Concurrent Self-Test with Partially Specified Patterns For Low Test Latency and Overhead

Michael A. Kochte, Christian G. Zoellin, Hans-Joachim Wunderlich
Institute of Computer Architecture and Computer Engineering, University of Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany

***Abstract*—Structural on-line self-test may be performed to detect permanent faults and avoid their accumulation. This paper improves concurrent BIST techniques based on a deterministic test set. Here, the test patterns are specially generated with a small number of specified bits. This results in very low test latency, which reduces the likelihood of fault accumulation. Experiments with a large number of circuits show that the hardware overhead is significantly lower than the overhead for previously published methods. Furthermore, the method allows to trade-off fault coverage, test latency and hardware overhead.**

***Index Terms*—Concurrent self test, BIST, test generation**

I. Introduction

Recent process technology nodes have shown increased variability as well as the susceptibility to transient and permanent faults [1–3]. Hence, many classical hardware based fault tolerance techniques are getting now new attention for yield and reliability improvement [4–6]. While on-line checking or on-line monitoring methods observe an encoded output space for detecting any error, on-line test techniques target structural faults to identify defective components [7]. Encoding for on-line checking up to duplication and comparison may be rather expensive in terms of hardware while structural on-line testing is restricted to certain fault models and less expensive in general.

Dormant faults and subsequent fault accumulation can decrease the system reliability [8] and require additional on-line test techniques [9]. On-line techniques that use a deterministic, precomputed test set are based on a fault model and allow to specifically target critical faults.

Non-concurrent on-line BIST executes a structural test only in periodic intervals or while the circuit is idle. The early deterministic scheme in figure 1 is called Store-and-Generate [10]. It keeps the aforementioned advantages and leads to rather low test time. If more advanced deterministic BIST schemes like [11–13] are employed, special care has to be taken that the CUT stays resumable, but test time will decrease even further. Thus, the test can be applied more frequently and the latency of fault detection is reduced [14]. Furthermore, special architectures make application of the deterministic test set less intrusive [15–18].

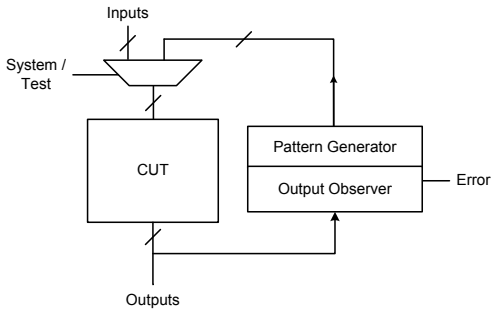

Fig. 1. Deterministic non-concurrent on-line test

Non-concurrent techniques can only be applied if the DUT is sufficiently idle.

Concurrent on-line self-test methods use the functional input vectors to execute a structural test over a longer time period. Duplication and comparison is a special, expensive implementation of the scheme in figure 2. Here, the pattern observer is just a copy of the CUT, the output observer is a pairwise comparator, and the error signal is generated by a disjunction of all the outputs. This leads to a significant hardware overhead of around 120% in the examples reported in section 5.

Hardware can be saved, if only a subset of patterns are monitored and the CUT is not duplicated completely. Saluja et al. have proposed a technique based on a pseudo-random test sequence [19, 20]. Here, a monitoring circuit checks the circuit inputs for vectors which are part of a pseudo-random test sequence (figure 2). Whenever such an input vector occurs, the circuit response is included in a test signature. In order to reduce the latency of detecting faults, deterministic test sets can be applied instead [21–24].

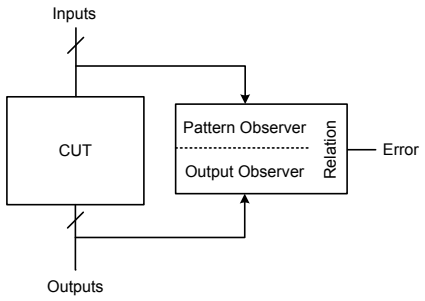

Fig. 2. Deterministic concurrent on-line test

The test set determines the test latency and fault coverage, but also the hardware overhead. Test latency is the time required to test all targeted faults at least once. The scheme is applicable to data-path logic and finite state machines (FSMs) as well. If in FSMs only state transition of branches are critical, a technique from processor control [25] can be applied in hardware. The technique compresses the target states in signatures and is known as control flow monitoring [26].

