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. Unlike duplication schemes, wherein a replica of the original FSM acts as a predictor-comparator that immediately detects potential faults, the proposed method selects and optimizes only a minimal portion of the original FSM, adequate to detect all possible faults, yet at the cost of potential fault detection latency. Furthermore, in contrast to concurrent error detection approaches, which presume the ability to re-synthesize the FSM and exploit parity-based state encoding, the proposed method is non-intrusive and does not interfere with the state encoding and implementation of the FSM. Experimental results on FSMs of various sizes and densities indicate that the proposed method detects more than 92% of all faults with an average latency of 4 clock cycles and more than 99% of all faults with an average latency of 35 clock cycles. Furthermore, a hardware overhead cost reduction of up to 30% is achieved, as compared to duplication.

1. Introduction

Electronic circuits are employed in a wide variety of modern life activities, ranging from mission critical applications, where the slightest malfunction may have catastrophic consequences, to simple commodity devices, where a potential malfunction may cause no damage other than the inconvenience of discarding and replacing them at an almost negligible cost. As a result, circuit designers are faced with a broad spectrum of dependability, reliability, and testability requirements, which now become system specifications similar to speed, cost, power consumption, etc. Consequently, several approaches of various cost and efficiency levels have been devised to provide an indication of the operational health of a circuit.

While the ability of a circuit to test itself during its intended functionality is a highly desirable attribute, designing a self-testable circuit which conforms to the rest of the design specifications is not a trivial task. Parameters to be considered include the additional hardware cost and design effort incurred, potential performance degradation due to interference of the circuit logic and the self-test logic, as well as the level of assurance that the circuit is required to provide. In this paper, we focus our interest on controller circuits and we explore the trade-offs between the aforementioned parameters, in order to devise a non-intrusive design methodology for concurrently self-testable FSMs. Non-intrusiveness implies that logic may only be 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 is expected to be able to test and detect any fault that may appear in the circuit, therefore rendering a self-testable design. Moreover, self-test needs to be performed concurrently and may not delay or degrade the normal functionality of the FSM.

To motivate the proposed methodology, we first examine related work in the areas of concurrent self-test, concurrent error detection, and on-line test. While the ability to detect all faults is a unifying theme across almost all previous research efforts in these areas, they typically address two ends of a spectrum. Towards the low end, we find low cost concurrent test methodologies, which provide coarse information once the whole circuit has been examined, therefore incurring tremendous fault detection latency. More specifically, [1] proposes the C-BIST method, which employs an input monitoring technique and existing off-line Built-In Self-Test hardware, such as LFSRs and MISRs, to perform concurrent test. While the hardware overhead is very low, the method relies on an ordered appearance of all possible input vectors before any indication of circuit correctness can be provided, resulting in very large fault detection latency. This problem is slightly alleviated in the R-CBIST method described in [2], where the ordered appearance requirement of all input combinations is revoked at the cost of a small RAM employed for input monitoring. Nevertheless, all input combinations still need to appear before any indication of the correctness of the circuit functionality can be provided. Additionally, though inexpensive and non-intrusive, such methods are limited to combinational circuits.

Towards the high end, we find expensive concurrent error detection methods for sequential circuits that provide fine-grained information once every clock cycle, therefore guaranteeing zero error detection latency. Duplication is the simplest of these methods, limited however by its ex-

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

pensive hardware overhead. Reducing the area overhead below the cost of duplication typically requires redesign of the original FSM, thus leading to intrusive methodologies. One of the first successful attempts along this direction is described in [3], where resynthesis is employed to favorably encode the FSM states, incorporating parity information in the FSM and employing TSC checkers and several alternative encoding schemes. Limitations of this method such as structural constraints requiring an inverter-free design were alleviated in [4], where a single parity bit and several circuit partitions are employed to reduce the incurred hardware overhead. Utilization of multiple parity bits is examined in [5] within the context of finite-state machines, leading to the most encouraging results so far. All these methods render totally self-checking circuits and guarantee error detection with zero latency but they alter the original FSM design and typically only provide hardware savings in the range of 10%, if at all, as compared to duplication.

