Defect Coverage-Driven Window-Based Test Compression

Xrysovalantis Kavousianos[1], Krishnendu Chakrabarty[2], Emmanouil Kalligeros[3] and Vasileios Tenentes[1]
[1] Dept. of Computer Science, University of Ioannina, Greece
[2] Electrical Engineering Dept., Duke University, USA
[3] Information and Comm. Systems Engineering Dept., University of the Aegean, Greece
e-mail: kabousia@cs.uoi.gr, krish@ee.duke.edu, kalliger@aegean.gr, tenentes@cs.uoi.gr

***Abstract*— Although LFSR reseeding based on test cubes for modeled faults is an efficient test compression approach, it suffers from the drawback of limited, and often unpredictable, coverage of unmodeled defects. We present a new defect coverage-driven window-based LFSR reseeding technique, which offers both high test quality and high compression. The efficiency of the proposed encoding technique in detecting defects is boosted by an efficient "output deviations" metric for grading the calculated LFSR seeds. We show that, compared to standard compression-driven LFSR reseeding, higher defect coverage is obtained without any loss of compression.**

Keywords-embedded testing; linear feedback shift register; defect-oriented testing

I. Introduction

As defect screening is essential for ensuring the quality of electronic products, more test patterns are needed to target new defect types introduced in nanometer technologies. The 2007 ITRS document predicted that the test-data volume and the test-application time for integrated circuits will be respectively about 38 and 17 times higher in 2015 than in 2007. Test-data compression offers promising solutions to these challenges [1-5, 7-16]. The most widely adopted compression approach is LFSR reseeding [8]. The classical reseeding approach of [8] compresses each test cube into one LFSR seed by solving a system of linear equations, considering the content of each LFSR cell as a binary variable. Classical LFSR reseeding fails to efficiently exploit the binary variables and thus it suffers from limited compression. Today, it has been supplanted by other LFSR reseeding methods that offer better exploitation of variables, and hence higher compression [3, 7, 9-13, 15, 16]. A particularly efficient approach is window-based reseeding [7], in which each seed generates more than one test vectors.

Even though all the above methods offer high compression, they do not target unmodeled defects (i.e., defects that are not explicitly targeted by the test data being compressed). The first test compression method to target unmodeled defects was proposed in [18] and it is based on the classical LFSR reseeding approach [8]. This method improves the defect coverage of the seeds using output deviations [19], which offer an effective probabilistic means to evaluate test vectors without being biased towards any particular fault model. As shown in [17], unbiased testing provides higher test quality than a test method that is biased by a fault model.

Despite its advantages, the method proposed in [18] suffers from several serious drawbacks. At first, as every classical LFSR reseeding method, it suffers from limited compression. In addition, it adopts an inefficient output-deviation metric to evaluate seeds, which does not exploit all the potential offered by output deviations for identifying the most effective (in terms of defect coverage) seeds. On top of that, it exploits only the free variables of each seed to improve the defect coverage of the seeds. This provides just a limited improvement in the output deviations and consequently in the defect coverage of the resulting test vectors. Finally, it is unsuitable for more efficient reseeding techniques, such as window-based reseeding, that typically leave almost no free variables after each seed is computed.

In this paper, we present a novel encoding method that offers high compression and increased unmodeled defect coverage at the same time. The encoding method is tailored to the highly efficient window-based LFSR reseeding approach and exploits every single seed variable solely for minimizing the seed count. The defect-detection potential of the generated seeds is enhanced using intelligent encoding of test cubes and thus compression is not compromised. Unmodeled defect coverage is further improved compared to [18] by using a new effective output-deviation-based metric.

Simulation results on stuck-at test sets of the ISCAS'89 and IWLS'05 [20] benchmark circuits show that the proposed defect coverage-driven window-based method offers higher defect coverage than the original compression-driven window-based method, without any adverse impact on compression. In addition, due to the highly effective (in terms of compression) encoding method and the efficient output-deviation metric, the proposed method clearly outperforms [18] in terms of both compression and defect coverage. Finally, by grading the seeds and applying the most efficient seeds first, faster coverage ramp-up is achieved, thus reducing the test-application time in an abort-at-first-fail environment.

