Non-Intrusive Design of Concurrently Self-Testable FSMs

Petros Drineas* and Yiorgos Makris
Departments of Computer Science and Electrical Engineering
Yale University
{petros.drineas, yiorgos.makris}@yale.edu

Abstract

We propose a methodology for non-intrusive design of concurrently self-testable FSMs. The proposed method is similar to duplication, wherein a replica of the original FSM acts as a predictor that immediately detects potential faults by comparison to the original FSM. However, instead of duplicating the complete FSM, the proposed method replicates only a minimal portion adequate to detect all possible faults, yet at the cost of introducing potential fault detection latency. Furthermore, in contrast to concurrent error detection approaches, which presume the ability to resynthesize the FSM and exploit parity-based state encoding, the proposed method is non-intrusive and does not interfere with the encoding and implementation of the original FSM. Experimental results on FSMs of various sizes and densities indicate that the proposed method detects 100% of the faults with very low average fault detection latency. Furthermore, a hardware overhead reduction of up to 33% is achieved, as compared to duplication-based concurrent error detection.

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 focus on controller circuits and we explore these trade-offs, in order to devise a non-intrusive design methodology for concurrently self-testable FSMs. Non-intrusiveness implies that hardware is only added in parallel to the given FSM, which is encoded, optimized, and implemented to meet specific requirements 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 delay or degrade the normal functionality of the FSM.

*The author is supported in part through NSF grant CCR-9820850.

The underlying principle of concurrent test is the addition of hardware that monitors the circuit inputs during normal operation and generates an *a priori* known property that is expected to hold for the circuit outputs. A property verifier is subsequently utilized to indicate any violation of the expected 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 any error in the circuit. However, it incurs significant hardware overhead that exceeds 100% of the original circuit. Given that 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 self-test method for FSMs, that reduces hardware overhead at the cost of introducing fault detection latency. The method is based on replication of a subset of state transitions, 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

To motivate the proposed methodology, we first examine related work in the areas of concurrent self-test, concurrent error detection, and on-line test. Almost all previous research efforts in these areas 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 the 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 la-

Proceedings of the 11 th Asian Test Symposium(ATS'02)
1081-7735/02 $17.00 © 2002 IEEE

tency. 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. In [4], a test vector based self-test method is described and evaluated for combinational circuits.

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. Reducing the area overhead below the cost of duplication typically requires redesign of the original circuit, thus leading to intrusive methodologies. One of the first attempts is described in [5], where resynthesis is employed to encode the states of the circuit, incorporating parity information and employing TSC checkers [6]. Limitations of this method, such as structural constraints requiring an inverter-free design, are alleviated in [7], where partitioning is employed to reduce the incurred hardware overhead. Utilization of multiple parity bits is examined in [8] within the context of FSMs. All these methods render totally self-checking circuits and guarantee error detection with zero latency; on the down side, they are intrusive and only provide savings in the range of 10% over duplication.

Among the few existing approaches in between the two ends, a method that exploits properties of non-linear adaptive filters is proposed in [9], achieving a 30% cost reduction. A similar technique is proposed in [10], where the frequency response of linear filters is used as an invariance property, achieving a 50% cost reduction but introducing latency. Finally, a CFD approach exploiting transparency of RT-Level components is described in [11], achieving over 90% fault security with 40% hardware overhead.

3. Proposed Method

The concurrent self-test method for FSMs proposed in this paper explores the trade-off between incurred hardware overhead and fault detection latency, while preserving the ability to detect all faults. As an additional constraint, we require that the proposed method be non-intrusive, leaving the original implementation of the circuit (i.e. state encoding and next state logic) intact. The proposed method targets the detection of faults as opposed to errors, therefore imposing more lenient requirements in terms of fault detection latency, as opposed to the stringent zero-latency required for concurrent error detection. Consequently, checking is performed frequently, yet not at every clock cycle. While the resulting circuits are not *fault-secure* and therefore do not guarantee correctness of the results, they are still *self-testable*, thus guaranteeing eventual detection of all faults. In the rest of this section, we describe the proposed methodology and we analyze its expected performance.

