# On Test Vector Reordering for Combinational Circuits

**Aiman El-Maleh** and Yahya Osais

The 16th International Conference on Microelectronics, pp. 772 – 775, 6-8 Dec. 2004.

# ON TEST VECTOR REORDERING FOR COMBINATIONAL CIRCUITS

*Aiman H. El-Maleh and Yahya E. Osais*

King Fahd University of Petroleum & Minerals, Computer Engineering Department,
Dhahran 31261, Saudi Arabia
{aimane,yosais}@ccse.kfupm.edu.sa

## ABSTRACT

The cost of testing is a major factor in the cost of digital system design. In order to reduce the test application time, it is required to order the test vectors in such a way that reduces the time a defective chip spends on a tester until the defect is detected. In this paper, we propose an efficient test vector reordering technique that significantly reduces both the time and memory complexities of reordering procedures based on fault simulation without dropping. Experimental results demonstrate both the efficiency and effectiveness of our proposed technique.

## 1. INTRODUCTION

In Automatic Test Pattern Generation (ATPG), the goal is to efficiently generate a complete test set for a given circuit. The test set size is directly proportional to the cost and time of test application. Hence, it is clear that a compact test set is highly desirable. In addition to that, the order of test vectors in a test set should be such that the fault coverage would show the maximal increase with every additional test vector. This helps in reducing the time a defective chip spends on a tester until the defect is detected, as the defect may be detected earlier.

Given a test set $T$, Algorithm 1 can be used to arrange the test vectors in a near optimal order. The test set $T'$ has the important property that every test vector achieves the maximum possible fault coverage of all test vectors that could be applied in its place. However, it requires the time consuming process of non-fault dropping simulation. The memory requirement is also very high since all the faults detected by every test vector have to be recorded.

Recently, a reordering procedure was proposed in [1]. It is based on double detection fault simulation. Basically, a fault is dropped once it is detected twice. Therefore, compared with Algorithm 1, the fault simulation effort is reduced dramatically. The proposed procedure, hereafter referred to as DD, can be applied iteratively to improve the quality of the final test set.

A major drawback in DD is that it does not consider all faults detected by a test vector. This is because a fault is dropped from the fault list when it is detected by two test vectors. This has an impact on the quality of the final test set, especially if the average number of test vectors detecting a fault is much greater than two.

In this paper, we propose solutions to the time and memory problems in test vector reordering procedures. Besides, we present a heuristic to control test vector selection.

**Algorithm 1** Order($T$)

```
1. Using non-fault dropping fault simulation, the set of faults
detected by each test vector is recorded.
2. Repeat the following steps until no test vector is left in $T$:
    2.1 Select a test vector $t_i$ from $T$ such that $t_i$ detects the
    largest number of faults.
    2.2 Remove $t_i$ from $T$ and append $t_i$ to the end of the ordered
    test set $T'$, which is initially empty.
    2.3 For each test vector $t_j$ in $T$, remove the faults detected
    by $t_i$ from the set of faults detected by $t_j$.
3. Return $T'$.
```

This paper is structured as follows. First, our proposed solutions are presented. Then, experimental results are discussed. Finally, conclusions are given.

## 2. OUR PROPOSED SOLUTIONS

In order to overcome the time problem in test vector reordering procedures, we propose the use of the CRItical Path Tracing (CRIPT) algorithm as a fault simulator. CRIPT was proposed in [2] as a fault simulator for combinational circuits. Compared with conventional fault simulation, CRIPT has the following features [3]:

1. It directly identifies the faults detected by a test vector. Hence, the work involved in propagating faults not detected by a test vector towards primary outputs is avoided,
2. It deals with faults implicitly. Hence, fault enumeration, fault collapsing, fault partitioning, fault insertion, and fault dropping are not needed,
3. It does not require computing values in the faulty circuits by gate evaluations or fault list processing, and
4. It is an approximate method. However, the impact of approximation is negligible.

From the above features, it can be concluded that CRIPT is faster and requires less memory than conventional fault simulation. Experimental results presented in [2] show that CRIPT is faster than concurrent fault simulation. Moreover, it has the important feature that the cost of fault simulation without dropping is the same as that of fault simulation with dropping.