II. Motivation

In the sequel, the term "test cube" refers to a test pattern with 'x' logic values, whereas the term "test vector" refers to a test pattern without 'x' logic values. The decompression architecture consists of an L-bit LFSR and a phase shifter that drives m scan chains ($m > L$). The LFSR is reseeded by the Automatic Test Equipment (ATE). In static reseeding, the seed of the LFSR is its initial state and is considered as a set of binary variables, a_0, ..., a_{L-1}, that are loaded from the ATE. A seed is determined by solving a system of linear equations, formed according to the specified bits of the test cube and the feedback polynomial of the LFSR [8].

The main disadvantage of classical reseeding [8] is that every new seed flushes the LFSR and thus any variables left unspecified (free) during the seed-computation process are wasted. To exploit the otherwise wasted free variables, the method proposed in [18] utilizes the notion of output deviations [19] for increasing the unmodeled defect coverage.

Output deviations are probability measures at primary outputs and pseudo-outputs that indicate the likelihood of error detection at these outputs. Test vectors with high deviations tend to be more effective for fault detection [19]. The authors of [18] generate multiple candidate seeds for each test cube by applying multiple random fillings on the free variables. The seeds generating the vectors with the highest output deviations are selected among all candidates.

The major drawback of [18] is that it offers limited compression. Window-based reseeding [7] offers much better compression than [8] and [18] since it efficiently exploits the seed variables. Specifically, each seed is expanded into $w > 1$ test vectors (w is referred to as the *window size*). Every position of the window can be used for encoding a test cube, and thus multiple incompatible as well as compatible test cubes can be encoded at the same window.

The random filling of free variables proposed in [18] cannot be applied to the window-based reseeding, which exploits almost all variables for minimizing the seed count. Moreover, [18] requires all candidate seeds for all test cubes to be computed before the selection process begins. This cannot be done in window-based reseeding, since the computation of the candidate seeds for a specific seed depends on the previously selected seeds (we have to know which cubes have been encoded by the previous seeds, in order to compute the candidate seeds for the next one). To overcome this problem, the proposed method generates multiple candidate seeds that implement different unique encodings of the test cubes. Therefore, the probability of generating a vector with high output deviation values increases. In fact the variations of the seeds in terms of output deviation values are more significant than those in [18], as the encoding of different combination of test cubes into a candidate seed affects the generated vectors more than the random replacement of the free variables. At the same time, high compression is ensured by intelligently generating the candidate seeds in such a way that exploits the variables for decreasing the seed volume.

III. Generation of Defect Coverage-Driven Seeds

A. Generation of Candidate Seeds

As observed in [7], high compression is achieved if the following properties are maintained throughout encoding:

1. The most-highly specified cubes are encoded first.
2. Among equally-specified test cubes, the one consuming the fewest seed variables is selected.
3. Each seed encodes as many test cubes as possible (i.e., no seed variable is left unspecified if it can be used to encode additional test cubes).

Even though these properties (denoted hereafter as *compression-maximization criterion*) offer high compression, they usually leave no free variables and thus they render [18] unsuitable for increasing the defect coverage of the seeds. We propose a new encoding method that enables the generation of multiple candidate seeds. The proposed encoding method relaxes the application of the above compression-based properties without compromising the exploitation of seed variables for encoding test cubes. Specifically, it considers different encoding alternatives that lead to the same or very similar compression results (i.e., all free variables are exploited for encoding test cubes in this case too). Such alternatives can be different test cubes with the same specified-bit volume, the same test cube encoded in different window positions, etc. Besides high compression efficiency, the candidate seeds offer wide diversity and thus significantly different defect-coverage options.

The proposed encoding method generates at each step T candidate seeds (T is a user-defined parameter) as follows: we start by encoding the most-specified test cube (say t_1) in the first position of a window. Next, for initiating the generation of the T candidate seeds, we independently apply in that window the compression-maximization criterion T times, excluding each time all the previous decisions. In other words, we identify the best T different test-cube encodings that can be independently performed in the window that embeds t_1 in its first position. As a result, T different windows with t_1 in their first position, and other test cubes in the remaining positions are determined.

