ESTA: An Efficient Method for Reliability Enhancement of RT-Level Designs

Naghmeh Karimi, Shahrzad Mirkhani, and Zainalabedin Navabi
Electrical and Computer Engineering Department
Faculty of Engineering – Campus #2 – University of Tehran, 14399 Tehran, IRAN
{naghmeh, shahrzad}@cad.ece.ut.ac.ir; navabi@ece.neu.edu

Abstract

This paper proposes a novel and efficient method for RT Level online testing. Our method makes every RT-Level resource online-testable, and guarantees high single stuck-at fault detection (i.e., high reliability) with low area/latency overhead. This method uses available resources in their dead intervals (the intervals during which a resource is not being used) to test active resources. The area and/or latency overhead are due to concurrent operation of active and inactive resources. This method is evaluated by fault simulating several benchmark designs before and after applying the proposed algorithm. Experimental results show that after applying our method, online fault coverage is significantly improved.

1. Introduction

Testing the functionality of a circuit during its normal operation has become a necessity of today's complex circuits. Online and concurrent testing [1] are used for testing a circuit during its operation. Unlike offline testing methods, e.g. traditional BIST structures, online testing methods detect a fault in faulty circuits as soon as the fault occurs. This increases the reliability of the systems, and at the same time, eliminates the need for the system to be halted during the test process.

Online testing methods are grouped into different categories. In one category, a set of pre-computed test vectors and their expected circuit responses is stored using extra hardware. In these methods, during the normal operation of the circuit, when a circuit input matches any of the stored test vectors, the circuit output is observed and compared with the corresponding pre-computed output to determine if the circuit is faulty or not. These methods require excessive hardware overhead for storing a sufficient number of test vectors and expected responses [1].

Another approach uses pseudo-random vector generators (e.g., LFSR's) and signature analyzers (e.g., MISR's) to generate test vectors and compress the outputs. Since the sequence of applying test vectors is specified here, the time to apply all necessary test vectors (test latency) is unpredictable (often excessive). The other disadvantage of these methods is that they cannot be applied to sequential circuits [2].

In another category of online testing methods, the test process can be done by simply duplicating the computation of disjoint components and voting on the results. Although these methods are naturally fault-secure [3], they bear a large hardware overhead. This is because several parts of a circuit (sometimes the whole circuit) must be duplicated for test purposes [4]. However, due to the extensive hardware overhead in the resource duplication methods, algorithmic duplication methods have been proposed. In the algorithmic duplication methods, the operations (as opposed to resources) are duplicated. This results in less, but still considerable hardware overhead [4-5]. Three examples of these techniques have been presented in [5, 6, and 18].

Another online testing method deals with utilizing coarse behavioral invariance either inherent in the design [7-8] or imposed through error detection codes [9-10], in order to check the correctness of the design functionality. In this case, while the circuit computes $f(x)$ for input x, an additional function, $g(x)$, with a well-defined, simple-to-check relation to $f(x)$ is also computed. The operational health of the circuit is verified by checking that the relation between $f(x)$ and $g(x)$ holds. Coarse behavioral design invariance is inherently available in limited domains [11].

In our method, the reliability of the design is increased by utilizing the available resources in their dead intervals (the clock cycles during which they are inactive). By applying this method, the hardware overhead is much less than those required by the mentioned duplication and memory-based methods.

At the same time, since we use the normal inputs of the circuit as test vectors, test latency of the circuit is much lower than the online testing methods using pre-computed or pseudo random inputs. Another advantage of this method is that the testable circuit has 100% resource coverage, i.e., all resources of the circuit are tested exactly once per an input to output flow. This method can be applied to a wide variety of designs. It

15th Asian Test Symposium (ATS'06)
0-7695-2628-4/06 $20.00 © 2006 IEEE

can be applied to both sequential and combinational circuits, and both non-linear and linear circuits.

