IPSJ Transactions on System LSI Design Methodology Vol.5 55–62 (Feb. 2012)

[DOI: 10.2197/ipsjtsldm.5.55]

Regular Paper

A Robust Algorithm for Pessimistic Analysis of Logic Masking Effects in Combinational Circuits

Taiga Takata[1,a)] Yusuke Matsunaga[1]

Received: May 27, 2011, Revised: September 2, 2011,
Accepted: October 19, 2011, Released: February 21, 2012

Abstract: Analyzing logic masking effects is an important key to evaluate soft error tolerance of circuits. The computing complexity of analyzing logic masking effects exactly is proportional to the square of circuit size, which is unacceptable to achieve a scalable analyzer. This paper shows a robust algorithm to analyze logic masking effects pessimistically with multiple CODCs (Compatible combinations of Observability Don't Cares). It is guaranteed that an upper bound of the susceptibility of each gate is estimated using the proposed algorithm. The computing complexity of the proposed algorithm is proportional to circuit size. Experimental results show that the proposed algorithm runs about 91 times faster than an algorithm which analyzes logic masking effects exactly with fault simulation. The proposed algorithm estimates average susceptibility 11.5% larger than that of the exact algorithm for circuits in ITC'99 benchmark set. The state-of-the-art heuristic *AnSER* estimates average susceptibility with 96% underestimation for circuits protected with partial TMR (Triple Modular Redundancy) on average, which can be fatal error for soft error tolerance evaluation. On the other hand, the proposed algorithm estimates average susceptibility with 37.9% overestimation for such circuits on average. The proposed algorithm is useful to estimate an upper bound of the susceptibility of each gate quickly.

Keywords: soft error, logic masking, don't care

1. Introduction

Soft errors are transient errors which are temporary inversions of the output values of logic gates or incorrect bit flips of the values held in flip-flops (FFs), caused by collisions of neutrons or alpha particles with silicon atoms in circuits. As transistor size shrinks, a particle strike could be more likely to cause a soft error. Soft errors can cause failures which mean incorrect values observed at primary outputs of circuits. Thus, soft errors are becoming an important issue for dependable circuit designs.

Soft error tolerance of circuits generally vary according to individual design. Thus, designers are needed to evaluate soft error tolerance of circuits quantitatively to know whether required reliabilities have been achieved or not at design phases. Generally, not all soft errors cause failures. It has been known that many soft errors can be masked and do not lead to failures. Thus, susceptibility of each gate, which means probability that an error occurring at the gate causes one or more failures, is an important information to evaluate soft error tolerance of a circuit. Furthermore, the susceptibility of each gate can be useful information to guide some techniques to mitigate failures with small overhead of delay, area or power. The susceptibility of each gate depends on the following three factors [6].

- Logic masking
  An error propagated at an input of a gate may not be propagated to the output of the gate due to the logic function of the gate. For example, if an input value of an AND gate is 0, an error at another input does not affect the output value of the gate.
- Electrical masking
  If an error occurring at an input of a gate is a short inversion of the value, it may be masked due to the electrical property of the gate.
- Latching-window masking
  If an error which is propagated to the input of an FF does not satisfy the setup and hold time conditions, it may be masked without being latched in the FF.

Analyzing the effects of each masking factor is the important key to estimate the susceptibility of each gate accurately. This paper focuses on the analysis of logic masking effects in combinational circuits. Analyzing logic masking effects and ignoring electrical masking and latching-window masking, an upper bound for the susceptibility of each gate can be estimated.

Whether an error is masked with logic masking effects or not depends on not only the error location and the structure of the combinational circuit but also the output value of each gate [*1]. The output value of each gate depends on an input vector. Since

[1] Department of Computer Science and Communication Engineering, Kyushu University, Fukuoka 819–0395, Japan
[a)] taiga@soc.ait.kyushu-u.ac.jp

This work is partially based on the presentation in the 17th IEEE International On-Line Testing Symposium, Athens, Greece, July 2011.