While each of the two extremes has both strong and weak points, very few attempts have been made to bridge the gap. Among them, a method based on properties of non-linear adaptive filters is proposed in [6], achieving a 20% cost reduction with minimal latency penalty. A similar technique is proposed in [7], where the frequency response of linear filters is used as an invariance property of the circuit. The corresponding scheme achieves a 50% cost reduction but introduces a more significant latency penalty. Finally, a concurrent test approach exploiting transparency behavior of RT-Level components is described in [8], where over 90% fault security is achieved with 40% hardware overhead. However, no estimation is given for the fault detection latency, although a few heuristics are described for selecting frequently activated transparencies.

2. Proposed Method

The method proposed in this paper attempts to further bridge the aforementioned gap by introducing a non-intrusive method for designing concurrently self-testable FSMs, where the original implementation of the circuit (i.e. state encoding and next state logic) may not be modified. While coverage may not be sacrificed, 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. Therefore, information is provided frequently, yet not at every clock cycle. While such circuits are not *fault-secure* and therefore do not guarantee correctness of the results, they are still *self-testable*, thus guaranteeing that there is a way to eventually detect any potential fault. In the rest of this section, we describe the proposed methodology and we provide an analysis and justification of its expected performance.

2.1. 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 to the design and the results of the two identical FSMs are compared at each clock cycle, thus detecting any error in the design through a monitoring output that becomes '1' if an error occurs. To avoid fault masking by common-mode failures, design diversity [9] has also been examined, wherein the duplicate FSM is functionally equivalent but structurally different and possibly more expensive than the original FSM.

In an effort to reduce the hardware 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 and a complete set of test vectors is obtained [2]. These test vectors are subsequently synthesized, rendering the prediction logic that will provide the expected next state of the original FSM when an input / previous state combination matches a test vector. In order to reduce the required hardware, input / previous state combinations that are not test vectors are treated as *don't cares* during synthesis. However, an additional function is now required, indicating whether an input / previous state combination is a test vector and directing (through the AND gate in Figure 1) whether the comparison outcome is a valid indication of correct functionality. Furthermore, 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 [5]. 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 for the extra function. Notice also that, in order to avoid false alarms, the predicted next state calculation is driven by the original FSM state register and not by the predicted state register. This is because the predicted state register may not contain the correct value after an input / previous state combination that is not a test vector, since the output for such combinations is arbitrarily decided during synthesis to minimize the prediction logic hardware.

As shown in Figure 1, unlike approaches such as [3, 4, 5] that attempt to reduce the hardware overhead by re-encoding the FSM using parity schemes, the proposed method leaves the original FSM design intact. Furthermore, assuming a sizeable yet not exhaustive set of vectors and despite the addition of one extra function, we expect a con-

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

[2] In the presence of a fault a state may become unreachable in an FSM. If ATPG selects such a state as part of the test vector that tests this fault, the fault becomes untestable. A state reachability analysis in the presence of each fault is required to preclude such vector/fault pairs. Our experimental observation is that such cases amount to less than 0.5% of all faults.