The above procedure implies that we initially target windows that embed two different test cubes. Note that this does not necessarily mean that the T chosen windows embed T different pairs of cubes (i.e., t_1 along with another cube). Test cube t_1 can be combined with the same test cube, t_i, more than once, if t_i can be encoded in different positions of the window and the corresponding solutions are among the T best solutions according to the compression-maximization criterion. Hence, among the T chosen windows, there may be more than one embedding t_1 and t_i, with t_i encoded in a different window-position every time. However, if all possible windows that embed t_1 with a second test cube are fewer in number than T, then we increase the volume of the already chosen windows by encoding in them a different third test cube. Two new different windows embedding three (n) test cubes can be derived from one window which embeds two ($n - 1$) test cubes, by separately encoding in the latter either two different test cubes (one for each new window), or the same cube in two different positions. The same procedure is repeated until we get T different windows, corresponding to the T candidate seeds. At this point, the set of candidate seeds has T members; therefore, we continue by encoding as many test cubes as possible in the window of each candidate seed by using only the compression-maximization criterion. The T generated candidate seeds are evaluated using the proposed metric (see Section III.B), and the best one is selected. The cubes encoded by the selected seed are dropped from the set of test cubes. We provide insights into the above process with the following example.

Example 1. Let t_1, t_2, ..., t_{10} be 10 test cubes sorted in descending order of volume of specified bits, w=4 be the window size, and T=5 be the number of candidate seeds. In Fig. 1, we present each window as a column with 4 cells, one for each window position. Each encoded test cube is reported inside the corresponding cell and the newly encoded test cubes are highlighted at each step. Initially, we encode t_1 (the most specified cube) in window position 1 (Fig. 1a). Let us assume that the systems of equations for test cubes t_2, t_4, t_8, and t_{10} are independently solvable in the same window with t_1 (t_2 is the first cube selected by the compression-

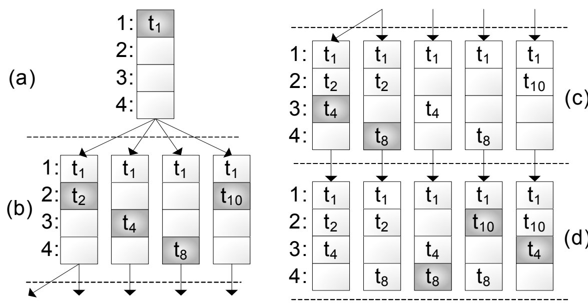

Fig. 1. An example to illustrate the generation of T candidate seeds.

maximization criterion, t_4 is the next selection, i.e., if we exclude t_2, and so on). We initiate the generation of four new candidate seeds (Fig. 1b) by encoding each one of these test cubes separately into the window that we previously encoded t_1 (i.e., each one of the four seeds encodes one of the following pairs of test cubes: t_1 and t_2, t_1 and t_4, t_1 and t_8, t_1 and t_{10}). Then, we encode a third test cube in the windows generated so far (Fig. 1c). After encoding test cube t_4 first, and then t_8, in the window embedding t_1 and t_2 (the compression-maximization criterion is again used for these selections), we reach the limit of 5 candidate seeds. The generation of new windows now terminates and we continue by encoding in each of the T windows only the test cubes that maximize compression (Fig. 1d). Finally, the 5 candidate seeds are evaluated using the proposed metric (note that the leftmost seed, provides the best compression). ■

As mentioned earlier, in contrast with [7], we examine various encoding options, apart from the one that maximizes compression (i.e., t_1 along with t_2 and t_4 in Example 2) for maximizing defect coverage. Also, by trying different encodings early on in the encoding process (i.e., after the selection of just the first cube for every window) we guarantee that the T candidate seeds will be sufficiently different (and hence they will potentially provide sufficiently different defect coverage). By selecting these different encodings using the compression-maximization criterion, we ensure that compression is not compromised.

One advantage of window-based reseeding is that the size of the window (w) offers a tradeoff between compression and test sequence length. Specifically, large values of w offer very high compression at the expense of relatively increased test sequence length, whereas small values of w offer short test sequence length at the expense of relatively reduced compression [15]. In the degenerate case of w=1, every seed generates only one test vector. The test-application time is minimized, but *only compatible* test cubes can be encoded by each seed. This restriction limits the encoding ability of the T candidate seeds' generation process described in the previous section, and consequently it adversely affects both the encoding ability and the defect-screening potential of the resulting seeds. However, the use of uncompacted test cubes combined with the defect coverage-driven compression-maximization criterion presented in this section, almost eliminates these adverse effects and also offers the potential for a wide range of encoding options. This is the significant difference between the classical and the window-based reseeding approach as for w=1; in classical reseeding, as proposed in [8] (and adopted in [18]), only one test cube is encoded by each seed, whereas in window-based reseeding for w=1, the utilization of the defect coverage-driven compression-maximization criterion offers an efficient way to combine more than one compatible test cubes in the same encoded pattern. Thus, as it will be shown in Section IV, the volume of the defect coverage-driven seeds is low and their quality is high for w=1 as well.