Section 2 discusses our proposed method (called ESTA). In this section, a brief description of ESTA is presented followed by the discussion of its algorithm. Section 3 shows the results of applying ESTA on several designs. The performance of ESTA is evaluated by performing fault simulation before and after applying this method on each test case. Since applying test vectors to a circuit during its normal operation follows the uniform distribution, we apply random test vectors to our test cases. We have gained an average of 76.5% *online* fault coverage in this method by applying random test vectors. Before applying this method, the online fault coverage of each circuit was 0%.

2. Efficient Self Testing Algorithm (ESTA)

This section presents our novel method for online testing. This method uses a small amount of test hardware overhead. It is a self test method, because the resulting circuit is tested during its normal operation. This circuit has 100% resource coverage, i.e., all resources are tested once when applying an input vector to the circuit. The goal of this method is to alter the original circuit controller such that the whole circuit becomes self-testable with a minimum area/latency overhead.

2.1. Overview

In ESTA the online testing hardware is inserted into the circuit during the RTL synthesis process, while several other methods insert it after synthesis process [2, 14-15]. These methods result in a large area/latency overhead, since in synthesis process, resources are scheduled and bound regardless of testability considerations.

Using our method results in a less area/latency overhead compared with methods that insert test circuit after the synthesis process. Utilizing each free resource for testing other resources in each clock cycle leads to less area overhead. In addition, testing resources concurrently with their normal operation leads to less test time (latency) and less required registers.

Advantages of our method are in its wide range of hardware structures it applies to, and its hardware requirement. Many proposed algorithms require excessive hardware overhead for storing their test vectors. On the other hand, the time of applying all of these test vectors to the CUT is unpredictable. Our method uses the normal input vectors of the CUT as their test vectors (if possible) in order to detect its existing faults. Therefore, test latency noticeably decreases compared with other methods. The proposed algorithm is described in the sections that follow.

2.2. Algorithm Description

In the synthesis process, the behavioral description of a circuit is converted to a Data Flow Graph (DFG). This graph is a directed graph with operations as vertices and data variables as arcs. The operations of this DFG are scheduled and then bound to available resources. Applying ESTA on the resulted scheduled and bound DFG makes this DFG online testable.

In this section, we discuss ESTA algorithm. We consider two constraints for constructing an online testable design: *T-area* and *T-delay*. T-area deals with the resources that are added to a DFG for self checking purposes, while T-delay deals with the added number of clock cycles to a DFG for self checking purposes.

2.2.1. T-area Constraint

First, consider the case in which T-area is our constraint. In each clock cycle, there can be free and busy resources of the same type. Let F_k and B_k be the number of free and busy resources of type k respectively. If in a clock cycle, F_k is not zero, we can test a free resource with a busy one or another free resource of the same type. Test process for these two RUTs (Resource Under Test) is performed by applying the same set of inputs to both resources and comparing their results with a simple equation operator. If the results are equal, there is a chance that both of these resources are fault free. Otherwise, one of these resources is faulty (***Scenario 1***).

If there is only one resource of kind k, a different procedure must be taken. In this case, since there are no other resources to test that unique resource, we have to add test hardware (e.g., LFSR/MISR) to test this resource individually. Test hardware is added if the corresponding cost is lower than resource duplication cost (***Scenario 2***).

To test a free resource with a busy one, the input data of the busy resource is applied to the free one. Otherwise (if both RUT's are free resources in that clock cycle) the controller must provide random data for both RUT's. Random data generation is achieved by either inserting a global random pattern generator or using other busy resources input data in the previous or current clock cycles. For using busy resources in the previous clock cycles, we have to use registers for saving data. The above scenario happens if in at least one clock, there are one or more free resources.

Scenario 1 happens when at least in one clock cycle, there are one or more free resources. If at all times all resources of the same type are busy (i.e., F_k is

always 0), then we expand the total latency by an extra clock cycle. Obviously in this clock cycle, all busy resources in the previous cycles are free and can be tested with each other (***Scenario 3***). Scenario 3 is also applied if after processing the whole DFG, there is at least one not-tested resource using the above scenarios.

