ON EFFICIENT EXTRACTION OF PARTIALLY SPECIFIED TEST SETS FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

Aiman El-Maleh and Khaled Al-Utaibi

King Fhad University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
emails:{aimane, alutaibi}@ccse.kfupm.edu.sa

ABSTRACT

Testing systems-on-a-chip (SOC) involves applying huge amounts of test data, which is stored in the tester memory and then transferred to the circuit under test (CUT) during test application. Therefore, practical techniques, such as test compression and compaction, are required to reduce the amount of test data in order to reduce both the total testing time and the memory requirements for the tester. Relaxing test sequences, i.e. extracting partially specified test sequences, can improve the efficiency of both test compression and test compaction. In this paper, we propose an efficient test relaxation technique for synchronous sequential circuits that maximizes the number of unspecified bits while maintaining the same fault coverage as the original test set.

1. INTRODUCTION

Rapid advancement in VLSI technology has lead to a new paradigm in designing integrated circuits where a system-on-a-chip (SOC) is constructed based on pre-designed and pre-verified cores such as CPUs, digital signal processors, and RAMs. Testing these cores requires a large amount of test data which is continuously increasing with the rapid increase in the complexity of SOC. This has a direct impact on the total testing time and the memory requirements of the testing equipment. Hence, reducing the amount of test data is considered as one of the challenging problems in testing SOC.

Test compression and compaction techniques are widely used to reduce the storage and test time by reducing the size of the test data. Test compression techniques can achieve better results if the test set is composed of test cubes (i.e. if the test set is partially specified). In fact, some compression techniques such as, LFSR-reseeding [1, 2], require the test vectors to be partially specified. Even those techniques which require fully specified test data can benefit from unspecified bits in the test set. For example, variable-to-fixed-length coding [3] and variable-to-variable-length coding [4, 5] are known to perform better for long runs of 0's. Hence, assigning 0's to the don't care values in the test set will improve the efficiency of these techniques. Similarly, run-length coding techniques [6] can specify the don't care values in a way that will reduce test vector activity (i.e. the number of transitions from 0 to 1 and vice versa), which in tern improves the compression efficiency. On the other hand, the amount of compression that can be achieved with statistical coding techniques depends on the degree of variation in the occurrences of unique test patterns (i.e. code words). If all test patterns occur with equal frequency, then no compression is achieved at all. Thus, using partially specified test vectors adds more flexibility to statistical coding techniques in a sense that test patterns containing don't care values can be encoded with various possibilities.

Test compaction techniques can also benefit from a partially specified test. For example, when merging two test sequences using the overlapping compaction techniques described in [7], a don't care value, 'X', can be merged with any one of the values: '0', '1', and 'X'. Therefore, increasing the number of X's in a test set will reduce the number of conflicts that may occur while merging two test sequences, and hence, improves the efficiency of the compaction process.

2. PROBLEM DEFINITION

The problem of test relaxation, i.e. extracting a partially specified test set from a fully-specified one, has not been solved effectively in the literature. This problem, which is targeted in this paper, can be defined as follows. *Given a synchronous sequential circuit and a fully specified test set, generate a partially specified test set that maintains the same fault coverage as the fully specified one while maximizing the number of unspecified bits*. One obvious way to solve this problem is to use a bitwise-relaxation technique, where we test for every bit in the test set whether changing it to an 'X' reduces the fault coverage or not. Obviously, this technique is $O(nm)$ fault simulation runs, where n is the width of one test vector, and m is the number of test vectors. Obviously, this technique is impractical for large circuits. A partially specified test set can also be obtained using dynamic ATPG compaction. In dynamic compaction, every test vector is processed immediately after its generation in order to specify unspecified primary inputs (PIs). This feature can be disabled to obtain a compact and relaxed test set. However, this technique does not solve the problem of relaxing an already existing test set. In addition, this technique cannot benefit from random test pattern generation, because it is fault oriented.