CRIPT does not solve the memory problem although it considers less faults. Algorithm 2, hereafter referred to as CPT+, shows our proposal for handling the memory problem. The algorithm

**Algorithm 2** CPT+($T$)

```
1. For $i$ = 1 to $n$:
    1.1. Run CRIPT. Do not store faults. Store only the number
         of faults detected by every test vector.
    1.2. Select the best $m$ test vectors.
    1.3. Drop all faults detected by the selected test vectors.
2. Run Algorithm 1. Use CRIPT as a fault simulator
```

**Table 1**. Example test set that show the effect of test vector selection.

| TV | Faults |
|---|---|
| $t_1$ | $f_1$,$f_2$,$f_3$,$f_4$ |
| $t_2$ | $f_1$,$f_2$,$f_5$,$f_6$ |
| $t_3$ | $f_5$,$f_7$,$f_8$ |
| $t_4$ | $f_9$,$f_{10}$ |

proceeds as follows. First, CRIPT is run for a number of iterations, which is equal to $n$. In every iteration, only the number of faults detected by every test vector is recorded. Besides, $m$ test vectors are selected such that they detect the maximum number of faults. Faults detected by the selected test vectors are dropped. Second, Algorithm 1 is run. In this algorithm, CRIPT is used as a fault simulator.

The selection step in Algorithm 1 is random when there is more than one test vector with the same number of faults. This has an impact on the quality of the final test set. For example, consider the set of test vectors shown in Table 1. Test vectors $t_1$ and $t_2$ have the same number of faults but different effects on the quality of the final test set. If we first choose $t_2$, then every remaining test vector detects only two faults. Therefore, after selecting the second test vector, the total number of faults that can be detected with two test vectors is six. On the other hand, if we first choose $t_1$, the total number of faults that can be detected with two test vectors is seven.

In order to account for the above phenomenon, we associate with every test vector a cost. Whenever there is more than one test vector with the same number of faults, the test vector with the minimum cost is selected first. The cost of a test vector $t_i$ is defined as follows.

$$Cost(t_i) = \sum_{j=1}^{NumF} NumTV(f_j),$$

where $NumF$ is the number of faults detected by $t_i$ and $NumTV(f_j)$ is the number of test vectors that detect fault $f_j$. In the above example, the costs of $t_1$ and $t_2$ are six and seven, respectively. Therefore, by first selecting $t_1$, only $t_2$ is affected. The new number of faults detected by $t_2$ is 2.

## 3. EXPERIMENTAL ANALYSIS & RESULTS

In order to demonstrate the effectiveness of our proposed technique, we have performed experiments on a number of the largest full-scanned versions of ISCAS89 benchmark circuits. The experiments were run on a SUN Ultra60 (UltraSparc II-450 MHz) with a RAM of 512 MB. We have used test sets generated by HITEC [4]. In addition, we have used the implementation of CRIPT provided in [5]. It should be pointed out that the implementation of CRIPT is not very efficient. This is because it is based on serial fault simulation of stems. However, the performance of CRIPT is not our main concern since it is experimentally guaranteed that CRIPT is faster than concurrent fault simulation.

**Table 2**. Runtimes spent by reordering procedures.

| Cct | # TVs | FC | Time (sec.) | | | |
|---|---|---|---|---|---|---|
| | | | Greedy | DD | CPT | CPT+ |
| s13207.1f | 633 | 98.462 | 11.97 | 14.04 | 37.04 | 74 |
| s15850.1f | 657 | 96.674 | 28 | 28 | 166 | 204 |
| s38417f | 1472 | 99.436 | 807 | 183 | 1800 | 2569 |
| s38584f | 1174 | 95.852 | 702 | 124 | 1410 | 2188 |
| s5378f | 359 | 99.131 | 1.97 | 1.12 | 6.01 | 15 |
| s9234.1f | 620 | 93.475 | 8 | 11 | 17 | 38 |

In Table 2, we report the runtimes spent by reordering procedures. The first, second, and third columns indicate the circuit name, test set size, and fault coverage, respectively. The runtime of the procedure Greedy is given under the column headed $Greedy$. This procedure is similar to Algorithm 1. In this procedure, fault simulation is performed using HOPE [6]. The runtimes of procedures DD, CPT, and CPT+ are given under the columns headed $DD$, $CPT$, and $CPT+$, respectively. Procedure CPT is similar to Algorithm 1, except that fault simulation is performed using CRIPT. Furthermore, test vector selection heuristic is employed to make smart selections. Procedure DD is run for four iterations. For procedure CPT+, $n$ = 4 and $m$ = 4.