[*1] Whether an error is masked with logic masking effects or not does not depend on the duration of the error. On the other hand, the duration of an error affects electrical masking and latching-window masking. If an error is a long inversion of a value, it is hard to be masked due to the electrical property of a gate, and it tends to be latched into an FF due to satisfying the setup and hold time conditions.

the number of all the input vectors increases exponentially with the number of primary inputs, it is difficult to check whether an error is logically masked or not for each of all the input vectors in large-scale circuits. Several methods to evaluate soft error tolerance of circuits [6], [11] employ Binary Decision Diagrams (BDDs) to represent functions whose domain is the set of all the input vectors efficiently. The exponential complexity, however, cannot be avoided with employing BDDs in the worst case. In this paper, a set of sampled input vectors is assumed to be given. The susceptibility of each gate is approximated with the ratio of the sampled input vectors where errors occurring at the gate are not masked with logic masking effects.

Observability don't care (ODC) is a useful idea for the analysis of logic masking effects, while it is often employed for logic optimization in combinational circuits. ODC of a gate is an input vector where the output of the gate does not affect any value of primary outputs. The maximum set of ODCs (MODC) of a gate is the set of all the ODCs of the gate in the set of sampled input vectors. Using the MODC of a gate, the susceptibility of the gate is estimated as the ratio of the input vectors which is not included in the MODC. The computing complexity of the MODCs for all the gates, however, is proportional to the square of circuit size, which is unacceptable to achieve a scalable analyzer.

AnSER [2] is a method to analyze logic masking effects in combinational circuits using approximate MODCs employed in Ref. [8]. The computing complexity of *AnSER* is proportional to circuit size. While an approximate MODC computed with *AnSER* may fail to include some ODCs, it may include some not-ODC input vectors. Thus, the size of an approximate MODC may larger than the MODC, which may result in optimistic analysis of susceptibility. *AnSER* tends to underestimate susceptibility especially for circuits partially protected with spatial redundancy, which seems to be not suitable for dependable circuit designs. On the other hand, a compatible combination of ODC sets (CODC) [1], [9] can be used to estimate the susceptibility of each gate pessimistically. A CODC is a combination of ODC sets for all the gates, where any value of primary outputs is not affected even if the output values of multiple gates are inverted simultaneously for an input vector commonly included in the ODC sets of the gates. The ODC set of each gate in a CODC is a subset of the MODC. Thus, the analysis for the susceptibility of each gate is guaranteed to be pessimistic for any circuit. The computing complexity of a CODC is proportional to circuit size. The ODC set of each gate in a CODC depends on given priority order of the inputs of each gate. The ODC set of a gate in one CODC is often too small to be used as an approximate MODC, which results in too pessimistic analysis.

This paper presents a robust algorithm to estimate the susceptibility of each gate using an approximate MODC based on multiple CODCs. At first, a constant number of different CODCs are calculated with variant priority orders of the inputs of each gate. Then, the union of the ODC sets in the CODCs is calculated for each gate. The union is considered as an approximate MODC of the gate. Since the ODC set for a gate in a CODC is a subset of the MODC, the union is also a subset of the MODC. Thus, the proposed algorithm is guaranteed to estimate the susceptibility of each gate pessimistically for any circuit. During a CODC calculation, the algorithm employs a heuristic technique to maximize the number of ODCs. The technique is to decide priority order of the inputs of each gate with considering the CODCs already calculated. The computing complexity of the algorithm is proportional to circuit size. Experimental results show that the proposed algorithm runs about 91 times faster than an algorithm which estimates the susceptibility of each gate using the exact MODCs. The average susceptibility computed with the proposed algorithm is about 11.5% larger than that computed with the exact MODCs for original circuits in ITC'99 benchmark set. Furthermore, the proposed algorithm estimates average susceptibility 37.9% larger than that for circuits protected with partial TMR, while *AnSER* estimates average susceptibility 96% smaller than that on average. The proposed algorithm is reasonable to estimate an upper bound of the susceptibility of each gate quickly, while *AnSER* may underestimate it significantly.

The rest of this paper is organized as the following. Section 2 describes several preliminaries. Section 3 shows the proposed algorithm. Section 4 presents a short description about an algorithm to compute the exact MODCs and *AnSER*. Experimental results are shown in Section 5. Section 6 concludes this paper.

2. Preliminaries

2.1 Boolean Network

In this section, Boolean network is defined to model combinational circuits.