B. Evaluation of Candidate Seeds

We assume that each seed s is expanded into w test vectors (w is the size of the window) and each one of them is applied using two capture cycles r_1 and r_2 (launch-on-capture). The Maximum Expected Deviation value $MED(i, r_k, v)$ for output i at capture cycle r_k (k=1, 2) and fault-free response v (v=0, 1) is an estimate of the maximum deviation value expected throughout the seed-computation process at output i, when its fault-free response is v at capture cycle r_k. Such an estimate is needed for discarding outputs that do not get high deviation values, when evaluating the candidate seeds. Note that the actual maximum deviation value at an output is known only after all the candidate seeds for all test cubes are generated, that is when the whole encoding process is over. However, we still need a means for assessing the quality of the candidate seeds when performing the encoding. That is the purpose of $MED(i, r_k, v)$, which is calculated as follows: initially, for every test cube, a predetermined number of single-vector seeds (i.e., seeds encoding only one test cube, as those used in [18]) are generated by randomly filling the free variables. For each output i, the generated test vectors are partitioned into four groups: those producing fault-free responses 0 and 1 at both capture cycles r_1 and r_2. The output-deviation values of all generated test vectors are calculated (please refer to [19] for details on output-deviation calculation) and the greatest value for every output i and fault-free response v = 0, 1 at capture cycle r_k, constitutes $MED(i, r_k, v)$. After calculating the MED values, the generated single-vector seeds are discarded.

Let $D(s, j, i, r_k, v)$ be the deviation value at output i for the j-th test vector in the window of candidate seed s ($j \in [1, w]$, where w is the window size), which produces fault-free response v at that output at capture cycle r_k. The value $D(s, j, i, r_k, v)$ is considered to be near-maximum if it is very close to $MED(i, r_k, v)$ for the same output, or equivalently, if:

$$D(s, j, i, r_k, v) \geq F_1 \cdot MED(i, r_k, v),\ v = 0, 1 \qquad (1)$$

For selecting seeds with near maximum output deviation values, F_1 should be close to 1. We verified that a value of $F_1 \in [0.99, 0.995]$ provides high-quality seeds.

Another characteristic that is incorporated in the proposed metric is that each output contributes to the metric according to its potential of observing errors due to defects. To do so, every output i is assigned a set of weights $wo(i, r_k, v)$, for k=1, 2 and v=0, 1, which are initially all set equal to the number of lines in the logic cone of the corresponding output. These weights are indicative of the volume of undetected defects that can be possibly detected for fault-free response v at output i during capture cycle r_k (the more the lines in the logic cone of an output, the highest the probability to detect more unmodeled defects at that output).

The weights $wo(i, r_k, v)$ and the output deviation values are used during the evaluation of the candidate seeds for determining the proposed metric $WS(s)$, which is a weight for every candidate seed s. Let j be one of the w window positions, i.e., $j\in[1, w]$. For test vector j, we define the sets $MS[s, j, r_k, v]$ which consist of all circuit's outputs i, for which the deviation value $D(s, j, i, r_k, v)$ satisfies inequality (1) [i.e., sets MS contain all the outputs, which get near-maximum deviation value during the application of vector j]. We can now calculate $WS(s)$ as a sum of output weights (wo), according to the following formula:

$$WS(s)=\sum_{k=1,2}\sum_{v=0,1}\sum_{j\in[1,w]}\sum_{i\in MS[s,j,r_k,v]} wo(i,r_k,v) \qquad (2)$$

This formula means that, for either fault-free response 0 or 1, only the weights of the outputs that get near-maximum deviation values for capture cycles r_k (i.e., of the outputs belonging to $MS[s, j, r_k, v]$) participate into the final weights sum $WS(s)$. Note that the first response targets timing-independent defects and the second response targets timing-dependent defects. The seed with the highest WS value is selected as the one with the best potential to detect timing-independent and timing-dependent unmodeled defects.

