Seed Selection in LFSR-Reseeding-Based Test Compression for the Detection of Small-Delay Defects*

Mahmut Yilmaz[†‡] and Krishnendu Chakrabarty[‡]

[†]Design for Test Group
Advanced Micro Devices
1 AMD Pl. Sunnyvale, CA 94085, USA

[‡]Dept. Electrical and Computer Engineering
Duke University
Durham, NC 27708, USA

Abstract

Test data volume and test application time are major concerns for large industrial circuits. In recent years, many compression techniques have been proposed and evaluated using industrial designs. However, these methods do not target sequence- or timing-dependent failures while compressing the test patterns. Timing-related failures in high-performance integrated circuits are now increasingly dominated by small-delay defects (SDDs). We present a SDD-aware seed-selection technique for LFSR-reseeding-based test compression. Experimental results show that significant test-pattern-quality increase can be achieved when seeds are selected to target SDDs.

I. Introduction

The complexity of today's integrated circuits and shrinking process technologies are leading to prohibitively high test data volumes. As a result, the 2007 ITRS document predicts that the test data volume for integrated circuits will be as much as 38 times larger and the test application time will be about 17 times larger in 2015 than today [1]. Test-data compression is therefore essential to reduce test-data volume and testing time.

It has also been shown recently that test patterns for sequence- and timing-dependent faults are more important for newer technologies [2]. This is mainly because very deep sub-micron (VDSM) technologies are susceptible to process variations, crosstalk noise, power-supply noise, and defects such as resistive shorts and opens. These defects typically manifest themselves as small delay variations for the circuit components where they occur. These delay variations are referred to in the literature as small-delay defects (SDDs) [3]–[6]. Testing for such delay defects leads to high test-data volume; for example, the test-data volume for delay faults is 2-5 times higher than that for stuck-at faults [7].

Although the delay introduced by each SDD is small, the overall impact can be significant if that path is critical, has low slack, or includes many SDDs. The overall delay of the path may become larger than the clock period, causing circuit failure or temporarily incorrect results. As a result, the detection of SDDs typically requires fault excitation through shortest-slack paths.

SDDs were first alluded to in [8]. The traditional transition delay-fault model has often been questioned because of its tendency to excite short paths [3], [4]. Therefore, most research on SDD detection has been aimed at finding the longest paths in a circuit [3], [9], [10]. Due to the growing interest in SDDs, commercial timing-aware ATPG tools were introduced recently, e.g., new versions of Mentor Graphics FastScan, Cadence Encounter Test, and Synopsys TetraMax tools. A problem with such timing-aware ATPG tools lies in excessive CPU times [11] and prohibitively high pattern counts [4]. Test-data compression can therefore help alleviate the problem of high test-data volume for SDDs.

Test compressions techniques have been extensively studied to reduce test-data volume [12]–[16]. Many of these compression techniques rely on the use of a linear-feedback shift register (LFSR) or an XOR network [12], [13]. These methods rely on the fact that test cubes contain very few specified bits. However, these techniques are typically tailored towards the embedding of test cubes for stuck-at faults, and they do not target sequence- and timing-dependent faults. Typically, unspecified bits in a generated seed are randomly filled without further evaluation of the quality of generated test patterns for the detection of timing-related defects. Therefore, new test-data compression methods are required to reduce test-data volume while targeting SDDs.

LFSR reseeding has long been recognized as an efficient test compression technique [13], [17], [18]. A test cube can be encoded with high probability into a compact seed of typical length S_{max} + 20, where S_{max} equals the number of specified bits in the test cube [17]. A seed can be computed for each test cube by solving a system of linear equations based on the feedback polynomial of the LFSR. The solution space for the system of linear equations is quite large, especially for test cubes with few specified bits. All patterns derived from the possible solutions (LFSR seeds) cover the original test cube. However, these patterns differ from each other in their ability to detect SDDs. Therefore, it is important to select LFSR seeds that can be used to target a large number of defects. Moreover, if more effective seeds are loaded first into the LFSR, a steeper coverage ramp-up can be obtained.

The choice of LFSR seeds for compression methods is either random, e.g., as an outcome of Gauss-Jordan Elimination, or seed selection is designed for better seed compression [14]. These methods do not target the coverage of SDDs. To enhance the effectiveness of LFSR reseeding for process variations and the resulting SDDs, new techniques are needed for pattern modeling and seed selection. While recent work has also focused on seed selection for increasing defect coverage [19], it is not applicable to SDDs since it only evaluates the effectiveness of single patterns, instead of pattern-pairs and long-path sensitization that are required for SDD detection.