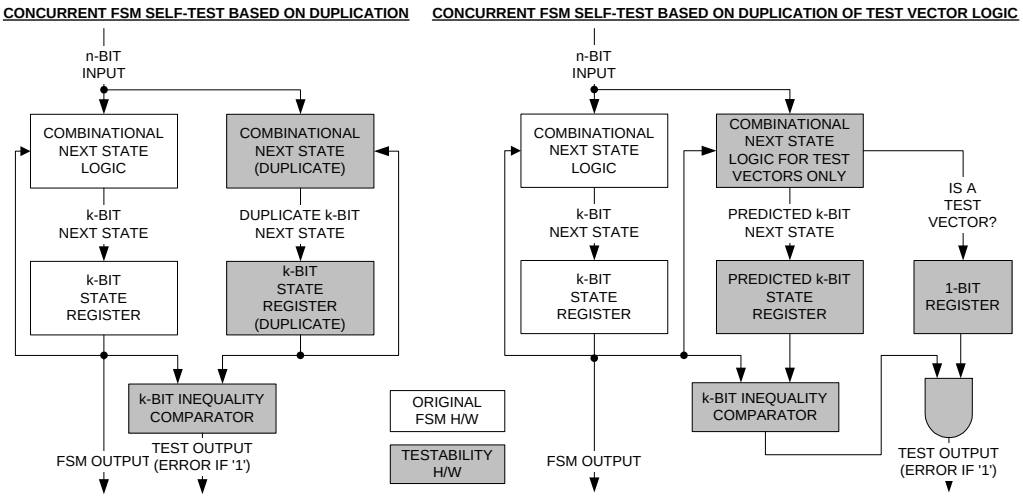

Figure 1. Duplication vs. Proposed Methodology

siderable hardware overhead reduction. On the down side, a fault may cause an error which will go undetected until a test vector detecting this fault appears, thus introducing fault detection latency. Given a sizeable vector set, tests are expected to be performed frequently and low average latency is expected. The following paragraph discusses and analyzes quantitatively these expectations.

2.2. Analysis

In this section we predict the performance of the methodology (fault coverage, hardware overhead and fault detection latency) proposed in Section 2.1. We focus mainly on the hardware overhead of the proposed scheme. After introducing some notation, we state a rather obvious remark on the fault coverage for the "stuck-at" fault model.

We use n to denote the number of input bits of a Finite State Machine (FSM), k to denote the number of state bits, $N = 2^n$ to denote the number of possible inputs and $K = 2^k$ to denote the maximum number of states of the FSM. We also use $s_i, i = 1 \dots 2^k$ to denote the states of the FSM and $v_i, i = 1 \dots 2^n$ to denote possible input vectors. We denote the original circuit implementing the NEXT STATE LOGIC of the FSM by A and the prediction logic by P. The prediction logic is split in two parts, the P_{NS} part, which predicts the next state and the P_{ISV} part, which decides whether the predicted next state is valid.

Remark 1 *Assuming that we can generate a set of test vectors that detects* 100% *of the faults in* A*, our methodology guarantees the same fault coverage as duplication does.*

Observe that A and P do not share any hardware. In the presence of non-redundant faults in A, P[3] is fault free; we also know that there exists some test vector (essentially a pair (s_i, v_i)) that detects the fault. Thus, when A reaches s_i and the next input is v_i, P and A will output different next states and the fault will be detected. We stress that no latency guarantees can be given; indeed, the fault in A might appear early and go undetected until (s_i, v_i) appears.

We now outline our intuition on the hardware overhead of the methodology. Our first remark relates the hardware[4] required to implement a random boolean function with n input bits and one "fully-specified" output bit vs. the hardware required to implement a random function with n input bits and one "a-specified" output bit. We start with the following definition:

Definition 1 *An* a*-specified output bit has at least* $\lceil a2^n \rceil$ don't cares *(*$a \in [0, 1]$*). The positions of the* don't cares *are fixed* a priori.

A fully-specified output bit is equivalent to a 0*-specified output bit.*

Remark 2 *Almost all boolean functions* $f : \{0,1\}^n \rightarrow \{0,1\}$ *require at least*

1. $2^n/n$ *gates, if the output bit is fully-specified.*
2. $(1-a)2^n/n$ *gates, if the output bit is* a*-specified.*

Proof (Sketch): The first statement is Shannon's counting argument [10]. For the second statement, we observe that the number of different functions $f : \{0,1\}^n \rightarrow \{0,1\}$ with $\lceil a2^n \rceil$ *don't cares* is $2^{\lceil (1-a)2^n \rceil}$. Thus, the same counting argument proves our statement.

We emphasize that for our statement to hold the positions of the *don't cares* are pre-specified.

◇

[3] The other direction (faulty P and fault free A) is similar.

[4] We assume that all functions are implemented as multi-level circuits using 2-input gates.

Our second remark relates the hardware required to implement one fully-specified bit with the hardware required to implement k fully specified bits when k is a small constant ($k << 2^n$) and the bits are *uncorrelated*.

