Hybrid fault simulation with compiled and event-driven methods

Kenjiro Taniguchi, Hideo Fujii, Seiji Kajihara, and Xiaoqing Wen
Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka 820-8502 Japan
{taniguchi, fujii, kajihara, wen}@aries30.cse.kyutech.ac.jp

1. Introduction

Fault simulation for a logic circuit, that calculates the behavior of a faulty circuit for given test patterns, plays an important role in VLSI design processes [1,2]. The objectives of fault simulation is fault grading, test pattern generation, or fault diagnosis. Run time of fault simulation increases with the circuit size and the number of test patterns. While the reduction of test costs is a critical issue for logic circuit testing, it is required to develop a hi-speed fault simulator.

There are some methods to accelerate fault simulation. Parallel fault simulation is a simple and well-known method that assigns one test pattern or one faulty circuit to each bit of a word. If one word of a machine is 32 bits, 32 patterns or 32 faulty circuits are simulated in parallel. Concurrent fault simulation [3] and deductive fault simulation [4] are the other acceleration methods. These methods calculate faults detectable at each line of the circuit and propagate the faults to outputs of the circuits.

Since the procedure of fault simulation consists of fault injection and logic simulation, keys for acceleration of fault simulation are shown as follows:

(1) To employ a fast logic simulation method.
(2) To avoid waste simulation such that faulty behavior is the same as the fault-free behavior.

As a fast logic simulation method, compiled simulation is well-known, which predetermines the order of lines to be evaluated and implements it in the assembly program. In fault simulation, however, it is difficult to use compiled simulation efficiently, because we don't have to compute the circuit behavior if a faulty circuit behaves the fault-free circuit. Event-driven simulation is more adequate for fault simulation because only difference between the fault-free circuit and faulty circuits can be simulated.

In this paper, we propose a method to speed-up fault simulation. The proposed method takes a hybrid approach with compiled simulation and event-driven simulation. Compiled simulation is applied for fan-out free regions (FFRs). FFRs to be simulated are selected with the event-driven manner. Since the event-driven simulation contributes to avoidance of waste simulation and the compiled simulation contributes to reduction of memory access, the proposed method can reduce the simulation time effectively. Note that this work targets on combinational circuits or a full-scan sequential circuit, and the single stuck-at fault model is assumed. Experimental results for benchmark circuits show that the proposed method could reduce runtime in half compared with concurrent (event-driven) fault simulation.

This paper is organized as follows. In Section 2, we define an FFR, and show the overview of the proposed simulation method. In Section 3, we give the proposed fault simulation method. In Section 4 we show experimental results and in Section 5 we conclude this paper.

2. Preliminary

2.1 FFR (Fanout Free Region)

At first, we define an FFR (Fanout Free Region). FFR is a subcircuit in which every gate has only one output. An input of an FFR is a primary input or a fanout branch, and an output of an FFR is a primary output or a fanout stem. An example is given in Fig. 1. A circuit in Fig. 1(a) consists of two FFRs as shown in Fig. 1(b).
Suppose that two faults *fa* and *fb* exist in a same FFR. If the effects of *fa* and *fb* are propagated to the output of the FFR, we can treat two faults as one fault on the output of the FFR. Therefore, simulation of each fault can be divided into two parts: one is from the fault site to the output of its FFR, and another is from the FFR output to primary outputs.

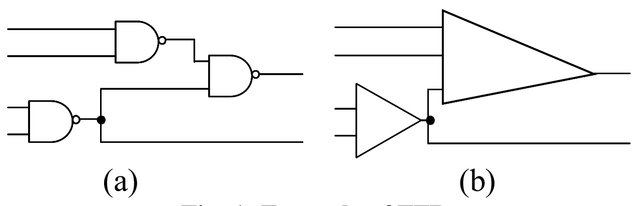

Fig. 1: Example of FFRs

2.2 Overview of the proposed simulation method

In this work we employ event-driven simulation, compiled simulation and parallel simulation. In fault simulation, critical path tracing is also used to check the possibility of fault propagation from a fault site to its FFR output. Faults are explicitly injected only at outputs of

FFRs which are a fanout stem or a primary output. When a fault is injected at a fanout stem, the fault-free value of the fanout stem is inverted. It means that an event appears at fanout branches from the fanout stem. Once an event appears at an input of an FFR, a logic value of the output of the FFR is calculated with compiled fault simulation. If the logic value of the FFR output is different from the fault-free value, it is treated as a new event. But if the logic value of the FFR output is the same as the fault-free value, no event is created. This process is repeated as long as any event exists at a fanout.

