Concurrent Fault Detection in Random Combinational Logic

Petros Drineas and Yiorgos Makris
Departments of Computer Science and Electrical Engineering
Yale University

Abstract

We discuss a non-intrusive methodology for concurrent fault detection in random combinational logic. The proposed method is similar to duplication, wherein a replica of the circuit acts as a predictor that immediately detects potential faults by comparison to the original circuit. However, instead of duplicating the circuit, the proposed method selects a small number of prediction logic functions which only partially replicate it. Selection is guided by the objective of minimizing the incurred hardware overhead at the cost of introducing fault detection latency. To achieve this, the proposed method replicates only a reduced width output function for every input combination, yet without compromising the ability to detect all faults. In contrast to concurrent error detection schemes which presume the ability to re-synthesize the circuit, the proposed method does not interfere with the implementation of the original design. As compared to previous approaches, the proposed method achieves significant hardware overhead reduction, while detecting all faults with very low average fault detection latency.

1. Introduction

Concurrent test provides circuits with the ability to self-examine their operational health during normal functionality and indicate potential malfunctions. While such an indication is highly desirable, designing concurrently self-testable circuits which also conform to the rest of the specifications is not trivial. Issues to be addressed include the hardware cost and design effort incurred, performance degradation due to interaction between the circuit and the self-test logic, as well as the level of assurance required. In this paper, we devise a non-intrusive concurrent test methodology for random combinational logic. Non-intrusiveness implies that hardware is only added in parallel to the original circuit, which is assumed to be optimized and may not be modified. The additional logic detects all faults in the circuit, therefore rendering a self-testable design. Moreover, self-test is performed concurrently and does not degrade normal functionality.

Concurrent test is based on the addition of hardware that monitors the inputs and generates an *a priori* known property that should hold for the outputs. A property verifier is utilized to indicate any violation of the property, thus detecting circuit malfunctions. The simplest approach is to duplicate the circuit, imposing an identity property between the original output and the replica output, which may be simply examined by a comparator. With the exception of common-mode failures [1], duplication will immediately detect all errors. However, it also incurs significant hardware overhead that exceeds 100% of the cost of the original circuit.

Since electronic circuits are employed in a wide range of applications, concurrent test methods of various cost and efficiency are required. Towards this end, we devise a concurrent fault detection method for random combinational logic that reduces hardware overhead at the cost of introducing fault detection latency. The method is based on Reduced Observation Width Replication (ROWR) of the circuit, sufficient to detect all *structural faults*, as opposed to duplication which detects all *functional errors*. After reviewing related work in section 2, the proposed method is presented and analyzed in section 3. Experimental results regarding hardware overhead, fault coverage, and fault detection latency of the proposed method are provided in section 4.

2. Related Work

Almost all previous efforts in concurrent test share the objective of being able to detect *all* faults. What typically distinguishes them, however, is their position within the trade-off space between hardware overhead and fault detection latency. Most approaches fall in one of two ends of this space.

Towards the low end, low cost self-test approaches have been proposed for combinational circuits. C-BIST [2] employs input monitoring and existing off-line Built-In Self-Test hardware, such as LFSRs and MISRs, to perform concurrent self-test. While hardware overhead is very low, the method relies on an ordered appearance of all possible input vectors before a signature indicating circuit correctness can be calculated, resulting in very long fault detection latency. This problem is alleviated in the R-CBIST method described in [3], where the requirement for a uniquely ordered appearance of all input combinations is relaxed at the cost of a small RAM. Nevertheless, all input combinations still need to appear before any indication of circuit correctness is provided.

