Technical Report on

Defect-Oriented LFSR Reseeding to Target Unmodeled Defects Using Stuck-at Test Sets

Vasileios Tenentes, Xrysovalantis Kavousianos

Department of Computer Science & Engineering

University of Ioannina

Technical Report Number: 03 - 2013

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

Co- financed by Greece and the European Union

Overview

Defect screening is a major challenge for nanoscale CMOS circuits, especially since many defects cannot be accurately modeled using known fault models. The effectiveness of test methods for such circuits can therefore be measured in terms of the coverage obtained for unmodeled faults. In this report, we present new defect-oriented LFSR reseeding techniques for test-data compression. The proposed techniques are based on a new "output deviations" metric for grading stuck-at patterns derived from LFSR seeds, and include a window-based static reseeding method as well as a dynamic reseeding method based on a ring generator. We show that, compared to standard compression-driven LFSR reseeding and a previously proposed deviation-based method, higher defect coverage is obtained using stuck-at test cubes without any loss of compression. The defect coverage for the proposed reseeding methods based on stuck-at test cubes is evaluated using two surrogate fault models, namely the transition fault model and the bridging fault model.

1 Introduction

The need for more and more complex fault models is rapidly increasing the test data volume for integrated circuits testing. Moreover, in VDSM technologies many defects cannot even be accurately modeled using known fault models [11]. It is, therefore, important to grade the tests patterns of well-practiced fault models (such as stuck-at) based on their ability to detect unmodeled defects. For this reason probabilistic fault models have been developed.

In [12–15] a probabilistic gate-level fault model is presented together with a technique to compare tests patterns for their ability to detect arbitrary defects. This model is based on probability measures, named *output deviations*, at primary outputs and pseudo-outputs (all referred to as outputs) that reflect the likelihood of error detection at these outputs. It was shown in [13], test patterns with high deviations tend to be more effective for fault detection.

In [12, 14] test set enhancement techniques for selection of test patterns that maximize output deviations were presented. The reordering technique of test patterns presented in [13] maximizes the defect coverage ramp-up for an abort-on-first-fail test environment, while [5] describes a static compaction technique for stuck-at test vectors that maximizes output deviations. The compression method of [15] presents a quality enchancement for classical LFSR reseeding [7] technique that maximizes the output deviations of the generated patterns by exploiting wasted variables.

In this report, a new encoding method that offers high compression and increased unmodeled defect coverage, for static and dynamic LFSR reseeding is presented. The main contributions are:

1. The new encoding method is suitable for both static window-based and dynamic LFSR reseeding, which are among the most efficient reseeding techniques.

2. High unmodeled defect coverage is achieved by using a new output- deviation-based metric that is more effective than [15] for detecting defects.

3. The encoding method enhances the defect-detection potential of the generated seeds without compromising compression.

4. Instead of exploiting free variables, defect detection is facilitated simply by carefully encoding the test cubes into seeds using compression as well as defect-oriented criteria.

Simulation results are presented for stuck-at test sets generated for the ISCAS'89 and IWLS'05 benchmark circuits [1]. These results show that the defect-aware window-based and dynamic reseeding methods offer higher defect coverage than the original (defect-unaware) window-based and dynamic reseeding methods, without any adverse impact on compression. In addition, due to the efficient output deviation metric introduced in this report, the new method clearly outperforms [15] in terms of defect coverage. Finally, by grading the seeds in the case of static window-based reseeding 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. Even for dynamic reseeding, where seed ordering is not an option, the carefully-tuned defect-oriented reseeding provides steeper coverage ramp-up.

The rest of the report is organized as follows. Section 2 presents motivation for this work and Section 3 presents the new output deviation-based metric. Section 4 describes the procedure used for generating high-quality seeds using this metric. Simulation results are presented in Section 5 and Section 6 concludes the report.

2 Motivation

Figure 1 presents the decompression architecture used in static as well as in dynamic reseeding. It consists of an L-bit sequential linear decompressor, which can be either an LFSR or a ring generator [9], and a phase shifter that receives the outputs of the decompressor and drives m scan chains ($m > L$). A test response compactor is also included in the scheme. The decompressor is reseeded by the Automatic Test Equipment (ATE), and it generates a test vector through the phase shifter. In static reseeding, the seed of the decompresor is its initial state and it is considered as a set of binary variables $a_0, \ldots, a_{L-1}$ that are loaded directly from the ATE. In dynamic reseeding, the decompressor is reseeded at every cycle from the ATE by injecting test bits into it through the ATE channels. Every injected test bit is considered as a binary variable and all variables injected during the generation of one test vector constitute the dynamic seed for this test vector. In both cases, a seed is determined by solving a system of linear equations, which is formed according to the specified bits of the test cubes and the feedback polynomial of the LFSR [6].

The main disadvantage of the static LFSR reseeding is that every new seed flushes the decompressor contents and thus any variables left unspecified (free) during the seed-computation process are wasted. The method proposed in [15] exploits these free variables in order to increase unmodeled defect coverage. In order to achieve this goal, it utilizes the notion of output deviations [13]. Output deviations are probability measures at primary outputs and pseudo-outputs that indicate the likelihood of error detection at these outputs. As it was shown in [13], test patterns with high deviations tend to be more effective for fault detection. The method proposed in [15] attempts to improve the output deviations of the seeds as follows: it first applies multiple random fillings on the variables that remain free after each system of linear equations is solved, in order to generate multiple candidate seeds for each test cube. Then it selects the seeds that generate the vectors with the highest output deviation values.

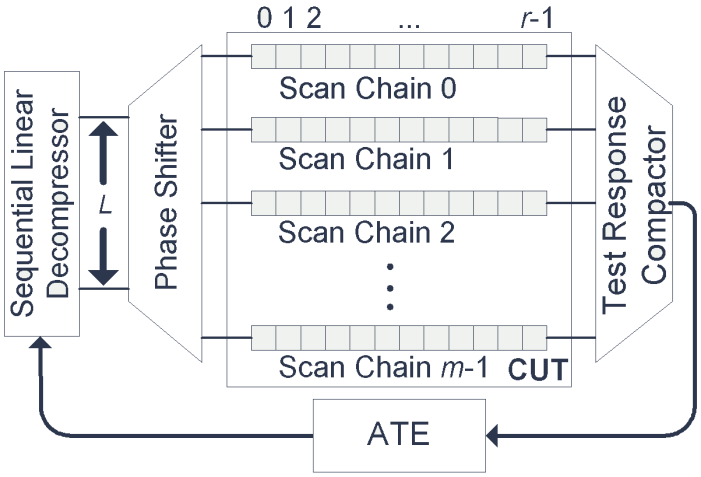
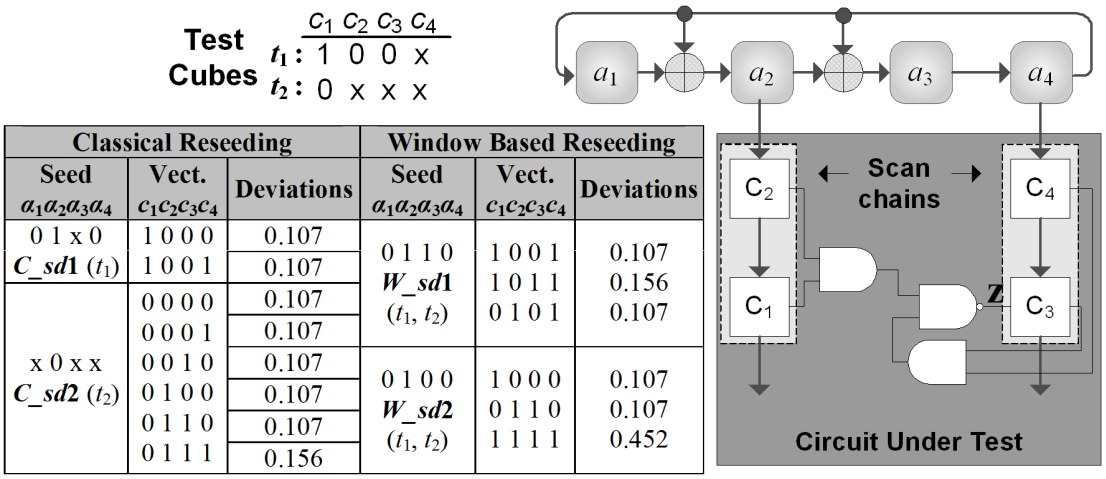