3. Details of the proposed method

3.1 Compiled Simulation

Primary inputs or fanout brances are input lines of FFRs, and primary outputs or fanout-stems are output lines of FFRs. The other lines are internal lines of FFRs. To achieve the speed-up of fault simulation, compiled simulation are applied inside FFRs. To realize this, we make logical formula for each FFR as one function. For example, we consider an FFR of Fig. 2. The input lines of the FFR are *a*, *b* and *c*. The output line of the FFR is *e*. Using un-compiled simulation, the value of line *d* is derived after accessing a fanin list of the AND gate and then the value of line *d* is calculated from the values of line *a* and line *b*. And the value of line *e* is derived from the values of line *d* and line *c* after accessing a fanin list of the OR gate.

On the other hand, in the compiled method a function (or a subprogram), where expression $e=(a \wedge b) \vee c$ is described, is prepared for the value of line *e*. It means that we don't have to access the fanin lists of the AND gate and the OR gate to calculate a value of line *e* during simulation. Hence the simulation speed goes up.

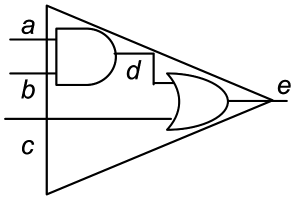

Fig. 2: Example of an FFR

A disadvantage of compiled simulation for fault simulation was that fault injection is difficult. In the above example in Fig. 2, fault injection to line *d* is impossible because a value of line *d* is calculated in the predetermined function. Since the proposed method injects faults at outputs of FFRs, such a problem can be avoided.

3.2 Function for calculating an FFR value

We explain a method of creating a function from a FFR. The basic rules are described in the following.

- Scan all lines of the FFR from the output to inputs with a depth-first manner.
- When we meet an input of a gate first, we output "(".
- When we go back to another input of the gate, we output the symbol of the gate(e.g. $\wedge, \vee$).
- When we go back to the output of the gate, we output ")".

We give an example for the FFR in Fig.2 as follows.

1. Start from line *e* and output the output line"*e*=".
2. Go to input line *d* of OR gate and output "(".
3. Go to input line *a* of AND gate and output "(".
4. Output "*a*".
5. Go to another input line *b* of the AND gate and output the symbol "$\wedge$".
6. Output "*b*".
7. Go back to output line *d* of the AND gate, and output ")".
8. Go to another input line *c* of the OR gate, and output the symbol "$\vee$".
9. Output "*c*".
10. Go back to output line *e* of the OR gate, and output ")".

As a result, we can derive the function "$e=((a \wedge b) \vee c)$" .

3.3 Fault simulation

Creating a function of each FFR is done at a preprocessing phase. A fault list is created at the preprocessing phase too. The main phase of fault simulation is given in the following.

1. Logic simulation for a given test pattern is performed to calculate fault-free values of each line.
2. If there is a fault in a FFR which can be sensitized and propagate to the output of the FFR, set an event to the output of the FFR. If not, stop this procedure. Note that when more than one FFR has an event, choose one that is the closest to primary inputs.
3. If an event exists at a primary output, mark the fault as "detected", and remove the event. If an event exists at a fanout stem, then call a function of the FFRs which include a fanout branch of the fanout stem and calculate the output value of the FFRs.
4. If the output value is different from the fault-free value, set a new event to the output of the FFR.
5. Return to 2.

We illustrate the overall procedure of fault simulation in Fig. 3.