Towards the high end, we find expensive concurrent error detection methods for sequential circuits that check the circuit functionality at every clock cycle, therefore guaranteeing zero error detection latency. However, reducing the area overhead below the cost of duplication typically requires redesign of the original circuit, thus leading to intrusive schemes. One of the first attempts is described in [4], where resynthesis is employed to encode the states of the circuit, incorporating parity and employing totally self-checking (TSC) checkers [5]. Limitations such as structural

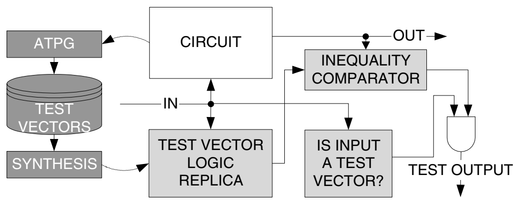

Figure 1. Test Vector Logic Replication (TVLR)

	I_1 I_0	O_1O_0	Non-Redundant Fault List $\{F_1,...,F_{10}\}$ Faults Detected At Each Output Bit:	
V_0	0 0	1 0	O_1:$\{F_1,F_2,F_3\}$	O_0:$\{F_7,F_9,F_{10}\}$
V_1	0 1	1 0	O_1:$\{F_1,F_3,F_7\}$	O_0:$\{F_4,F_5,F_6\}$
V_2	1 0	0 1	O_1:$\{F_1,F_7,F_8\}$	O_0:$\{F_4,F_5,F_7\}$
V_3	1 1	1 1	O_1:$\{F_1,F_2,F_7\}$	O_0:$\{F_4,F_9,F_{10}\}$

Figure 2. Reduced Observation Width Example

constraints requiring an inverter-free design, are alleviated in [6], where partitioning is employed to reduce the incurred hardware overhead. Utilization of multiple parity bits is examined in [7]. While these methods render TSC circuits and guarantee error detection with zero latency, they are intrusive and only provide savings of up to 15% over duplication.

Among the few approaches in between the two ends, a method that exploits properties of non-linear adaptive filters is proposed in [8]. A similar technique introducing latency is proposed in [9], where the frequency response of linear filters is used as invariance. Additionally, an approach exploiting transparency of RT-Level components is described in [10].

Finally, a concurrent fault detection method for combinational logic is described in [11]. This method, which we will refer to as Test Vector Logic Replication (TVLR), is depicted in Figure (1). Since TVLR is similar to the method proposed herein, we describe it briefly to provide a basis for comparison. ATPG is employed to generate a complete set of test vectors, capable of detecting all non-redundant faults in the circuit. This set is subsequently synthesized to form the prediction logic, which is now capable of generating the correct circuit response only for the complete set of test vectors. Since the objective is to minimize the hardware cost of the prediction logic, the remaining input combinations are used as “don’t cares” during synthesis; therefore, the prediction logic will not generate correctly the circuit output for these combinations. To avoid false alarms, an additional function is used to indicate whether the input combination is a test vector, and consequently, whether the output of the comparator should be considered or discarded through the additional AND gate. TVLR is non-intrusive and assuming that ATPG yields a complete set of test vectors, it is capable of detecting all faults. However, it introduces latency in the detection of an activated fault, which will remain undetected until a corresponding test vector appears at the circuit inputs.

3. Reduced Observation Width Replication

While TVLR delivers hardware reduction over duplication at the cost of introducing fault detection latency, it is only one possible solution from a wide array of choices. In an effort to explore the solution space, we observe that for every input combination, each output bit has the ability to detect a subset of all faults in the circuit, as shown in Figure (2). Guaranteeing detection of all non-redundant faults requires that the prediction logic be capable of generating an adequate set of output bits, such that the union of detected faults yields the complete non-redundant fault list. TVLR selects such a set subject to the constraint that when an output bit is included for a given input combination, all output bits for this input combination are included. Subsequently, the optimization objective is to minimize the number of selected input combinations (test vectors), which is achieved through the Test Compaction phase of ATPG. The underlying assumption is that the output width of the prediction logic has to be equal to the output width of the circuit.