The weight $WS(s)$ enables the selection of seeds that generate vectors with the maximum deviation values at the outputs of large cones of the CUT. However, maximizing the deviations only at a subset of outputs may result in low defect coverage, even when this subset consists of the outputs of the largest logic cones. To this end, for every selected seed, every output i which satisfies equation (1) is identified, and the respective weight $wo(i, r_k, v)$ is divided by a constant factor F_2. Thus, this output's weight has small impact on the selection of the next seeds. Note that, if seed s provides a high deviation at output i for fault-free response v at capture cycle r_k, then it is likely that many defects at the fan-in cone of i will be detectable at output i when s is applied. Consequently, test vectors that maximize the deviation at output i for the same fault-free response and the same capture cycle will be less effective for increasing the defect coverage during the application of the next seeds. We verified experimentally that a value of F_2 in the range [2, 10] is sufficient to maximize the deviations at all outputs.

We have to note that, contrary to the proposed metric, the metric presented in [18] is unsuitable for window-based reseeding, as it requires the generation of all candidate seeds for all test cubes before the evaluation process begins. This is also the case for the more efficient output deviation-based metric that was proposed in [6]. Moreover, both metrics in [6] and [18] evaluate test vectors for either timing-independent or timing-dependent defects, whereas the proposed metric improves the detection of both timing-related and timing-independent defects at the same time.

Even though the best candidate seed is selected each time, this can still be inferior compared to seeds selected at later iterations. For example, the encoding of a combination of sparsely specified test cubes at a later step may produce a seed with higher weight than a seed produced by encoding a combination of densely specified test cubes at an earlier step. Thus, high defect coverage ramp-up cannot be guaranteed if the seeds are applied in the order they are selected. To alleviate this problem, we rank the selected seeds as a final step.

TABLE I. TEST-DATA VOLUME (IN KBITS)

Circuit	Classical Reseeding		Window-Based Reseeding			
			w=1		w=5	
	TS Size	[8], [18]	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	28.7	16.1	8.0	8.0	6.2	6.3
s9234	41.0	23.2	16.9	18.4	14.2	14.3
s13207	188.3	78.0	12.0	12.8	8.2	8.0
s15850	99.0	47.0	18.6	19.0	13.6	14.0
s38417	238.0	117.3	64.6	65.4	58.2	59.7
s38584	270.8	148.0	34.0	34.0	27.2	26.9
ac97_ctrl	148.7	68.6	11.0	10.9	7.2	7.3
mem_ctrl	720.0	373.9	113.9	117.8	79.7	86.6
pci_bridge	1160.6	343.2	111.4	110.3	100.2	99.7
tv80	281.6	151.4	99.8	102.5	54.3	55.7
usb_funct	252.7	129.2	57.5	57.4	48.9	49.4
ethernet	$11.8\text{x}10^3$	$1.7\ \text{x}10^3$	203.8	225.5	162.5	165.1

Specifically, the selected seeds are evaluated using a process that is similar to the T candidate seeds evaluation procedure, which is now applied to the selected seeds and not to candidate seeds. Since all seeds are known at this step, the actual maximum deviation value $MD(i, r_k, v)$ for each output i and fault-free response v=0, 1 at capture cycle r_k can be easily computed (it is the largest among the output-deviation values of all test vectors generated by all computed seeds). Equation (2) is applied in this case too, but the set $MS[s, j, r_k, v]$ is calculated by replacing values $MED(i, r_k, v)$ with values $MD(i, r_k, v)$ in inequality (1).

IV. SIMULATION RESULTS

We conducted experiments using the largest ISCAS'89 and a subset of the IWLS'05 circuits [20]. The number of scan chains was set equal to 30 for the ISCAS, 50 for the medium sized IWLS, and 100 for the large *ethernet* IWLS circuit. We have implemented the following techniques: a) the classical reseeding method [8], b) the defect coverage-driven classical reseeding method of [18], c) the compression-driven window-based reseeding method for w=1 and w=5 denoted as "Cmp", and d) the proposed defect coverage-driven window-based reseeding for w=1 and w=5, denoted as "Cmp & Def". In order to minimize the seed count and the test sequence length (TSL) for the classical reseeding methods, we used compacted test sets. For maximizing the effectiveness of window-based LFSR reseeding methods we used uncompacted test sets (in all cases the same commercial ATPG engine was used). For each benchmark circuit, a dedicated LFSR with a characteristic primitive polynomial of near minimum size was selected. For the proposed "Cmp & Def" method, T, F_1 and F_2 were set equal to 30, 0.995 and 8 respectively.