*This work was supported in part by SRC under Contract no. 1588 and by NSF under Grant no. ECCS-0823835.

978-3-9810801-5-5/DATE09 © 2009 EDAA

TABLE I
EXAMPLE DDPM A 2-INPUT OR GATE

		Initial Input State			
	prob	**00**	**01**	**10**	**11**
Inputs	**IN0**	0.21	0	0.5	0.11
	IN1	0.12	0.20	0	

In this paper, we use an LFSR-reseeding-based test compression technique to detect SDDs by selecting the best LFSR seeds, and by ensuring that the resulting patterns sensitize more long paths in the circuit. The proposed method is based on the use of the output-deviations metric for delay faults, which was recently introduced in [11]. As in [11], the output deviation measure is used as a surrogate coverage-metric for SDDs. The seeds are selected to ensure high output deviations and high coverage of least-slack (long) paths. Simulation results for the IWLS 2005 benchmark circuits [20] show that compared to several baseline seed-selection methods, the proposed method leads to patterns that provide higher coverage of long paths. In addition, the selected patterns also provide steeper ramp-up of long-path coverage as well as higher coverage of injected delay-defects.

The remainder of the paper is organized as follows. Section II describes the output-deviations metric. Section III describes the proposed seed-selection procedure. In Section IV, we present experimental results for the IWLS 2005 benchmarks. Section V concludes the paper.

II. OVERVIEW OF OUTPUT DEVIATIONS

The concept of gate-delay defect probabilities (DDPs) and signal-transition probabilities were introduced in [11]. These probabilities extend the notion of confidence levels, defined in [21] for a single pattern, to pattern-pairs.

In [11], DDPs were assigned to the gates in a netlist. DDPs for a gate are provided in the form of a matrix called the Delay Defect Probability Matrix (DDPM). An example of a DDPM (with entries chosen arbitrarily) for a 2-input OR gate is shown in Table I. The rows in the matrix correspond to each input port of the gate and the columns correspond to the initial input state during a transition. Each entry denotes the probability that the corresponding $L \rightarrow H$ (rising) or $H \rightarrow L$ (falling) output transition is delayed beyond a threshold. For instance, the entry in the first row and the third column for the DDPM in Table I shows that there is 50% probability that there will be a delay defect because of the transition on `IN0` when the output makes a $H \rightarrow L$ transition starting with the initial input state of "`10`".

For an N-input gate, the DDPM consists of $N \cdot 2^N$ entries, each holding one non-zero probability value. If the gate has more than one output, each output of the gate has a different DDPM, which depends on the inputs affecting the output. Note that the DDP is 0 if the corresponding final input state cannot provide the expected output transition.

We next discuss how a DDPM is generated. Each entry in DDPM indicates the probability that the delay of the gate is more than a predetermined value, i.e., the *critical delay value* (T_{CRT}). Given the probability density function (pdf) of a delay distribution, the DDP is calculated as:

$$DDP = Prob(x > T_{CRT}) = \int_{T_{CRT}}^{\infty} pdf(x)\, dx \qquad (1)$$

For instance, if we assume a Gaussian delay distribution for all gates (with mean μ) and set the critical delay value to $\mu + X$ ps, each DDP entry can be calculated by replacing T_{CRT} with $\mu + X$ and using a Gaussian pdf. Note that the delay for each input-to-output transition delay may have a different mean (μ) and standard deviation (σ). The delay distribution can be obtained in different ways: (i) Using the delay information provided by the Standard Delay Format (SDF) file; (ii) Using slow, nominal, and fast process corner transistor models; (iii) Simulating process variations. In the third method, employed here and in [11], transistor parameters affecting the process variation and the limits of the process variation (3σ) are first determined. Monte Carlo simulations are next run for each library gate under different capacitive loading and input slew rate conditions. Once the distributions are found for library gates, depending on the layout, the delay distributions for each individual gate can be updated. T_{CRT} can now be appropriately set to compute the DDPM entries.

Since pattern pairs are required to detect TDFs, there can be a transition on each net of the circuit for every pattern-pair. If we assume that there are only two possible logic values for a net, i.e., `LOW` (L) and `HIGH` (H), the possible signal transitions are $L \rightarrow L$, $L \rightarrow H$, $H \rightarrow L$, and $H \rightarrow H$. Each of these transitions has a corresponding probability, denoted by $P_{L \rightarrow L}$, $P_{L \rightarrow H}$, $P_{H \rightarrow L}$, and $P_{H \rightarrow H}$, respectively: $< P_{L \rightarrow L}, P_{L \rightarrow H}, P_{H \rightarrow L}, P_{H \rightarrow H} >$. Note that a $L \rightarrow L$ or $H \rightarrow H$ implies that no transition occurs.