Figure 1: Generic LFSR reseeding architecture

Even though [15] constitutes an effective way to utilize the otherwise wasted variables, it still suffers from limited compression. Window-based reseeding [4] and dynamic reseeding [7, 8, 10] offer considerably better compression than [6], [15] as they manage to exploit very efficiently the seed variables. In the case of window-based reseeding every 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 different test cube, and thus multiple incompatible test cubes (i.e., test cubes that differ in at least one of their specified bit positions) as well as multiple compatible test cubes can be encoded at the same window (i.e. seed). Moreover, by carefully encoding each test cube at the window position that requires the replacement of the fewest variables, the probability of encoding additional test cubes at the same seed using the remaining variables increases. In the case of dynamic reseeding high compression is achieved by continuously injecting test data from the ATE channels into the decompressor, through linear (exclusive-or) operations with the current data of the decompressor. Thus the decompressor is not flushed and the unspecified variables remain inside the decompressor and they are exploited at a later step for encoding other test cubes.

Both window-based and dynamic reseeding techniques exploit almost all variables in order to reduce the seed volume, thus they offer high compression. Moreover, [15] requires all the candidate seeds for all the test cubes to be computed before the selection process begins, which is not feasible in window-based and dynamic reseeding. Therefore, it is clear that the approach of [15] cannot be used in these cases. To overcome this problem, a different approach is followed in this report, which efficiently compresses the test cubes, and also provides high defect coverage of the resulting vectors. The main idea of the new method is to generate multiple candidate seeds that implement different unique encodings of the test cubes. The encoding of each candidate seed is different from the encodings of the other candidate seeds, therefore, the probability of generating a vector with high output deviation values increases. At the same time, high compression is ensured by intelligently generating the candidate seeds in such a way that best exploits the variables for decreasing the seed volume. Let us see an illustrative example.

Example 1. Consider the circuit shown in Figure 2, which consists of two scan chains loaded



Classical Reseeding			Window Based Reseeding		
Seed $\alpha_1\alpha_2\alpha_3\alpha_4$	Vect. $c_1c_2c_3c_4$	Deviations	Seed $\alpha_1\alpha_2\alpha_3\alpha_4$	Vect. $c_1c_2c_3c_4$	Deviations
0 1 x 0	1 0 0 0	0.107	0 1 1 0	1 0 0 1	0.107
C_sd1 (t_1)	1 0 0 1	0.107	***W_sd1***	1 0 1 1	0.156
	0 0 0 0	0.107	(t_1, t_2)	0 1 0 1	0.107
	0 0 0 1	0.107			
x 0 x x	0 0 1 0	0.107	0 1 0 0	1 0 0 0	0.107
C_sd2 (t_2)	0 1 0 0	0.107	***W_sd2***	0 1 1 0	0.107
	0 1 1 0	0.107	(t_1, t_2)	1 1 1 1	0.452
	0 1 1 1	0.156			

Figure 2: Example of classical and window-based LFSR reseeding

using a 4-bit LFSR. The initial state of the LFSR is $\alpha_1\alpha_2\alpha_3\alpha_4$. Let us first encode test cubes t_1, t_2 shown in Figure 2 into seeds using [15]. Since test cubes t_1, t_2 are incompatible (they differ at the specified test bit corresponding to c_1), they can not be encoded by the same seed, and consequently two separate seeds are required. By solving the system of equations for t_1, we compute the seed $\alpha_1\alpha_2\alpha_3\alpha_4 = 01x0$ (denoted as C_sd1 in the Table of Figure 2). Note that α_3 is a free variable and it can be replaced by either logic value '0' or '1' providing thus two candidate seeds $\alpha_1\alpha_2\alpha_3\alpha_4 = 0100$ or 0110. In the same way we compute seed C_sd2 for encoding t_2 and we calculate the eight candidate seeds by replacing the three free variables with all eight possible binary combinations. The test vector applied to the circuit for each of these seeds and the respective output deviations are shown in Columns 2, 3 of the Table (the output deviations are computed as in [15]). By selecting the candidate seed corresponding to the first vector in the case of seed C_sd1 and the last vector in the case of C_sd2 the maximum output deviation value achieved is equal to 0.156. Now let us apply the window-based encoding for window size equal to 3. In this case, higher compression can be achieved because both test cubes t_1, t_2 can be encoded into a single seed. For example t_1 can be encoded at the first position of the window, and t_2 can be encoded at the third position. However, the computed seed (labeled as W_sd1) has no free variables and thus it cannot provide any candidate seeds. We can easily see though, that t_2 can be also encoded at the second position of the window (W_sd2 in Table 1), which provides a second candidate seed offering the same compression as the first one (i.e. a single seed suffices to encode both test cubes in this case too). However, it is obvious from the output deviation values of the respective test vectors of both seeds (sixth column of table), that by selecting W_sd2 instead of W_sd1, the maximum value of output deviation increases from 0.156 to 0.452. ■

It is therefore obvious that different window-based encodings yield similar results in terms of seed volume, but they exhibit significant variations in terms of output deviation values and potentially of defect coverage. In fact these variations are more significant than in [15]. This is because the encoding of different combinations of test cubes into a candidate seed affects the generated vectors much more than the random replacement of the free variables.

Finally, we emphasize that the metric proposed in [15] is not efficient since it ignores important parameters such as the structure of the circuit under test and the output deviations of previously selected seeds. A more efficient output deviation-based metric was proposed in [5] for generating test sets with high defect coverage. This metric takes into account structural information of the circuit under test in order to further increase the defect coverage. A major limitation of this metric is that it evaluates each test vector for either timing-independent or timing-dependent defects. In this report, we use a new deviation based metric for grading test patterns in order to evaluate the unmodeled defect coverage of a set of test vectors.

3 A Deviation-based Metric for Time-related Defects

In this section we present the proposed output deviation-based metric for evaluating a candidate seed s (we present the metric assuming that the s is calculated using window-based reseeding, as the extension to the dynamic reseeding case is straightforward). 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, r_2. In other words we assume the Launch-On-Capture (LOC) technique as it is common in industry. A metric is described based on output deviations in order to grade window-based LFSR encoding seeds. During window-based LFSR encoding a seed generates w vectors (contrary to the other LFSR based techniques). So, it is not clear how a seed is evaluated using the output deviations metric, which is applied on a single vector. The answer is that the metric is applied to all the vectors inside w simultaneously, as if all w vectors were a single huge test vector. Moreover, the calculation of maximum expected deviations values is done with random assignement. In this report, the metric is applied on linear decompressors and the maximum values should capture any limitations imposed by the linear correlations of the decompressors used. Below, the formulation of the new metric is given:

The Maximum Expected Deviation value for output i at capture cycle r_k ($k = 1, 2$) and fault-free response v ($v = 0, 1$), denoted as $MED(i, r_k, v)$ is an estimate of the maximum deviation value expected throughout the seed-computation process on output i when its fault-free response is v at capture cycle r_k. It is calculated as follows: initially, for every test cube, a predetermined number of single-vector seeds (i.e., seeds encoding only one test cube) are generated by randomly replacing 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 capture cycles r_1 and r_2. The output-deviation values of all generated test vectors are calculated and the greatest value for every output i and for each fault- free response $v = 0, 1$ at capture cycle r_k constitutes $MED(i, r_k, v)$. After calculating the $MED(i, r_k, v)$ values, the generated single-vector seeds are discarded.