Table I presents the test-data volumes in Kbits (1Kbit $=10^3$ bits) for the window-based and the classical reseeding approaches. The first column lists the names of the benchmark circuits. The next column presents the sizes of the stuck-at test sets used for the evaluation of both [8] and [18]. The third column shows the volumes of the compressed stuck-at test data for these two methods. Note that the test-data volumes are the same for the two methods, as [18] differs from classical LFSR reseeding only in the way that the free variables are filled. The next two pairs of columns present the test data volumes for w=1 and w=5, in their com-

TABLE II. TEST-SEQUENCE LENGTH (# OF VECTORS APPLIED)

Circuit	Classical Reseeding	Window-Based Reseeding			
		w=1		w=5	
	[8], [18]	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	134	199	199	770	785
s9234	166	282	307	1185	1195
s13207	269	300	320	1030	1005
s15850	162	310	316	1130	1170
s38417	143	808	818	3635	3730
s38584	185	485	485	1945	1920
ac97_ctrl	66	274	273	895	910
mem_ctrl	603	876	906	3065	3330
pci_bridge	330	1238	1226	5565	5540
tv80	757	1663	1708	4525	4645
usb_funct	136	959	956	4075	4115
ethernet	1111	2912	3222	11610	11790

TABLE III. TRANSITION-FAULT COVERAGE (%)

Circuit	Classical Reseeding		Window-Based Reseeding			
			w=1		w=5	
	[8]	[18]	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	61.11	63.49	62.90	66.38	65.66	70.32
s9234	40.69	49.63	43.04	53.08	53.94	58.41
s13207	61.95	69.48	62.94	68.28	64.31	70.32
s15850	52.75	55.25	53.58	56.95	57.58	58.31
s38417	79.16	80.24	85.42	87.93	88.85	90.60
s38584	61.45	62.21	65.03	66.32	68.10	69.07
ac97_ctrl	42.71	45.60	47.18	56.42	52.40	63.95
mem_ctrl	41.09	44.24	42.69	46.01	44.03	47.36
pci_bridge32	65.17	69.50	77.39	85.80	82.96	87.50
tv80	53.78	59.31	60.16	64.76	61.97	64.90
usb_funct	63.24	64.49	71.40	75.53	74.53	79.39
ethernet	47.60	49.56	53.94	63.79	71.37	83.14

pression-driven versions ("Cmp"), as well as in their proposed defect coverage-driven versions ("Cmp & Def").

As can be seen in Table I, the window-based reseeding approach clearly outperforms the classical static reseeding approaches ([8] and [18]), while the highest compression is always achieved by window-based reseeding for *w*=5. Moreover, the proposed defect coverage-driven encoding (columns labeled "Cmp & Def"), for both window sizes *w*=1 and *w*=5, provide nearly the same compression as the original compression-driven encoding (columns labeled "Cmp") for the respective window sizes. In a few cases, the proposed encoding provides even better compression, which is due to the heuristic nature of the encoding algorithm.

Table II presents the TSL of the examined reseeding methods, in terms of the test vectors applied to each circuit. Column 2 presents the TSLs of the classical reseeding approaches (which are the same for both [8] and [18]). The next two pairs of columns present the TSLs of the *w*=1 and *w*=5 "Cmp" and "Cmp & Def" cases. As expected, the classical and window-based reseeding for *w* = 1 offer short and comparable, in many cases, TSLs, while the TSLs of window-based reseeding increases when *w* increases to 5. However, we have to note that the long TSLs in the window-based reseeding, are mainly attributed to the very small LFSRs used. Larger LFSRs can be well exploited by window-based reseeding to offer considerably smaller number of seeds with minimal impact on compression.

For evaluating the defect detection potential of the proposed defect coverage-driven reseeding method, we consider the coverage of unmodeled faults by means of surrogate fault models (i.e., fault models which are not considered during ATPG). Specifically, we compute the transition and bridging fault coverage obtained by applying to the circuit under test the test vectors generated by the computed seeds (note that seeds compress test cubes which target only stuck-at faults). As is common in industry, we use the launch-on-capture scheme, to apply test-vector pairs. In the case of [18], which considers only one response of every vector pair, we chose to evaluate the generated seeds using the second response of each vector pair (i.e., timing-dependent defects are favored).

First we evaluate the proposed encoding with respect to the achieved transition-fault coverage. The corresponding results are shown in Table III. It is obvious that the proposed output-deviation metric increases the transition fault coverage significantly. Even though the method described in [18] achieves higher transition-fault coverage than [8], it is still much less effective than the proposed defect coverage- driven window-based reseeding for *w*=1 and *w*=5, in nearly all cases. Additionally, the defect coverage achieved by the proposed method for *w*=5 is higher than the defect coverage achieved by the proposed method for *w*=1. This is mainly a result of the increased diversity of the candidate seeds in case of *w*=5. This diversity can be attributed in part to the fact that many seeds encode incompatible test cubes when *w*>1. Note that the increased TSL in the case of *w*=5 contributes also to the increased defect coverage, compared to the other cases.