The huge amounts of time required by the procedure Greedy for reordering the test sets of s38417f and s38584f show the effect of non-dropping fault simulation on the performance of test vector reordering procedures. It can be seen that the runtime of DD is significantly less than that of Greedy. This is due to double detection fault simulation. CPT+ requires more runtime than that required by CPT. This is because in CPT+, CRIPT is run more than once.

Table 3 shows the memory required for every circuit under DD, CPT, and CPT+. The first and second columns in the table indicate the circuit name and size of fault list, respectively. The third and fourth columns give the total amount of memory required by DD and CPT, respectively. The fifth column gives the amount of memory required by CPT+. Finally, the sixth and seventh columns give the percentage reductions in memory compared to DD and CPT, respectively.

As can be seen from Table 3, the memory required by CPT is much larger than that required by DD. Besides, the memory required by CPT+ is significantly much smaller than that required by CPT. It should be pointed out that the memory required by CPT+ is computed after selecting the $16^{th}$ test vector and dropping the faults detected by the selected test vectors. For example, the memory required to record the remaining faults in s5378f is 5594.

To evaluate the relative efficiency and effectiveness of the reordering procedures, we show in Tables 4 and 5 snapshots of the fault coverage obtained after applying the number of test vectors given under the column headed $\#TVs$. Compared with the number of test vectors in the original test sets required to achieve certain fault coverage, fewer test vectors are required after the original test sets are reordered by Greedy, DD, CPT, or CPT+. For example, for s38584f, to achieve 80% fault coverage, 90 test vectors are required using the original test set. On the other hand, 35 test vectors are required using the order of test vectors generated

**Table 4**. Efficiency of reordering procedures.

| | s13207.1f | | | | | s15850.1f | | | | | s38417f | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # TVs | Orig. | Greedy | DD | CPT | CPT+ | Orig. | Greedy | DD | CPT | CPT+ | Orig. | Greedy | DD | CPT | CPT+ |
| 1 | 20.89 | 25.02 | 23.87 | 25.02 | 25.02 | 19.55 | 23.22 | 21.48 | 23.22 | 23.22 | 20.34 | 22.89 | 21.6 | 22.89 | 22.89 |
| 5 | 45.86 | 56.93 | 54.08 | 56.93 | 56 | 48.88 | 56.41 | 53.69 | 56.41 | 56.08 | 52.65 | 56.15 | 54.26 | 56.15 | 51.86 |
| 10 | 55.99 | 66.63 | 64.88 | 66.63 | 66.29 | 55.27 | 67.49 | 64.66 | 67.49 | 66.37 | 60.94 | 67.61 | 64.92 | 67.61 | 60.9 |
| 15 | 58.83 | 71.26 | 70.11 | 71.26 | 71.02 | 59.44 | 72.97 | 70.24 | 72.97 | 71.42 | 65.21 | 73.23 | 70.47 | 73.23 | 69.71 |
| 20 | 60.44 | 74.38 | 73.32 | 74.41 | 74.24 | 61.54 | 76.35 | 73.86 | 76.35 | 75.2 | 67.53 | 76.71 | 74.22 | 76.71 | 74.57 |
| 25 | 61.68 | 76.74 | 75.66 | 76.83 | 76.87 | 62.85 | 78.71 | 76.48 | 78.7 | 78.01 | 69.59 | 79.24 | 76.78 | 79.24 | 77.62 |
| 30 | 62.29 | 78.63 | 77.55 | 78.66 | 78.89 | 63.44 | 80.61 | 78.33 | 80.61 | 80.15 | 70.7 | 81.08 | 78.96 | 81.08 | 79.76 |
| 35 | 63.55 | 80.24 | 79.28 | 80.24 | 80.5 | 64.58 | 82.1 | 80.06 | 82.14 | 81.72 | 71.73 | 82.55 | 80.54 | 82.56 | 81.5 |
| 40 | 64.17 | 81.58 | 80.7 | 81.57 | 81.75 | 66.06 | 83.28 | 81.54 | 83.38 | 83.07 | 72.68 | 83.72 | 81.87 | 83.75 | 82.88 |
| 45 | 65.03 | 82.62 | 81.86 | 82.64 | 82.76 | 66.56 | 84.26 | 82.78 | 84.38 | 84.15 | 73.68 | 84.75 | 83.08 | 84.76 | 84.02 |
| 50 | 65.19 | 83.56 | 82.95 | 83.49 | 83.69 | 68.34 | 85.14 | 83.94 | 85.28 | 85.05 | 74.48 | 85.58 | 84.1 | 85.57 | 84.96 |
| 60 | 65.82 | 85.19 | 84.73 | 85.2 | 85.22 | 69.08 | 86.55 | 85.69 | 86.7 | 86.48 | 75.11 | 86.91 | 85.74 | 86.9 | 86.48 |
| 70 | 66.4 | 86.52 | 86.14 | 86.52 | 86.51 | 71.82 | 87.71 | 87.02 | 87.85 | 87.62 | 76.18 | 87.91 | 86.86 | 87.87 | 87.58 |
| 80 | 66.98 | 87.62 | 87.33 | 87.62 | 87.58 | 73.74 | 88.68 | 88.09 | 88.77 | 88.58 | 76.6 | 88.75 | 87.84 | 88.69 | 88.49 |
| 90 | 67.42 | 88.59 | 88.32 | 88.6 | 88.56 | 75.1 | 89.43 | 88.94 | 89.48 | 89.34 | 77.88 | 89.44 | 88.61 | 89.37 | 89.27 |
| 100 | 68.02 | 89.44 | 89.23 | 89.44 | 89.39 | 76.15 | 90.09 | 89.64 | 90.09 | 90 | 78.69 | 90.03 | 89.28 | 89.95 | 89.9 |