3.1. Description

The proposed scheme is depicted and contrasted to duplication in figure 1. In duplication-based concurrent error detection, a replica of the FSM[1] is added and the results of the two FSMs are compared at each clock cycle. Errors are indicated by a test output that becomes '1'. To avoid fault masking by common-mode failures, design diversity [1] has been examined, wherein the duplicate FSM is functionally equivalent but structurally different than the original FSM.

In order to reduce the overhead of duplication, the proposed method replicates only a portion of the original FSM, capable of detecting all faults in the design. More specifically, ATPG is performed on the combinational next state logic of the original FSM, treating the previous state bits as primary inputs, and a complete set of test vectors is obtained[2]. These test vectors are subsequently synthesized into a prediction logic that generates the expected next state of the FSM when an input / previous state combination matches a test vector. The outputs of the prediction logic for input / previous state combinations that are not included in the test vector set are treated as *don't cares*. Thus, during synthesis, these outputs are set to appropriate values minimizing the required hardware. As a result, the prediction logic is less expensive than the duplicate next state logic.

However, since the output prediction logic will only generate the correct next state for input / previous state combinations included in the test vector set, the issue of *false alarms* needs to be addressed. More specifically, the concurrent test output should not be asserted during normal functionality, unless a fault has been detected. Therefore, an additional function is now required, indicating whether an input / previous state combination is a test vector. In the opposite case, the comparison outcome is not a valid indication of operational health of the FSM and is therefore masked through the AND gate in figure 1. Notice also that the predicted next state calculation is driven by the original FSM state register and not by the predicted state register, since the latter may not contain the correct value after an input / previous state combination that is not a test vector.

The test vector set detects faults in the combinational next state logic. In order to also detect the faults in the state register, we delay the comparison of the predicted next state by one clock cycle, similarly to [8]. Thus, instead of comparing the outputs of the predicted logic, we compare the outputs of the state registers one clock cycle later, at the cost of an additional flip-flop. Assuming that test responses comprise both a logic '1' and a logic '0' at every bit position, all faults in the state register will also be detected.

[1] For simplicity, we assume that the FSM outputs are directly driven by the state register. The method can be extended to include output logic.

[2] An additional state reachability analysis step is needed to guarantee effectiveness of the test vectors on the complete FSM, as explained shortly.