The nets that are directly connected to the test-application points are called *initialization nets* (INs). These nets have one of the signal-transition probabilities, corresponding to the applied transition test pattern, equal to 1. All the other signal-transition probabilities for INs are set to 0. When signals are propagated through several levels of gates, the signal-transition probabilities can be computed using the DDPM of the gates.

In [11], several rules were presented for the propagation of signal-transition probabilities. These rules allow us to compute deviations for all the nets in the circuit in an efficient, one-pass, feed-forward manner. A key advantage of this approach is that output deviations can be used to compare path lengths, without the need for explicit path enumeration. Note that path enumeration (adopted in previous work such as [3]–[5]) is a complex and time-consuming procedure. As in the case of path delays, the deviations for nets also increase as the signal propagates through a sensitized path [11].

Example 1: Fig. 1 shows signal-transition probabilities and their propagation for a simple circuit without considering interconnect delays and their associated variations. The test stimuli and the expected fault-free transitions on each net are shown in dark boxes. The calculated signal-transition probabilities are shown in angled brackets ($\langle ... \rangle$). The DDPMs of the gates used in this circuit are given in Tables II and I. The entries in Tables II and I are chosen arbitrarily. A depth-first procedure is used to compute signal-transition probabilities for large circuits. If the number of test

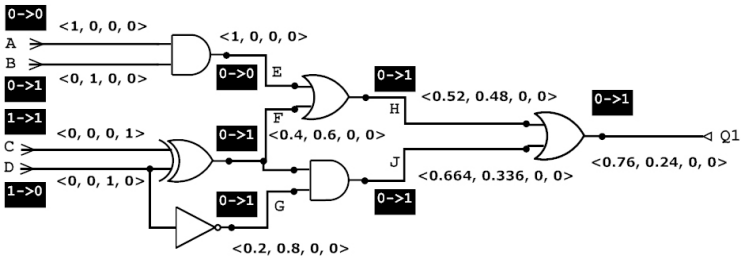


Fig. 1. Propagation of signal-transition probabilities.

TABLE II
EXAMPLE DDPM FOR THE BASIC LOGIC GATES IN FIG. 1

		Initial Input State			
AND	**prob**	**00**	**01**	**10**	**11**
Inputs	**IN0**	0.3	0.3	0	0.2
	IN1		0	0.2	0.3
XOR	**prob**	**00**	**01**	**10**	**11**
Inputs	**IN0**	0.3	0.4	0.1	0.2
	IN1	0.3	0.4	0.2	0.4
INV	**prob**	**0**	**1**		
Inpus	**IN0**	0.2	0.2		

patterns is k and the number of nets in the circuit is N, the worst-case time-complexity of the algorithm is O(kN). However, since the calculation for each pattern is independent of other patterns, the algorithm can easily be made multi-threaded. In this case, if the number of threads is T, the complexity of the algorithm is reduced to O(kN/T).

III. SEED SELECTION

In this section, we describe how to use output deviations to select high-quality seeds for the LFSR. As stated in Section I, the solution of the system of linear equations in any LFSR reseeding scheme typically leads to partially-specified seeds, hence there are many choices for selecting a seed for a given test cube.

A high-level description of the LFSR seed-selection algorithm is shown in Fig. 2. Given a set of test cubes T, we generate N_s seeds for each test cube. Then, each test cube t_p is expanded into a fully-specified test pattern t_i. The seeds and their corresponding patterns are stored in lists S and V, respectively (lines 5-10). In the next stage, deviation computation is done for the patterns in V (line 11).

Procedure: Find Seeds $R(T, N_s, N_p)$
1: list $R[|T|]$;
2: **for all** test cube t_p, $p = 1, 2, ..., |T|$ **do**
3: list $S[N_s]$, $V[N_s]$;
4: **for** $i = 1$ to N_s **do**
5: generate a new seed $s_{p,i}$;
6: expand seed $s_{p,i}$ into test pattern $v_{p,i}$;
7: add $s_{p,i}$ to S, add $v_{p,i}$ to V;
8: **end for**
9: D_p = Compute_Deviations(S,V,N_p);
10: R_p = $max(D_p)$; add $S[R_p]$ to R;
11: **end for**
12: return R;

Fig. 2. Procedure to select LFSR seeds.