The size of the test set directly corresponds to the hardware of the observers in figure 2 and special care is required during its generation. Therefore, the authors of [22, 23] enumerate all possible tests for each fault and find a minimum cover. This effectively minimizes the hardware overhead of the observers. For larger circuits, [24] proposes to use test sets from regular ATPG, but acceptable overhead is achieved only in a few, rare cases. Moreover, for circuits with more than 32 inputs, completely specified patterns result in a fault latency that is several orders of magnitude too high.

The method presented here modifies a deterministic test set by reducing the number of specified bits such that it can be applied to

the schemes of figures 1 and 2. Algorithms for test pattern generation [27, 28] and test set modification [29–31] are available targeting similar goals. In contrast to the existing test generation methods, the technique allows to bound the test latency and it can take advantage of the trade-off between fault coverage and hardware overhead.

The test set is computed from an ATPG test set by unspecifying bits that are not required for fault detection, and by splitting patterns. The large number of unspecified bits results in a hardware overhead that is significantly lower compared to previously published methods based on deterministic test sets. Furthermore, the method is completely automated and can synthesize self-testable circuits for given random logic macros. The proposed method is evaluated with a wide set of circuits. If for a given application a fault coverage of 90% is sufficient, a large number of circuits can be tested with less than 50% overhead. Moreover, the results show that hardware overhead is as low as 8% and 80% on average when providing complete fault coverage as compared to 120% for duplication.

Due to lack of space, the method is only applied to the concurrent on-line test scheme in this paper. However, the generalization to the non-concurrent case is straightforward by exploiting the don't cares as described in [32].

The rest of the paper is organized as follows: Section 2 shows how the input and output observer are synthesized from partially specified patterns. Section 3 shows the overall flow of the synthesis procedure and section 4 presents how to estimate the test latency for such a circuit. The method is evaluated using a set of benchmark circuits in section 5.

II. Input/output relation generated from partially specified patterns

The input/output observer of Fig. 2 determines if the input vector I and the corresponding output response O are in the correct relation $\Phi(I, O)$. This relation is easily expressed by a Boolean function (Fig. 3). For example, in the case of duplication the relation matches the Boolean function of the CUT.

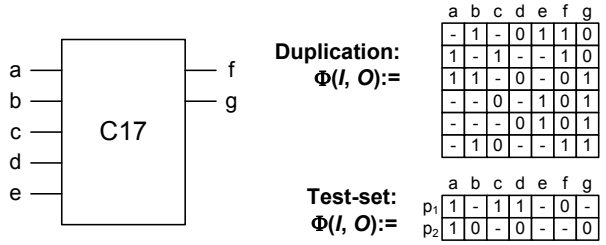

Fig. 3. Input output relation with partially specified patterns

In order to reduce the overhead for the relation $\Phi(I, O)$, the relation may be derived from a set of deterministic patterns that detect a given set of faults. In the example (Fig. 3), the partially specified test set contains two patterns for the well-known benchmark circuit C17. The output values are derived from logic simulation of these patterns

If partially specified patterns are used in the input monitor, then a single pattern matches a very large set of input vectors. Each additional bit left unspecified in a pattern increases the likelihood that the pattern occurs by a factor of 2.

The large input don't care space significantly contributes to optimization by the synthesis tool. Moreover, due to common sensitization criteria many patterns in a test set may be very similar. Hence, much of the logic in the monitors may be shared and it is beneficial to synthesize input and output monitor together. In existing techniques, the relation is usually implemented as a RAM, ROM or PLA [24, 33]. But since they are based on completely specified test patterns, almost no optimization is possible.

The integration of the pattern observer and output observer of figure 2 to the Boolean representation of the input/output relation in figure 3 has benefits at the input and output side at the same time.

1) Input side: Most pattern monitoring techniques need decoding logic for indicating a test pattern. The decoding logic is implicitely integrated and exploited during logic synthesis.

2) Output side: An additional comparator is not needed. For each pattern, comparison is only done for the subset of relevant output literals, and this logic is also integrated and used during logic synthesis.

III. Generation of the Concurrent BIST Circuitry

The deterministic test set from which the monitors are derived has a major impact on both hardware overhead and test latency. In the following we briefly outline the overall flow of the generation of the concurrent BIST circuitry and then discuss the steps in more detail. Figure 4 gives an overview of the process. The input for the self-test generation procedure are the circuit description, the set of targeted faults and a precomputed test set of completely or partially specified test patterns. The flow allows to specify a bound on the test latency and the targeted fault efficiency for the resulting scheme. The test latency is determined by the maximum number of specified bits in the patterns. Fault efficiency is the ratio of covered faults and detectable faults. The two parameters allow to trade-off coverage, latency and hardware overhead.

