A PARALLEL SCHEME FOR TEST-PATTERN GENERATION

Akira Motohara, Kenji Nishimura, Hideo Fujiwara and Isao Shirakawa

Reprinted from the IEEE International
Conference on Computer-Aided Design
November 11-13, 1986 Santa Clara, California

IEEE COMPUTER SOCIETY REPRINT

IEEE COMPUTER SOCIETY
1730 Massachusetts Avenue, NW
Washington, D.C. 20036

1884 1984 THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY PRESS

A PARALLEL SCHEME FOR TEST-PATTERN GENERATION

Akira Motohara*, Kenji Nishimura*,
Hideo Fujiwara** and Isao Shirakawa***

* Matsushita Electric Industrial Co., Ltd. Osaka 570 Japan
** Meiji University, Kawasaki 214 Japan
*** Osaka University, Osaka 545 Japan

ABSTRACT

An approach to the parallel processing for generating test-patterns for combinational circuits is described. In general, difficulty in the test-pattern generation lies in the two points; how to deal with a large number of faults, and what to do with the fault that is hard to generate test-pattern and that leads to cause a great number of backtracks.

Our new test-pattern generation system, consisting of a test-pattern generation algorithm and a fault simulation, is capable of parallel processing and has resulted in the improvement of those difficulties.

In order to confirm its performance, we implemented the proposed method on a multi-microcomputer system and applied it to the combinational circuits with 1,000 gates or less. It is our conclusion that the proposed approach to the parallel processing is effective to obtain a good fault coverage within a short period of time as compared with the conventional method run on the general purpose von-Neumann computers.

1. INTRODUCTION

The growing logic complexity of VLSI circuits has made test-pattern generation more and more difficult. To improve this problem, many efforts have been made to accelerate test-pattern generation and/or fault simulation. Indeed, for the fault simulation, special purpose high-speed hardware have been developed and have begun to be marketed. However, they are not sufficient because test-pattern generation, which is known as NP-complete even with monotone circuits[1], still remains unimproved. To obtain a good fault coverage within a practical computational time, it is required to accelerate not merely fault simulation but also test-pattern generation. An attempt to accelerate test-pattern generation through parallel processing techniques has been reported,[2] but the technique is not suitable for large scale circuits.

In general, it is relatively easier for the greater part of faults contained in a circuit to generate tests, while some of them are extremely difficult to generate tests that cause a great number of backtracks, and are often failed to generate tests in the practical application. The problems to be solved are summarized as follows:

1) How to deal efficiently with a large number of faults.
2) How to generate test-patterns for the faults which cause a great number of backtracks.

In the light of those, we propose an approach to parallel processing for generating test-patterns for combinational circuits and we have implemented the proposed method on a multi-microcomputer system called LINKS-1[3] to evaluate its performance. From the experiments with real combinational circuits of up to 1,000 gates, it is confirmed that the proposed parallel processing method can offer a high parallelism, achieving an increased acceleration for generating the test-patterns.

2. PARALLEL PROCESSING TECHNIQUES

To find solution to the previously mentioned problems, there have reported many approaches, including the following two methods which are effective and have been adopted in our test-pattern generation system.

1) To use test-pattern generation algorithm and fault simulation alternately in order to handle many faults efficiently.
2) To use efficient test-pattern generation algorithm such as the PODEM[4] or the FAN[5] that could avoid a number of backtracks.

Our system uses a test-pattern generation algorithm based on the PODEM and a fault simulation based on the concurrent fault simulation technique[6]. The test-pattern generation systems that adopt above two methods is illustrated in Figure 1. At the higher level of hierarchy, both test-pattern generation algorithm and fault simulation are alternately performed for each fault, and at the lower level, assignment for the primary inputs (PI Assignment) and implication are iterated until a test-pattern is generated or the given fault is found to be undetectable. In section 2.1 and 2.2, we will discuss the parallelism in the algorithm in Figure 1 and concrete methods for parallel processing.

Our new test-pattern generation system consists of two phases; all the faults are subjected to test-pattern generation under a limited backtracking condition, aborted faults exceeding backtrack limitation are considered to be difficult to generate tests, and are subjected again to the next method. The parallel processing techniques are used in both phases.