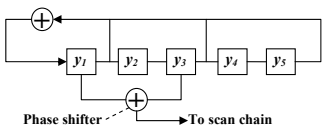
Procedure: Compute Deviations $D(S, V, N_p)$
1: **for all** observation point PO_j, $j = 1, 2, ..$ **do**
2: create list $EFF_j[N_p]$
3: **end for**
4: Max_Dev = 0;
5: **for all** test pattern t_i, $i = 1, 2, ..., N_s$ **do**
6: compute output deviations;
7: **for all** observation point PO_j, $j = 1, 2, ..$ **do**
8: Dev = deviation of PO_j;
9: **if** $Dev >$ Max_Dev **then** Max_Dev = Dev;
10: **if** $Dev > 0.5 \cdot$Max_Dev **then**
11: **if** EFF_j includes Dev **then** Next observation point;
12: **if** EFF_j is not full **then**
13: add t_i and Dev to EFF_j;
14: **else if** $Dev > min(EFF_j)$ **then**
15: remove $min(EFF_j)$, add t_i and Dev to EFF_j;
16: **end if**
17: **end if**
18: **end for**
19: **end for**
20: list $D[N_s]$ = all 0s;
21: **for all** test pattern t_i, $i = 1, 2, ..., N_s$ **do**
22: **for all** observation point PO_j, $j = 1, 2, ..$ **do**
23: **if** EFF_j includes t_i **then** increment $D[i]$;
24: **end for**
25: **end for**
26: return D;

Fig. 3. Procedure to compute output deviations.

The deviation-computation algorithm is described in Fig. 3. Each test observation point PO_j, we keep a list of N_p most effective patterns in EFF_j (lines 1-3). The patterns in EFF_j are the best unique-pattern candidates for exciting a long path through PO_j. During deviation computation, no pattern (t_i) is added to EFF_j if the output deviation at PO_j is smaller than half of the maximum instantaneous output deviation (line 10). If the output deviation is larger than this limit, we first check whether we have added a pattern to EFF_j with the same deviation (line 11). It is very unlikely that two different patterns will create the same output deviation on PO_j while exciting different non-redundant paths. Since we want a higher topological path-coverage, we skip these cases (line 11). If we observed a unique deviation on PO_j, we first check whether EFF_j is full (already includes N_p patterns) or not (line 12). Pattern t_i is added to EFF_j along with its deviation if EFF_j is not full or if t_i has a larger deviation than the minimum deviation stored in EFF_j (lines 12-16). The effectiveness of a pattern is measured by the number of occurrences of this pattern in EFF_j for all values of j. The list of pattern effectiveness is generated after the deviation computation (lines 20-25).

In the next step (returning to Fig. 3), we select the most effective seed (line 12) and move to the next test cube. Once all the seeds are specified, the final fully-specified patterns are also sorted on the basis of the pattern effectiveness calculated during the deviation computation. In this way, the patterns that cover wider range of long paths are pushed to the top of the pattern list.

Example 2: Fig. 4 shows an external-XOR LFSR with feedback polynomial x^5+x^3+x+1, and a 1-stage phase shifter. For the test cube 101xxxxx (the leftmost bit '1' is loaded into the first scan cell that is next to the scan out pin), we can obtain a system of linear



(a)

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b)

Fig. 4. (a) An LFSR and phase shifter; (b) State transition matrix of the LFSR.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

(a) System of linear equations

$$\left(\begin{array}{\overbrace{ccc}^{Pivots} \overbrace{cc}^{Free\ Variables} | c} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right)$$

(b) Gauss-Jordan elimination

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = y_4 \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + y_5 \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

(c) Solution space

Fig. 5. Example to illustrate the solution for system of linear equations.

equations as shown in Fig. 5(a). Gauss-Jordan Elimination can be used to transform a set of columns in A into an identity matrix (these columns are referred to as pivots) while the remaining columns are free-variables, as shown in Fig. 5(b). The set of solutions for the pivots can be represented as a linear combination of the free-variables, as shown in Fig. 5(c). To obtain multiple seeds for each test cube, we first make random assignments to free-variables and then compute pivots.

If we assume that $N_s = 2$, we can randomly assign bit values to y_4 and y_5 to find 2 different seeds: If $y_4 = 1$ and $y_5 = 1$, we find $(y_1, y_2, y_3) = (0, 0, 1)$. Similarly, if $y_4 = 0$ and $y_5 = 0$, we find $(y_1, y_2, y_3) = (1, 1, 0)$. Once we find the complete set of seeds, we can start the deviation computation for the expanded patterns. Assume that we set $N_p = 1$ and the circuit under test has 20 observation points. If seed-1 creates largest deviation for 15 observation points, and seed-2 creates largest deviation for 5 observation points, we select seed-1, i.e., $(y_1, y_2, y_3, y_4, y_5) = (0, 0, 1, 1, 1)$. Note that N_p is smaller than the number of candidate seeds.