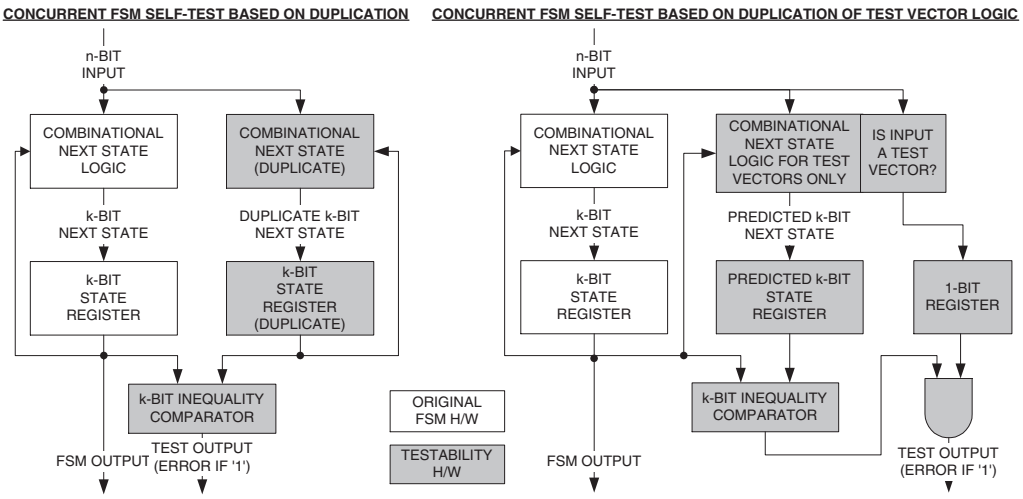

Figure 1. Duplication vs. Proposed Methodology

Finally, as mentioned in footnote 2, additional processing of the vectors generated by ATPG on the combinational next state logic is necessary to guarantee their effectiveness in the complete FSM. In the presence of a fault F, some valid FSM states may become unreachable. In order to guarantee detection of F, we need to ensure that there exists at least one test vector in the selected test set that detects F from a state that is reachable in the presence of F. Therefore, for every fault F the faulty FSM is built and a Breadth-First-Search algorithm is applied to identify states that are unreachable in the presence of F. In case the generated test set does not include such a vector, an additional ATPG run is performed, excluding test vectors that comprise unreachable previous states as part of the input / previous state combination. Our experimental observation is that such cases amount for less than 1% of all faults.

As shown in figure 1, unlike approaches such as [5, 7, 8] that reduce the hardware overhead by re-encoding the FSM, the proposed method leaves the original FSM intact. Furthermore, and despite the addition of one extra function (IS INPUT A TEST VECTOR), a considerable hardware overhead reduction is expected. On the down side, faults go undetected until an appropriate test vector appears, thus introducing latency. However, given a sizeable test set, tests are performed frequently and low average latency is expected.

3.2. Analysis

Assuming that ATPG yields a complete test set, the proposed method guarantees fault coverage equivalent to duplication. We emphasize that test vectors are split in two parts: the previous FSM state bits and the inputs bits.

The proposed scheme introduces latency in the detection of an activated fault. Even though the dependence on the circuit inputs makes it impossible to predict the introduced latency, we stress that, assuming random inputs, the proposed method checks for faults every time an input / previous state combination that is a selected test vector appears. Empirical observations, as well as some theoretical analysis [12], indicate that, for random next state prediction logic, the ratio of such input / previous state combinations over all possible input combinations is significant. Additionally, we may reasonably assume that most "stuck-at" faults are detected by many input / previous state combinations. Thus, we expect the average latency to be very small.

We now outline a few observations on the hardware overhead. We denote by n the number of inputs to the FSM and by k the number of state bits; let $m = n + k$. The following remark relates the hardware – assuming multilevel implementation using 2-input gates – required to implement a "fully-specified" random boolean function with m input bits and one output bit to the hardware required to implement an "α-specified" random function with m input bits and one output bit. We first define α-specified functions:

Definition 1 *An α-specified function $f : \{0,1\}^m \rightarrow \{0,1\}$ has at least $\lceil \alpha 2^m \rceil$ "don't cares" ($\alpha \in [0,1]$). The positions of the "don't cares" are fixed* **a priori**. *A fully-specified function is equivalent to a 0-specified function.*

Remark 1 *Almost all boolean functions $f : \{0,1\}^m \rightarrow \{0,1\}$ require at least $2^m/m$ gates, if they are fully-specified and $(1-\alpha)2^m/m$ gates, if they are α-specified.*

Proof (Sketch): The first statement is Shannon's counting argument [13]. For the second statement, we observe that the number of functions $f : \{0,1\}^m \rightarrow \{0,1\}$ with $\lceil a 2^m \rceil$ "don't cares" is $2^{\lceil (1-a)2^m \rceil}$. Thus, the same counting argument proves our statement. For the statement to hold, positions of the "don't cares" should be pre-specified.

We now relate the hardware required to implement a fully-specified function with m input bits and one output bit to the hardware required to implement a fully-specified function with m input bits and k output bits, where k is a small constant ($k \ll 2^m$) and the k bits are *uncorrelated*.

Remark 2 *Almost all boolean functions* $f : \{0,1\}^m \rightarrow \{0,1\}^k$ *require at least* $k2^m/m$ *gates* if *the* k *output bits are uncorrelated and fully specified.*

Proof (Sketch): Again, we observe that the number of functions $f : \{0,1\}^m \rightarrow \{0,1\}^k$ is $\left(2^{2^m}\right)^k = 2^{2^m k}$. Thus, Shannon's counting argument proves our statement.