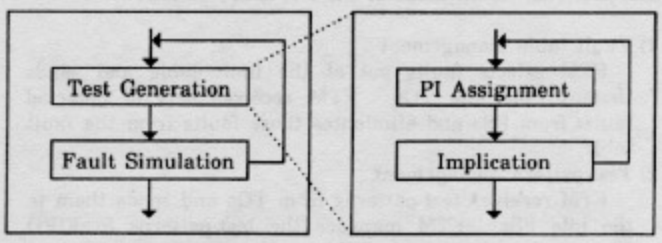


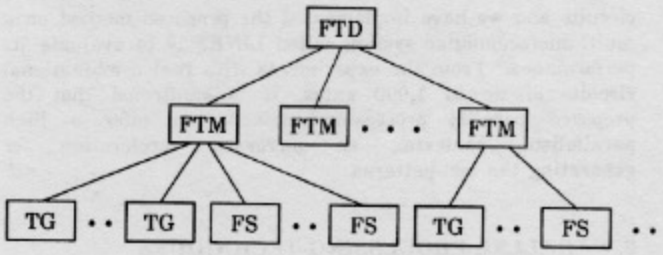
Figure 1: The flow of test-pattern generation

CH2353-1/86/0000/0156$01.00 © 1986 IEEE

2.1 Concurrent Process of Faults

Large scale circuits usually have many faults that can be handled independently. The speed-up of test-pattern generation and fault simulation can be achieved by distributing faults and tasks to separate processors and by obtaining test-patterns of distributed faults simultaneously.

Figure 2 shows the configuration of the multi-processor system that can perform above parallel processing. The multi-processor system of Figure 2 consists of a **Fault Table Divider (FTD)** to divide the fault table prior to the test-pattern generation, **Fault Table Managers (FTMs)** to manage fault subtables given by FTD, **Test Generators (TGs)** to generate test-patterns for the faults given by the FTM, and **Fault Simulators (FSs)** to perform fault simulation with test-patterns given by FTM.



FTD: Fault Table Divider
FTM: Fault Table Manager
TG: Test Generator
FS: Fault Simulator

Figure 2: System Configuration for Concurrent Process of Faults

FTD performs the following tasks:

1) Communication with host computer
   FTD receives the information necessary to test-pattern generation (such as circuit configuration data, limitation of backtracks, etc) from host computer, and sends the result of test-pattern generation to the host computer,
2) Fault table division
   FTD creates a fault table before the test-pattern generation process is started. This fault table is usually obtained by fault modeling and collapsing, or it is given by the host computer. FTD then divides the fault table into a number of small subtabels,
3) Fault table distribution
   FTD sends the divided subtables to the FTMs,
4) Watch of FTMs
   After distributing fault subtables, FTD keeps watching all the FTMs until all the FTMs have completed test-pattern generation.

On receiving the fault subtable, FTM begins to generate test-patterns. Main tasks of the FTMs are as follows:

1) Fault table management
   FTM selects faults out of the fault table and sends them to the idle TGs. FTM receives sets of detected faults from FSs and eliminates those faults from the fault table,
2) Test-pattern management
   FTM receives test-patterns from TGs and sends them to the idle FSs. FTM manages the test-patterns in FIFO manner, in other word, FTM works as a buffer between TGs and FSs,
3) Watch of TGs and FSs
   When FTM is free from above tasks, FTM always watches TGs and FSs, and if an idle processor is found, FTM immediately gives the appropriate tasks.

In the above mentioned parallel processing system, FTD performs **static fault distribution**, where the faults are statically distributed to the test-pattern generation subsystems. Static fault distribution has the following merits and demerits:

(Merits)
1) Throughput is improved by concurrent process of many faults.
2) Communication overhead is negligible because each processor communicates with other processors few times.
3) The number of run for fault simulation could be reduced because of the small amount of faults in the fault table.

(Demerits)
1) Duplicated process might occur for the faults that could be detected by one test-pattern.
2) Test-pattern length is longer than sequential process.

On the other hand FTM performs **dynamic fault distribution**, where the faults are dynamically distributed to the idle slave processor.