2.2.2. T-delay Constraint

Considering T-Delay as the constraint is different from considering the T-area constraint only if there is more than one resource of type k and all these resources are busy during all clock cycles, or when after processing the whole DFG, there is at least one not-tested resource. Under this condition, we add $\lceil n \div m \rceil$ resources to the data path, where n is the number of busy resources of type k in each clock cycle and m is the latency of the given DFG. Adding these extra resources to the original DFG results in a DFG that can be tested using Scenario 1.

2.3. Resource Usage Graph

According to the above discussion, at each clock cycle we need to know the number of free and busy resources. Since a DFG represents only the busy resources during the circuit operation, we present another flow graph which can demonstrate the number of free resources as well as busy ones. We call this flow graph the *Resource Usage Graph* (RUG). RUG is a graph in which vertices represent the available resources and the edges represent the data transfer between these resources.

Consider the scheduled and bound DFG shown in Fig. 1. All operations of this circuit are performed during four clock cycles. Shown below are binding of operations of this DFG to adders (*A1* and *A2*), multipliers (*M1* and *M2*), and subtractor (*S1*):

- Operations $+_1$ to $+_4$ are bound to *A1*
- Operations $+_5$ to $+_8$ are bound to *A2*
- Operations $*_1$ to $*_4$ are bound to *M1*
- Operations $*_5$ to $*_7$ are bound to *M2*
- Operation $*_8$ is bound to *M3*
- Operation $-_1$ are bound to *S1*

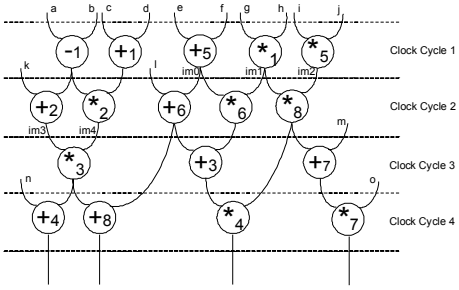

Figure 1. A simple DFG

Therefore, the datapath consists of two adders, three multipliers and one subtractor. Figure 2 shows the corresponding RUG of the DFG in Fig. 1.

Graph of Fig. 2 shows utilization of available resources versus time. Applying the method in Section 2.2 on the RUG of Fig. 2, various scenarios discussed regarding T-area and T-delay constraints will be done. In this RUG, adders *A1* and *A2* are used in clock cycles 1 to 4. Then we have to add one ($\lceil 2 \div 4 \rceil$) extra adder for testing *A1* and *A2*.

In this case if *A1* and the extra adder are tested in clock cycle i ($1 \le i \le 4$), *A2* and the extra adder can be tested in any clock cycle other than i (with Scenario 1). In Fig. 2, only one subtractor is available. Thus to test this resource, we have to add another subtractor to the list of available resources or test logic (e.g., LFSR and MISR) around *S1*, depending on their cost. In the former case (extra subtractor), *S1* can be tested with the extra subtractor using *S1* input data during clock cycle 1. They can also be tested with random input data in clock cycles 2, 3 or 4. However in the latter case (test logic) *S1* can be tested in cycles 2, 3 or 4.

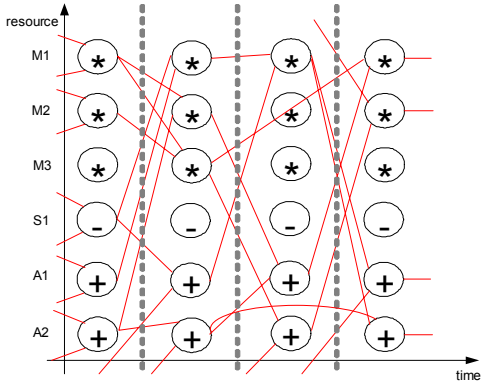

Figure 2. RUG constructed from the simple DFG

As for the multipliers of Fig. 2, in clock cycle 1, *M3* is free while *M1* and *M2* are busy. According to Scenario 1, one of *M1* or *M2* can be tested with *M3* in clock cycle 1 and the other one can be tested with *M3* in clock cycle 3 or 4. Note that in clock cycle 2 none of the multipliers can be tested since all of them are busy.

