IPSJ Transactions on System LSI Design Methodology Vol.6 127–134 (Aug. 2013)

[DOI: 10.2197/ipsjtsldm.6.127]

Regular Paper

Efficient Fault Simulation Algorithms for Analyzing Soft Error Propagation in Sequential Circuits

Taiga Takata[1,a) Masayoshi Yoshimura[2,b) Yusuke Matsunaga[2,c)

Received: November 28, 2012, Revised: March 8, 2013,
Accepted: April 26, 2013, Released: August 5, 2013

Abstract: This paper presents two acceleration techniques of fault simulation for analyzing soft error propagation in sequential circuits. One is an exact technique and the other is a heuristic technique. Since these techniques are independent on how the logic functions of circuits are evaluated, they can be combined with other techniques which accelerate evaluations of the logic functions of circuits, such as event-driven simulation, single pattern parallel fault propagation (SPPFP). Experimental results show that applying the exact technique makes a fault simulator with event-driven simulation and SPPFP 30–143 times faster. A fault simulator with the exact technique finished for several large-scale circuits in 4.6 hours or less, while a fault simulator without the exact technique could not finish for such circuits in 72 hours. Furthermore, applying the heuristic technique makes a fault simulator with the exact technique about 7–17 times faster with only 0.5–2.2% estimation error.

Keywords: soft error, fault simulation, acceleration technique

1. Introduction

Collisions of neutrons into silicon atoms in circuits may cause an abnormal pulse at the output of a logic gate or an incorrect bit flip of a value held in a flip-flop (FF). These phenomena are called Single Event Fault (SEF). An SEF may cause incorrect values of FFs at the next clock edge. For example, an SEF occurring at a logic gate and being propagated into FFs makes incorrect values of FFs. Incorrect values of FFs are called sequential transient error (STE) in this paper. An STE may cause incorrect values at primary outputs. Incorrect values at primary outputs are called failure of circuit. Applying SEF mitigation techniques [1], [8] generally cause overhead of delay, area or power. Analyzing SEF-tolerance of a circuit and identifying vulnerable parts in the circuit are important to mitigate SEF-induced failure with small overhead.

Analysis of STE propagation is one of the important keys for analyzing SEF-tolerance of a circuit and identifying vulnerable parts in the circuit. The goal of the analysis is to estimate occurrence probability of failure under the condition of an STE occurrence. Three kinds of approaches have been studied in the past. The first approach is analyzing the behavior of a finite state machine stochastically with representing the state transitions of the finite state machine as a Markov chain model [4]. The method [4] is exact if occurrences of input vectors have no temporal correlation. The time complexity of the method [4], however, is exponential to the number of primary inputs and FFs. The second approach is analyzing circuit behavior during a certain clock cycles after STE occurrence using time expansion model [6], [7]. The time complexity of the second approach is also exponential to the number of primary inputs and FFs. The methods [6], [7] use Binary Decision Diagrams (BDDs) for efficient analysis. Memory size of a BDD is, however, often larger than several billion bytes if a circuit has more than 40 primary inputs or FFs. Thus, these methods [4], [6], [7] are not practical for large-scale circuits. The last approach is based on fault simulation with a given sequence of input vectors [3]. Since the time complexity is polynomial to circuit size multiplied by the length of a given sequence of input vectors, the last approach is applicable to large-scale circuits. Thus, the last approach is practical if a sequence of input vectors is provided.

The problem of fault simulation for analyzing STE propagation is run-time. Fault simulation for analyzing STE propagation is different from that for permanent faults in the following points.

- An SEF never change the logic function of a circuit, while a permanent fault changes it.
- If an STE does not affect values of FFs at the next clock edge, simulation for the following clock cycles is unnecessary. On the other hand, simulation for every clock cycle is necessary for a permanent fault, since a permanent fault changes the logic function of a circuit.
- Let S and T be the size of a given circuit and the number of clock cycles. An SEF is distinguished from other SEFs which occur at different clock cycles on the same location. Thus, the number of STEs can be proportional to $S \times T$, while the number of permanent faults is proportional to S.

[1] Cadence Design Systems, Japan, Yokohama, Kanagawa 222–0033, Japan
[2] Department of Advanced Information Technology, Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka 819–0395, Japan
[a)] taiga@soc.ait.kyushu-u.ac.jp
[b)] yosimura@soc.ait.kyushu-u.ac.jp
[c)] matsunaga@soc.ait.kyushu-u.ac.jp

The time complexity of fault simulation for one STE is proportional to $S \times T$, Therefore, time complexity of a naive fault simulation for all the STEs is proportional to $S^2 \times T^2$. That is not practical for a large-scale circuit with large number of clock cycles.

This paper presents acceleration techniques for fault simulation. There are two points for improving a naive fault simulation.

- Several different STEs at a clock cycle often lead to same STEs at the next clock cycle. Since SEFs do not change the function of a circuit, simulation results for the same STEs are guaranteed to be the same. Therefore, individually simulating each of the same STEs is inefficient.
- An STE which does not cause failure and does not affect any FF at the next clock cycle is called as 'masked'. Empirically, an STE which remains through many clock cycles after an SEF occurrence is rarely masked. Thus, such an STE may be considered as failure without simulating the following clock cycles.

Based on the above points, the following acceleration techniques are proposed.