**Table 5**. Efficiency of reordering procedures.

| | s38584f | | | | | s5378f | | | | | s9234.1f | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| #TVs | Orig. | Greedy | DD | CPT | CPT+ | Orig. | Greedy | DD | CPT | CPT+ | Orig. | Greedy | DD | CPT | CPT+ |
| 1 | 14.18 | 23.54 | 22.92 | 23.54 | 23.54 | 22.03 | 24.44 | 22.03 | 24.44 | 24.44 | 16.11 | 17.58 | 16.8 | 17.58 | 17.58 |
| 5 | 41.03 | 55.71 | 53.57 | 55.63 | 49.25 | 50.66 | 61.29 | 55.31 | 61.29 | 59.24 | 34.3 | 42.7 | 40.31 | 42.7 | 42.7 |
| 10 | 58.9 | 66.63 | 64.38 | 66.48 | 60.39 | 59.57 | 73.89 | 68.76 | 73.89 | 72.58 | 39.22 | 53.46 | 51.75 | 53.39 | 53.39 |
| 15 | 64.94 | 71.66 | 69.85 | 71.7 | 68.31 | 64.22 | 79.58 | 76.45 | 79.56 | 78.84 | 42.75 | 59.12 | 57.64 | 59.13 | 59.31 |
| 20 | 68.13 | 75.04 | 73.1 | 74.91 | 72.62 | 68.76 | 82.84 | 80.75 | 82.84 | 82.25 | 44.72 | 63.19 | 61.73 | 63.03 | 63.03 |
| 25 | 70.46 | 77.37 | 75.64 | 77.32 | 75.79 | 70.71 | 85.05 | 83.66 | 85.05 | 84.84 | 45.88 | 66.28 | 64.91 | 66.18 | 66.18 |
| 30 | 72.08 | 79.13 | 77.57 | 79.15 | 78.06 | 71.78 | 86.73 | 85.53 | 86.73 | 86.66 | 47.71 | 68.9 | 67.58 | 68.6 | 68.6 |
| 35 | 73.47 | 80.59 | 79.07 | 80.62 | 79.73 | 73.19 | 88.05 | 86.99 | 88.05 | 87.99 | 49.13 | 71.04 | 69.71 | 70.68 | 70.68 |
| 40 | 74.98 | 81.75 | 80.16 | 81.84 | 81.09 | 75.67 | 89.18 | 88.29 | 89.16 | 89.07 | 50.43 | 72.8 | 71.53 | 72.3 | 72.3 |
| 45 | 75.59 | 82.76 | 81.27 | 82.85 | 82.18 | 76.15 | 90.14 | 89.4 | 90.14 | 90.01 | 51.67 | 74.33 | 73.15 | 73.78 | 73.78 |
| 50 | 76.32 | 83.61 | 82.25 | 83.72 | 83.11 | 76.54 | 90.98 | 90.29 | 90.98 | 90.81 | 52.16 | 75.7 | 74.55 | 75.04 | 75.04 |
| 60 | 77.91 | 85.05 | 83.84 | 85.15 | 84.65 | 78.3 | 92.31 | 91.81 | 92.22 | 92.16 | 54.27 | 77.96 | 76.95 | 77.7 | 77.7 |
| 70 | 78.88 | 86.19 | 85.15 | 86.28 | 85.9 | 79.64 | 93.31 | 92.85 | 93.22 | 93.18 | 55.02 | 79.72 | 78.82 | 79.49 | 79.49 |
| 80 | 79.49 | 87.09 | 86.16 | 87.18 | 86.92 | 81.08 | 94.03 | 93.7 | 93.98 | 93.96 | 56.23 | 81.2 | 80.37 | 80.87 | 80.87 |
| 90 | 80.33 | 87.87 | 87.05 | 87.93 | 87.74 | 82.82 | 94.68 | 94.35 | 94.63 | 94.61 | 58.03 | 82.42 | 81.74 | 82.13 | 82.31 |
| 100 | 81.22 | 88.54 | 87.79 | 88.57 | 88.43 | 83.38 | 95.13 | 94.89 | 95.09 | 95.11 | 60.89 | 83.46 | 82.94 | 83.15 | 83.15 |