The generation of the concurrent BIST test set comprises four steps:

1) Generation of a partially specified test set with as few specified bits as possible and a strict limit on the number of specified bits per pattern.
2) Selection of patterns from this test set such that the targeted fault efficiency is achieved.
3) Selection of a subset of output values to be compared.
4) Generation of the input/output relation of the monitor.

The first step in the process is accomplished by test set stripping. Test set stripping is the identification of those bits in the test set that are not required to be specified for fault detection. The method used here allows to strictly limit the maximum number of specified bits in any pattern. To achieve the given limit, additional patterns may be created. Enforcing this limit bounds the test latency of the final concurrent test.

From the resulting test set, a subset of patterns is chosen such that a certain fault coverage or efficiency is achieved. In this second step, we propose a greedy heuristic that results in low hardware overhead for the requested target coverage.

In the third step, the outputs to be compared by the output monitor are selected. For many test patterns, a very small number of outputs is sufficient to detect all faults targeted by the pattern. Hence, the final test set is analyzed with fault simulation and a subset of the circuit response is selected with a simple heuristic.

After this step, the concurrent BIST circuitry is generated in step 4. During the whole self-test generation process the original circuit is not altered in any way.

1. Test set stripping

Several algorithms have been proposed for test set stripping resp. relaxation [29–31, 34]. In this paper, we adapt the stripping method

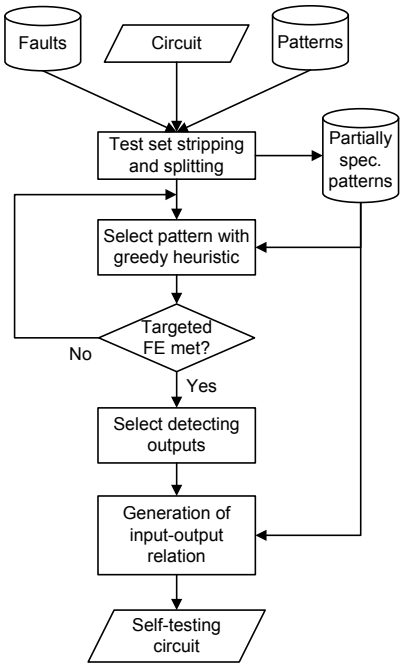

Fig. 4. Generation of self-testing circuit

from [31] which allows to limit the number of specified bits in a test pattern and uncovers a significant amount of over-specified bits in excess of the number of already unspecified bits in the original test set. In contrast to constrained ATPG, test set stripping avoids the computational complexity associated with ATPG.

Using the method presented in [31], the resulting maximum number of specified bits in a pattern can still sometimes exceed 100 bits. When applied to input vector monitoring, the likelihood that such a pattern occurs in the input vectors of the circuit is 2^{-100}. Hence, a viable concurrent BIST requires that a very low limit is strictly enforced. In contrast to the algorithm in [31], the extension presented here obtains a significant reduction in the maximum number of specified bits in a pattern.

To achieve this, the method has been extended with pattern splitting to avoid a loss in fault coverage. The concept of pattern splitting was first proposed in [29], yet that algorithm did not target the maximum number of specified bits. Pattern splitting occurs if the faults assigned to a pattern would result in a number of specified bits in excess of the limit. Pattern splitting results in a duplication of the originating pattern. Faults can then be assigned to the two duplicates. This process is repeated if necessary.

The splitting algorithm is outlined in Alg. 1 and invoked after test set stripping. Let F be the set of target faults. Then for each fault f not detected by the stripped test set we search for a detecting pattern p in the original test set that detects f with lowest cost, i.e. with a minimum number of specified bits. If the number of specified bits is below the given limit, p is duplicated and the bits required for detection of f are identified. The resulting pattern p' is added to the final test set T_p. Fault simulation of p' helps to find additional faults detected by p'.

Since the stripped and duplicated patterns originated in the same pattern of the test set, the similarity of these patterns can be exploited during the synthesis of the concurrent BIST circuit.

The resulting set of partially specified test patterns is input to the next step:

Algorithm 1 Pattern splitting

```
for all f ∈ F do
  if f is undetected then
    Let L_f := {p ∈ T | p detects f}
    Select p ∈ L_f with minimum cost(p, f)
    if cost(p, f) ≤ limit then
      p' := duplicate(p)
      identify_req_bits(p', f)
      T_p := T_p ∪ {p'}
      Fault simulation on specified bits of p'
    end if
  end if
end for
```

2. Selection of patterns

In some cases, the designer may choose to sacrifice some of the fault coverage achieved by the original test set and significantly reduce hardware overhead. For this, a heuristic selects a subset of patterns from the stripped test set until a given fault efficiency has been reached.