As an example, the minimal test set for the simple logic of Figure (2) comprises test vectors V_0, V_1, V_2 and therefore the prediction logic has to be able to generate a 2-bit function for each of the 3 vectors, in total six bits. Notice, however, that only four of these bits are sufficient to detect all faults, while the remaining two are an additional overhead imposed by the constraint of the method mentioned above. More specifically, bits O_1 and O_0 for vector V_0, bit O_0 for vector V_1 and bit O_1 for vector V_2 suffice to detect all faults. Furthermore, notice that there exists a set of output bits capable of detecting all faults that requires replication of *only one* circuit output for every input combination. More specifically, bit O_1 for vectors V_0 and V_2, and bit O_0 for vectors V_1 and V_3 suffice to detect all faults. It is, therefore, possible that a less expensive prediction logic, generating a 1-bit function for all 4 input combinations, as opposed to a 2-bit function for 3 of the 4 input combinations, will suffice for detecting all faults. This observation is the basis for ROWR, the proposed method.

3.1. Description

The optimization objective of ROWR is to minimize the output width of the prediction logic. Based on the observation that a subset of output bits per input combination is typically sufficient to detect all faults, the method aims at identifying a minimal such set. More specifically, the prediction logic now generates only k out of the n circuit output bits, where k is the minimum number of predicted bits per vector that detects all faults. Hardware savings are anticipated due to the reduced output width of the predicted function.

The proposed scheme is depicted in Figure (3). For every m-bit input combination, the prediction logic generates k outputs that match a subset of k out of the n output bits of the circuit. Consequently, a Selection Logic chooses which of the n circuit outputs to drive to the comparator for each m-bit input combination. Two key issues need to be addressed; namely, identification of the output bits to be generated by the prediction logic and cost-effective selection of the circuit outputs to which they should be compared.

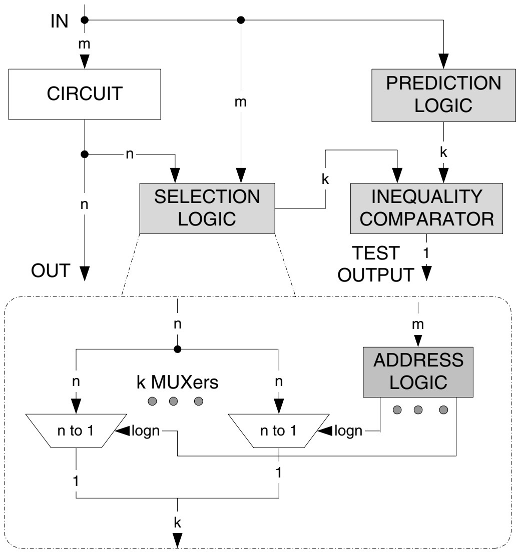

Figure 3. Reduced Observation Width Replication

Regarding the first issue, an ATPG tool capable of generating all test vectors and reporting both the good circuit and the faulty circuit output for every fault is required. This information indicates the faults that can be detected at each output bit for each input combination and may be used to construct a table similar to the one shown in Figure (4). Intuitively, it may seem that the optimal solution should comprise a set of columns that covers all faults, such that the maximum number of output bits to be observed for any input vector is minimized. This is not true, however, since the exact selection of columns has a direct and significant impact on the cost of the Selection Logic, bringing us to the second issue mentioned above. More specifically, since the prediction logic only generates a k-bit function, additional logic is necessary to select k among the n circuit outputs to which the predicted k bits will be compared. As shown in Figure (3), this can be viewed as k n-to-1 multiplexers, each of which requires $\log n$ address bits. Therefore, if we allow all possible subsets of size k for every m-bit input combination, the Address Logic will comprise $k \cdot \log n$ m-input functions. As compared to duplication, the prediction logic would implement $n - k$ fewer m-input functions, at the cost of implementing $k \cdot \log n$ m-input functions and k n-to-1 multiplexers for the Selection logic. Obviously, this is not a winning strategy if $k > n/(\log n + 1)$, in which case $k \cdot \log n > n - k$. Furthermore, the cost of the Selection Logic is hard to estimate as k increases, reducing to a zero-cost identity function in the extreme case of $k = n$.

Therefore, restrictions need to be imposed on the complexity permitted for the Address Logic and, by extension, to the acceptable sets of columns to cover the faults in the table of Figure (4). In the proposed methodology we eliminate the Address Logic all together, therefore allowing that the $\log n$ select inputs of each multiplexer may only be driven directly by any $\log n$ out of the m inputs bits.