- Exact technique - a simple technique to collapse overlapping STEs at a clock cycle into one STE. Employing this technique has no loss of accuracy, since simulation results for the same STEs are guaranteed to be the same. This technique seems to be trivial, however, impacts of this technique have not been demonstrated quantitatively in the past. This technique cannot be applied to fault simulations for permanent faults, since a permanent fault changes the function of a circuit depending on the location of the fault.
- Heuristic technique - a technique to terminate simulation for a set of STEs where enough masked STEs are found. At first, in simulation for each clock cycle, the set of STEs is partitioned into subsets with respect to each SEF occurrence clock. The total number N of STEs which do not cause failure is adequately guessed for each subset. Then, if the current number of masked STEs in a subset exceeds a certain rate of N, then simulation for all the other STEs in the subset are terminated, and they are considered as failure. Since STEs which cause failure never estimated as not failure with this technique, a pessimistic analysis is guaranteed.

Since these techniques are independent on how the logic function of a circuit is evaluated, they can be combined with other techniques which accelerate evaluations of the logic functions of circuits, such as event-driven simulation and SPPFP [2], [3], [9]. Experimental results show that applying the exact technique makes a fault simulator with event-driven simulation and SPPFP about 30–143 times faster. A fault simulator with the exact technique finished for several large-scale circuits in about 4.6 hours or less, while a fault simulator without the exact technique could not finish for such circuits in 72 hours. Furthermore, applying the heuristic technique makes a fault simulator with the exact technique about 7–17 times faster with only 0.5–2.2% overestimates on the rate of STEs which cause failures. Since the analysis with the heuristic technique is slightly pessimistic, the heuristic technique is useful to judge whether required soft error tolerance have been achieved or not quickly.

The rest of this paper is organized as follows. Section 2 gives an definitions and a problem formulation. Section 3 shows a naive fault simulation algorithm. Section 4 shows the proposed acceleration techniques. Section 5 shows experimental results. Finally, section 6 concludes this paper.

2. Preliminaries

Let $G = (V, E)$ denote a graph which represents the relation of connection for a sequential circuit. A node in V corresponds to a primary input, a primary output, a logic gate, or an FF. Let PI, PO, LG, FF denote the set of all the primary inputs, the set of all the primary outputs, the set of all the logic gates and the set of all the FFs, respectively. If and only if the output of v is an input of w, an edge (v, w) is in E. A **fanin** of a node v is an immediate predecessor of v. The set of all the fanins of v is defined by $FI(v) = \{u \mid \exists(u, v) \in E\}$. A Boolean function which represents the output value of $v \in LG$ corresponding to values of $FI(v)$ is called **local function** of v, denoted by $LF_v : B^{|FI(v)|} \rightarrow B$. A Boolean function which represents the output value of v corresponding to values of primary inputs and FFs is called **global function** of v, denoted by $GF_v : B^{|PI|} \times B^{|FF|} \rightarrow B$. The global function of node $v \in LG$ is equivalent to $LF_v(GF_{v_0}, GF_{v_1}, ..., GF_{v_n})$, where $FI(v) = \{v_0, v_1, ..., v_n\}$.

An **input vector** is a bit vector which represents values of all the primary inputs. The m th value of an input vector represents the value of the m th primary input. An **input sequence** is a sequence of input vectors. The i th input vector of an input sequence IS is denoted by IS_i. The length of an input sequence IS is denoted by $|IS|$. An **output vector** is a bit vector which represents values of all the primary outputs. An **FF vector** is a bit vector which represents values of all the FFs. Let FV_i be an FF vector at the i th clock, which means that the m th FF latches the m th value of FV_i at the i th clock edge. Then, $i+1$ th output vector OV_{i+1} for each $i \in \{0, 1, ..., |IS|-1\}$ is defined with the following expression, where $PO = \{o_0, ..., o_m\}$.

$$OV_{i+1} = (GF_{o_0}(IS_i, FV_i), ..., GF_{o_m}(IS_i, FV_i))$$

Similarly, $i+1$ th FF vector FV_{i+1} for each $i \in \{0, 1, ..., |IS|-1\}$ is defined with the following expression, where $FF = \{f_0, ..., f_n\}$.

$$FV_{i+1} = (GF_{f_0}(IS_i, FV_i), ..., GF_{f_n}(IS_i, FV_i)) \tag{1}$$

An **STE** denotes a pair of an FF vector and a clock[*1]. "STE (FV^F, j) is injected" means that FFs latch the values in FV^F at the j th clock edge, no matter how FV^F is different from FV_j in Eq. (1). j is called the injection clock of STE (FV^F, j). Let $OV_i^F(e)$ and $FV_i^F(e)$ denote the output vector and the FF vector at i th clock where an STE e is injected, respectively. Then, an STE e is called to be failure if and only if the following function $Fail$ is *true*.

$$\begin{aligned} &Fail(e, G, IS, FV_0) = \\ &\quad (\exists i \in \{1, ..., |IS|\}, OV_i \neq OV_i^F(e)) \\ &\quad \vee (FV_{|IS|} \neq FV_{|IS|}^F(e)) \end{aligned} \tag{2}$$

[*1] STE is briefly defined in Section 1, however, the definition is not mathematical. STE is mathematically redefined here.

The problem of fault simulation for STEs is computing $Fail$ for given G, FV_0, IS and a set of STEs $ESet$. **Figure 1** shows a model of fault simulation where STE $(FV_1^F, 1)$ is injected. An FF vector at the first clock of the lower circuit is changed from FV_1 to FV_1^F. Then, OV_2^F, OV_3^F and FV_3^F are computed and respectively compared to OV_2, OV_3 and FV_3.