First, the test set from step one is analyzed with fault simulation. For each pattern we determine the set of faults that it is able to detect. Then, we repeatedly add the pattern that detects the highest number of additional faults to the final test set until the desired fault efficiency is achieved.

This straightforward heuristic provides the most consistent results across a wide variety of benchmark circuits. Alternative heuristics can weight patterns with number of care bits for each additional fault, or use the hamming distance to the most similar pattern to take into account logic sharing. In our experience, the logic synthesis of the final BIST function with commercial synthesis tools does not benefit from these heuristics.

3. Selection of outputs

Now, the final test set completely defines the input part of the relation in section II. For the output part, it is sufficient to use the correct response from the logic simulation of the test set. But often, only very few outputs have to be compared to detect all the faults testable by a pattern.

In this step, the method iterates over all the patterns. For each pattern, fault simulation yields the outputs at which the fault effects can be observed. If none of these outputs has already been selected for observation, the list of detecting outputs is ordered and the first output is selected. Here, the outputs are ordered according to the original circuit description.

Instead of this heuristic, set covering may be used to find a minimum set of outputs for each pattern. But similar to the pattern selection in step 2, there is no consistent advantage of the more complex approach when the results from final logic synthesis are taken into account.

4. Generation of Input/Output Relation

After selection of the outputs, the function of the Boolean relation for the monitor is derived.

For each test pattern t of the final test set T_p we now derive the relation with respect to the input vector I of the circuit under test as

$$\Phi_t(I, O) := (t == I) \wedge (O_t == O)$$

where O_t is the output vector constructed from the selected outputs in step 3 and O is the output vector of the CUT. The operator $==$ is defined as the equality of the two vectors taking into account don't cares.

The error signal of the circuit is now simply

$$error := \bigvee_{t \in T_p} \neg \Phi_t(I, O) \ .$$

The concurrent BIST circuit is then synthesized with the regular logic synthesis flow. The large number of don't cares as well as the large commonalities between input and output relations gives the synthesis a large degree of freedom to perform efficient synthesis.

IV. Concurrent test length

To evaluate the test length of the concurrent test scheme we use two metrics. Firstly, we determine the expectation value of the number of cycles required for completion of the concurrent test as proposed in [35]. Secondly, we employ a well-known metric from pseudo-random testing. We compute the test length required to achieve a given confidence level such that no targeted fault resp. pattern in the test set escapes. The required test length is a more meaningful metrics to determine the fault latency.

The considerations are based on three assumptions also found in [24, 35, 36]:

- All possible input patterns for the circuit occur in the input stream.
- A pattern occurs in every clock cycle.
- All patterns occur with the same probability.

A. Computation of the expected value of the test length

The expectation value of the number of input vectors required for completion of the concurrent test is called the concurrent test length (CTL). The concurrent test is completed when every pattern of the test set has occurred as input vector of the circuit at least once.

Let test set T_c consist of t completely specified patterns for a circuit with n inputs. $N = 2^n$ is the number of all possible input patterns for that circuit. The test is completed when every of the t patterns has been hit, i.e. occurred at least once in the input stream.

The mean number of cycles required to hit all of the t completely specified patterns in T_c at least once is derived in [35]:

$$CTL = \sum_{i=1}^{t} \frac{N}{i} = N \sum_{i=1}^{t} \frac{1}{i} \tag{1}$$

Now let test set T_p consist of t partially specified patterns for a circuit with n inputs. The number of specified bits in a single pattern $p \in T_p$ is denoted as n_p. Let $p_{max} \in T_p$ denote the pattern with the maximum number of specified bits $n_{pmax} = max_{p \in T_p}\{n_p\}$. p_{max} is the pattern least likely to be hit by an input vector.

To compute an upper bound of the CTL we will assume that all t patterns of T_p have the same number of specified bits as p_{max}. Thus, the probability h_p that a pattern is hit is assumed to be identical for all t patterns:

$$h_p = h_{pmax} = \frac{2^{n-n_{pmax}}}{N} = 2^{-n_{pmax}}, \tag{2}$$

where $2^{n-n_{pmax}}$ is the number of completely specified patterns covered by p_{max}.

Then, following the reasoning of [35], an upper bound of the CTL of T_p can be computed as

$$CTL = \sum_{i=1}^{t} \frac{1}{h_p \cdot i} = \frac{1}{h_p} \sum_{i=1}^{t} \frac{1}{i} = 2^{n_{pmax}} \sum_{i=1}^{t} \frac{1}{i} \tag{3}$$

Similar to [24] we conducted a series of simulations on different circuits and test sets and successfully validated the computation of the CTL.