With the use of the MED values, the evaluation of the candidate seeds at each step of the seed computation process is done as follows. 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 maximum if it is very close to $MED(i, r_k, v)$, or equivalently, if the following inequality is true:

$$D(s, j, i, r_k, v) \geq F_1 \cdot MED(i, r_k, v), \quad v = 0, 1 \tag{1}$$

F_1 is a real-valued parameter that must be close to 1 for selecting seeds with output deviation values that are very close to the maximum expected deviation. However, a value of $F_1 = 1$ must be avoided since sometimes it results in a failure to select seeds (the predicted $MED(i, r_k, v)$ value becomes hard to reach during the selection of the seeds). As in the case of [5] we also verified that a value of F_1 in the interval $[0.99, 0.995]$ provides high-quality seeds in all cases, and for our experiments we set $F_1 = 0.995$.

The second task of the evaluation process is to rank all outputs according to their potential of observing errors due to defects. Every output i is assigned two pairs of weights $wo(i, r_k, 0)$, $wo(i, r_k, 1)$ for $k = 1, 2$, 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 both fault-free responses 0 and 1 at output i during both capture cycles r_1, r_2. The set of weights $\{wo(i, r_1, 0)$, $wo(i, r_1, 1)$, $wo(i, r_2, 0)$, $wo(i, r2, 1)\}$ and the output deviation values are used during the evaluation of the candidate seeds for determining a weight $WS(s)$ for every candidate seed s as follows: assume that seed s is expanded into a window of w test vectors, and let j be one of the w window positions, i.e., $j \in [1, w]$. Let the number of observable outputs in the circuit be k. For test vector j, the sets $MS[s, j, r_1, 0]$, $MS[s, j, r_1, 1]$, $MS[s, j, r_2, 0]$, $MS[s, j, r_2, 1]$ consist of all outputs i, with $1 \leq i \leq k$, for which any of the deviation values $D(s, j, i, r_1, 0)$, $D(s, j, i, r_1, 1)$, $D(s, j, i, r_2, 0)$, $D(s, j, i, r_2, 1)$ satisfy inequality 1.

Finally, for evaluating each candidate seed, the sum of its weights is calculated using the formula:

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

The above formula means simply 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_1, r_2 (i.e., those belonging to $MS[s, j, r_1, 0]$,$MS[s, j, r_1, 1]$, $MS[s, j, r_2, 0]$, and $MS[s, j, r_2, 1]$) participate into the final weights sum $WS(s)$. Note that the first response targets the timing-independent defects, while 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 as well as 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. The larger the cones are the greater is the probability of detecting unmodeled defects. 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. In that way, the outputs with reduced weight have much smaller impact on the selection of the next seeds. This is motivated by the fact 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. Thus, 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 have chosen the value of F_2

to be equal to 8, as we verified experimentally that a value of F_2 in the interval $[2, 10]$ is sufficient to maximize the deviations at all outputs.

4 Generation of Defect-Aware Seeds

In this section, we first describe the encoding algorithm for defect-aware window-based LFSR reseeding. Next we discuss the special case of window-based reseeding with window size $w = 1$, and the defect-aware dynamic reseeding method.

4.1 Window-Based Reseeding

The window-based reseeding approach proposed in [4] attempts to maximize compression by using the following very effective encoding criterion:
Compression Maximization Criterion: "The first test cube encoded by every seed is the one with the highest number of specified bits, and it is encoded at the first window position. The next most-specified test cube is then encoded at the window position that results to the replacement of the minimum number of variables. If more than one such test cubes exist, the test cube that requires the replacement of the fewest variables is encoded. This continues until no system for any of the unencoded test cubes can be solved in the same window."

This criterion efficiently exploits all the properties offered by window-based reseeding [2,3]. In addition, it attempts to increase the number of densely specified test cubes encoded by every seed, which, as shown in [4], tends to decrease the overall seed volume even more. These are all major differences with other strategies used in the literature for encoding test cubes, e.g., the incremental solver proposed in [10].

In the method presented in this report, two objectives are simultaneously addressed: the efficient compression of test cubes, and the high defect coverage of the resulting patterns. High defect coverage is targeted by generating candidate seeds implementing different unique encodings of test cubes. For the selection of every seed, T candidate seeds (T is a user-defined parameter) are first generated and then evaluated using the output- deviation-based metric presented in section 3. The best candidate seed according to this metric is selected each time. High compression is ensured by carefully generating the candidate seeds using a new encoding criterion which ensures that all candidate seeds provide nearly the same level of compression that is obtained if the seed is generated using the compression-maximization criterion (i.e., according to [4]).

The generation of the T candidate seeds is done as follows: we start by encoding the most-specified test cube (say t_1) in the first position of the corresponding window. Next, for initiating the generation of the T different candidate seeds, we independently apply the compression-maximization criterion T times in that window, 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 rest of the 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

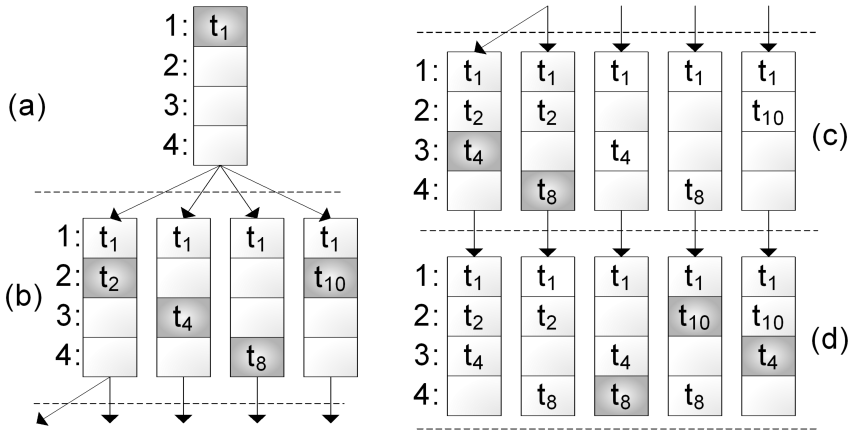


Figure 3: An example to illustrate the generation of T candidate seeds

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. Finally, the T generated candidate seeds are evaluated using formula 2, and the most promising one for increasing the defect coverage is selected. Then, the test cubes encoded in its window are dropped and the seed-computation process continues in the same way for selecting the next seed. We provide insights into the above process with the following example.

Example 2. Let $t_1, t_2, \ldots, t_{10}$ be 10 test cubes sorted in descending order according to their number of specified bits, $w = 4$ be the window size, and $T = 5$ be the number of candidate seeds. In Figure 3, 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 test cube t_1 (the most specified one) in window position 1 (Figure 3a). 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-maximization criterion, t_4 is the next selection, i.e., if we exclude t_2, and so on). As a result, we initiate the generation of four new candidate seeds (Figure 3b) 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}). Since though, the upper limit of $T = 5$ candidate seeds has not been reached yet, we continue and attempt to encode a third test cube in the windows generated so far. This is shown in Figure 3c. 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. Therefore, the expansion of the tree (i.e., 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 3d). Finally, we have generated five candidate seeds $s_1, \ldots, s_5$ which are subsequently evaluated using formula 2. Note that the leftmost seed (s_1 in this case) provides the best compression. Assuming though that seed s_3 has the highest weight among all seeds according to 2, s_3 is selected and test cubes t_1, t_4 and t_8 are dropped from the set of test cubes to be encoded. ■

In contrast to [4], we examine various encoding options, apart from the one that maximizes compression (i.e., t_1 along with t_2 and t_4 used in the previous example). Thus, several choices are available for maximizing defect coverage. 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.

Note that by generating multiple candidate seeds with diverse encoding, we cannot always guarantee an increase in defect coverage. However, we experimentally found that among the candidate seeds, there exist seeds that increase defect coverage, and these seeds are effectively identified and selected by the metric presented in Section 3. In addition, note that by trying different encodings, the complexity of the encoding process increases. However, we found in our work that even a small value of T can provide significant increase in the defect coverage offered by the resulting seeds, and thus the encoding process is feasible for large circuits. This can be easily concluded with the following analysis. Suppose that the proposed deviation based enhancement method is applied on an encoding technique with complexity $O(N)$, where N is the size of the test set that needs to be encoded. The proposed method adds a multiplier factor T at the complexity of the encoding method that is applied on, and the new complexity becomes $O(T \cdot N)$. The gain in unmodeled defect coverage saturates very fast as T increases. As a result, an efficient gain can be achieved for very small values of T (experiments show that small values of T even with $T \leq 30$ can almost maximize that gain and, so, we selected $T = 30$ for all our experiments). Given that, the proposed method can be treated as adding a constant complexity factor to the complexity of the encoding method and consequently the complexity remains unaffected.