3. Naive fault simulation

NaiveFaultSimulation in **Fig. 2** denotes a pseudo code of a naive fault simulation algorithm. Both $FVSet$ and $FVSet'$ denote a set of pairs of an FF vector and an STE. If STE e is injected and if the FF vector at the i th clock is FV_i^F, (FV_i^F, e) is stored in $FVSet$. $MESet$ is a set which holds masked STEs. $ESet(i)$ in line 6 represents the set of all the STEs whose injection clock is i. Judge in line 9 computes whether e can be judged as failure or masked at the $i+1$ th clock. If and only if e can be judged as failure or masked with the expression 2, Judge returns $true$. If Judge returns $false$, the pair of the FF vector at $i+1$ th clock and e is added to $FVSet'$ in line 10. NaiveFaultSimulation returns $MESet$. If and only if e is not included in $MESet$, $Fail(e, G, IS, FV_0)$ is $true$.

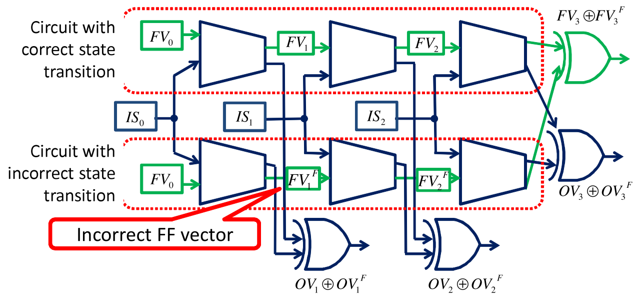


Fig. 1 A model of fault simulation for STE $(FV_1^F, 1)$ where $|IS| = 3$.

```
NaiveFaultSimulation($(V, E)$, $IS$, $FV_0$, $ESet$){
1:  $FVSet$;   //declare a set of pairs(an FF vector, an STE)
2:  $FVSet'$;  //declare a set of pairs(an FF vector, an STE)
3:  $MESet$; //declare a set of STEs
4:  foreach $i$ from 0 to $|IS|-1${
5:    Compute $OV_{i+1}$ and $FV_{i+1}$;
6:    $FVSet \leftarrow FVSet \cup \{(FV_i^F, e) \mid e \in ESet(i)\}$;
7:    foreach $(FV_i^F, e) \in FVSet${
8:      Compute $OV_{i+1}^F$ and $FV_{i+1}^F$;
9:      bool $finish \leftarrow$ Judge($\{e\}, OV_{i+1}, FV_{i+1},$
                $OV_{i+1}^F, FV_{i+1}^F, i+1, IS, MESet$);
10:     if $(finish = false)$
          $FVSet' \leftarrow \{(FV_{i+1}^F, e)\} \cup FVSet'$;
11:   }
12:   $FVSet \leftarrow FVSet'$;
13:   $FVSet' \leftarrow \phi$;
14: }
15: return $MESet$;
}

bool Judge($Er, OV, FV, OV^F, FV^F, i, IS, MESet$){
16: if $((OV = OV^F) \wedge (FV = FV^F))${
17:   $MESet \leftarrow MESet \cup Er$;
18:   return $true$;
      // the STEs in $Er$ are masked
19: }
20: else if $((OV \neq OV^F) \vee ((i = |IS|) \wedge (FV \neq FV^F)))${
21:   return $true$;
      // the STEs in $Er$ cause failure
22: }
23: return $false$;
}
```

Fig. 2 A pseudo code of a naive fault simulation algorithm for analyzing incorrect FF vectors.

Both the time complexity of line 5 and that of line 8 are $O(|V|)$. The time complexity of line 9 is $O(|PO| + |FF|)$. $|FVSet|$ for each $i \in \{0, ..., |IS|-1\}$ is equal to $i \cdot |V|$ in the worst case. Thus, the time complexity from line 5 to line 13 is $O(|IS| \cdot |V|^2)$. Based on the above, the time complexity of NaiveFaultSimulation is $O(|IS|^2 \cdot |V|^2)$.

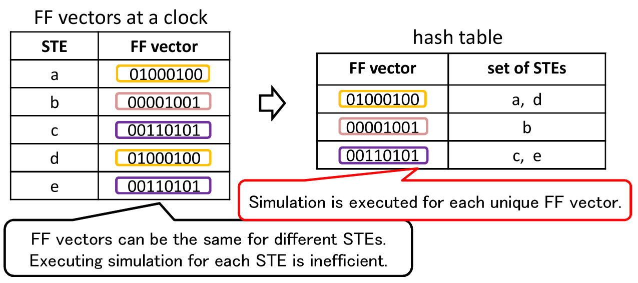
There are several acceleration techniques [2], [3], [9] which can be applied to the computation of line 8. In the experiments of this paper, SPPFP has employed to simulate 64 FF vectors in $FVSet$ simultaneously. Event-driven simulation has also employed [9]. These techniques don't affect the time complexity of this algorithm.

4. Acceleration Techniques for Fault Simulation

4.1 An Exact Technique with Eliminating Overlaps of FF Vectors

This section presents a simple technique to collapse overlapping same FF vectors induced by different STEs into one FF vector. If an FF vector FV_i^F induced by one STE is identical to other FF vectors induced by other STEs, the simulation results of the following clocks are also the same, since the function of a circuit is the same for any STE. Thus, one time of simulation for FV_i^F is enough for such STEs. **Figure 3** shows an example where overlapping FF vectors are collapsed with a hash table. The key of the hash table is FF vector, and the value of the hash table is set of the corresponding STEs. This technique accelerates fault simulation with reducing the number of times of simulations without any loss of accuracy.