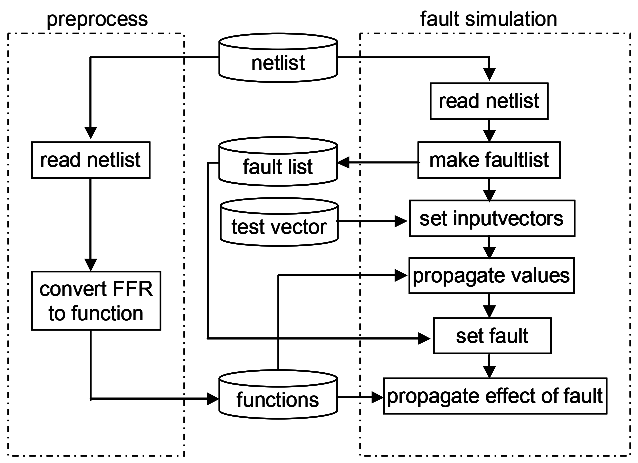

Fig. 3: Procedure of fault simulation

4. Experimental results

We implemented the proposed method using C programming language on a PC (OS: FreeBSD 4.11 Release, CPU: Pentium4 530J (3.0GHz), memory: 512MB), and applied to combinational parts of ITC'99 benchmark circuits. We first made an experiment of logic simulation by the proposed method and a traditional method that calculate logic values by accessing the net list for each gate. About input vectors, we used 10,000 random patterns. We show experimental results in Table1. The first three columns in Table1 show the circuit name, the number of lines and the number of gates. The following two columns show CPU time in seconds, about the proposed method and the traditional method. The last two columns show the reduction time and the percentage of reduction by the proposed method. By introducing compiled simulation, the run time of logic simulation was reduced to a few percents of the traditional method.

We also give experimental results of fault simulation by the proposed method and the traditional method. The method of fault simulation is based on PPSFP (parallel pattern single fault propagation). About input vectors, we used 1,000 random patterns. And during fault simulation, no fault dropping was done, we show results in Table2. The first three columns in Table2 show the circuit name, the number of lines and the number of gates. The following two columns show CPU time in seconds, about the proposed method and the traditional method. The last two columns show the reduction time and the percentage of reduction by the proposed method.

5. Conclusion

We proposed a speed-up method of fault simulation based on a compiled approach and an event-driven approach. Though it had been difficult for fault simulation to use the compiled approach, the proposed method allowed it by partitioning the functions of compiled codes into FFRs. Experimental results for ITC'99 benchmark circuits showed that fault simulation time was reduced in half compared with a traditional PPSFP method. Now this fault simulation can be applied only for combinational circuits. So we are extending this fault simulation to one for sequential circuits.

References

[1] M. Abramovici, M. A. Breuer, A. D. Friedman, *Digital Systems Testing and Testable Design,* Piscataway, New Jersey: IEEE Press, 1990.
[2] M. L. Bushnell, and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory & Mixed-Signal VLSI Circuits*, Kluwer Academic Publishers, 2000.
[3] E. G. Ulrich, T. Baker, “The concurrent simulation of nearly identical digital networks,” *Proc. of Design Automation Workshop*, vol. 6, pp. 145-150, 1973.
[4] D.B. Armstrong, “A deductive method for simulating faults in logic circuits,” *IEEE Trans. on Computer,* vol. C-21, No. 5, pp. 464-471, 1972.

Table1: Results of logic simulation

circuit information			simulation time		evaluation	
circuits	# of lines	# of gates	traditional	proposed	reduction time	%
b04s	1373	512	0.27	0.06	0.21	77.78
b07s	1015	362	0.21	0.04	0.17	80.95
b11s	1190	437	0.23	0.03	0.20	86.96
b12	2517	904	0.52	0.08	0.44	84.62
b14s	11645	4444	10.84	0.26	10.58	97.60
b15s	21804	8338	22.54	0.45	22.09	98.00
b17s	60253	22645	69.41	1.31	68.10	98.11
b20s	23045	8875	24.14	0.52	23.62	97.85
b21s	24053	9259	25.48	0.53	24.95	97.92
b22s	36707	14282	40.95	0.78	40.17	98.10

Table2: Results of parallel pattern simulation

circuit information			simulation time		evaluation	
circuits	# of lines	# of gates	traditional	proposed	reduction time	%
b04s	1373	512	1.79	1.12	0.67	37.43
b07s	1015	362	1.38	0.96	0.42	30.43
b11s	1190	437	2.60	1.57	1.03	39.62
b12	2517	904	2.60	1.73	0.87	33.46
b14s	11645	4444	132.30	58.59	73.71	55.71
b15s	21804	8338	253.86	103.19	150.67	59.35
b17s	60253	22645	761.21	338.96	422.25	55.47
b20s	23045	8875	286.36	137.02	149.34	52.15
b21s	24053	9259	318.04	156.67	161.37	50.74
b22s	36707	14282	489.14	231.27	257.87	52.72