Parallel Fault Simulation Techniques for Large Digital Circuits Including Functional Elements

Akihiko YAMADA*, Nobuo WAKATSUKI* and Kyouji TOMITA*

ABSTRACT

Parallel fault simulation techniques for functional elements are described in this paper. This technique can be applied to synchronous and asynchronous large digital circuits consisting of gates, flip-flops, ROMs, RAMs, Content Addressable Memories and Register Files.

1. INTRODUCTION

Because of the rapid progress of semi-conductor fabrication technology, the density and complexity of integrated circuits have been remarkably increased. This tendency will become even stronger in the future. As a result, with this increase, many functional memory elements such as ROM and RAM are being used in logic circuits recently. The use of these functional elements has made the test of the logic circuits more difficult and has made the programs in this field more complicated.

The main purpose of this paper is to introduce simple and convenient parallel fault simulation techniques of the functional elements, which have been developed for and used in this fault simulator.

2. PARALLEL EVALUATION TECHNIQUES FOR FUNCTIONAL ELEMENTS

A gate level fault simulator is not efficient to accommodate larger circuits than those with several thousands gates, because a large storage area is necessary to store a large number of fault and a fault-free circuits. Furthermore simulation time for the large circuits will be increased.

It seems that a functional parallel fault simulator with functional models described in this paper provides great possibilities to fault simulation of large scale circuits.

This paper first appeared in Japanese in Joho—Shori (Journal of the Information Processing Society of Japan), Vol. 17, No. 7 (1976), pp. 569~576.

* Systems Development Department, Computer Engineering Division Nippon Electric Co., Ltd.

Flip-Flops

Many MSIs such as counters and shift registers are expanded to their equivalent circuit configurations composed of several flip-flops and gates; all kinds of flip-flops are processed in this fault simulator as the primitive elements having N-inputs and 2-outputs for simulation efficiency, where N is decided by a type of flip-flop. This basic concept for flip-flops makes it possible to simulate various functional register modules made up of some flip-flops.

However, the following several problems must be solved to simulate a flip-flop as a black-box in parallel.

- o For an edge triggered flip-flop, the positive edge of the clock pulse given to the clock pin of the flip-flop must be identified by its evaluation routine in parallel.
- o All functions of a flip-flop must be completely converted into a boolean equation representation to derive the high efficiency of parallel fault simulators.

For an edge triggered flip-flop, two value areas are reserved. The first value area is used to represent the current outputs of the flip-flop, Q and $\overline{Q}$. The second value area is for examining a positive edge of a clock pulse to be supplied to the flip-flop. That is, the preceding clock value to the flip-flop is stored in the second value area when the flip-flop is evaluated by its evaluation routine. Detection of the positive edge is derived from eq. (1) by simple boolean manipulation using current clock input value and the proceding clock value already stored in the second value area.

$$Cr = Cn \cdot \overline{Cn-1} \qquad (1)$$

Where Cr = representing positive edge of clock pulse,

Cn = current clock value, Cn-1 = preceding clock value.

Further Cr derived from eq. (1) is used in eqs.(2) through (5), in which functions of several typical edge triggered flip-flops with asynchronous set and reset inputs are expressed. Sa and Ra in these equations mean asynchronous set and reset inputs.

D type flip-flop $Q = Sa + \overline{Ra} \cdot (Cr \cdot D + \overline{Cr} \cdot Qn)$ (2)

J-K type flip-flop $Q = Sa + \overline{Ra} \cdot \left\{Cr \cdot (\overline{K} \cdot Qn + J \cdot \overline{Qn}) + \overline{Cr} \cdot Qn\right\}$ (3)

R-S type flip-flop $Q = Sa + \overline{Ra} \cdot \left\{Cr \cdot (S + \overline{R} \cdot Qn) + \overline{Cr} \cdot Qn\right\}$ (4)

T type flip-flop $Q = Sa + \overline{Ra} \cdot (Cr \oplus Qn)$ (5)

Parallel Address Decoding Technique

This is a fundamental technique which allows parallel simulation of ROM, RAM, etc. These functional elements have a built-in address decoder to select the desirable word. An example of the parallel address decoding technique is shown in Fig. 1. An address decoder with two inputs and four outputs and its truth table are indicated in Fig. 1 (a). Apparently, the relation between inputs and outputs is based on binary notation of inputs. A simple parallel evaluation routine is shown in Fig. 1(b) and is quite acceptable to simulate the address decoder. The routine observes the bits configuration of a variable b. A variable m represents the number of inputs to the decoder. According to the bits configuration of the variable b. The routine derives boolean equations as follows:

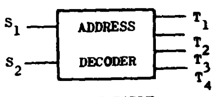
$$\left.\begin{array}{ll} \text{When } b = 00, & T_1 = \overline{S_2} \cdot \overline{S_1} \\ 01, & T_2 = \overline{S_2} \cdot S_1 \\ 10, & T_3 = S_2 \cdot \overline{S_1} \\ 11, & T_4 = S_2 \cdot S_1 \end{array}\right\} \qquad (6)$$

An example of parallel fault simulation for the fault-free address decoder presented in Fig. 1(a) is shown in Fig. 1(c). Three faults F_1, F_2 and F_3 propagate on the inputs of the decoder. G means a fault-free circuit. Substituting $S_1 = 0011$ and $S_2 = 0110$ into eq. (6), simulation resultants T_1 through T_4, as shown in Fig. 3, can be obtained. Of course, this technique is applicable to the parallel simulation of various decoders and multiplexers often used in logic circuits.