B. Required test length

To compute the test length required to achieve a given fault escape probability we apply concepts known from pseudo-random testing.

In pseudo-random testing, the detection probability q_f of a fault f depends on the size of the test set T_f of the fault, i.e. the number of patterns that detect it:

$$q_f = \frac{\mid T_f \mid}{2^n}. \tag{4}$$

Assume that all faults have disjoint test sets. The fault with the lowest detection probability resp. smallest test set is called the *worst* fault. A valid simplification for the computation of the required test length is to consider only the set of faults with detection probabilities close to that of the worst fault, i.e. within the factor of 2 from the detection probability of the worst fault [36]. Faults with higher detection probabilities do not significantly impact the required test length.

We can further simplify the computation by assuming that all of these faults have the same detection probability as the worst fault since we want to determine an upper bound for the test length.

Then, $T_k^U(e_{th})$ is an upper bound of the required test length to achieve a fault escape probability not larger than e_{th}, given the k worst faults with disjoint test sets and identical detection probability q [36]:

$$T_k^U(e_{th}) = \left\lceil \frac{ln(e_{th}/k)}{ln(1-q)} \right\rceil \tag{5}$$

Due to the stripping process, the set of partially specified test patterns T_p has the property that for each test pattern $p \in T_p$ there exists at least one fault f which is only detected by p. Thus, the detection probability of f is $q = 2^{-n_p}$ where n_p is the number of specified bits in p. For each of these faults, the detection probability is identical to the probability that the particular pattern occurs in the input stream.

The k patterns that detect the k worst faults are now called the worst patterns. They have the highest number of specified bits in the test set T_p. Then eq. (5) can be used to compute an upper bound of the test length required to achieve a pattern escape probability not larger than e_{th}.

V. Results

The scheme proposed in section III has been implemented into our in-house CAD tool. For test generation and synthesis, we use third-party tools. Firstly, we evaluate the scheme in terms of hardware overhead and test latency for a comprehensive set of benchmark circuits from the ISCAS-85, ISCAS-89 and ITC-99 suite. Secondly, the scheme is compared to duplication as well as the state-of-the-art ATPG test-set based concurrent BIST method presented in [24]. The method presented in [22] is not compared here, since the circuits used for its evaluation are not openly available and it requires complete enumeration of all possible tests.

In this experiment we bound the test latency by strictly limiting the number of specified bits in the test patterns to 32 bits. Given a clock frequency of 500 MHz, the expected duration of the concurrent test ($\approx 3E10$ cycles) is about one minute of system operation. The test sets for the circuits target stuck-at faults and have been generated using the method from [37] and a commercial ATPG tool. The resulting circuitry has been synthesized using a commercial synthesis tool and mapped to the abstract technology library of the tool. The original circuit has been mapped to that technology library as well. The hardware overhead is the ratio of the resulting area of the concurrent BIST circuitry and the benchmark circuit. Here, we

investigate the hardware overhead w.r.t. the targeted fault efficiency. Table I lists the hardware overhead for fault efficiencies of 70%, 80% and 90% as well as for the maximum fault efficiency to be achieved given the limit of 32 bits.

The maximum achieved fault efficiency due to strictly enforcing the limit and the resulting hardware overhead are given in columns 5 and 6. The last column gives the required test length to achieve an escape probability no larger than 0.01, computed according to section IV-B.

The two circuits c499 and c1355, where not even 70% fault efficiency was reached, are both 32-bit SEC circuits consisting of large XOR trees. Most of the faults in these structures require all of the 41 inputs to be specified and cannot be detected with a limit of 32 bits.

For the sequential circuits of ISCAS89 and ITC99, the observation logic is connected to the pseudo-primary inputs and outpus as well. During synthesis, the arrival times of the outputs should be taken into account to reduce the impact on clock frequency. To avoid any impact on the critical path of the circuit, the checking may be moved into the subsequent cycle. Obviously, this will require duplication of the flip-flops, but this holds for duplication-based checking as well and also increases the overhead there. But in contrast to the duplex system, here the synthesis tool may use sequential re-timing to reduce the number of flip-flops.