The original circuit is a function $f : \{0,1\}^{n+k} \rightarrow \{0,1\}^k$. Our technique generates a set of input / previous state combinations that comprise a set of test vectors for f or, equivalently, a function $\tilde{f} : \{0,1\}^{n+k} \rightarrow \{0,1\}^k$. Assuming for the moment that the k output bits are uncorrelated and the cardinality of the selected set of test vectors is $\alpha 2^{n+k}$, for some $\alpha \in [0,1]$, the hardware required for the IS INPUT A TEST VECTOR component of figure 1 should be $1/k$ times the hardware required for the original circuit. Similarly, the hardware required for the NEXT STATE LOGIC FOR TEST VECTORS component of figure 1 is expected to be α times the hardware required for the original circuit. Thus, the *minimum* hardware required for our scheme should be $(\alpha + 1/k)$ times the *minimum* hardware required for the the original circuit. Depending on α and k, the hardware overhead of our technique might be significantly smaller than duplication. We point that we can only examine how the *lower bound* of the size of the prediction logic behaves; indeed, tight bounds for circuit sizes are notoriously hard to prove even under stringent assumptions. In practice, the k state bits representing the output of the next state prediction logic are correlated. It is not clear, however, that as the state bits become more and more correlated the size of the prediction logic over the original circuit size increases; one expects the size of the NEXT STATE LOGIC FOR TEST VECTORS to decrease as correlation increases. On the other hand, it seems natural that the size of the IS INPUT A TEST VECTOR component over the original circuit would increase. Finally, the NEXT STATE LOGIC FOR TEST VECTORS component and the IS INPUT A TEST VECTOR component are not implemented separately; indeed, in order to maximize logic sharing they are synthesized together. Thus, the cost of the prediction logic is less than the sum of the individual costs.

4. Experiments

We compare the proposed method to duplication in terms of area overhead, fault coverage, and fault detection latency. The experimental setup is described in this section, followed by a presentation and discussion of the results.

4.1. Setup

In order to preserve generality, the experiments are applied on random FSMs. These FSMs are converted to *pla* format, synthesized and optimized using the *rugged* script of the SIS system [14], and mapped to a standard cell library comprising only 2-input gates. The hardware cost is reported by SIS through the *print_map_stats* command. The circuit is then converted to ISCAS89 [15] format.

An ATPG run is performed on the combinational next state logic of the FSM using ATALANTA [16] to obtain a complete test set. These vectors and the corresponding fault-free responses, along with the additional function that indicates whether an input combination is a test vector are then converted to *pla* format, synthesized and optimized using the *rugged* script of the SIS system [14], and mapped to a standard cell library comprising only 2-input gates. The hardware cost is reported by SIS through the *print_map_stats* command. Subsequently, the circuits are converted to ISCAS89 [15] format, rendering the hardware implementation of the prediction logic. The original FSM and the prediction logic are combined and the concurrently self-testable FSM is constructed. Two sequential ATPG runs are then performed using HITEC [17]. In the first ATPG run, both the test output and the original FSM outputs are made observable, while in the second ATPG run, only the test output is made observable. We emphasize that the resulting fault coverage comprises faults both in the original FSM and the prediction logic.

In order to calculate the average fault detection latency, we generate sets of random inputs, which we fault simulate twice on the constructed FSM using HOPE [18]. During the first fault simulation we observe all outputs (including the TEST output), while during the second fault simulation we only observe 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. The number of faults detected only in the first fault simulation but not in the second, i.e. faults sensitized but not detected by the random vectors, is also reported.

4.2. Results

The experimental setup was applied on 10 different "types" of FSMs, namely 10 different (K, n) combinations, (8,1), (8,2), (16,1), (16,2), (32,1), (32,2), (32,3), (64,1), (64,2), and (64,3). The proposed method was applied on 5 FSMs of each type and average results are reported.