Remark 3 *Almost all boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}^k$ require at least $k2^n/n$ gates* if *the k output bits are uncorrelated and fully specified.*

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

◇

The NEXT STATE LOGIC of our FSM is a combinational circuit with $n + k$ inputs and k outputs (the next state). Our technique generates a set of test vectors for the NEXT STATE LOGIC, which can be viewed as a function $f : \{0,1\}^{n+k} \rightarrow \{0,1\}^k$. Assume for the moment that the k output bits are uncorrelated and the cardinality of the test vector set is $a2^{n+k}$. Then, using remark 3, the hardware required for P_{ISV} would be $1/k$ times the hardware required for the NEXT STATE LOGIC. Similarly, the hardware required for P_{NS} would be $(1-a)$ times the hardware required for the NEXT STATE LOGIC. Thus, the *minimum* hardware required for P_{NS} would be $(1-a+1/k)$ times the *minimum* hardware required for the NEXT STATE LOGIC. Depending on a and k, the hardware overhead of our technique might be significantly smaller than duplication.

In practice, the k state bits representing the next state are correlated (otherwise some states of our FSM would not be reachable). It is not clear though that as the state bits become more and more correlated the size of P_{NS} over A increases; one could expect the size of P_{NS} to decrease as correlation increases. On the other hand, it seems natural that the size of P_{ISV} over A would increase. We also note that we can only examine how the *lower bound* of the size of P is diminishing; indeed, tight bounds for circuit sizes are notoriously hard to prove even under stringent assumptions. Finally, P_{NS} and P_{ISV} are not implemented separately; indeed, in order to minimize the cost of P we try to maximize their sharing. Thus, we expect the cost of P to be less than the sum of the individual costs of P_{NS} and P_{ISV}.

Although it is essentially impossible to predict the latency of our technique, we would like to stress that, assuming random inputs, our technique checks for faults in a clock cycle with probability a. In our experiments, typical values for a range from 0.4-0.5; practically, half of the 2^{n+k} possible vectors are test vectors! This empirical observation can also be justified using results of [11]. Thus, we are checking in almost every other clock cycle. Also, we may reasonably assume that most "stuck-at" faults are detected by many test vectors. Thus, we expect the latency of our scheme to be very small on average; in Section 3.3 we see that this prediction is experimentally justified.

3. Experiments

In an effort to assess experimentally the proposed methodology, we compare it to the traditional duplication scheme in terms of area overhead, fault coverage, and fault detection latency. The experimental setup, comprising tools and FSMs employed for this purpose, is described in this section, followed by a presentation and discussion of the obtained results.

3.1. Setup

The experimental setup employed is shown in Figure 2. For the purpose of preserving generality, the experiments are applied on random FSMs, generated using MATLAB [12]. The characteristics of these FSMs are described in the following subsection. Subsequently, these FSMs are synthesized and optimized using the *rugged* script of the SIS system [13], mapped to a standard cell library, and converted to ISCAS89 [14] format. This step serves only as a standard method to provide us with the actual implementation of the FSMs and no assumptions as to how the FSMs are encoded and optimized are made. Since the proposed methodology is non-intrusive, the hardware implementation of the FSMs is our starting point.

Given the implementation of an FSM, the combinational next state logic is extracted and a combinational ATPG run is performed on it using ATALANTA [15]. Under the assumption that the next state logic comprises no redundancies, a complete test set achieving 100% fault coverage is obtained. These vectors, along with an additional function that indicates whether an input combination is a test vector, are then synthesized using the *rugged* script of the SIS system [13], mapped to a standard cell library, and converted to ISCAS89 [14] format, rendering the hardware implementation of the prediction logic for the proposed method. Since the prediction logic is the only difference between the proposed method and duplication, a comparison to the next state logic of the FSM - which is used for prediction in duplication - indicates the hardware savings of the proposed method. Subsequently, the proposed self-testing FSM is constructed, and a sequential ATPG run using HITEC [16] is performed to provide the total number of faults and the number of testable faults in the circuit. Notice that these numbers comprise faults both in the original FSM and the self-testing logic.