Circuit	HW overhead for FE			Max. achieved FE		
	0.7	0.8	0.9	FE	HW	Req. length
c432	82%	100%	127%	100%	183%	$1.98E10$
c499	—	—	—	34%	53%	$2.45E10$
c880	61%	73%	99%	100%	154%	$3.22E10$
c1355	—	—	—	23%	39%	$2.92E10$
c1908	22%	36%	54%	100%	94%	$1.73E10$
c2670	41%	66%	—	89%	90%	$3.63E10$
c3540	15%	21%	29%	100%	45%	$2.67E10$
c5315	32%	44%	64%	100%	104%	$3.63E10$
c6288	5%	6%	9%	100%	16%	$3.11E10$
c7552	25%	41%	—	88%	78%	$3.89E10$
s298	34%	47%	60%	100%	79%	$9.34E04$
s344	33%	47%	55%	100%	68%	$3.86E07$
s349	35%	44%	51%	100%	69%	$3.86E07$
s382	42%	47%	63%	100%	82%	$7.73E07$
s386	47%	56%	68%	100%	87%	$2.79E04$
s400	36%	47%	63%	100%	80%	$8.89E07$
s444	33%	43%	61%	100%	83%	$4.07E05$
s1196	37%	50%	66%	100%	96%	$6.28E06$
s1238	39%	49%	66%	100%	97%	$7.73E07$
s1423	41%	53%	67%	100%	89%	$3.17E10$
s1488	23%	30%	41%	100%	60%	$1.19E05$
s1494	24%	32%	41%	100%	57%	$1.10E05$
s5378	33%	45%	62%	100%	91%	$3.89E10$
s9234	27%	35%	46%	98.1%	60%	$4.18E10$
s13207	25%	33%	43%	100%	57%	$3.60E10$
s15850	26%	33%	44%	99.9%	61%	$4.58E10$
b01	32%	46%	60%	100%	90%	$4.39E02$
b02	52%	57%	73%	100%	91%	$2.15E02$
b03	49%	58%	71%	100%	93%	$1.24E09$
b04	41%	51%	68%	100%	103%	$2.81E10$
b05	22%	31%	41%	100%	66%	$1.98E10$
b06	43%	56%	73%	100%	97%	$4.32E02$
b07	38%	42%	52%	100%	77%	$2.57E10$
b08	53%	61%	71%	100%	95%	$2.39E07$
b09	41%	49%	62%	100%	95%	$1.55E08$
b10	52%	64%	80%	100%	110%	$2.78E06$
b11	31%	39%	49%	100%	70%	$5.69E09$
b12	46%	59%	79%	100%	103%	$3.38E10$
b13	50%	62%	74%	100%	99%	$2.75E10$
Average	37.4%	47.3%	61.3%		83.7%	$1.61E10$

TABLE I
HARDWARE OVERHEAD DEPENDING ON TARGETED FAULT EFFICIENCY WITH LIMITED TEST LENGTH ($\approx 3E10$ CYCLES)

For comparison, table II gives the concurrent test latency (CTL) and hardware overhead for duplication, the MICSET method and the method presented here for the ISCAS-85 benchmark circuits. MICSET [24] is a concurrent BIST architecture based on a deterministic test set. The duplicated logic consists of a copy of the combinational logic and one comparator per output. For duplication, the CTL is 0 by definition. In this experiment, the test-set based methods (MICSET and the method described here) achieve 100% fault efficiency for stuck-at faults. The MICSET values have been computed according to the information in [24].

Circuit	#PI	Dupl. HW overh.	MICSET CTL	MICSET HW overh.	Proposed Scheme CTL	Proposed Scheme HW overh.
c432	26	125%	$3.02E11$	1307%	$5.98E08$	183%
c499	41	110%	$9.94E12$	1236%	$1.05E13$	114%
c880	60	135%	$5.25E18$	1283%	$2.16E10$	154%
c1355	41	104%	$1.11E13$	947%	$1.20E13$	127%
c1908	33	116%	$4.58E10$	838%	$1.15E10$	94%
c2670	233	134%	$> 1.38E70$	N/A	$2.00E17$	108%
c3540	50	117%	$6.26E15$	695%	$1.18E10$	45%
c5315	178	105%	$> 3.83E53$	N/A	$3.10E10$	104%
c6288	32	110%	$1.69E10$	79%	$1.66E10$	16%
c7552	207	122%	$> 2.06E62$	N/A	$2.89E28$	86%
Average		118%		912%		103%

TABLE II
COMPARISON WITH DUPLICATION AND MICSET [24] WITHOUT COMPROMISING FAULT EFFICIENCY

In contrast to MICSET, the method presented here shows far better results with respect to hardware overhead and especially the concurrent test latency (CTL). The authors of [24] found that from the ISCAS-85 suite only a single circuit (c6288) was suitable for the MICSET approach in terms of hardware overhead and CTL. With the approach presented here, more than half of the circuits show acceptable CTL and require an overhead lower than duplication.