After all the seeds are generated, they are sorted according to their potential to detect defects. Seeds with higher potential are loaded first in the LFSR in order to detect defects as quickly as possible and thus to decrease the test application time in an abort-at-first-fail environment. The ranking of the seeds is based on an evaluation process that is similar to the T candidate seeds evaluation procedure. The difference lies in the fact that this procedure is now applied to all the selected seeds, and not to candidate seeds. Moreover, 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

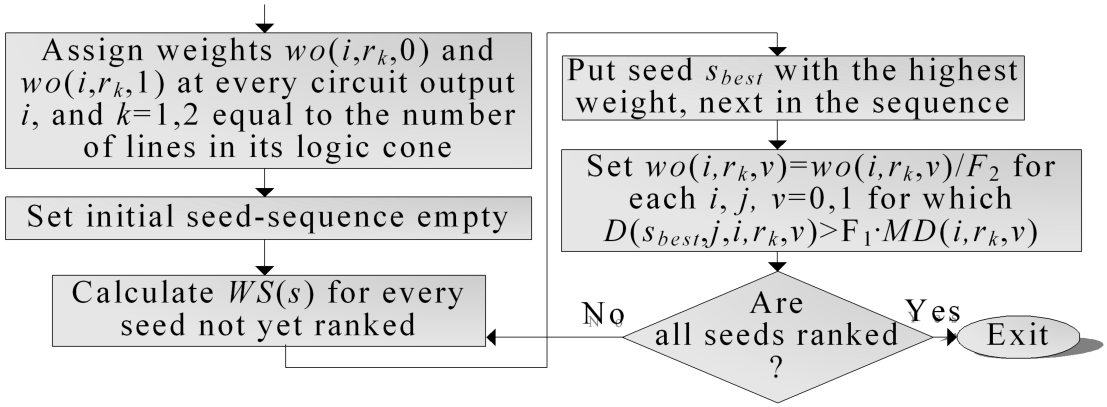


Figure 4: Final ranking of the selected seeds

(it is the largest among the output-deviation values of all test vectors generated by all calculated seeds). Equation 2 is applied in this case too, but this time 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. A flowchart for this final ranking of the seeds is shown in Figure 4.

4.2 Classical Static LFSR Reseeding and Dynamic Reseeding

One of the advantages 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. 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 aware compression-maximization criterion presented in the previous 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 [6] (and adopted in [15]), only one test cube is encoded by each seed, whereas in window-based reseeding for $w = 1$, the utilization of the defect-aware compression-maximization criterion offers an efficient way to combine more than one compatible test cubes in the same encoded pattern. Thus, as will be shown in the experimental section, the volume of the defect-aware seeds is low and their quality is high for $w = 1$ as well.

Dynamic reseeding resembles window-based reseeding for $w = 1$, in the sense that they both generate one test vector per seed. Thus, even though the way in which the systems of equations are formed is different from static reseeding, the criterion for generating the T candidate dynamic seeds can be applied in this case too. However, using this criterion, the seeds are generated in no particular order of effectiveness in terms of defect coverage. Most of the time, a seed selected near the end of the selection process may be more efficient than

a seed near the beginning. In the case of static window-based reseeding, the final sorting of the generated seeds, according to their output deviations (Figure 4) solves this problem and provides steep defect coverage ramp-up. Note that in the case of static reseeding, there is no dependency between the static seeds, because each seed flushes the contents of the LFSR; therefore, sorting of the seeds is possible. However, in the case of dynamic reseeding, reordering of the dynamic seeds is not possible since the LFSR is never flushed. To provide both high defect coverage and steep coverage ramp-up in dynamic reseeding, we allow a small reduction in compression offered by the T-candidate seeds generation process, in order to facilitate the generation of high-quality seeds. Specifically, instead of encoding the most-specified test cube as the first test cube of every candidate seed, we select the T most-specified test cubes which have not yet been encoded. Each one of these test cubes is encoded as the first test cube of each of the corresponding T candidate seeds. Consequently, every candidate seed encodes a different test cube as its first test cube. Then, for each candidate seed, we continue by encoding the test cubes providing the highest compression (i.e., the most specified ones that also require the replacement of the fewest variables), excluding all the T test cubes selected at the first step.

This modification allows us to increase the likelihood of generating high- quality candidate seeds as early as possible but it can also potentially reduce the amount of test compression. However, in the modified criterion, the candidate seeds are still among the most efficient ones in respect to the achieved compression. Experimental results show that the reduction in compression is infrequent and very small.

Very frequently during the candidate-seed generation process, a single test cube should be selected from a subset of equivalent, according to the Compression Maximization Criterion, test cubes (i.e., test cubes that include the same maximum number of specified bits and, at the same time, their encoding requires the replacement of the same minimum number of variables). In most of these cases, only one of them can be encoded, because the selection of any such test cube prevents the encoding of the others in the same seed (i.e., after any one of them is encoded the rest become un-encodable). We exploit this property to increase the quality of the candidate seeds without sacrificing compression. Specifically, during the generation of every candidate seed, the first time that a set of test cubes, say ST, is found with the above property, we select m of them (m is a predetermined parameter) and we separately encode them in the candidate seed. Thus the candidate seed is replaced by m new ones, and each one of them embeds all the test cubes of the initial candidate seed (i.e., the one that we replace with the m new ones) as well as one of the test cubes of set ST. Note that this is done only for the first (and consequently most- specified) m test cubes found for each one of the initially generated T candidate seeds, in order to keep the candidate-seeds' volume low. To bound the number of candidate-seeds' volume, we set the maximum value of m equal to 2. Thus, the volume of generated candidate seeds cannot exceed $2 \cdot T$, which is relatively small.

*Example 3.*Let $t_1, t_2, \ldots, t_{10}$ be 10 test cubes sorted in descending order according to their number of specified bits, $T = 3$ be the number of candidate seeds, and $m = 2$. Figure 5 presents the various steps of the encoding process. Each dynamic seed encoding test cubes t_a and t_b is denoted as $s(a, b)$. At the first step (Figure 5a) test cubes t_1, t_2, t_3 are selected (they are the most specified ones) and the candidate seeds $s(1)$, $s(2)$, $s(3)$ are determined. Next we proceed with the seed $s(1)$ and we select the most specified test cubes (excluding

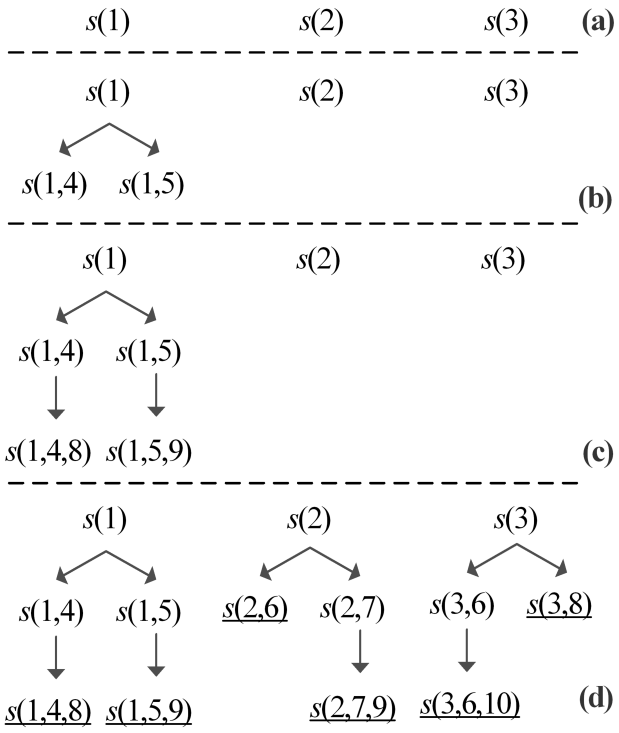


Figure 5: Illustration of the generation of candidate seeds for dynamic reseeding

tC_2, t_3) that can also be encoded by this seed. Let cubes t_4, t_5 and t_6 have the same number of specified bits. Suppose that they also require the replacement of the same number of variables, and let all three of them be separately encode-able (only one at a time) at seed $s(1)$. Since $m = 2$ we select the first two among them, namely t_4, t_5. Thus candidate seed $s(1)$ is replaced by candidate seeds $s(1, 4)$ and $s(1, 5)$ – see Figure 5b. We proceed with each one of them separately by encoding cubes t_8 and t_9 at seed $s(1, 4)$ and $s(1, 5)$, respectively – Figure 5c. At this point, the generation of the first two candidate seeds terminates and we proceed with the next candidate seed $s(2)$ in the same way. Finally, six (equal to the upper limit of $2 \cdot T$) candidate seeds are generated. These seeds are underlined in Figure 5d. ■