As discussed in the previous section, while the proposed method guarantees, by construction, that all non-redundant faults in the circuit are testable by the self-testing mechanism, the hardware savings over duplication come at the cost of fault detection latency. Similar to [1], in order to calculate the average fault detection latency, we generate sets of random inputs, which we fault simulate twice on the

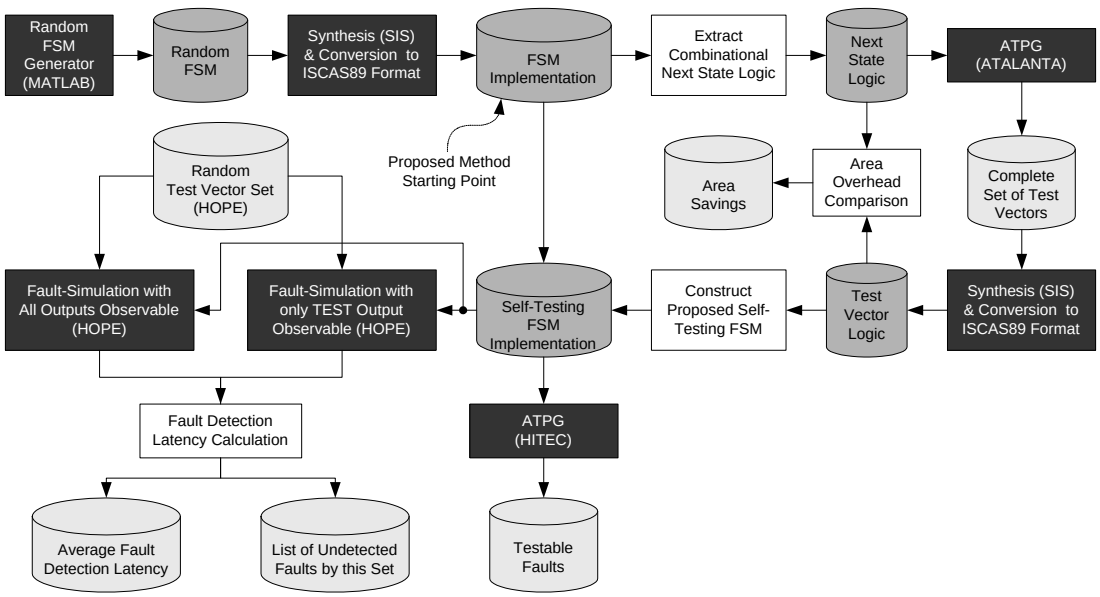

Figure 2. Experimental Setup

constructed FSM, the first time observing all outputs (including the TEST output) and the second time observing only the TEST output. For fault simulation we use HOPE [17], which reports the newly detected faults for each simulated vector. The two fault simulation reports are, then, processed and the time step difference between detecting each fault in the first and second fault simulation is obtained and averaged over all faults. The result indicates the average fault detection latency for the simulated random set of vectors. The number of faults that are only detected in the first fault simulation but not in the second, i.e. faults that the proposed method did not detect within the limited number of simulated test vectors, is also reported.

3.2. Random FSMs

We briefly outline the process of generating random FSMs with K states and n inputs (for notation see Section 2.2). We start by building the "connected component" of the FSM, to guarantee that there exists a path from some ROOT node to every state.

Starting from the ROOT, we add a random number r of children to each state-node; r is picked uniformly at random from $0 \ldots N$ and independently for each state-node. We visit the states-nodes in a breadth-first search order and we stop when a full tree with all K states is built. We subsequently add $N - r_i$ edges from state-node s_i to other nodes in the tree[5], where r_i is the number of children of s_i. Finally, we label the K states-nodes using a random permutation of $1 \ldots K$; we also label the N out-edges from each s_i using random permutations of $1 \ldots N$.

[5]We pick these nodes uniformly at random with replacement.