	VECTOR 0			. . .	VECTOR 2^m-1		
	OUT_0	...	OUT_{n-1}	. . .	OUT_0	...	OUT_{n-1}
$Fault_1$	1						
$Fault_2$					1		1
⋮	⋮	⋮	⋮	. . .	⋮	⋮	⋮
$Fault_M$			1	. . .	1		

Figure 4. Fault Detection Table

We now formally state the problem, assuming that we are given the table (say A) described above. The first step partitions the 2^m input combinations in $O(n)$ disjoint subsets, by selecting $O(\log n)$ input bits (out of m). In each subset, we include all inputs $(0 \ldots 2^m)$ that have the same value in the $O(\log n)$ selected bits. In the second step, *for each* subset, a set of k $(1 \leq k \leq n)$ output bits is selected. We emphasize that k is fixed for all subsets. Thus, $k \cdot 2^m$ columns of A are selected (out of $n \cdot 2^m$). We seek an algorithm to select k bits for each subset so that all faults are covered **and** k is minimized. The problem is NP-complete; we outline a randomized algorithm to approximate it.

We solve the problem for a fixed k; finding the minimum k is trivial using binary search and repeating the following algorithm $\log n$ times. We assume that the inputs are split in M subsets, denoted by $S_1 \ldots S_M$ and, for each S_i, we seek to identify a set R_i of k output bits.

Step 1: For each S_i, find all faults F_i covered by exactly one column of A. Include the bit corresponding to that column in R_i. Remove all faults (rows of A) covered by this bit and a vector in S_i. Repeat until all remaining faults are covered by two or more columns of A.

Step 2: For each S_i, find all faults F_i that are covered by vectors only in S_i. Include in R_i the minimum number of bits required to cover all these faults (this is done by exhaustive enumeration in our experiments; elaborate approximation algorithms exist [12]). Remove all faults (rows of A) covered by these bits and a vector in S_i. Repeat until all remaining faults are covered by two or more vectors in different subsets.

Step 3: If any $|R_i| \geq k$ report failure. Otherwise, randomly pick values for the $k - |R_i|$ remaining bits $(i = 1 \ldots M)$. We repeat this step K times; if no combination covering all faults is found, we report failure. We actually use an *adaptive scheme* for the random sampling of values.

Similarly to TVLR, ROWR is non-intrusive and guarantees 100% fault coverage. Furthermore, since ROWR predicts and compares the appropriate portion of the circuit output for every input combination, no false alarm is possible. ROWR also introduces latency in the detection of an activated fault, which will remain undetected until an appropriate vector appears at the circuit inputs. We stress, however, that ROWR checks for faults for *every* input combination, unlike TVLR which checks more infrequently. Since most stuck-at faults are detected by many input vectors, we expect the average latency of ROWR to be less than that of TVLR.

CIRCUIT	DUPLICATION	TVLR		ROWR		COST COMPARISON	
	COST	VECTORS	COST	BITS	COST	TVLR vs DUPL.	ROWR vs. DUPL.
4_4	38048	11 / 16	47792	2 / 4	22272	125.60 %	58.53 %
5_5	117856	21 / 32	113680	2 / 5	56608	96.45 %	48.03 %
6_6	126768	40 / 64	230144	3 / 6	137808	88.25 %	52.84 %
7_7	552624	66 / 128	419920	3 / 7	275472	75.98 %	49.84%
8_8	1219856	143 / 256	879744	3 / 8	527568	72.11 %	43.24 %
9_9	2467088	254 / 512	1766912	4 / 9	1285280	71.61 %	52.09 %