EfficientFaultSimulation in **Fig. 4** denotes a pseudo code of a fault simulation algorithm with eliminating overlaps of FF vectors. Judge in EfficientFaultSimulation is the same with that shown in Fig. 2. $HashMap$ denotes a hash table whose key is an FF vector and whose value is a set of STEs. $HashMap(FV)$



FF vectors at a clock

STE	FF vector
a	01000100
b	00001001
c	00110101
d	01000100
e	00110101

hash table

FF vector	set of STEs
01000100	a, d
00001001	b
00110101	c, e

Simulation is executed for each unique FF vector.
FF vectors can be the same for different STEs.
Executing simulation for each STE is inefficient.

Fig. 3 An example of a hash table used in the exact technique.

```
EfficientFaultSimulation($(V, E)$, $IS$, $FV_0$, $ESet$){
    //declare a set of pairs(an FF vector, a set of STEs)
1:  $FVSet$;
    //declare a hash table of (an FF vector, a set of STEs)
2:  $HashMap$;
3:  $MESet$; //declare a set of STEs
4:  foreach $i$ from 0 to $|IS| - 1${
5:      Compute $OV_{i+1}$ and $FV_{i+1}$;
6:      foreach $(FV_i^F, Er) \in FVSet${
7:          Compute $OV_{i+1}^F$ and $FV_{i+1}^F$;
8:          bool $finish \leftarrow$ Judge($Er, OV_{i+1}, FV_{i+1},$
                    $OV_{i+1}^F, FV_{i+1}^F, i+1, IS, MESet$);
9:          if ($finish = false$)
                SetNextVec($FV_{i+1}^F, Er, HashMap$);
10:     }
11:     if ($i \neq |IS| - 1$) {
12:         foreach $(FV^F, i+1) \in ESet(i+1)$ {
13:             SetNextVec($FV^F, \{(FV^F, i+1)\}, HashMap$)
14:         }
15:     }
16:     $FVSet \leftarrow \phi$;
17:     $FVSet \leftarrow$ a set of entries in $HashMap$;
18:     Make $HashMap$ empty;
19: }
20: return $MESet$;
}

SetNextVec($FV^F$, $Er$, $HashMap$){
21: if ($FV^F$ has not registered in $HashMap$)
        Register $(FV^F, Er)$ to $HashMap$;
22: else
        $HashMap(FV^F) \leftarrow HashMap(FV^F) \cup Er$;
}
```

Fig. 4 A pseudo code of fault simulation with eliminating overlaps of FF vectors.

denotes the reference to a set of STEs which corresponds to a key FV. If FV^F has not registered on $HashTable$ in SetNextVec, (FV^F, Er) is registered on $HashTable$. Otherwise, each STE in Er is added to $HashTable(FV_i^F)$. The time complexity of EffcientFaultSimulation is the same with that of NaiveFaultSimulation, however, EfficientFaultSimulation can be faster since $|FVSet|$ is significantly reduced.

4.2 A Heuristic Technique with Terminating Simulation

An STE which remains through many clock cycles after an SEF occurs is rarely masked. Thus, such an STE is considered as failure without simulating the following clock cycles. This section presents a heuristic technique to terminate simulation for a set of STEs where enough masked STEs are found.

Let $SubESet$ be a subset of $ESet$. Terminate($SubESet$, $FVSet$) in **Fig. 5** is a procedure which removes all the STEs in $SubESet$ from $FVSet$. Inserting procedure Terminate($SubESet$, $FVSet$) at between line 5 and line 6 in Fig. 4 makes simulations for STEs in $SubESet$ terminate, which may reduce $|FVSet|$ in line 6 in Fig. 4. Reducing $|FVSet|$ at line 6

```
Terminate($SubESet$, $FVSet$){
1:  foreach $(FV^F, Er) \in FVSet${
2:      foreach $e \in Er${
3:          if ($e \in SubESet$)
4:              $Er \leftarrow Er - \{e\}$;
5:      }
6:      if ($Er = \phi$)
7:          $FVSet \leftarrow FVSet - \{(FV^F, Er)\}$;
8:  }
}
```

Fig. 5 A pseudo code of a part of terminating simulation.

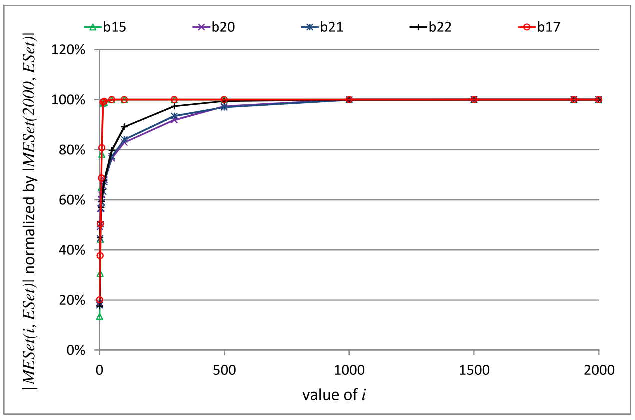


Fig. 6 $|MESet(i, ESet)|$ normalized by $|MESet(2000, ESet)|$ for various values of i.

would cause speed-up of fault simulation since the operation from line 7 to line 9 in Fig. 4 may be executed less times. Inserting Terminate($SubESet$, $FVSet$), however, may cause loss of accuracy. Inserting Terminate($SubESet$, $FVSet$) makes all the STEs in $SubESet$ be not included in $MESet$, which means that they are judged as failure. If $SubESet$ includes STE e which is going to be masked, Terminate($SubESet$, $FVSet$) makes e be incorrectly judged as failure. Since any STE which causes failure never judged as not failure with Terminate($SubESet$, $FVSet$), the simulation results of inserting Terminate($SubESet$, $FVSet$) are guaranteed to be pessimistic. If $|SubESet|$ is large and almost all the STEs in $SubESet$ cause failure, such $SubESet$ is expected to make large speed-up with keeping high accuracy.