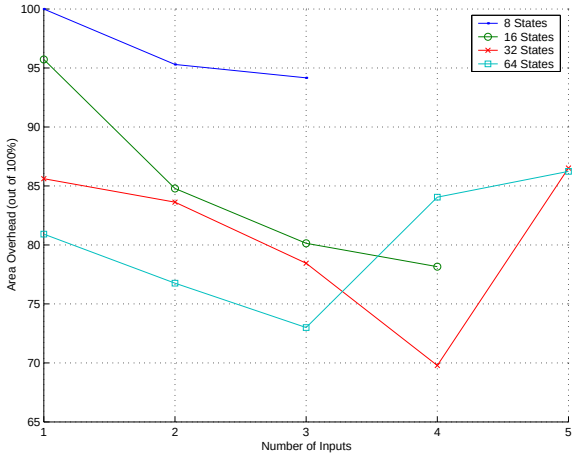

Figure 3. Average Area Overhead = 84%

Our objective is to build large and complex FSMs in order to test our technique; one could potentially suggest different strategies for the same task. It is important though to note that our strategy can generate *all possible* FSMs with K states and n inputs.

3.3. Results

As we mentioned in Section 2.2, our technique guarantees the same fault coverage as duplication does (subject to the restriction of footnote 2). Thus, our experiments focus on the hardware overhead and the latency of our scheme.

We ran our experiments on 17 different "types" of FSMs, namely 17 different (K, n) combinations. We remind that K denotes the number of states and n the number of input bits. The "types" are (8,1), (8,2), (8,3), (16,1), (16,2),

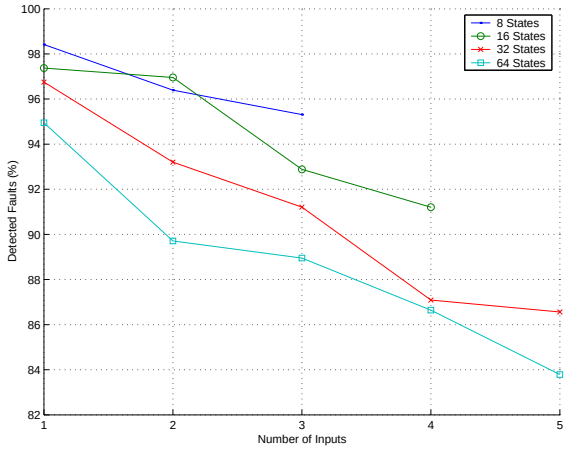

Figure 4. The proposed method detected 92.2% of the faults that appeared in the first 100 cycles.

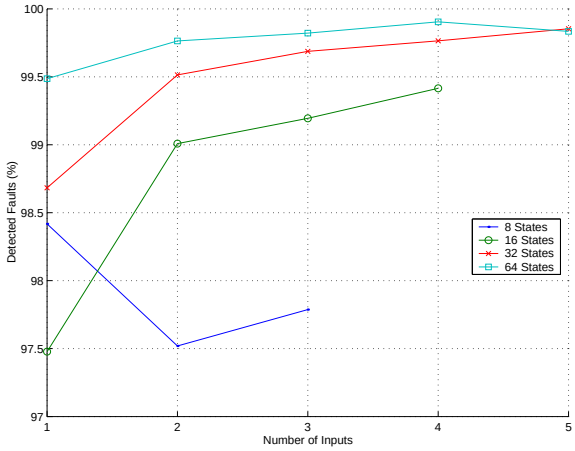

Figure 6. The proposed method detected 99.1% of the faults that appeared in the first 10000 cycles.

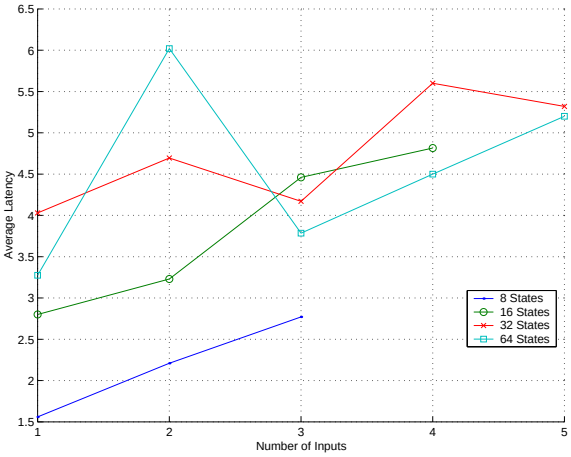

Figure 5. The average latency for faults detected in the first 100 cycles was 4.02 cycles