If we change the constraint from T-delay to T-area in the above example, testing the adders requires extension of the latency of the RUG from 4 to 5 clock cycles. In clock cycle 5 adders *A1* and *A2* can be tested using Scenario 1. Testing *S1*, *M1*, *M2* and *M3* are done the same as the T-delay constraint.

2.4. ESTA Pseudo Code

Figure 3 shows a pseudo code for our proposed method. The following definitions are used in this pseudo code.

- N_{cc}: Total clock cycles in the original RUG.

- N_{types}: Number of available resource types.
- F_k: Number of k-type free resources in the current clock cycle ($1 \le k \le N_{types}$).
- B_k: Number of k-type busy resources in the current clock cycle.
- N_k: Number of k-type resource instances. Note that N_k is constant during all clock cycles. ($N_k=F_k+B_k$ at each clock cycle)
- NTF_k: Number of k-type free resources that have not been tested yet.
- NTB_k: Number of k-type busy resources that have not been tested yet.
- $R_{k,j}$: The j^{th} k-type resource instance ($1 \le k \le N_{types}$ and $1 \le j \le N_k$).

Note that parameters F_k, B_k, NTF_k, and NTB_k are re-calculated at the beginning of each clock cycle.

In the simple RUG shown in Fig. 2, k is 1, 2, and 3 for the adder, multiplier, and subtractor, respectively. The above parameters are evaluated as follows.

- $N_{cc} = 4$.
- $N_{types} = 3$.
- F_k: in clock cycles 1 to 4
  - F_1: 0, 0, 0, 0
  - F_2: 1, 0, 2, 1
  - F_3: 0, 1, 1, 1
- B_k: in clock cycles 1 to 4
  - B_1: 2, 2, 2, 2
  - B_2: 2, 3, 1, 2
  - B_3: 1, 0, 0, 0
- N_k:
  - N_1: 2
  - N_2: 3
  - N_3: 1
- NTF_k: Is determined during the algorithm.
- NTB_k: Is determined during the algorithm.
- $R_{k,j}$:
  - Adder: R_{11}, R_{12}
  - Multiplier: R_{21}, R_{22}, R_{23}
  - Subtractor: R_{31}

This algorithm uses a RUG as input, processes the RUG from clock cycle 1 to N_{cc}, and results a modified RUG. In the output RUG all resource instances are testable. This algorithm treats each operation independently, and for an operation all its instances are simultaneously processed. Therefore, while processing one resource type, the other resource types are ignored.

As shown in Fig. 3, for each resource type, the clock cycles of the given RUG are analyzed consecutively until all instances of that resource type are made testable. When every instance of a resource is made testable, it is marked as tested (i.e., $R_{j,k}.tested$ = true). Depending on the status of the analyzed resources (busy or free in the specified clock cycle) two strategies can be applied:

- ***Strategy 1 - One busy with one free resource:*** This strategy is to make one busy (free) resource instance (clock cycle c) testable using another free (busy) resource instance (in clock cycle c).
- ***Strategy 2 - One free with another free resource:*** This strategy is to make one free resource instance testable using another free resource instance.

```
for k=1 to N_types loop
Lable1:cc := 1;
  while (true) loop
    Calculate F_k and B_k in clock cycle=cc;
    Calculate NTF_k from F_k; Calculate NTB_k from B_k;
    if (N_k>1) { --N_k=B_k+F_k
      if (NTB_k /= 0) {
        if (F_k /= 0) {
          Strategy1: test free-busy and mark both as tested;
           Recalculate F_k, B_k, NTF_k, and NTB_k; }}
      if (NTF_k /= 0) {
        if (B_k /= 0) {
          Strategy1: test busy-free and mark both as tested;
          Recalculate F_k, B_k, NTF_k, and NTB_k;
        else --B_k=0
          Strategy2: test free-free and mark both as tested;
          Recalculate F_k, and NTF_k; }}
    else --N_k = 1
      if (not R_k,1.tested) {
        if (Area(Test_Logic) > Area(R_k,1)) {
          N_k++;
          Goto Lable1;
        else
          Add Test_Logic around R_k,1;
          R_k,1.tested := true; }}}
    cc++; --go to the next clock cycle
    if ((cc > N_cc) OR (all R_k,j.tested are true))   break;
  end loop;
  if (not(all R_k,j.tested are true)) {
    --there are still not tested resources which were always busy
    if (constraint = T_Delay) {
      Add |NTB ÷ N_cc| resources of type k to N_k; Goto Lable1;

    else --constraint = T_Area
      N_cc++; --add one extra cycle
      Strategy2: Test free-free resources and check both as tested;
      Recalculate F_k, B_k, NTF_k, and NTB_k; }}
end loop;
```