FSM Type (States_Inputs)	Cost of Next State Logic	Number of Test Vectors	Cost of Test Vector Logic	Hardware Overhead
8_1	33408	10 / 16	33872	101.39 %
8_2	70374	19 / 32	65578	93.18 %
16_1	95275	20 / 32	83210	87.33 %
16_2	186219	35 / 64	153429	82.39 %
32_1	222411	38 / 64	178794	80.38 %
32_2	423014	69 / 128	325882	77.03 %
32_3	832571	128 / 256	615573	73.93 %
64_1	504368	70 / 128	384346	76.20 %
64_2	937744	129 / 256	688112	73.37 %
64_3	1809757	237 / 512	1227689	67.83 %

Figure 2. Hardware Area Overhead of Proposed Method

FSM Type	8_1	8_2	16_1	16_2	32_1	32_2	32_3	64_1	64_2	64_3
Testable Faults	148	249	324	584	682	1209	2269	1378	2550	4651
Detected Faults	142	243	316	578	672	1199	2269	1366	2538	4639

Figure 3. Fault Coverage of Proposed Method

The area overhead is compared to duplication in the table of figure 2. The cost of the next state logic which is used as a predictor in duplication is first reported, followed by the number of test vectors and their cost, which comprises both the NEXT STATE LOGIC FOR TEST VECTORS component and the IS INPUT A TEST VECTOR component. The rightmost column shows the incurred overhead as a percentage of the cost of duplication. An average reduction of 20% is achieved. We note that the reduction increases with the size of the FSM, reaching 33% for the (64,3) example.

The actual hardware overhead is close to the expected overhead based on the analysis of section 3.2. We predicted the ratio of our method over duplication to be $a+1/k$, where a is the ratio of the 2^{n+k} (state, input) combinations that are test vectors. On average, the deviation between our prediction and the observed overhead is only 3.12%, which we attribute to the correlation of next state bits.

Figure 3 reports the average fault coverage results. The middle row shows the number of testable faults in the concurrently self-testable FSMs, as reported by sequential ATPG. The bottom row provides the number of faults that are detected by the proposed methodology. All but a small number of faults are detected. The few faults that are not detected are faults on the primary outputs, which no comparison-based method can detect [10]. Therefore, the method detects the same number of faults as duplication on the original FSM, as well as all faults in the prediction logic.

To obtain an experimental indication of the introduced latency, 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 *missed* (not yet detected). We also provide the maximum fault detection latency and the average fault detection latency for the faults that are both activated and detected. Worst-case results are summarized in figure 4 for the $(64, 3)$ FSM. Figure 5 presents a plot of faults activated and faults detected on the $(64, 3)$ FSM, as well as a plot of the average fault detection latency on the $(64, 1)$, $(64, 2)$, and $(64, 3)$ FSMs.

While the maximum latency ranges up to 4203 clock cycles for the (64,3) FSM, the average latency is small, ranging up to only 91.05 clock cycles, which is 2.16% of the maximum latency. Additionally, most faults are detected quickly, with 90% of the faults detected within 50% of the average latency, while the other 50% is contributed by the remaining 10%. For example, once 500 random vectors are applied to the (64,3) circuit, 96.65% of the faults are activated and 90.58% are detected. The average fault detection latency at this point is 38.94, which is 42.76% of the average latency when all faults are detected. Furthermore, the plot of figure 5(a) shows that the number of faults activated but not yet detected by the proposed method is small. As indicated in the plot of figure 5(b), the average and the maximum latency increase sub-linearly with the circuit size.

5. Conclusions

Cost-efficient concurrent fault detection in FSMs necessitates a careful consideration of the conflicting objectives of low hardware overhead, low fault detection latency, and high fault coverage. Along these lines, we propose a methodology for designing FSMs that can be self-tested concurrently with their normal functionality, without paying the cost of duplication and without altering the FSM implementation. Experimental results demonstrate that the proposed methodology reduces the incurred hardware overhead by as much as 33% over duplication-based concurrent error detection, while preserving the ability to detect all faults in the circuit, yet at the cost of non-zero fault detection latency. Nevertheless, the experimentally observed average fault detection latency is very low, ranging up to 92 clock cycles for detecting all faults in the largest FSM.