The area overhead for a duplex system is about 120%, since a comparator has to be added for each (pseudo)-primary output of the circuit. Here, the duplication is done using a simple copy of the original circuit. But to avoid common mode failures, the copy of the circuit should be implemented with diversity in mind, thus further increasing overhead [38]. For comparison, self-checking for random logic macros is far more costly and complex. In [38] it was shown, that for larger macros both parity prediction and state encoding have overhead often exceeding simple duplication.

The presented method features diversity by design. If 100% fault efficiency is targeted, the average hardware overhead is 103% for the ISCAS85 circuits. If all the circuits in table I are taken into account, it is 83.7% on average for this case. However, often reliability of the system can already be significantly improved by just accounting for a subset of faults (as low as 70% in [39]). For 90% fault efficiency, the hardware overhead is reduced to just 61.3% on the average. Ten out of the 39 circuits can even be tested for 90% of the faults with less than 50% overhead. And if the application allows to deal with even lower fault coverage, the hardware overhead can be further reduced.

VI. CONCLUSIONS

The presented method for concurrent BIST provides a test latency which is several orders of magnitude lower than existing methods.

This significantly reduces fault latency and subsequent fault accumulation. The large number of unspecified bits results in low hardware overhead. The evaluation has shown that the overhead is just 83.7% on average if complete fault coverage is required. An even lower overhead may be achieved by trading off fault coverage, for example to obtain 90% coverage just 61.3% overhead is required on average. The method is completely automated and generates concurrently self-testable circuits for a wide variety of random logic macros.

ACKNOWLEDGMENT

This work has been supported by the Deutsche Forschungsgesellschaft (DFG) under grants Wu245/3-3 and Wu245/5-2.

REFERENCES

[1] S. Borkar, T. Karnik, and V. De, "Design and reliability challenges in nanometer technologies," in *Proc. ACM/IEEE Design Automation Conference (DAC 2004)*, 2004, p. 75.

[2] S. R. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. IEEE Custom Integrated Circuits Conference*, 2001, pp. 223–228.

[3] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, no. 3, pp. 258–266, May-June 2005.

[4] P. Lala, *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann, 2001.

[5] I. Koren and C. Krishna, *Fault-Tolerant Systems*. Morgan Kaufmann, 2007.

[6] D. Pradhan, *Fault-Tolerant Computer Design*. Prentice Hall, 1996.

[7] M. Nicolaidis and Y. Zorian, "On-line testing for VLSI: A compendium of approaches," *Journal of Electronic Testing*, vol. 12, no. 1, pp. 7–20, 1998.

[8] C. Scherrer and A. Steininger, "Dealing with dormant faults in an embedded fault-tolerant computer system," *IEEE Transactions on Reliability*, vol. 52, no. 4, pp. 512–522, 2003.

[9] A. Steininger and C. Scherrer, "On the necessity of On-Line-BIST in safety-critical applications - a case study," in *Proc. Int'l Symposium on Fault-Tolerant Computing (FTCS'99)*, 1999, pp. 208–215.

[10] V. K. Agarwal and E. Cerny, "Store and generate built-in-testing approach," in *Proc. International Symposium on Fault-Tolerant Computing (FTCS'81)*, 1981, pp. 35–40.

[11] S. Hellebrand, S. Tarnick, B. Courtois, and J. Rajski, "Generation of vector patterns through reseeding of multipe-polynominal linear feedback shift registers," in *Proc. IEEE Int'l Test Conference (ITC)*, 1992, pp. 120–129.

[12] N. A. Touba and E. J. McCluskey, "Altering a pseudo-random bit sequence for scan-based BIST," in *Proc. IEEE International Test Conference (ITC)*, 1996, pp. 167–175.

[13] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ICCAD '96)*, 1996, pp. 337–343.

[14] S. Swaminathan and K. Chakrabarty, "A deterministic scan-BIST architecture with application to fieldtesting of high-availability systems," in *IEEE Conference on Custom Integrated Circuits*, 2001, pp. 259–262.

[15] L. Shombert and D. P. Siewiorek, "Using redundancy for concurrent testing and repairing of systolic arrays," in *International Symposium on Fault-Tolerant Computing (FTCS), Pittsburgh, PA*, 1987, pp. 244–249.

[16] M. Abramovici, C. E. Stroud, C. Hamilton *et al.*, "Using roving stars for on-line testing and diagnosis of fpgas in fault-tolerant applications," in *Proc. IEEE International Test Conference (ITC)*, 1999, pp. 973–982.