Figure 3. The proposed pseudo code for ESTA

This algorithm is developed based on the following heuristics:

At each clock cycle if any resource instance can be made testable (using another available instance), it will be made testable, i.e., the process of making instances testable is done as soon as possible.

The priority of Strategy 1 is higher than Strategy 2. Also, the priority of making a busy instance testable using a free not-tested instance is higher than making a busy instance testable using a free tested instance (since in the former case, two resource instance are made testable simultaneously).

Based on the above heuristics and the two strategies discussed above, we will describe the details of each strategy. In Strategy 1 (testing a ***busy*** resource with a ***free*** resource), three cases can occur:

- $NTB_k = F_k$: Here, each not-tested busy instance can be paired with a corresponding free instance.

- $NTB_k > F_k$: Here, only a number of not-tested busy instances (equal to F_k) can be made testable. The rest of not tested busy instances (NTB_k-F_k) will be postponed to consequent clock cycles.
- $NTB_k < F_k$: Here, all not-tested busy instances can be paired with free (not-tested has priority) instances. If there are still free not-tested instances, according to the above heuristics, they can be paired with tested busy instances. In this case, even more than one free instance can be tested with a busy instance. If the number of tested busy instances is zero, the remaining not-tested free instances will be paired with each other (two or more instances can be tested with each other).

In Strategy 2 (testing a ***free*** resource with another ***free*** resource), the number of busy instances is zero. Thus, a free instance should be made testable using another free instance. We can have two scenarios here:

- NTF_k is even (= $2n$): Two not-tested instances are paired, i.e., we have n different pairs.
- NTF_k is odd (=$2n$+1): We make n testable pairs using $2n$ not-tested instances and one testable pair using one remaining not tested instance with one tested instance. We are sure that we have at least one tested instance, because according to the algorithm we have more than one instance here.

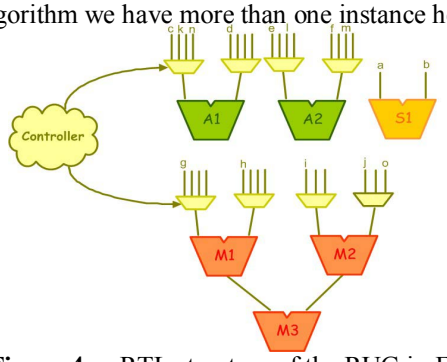

Figure 4-a. RTL structure of the RUG in Fig. 2

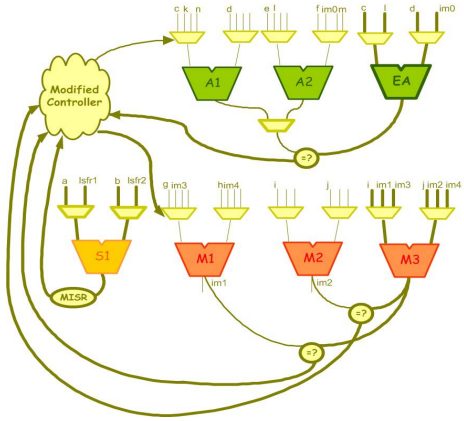

Figure 4-b. ESTA RTL structure of the circuit in Fig. 4-a

Strategies 1 and 2, discussed above, deal with Scenario 1 in the pseudo code shown in Fig. 3. The rest of this pseudo code relates to Scenarios 2 and 3 which were discussed in details in Section 2.2.