Note finally that in dynamic reseeding, each dynamic seed may not be fully specified. Some variables may remain unspecified and are utilized by the next seeds. However, in order to apply relation 2 for computing the output-deviation metric of each candidate dynamic seed, every unspecified variable has to be assigned either the logic '0' or '1' value. To overcome this problem, the encoding estimates the output deviation values by temporarily replacing these variables randomly, and based on these estimates, it selects the best dynamic candidate seed. Then it removes the random assignment from the variables (i.e., they become unspecified again) and proceeds to the computation of the next dynamic seed. In this way, compression is not compromised as the variables are utilized only for encoding test cubes, whereas, at the same time, a good estimate of the defect-screening potential of each candidate seed is obtained.

5 Fault simulation Results

In this section, we evaluate the effectiveness of the defect- aware reseeding methods. The simulation platform was developed using the C programming language, and all ATPG and fault simulations were carried out using commercial tools. We conducted experiments using the largest ISCAS'89 circuits and a subset of the IWLS'05 circuits [1]. The number of scan chains was set equal to 30 for the ISCAS circuits, 50 for the medium sized IWLS circuits, and 100 for the large ethernet IWLS circuit. For evaluating the window-based reseeding method, we considered two window sizes, $w = 1$ and $w = 5$. For each benchmark circuit, a dedicated LFSR with a characteristic primitive polynomial of near minimum size was selected following the $s_{max} + 20$ rule [6].

For evaluating the dynamic reseeding method, we conducted experiments with various ATE-channel volumes (the best results are reported). In addition, the LFSRs used in window-based reseeding were replaced by ring generators of the same size in dynamic reseeding. In the rest of the section, two cases are reported for both reseeding approaches: a) the case, denoted as "Cmp" which refers to the encoding that targets only compression (the original approach without applying the proposed enhancement method), and b) the case noted as "Cmp & Def", which refers to the encoding that targets compression and defect coverage at the same time. For the "Cmp & Def" case, T was set to 30, m was set to 2 (m is used only in dynamic reseeding) and the constants F_1 and F_2 were set equal to 0.995 and 8 respectively.

To demonstrate the advantage of the proposed method compared to the classical reseeding-based method of [15], which uses a different output deviation-oriented metric, we have implemented the method of [15] as well as the defect-unaware classical reseeding method [6]. We conducted experiments for these two methods using compacted test sets generated by the same commercial ATPG engine used for the rest of the experiments. Note that in contrast to the other methods (window-based and dynamic reseeding), for the classical LFSR reseeding approaches of [6] and [15] we used compacted stuck-at test sets in order to minimize both the number of required seeds as well as the test-sequence length. These methods are not accompanied by a dynamic compaction technique. So, their TSL is the same with the size of the test set. As a result, these methods exhibit the best TSL and TDV when they are applied on compacted test sets.

In Table 1, we present the test-data volumes in Kbits (1Kbit $=10^3$ bits) for the window-based and dynamic reseeding methods, as well as for the classical (static) reseeding approaches. The first column lists the names of the benchmark circuits. The next column presents the size of the compacted test set used for the evaluation of both the classical defect unaware LFSR reseeding method ([6]) and the method proposed in [15]. The third column presents the test data volume for these two methods, which is the same for both of them (note that [15] differs from the classical LFSR reseeding approach only in the way that the free variables are filled and does not impact TDV). The next three pairs of columns present the test data volumes of the $w = 1$, $w = 5$ and dynamic reseeding cases, in their defect-unaware versions ("Cmp") as well as in their proposed defect-aware versions ("Cmp & Def").

As it is obvious from Table 1, window-based and dynamic reseeding clearly outperform the classical static reseeding approaches ([6] and [15]), while the highest compression is always achieved by window-based reseeding for $w = 5$. Dynamic and window-based

Table 1: Test data volume results (in Kbits)

Circuit	Classical Reseeding		Window-Based Reseeding				Dynamic Reseeding	
			$w = 1$		$w = 5$			
	Test Set Size	[6, 15]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	28.7	16.1	8	8	6.2	6.3	7.6	7.9
s9234	41	23.2	16.9	18.4	14.2	14.3	17.3	17.6
s13207	188.3	78	12	12.8	8.2	8	15.1	15.5
s15850	99	47	18.6	19	13.6	14	16.2	16.7
s38417	238	117.3	64.6	65.4	58.2	59.7	65.6	68.4
s38584	270.8	148	34	34	27.2	26.9	42.3	41.6
ac97_ctrl	148.7	68.6	11	10.9	7.2	7.3	13.9	13.7
mem_ctrl	720	373.9	113.9	117.8	79.7	86.6	126.8	128.8
pci_bridge	1160.6	343.2	111.4	110.3	100.2	99.7	123	122.9
tv80	281.6	151.4	99.8	102.5	54.3	55.7	81.4	82.7
usb_funct	252.7	129.2	57.5	57.4	48.9	49.4	55.1	56.2
ethernet	11.8×10^3	1.7×10^3	203.8	225.5	162.5	165.1	231.3	233.1

reseeding for $w = 1$ provide comparable results. The most important observation though is that for both the window-based and the dynamic reseeding method, the proposed defect-aware encoding (columns labeled "Cmp & Def") provides nearly the same compression as the original defect-unaware encoding (columns labeled "Cmp"). In a few cases, the proposed defect-aware encoding provides even better compression than the original defect-unaware encoding. We attribute this result to the window-based encoding criteria, which consist a heuristic encoding approach and they do not offer an optimal solution. As a result, arbitary changes on the encoding order caused by the proposed method may result to better compression results. Nevertheless, it is obvious that the utilization of the output-deviation metric has no significant adverse impact on compression for both static and dynamic reseeding methods.

Table 2 presents the test-sequence lengths (TSLs) 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 [6] and [15]). The next three pairs of columns present the TSLs of the $w = 1$, $w = 5$ and dynamic reseeding cases, in their defect unaware versions ("Cmp") as well as in their proposed defect aware versions ("Cmp & Def"). As expected, the classical, dynamic, and window-based reseeding for $w = 1$ offer short and comparable, in many cases, test sequence lengths. Note that the TSLs of the classical reseeding approaches are shorter than those of window-based reseeding for $w = 1$, due to the use of compacted test sets in the former case. As expected, the TSLs of window-based reseeding for $w = 5$ are greater than those of the other methods, due to the larger value of w. However, this can be also attributed to the small LFSR sizes used here. Larger LFSRs offer considerably shorter TSLs (due to the smaller number of calculated seeds) with minimal impact on compression. For example, if we increase the size of the LFSR used for the pci_bridge circuit from 90 bits to 200 bits, the test-sequence length decreases form 5565 vectors to 2695 vectors, whereas the compressed test data volume has

Table 2: Test sequence length results (# vectors applied)

Circuit	Classical Reseeding [6], [15]	Window-Based Reseeding				Dynamic Reseeding	
		w=1		w=5			
		Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	134	199	199	770	785	232	243
s9234	166	282	307	1185	1195	317	324
s13207	269	300	320	1030	1005	313	322
s15850	162	310	316	1130	1170	385	396
s38417	143	808	818	3635	3730	585	611
s38584	185	485	485	1945	1920	288	283
ac97_ctrl	66	274	273	895	910	151	149
mem_ctrl	603	876	906	3065	3330	879	894
pci_bridge	330	1238	1226	5565	5540	867	866
tv80	757	1663	1708	4525	4645	1693	1719
usb_funct	136	959	956	4075	4115	724	739
ethernet	1111	2912	3222	11610	11790	2155	2182

only a limited increase from 100.2 Kbits, to 107.8 Kbits. It is obvious that as the size of the LFSR increases, the test application time drops considerably, while at the same time the compression is not significantly affected. In our experiments we selected the size of the LFSRs based on the $s_{max} + 20$ rule [6], where s_{max} the number of defined bits of the most specified test cube.