To clarify the address decoding technique, one example is given. Figure 2 is a simplified evaluation routine of RAM with 2^m words of n bits. As a result of the address decoding operation, address inputs (S_{1-m}) are decoded and a mask bits pattern (A) needed to select a desired word is generated. Every word ($B_{j,k}$; kth bit of jth word) of RAM is examined with A to either write-in or retrieve data.

3. FAULT SIMULATOR AND SIMULATION RESULTS

The parallel fault simulator has been implemented using above mentioned parallel evaluation technique for functional elements. It employes selective tracing simulation technique, and maximum 960 fault and fault-free circuits can be simultaneously processed at one path. Functional models for Flip-Flops, ROMs, RAMs, CAMs and Register Files are available.



TRUTH TABLE

S_2	S_1	T_1	T_2	T_3	T_4
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(a) 2-inputs and 4-outputs decoder and its truth table.

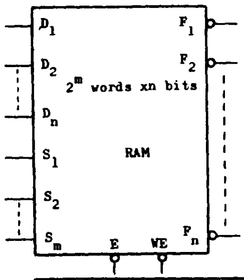
```
m = 2
b = 0
DO j = 1 TO $2^m$
    Tj = all 1
    DO k = 1 TO m
        IF kth bit of b = 0
            THEN $T_j = T_j \cdot \overline{S_k}$
            ELSE $T_j = T_j \cdot S_k$
    END
    b = b + 1 (b is incremented by one)
END
```

(b) Parallel evaluation routine for the decoder.

	G	F_1	F_2	F_3
S_1	0	0	1	1
S_2	0	1	1	0
T_1	1	0	0	0
T_2	0	0	0	1
T_3	0	1	0	0
T_4	0	0	1	0

(c) An example of parallel fault simulation for the decoder.

Fig. 1 Explanation of parallel address decoding technique



Where $D_1 \sim D_n$ = Data Inputs,

$S_1 \sim S_m$ = Address Inputs,

E = Chip Select Input,

WE = Write Enable Input,

$F_1 \sim F_n$ = Data Outputs.

```
/* INITIALIZATION */
DO j = 1 TO n
    $F_j$ = all 0
END
b = 0
$w = \overline{E} \cdot \overline{WE}$
DO j = 1 TO $2^m$
    A = all 1
    /* ADDRESS DECODING */
    DO K = 1 TO m
        IF k th bit of b = 0
            THEN $A = A \cdot \overline{S_k}$
            ELSE $A = A \cdot S_k$
    END
    /* WRITE AND READ */
    DO k = 1 TO n
        $B_{j,k} = W \cdot A \cdot D_k + W \cdot A \cdot B_{j,k}$
        $F_k = F_k + A \cdot B_{j,k}$
    END
    b = b + 1 (b is incremented by one).
END
    /* OUTPUT JUSTIFICATION */
    $r = \overline{E} \cdot WE$
DO j = 1 TO n
    $F_j = \overline{r \cdot F_j}$
END
```

Fig. 2 Simplified evaluation routine for 2^m words x n bits RAM

This fault simulator has been programmed in Assembler and PL/1-like languages. It runs on an in-house NEAC 2200/700 having 524 kilo-characters memories and can presently handle a circuit of 3,000 gates. The comparison of this parallel fault simulator and notable fault simulators is performed in Table 1 using several actual simulation results.

The simple experiment was carried out for a circuit F which consists of thirty 4x4 Register File elements, control gates and flip-flops. Two simulation models, Fa and Fb, were made for the circuit to study the efficiency of functional simulation. The result shows that the efficiency of functional model, Fa, is two times superior to that of gate model, Fb, in which each Register File element has been expanded to the equivalent circuit composed of sixteen T type flip-flops and fifty two gates. Computation time and storage saving are quite remarkable in functional fault simulation.

Table 1 Comparison of this fault simulator and notable fault simulators

	Circuit		Gates (No.)	Flip-Flops (No.)	Functional Elements (No.)	Faults simulated (No.)	Vectors simulated (No.)	Faults detected (%)	CPU time (sec.)	Simulation efficiency = $\frac{\text{FaultsxVectors}}{\text{CPU time (ms.)}}$
(1)	A		604	40	0	1492	3024	93	1068	4.22
	B		1156	32	0	3471	778	82	705	3.83
	C		435	12	1K ROM : 3	1175	2431	90	975	2.93
	D		586	32	1K ROM : 3 4x4 CAM: 3	1646	2160	84	3043	1.17
	E		446	11	16x4 RAM:9	1428	676	94	840	1.15
	F	a	527	12	4x4 Register File : 30	1897	372	90	572	1.23
		b	2099	492	0	8425		98	5340	0.59
	G		161	0	16x4 RAM: 30	803	842	92	1524	0.44
(2)	H		1003	-	0	155	1734	98	373	0.72
	I		1025	48	0	310	1235	68	534	0.72
(3)	J		2476	-	0	2361	16	-	327	0.12
	K		6602	-	0	2147	377	-	510	1.59

(1) The parallel fault simulator described in this paper.
(2) Concurrent fault simulator of GTE on IBM 370/158 with 0.6 megabytes [2]
(3) Deductive fault simulator of BTL on IBM 360/67 with 4 megabytes [1]

REFERENCES

[1] H.Y. Chang, S. G. Chappell, C. H. Elemendorf and L. D. Schmidt: "Comparison of Parallel and Deductive Fault Simulation Methods" IEEE Transactions on Computers, November 1974.

[2] D. M. Schuler, E. G. Ulrich, T. E. Baker and S. P. Bryant: "Random Test Generation using Concurrent Logic Simulator" Proceedings of 12th DA Conference, June 1975.