As we mentioned earlier in this section, the goal of our algorithm is to modify the circuit's controller to produce extra control signals in order to apply test data to resources during the circuit's normal operation and verify the correctness of the whole circuit. Figure 4 shows the corresponding RTL circuit before (4-a) and after (4-b) applying our proposed algorithm to the RUG in Fig. 2. Note that, for the sake of clarity, not all registers and routings are shown, and only the necessary parts of the circuit are depicted here.

3. Experimental Results

To evaluate our method, we have applied the proposed algorithm to several benchmarks. The selected benchmarks are a 6th order FIR filter, a 3rd order IIR filter, a Discrete Cosine Transform (DCT), Wavelet, Paulin, and differential equation circuits [13].

The selected RT-level benchmarks have been synthesized and then the corresponding gate-level circuits have been fault simulated using DSM-FS (a VHDL fault simulator) [12] with several (between 200 and 1000) random test vectors. Note that we have used *single stuck-at* fault model in our measurements.

We have inserted online testing hardware into each RT-level benchmark using ESTA (this is done in the pre-synthesized model). Note that considering T-area or T-delay constraints in our benchmarks has the same results due to the structures of these benchmarks.

After synthesizing the modified designs, we fault simulated the resulted gate-level circuit and then measured fault coverage considering the inserted output pin as the **only** primary output. The results are shown in Table 1. Note that in ESTA, we did not measure fault coverage for other primary outputs.

As it can be seen in Table 1, the fault coverage before applying ESTA (*Off-line fault coverage (before ESTA)* column in Table 1) is much more than the fault coverage after applying it (*On-line fault coverage (ESTA)* column in Table 1). The off-line fault coverage with all circuit outputs being observable is very different from on-line coverage with only the error flag being observable. The off-line coverage in this table is only for reference, and not to be compared with our on-line results. In off-line case a test instrument is required to test the design or an extra logic (BIST logic), which can only apply *pseudo-random* vectors, must be added to the design. But in our on-line case, without using any extra test instrument or without using pseudo-random test vectors, the circuit can be tested during its normal operation. Note that in the

normal circuit (before applying ESTA) no faults can be detected during the normal operation, i.e. the online fault coverage in this situation is 0%.

To have a better measure of performance of on-line testing, we have used *fault security parameter* [11]. This parameter is the ratio of the number of detected faults in ESTA to the number of detected faults in off-line method. We have used the same set of test vectors for both off-line and ESTA methods. As it can be seen in Table 1, this parameter is about 76% in our test cases. It is obvious that in each duplication-based on-line testing method, the faults on primary inputs of the circuit cannot be detected. However, the results in Table 1 include primary input faults.

Comparing our results with results of other on-line testing methods, we have less area and delay overhead than [4, 16]. As shown in Table 1 in some test cases due to register retiming the critical path delay has been decreased. ESTA fault coverage cannot be compared with several methods [4, 5, 11, 16, and 17] since they have not reported their resulted fault coverage or they use a set of test cases using only one instance for each resource type, which is not suitable for our method.

4. Conclusions

This paper presents an online testing method for error detection and reliability. This method uses free resources for adding testability to circuit under test. Its main advantage is low area overhead as compared with on-line test methods using resource duplication. This method guarantees testing every resource during a flow of data from circuit inputs to its outputs.

References

[1] M. Bushnell, V. Agrawal, "*Essentials of Electronic Testing for Digital Memory and Mixed Signal VLSI Circuits,*" Kluwer Academic Publishers, 2000.

[2] I. Voyiatzis, A. Paschalis, "*R-CBIST: An Effective RAM-Based Input Vector Monitoring Concurrent BIST Technique,*" in Proc. of ITC, 1998.