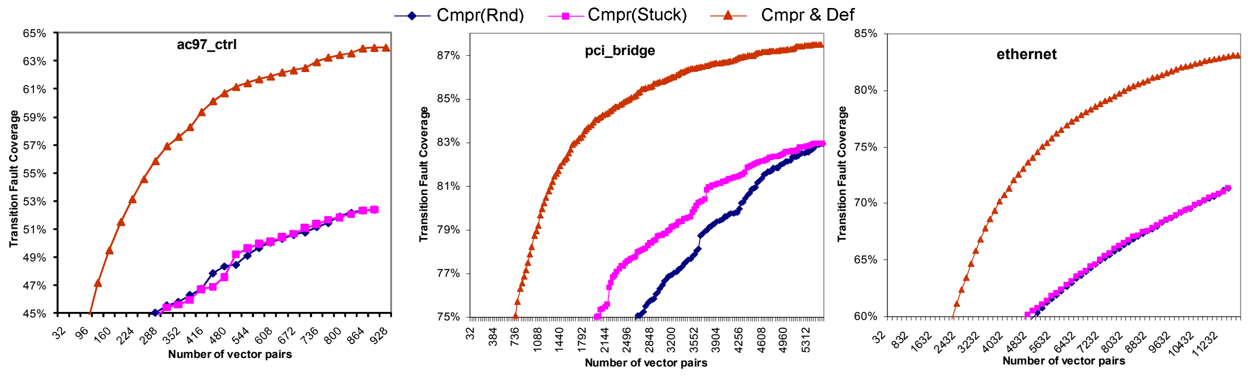

Fig. 2. Transition fault coverage ramp-up for window-based reseeding (*w*=5).

TABLE IV. BRIDGING-FAULT COVERAGE RESULTS (%)

Circuit	Classical Reseeding		Window-Based Reseeding			
			w=1		w=5	
	[8]	[18]	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	94.14	94.35	94.85	95.19	95.72	96.26
s9234	86.56	86.58	87.95	88.29	88.70	89.00
s13207	91.99	92.14	92.08	92.95	92.92	93.57
s15850	93.47	93.59	94.38	94.51	94.71	94.89
s38417	97.13	97.15	97.88	98.15	98.26	98.44
s38584	89.85	89.91	90.89	91.09	91.67	91.98
ac97_ctrl	97.02	97.02	98.75	98.87	99.10	99.23
mem_ctrl	74.60	74.61	75.08	75.44	75.78	76.10
pci_bridge32	96.78	96.82	98.14	98.28	98.45	98.55
tv80	89.26	89.33	90.86	91.23	91.57	91.74
usb_funct	95.15	95.19	96.73	97.16	97.17	97.45
ethernet	90.63	90.77	93.59	94.18	95.57	95.71

However, according to the results shown in Table III, this contribution is less significant than the contribution of the proposed encoding method (note that in most cases, the compression-driven window-based reseeding for *w*=5 offers lower transition fault coverage than the defect coverage-driven window-based reseeding for *w*=1, even though the test sequences in the former case are longer).

Fig. 2 illustrates the transition fault coverage ramp-up achieved by the window-based reseeding method for *w*=5, for selected circuits (the complete set of charts for *w*=1 and *w*=5 can be found on [21]). The x-axis presents the number of the applied vector pairs and the y-axis the transition-fault coverage. The seeds for the compression-driven window-based reseeding method have been sorted: a) randomly ("Cmp(Rnd)"), and b) in descending order of their stuck-at-fault coverage ("Cmp(Stuck)"). The proposed method exhibits higher coverage ramp-up than the other methods, with the "Cmp(Stuck)" being better than the "Cmp(Rnd)". Especially for the largest benchmark *ethernet*, which consists of 136.2K gates and 10.5K scan cells and is representative of real-life industry circuits, the improvement is striking.