Recently, two test relaxation techniques for combinational and full-scan sequential circuits were proposed in [8, 9]. The main idea of both techniques is to determine logic values in the fully-specified test set that are necessary to cover (i.e. detect) all faults which are detectable by this test set. Unnecessary logic values are set to X's.

As far as synchronous sequential circuits are concerned, the only existing solution to the problem of relaxing a given test set is the bitwise-relaxation method. In this paper, we propose an effi-

The authors would like to thank King Fhad University of Petroleum and Minerals for support.

cient test-relaxation technique that extends the technique proposed in [9] to cover synchronous sequential circuits.

3. PROPOSED TECHNIQUE

The general behavior of the proposed test-relaxation technique can be described as follows. At any time frame i, all logic values which are necessary to excite a newly detected fault and propagate it to some primary output p are marked as required. Next, these logic values are justified backwards starting from p towards primary inputs and/or memory-elements. At the end, unmarked primary inputs are not required and can be relaxed. On the other hand, required values on the memory-elements are justified when the next time frame, $i-1$, is processed. Note that justifying the detected faults based on logical values alone may result in masking some of the detected faults. Therefore, the proposed technique uses some rules based on fault-reachability analysis to avoid fault masking.

Due to the nature of sequential circuits (i.e. feedback from memory-elements), a fault excited in one time frame might propagate through several time farms before it gets detected. Hence, several time frames may need to be traced back to justify such faults. Therefore, we need to store enough information about fault propagation, detection and justification in order to perform the justification process frame by frame. Four lists are used to store the the required information: *POJustificationList*, *FFJustificationList*, *FaultPropagationList*, and *EventList*. The purpose of each one of these four lists is explained below.

The purpose of the *POJustificationList* is to store newly detected faults in every time frame. These faults will be justified backwards starting from the time frames where they first get detected. During fault simulation, if a fault f propagates to one or more memory-elements, then these memory-elements and their faulty values are added to the *FaultPropagationList*. The *FFJustificationList* is used to store faults that can't be completely justified during a certain time frame. Notice that if one or more memory-elements are required to justify a fault f during some time frame i, then f can't be completely justified during this time frame. Hence, the justification of f will continue during time frame $i-1$. The *EventList* keeps track of the gates that need to be justified for a certain fault. Gates are inserted in event list according to their levels in the circuit.

Figure 1 shows an outline of the proposed justification technique which consists of three phases. The first phase initializes the four lists. Fault simulation is performed in the second phase to identify newly detected faults. These faults are stored in *POJustificationList*[i] for every test vector i. During fault simulation, if a fault f propagates to one or more memory-elements, then these memory-elements together with their faulty values are added to *FaultPropagationList*[f]. The information in this list will be used to mark reachable lines of the circuit during the justification phase. It is important to point out here that we need to store the logical values of the memory-elements for all the time frames. This will enable the third phase to perform logic simulation in a certain time frame independent of the other time frames.

The third phase starts from the last time frame down to the first one. In every time frame, i, the algorithm performs the following. First, it logic simulates the circuit under the test vector i to determine the good value of every gate. Then, it checks *FFJustificationList*[i] for any fault that has not been completely justified in time frame $i+1$. Unjustified faults are removed from the list and justified one by one. Next, it checks *POJustificationList*[i] for newly detected faults and justifies them. Justifying a fault, f, involves two operations: *marking reachable lines* and *backward justification*, which are described below.

The first operation marks all the gates which are reachable from a given fault f using local fault simulation. It starts by injecting the fault f at its corresponding line in the circuit. Then, it sets the faulty values of the memory-elements according to the faulty values propagating from the time frame $i-1$. Next, the fault effects on the faulty-line and memory-elements are forward propagated. During this fault propagation, if the faulty value of a gate g is found to be different from its good value, then g is marked as reachable.