Let $MESet(i, ESet)$ be the set of STEs which are in $Eset$ and masked during i clock cycles after the injection clock. **Figure 6** shows $|MESet(i, ESet)|$ normalized by $|MESet(2000, ESet)|$ on large benchmark circuits in ITC'99 benchmark set. This figure shows that the slope of $|MESet(i, ESet)|$ is monotonically decreased for increasing i. Let assume the slope of a tangent of $|MESet(i, ESet)|$ is approximated with $|MESet(i, ESet) - MESet(i-1, ESet)|$. Then, the number of all the STEs masked during $|IS|$ clocks is estimated with the following $N(i, ESet)$ at i th clock.

$$N(i, ESet) = |MESet(i, ESet) - MESet(i-1, ESet)| \cdot (|IS| - i) + |MESet(i, ESet)|$$

Figure 7 shows an example of a curve of $ESet(i, ESet)$ and $N(500, ESet)$. If the slope of $|MESet(i, ESet)|$ is assumed to be monotonically decreased, $N(i, ESet)$ is guaranteed to be larger

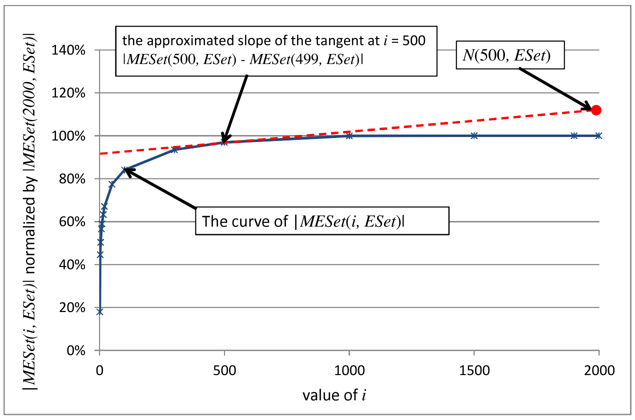


Fig. 7 An example of $|MESet(i, ESet)|$ and $N(500, ESet)$.

```
TerminateSimulation($i$, $P$, $ESet$, $FVSet$){
1:  foreach $j \in \{0, ..., i-1\}$ {
2:      if ($|MESet(i-j, ESet(j))| \geq N(i-j, ESet(j)) \cdot P$)
3:          Terminate($ESet(j) - MESet(i-j, ESet(j)), FVSet$);
4:  }
}
```

Fig. 8 A pseudo code of Terminating Simulation.

than $|MESet(|IS|, ESet)|$.

A pseudo code of the proposed heuristic technique is shown in **Fig. 8**. A parameter P where $0 \leq P \leq 1$ is given to this technique. TerminateSimulation$(i, P, ESet, EVSet)$ is inserted between line 5 and line 6 in Fig. 4. The termination is executed with respect to each set of STEs which have the same injection clock. If $|MESet(i-j, ESet(j))|$ is larger than $N(i-j, ESet(j)) \cdot P$, then $MESet(i-j, ESet(j))$ is considered to be large enough. Then, simulation for STEs in $ESet(j) - MESet(i-j, ESet(j))$ are terminated, and the STEs are judged as failure. The decision $e \in SubESet$ of line 3 in Fig. 5 can be done simply with checking whether the injection clock of e is equal to j at line 3 in Fig. 8.

5. Experiments

5.1 Settings

Experimental results to evaluate the proposed techniques are shown in this section. The following algorithms are implemented on programs using $C++$ program language.

- *BA* (basic) : A fault simulator shown in Section 3.
- *EF* (efficient) : A fault simulator with the proposed exact technique shown in Section 4.1.
- *BA_AP* (approximate *BA*) : A fault simulator shown in Section 3 with the heuristic technique shown in Section 4.2.
- *EF_AP* (approximate *EF*) : A fault simulator with the proposed exact technique shown in Section 4.1 and the heuristic technique shown in Section 4.2.

Event-driven simulation and SPPFP [9] are used in *BA*, *EF*, *BA_AP* and *EF_AP*. The accuracy of *BA_AP* and that of *EF_AP* is evaluated with estimation error on failure rate *FR* which is defined with the following expression.

$$FR(ESet, MESet) = \frac{|ESet - MESet|}{|ESet|}$$

A set of STEs $ESet$ is generated with an analysis of combinational circuits. In the analysis of combinational circuits, an SEF is assumed to occur for every pair of a logic gate and a clock. An STE is generated with respect to each given SEF, simulating the propagation of the SEF effects in the combinational circuit. Only logic masking effects are considered on the simulation, which means that electrical masking effects and temporal masking effects [5] are ignored.