IV. Experimental Results

In this section, we present experimental results obtained for the IWLS 2005 benchmark circuits. We do not consider the ISCAS benchmarks because these circuits are small and it is easier for an ATPG tool to excite all long paths with a small number of patterns. We first provide details of the experimental set-up. After that, we present the simulation results.

A. Experimental Set-up

All experiments were performed on a pool of state-of-the-art servers running Linux. The program to generate seeds, compute output deviations, and sort patterns was implemented using C++. A commercial tool was used to perform Verilog netlist synthesis and scan insertion for the IWLS benchmark circuits, which are available in Verilog RTL format [20]. Benchmark statistics are shown in Table III. We used a commercial ATPG tool to generate transition-delay fault (TDF) test cubes for these circuits.

TABLE III
Benchmark statistics

	# I/O	# flip-flops	# library cells	# specified bits (max)
wb_dma	1969	881	6783	52
tv80	763	359	7435	120
mem_ctrl	2537	1138	11119	129
usb_funct	3740	1766	17495	83
aes_core	1472	554	17636	169
ac97_ctrl	4682	2289	21708	42
dma	5274	2197	27236	287
wb_conmax	4182	818	34690	99
pci_bridge32	8166	3677	36647	88
ethernet	21294	10545	125331	76
vga_lcd	196	17102	204810	71

The ATPG tool was forced to generate launch-on-capture (LOC) transition fault patterns. The primary-input change during capture cycles and the observation of primary outputs was prevented in order to simulate realistic test environments. The path delays were calculated using an in-house dynamic path-timing simulator.

B. Generating DDPMs for Gate Instances

DDPM of gate instances were generated by running 200 Monte Carlo (MC) simulations on each gate, for all possible input signal transitions. 180nm process technology parameters are used for HSpice simulations. MC simulations were run using the following realistic process-variation parameters for a Gaussian distribution: *(i)* Transistor gate length $L : 3\sigma = 10\%$; *(ii)* threshold voltage $V_{TH} : 3\sigma = 30\%$; *(iii)* gate-oxide thickness $t_{OX} : 3\sigma = 3\%$. We did not model the net delays, and it is left for future work.

C. Results

We configured the size of the LFSR depending on the maximum number of specified bits for each benchmark (Table III). The size of the LFSR is set to $S_{max} + 20$ where S_{max} is the maximum number of specified bits in the test cubes of the corresponding benchmark [17].

We set the number of seeds per test cube (N_s) to 32 for all benchmarks. These seeds were randomly selected among the candidate seeds for the corresponding test cube. The resulting fully-specified TDF test patterns were used to compute output deviations and to find the most effective seed, as described in Section III. The size of the effective pattern-list per observation point (N_p) is set to 5. In addition to output-deviations-based seed selection, we also implemented the filling of unspecified bits of the seeds with "0" (fill-0), with "1" (fill-1), and randomly (random-fill). The difference between random-fill and deviation-based seed selection is that the deviation-based method selects the seed from a pool of randomly filled seeds, whereas random-fill selects the first randomly-filled seed.

We evaluated the CPU time needed for all the tested methods. As expected, the deviation-based method takes more CPU time because of the need to run deviation computation on 32x more patterns. Within the deviation-based method's run time, seed generated takes 5% of the run time, and deviation computation

TABLE IV
THE COMPARISON OF NUMBER OF EXCITED DISTINCT LONG PATHS FOR DIFFERENT LONG PATH LIMITS:70%, 80%, AND 90% OF THE CLOCK PERIOD.

	70%					80%					90%				
Benchmarks	**fill-0**	**fill-1**	**rand**	**dev**	**[19]**	**fill-0**	**fill-1**	**rand**	**dev**	**[19]**	**fill-0**	**fill-1**	**rand**	**dev**	**[19]**
wb_dma	1102	925	2465	3958	5862	0	0	558	1175	1397	0	0	313	299	577
tv80	807	1277	1616	6219	4511	267	769	783	3802	1737	165	356	373	2675	1131
mem_ctrl	3220	2406	5550	9716	5868	880	332	1137	3544	891	0	0	10	0	1
usb_funct	679	929	1674	3443	2899	490	822	1553	3309	2732	0	12	312	1646	1124
aes_core	57069	58165	86160	123063	85019	4430	4367	7249	11351	7024	150	190	368	596	308
ac97_ctrl	1220	1314	2777	3331	3236	320	345	500	583	544	292	291	292	292	292
dma	455	292	827	3899	1262	152	93	238	1500	361	25	4	26	530	49
wb_conmax	6421	5704	11626	56282	26718	1622	1339	3530	16507	5899	65	54	86	452	121
pci_bridge32	8926	9016	24955	44363	28552	7422	7985	21939	41578	25512	3463	3113	10749	24853	12864
ethernet	62287	59803	160552	195844	172679	41172	39736	120449	155960	136368	9103	8973	29329	41932	34218
vga_lcd	36141	36260	48422	49524	48511	36141	36260	48422	49524	48511	26629	26678	41354	44202	41984