The second operation processes the event list level by level starting from the maximum level. In each level, the logical values of the stored gates are justified as follows. If g is a primary input (*PI*), then the logical value of g is required to detect the fault f. Therefore, the corresponding bit in the *RelaxedTestSet* is set to the logical value of g. If g is a memory-element (*DFF*), then the logical value of g can not be justified in the current time frame. Therefore, the fault f is added to the justification list of time frame $i-1$ (*FFJustificationList*[$i-1$]). If g is an XOR, XNOR, or a single-input gate, then all its inputs need to be justified. Hence, all the inputs of g are added to the event list according to their levels in the circuit. If g is an AND, OR, NAND or NOR gate with a non-controlling value, then we need to justify all the inputs of g. However, if g has a controlling value, then we need to check if it has an unreachable input with a controlling value. If it has, then it is sufficient to justify that input. Otherwise, we check whether g is reachable or not. If it is not reachable, then we need to justify only the reachable inputs of g. Otherwise, all the inputs of g need to be justified.

4. SELECTION CRITERIA

When justifying a controlling value through the inputs of a given gate, there could be more than one choice. In this case, priority is given to the input that is already selected to justify other gates. Otherwise, cost functions are used to guide the selection. Cost functions give a relative measure on the number of primary inputs required to justify a given value. Hence, they can guide the relaxation procedure to justify the required values with the smallest number of assignments on the primary inputs.

The cost functions proposed in [9] combine the *regular* recursive controllability cost functions [10] with new cost functions called *fanout-based* cost functions. The regular cost functions are computed as follows. For every gate g, we compute two cost functions $C_{reg0}(g)$ and $C_{reg1}(g)$. For example, if g is an AND gate with i inputs, then the cost functions are computed as:

$$C_{reg0}(g) = \min_i C_{reg0}(i)$$

$$C_{reg1}(g) = \sum_i C_{reg1}(i)$$

These costs functions are computed for other gates in a similar manner. The fanout-based cost functions can be computed for an AND gate as follows. Let g be an AND gate with i inputs. Let $F(g)$ denotes the number of fanout branches of g. Then, the fanout-based cost functions are computed as:

$$C_{fan0}(g) = \frac{\min_i C_{fan0}(i)}{F(g)}$$

```
(*Initialization phase*)
for every fault, f, in the fault list of the given circuit do
    Let FaultPropagationList[f] ← φ
for every test vector i do
    Let POJustificationList[i] ← φ
    Let FFJustificationList[i] ← φ
    for every level, l, of the given circuit do
        Let EventList[l] ← φ

(*Fault simulation phase*)
for i ← 1 to n do
    Fault simulate the circuit under test vector i
    for every fault, f, newly detected in i do
        Add f to POJustificationList[i]
    for every fault f propagating to the next time frame do
        Add all memory-elements affected by f together with
        their faulty values to FaultPropagationList[f]

(*Fault justification phase*)
for i ← n downto 1 do
    Logic simulate the circuit under the test vector i
    while FFJustificationList[i] ≠ φ do
        Remove f from FFJustificationList[i]
        Mark lines reachable from f
        for every memory-element, d, whose value is re-
        quired to justify f in time frame i + 1 do
            Let j be the input of d
            Let l be the level of j in the given circuit
            Add j to the EventList[l]
        Justify the fault f
    while POJustificationList[i] ≠ φ do
        Remove f from POJustificationList[i]
        Mark lines reachable from f
        Let j be a primary-output at which f get detected
        Let l be the level of j in the given circuit
        Add j to the EventList[l]
        Justify the fault f
```

Figure 1: Proposed Algorithm

$$C_{fan1}(g) = \frac{\sum_i C_{fan1}(i)}{F(g)}$$

The regular cost functions are accurate for fanout-free circuits. However, when fanouts exist, regular cost functions do not take advantage of the fact that a stem can justify several required values. In general, the fanout-based cost functions provide better selection criterion than the regular cost functions. However, there are some cases where the regular cost functions can perform better than the fanout-based cost functions. To take advantage of both cost functions, a weighted sum cost function of the two cost functions was proposed in [9]. The combined cost functions are defined below, where A and B represent the weights given to the regular and fanout-based cost functions respectively:

$$C_0(g) = A \cdot C_{reg0}(g) + B \cdot C_{fan0}(g)$$
$$C_1(g) = A \cdot C_{reg1}(g) + B \cdot C_{fan1}(g)$$

In synchronous sequential circuits, the controllability values of the circuit in one time frame depend on the controllability values computed in the current frame as well as the values computed in the previous frames. Therefore, the controllability values should be computed in an iterative manner starting from the first time frame. However, this may cause the regular cost function to grow much faster than the fanout-based cost function such that the effect of the second cost function in the weighted sum becomes negligible. Therefore, the regular cost function is adjusted to reduce the difference between the two cost functions [11].