For evaluating the effectiveness of the proposed defect-aware reseeding methods for defect screening, we consider the coverage of unmodeled faults, namely transition and bridging faults, obtained by applying to the circuit under test the test vectors generated by the computed seeds. As is common in industry, we use the launch-on-capture (LOC) scheme, also referred to as broadside scan, to apply test-vector pairs. Note that none of these two fault models were targeted by the test sets (they are only used as surrogate fault models). The concept of n-detection has been also used in the literature as a surrogate defect coverage model. However, as shown in [5] n-detection is not always indicative of defect coverage, therefore we do not use this metric in this report. Finally, as mentioned in Section 3, for the proposed reseeding methods ($w = 1$, $w = 5$ and dynamic reseeding) the output-deviation metric considers both the responses of each test vector pair. On the other hand, [15] considers only one response (either the first or the second). Therefore, for generating results using [15], we chose to evaluate the generated seeds using the second response of each test-vector pair. This decision favors the timing-dependent defects of [15] i.e. the transition-fault coverage of its generated patterns.

First we evaluate the proposed encoding with respect to the achieved transition-fault coverage. The corresponding results are shown in Table 3. Columns 2, 4, 6 and 8 present the transition-fault coverage achieved by the classical defect-unaware, window-based (for $w = 1$ and $w = 5$) and dynamic reseeding approaches respectively, while columns 3, 5, 7 and 9 present the transition fault coverage achieved by the classical defect-aware, reseeding of [15] and the proposed defect-aware approaches. We see that in both the window-based and dynamic reseeding, the use of the proposed output- deviation metric increases the transition

Table 3: Transition-fault coverage (%)

Circuit	Classical Reseeding		Window-Based Reseeding				Dynamic Reseeding	
			w=1		w=5			
	[6]	[15]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	61.1	63.49	62.9	66.38	65.66	70.32	63.64	66.2
s9234	40.7	49.63	43.04	53.08	53.94	58.41	45.84	53.52
s13207	62	69.48	62.94	68.28	64.31	70.32	60.18	68.52
s15850	52.8	55.25	53.58	56.95	57.58	58.31	53.87	58.04
s38417	79.2	80.24	85.42	87.93	88.85	90.6	84.99	87.32
s38584	61.5	62.21	65.03	66.32	68.1	69.07	63.25	64.02
ac97_ctrl	42.7	45.6	47.18	56.42	52.4	63.95	45	50.81
mem_ctrl	41.1	44.24	42.69	46.01	44.03	47.36	43.32	45.72
pci_bridge	65.2	69.5	77.39	85.8	82.96	87.5	73.78	82.97
tv80	53.8	59.31	60.16	64.76	61.97	64.9	59.44	62.45
usb_funct	63.2	64.49	71.4	75.53	74.53	79.39	70.01	74.2
ethernet	47.6	49.56	53.94	63.79	71.37	83.14	50.75	54.92

fault coverage significantly. Compared to [6], the method described in [15] achieves higher transition- fault coverage. Moreover, in nearly all cases, the proposed window-based and dynamic reseeding approaches offer higher transition-fault coverage than [15].

It is obvious from Table 3 that the defect coverage achieved by the proposed method for $w = 5$ is higher than the defect coverage achieved by the proposed method for $w = 1$ and dynamic reseeding. 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 test sequence length in the case of $w = 5$ contributes also to the increased defect coverage compared to the other cases. However, according to the results shown in Table 3, this contribution is less significant than the contribution of the proposed encoding method. Specifically, in most cases, the defect- unaware window-based reseeding for $w = 5$ offers lower transition fault coverage than the defect-aware window-based reseeding for $w = 1$, even though the test sequences in the former case are much longer.

Figure 6 illustrates the transition fault coverge ramp-up achieved by the window-based reseeding method for $w = 5$ for selected circuits. In each chart, the x-axis presents the number of the applied vector pairs and the y-axis the transition-fault coverage. The seeds for the defect-unaware window-based reseeding method have been sorted: a) randomly (curves "Cmp(Rnd)"), and b) in descending order of their stuck-at-fault coverage (curves "Cmp(Stuck)"). The curves "Cmp & Def" correspond to the proposed defect-aware window-based reseeding method. It can be seen that the defect coverage of the "Cmp & Def" method is considerably higher than that for the other methods. Moreover, the proposed method exhibits higher coverage ramp-up than both the other methods, with the "Cmp(Stuck)" being better than the "Cmp(Rnd)". Finally, for the largest benchmark ethernet, which consists of 136.2K gates and 10.5K scan flip flops and is more representative of real-life industry circuits, the improvement in transition-fault coverage is striking. We have also verified that the "Cmp & Def" method in the case of window-based reseeding for

$w = 1$ exhibits also higher ramp-up than the "Cmp" method.

Figure 7 shows the coverage ramp-up achieved by the dynamic reseeding method for the same circuits reported in Figure 6. As we can see, the proposed defect- aware method offers steeper coverage ramp-up than the baseline defect- unaware approach for the dynamic reseeding case as well.

The transition-fault coverage (or the coverage of any other fault model) can be further improved by using ATPG to generate top-off test cubes, and by subsequently compressing these test cubes using either static or dynamic reseeding. The advantage offered by this strategy, when combined with the proposed encoding method is twofold: first, the encoding of the baseline stuck-at test cubes using the defect-aware encoding will cover a large number of the targeted faults (i.e., the transition faults in our case) and thus the number of generated top-off test cubes will be relatively small. Second, if the encoding of the newly generated top-off test cubes is properly tuned using the proposed output-deviation metric, the generated seeds for them will offer high coverage of other unmodeled defects.

In our final experiment, we evaluate the proposed method and [15] in terms of the achieved bridging-fault coverage. For evaluating the examined reseeding methods in terms of their bridging fault coverage, both the BCE^+ metric and the random bridging fault coverage were used.

The bridging fault coverage cannot be accurately measured since the set of bridging faults is huge and not all of them are equally possible. Bridging fault coverage can only be accurately (and reasonable) estimated when layout information is available (after the routing of the interconnections during the final steps of a designing process). Using layout information, it is feasible to isolate the most possible pairs and drop the complexity of estimating accurately the bridging fault coverage. However, this approach is infeasible during test generation, because that is taking place during the early stage of designing process, when layout information is not yet available. As a result various metrics and methods have been proposed to measure the bridging fault coverage during the early stages of a designing process. Although these metrics do not offer an accurate estimation of the bridging fault coverage, they are very useful for comparison purposes. In this dissertation we used the following two approaches:

- **The Bridging Coverage Estimation (BCE^+) metric:** In [13] a metric has been proposed for evaluation of tests in terms of their achieved bridging-fault coverage. That BCE^+ metric is:

$$BCE^+ = \sum_{i=1}^{n} \frac{f_i^{sa-v}}{|F|} \cdot \left[\sum_{j=1}^{|S|} \frac{1}{|S|} \left(1 - (1 - p_{j,v})^i\right) \right]$$

  where $v = 0, 1$. The parameter f_i^{sa-v} refers to the number of stuck-at-0 faults (for $v = 0$) and stuck-at-1 faults (for $v = 1$) that are detected i times by the test vectors (n is the maximum number of detections for any stuck-at fault). $|S|$ is the number of circuit lines, $|F|$ is the total number of stuck-at faults and $p_{j,v}$ is the probability of signal j to receive the logic value 0 (for $v = 0$) and 1 (for $v = 1$). As noted in [13], BCE^+ is not very accurate for estimating the real bridging fault coverage of a method, but it is very useful for comparing two different methods (the method with the highest value of BCE^+ is deemed to be more effective for defect screening).