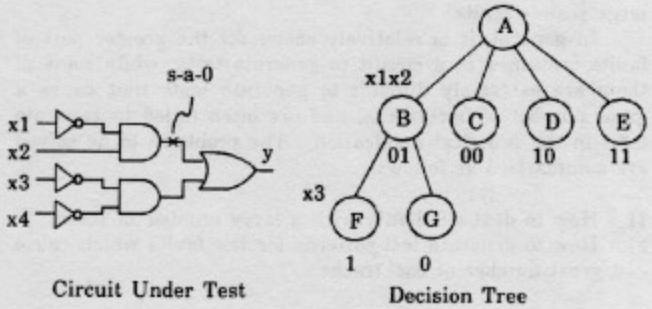
The Following characteristics are seen in the dynamic fault distribution:

(Merits)
1) The performance of test-pattern generation is improved because each slave processor concurrently performs a portion of the test-pattern generation.
2) Possibility of duplicated process is less than static fault distribution due to the way faults are handled.

(Demerits)
1) Total performance saturates with increased slave processors because of the communication bottle neck at the FTM.

2.2 Concurrent Process of Nodes in Decision Tree

The algorithm in our system is based on the PODEM, in which the multiple backtrace technique of the FAN is adopted. The test-pattern generation algorithms such as the PODEM or the FAN use the branch-and-bound method, in which each node in the decision tree is checked in the depth-first-search manner. Each node corresponds to the input pattern; for example, the nodes A, B and F of Figure 3 represent the patterns (xxxx), (01xx) and (011x) respectively.



Circuit Under Test Decision Tree

Figure 3: Branch-and-bound

We propose another parallel processing technique to concurrently handle many nodes in the decision trees. This parallel processing is applied to the faults that have been found to be aborted in the first phase processing. Figure 4 illustrates the difference between conventional (sequential) branch-and-bound and our new method. Each node in the decision tree can be handled independently. Our new method is capable of parallel processing to concurrently handle many nodes in the decision tree.

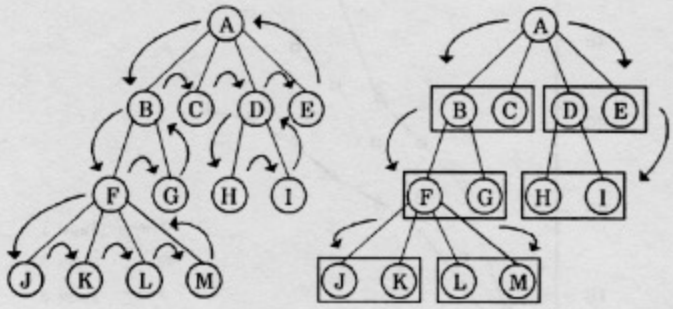
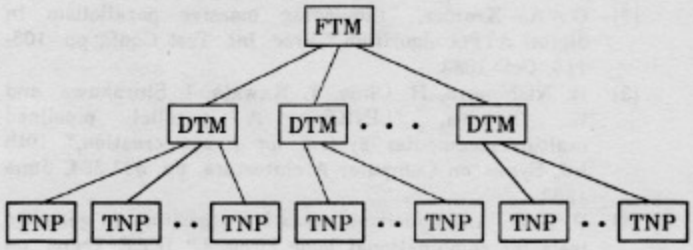


Figure 4: Concurrent Process of Nodes in Decision Tree

Figure 5 shows the configuration of the multi-processor system for the second phase parallel processing. The multi-processor system of Figure 5 consists of **Fault Table Manager (FTM)** to perform the same tasks as FTMs in Figure 2, **Decision Tree Managers (DTMs)** to manage a decision tree, and **Tree Node Processors (TNPs)** to handle each node of the decision tree. DTM performs **tree-node-distribution**, where the nodes of the decision tree is distributed to the slave processors and handled concurrently. On receiving the faults from FTM, DTM begins following tasks:

1) Decision Tree Management
   DTM makes the decision tree empty before generating test-patterns, and updates it if there are some nodes to be attached.
2) Watch of TNPs
   DTM always watches its own successors (TNPs), and when one of them is idle, the DTM starts communication with it -- the DTM receives sets of primary inputs and sends next set of tree-nodes (input-patterns).