Figure 5. Hardware Overhead Comparison

4. Experimental Results

In this section, we compare TVLR and ROWR to duplication, in terms of hardware overhead, fault coverage, and fault detection latency. We experimented with and report results for random logic, synthesized using SIS [13], and mapped to a standard cell library. ATPG is performed using ATALANTA [14]. The test vector set is synthesized, rendering the prediction logic for TVLR. The prediction for non-vectors is "don't care", allowing SIS [13] to minimize the required hardware. ATALANTA [14] is used to generate all possible vectors detecting each fault, and HOPE [15] is employed to provide the good machine and the bad machine responses for every *(vector, fault)* pair, revealing the output bits at which each fault may be detected for every vector. This information is used to construct the table and identify the prediction function necessary for ROWR, which is also synthesized using SIS [13]. The concurrently testable circuits are, then, constructed as described in sections 2 and 3.1 for TVLR and ROWR respectively. Comparison of the three alternative methods, Duplication, TVLR, and ROWR is now possible.

4.1. Hardware Overhead

Results are summarized in the table of Figure (5) for six different sizes X_Y of random circuits, where X is the number of inputs and Y is the number of outputs. The number of TVLR test vectors and the width of the predicted ROWR function are also reported. ROWR achieves significantly better savings than TVLR, over duplication. Even for the smallest circuit, ROWR incurs 58.53% overhead, while TVLR costs more that duplication. For the above circuits, TVLR saves on average around 16% over duplication, while ROWR saves on average around 50%.

4.2. Fault Coverage

Both TVLR and ROWR detect all non-redundant faults in the original circuit. To demonstrate this, we construct the complete concurrently testable circuits as described in sections 2 and 3.1 for TVLR and ROWR respectively. Only the test output is made observable, and ATALANTA [14] is used to generate test vectors for all non-redundant faults in the original circuit. The results are summarized in the table of Figure (6), where as expected, all faults are detectable by both the TVLR and the ROWR method. Also by construction, both methods are expected to detect all non-redundant faults in the prediction logic circuit. For these faults, the original circuit acts as a duplicate, thus detecting them through the comparator. To demonstrate this, we performed ATPG for all faults in the complete circuit using ATALANTA [14], observing both the test output and the primary outputs of the circuit, thus obtaining the list of all non-redundant faults in the circuit. A final ATPG run for these faults, observing only the test output, shows that all non-redundant faults in the circuit are detectable both by TVLR and by ROWR.

4.3. Fault Detection Latency

Although the exact latency introduced by TVLR and ROWR may not be predicted, an experimentally obtained indication is necessary for their evaluation. Similarly to [2, 3, 11], we assume a uniform distribution at the circuit inputs and employ fault simulation of randomly generated input sequences. More specifically, for each method we use HOPE [15] to perform *two* fault simulations of the *same* sequence of randomly generated inputs, once observing both the test output and the circuit outputs, and a second time observing only the test output. The time step at which a fault is detected during the first fault simulation is the *Fault Activation* time, while the time step at which a fault is detected during the second fault simulation is the *Fault Detection* time. Fault Detection Latency is defined as the time difference between Fault Activation and Fault Detection, therefore we can easily calculate the *Fault Detection Latency* for each fault, as well as the average Fault Detection Latency.

Results are summarized in the table of Figure (7) for both TVLR and ROWR. We fault simulate a total of 5000 random patterns and snapshots of the results are shown after 10, 50, 100, 500, 1000, and finally all 5000 patters have been applied. For each snapshot, we provide the number of faults remaining non-activated, the number of faults activated and detected, and the number of faults activated but not yet detected. We also provide the maximum and the average fault detection latency for the faults that are both activated and detected. Based on the results, we observe the following:

CIRCUIT	TVLR		ROWR	
	ORIGINAL FAULTS	ALL FAULTS	ORIGINAL FAULTS	ALL FAULTS
4_4	60 / 60	184 / 184	60 / 60	150 / 150
5_5	181 / 181	417 / 417	181 / 181	348 / 348
6_6	387 / 387	830 / 830	387 / 387	676 / 676
7_7	819 / 819	1567 / 1567	819 / 819	1326 / 1326
8_8	1806 / 1806	3290 / 3290	1806 / 1806	2693 / 2693
9_9	3710 / 3710	6583 / 6583	3710 / 3710	5781 / 5781