Table 4: BCE^+ and random bridging-fault coverage results (%)

Circuit	BCE^+								Random Bridging Faults							
	Classical Reseeding		Window Based Reseeding				Dynamic Reseeding		Classical Reseeding		Window Based Reseeding				Dynamic Reseeding	
			w=1		w=5						w=1		w=5			
	[6]	[15]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def	[6]	[15]	Cmp	Cmp & Def	Cmp	Cmp & Def	Cmp	Cmp & Def
s5378	95.06	95.22	95.49	95.91	95.77	96.4	95.48	96.01	94.14	94.35	94.85	95.19	95.72	96.26	94.99	95.32
s9234	87.66	87.39	88.76	89.11	88.61	89.08	89.1	89.22	86.56	86.58	87.95	88.29	88.7	89	88.05	88.37
s13207	92.85	92.93	92.96	93.78	93.43	94.15	92.81	93.2	91.99	92.14	92.08	92.95	92.92	93.57	91.82	92.14
s15850	94.2	94.17	94.45	94.58	94.58	94.76	94.5	94.62	93.47	93.59	94.38	94.51	94.71	94.89	94.45	94.54
s38417	98.2	98.1	98.29	98.56	98.27	98.48	98.43	98.6	97.13	97.15	97.88	98.15	98.26	98.44	97.85	98.03
s38584	90.36	90.35	91.36	91.56	92.43	92.81	90.81	91.04	89.85	89.91	90.89	91.09	91.67	91.98	90.41	90.52
ac97_ctrl	94.66	94.57	97.96	98.27	98.71	98.92	97.33	97.45	97.02	97.02	98.75	98.87	99.1	99.23	98.42	98.49
mem_ctrl	62.34	62.33	63.3	63.62	64.33	64.74	63.46	63.59	74.6	74.61	75.08	75.44	75.78	76.1	75.2	75.36
pci_bridge	96.04	96.05	98.27	98.46	98.68	98.86	98.13	98.18	96.78	96.82	98.14	98.28	98.45	98.55	98.06	98.06
tv80	91.4	91.37	93.49	93.79	93.7	93.94	93.62	93.78	89.26	89.33	90.86	91.23	91.57	91.74	90.91	91.1
usb_funct	93.71	93.71	95.47	96.1	96.03	96.48	95.38	95.65	95.15	95.19	96.73	97.16	97.17	97.45	96.63	96.83
ethernet	88.78	88.83	92.79	93.66	96.06	96.32	91.63	92.24	90.63	90.77	93.59	94.18	95.57	95.71	92.81	93.17

- **Random bridging fault coverage:** Another approach to compare the bridging fault coverage between two test sets is to use fault simulation against a set of random bridging faults e.g. n pairs of lines can be selected randomly from the CUT. For each pair, four bridging faults are simulated by considering the four dominant-AND/dominant-OR bridging faults $4 \cdot n$ faults are finally simulated. The larger the n the better the estimation on bridging fault coverage.

As noted before, BCE^+ is not very accurate for estimating the real bridging fault coverage of a method, but it is very useful for comparing two different methods (the method with the highest value of BCE^+ is deemed to be more effective for defect screening). Table 4 presents the results. Regarding the proposed window-based and dynamic reseeding approaches, we find that in all cases, both BCE^+ values and random bridging-fault coverage indicate that the proposed defect-aware encoding "Cmp & Def" achieves higher coverage of bridging faults than the original "Cmp" method. In contrast, in the method described in [15], the improvement is small compared to the classical defect-unaware reseeding [6], and in some cases, there is even a decrease in the BCE^+ values. Moreover, all the proposed encoding methods offer higher BCE^+ values as well as bridging fault coverage than [15]. The main reason for this observation is that [15] 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 (i.e., the response of each stuck-at test). This is another weakness of [15], compared to the proposed method, which is able to consider both responses of each pair. Consequently, we conclude that the proposed method improves the bridging fault coverage, which is also a significant advantage over [15].

Finally, we evaluate the execution time of the proposed method. Please note, that the execution time of the proposed method is very fast because its complexity is linear. For a single threaded implementation the overall complexity is $O(C \cdot O(orig))$ where $O(orig)$ is the complexity of the original encoding method that our method is applied on. For very small values of C the expression C can be considered as constant and the complexity is unaffected. Although, higher C values theoritically result to better quality results, our experiments indicate that the quality gain saturates as these values increase. In all our case studies the value of $C = 30$ candidate test vectors achieve a near the maximum quality gain result with very fast execution time (it requires some minutes for all the benchmark circuits and almost an hour for the largest ethernet benchmark circuit). Moreover, the proposed method can be easily parallelized by letting each thread handle a seed. This way the C factor that impacts complexity can be shortened. A parallel implementation with 32 threads on a 4-cores CPU increases by only 4X the execution time for 30 candidates (that would otherwise theoritically increase the execution time by 30X).

6 Conclusions

We have presented a defect-oriented LFSR reseeding technique that allows us to detect unmodeled defects using stuck-at test sets in a test- compression environment. This technique is based on the output-deviations metric for grading the test patterns produced by the LFSR seeds. We have considered both static and dynamic reseeding, and evaluated

unmodeled defect coverage using transition faults and bridging faults as surrogate fault models. Our results show that compared to compression-driven LFSR reseeding, which is largely in use today, higher defect coverage and faster coverage ramp-up are obtained using stuck-at tests and output deviations, without any loss of compression.

Acknowledgement

This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

References

[1] IWLS'05 circts., online: http://www.iwls.org/iwls2005/benchmarks.html.

[2] E. D. Kaseridis, Kalligeros, X. Kavousianos, and D. Nikolos, "Efficient multiphase test set embedding for scan-based testing," in *Inf. Pap. Dig. ETS*, 2005, pp. 147–150.

[3] E. Kalligeros, D. Kaseridis, X. Kavousianos, and D. Nikolos, "Reseeding-based test set embedding with reduced test sequences," in *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, 2005, pp. 226–231.

[4] E. Kalligeros, X. Kavousianos, and D. Nikolos, "Efficient multiphase test set embedding for scan-based testing," in *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on*, 2006, pp. 432–438.

[5] X. Kavousianos and K. Chakrabarty, "Generation of compact test sets with high defect coverage," in *Proc. DATE*, 2009, pp. 1130–1135.

[6] B. Koenemann, "Lfsr-coded test patterns for scan designs," in *Proc. ETS/ETC, VDE Verlag*, 1991, pp. 237–242.

[7] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A smartbist variant with guaranteed encoding," in *Test Symposium, 2001. Proceedings. 10th Asian*, 2001, pp. 325–330.

[8] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial lfsr reseeding," in *Proc. ITC*, 2001, pp. 885–893.

[9] G. Mrugalski, J. Rajski, and J. Tyszer, "Ring generators - new devices for embedded test applications," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 9, pp. 1306–1320, Sept. 2004.

[10] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee, "Embedded deterministic test," *IEEE Trans. Comput.-Aided Des.*, vol. 23, no. 5, pp. 776–792, 2004.

[11] B. Vermeulen, C. Hora, B. Kruseman, E. Marinissen, and R. van Rijsinge, "Trends in testing integrated circuits," in *Test Conference, 2004. Proceedings. ITC 2004. International*, 2004, pp. 688–697.

[12] Z. Wang and K. Chakrabarty, "An efficient test pattern selection method for improving defect coverage with reduced test data volume and test application time," in *Test Symposium, 2006. ATS '06. 15th Asian*, 2006, pp. 333–338.

[13] ——, "Test-quality/cost optimization using output-deviation-based reordering of test patterns," *IEEE Trans. Comput.-Aided Des.*, vol. 27, no. 2, pp. 352 –365, Feb. 2008.

[14] Z. Wang, K. Chakrabarty, and M. Goessel, "Test set enrichment using a probabilistic fault model and the theory of output deviations," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, vol. 1, 2006, pp. 1–6.