FTD: Fault Table Divider
FTM: Fault Table Manager
DTM: Decision Tree Manager
TNP: Tree Node Processor

Figure 5: System Configuration for Concurrent Process of Nodes in Decision Tree

Tasks of TNPs are as follows:

1) Tree-node management
   TNP has two FIFOs; input-FIFO to keep tree-nodes given by the DTM and output-FIFO to store sets of primary inputs that are required to assign a logical values in the next step.
2) Tree-node checking
   TNP performs implication, search of initial objectives and multiple backtrace with each tree-node in the input-FIFO. If some inconsistency is found in implication or search of initial objectives, the node is found to have no test-patterns in its successors and is then abandoned. After multiple backtrace, the set of primary inputs that are to be assigned values are stored in output-FIFO. As soon as input-FIFO has been exhausted, the TNP starts to communicate with the DTM. If the test-pattern is found, TNP gives up handling the rest of nodes in input-FIFO and tries to communicate with DTM to announce the completion of test-pattern generation.

Essentially each tree-node can be handled independently. Therefor, it is expected that computational time for the hard-to-test faults and fault coverage will be improved by tree-node-distribution.

3. IMPLEMENTATION AND EXAMPLES

We have implemented the proposed parallel processing system for the test-pattern generation on a multi-microcomputer system called LINKS-1 and applied it to the actual combinational circuits of up to 1,000 gates in order to evaluate its performance. The LINKS-1 consists of a number of **unit computers** (UCs) which is composed of a CPU Z8000 and a 1MB memory. Each UC can communicate with other UCs through an **inter-computer memory swapping unit (IMSU)** and **bus switch (BS)**. As illustrated in Figure 6, the architecture -- a tree-structure out of the LINKS-1 has equipped with our parallel processing system. The characteristics of the circuits under test are described in Table 1. Fault coverages are obtained by sequential processing using a single unit computer. The faults remained undetected after more than 100 backtracks are aborted.

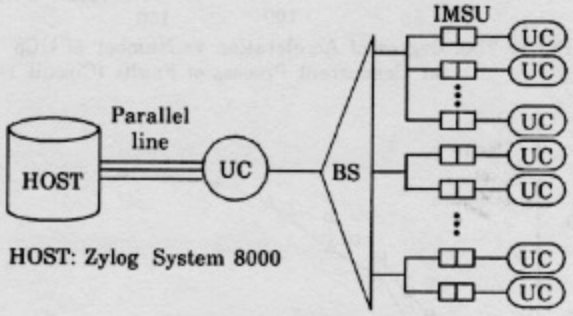


Figure 6: LINKS-1

Table 1: Characteristics of Test Circuits

Circuit Name	Circuit Type	Number of				Fault Coverage
		Gates	Faults	Inputs	Outputs	
circuit 1	ECC	464	1338	8	8	99.8%
circuit 2	ECC	876	1871	33	33	99.3%

3.1 Phase 1

We programmed two kinds of processes for Phase 1; one is for the FTMs and the another is for TGs and FSs. There is no difference in performance between TGs and FSs. Each slave processor has both functions of test-pattern generation and fault simulation, and generates test-patterns if it receives faults. If it receives test-patterns, it begins fault simulation.

FTD's tasks are not implemented because of two reasons; there are only fifty UCs available for experiment, and the time for its tasks is obviously negligible. We have run the parallel processing system with each divided fault table and measured the maximum CPU time as the time for the whole system.

Figure 7 shows the experimental results obtained in phase 1. **Degree of Acceleration** is defined as follows:

Degree of Acceleration = (CPU time with 1 UC) / (CPU time).

3.2 Phase 2

We programmed two more processes; DTM's tasks and TNP's tasks. Experiment has done to measure the time to generate a test-pattern for each fault that is aborted in phase 1.

Figure 8 shows the results for the phase 2 process.

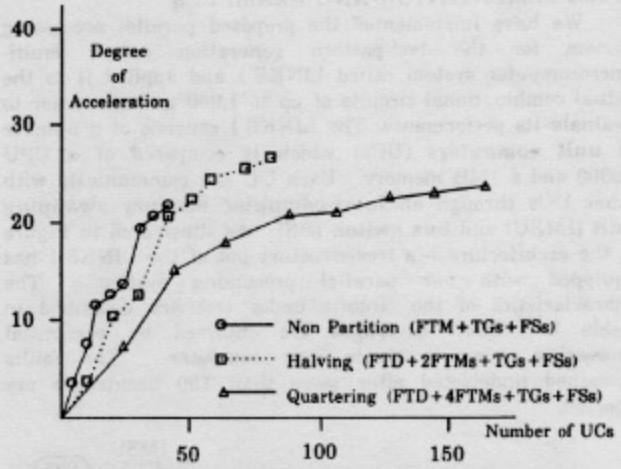


Figure 7(a): Degree of Acceleration vs Number of UCs for Concurrent Process of Faults (Circuit 1)