Figure 6. Fault Coverage by TVLR and ROWR

CIRCUIT	TESTABLE FAULTS	STATISTIC	RANDOM 10		RANDOM 50		RANDOM 100		RANDOM 500		RANDOM 1000		RANDOM 5000	
			TVLR	ROWR	TVLR	ROWR	TVLR	ROWR	TVLR	ROWR	TVLR	ROWR	TVLR	ROWR
4_4	0	REMAINING	6	6	0	0								
		DETECTED	52	53	60	60								
		MISSED	2	1	0	0								
		MAX LAT	1	9	24	24								
		AVG LAT	0.01	0.56	1.11	1.05								
5_5	181	REMAINING	55	55	5	5	1	1	0	0				
		DETECTED	99	106	164	168	179	176	181	181				
		MISSED	27	20	12	8	1	4	0	0				
		MAX LAT	7	8	23	44	91	81	135	155				
		AVG LAT	0.53	0.43	2.79	3.17	6.17	4.47	6.85	7.24				
6_6	387	REMAINING	179	179	37	37	10	10	0	0				
		DETECTED	148	177	328	330	368	372	387	387				
		MISSED	60	31	22	20	9	5	0	0				
		MAX LAT	8	9	43	43	72	80	372	137				
		AVG LAT	0.95	0.18	4.23	2.21	6.05	3.74	9.67	5.01				
7_7	819	REMAINING	411	411	117	117	36	36	0	0				
		DETECTED	330	340	614	639	745	760	819	819				
		MISSED	78	68	88	63	38	23	0	0				
		MAX LAT	9	9	46	46	80	70	474	405				
		AVG LAT	1.35	0.52	4.67	2.10	7.91	4.00	17.04	8.41				
8_8	1806	REMAINING	1120	1120	459	459	222	222	11	11	0	2	0	0
		DETECTED	512	523	1027	1184	1339	1493	1763	1782	819	1803	1806	1806
		MISSED	174	163	320	163	245	91	32	13	0	1	0	0
		MAX LAT	5	9	45	44	91	98	462	448	514	832	1910	1910
		AVG LAT	0.22	0.78	3.97	3.25	8.77	6.20	28.84	15.20	17.04	18.90	40.03	19.93
9_9	3710	REMAINING	2541	2541	1243	1243	722	722	68	68	2	9	0	0
		DETECTED	769	877	1690	2235	2471	2774	3473	3582	1799	3693	3710	3710
		MISSED	400	292	777	232	517	214	169	60	5	8	0	0
		MAX LAT	5	9	41	47	93	96	474	486	868	959	2075	1738
		AVG LAT	0.26	0.43	6.30	2.34	13.22	3.94	37.40	14.50	36.30	21.71	67.67	23.60