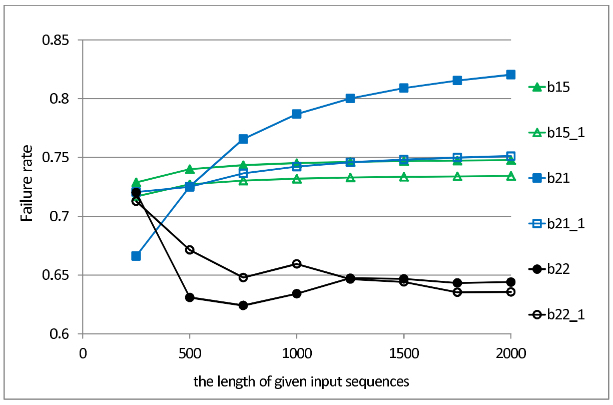


Fig. 9 Failure rate for various length of input sequences.

The benchmark circuits are the 14 largest sequential circuits in ITC'99 benchmark set. The CPU of the computing machine is *Intel Xeon* 2.67 *GHz*, and the memory size is 12 *GB*.

Input sequences with sufficient lengths are generally required to analyze soft error propagation appropriately. If the lengths of input sequences are not sufficient, the results are more likely to be biased by individual input vector. Longer input sequences, however, cause longer run time. **Figure 9** shows failure rates computed with the exact algorithms, *BA* or *EF*, for various length of input sequences. b15, b15_1, b21, b21_1, b22 and b22_1 shown in the figure are the names of several benchmark circuits. Since each curve for the other benchmark circuits has similar form to one of them, the curves for the other circuits are omitted to avoid congestion. The figure shows that the failure rates are nearly stable for input sequences whose length is 1,750 or more. The experiments in this paper employ input sequences for 2,000 clock cycles in order to compute accurate failure rate with short run time.

5.2 Results

Table 1 shows experimental results of *BA* and *EF*, where a randomly generated input sequence for 2,000 clock cycles is given for each circuit. "# of FF vectors" shows the number of FF vectors which have been simulated. It corresponds to $|FVSet|$ in line 7 of Fig. 2 or line 6 of Fig. 4. The results show that *EF* runs about 30–143 times faster than *BA*. The overhead of employing hash table in *EF* accounts for fifth part of total run time at the most. The proposed algorithm significantly reduces the number of FF vectors using hash tables, which causes drastic speed-up. While *BA* could not finish each simulation of *b*17, *b*17_1, *b*18, *b*18_1, *b*19 and *b*19_1 in 260,000 seconds, *EF* finished each simulation in only 16,607 seconds or less. *EF* runs 55.8 times faster than *BA* on average. These results show that the proposed exact technique accelerates fault simulation significantly.

Figures 10 and **11** shows the run-time of *EF* for various number of clock cycles, where the input sequences are randomly gen-

Table 1 Experimental results of *BA* and *EF*.

Circuit	#LG	# of FF vectors		Run time (sec.)			
		BA	EF	BA	EF		BA/EF
				Total	Total	Overhead	
b15	11,419	3,750,879,446	9,320,005	27,023	387	22	69.8
b15_1	17,107	394,072,3711	11,323,733	24,574	417	25	58.9
b20	26,754	188,862,817	2,070,397	1,239	42	5	29.8
b20_1	18,567	792,964,707	2,363,589	5,771	104	18	55.5
b21	27,246	1,976,928,828	4,942,362	15,110	257	46	58.8
b21_1	18,679	1,748,519,100	6,759,436	12,534	229	42	54.7
b22	39,679	363,413,902	3,091,915	3,662	82	10	44.6
b22_1	28,136	202,093,396	2,864,929	1,842	58	7	31.8
b17	49,489	n/a	36,268,185	>260,000	2,328	431	>111.7
b17_1	51,795	n/a	17,201,835	>260,000	1,815	364	>143.2
b18	95,029	n/a	38,925,238	>260,000	5,588	1,122	>46.5
b18_1	89,681	n/a	46,462,600	>260,000	6,025	1,204	>43.2
b19	191,775	n/a	77,639,758	>260,000	14,906	2,756	>17.4
b19_1	181,592	n/a	88,332,371	>260,000	16,607	2,974	>15.7
Mean							55.8

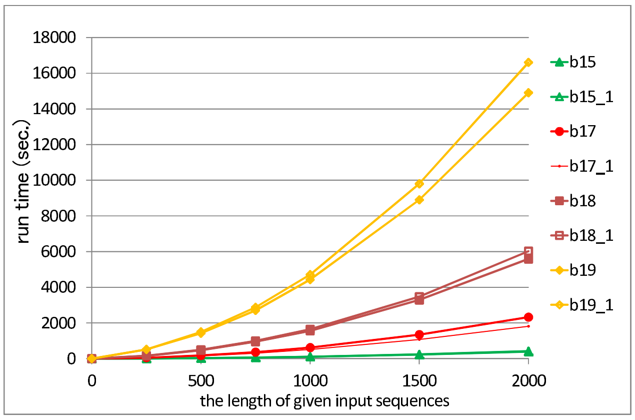


Fig. 10 Run-time of *EF* for various lengths of input sequence with large circuits.