takes 95%. The final pattern-sorting step takes negligible time. The high CPU time is not a serious concern since it is carried out only once during design. Furthermore, if multiple processors are available, deviation-based seed selection method can be spread over any number of processors to decrease the effective time to obtain the results.

Next, we evaluate the long-path excitation capability of the fully-specified patterns for fill-0, fill-1, random-fill, and deviation-based seed selection method. We ran detailed dynamic timing analysis for all the generated patterns in order to find all the excited path delays. Then, we defined different long path limits and counted the number of excited distinct long paths for each case. Table IV shows the results of this analysis for a range of long path limits, from 70% of the system clock period to 90% of the period. As seen, deviation-based seed-selection method clearly outperforms fill-0 and fill-1 methods, and and does significantly better than random-fill and seeds based on [19]. As we increase the long-path delay limit, we start targeting more low-slack paths. For such paths, the fill-0 and fill-1 methods provide very low coverage compared to the deviation-based method, while the deviation-based method is still considerably better than random-fill.

To evaluate the fault coverage ramp-up provided by all four methods, we ran fault injection simulations. For each benchmark, we inserted 50000 delay defects on randomly chosen nets. We assumed that the additional delay introduced by the injected defects has a distribution of e^{-Ax} as used in [7] and [5]. We let $A = \frac{5}{T_{CLK}}$, where T_{CLK} corresponds to the rated clock period of the circuit under test.

Table V lists the number of faults detected by each method. The number of detected faults for all the benchmarks is presented as Venn diagrams in Fig. 6. We find that the deviation-based method clearly outperforms both random-fill and combined fill-0/1. If we consider test escapes, we see that the deviation-based method missed the least number of faults that would otherwise be detected by other methods.

Fig.7 shows how the fault coverage increases with the number of patterns for the largest benchmark circuits. Early detection of defects is important, and it can save considerable test time in an abort-on-first-fail methodology. Each plot in these figures show results for the deviation-based method (dev), random-fill (rand), fill-0 and fill-1. We find that the coverage rises more steeply for the proposed method (dev) compared to other methods.

TABLE V
FAULT INJECTION RESULTS: NUMBER OF DETECTED FAULTS

Benchmark	**Fill-0**	**Fill-1**	**Random**	**Deviation based**	**Seeds based on [19]**
wb_dma	2,322	2,223	3,239	4,014	4,428
tv80	3,307	3,618	3,983	6,222	5,383
mem_ctrl	2,712	2,699	3,258	3,795	3,622
usb_funct	1,657	1,909	2,273	2,930	2,781
aes_core	10,979	11,002	11,283	11,780	11,243
ac97_ctrl	4,109	4,140	5,522	6,114	5,934
dma	1,301	1,145	1,600	2,828	1,921
wb_conmax	2,018	2,064	2,224	2,518	2,124
pci_bridge32	2,988	3,003	3,416	3,776	3,395
ethernet	4,375	4,311	5,994	6,425	6,012
vga_lcd	9,689	9,734	10,709	11,181	10,888

V. CONCLUSIONS

We have presented a SDD-aware seed-selection technique for LSFR-reseeding-based test compression. We used the output deviations metric to drive the seed selection algorithm. The selected seeds lead to greater excitation of short-slack paths and achieved significantly better fault detection rates compared to other methods. The proposed method embeds ATPG-generated test cubes for transition-delay faults, hence high transition-fault (TDF) coverage is ensured. We are currently evaluating the number of the top-off stuck-at patterns needed to also ensure complete stuck-at fault coverage; typically, only a few additional patterns beyond TDF patterns are needed. We are also studying other LFSR reseeding methods, e.g., those based on continuous-flow decompressors.

REFERENCES

[1] Semiconductor Industry Association, "2007 ITRS, http://www.itrs.net/links/2007itrs/home2007.htm."

[2] F.-F. Ferhani and E. McCluskey, "Classifying bad chips and ordering test sets," in *Proc. IEEE ITC*, 2006.

[3] N. Ahmed, M. Tehranipoor, and V. Jayaram, "Timing-based delay test for screening small delay defects," in *Proc. IEEE Design Automation Conf.*, 2006, pp. 320–325.