Figure 7. Fault Detection Latency for TVLR and ROWR

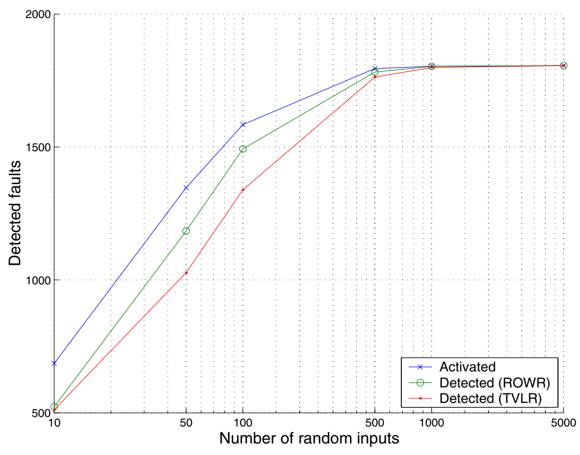

Figure 8. Faults vs Number of Patterns for 8_8

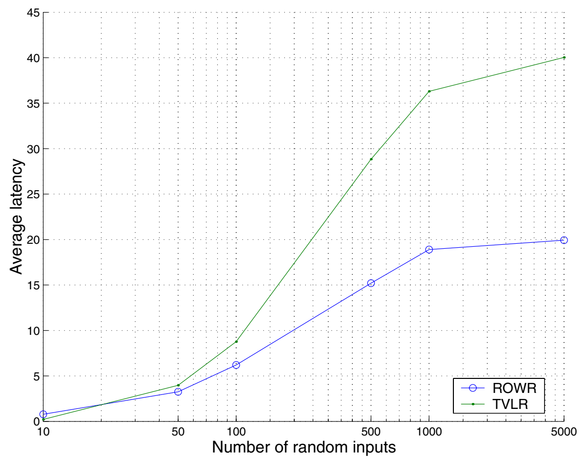

Figure 10. Latency vs Number of Patterns for 8_8

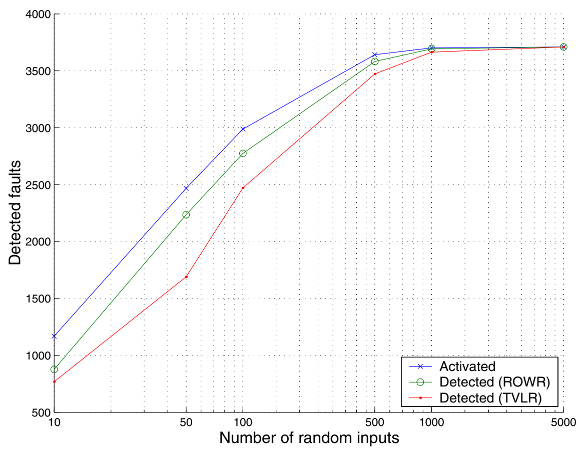

Figure 9. Faults vs Number of Patterns for 9_9

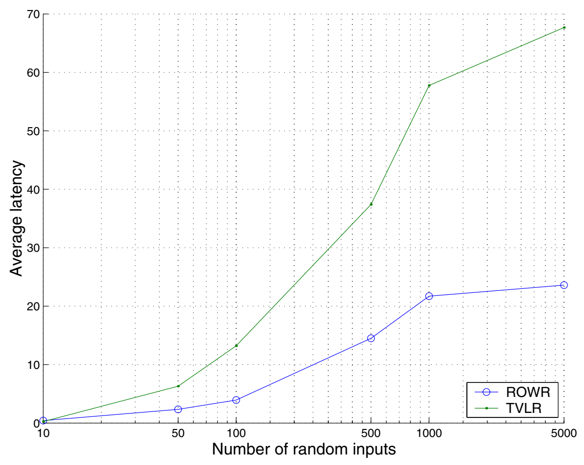

Figure 11. Latency vs Number of Patterns for 9_9

- While the MAX latency is O($N \cdot \log N$) vectors, the AVG latency ranges only up to 68 vectors for TVLR and up to 24 vectors for ROWR. For example, once all faults are detected in the 9_9 circuit, where $N \cdot \log N$=4608, the MAX latency is 2075 vectors for TVLR and 1738 vectors for ROWR. However, the AVG latency is 67.67 vectors for TVLR and 23.60 vectors for ROWR, which is only the 3.26% and 1.35% of the respective MAX latency. Similar observations hold for all circuits.

- For both TVLR and ROWR most faults are detected quickly and a 90-10 rule applies for the AVG latency: 90% of the faults are detected within 50% of the AVG latency, while the other 50% is contributed by the remaining 10% of the faults. For example, once 500 vectors are applied to the 9_9 circuit, 98.17% of all faults are activated, out of which 93.65% are detected by TVLR and 96.54% by ROWR. The AVG fault detection latency at this point is 37.40 vectors for TVLR and 14.50 vectors for ROWR, which represents the 55.26% and 61.44% of the AVG latency when all faults are detected. Similar observations hold for all circuits.