**Table 3**. Memory required for every circuit under DD, CPT, and CPT+.

| Cct | # Faults | Total Memory (bytes) | | | Memory Reduction | |
|---|---|---|---|---|---|---|
| | | DD | CPT | CPT+ | DD | CPT |
| s13207.1f | 9815 | 19630 | 1364941 | 17886 | 9 | 98.7 |
| s15850.1f | 11725 | 23450 | 1546227 | 29359 | 0 | 98 |
| s38417f | 31180 | 62360 | 9745991 | 287771 | 0 | 97 |
| s38584f | 36303 | 72606 | 9317334 | 321100 | 0 | 96.6 |
| s5378f | 4603 | 9206 | 353358 | 5594 | 39.2 | 98.4 |
| s9234.1f | 6927 | 13854 | 640525 | 16631 | 0 | 97.4 |

by Greedy and CPT. Using the order of test vectors generated by DD and CPT+, 40 test vectors are required to achieve 80% fault coverage.

It can be seen that for most of the circuits, the quality of test sets reordered by Greedy is better than that of test sets reordered by DD. It can also be seen that for most of the circuits, the quality of test sets reordered by CPT is better than that of test sets reordered by DD, especially with the first few test vectors. This is because DD considers only two test vectors per a fault. Thus, the larger the average number of test vectors that detect a fault is, the poorer the quality of test sets reordered by DD is expected to be.

The quality of test sets reordered by Greedy and CPT is generally the same. However, for some circuits, CPT generates slightly better test sets. This is due to the test vector selection heuristic employed in CPT.

In four circuits out of six, CPT+ is better than DD. For the remaining two circuits, after the first 16 test vectors, CPT+ is better than DD. For example, for s5378f, a difference of 4% is noticed after applying the first ten test vectors. Also, for s15850.1f, a difference of 2% is noticed after applying the first 20 test vectors.

Figures 1 and 2 show the fault coverage curves obtained by the original test sets and test sets generated by CPT+ and DD after one pass and four passes for the circuits s5378f and s38417f, respectively. We show the curves of two circuits only due to space limitation. Information on other circuits can be found in [7]. Clearly, for both circuits, the quality of test sets reordered by CPT+ is better than that reordered by DD.