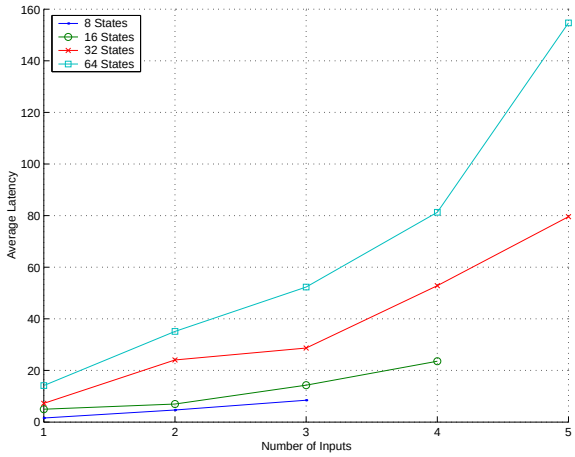

Figure 7. The average latency for faults detected in the first 10000 cycles was 34.97 cycles

(16,3), (16,4), (32,1), (32,2), (32,3), (32,4), (32,5), (64,1), (64,2), (64,3), (64,4), (64,5). We generated 2 FSMs for each type and we report the average. Figure 3 shows the area overhead of our technique over duplication. We present 4 curves, for FSMs with 8, 16, 32 and 64 states.

To report results regarding the latency, we examine our FSMs using 100 random input vectors. Figure 4 shows the percentage of non-redundant faults that appeared *and* we were able to detect using our scheme. Figure 5 shows the average latency for the faults that we were able to detect. We should stress that not all possible faults appeared in this small simulation; our data though indicate that on average almost 84.16% of the faults did appear! The reported latency is the average latency over the detected faults.

We now examine our FSMs using the previous 100 random input vectors *and* 9900 additional random vectors. Figures 6,7 show the percentage of non-redundant faults that we were able to detect and their average latency. On average, 99.6% of all non-redundant faults appeared!

One can see that the latency of our technique is small, as predicted in Section 2.2. We will now show that we also predicted the hardware overhead quite accurately. We remind that we predicted the ratio of our methodology over duplication to be $1 - a + 1/k$. We remind that a is the ratio of the 2^{n+k} (state, input) combinations that are test vectors. Tables 1 and 2 show our prediction for the specific values of a (as generated by our experiments) and k for each FSM.

On average, our prediction and the observed hardware overhead differ by 9.78%. We remind that our prediction does not take into account the potential sharing between the P_{NS} and P_{ISV}; thus, the difference might be even smaller. We believe that the discrepancy is the effect of two factors: correlation of next state bits (see Section 2.2) and inability of the tools used to fully optimize P_{NS} in large circuits.

FSM	8,1	8,2	8,3	16,1	16,2	16,3	16,4	32,1	32,2
Test Vectors	12	20.5	35	21	35	58	119	37	67.5
Prediction (%)	108	97	88	91	80	70	71	78	73

Table 1. PREDICTED OVERHEAD

FSM	32,3	32,4	32,5	64,1	64,2	64,3	64,4	64,5
Test Vectors	125	233	481	71.5	125	242	491	959
Prediction (%)	69	66	67	73	65	64	65	63

Table 2. PREDICTED OVERHEAD **(cont'd)**

4. Conclusions

Cost-efficient design of FSMs with embedded concurrent fault detection capability necessitates a careful consideration of the conflicting objectives of low hardware overhead, low fault detection latency, and high fault coverage. The corresponding trade-offs become even more stringent in the presence of design specification constraints requiring that the original FSM implementation be left intact. Along these lines, we discussed a test vector logic replication and optimization methodology for designing FSMs that can be fully self-tested concurrently with their normal functionality, without paying the cost of duplication and without interfering with the original FSM implementation. Experimental results demonstrate the ability of the proposed methodology to reduce the hardware overhead incurred by duplication schemes by as much as 30%, while preserving the ability to detect all faults in the circuit, yet at the cost of non-zero latency. Nevertheless, the experimentally observed average fault detection latency of the method is very low, ranging from 4 clock cycles for 92% of the faults to 35 clock cycles for 99% of the faults.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

[12] "MATLAB," Available from `http://www.mathworks.com/`.

[13] 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.

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

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

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

[17] 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.