Finally, we evaluate the proposed method, [8] and [18], in terms of bridging-fault coverage. 100K pairs of lines were selected randomly for each circuit and for each pair, four bridging faults were simulated by considering both lines as aggressors and victims, and both logic values 0 and 1 at the aggressors. Table IV presents the results. In all cases, the proposed encoding "Cmp & Def" achieves higher coverage of bridging faults than the original "Cmp" method. In contrast, in the method described in [18], the improvement is small compared to the classical reseeding [8]. Moreover, the proposed encoding method offers higher bridging fault coverage than [18]. The main reason for this observation is that [18] considers only one of the two responses of each LOC vector-pair (either the first or the second) for calculating the output deviations. In our experiments, we considered only the second response, as stated earlier, to enhance the detection of timing related defects. However, bridging faults are detected by the first response. This is another weakness of [18], compared to the proposed method, which is able to consider both responses of each pair. Thus, we conclude that the proposed method improves the bridging fault coverage, which is also a significant advantage over [18].

V. CONCLUSIONS

We have presented a defect-driven window-based LFSR reseeding technique, which offers high unmodeled defect coverage using stuck-at test sets. Unmodeled defect coverage has been evaluated using transition and bridging faults as surrogate fault models. Results show that the proposed method achieves higher defect coverage and faster coverage ramp-up than the compression-driven window-based reseeding, without compromising compression.

ACKNOWLEDGMENT

The work of K. Chakrabarty was supported by SRC under contract no. 1588.

REFERENCES

[1] A. Chandra, K. Chakrabarty, "Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length codes" IEEE Trans. on Comp, vol. 52, pp 1076-1088, 2003.

[2] P. Gonciari, B. Al-Hashimi and N. Nicolici, "Variable-length input Huffman coding for system-on-a-chip test", IEEE Trans. on CAD, vol. 22, pp. 783-796, June 2003.

[3] S. Hellebrand et al., "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers", IEEE Trans. on Comp., vol. 44, pp. 223-233, Feb. 1995.

[4] A. Jas, et. all, "An efficient test vector compression scheme using selective Huffman coding", IEEE Trans. on CAD, vol. 22, pp. 797-806, June 2003.

[5] X. Kavousianos et al., "Test Data Compression Based on Variable-to-Variable Huffman Encoding With Codeword Reusability", IEEE Trans. CAD, vol. 27, pp. 1333-1338, July 2008.

[6] X. Kavousianos and K. Chakrabarty, " Generation of Compact Test Sets with High Defect Coverage" Proc. of DATE, pp. 1130-1135, 2009

[7] E. Kalligeros et al., "Efficient Multiphase Test set embedding for scan-based testing", in Proc. ISQED, 2006, pp. 433-438.

[8] B. Koenemann, "LFSR-coded test patterns for scan design", in Proc ETC, 1991, pp. 237-242.

[9] B. Koenemann, et al., "A SmartBIST variant with guaranteed encoding", in Proc. ATS, 2001, pp. 325-330.

[10] C. Krishna and N. Touba, "Reducing test data volume using LFSR reseeding with seed compression", in Proc. ITC, 2002, pp. 321-330.

[11] C. Krishna, A. Jas, and N. Touba, "Test Vector Encoding Using Partial LFSR Reseeding", in Proc. ITC, 2001, pp. 885-893.

[12] S. Mitra and K. Kim, "XPAND: An efficient test stimulus compression technique", IEEE Trans. on Comp., vol. 55, pp. 163-173, Feb. 2006.

[13] J. Rajski et al., "Embedded deterministic test", IEEE Trans. on CAD, vol. 23, pp. 776-792, May 2004.

[14] L. Schäfer, et al., "RESPIN++ – deterministic embedded test", in Proc. ETW, 2002, pp. 37-44.

[15] V. Tenentes et al., "State Skip LFSRs: Bridging the Gap between Test Data Compression and Test Set Embedding for IP Cores", in Proc. DATE 2008, pp. 474-479.

[16] E. Volkerink, and S. Mitra, "Efficient seed utilization for reseeding based compression", in Proc. VTS 2003, pp. 232-237.

[17] L. Wang et al., "On the Decline of Testing Efficiency as Fault Coverage Approaches 100%", in Proc. VTS, pp.74-83, 1995

[18] Z. Wang, et al., "Deviation-based LFSR reseeding for test-data compression", IEEE Trans. on CAD, vol. 29, pp. 259-271, 2009.

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

[20] IWLS'05 circuits: http://www.iwls.org/iwls2005/benchmarks.html

[21] Web site: http://www.cs.uoi.gr/~kabousia/window-reseeding.htm.