original pattern set is same (shown in Row 2). (ii) the number of selected patterns also decreases significantly since fewer patterns can sensitize the extremely long paths specified by the long path threshold. Due to our W_{thr} definition, the selected patterns will sensitize all the long paths specified by the long path threshold (shown in Row 3). (iii) As the number of selected patterns decreases, the number of top-off patterns increases to meet the fault coverage requirement. However, the number of sensitized long paths of top-off pattern set may not necessarily increase due to the increase of LP_{thr} (shown in Row 4). (iv) the final pattern sets does not change much except when $LP_{thr} = 0.5T$, where a lot of patterns from the original 10-detect pattern set are selected because of the small LP_{thr}. The number of sensitized unique long paths decreases significantly due to the facts that with a larger LP_{thr}, the total number of long paths in the design decreases and only fewer patterns (e.g., 28 when $LP_{thr} = 0.8T$ and 13 when $LP_{thr} = 0.9T$) are selected. In this way, most patterns in the final pattern set (shown as "total" in Row 5) are top-off patterns, which are generated using timing-unaware 1-detect ATPG.

Currently, the main problem of this procedure is its required disk space, since it reports sensitized paths based on the transition delay fault lists, which may take a large disk space for industry designs. This problem can be solved by elaborating the data sources, like parsing VCD (Value Change Dump) files for sensitized paths, etc.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented an efficient pattern grading and selection procedure for screening SDDs using SDF timing information. n-detect pattern sets were used as the original pattern repository for our pattern selection. Our procedure takes advantage of n-detect ATPG for SDD detection, and reduces the pattern count significantly by selecting the most effective patterns in the pattern repository. The method was implemented on several IWLS benchmarks and the experimental results demonstrate the efficiency of our 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] 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.

[5] Y. Huang, "On N-Detect Pattern Set Optimization," in *Proc. IEEE the 7th International Symposium on Quality Electronic Design (ISQED'06)*, 2006.

[6] Synopsys Inc., "SOLD Y-2007, Vol. 1-3," Synopsys Inc.,ct., 2007.

[7] Mentor Graphics, "Understanding how to run timing-aware ATPG," Application Note,2006.

[8] P. Gupta, and M. S. Hsiao, "ALAPTF: A new transition fault model and the ATPG algorithm," in *Proc. Int. Test Conf. (ITC'04)*, 2004.

[9] 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.

[10] 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*, pp. 223-231, 2004.

[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 International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT'06)*, 2006.

[13] N. Ahmed, M. Tehranipoor and V. Jayaram, "Timing-Based Delay Test for Screening Small Delay Defects," *IEEE Design Automation Conf.*, pp. 320-325, 2006.

[14] 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.

[15] 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.

[16] IWLS 2005 Benchmarks, "http://iwls.org/iwls2005/benchmarks.html".

[17] ISCAS 89 Benchmarks, "http://www.fm.vslib.cz/ kes/asic/iscas/".