[3] M. Nicolaidis, Y. Zorian, "*On-line Testing for VLSI- A compendium of approaches,*" JETTA Journal, Vol. 12, No. 1-2, 1988, pp. 7-20.

[4] P. Oikonomakos, M. Zwolinski, "*Using High-Level Synthesis to Implement On-line Testability,*" IEEE Real-Time Embedded System Workshop, 2001.

[5] S. N. Hamilton, A. Orailoglu, "*On-Line Test for Fault-Secure Fault Identification,*" IEEE Trans. on VLSI Systems, Vol. 8, No. 4, August 2000.

[6] A. Orailoglu, R. Karri, "*Automatic Synthesis of Self-Recovering VLSI Systems,*" IEEE Trans. on Computers, Vol. 45, No. 2, February 1996.

[7] A. Chatterjee, R. K. Roy, "*Concurrent Error Detection in Nonlinear Digital Circuits with Applications to Adaptive Filters,*" in Proc. of International Conference on Computer Design, 1993, pp. 606-609.

[8] I. Bayraktaroglu, A. Orailoglu, "*Concurrent Test for Digital Linear Systems,*" IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 20, 2001, pp. 1775-1791.

[9] C. Stroud, et al., "*A parametrized VHDL Library for On-line Testing,*" in Proc. International Test Conference, 1997, pp.479-488.

[10] C. Zeng, N. Saxena, E. J. McCluskey, "*Finite State Machine Synthesis with Concurrent Error Detection,*" in Proc. ITC, 1999, pp. 672-679.

[11] Y. Makris, I. Bayraktaroglu, A. Orailoglu, "*Enhancing Reliability of RTL Controller-Datapath Circuits via Invariant-Based Concurrent Test,*" IEEE Trans. on Reliability, Vol. 53, No. 2, 2004.

[12] Z. Navabi, et al., "*Using RT Level Component Descriptions for Single Stuck-at Hierarchical Fault Simulation,*" JETTA Journal, Vol. 20, 2004.

[13] M. T. Lee, *High-Level Test Synthesis of Digital VLSI Circuits*, Artech House, 1997.

[14] I. Voyiatzis, et al., "*An Efficient Comparative Concurrent Built-in Self Test Technique,*" in Proc. ATS, 1995.

[15] I. Voyiatzis, et al., "*A Concurrent Built-In Self Test Architecture Based on a Self-Testing RAM,*" IEEE Trans. on Reliability, Vol. 54, No. 1, March 2005.

[16] P. Oikonomakos, et al., "*Versatile High-level Synthesis of Self-checking Datapaths Using an On-line Testability Metric,*" in Proc. DATE Conference, 2003.

[17] R. Karri, B. Iyer, "*Introspection: A Register Transfer Level Technique for Concurrent Error Detection and Diagnosis in Data Dominated Designs,*" ACM Trans. on Design Automation of Electronic Systems, Vol. 6, No. 4, 2001, pp. 501-515.

[18] C. Bolchini, et al., "*Concurrent Error Detection at Architectural Level,*" Proc. intl symposium on System synthesis, 1998, pp. 72-75.

Table 1. Applying ESTA on several benchmarks.

Circuit Name	Area before/after ESTA	Area overhead	Delay before/after ESTA	Off-line Fault Coverage (Before ESTA)	On-line Fault Coverage (ESTA)	**Fault security parameter**
6th order FIR filter	378/453	19.8%	12.17/11.8*	92%	62%	**77.62%**
3rd order IIR filter	406/475	16.9%	14.17/15.09	95%	61%	**73.13%**
DCT	588/701	19.2%	35.24/18.83*	85%	75%	**80.5%**
Wavelet	700/752	7.4%	34.78/38.84	91%	70%	**81.67%**
Paulin	462/568	22.9%	32.53/35.89	80%	56%	**70%**
Diff. Eq.	461/565	22.5%	32.50/36.06	81%	51%	**76%**

** In these cases, the delay has been decreased after applying ESTA that is because of register re-timing.*