## 4. CONCLUSIONS

In this paper, we have proposed an efficient test vector reordering technique based on critical path tracing (CRIPT). CRIPT has the advantage that it has the same cost for fault simulation without dropping and with dropping. Our proposed solution significantly reduces both the CPU time and memory requirements of test vector reordering procedures based on fault simulation without dropping.

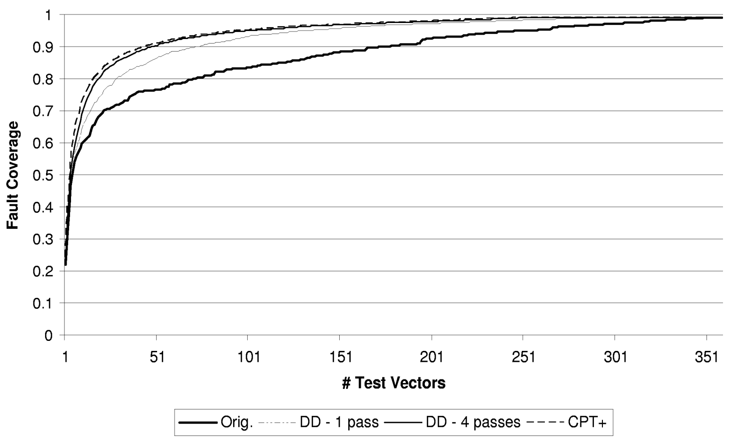

**Fig. 1**. Fault coverage curves for s5378f.

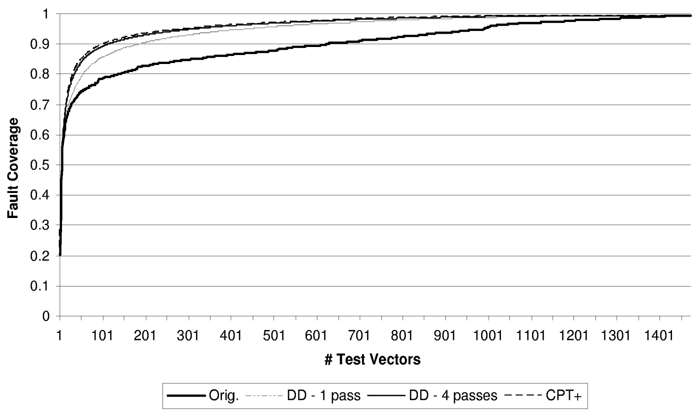

**Fig. 2**. Fault coverage curves for s38417f.

Based on experimental results, our proposed technique provides a better test vector ordering than the recently proposed ordering technique based on double-fault detection with competitive CPU time and memory requirements.

## 5. ACKNOWLEDGMENT

The authors would like to thank King Fahd University of Petroleum & Minerals for support.

## 6. REFERENCES

[1] Xijaing Lin, Janusz Rajski, Irith Pomeranz, and Sudhakar M. Reddy, “On Static Test Compaction and Test Pattern Ordering for Scan Designs,” in *Proc. of the Int’l Test Conference*, 2001, pp. 1088–1098.

[2] M. Abramovici, P. R. Menon, and D. T. Miller, “Critical Path Tracing – An Alternative to Fault Simulation,” *IEEE Design and Test*, pp. 83–92, Feb. 1984.

[3] Miron Abramovici, Melvin A. Bruer, and Arthur D. Friedman, *Digital Systems Testing and Testable Design*, 1990.

[4] T. M. Niermann and J. H. Patel, “HITEC: A Test Generation Package for Sequential Circuits,” in *Proc. of the European Conference on Design Automation*, Feb. 1991, pp. 214–218.

[5] Ali Saleh Al-Suwaiyan, “Efficient Test Relaxation Techniques for Combinational Circuits,” *MS Thesis, King Fahd University of Petroleum and Minerals*, Oct. 2002.

[6] Hyung Ki Lee and Dong Sam 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, Sept. 1996.

[7] Yahya E. Osais, “Efficient Static Compaction Algorithms for Combinational Circuits Based on Test Relaxation,” *MS Thesis, King Fahd University of Petroleum and Minerals*, Oct. 2003.