5. EXPERIMENTAL RESULTS

In order to demonstrate the effectiveness of our proposed test relaxation technique, we have performed some experiments on a number of the ISCAS89 benchmark circuits. The experiments were run on a SUN Ultra60 (UltraSparc II 450MHz) with a RAM of 512MB. We have used test sets generated by HITEC[12]. In addition to that, we have used the fault simulator HOPE[13] for fault simulation purposes.

In Table 1, we compare the proposed test relaxation technique with the bitwise-relaxation method. The two techniques are compared in terms of the percentage of X's extracted, and the CPU time taken for relaxation. It is important to point out here that in order to have a fair comparison between our technique and the bitwise-relaxation method, we have constrained the bitwise-relaxation method such that all faults detected at a particular time frame remain detected in the same time frame after relaxation. However, the results obtained by both constrained and unconstrained bitwise-relaxation are shown in Table 1.

It is clear that, for all the circuits, the CPU time taken by our technique is less than that of the bitwise-relaxation method by several orders of magnitude. The bitwise-relaxation method requires enormous CPU times, and hence is impractical for large circuits.

The percentage of X's obtained by our technique is also close to the percentage of X's obtained by the bitwise-relaxation method for most of the circuits. The difference in the percentage of X's ranges between 0.5% and 16% (4% and 20% when compared with the unconstrained bitwise-relaxation method), while the average difference is about 5% (8% when compared with the unconstrained bitwise-relaxation method). It should be observed that the bitwise-relaxation method implicitly chooses the output for detecting a fault that maximizes the number of X's according to the order used. However, our technique does not do any optimization in selecting the best output for detecting a fault. This can be investigated in future work.

Table 2 shows the effect of varying the weights of the adjusted regular cost function and fanout-based cost function on the percentage of X's. As can be seen from the table, the use of cost functions results in higher percentage of X's. Also, it is worth mentioning here that neither the adjusted regular cost function nor the fanout-based cost function consistently performs better for all the circuits. However, when both cost functions are combined, better results are obtained. The table, also, shows that a weight of 1 for the adjusted regular cost function and a weight of 95 for the fanout-based cost function seems to be a good heuristic as it gives the highest percentage of X's on average.

6. CONCLUSION

In this paper, we have proposed an efficient test relaxation technique for synchronous sequential circuits. Comparison between our technique and the bitwise-relaxation method for a number of

Table 1: Test relaxation comparison between the proposed technique and the bitwise-relaxation method.

	Percentage of X's			CPU Time (seconds)	
Circuit	**Bitwise-Relaxation**	**Proposed Technique**	**Diff.**	**Bitwise-Relaxation**	**Proposed Technique**
s1423	69.922/74.392	54.314	15.61/20.08	943	1.300
s1488	76.154/81.090	71.902	4.252/9.188	12553	4.500
s1494	76.295/82.962	72.460	3.835/10.50	13146	3.550
s3271	83.894/85.527	77.265	6.629/8.262	87726	8.500
s3330	87.738/90.082	85.337	2.401/4.745	115585	6.783
s3384	78.579/81.655	77.943	0.636/3.712	16549	2.783
s4863	84.832/87.542	81.680	3.152/5.862	162894	8.267
s5378	87.738/88.969	85.815	1.923/3.154	218137	20.45
AVG	80.644/84.027	75.850	4.805/8.188		

Table 2: Cost function effect on the extracted percentage of X's.