[4] X. Lin et al., "Timing-aware ATPG for high quality at-speed testing of small delay defects," in *Proc. IEEE ATS*, 2006, pp. 139–146.

[5] Y. Sato et al., "Invisible delay quality - SDQM model lights up what could not be seen," in *Proc. IEEE ITC*, 2005.

[6] R. Putman and R. Gawde, "Enhanced timing-based transition delay testing for small delay defects," in *Proc. IEEE VTS*, 2006, pp. 336–342.

[7] B. Keller et al., "An economic analysis and ROI model for nanometer test," in *Proc. IEEE ITC*, 2004, pp. 518–524.

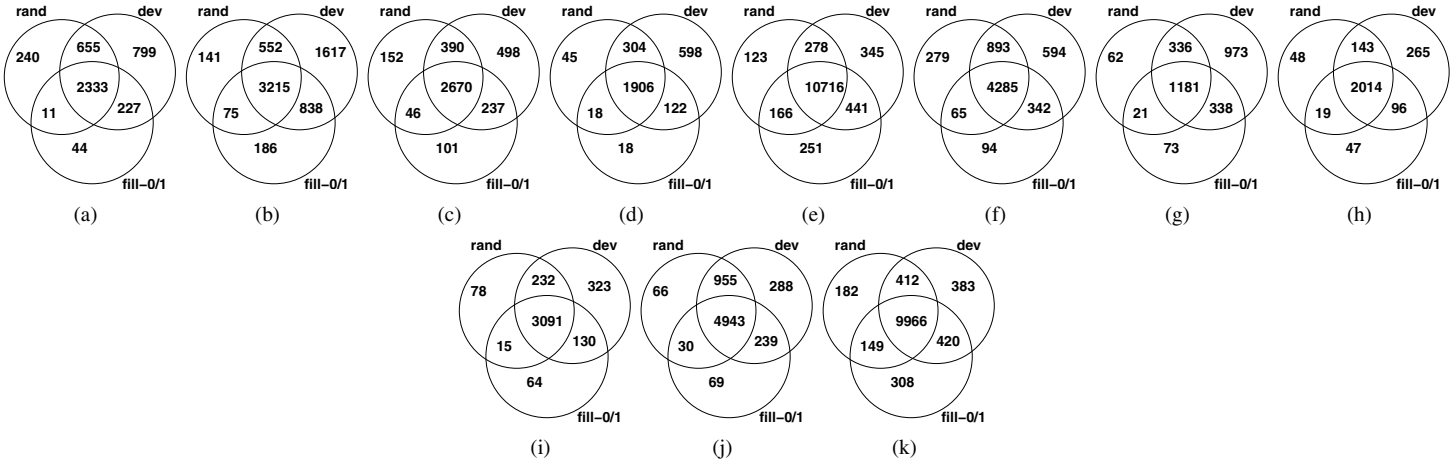


Fig. 6. The number of defects detected by the selected patterns for deviation-based (`dev`), random-fill (`rand`), and the fill-0 or fill-1 (`fill-0/1`) methods: (a) wb_dma; (b) tv80; (c) mem_ctrl; (d) usb_funct; (e) aes_core; (f) ac97_ctrl; (g) dma; (h) wb_conmax; (i) pci_bridge32; (j) ethernet; (k) vga_lcd