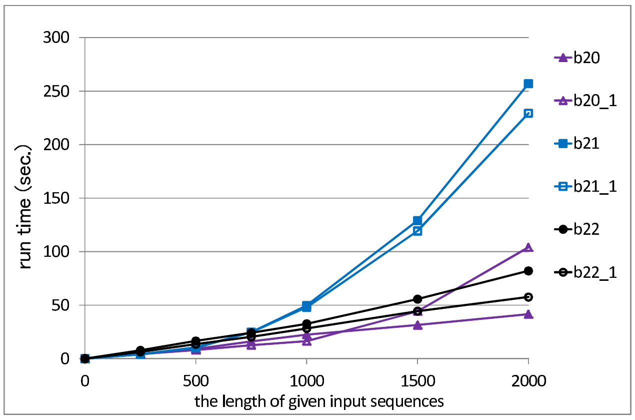


Fig. 11 Run-time of *EF* for various lengths of input sequence with small circuits.

erated. These results show that the run-times for *b*20, *b*20_1, *b*21, *b*21_1, *b*22 and *b*22_1 increase slightly with the grouth of lengths of input sequences. They are less than 300 seconds even for 2,000 clock cycles. On the other hand, the run-times of *b*15, *b*15_1, *b*17, *b*17_1, *b*18, *b*18_1, *b*19 and *b*19_1 increase rapidly with the growth of lengths of input sequences. This results show that there are circuits where further speed-up may be needed, such as *b*15, *b*15_1, *b*17, *b*17_1, *b*18, *b*18_1, *b*19 and *b*19_1.

Figures 12 and **13** shows estimation error and speed-up of *EF_AP* for various values of *P*, where an input sequence for 2,000 clock cycles is randomly generated for each circuit. Let

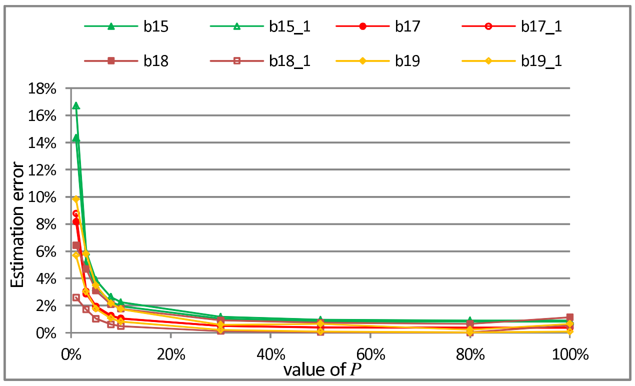


Fig. 12 Estimation error of *EF_AP* for various values of *P*.

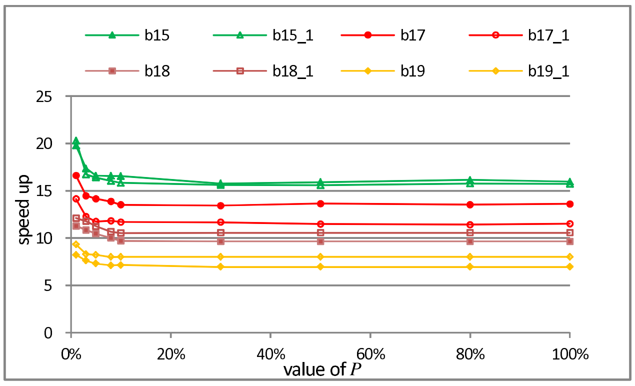


Fig. 13 speed-up of *EF_AP* for various values of *P*.

FR_{EF_AP} and FR_{EX} denote *FR* estimated by *EF_AP* and the exact *FR*, respectively. Estimation error in the figure is computed with $(FR_{EF\_AP} - FR_{EX})/FR_{EX}$. Speed-up in the figure is the run-time of *EF* divided by the run-time of *EF_AP*. This results show that estimation error ranges from 0% to 17%. Speed-up ranges from 7 to 20. Employing 1–5% for *P* makes drastic speed-up, however, it might cause unignorable estimation error. On the other hand, 10% or larger values for *P* keeps considerable speed-up with only a few percent of estimation error. This results show that 10% for *P* seems to be a good value for the benchmark circuits. While the value is reasonable for all the benchmark circuits in this experiments, the good value of *P* may vary on a given circuit or given input sequences. Employing a preprocessing, for example *EF_AP* for a limited number of randomly sampled errors with

Table 2 Experimental results of *BA_AP* and *EF_AP* with $P = 10\%$.

Circuit	Failure rate					# of FF vectors		Run time (sec.)				
	Exact	*BA_AP*	*EF_AP*	$\frac{BA\_AP-Exact}{Exact}$	$\frac{EF\_AP-Exact}{Exact}$	*BA_AP*	*EF_AP*	*BA_AP*	*EF_AP*	$\frac{BA}{BA\_AP}$	$\frac{BA}{EF\_AP}$	$\frac{EF}{EF\_AP}$
b15	0.748	0.762	0.762	2.0%	2.0%	61,381,526	1,999,788	162	24	166.5	1116.4	16.0
b15_1	0.734	0.751	0.751	2.2%	2.2%	64,379,977	2,028,259	175	27	140.3	922.2	15.7
b17	0.815	0.823	0.823	1.0%	1.0%	217,551,541	5,548,152	2,205	158	>118	>1645	14.7
b17_1	0.804	0.812	0.812	1.1%	1.1%	201,556,406	5,170,233	1,886	173	>138	>1505	10.5
b18	0.811	0.825	0.825	1.8%	1.8%	269,642,633	9,899,762	5,914	576	>44	>451	9.7
b18_1	0.835	0.840	0.840	0.5%	0.5%	268,239,346	9,961,463	5,787	573	>45	>454	10.5
b19	0.795	0.809	0.809	1.7%	1.7%	750,574,411	20,249,488	34,494	2,085	>8	>125	7.1
b19_1	0.829	0.836	0.836	0.8%	0.8%	768,613,671	20,241,906	34,660	2,071	>8	>126	8.0
Mean				1.4%	1.4%					83.3	793.1	11.5

various values of P, may be useful to explore a good value of P.