Furthermore, a comparative latency examination of TVLR and ROWR leads to the following two observations:

- ROWR detects more faults slightly faster than TVLR. A plot of the faults activated, faults detected by TVLR, and faults detected by ROWR as the number of applied random patterns increases is given in Figures 8 and 9 for circuits 8_8 and 9_9, respectively. As demonstrated, ROWR consistently detects more faults faster than TVLR, up to the convergence point where all faults are detected by both methods. The observation holds for all circuits and, interestingly, the gain is larger as the size of the circuit increases.

- ROWR detects faults with significantly lower AVG latency than TVLR. A plot of the AVG fault detection latency of ROWR and TVLR as the number of applied random patterns increases is given in Figures 10 and 11 for circuits 8_8 and 9_9 respectively. As demonstrated, ROWR consistently detects faults with lower AVG latency than TVLR. Once again, the observation holds for all circuits and, interestingly, the gain is larger as the size of the circuit increases.

5. Conclusions

Cost-effective concurrent fault detection requires careful examination of the trade-offs between the conflicting objectives of low hardware overhead, low fault detection latency, and high fault coverage. ROWR explores the trade-off between fault detection latency and hardware overhead, under the constraint that the original circuit may not be altered. Thus, a comparison-based approach is employed, where the original circuit is partially replicated into a prediction logic that selectively tests the circuit during normal operation. The problem of identifying cost-effective prediction logic functions is theoretically formulated and an algorithm for efficient partial replication is proposed. Experimental results demonstrate that ROWR reduces significantly the hardware overhead incurred by either duplication or TVLR, while preserving the ability to detect all permanent faults in the circuit. Further reduction of this overhead is anticipated as the size of the circuit increases. While these savings come at the cost of introducing fault detection latency, the experimentally observed average latency is lower than the latency of TVLR and scales sub-linearly with the size of the circuit. Thus, when non-zero fault detection latency may be tolerated, ROWR is a superior alternative to both duplication and TVLR.

References

[1] A. Avizienis and J. P. J. Kelly, "Fault tolerance by design diversity: Concepts and experiments," *IEEE Computer*, vol. 17, no. 8, pp. 67–80, 1984.

[2] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits," *IEEE TCAD*, vol. 7, no. 12, pp. 1250–1260, 1988.

[3] I. Voyiatzis, A. Paschalis, D. Nikolos, and C. Halatsis, "R-CBIST: An effective RAM-based input vector monitoring concurrent BIST technique," in *ITC*, 1998, pp. 918–925.

[4] N. K. Jha and S.-J. Wang, "Design and synthesis of self-checking VLSI circuits," *IEEE TCAD*, vol. 12, no. 6, pp. 878–887, 1993.

[5] D. Nikolos, "Optimal self-testing embedded parity checkers," *IEEE TCOMP*, vol. 47, no. 3, pp. 313–321, 1998.

[6] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE TCAD*, vol. 16, no. 7, pp. 783–789, 1997.

[7] C. Zeng, N. Saxena, and E. J. McCluskey, "Finite state machine synthesis with concurrent error detection," in *ITC*, 1999, pp. 672–679.

[8] A. Chatterjee and R. K. Roy, "Concurrent error detection in non-linear digital circuits with applications to adaptive filters," in *ICCD*, 1993, pp. 606–609.

[9] I. Bayraktaroglu and A. Orailoglu, "Low-cost on-line test for digital filters," in *VTS*, 1999, pp. 446–451.

[10] Y. Makris, I. Bayraktaroglu, and A. Orailoglu, "Invariance-based on-line test for RTL controller-datapath circuits," in *VTS*, 2000, pp. 459–464.

[11] R. Sharma and K. K. Saluja, "An implementation and analysis of a concurrent built-in self-test technique," in *FTCS*, 1988, pp. 164–169.

[12] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 3rd edition, 1995.

[13] E. M. Sentovich et al., "SIS: a system for sequential circuit synthesis," ERL MEMO. No. UCB/ERL M92/41, EECS UC Berkeley CA 94720, 1992.

[14] "ATALANTA combinational test generation tool," Available from `http://www.ee.vt.edu/ha/cadtools`.

[15] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE TCAD*, vol. 15, no. 9, pp. 1048–1058, 1996.