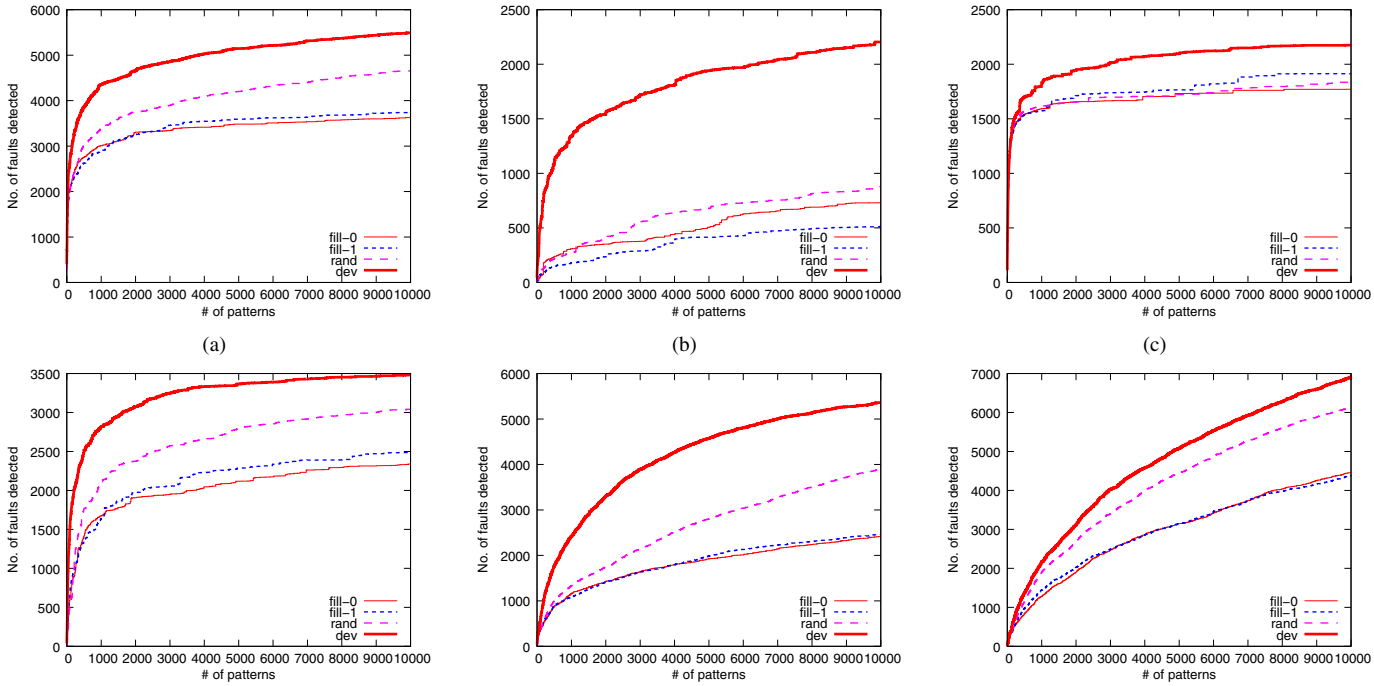


Fig. 7. The fault coverage ramp-up using the specified patterns of deviation-based (`dev`), random-fill (`rand`), and the fill-0 or fill-1 (`fill-0/1`) methods: (a) ac97_ctrl; (b) dma; (c) wb_conmax; (d) pci_bridge32; (e) ethernet; (f) vga_lcd.

[8] E. Park, M. Mercer, and T. Williams, "Statistical delay fault coverage and defect level for delay faults," in *Proc. IEEE ITC*, 1988, pp. 492–499.

[9] P. Gupta and M. Hsiao, "ALAPTF: A new transition fault model and the ATPG algorithm," in *Proc. IEEE ITC*, 2004, pp. 1053–1060.

[10] W. Qiu et al., "K longest paths per gate (KLPG) test generation for scan-based sequential circuits," in *Proc. IEEE ITC*, 2004, pp. 223–231.

[11] M. Yilmaz, K. Chakrabarty, and M. Tehranipoor, "Test-pattern grading and pattern selection for small-delay defects," in *Proc. IEEE VTS*, 2008, pp. 233–239.

[12] J. Rajski et al., "Embedded deterministic test," *IEEE Trans. CAD*, vol. 23, pp. 776–792, 2004.

[13] C. Krishna, A. Jas, and N. A. Touba, "Achieving high encoding efficiency with partial dynamic LFSR reseeding," *ACM Trans. Design Automation of Electronic Systems*, vol. 9, pp. 500–516, Oct 2004.

[14] C. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. IEEE ITC*, 2002, pp. 321–330.

[15] K. Balakrishnan, N. A. Touba, and S. Patil, "Compressing functional tests for microprocessors," in *Proc. IEEE ATS*, 2005, pp. 428–433.

[16] I. Pomeranz and S. M. Reddy, "Test data compression based on input-output dependence," *IEEE Trans. CAD*, vol. 22, no. 10, pp. 1450–1455, 2003.

[17] B. Koenemann, "LFSR-coded test patterns for scan design," in *Proc. IEEE ETS*, 1991, pp. 237–242.

[18] S. Hellebrand *et al.*, "Built-in test for circuits with scan based on reseeding of multiplepolynomial linear feedback shift registers," *IEEE Trans. Computers*, vol. 44, pp. 223–233, Feb 1995.

[19] Z. Wang, K. Chakrabarty, and M. Bienek, "A seed-selection method to increase defect coverage for LFSR-reseeding-based test compression," in *Proc. IEEE ETS*, 2007, pp. 125–130.

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

[21] Z. Wang and K. Chakrabarty, "Test-quality/cost optimization using output-deviation-based reordering of test patterns," *IEEE Trans. CAD*, vol. 27, pp. 352–365, Feb 2008.