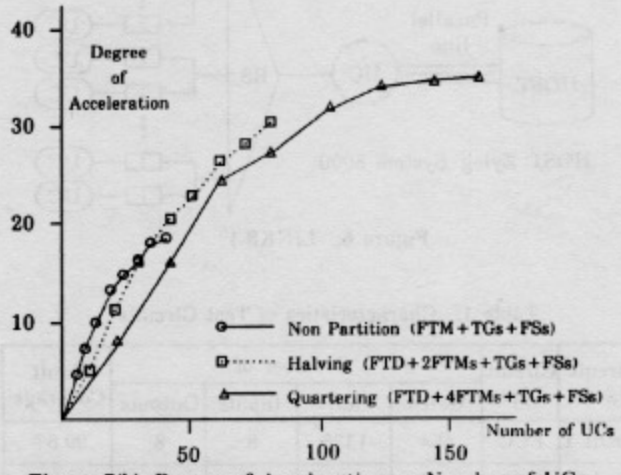


Figure 7(b): Degree of Acceleration vs Number of UCs for Concurrent Process of Faults (Circuit 2)

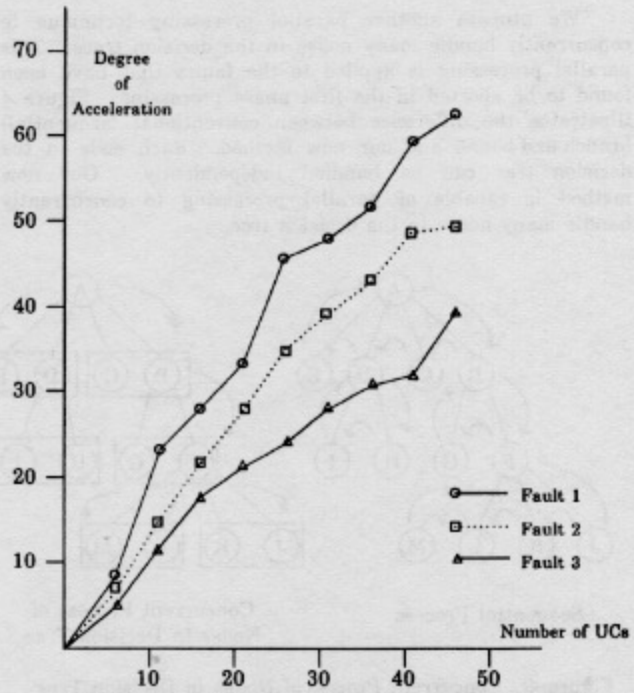


Figure 8: Degree of Acceleration vs Number of UCs for Concurrent Process of Nodes in Decision Tree (Circuit 2)

4. CONCLUSIONS

From the experiments, we can conclude that the proposed approach to parallel processing has proved to be effective to obtain a high parallelism and that it has achieved a high degree of acceleration of test-pattern generation. Also, it has a capability of generating test-patterns for the faults that are extremely difficult with the conventional sequential method.

ACKNOWLEDGEMENT

We would like to thank Prof. K. Ohmura of Osaka University for giving us an opportunity of using LINKS-1.

REFERENCES

[1] H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits," IEEE Trans. on Comp., Vol. C-31, No. 6, pp.555-560, June 1982.

[2] G. A. Kramer, "Employing massive parallelism in digital ATPG algorithm," Proc. Int. Test Conf., pp. 108-114, Oct. 1983.

[3] H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa and K. Ohmura, "LINKS-1: A parallel pipelined multimicrocomputer system for image creation," 10th Int. Symp. on Computer Architecture, pp. 387-394, June 1983.

[4] P. Goel "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Trans. on Comp., Vol. C-30, No. 3, pp. 215-222, March 1981.

[5] H. Fujiwara and T. Shimono, "On the acceleration for test generation algorithms," IEEE Trans. on Comp., Vol. C-32, No. 12, pp. 1137-1144, Dec. 1983.

[6] E. G. Ulrich and T. Backer, "The concurrent simulation of nearly identical digital networks," Proc. 10th Design Automation Workshop, pp. 25-27, June 1973.