Table 2 shows experimental results of *BA_AP* and *EF_AP* with $P = 10\%$. EX in the table denotes the exact FR. The run-time of BA and that of EF are omitted since they have already shown in Table 1. The results show that *EF_AP* with $P = 10\%$ runs about 7–17 times faster than EF, and it runs 125–1,645 times faster than BA. *EF_AP* with $P = 10\%$ runs about 12 times faster than EF on average, and it runs more than 793 times faster than BF on average. The estimation error on FR ranges only 0.5–2.2% for all the circuits. This results show that the proposed heuristic technique can accelerate run-time of a fault simulation with the exact technique furthermore with slight loss of accuracy. Since the analysis with the heuristic technique is guaranteed to be pessimistic, the heuristic technique is useful to judge whether required soft error tolerance have been achieved or not.

6. Conclusions

This paper presented two acceleration techniques of fault simulation for analyzing soft error propagation. One is an exact technique and the other is a heuristic technique. Experimental results show that applying the exact technique makes a fault simulator with event-driven simulation and SPPFP about 30–143 times faster. Furthermore, applying the heuristic technique makes a fault simulator with the exact technique about 7–17 times faster with only 0.5–2.2% estimation error on failure rate for all the circuits. Since the analysis with the heuristic technique is slightly pessimistic, the heuristic technique is useful to judge whether required soft error tolerance have been achieved or not quickly.

Acknowledgments This work has been partially supported by CREST-DVLSI of JST.

References

[1] von Neuman, J.: Probablistic Logics and Synthesis of Reliable Organisms from Unreliable Components, *Automatic Studies*, Shannon, C. and McCarthy, J. (Eds.), pp.43–98, Princeton University Press (1956).
[2] Cheng, W.-T. and Yu, M.-L.: Differential Fault Simulation — a Fast Method using Minimal Memory, *Proc. 26th Design Automation Conference*, pp.424–428 (1989).
[3] Alexandrescu, D. and Costenaro, E.: Towards Optimized Functional Evaluation of SEE-Induced Failures in Complex Designs, *Proc. 18th IEEE International On-Line Testing Symposium*, pp.182–187 (2012).
[4] Yoshimura, M., Akamine, Y. and Matsunaga, Y.: A Soft Error Tolerance Estimation Method for Sequential Circuits, *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp.268–276 (2011).
[5] Miskov-Zivanov, N. and Marculescu, D.: MARS-C: Modeling and Reduction of Soft Errors in Combinational Circuits, *Proc. 43rd ACM/IEEE Design Automation Conference*, pp.767–772 (2006).
[6] Miskov-Zivanov, N. and Marculescu, D.: MARS-S: Modeling and Reduction of Soft Errors in Sequential Circuits, *Proc. 8th International Symposium on Quality Electronic Design*, pp.893–898 (2007).
[7] Miskov-Zivanov, N. and Marculescu, D.: Soft Error Rate Analysis for Sequential Circuits, *Proc. Conference on Design, Automation and Test in Europe*, pp.1436–1441 (2007).
[8] Mohanram, K. and Touba, N.: Partial Error Masking to Reduce Soft Error Failure Rate in Logic Circuits, *Proc. 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pp.433–440 (2003).
[9] Waicukauski, J.: Fault Simulation of Structured VLSI, *VLSI Systems Design* (1985).

Taiga Takata received his B.E., M.E. and Ph.D. degrees from Kyushu University, Fukuoka, Japan, in 2005, 2007 and 2010, respectively. He joined the Department of Advanced Information Technology in Kyushu University, Japan, in 2010, and had been involved in research of logic synthesis, technology mapping and reliability-aware design methodology for VLSI. He currently works for Cadence Design Systems, Japan.

Masayoshi Yoshimura received his B.E. and M.E. degrees from Osaka University, Osaka, Japan in 1996 and 1998, respectively. He joined Matsushita Electrical Industrial Corporation in Osaka, Japan, in 1998 and began working in computer-aided design. He received his Ph.D. from Osaka University in 2003. He became an assistant professor at Kyushu University, Fukuoka, Japan in 2007. His current research interests include reliability-aware design methodology, design for testability, automatic test pattern generation, secure-aware design methodology. He is a member of IPSJ, IEEE and IEICE.

Yusuke Matsunaga received his B.E., M.E. and Ph.D. degrees in Electronics and Communications Engineering from Waseda University, Tokyo, Japan, in 1985, 1987 and 1997, respectively. He joined Fujitsu Laboratories in Kawasaki, Japan, in 1987 and he has been involved in research and development of the CAD for digital systems. From October 1991 to November 1992, he has been a visiting Industrial Fellow at the University of California, Berkeley, in the Department of Electrical Engineering and Computer Sciences. In 2001, he joined the faculty at Kyushu University. He is currently an associate professor of the Department of Computer Science and Communication Engineering. His research interest includes logic synthesis, formal verification, high-level synthesis and automatic test patterns generation. He is a member of IEICE, IEEE, ACM and IPSJ.

(Recommended by Associate Editor: *Tetsuya Iizuka*)