Stats	Rand 10	Rand 50	Rand 100	Rand 500	Rand 1K	Rand 5K
Remaining	1970	1081	640	91	25	0
Detected	589	1286	1669	2492	2657	2751
Missed	192	384	442	168	69	0
Max - Lat	7	41	94	451	936	4203
Avg - Lat	0.18	2.97	7.86	38.94	60.75	91.05

Figure 4. Fault Detection Latency of Proposed Method

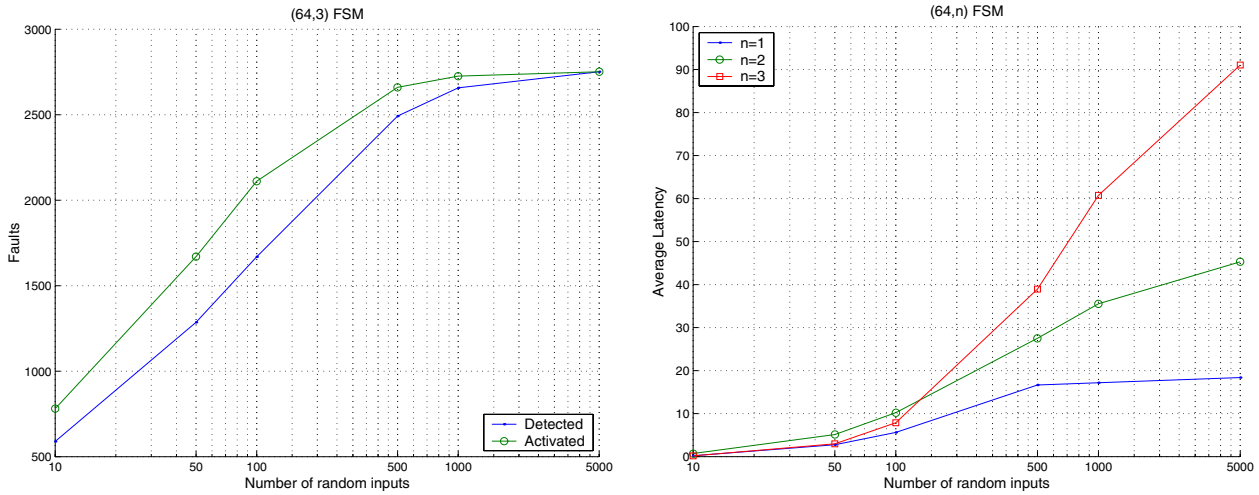

Figure 5. (a) Fault Coverage vs Number of Random Inputs (b) Average Latency vs Number of Random Inputs

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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 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 *International Test Conference*, 1998, pp. 918–925.

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

[5] N. K. Jha and S.-J. Wang, "Design and synthesis of self-checking VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 6, pp. 878–887, 1993.

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

[7] N. A. Touba and E. J. McCluskey, "Logic synthesis of multilevel circuits with concurrent error detection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 7, pp. 783–789, 1997.

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

[9] A. Chatterjee and R. K. Roy, "Concurrent error detection in non-linear digital circuits with applications to adaptive filters," in *International Conference on Computer Design*, 1993, pp. 606–609.

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

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

[12] K. Raahemifar and M. Ahmadi, "Novel test generation algorithm for combination circuits," *Journal of Circuits, Systems, and Computers*, vol. 10, pp. 27–65, 2000.

[13] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell System Technical Journal*, vol. 28, pp. 59–98, 1949.

[14] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldahna, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: a system for sequential circuit synthesis," ERL MEMO. No. UCB/ERL M92/41, EECS UC Berkeley CA 94720, 1992.

[15] "ISCAS'89 benchmark circuits information," Available from `http://www.cbl.ncsu.edu`.

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

[17] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *European Conference on Design Automation*, 1992, pp. 214–218.

[18] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.