[15] Z. Wang, H. Fang, K. Chakrabarty, and M. Bienek, "Deviation-based lfsr reseeding for test-data compression," *IEEE Trans. Comput.-Aided Des.*, vol. 28, no. 2, pp. 259 –271, Feb. 2009.

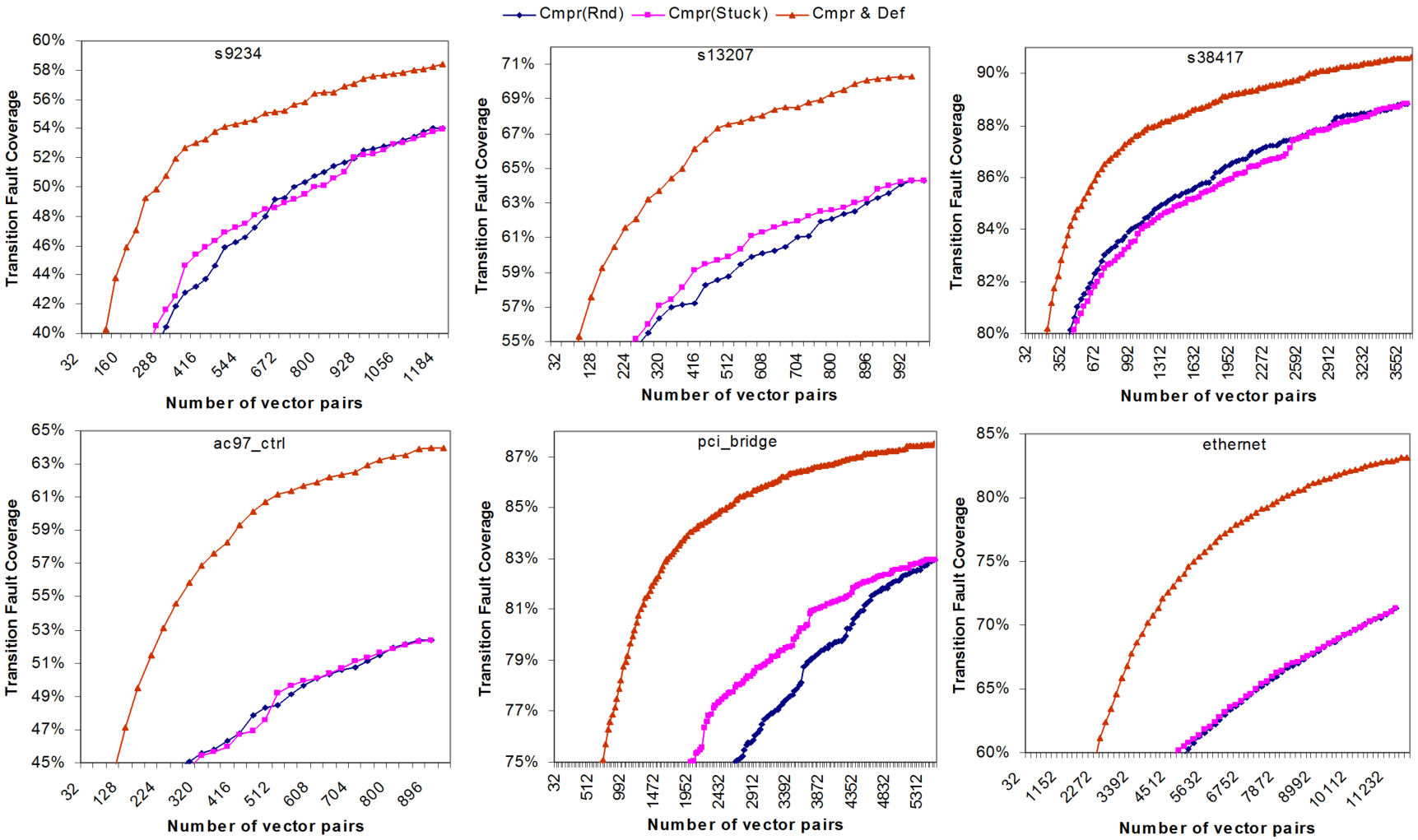


Figure 6: Transition fault coverage ramp-up for window-based reseeding ($w = 5$)

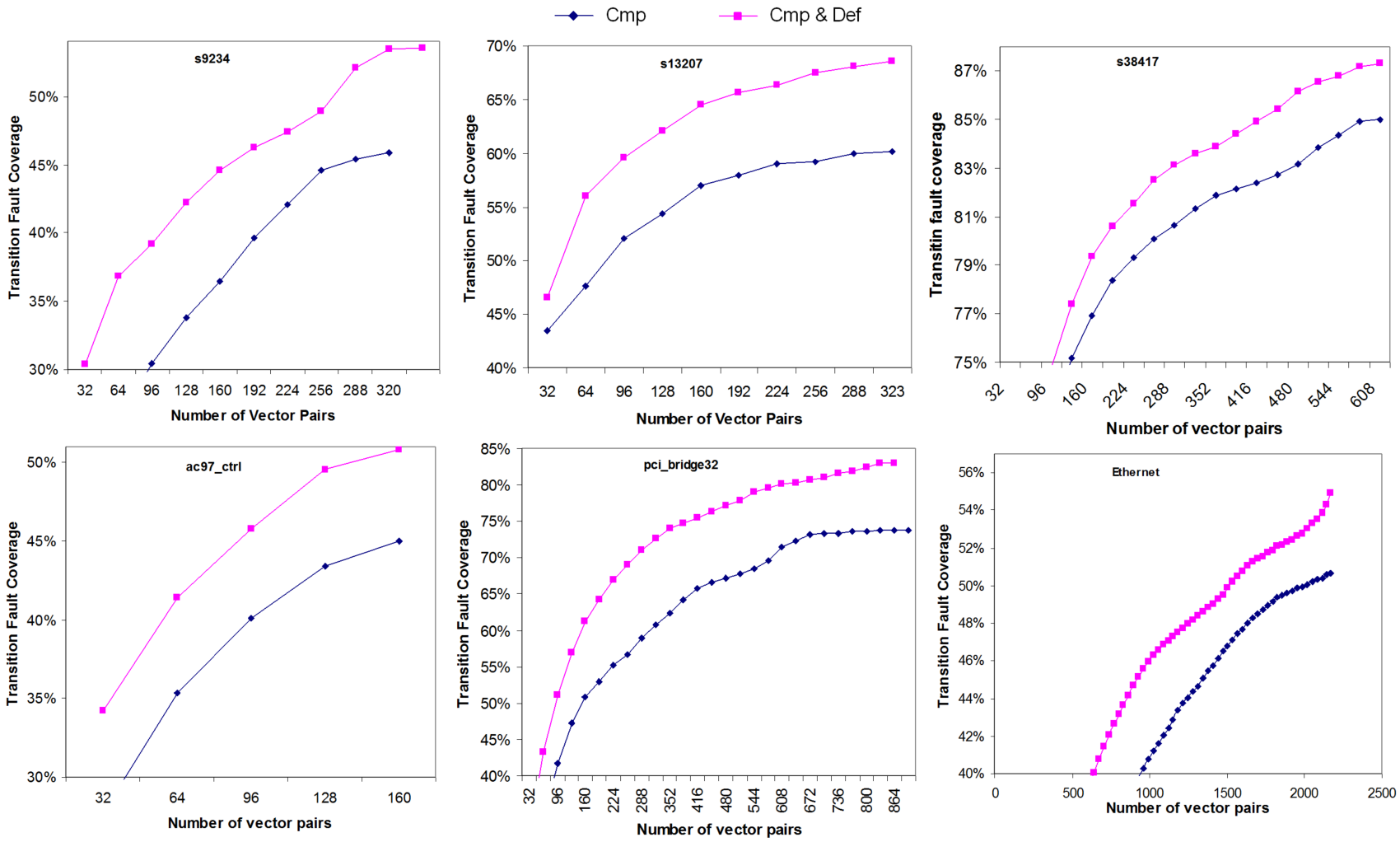


Figure 7: Transition fault coverage ramp-up for dynamic reseeding