CKT NAME	**A=0 B=0**	**A=1 B=0**	**A=0 B=1**	**A=1 B=15**	**A=1 B=35**	**A=1 B=55**	**A=1 B=75**	**A=1 B=95**
s1423	33.922	45.725	48.314	53.686	54.314	**54.431**	**54.431**	54.314
s1488	43.355	71.041	56.346	66.816	69.744	71.004	71.293	**71.902**
s1494	44.588	72.390	57.229	67.470	70.452	71.536	71.888	**72.460**
s3271	41.640	71.824	**80.872**	77.558	77.536	77.476	77.373	77.265
s3330	68.841	85.069	84.689	85.208	85.307	**85.385**	85.329	85.337
s3384	70.027	71.862	77.943	**77.972**	**77.972**	77.943	77.943	77.943
s4863	72.173	79.056	**83.425**	82.712	82.570	82.232	82.074	81.680
s5378	77.773	85.388	81.983	84.486	84.837	84.856	84.912	**85.815**
AVG	56.540	72.794	71.350	74.489	75.342	75.608	75.655	**75.850**

ISCAS89 benchmarks showed that our technique is faster by several orders of magnitude. The percentage of X's obtained by our technique is close to the percentage of X's obtained by the bitwise-relaxation method. The difference is about 5% on average.

Having a relaxed test set increases the effectiveness of both compression and compaction techniques. Also, the proposed technique can be used for extracting self-synchronizing test sequences. This will be investigated in future work.

7. REFERENCES

[1] B. Koenemann, "LFSR-Coded Test Patterns for Scan Designs", in *Proc. European Test Conference*, 1991, pp. 237–242.

[2] S. Hellebrand, S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Feedback Shift Registers", in *IEEE International Test Conference*, Sep. 1992, pp. 120–129.

[3] A. Jas and N. Touba, "Test Vector Decompression via Cyclical Scan Chains and Its Application to Testing Core-Based Designs", in *Proc. International Test Conference*, 1998, pp. 458–464.

[4] A. Chandra and K. Chakrabarty, "Test Data Compression for System-On-a-Chip using Golomb Codes", in *Proc. of IEEE VLSI Test Symposium*, 2000, pp. 113–120.

[5] A. Chandra and K. Chakrabarty, "Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression", in *19th IEEE Proceedings on. VTS*, 2001, pp. 42–47.

[6] T. Yamaguchi, M. Tilgner, M. Ishida and D. S. Ha, "An Efficient Method for Compressing Test Data", in *Proc. International Test Conference*, Nov. 1997, pp. 79–88.

[7] R. Roy, T. Niermann, J. Patel, J. Abraham, and R. Saleh, "Compaction of ATPG-Generated Test Sequences for Sequential Circuits", Nov. 1988, pp. 382–385.

[8] S. Kajihara and K. Miyase, "On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits", in *Proc. IEEE ICCAD*, Nov. 2001, pp. 364–369.

[9] A. El-Maleh and A. Al-Suwaiyan, "An Efficient Test Relaxation Technique for Combinational & Full-Scan Sequential Circuits", in *Proc. IEEE VLSI Test Symposium*, 2002, pp. 53–59.

[10] M. Abramovici, M. Breuer and A. Friedman, *Digital System Testing and Testable Design*, IEEE Press, 1990.

[11] K. Al-Utaibi, *An Efficient Test-Pattern Relaxation Technique for Synchronous Sequential Circuits*, M.S. thesis, King Fhad University of Petroleum and Minerals, Dhahran, 2002.

[12] Thomas M. Niermann and Janak H. Patel, "HITEC: A test generation package for sequential circuits", in *Proc. of the European Conference on Design Automation (EDAC)*, 1991, pp. 214–218.

[13] H. K. Lee and D. S. Ha, "HOPE: An Effecient Parallel Fault Simulator for Synchronous Sequential Circuits", *IEEE Trans. on Computer Aided Design*, vol. 15, no. 9, pp. 1048–1058, Sep. 1996.