[17] H. Al-Asaad and P. Moore, "Non-concurrent on-line testing via scan chains," in *IEEE Systems Readiness Technology Conference*, 2006, pp. 683–689.

[18] H. Inoue, Y. Li, and S. Mitra, "VAST: Virtualization-assisted concurrent autonomous self-test," in *Proc. IEEE Int'l Test Conference (ITC)*, 2008, p. 12.3.

[19] K. Saluja, R. Sharma, and C. Kime, "Concurrent comparative testing using BIST resources," in *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1987, pp. 336–337.

[20] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 7, no. 12, pp. 1250–1260, 1988.

[21] R. Sharma and K. Saluja, "An implementation and analysis of a concurrent built-in self-test technique," in *18th International Symposium on Fault-Tolerant Computing (FTCS)*, 1988, pp. 164–169.

[22] P. Drineas and Y. Makris, "Concurrent fault detection in random combinational logic," in *4th International Symposium on Quality of Electronic Design (ISQED 2003), 24-26 March 2003, San Jose, CA, USA*, 2003, pp. 425–430.

[23] P. Drineas and Y. Makris, "SPaRe: selective partial replication for concurrent fault-detection in FSMs," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 6, pp. 1729–1737, 2003.

[24] I. Voyiatzis, A. Paschalis, D. Gizopoulos *et al.*, "An input vector monitoring concurrent BIST architecture based on a precomputed test set," *IEEE Transactions on Computers*, pp. 1012–1022, 2008.

[25] M. A. Schuette and J. P. Shen, "Processor control flow monitoring using signatured instruction streams," *IEEE Trans. Computers*, vol. 36, no. 3, pp. 264–276, 1987.

[26] R. Leveugle and G. Saucier, "Optimized synthesis of concurrently checked controllers," *IEEE Trans. Computers*, vol. 39, no. 4, pp. 419–425, 1990.

[27] M. Karkala, N. A. Touba, and H.-J. Wunderlich, "Special ATPG to correlate test patterns for low-overhead mixed-mode BIST," in *Proceedings of the 7th Asian Test Symposium (ATS '98)*, 1998, pp. 492–499.

[28] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern generation for a deterministic BIST scheme," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 1995, pp. 88–94.

[29] I. Pomeranz and S. Reddy, "Reducing the number of specified values per test vector by increasing the test set size," *IEE Proceedings on Computers and Digital Techniques*, vol. 153, no. 1, pp. 39–46, 2006.

[30] K. Miyase and S. Kajihara, "XID: Don't care identification of test patterns for combinational circuits." *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 23, no. 2, pp. 321–326, 2004.

[31] M. A. Kochte, C. G. Zoellin, M. E. Imhof, and H.-J. Wunderlich, "Test set stripping limiting the maximum number of specified bits," in *Proc. IEEE International Symposium on Electronic Design, Test and Applications (DELTA)*, 2008, pp. 581–586.

[32] V. Gherman, H.-J. Wunderlich, H. P. E. Vranken *et al.*, "Efficient pattern mapping for deterministic logic bist," in *Proceedings 2004 International Test Conference (ITC 2004), October 26-28, 2004, Charlotte, NC, USA*, 2004, pp. 48–56.

[33] I. Voyiatzis, A. M. Paschalis, D. Nikolos, and C. Halatsis, "R-CBIST: an effective RAM-based input vector monitoring concurrent BIST technique," in *Proc. IEEE Int'l Test Conference (ITC)*, 1998, pp. 918–925.

[34] A. El-Maleh and A. Al-Suwaiyan, "An efficient test relaxation technique for combinational & full-scan sequential circuits," in *Proceedings of the 20th IEEE VLSI Test Symposium*, 2002, pp. 53–59.

[35] R. Sharma and K. K. Saluja, "Theory, analysis and implementation of an on-line BIST technique," *VLSI Design*, vol. 1, no. 1, pp. 9–22, 1993.

[36] P. Bardell, W. McAnney, and J. Savir, *Built-in test for VLSI: pseudorandom techniques*. Wiley-Interscience New York, NY, USA, 1987.

[37] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. M. Reddy, "Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 14, no. 12, pp. 1496–1504, 1995.

[38] S. Mitra and E. J. McCluskey, "Which concurrent error detection scheme to choose?" in *Proc. IEEE Int'l Test Conference (ITC)*, 2000, pp. 985–994.

[39] K. Mohanram, E. S. Sogomonyan, M. Gössel, and N. A. Touba, "Synthesis of low-cost parity-based partially self-checking circuits," in *Proc. IEEE Int'l On-Line Testing Symposium (IOLTS)*, 2003, pp. 35–40.