Boolean network is DAG (Directed Acyclic Graph). Each node in a Boolean network represents a Boolean function. Let G denote a Boolean network. Then, $V(G)$ and $E(G)$ denote the set of all the nodes and the set of all the edges, respectively. 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 **fanout** of a node v is an immediate successor of v. The set of all the fanouts of v is defined by $FO(v) = \{w \mid \exists(v, w) \in E\}$. A node v with $FI(v) = \phi$ is called a **primary input**. One or more nodes are labelled as **primary output**. $PI(G)$ and $PO(G)$ denotes the set of all the primary inputs and the set of all the primary outputs of G, respectively. If the length of a bit vector i is $|PI(G)|$ and if a bijection function from bits in i to all the primary inputs is defined, then i is called an **input vector** of G.

The Boolean function of node $v \in V(G)$ itself is called **local function** of v, and denoted by $f_v : B^{|FI(v)|} \rightarrow B$. On the other hand, the Boolean function from input vectors to the output values of v is called **global function** of v, and denoted by $F_v : B^{|PI(G)|} \rightarrow B$. The global function of a primary input is a Boolean variable which represents the value of the primary input itself. Otherwise, $F_v \equiv f_v(F_{v_0}, F_{v_1}, ..., F_{v_n})$, where $FI(v)$ is represented by $\{v_0, v_1, ..., v_n\}$.

2.2 Maximum Set of Observability Don't Cares (MODC)

If the value at the output of a node v does not affect any value of primary outputs for an input vector i, then i is called an **observability don't care (ODC)** of v. More exact definition of ODC is the following.

Replacing a node $v \in V(G)$ with another node v' denotes an op-

eration which removes edge (v, w) from $E(G)$ for each $w \in FO(v)$ and adds edge (v', w) to $E(G)$. Then, the global function of each $w \in FO(v)$ changes from $f_w(..., F_v, ...)$ to $f_w(..., F_{v'}, ...)$. Please notice that the global function of each node after replacing does not depend on the local function of v'. Let G' denote a Boolean network which is obtained with replacing node $v \in V(G)$ with node v', and let $po : PO(G) \rightarrow PO(G')$ denote a bijection function which represents correspondences of primary outputs between G and G'. If the condition $\forall o \in PO(G), F_o \equiv F_{po(o)}$ is met, then the global function $F_{v'}$ is called a **permissible function** of v. If and only if the following function F' is a permissible function of v, an input vector i is an ODC of v.

$$F'(x) = \begin{cases} \overline{F_v(x)} & \text{if } x = i \\ F_v(x) & \text{otherwise} \end{cases}$$

Figure 1 shows an example of a Boolean network. Let an input vector $abcd$ be 1000. Then, the values of n, m, h, k are $0, 0, 1, 0$, respectively. Even if the value of h is inverted, the value of k stays 0, which indicates that 1000 is an ODC of h. On the other hand, if the value of m is inverted, the value of K changes to 1, which indicates that 1000 is not ODC of m.

Let I denote a given set of input vectors. The **maximum set of observability don't cares (MODC)** of a node v is the set of all the ODCs which are included in I. Then, the **susceptibility** of a node v for a given set of input vectors, which is denoted by $SCP(v, I)$, is defined with the following equation.

$$SCP(v, I) = \frac{|I - MODC(v, I)|}{|I|} \quad (1)$$

$MODC(v, I)$ in the above equation denotes the MODC of v. The MODC of a node v can be calculated with employing fault simulation. The detail of the algorithm is described in Section 4.1.

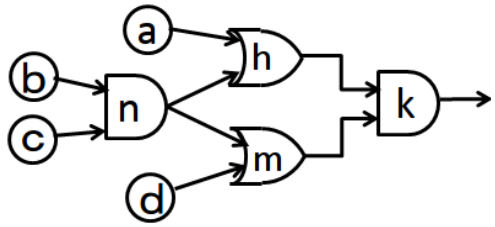

Fig. 1 An example of a Boolean network.

2.3 Compatible Combination of ODC Sets (CODC)

Let $ODCS$ be a combination of ODC sets for all the nodes, and let $ODCs(v)$ be the set of ODCs for a node v contained in the $ODCS$. Let V' be arbitrary subset of $V(G)$ and let i be arbitrary input vector which is commonly included in $ODCs(v)$ for each $v \in V'$. If any values of primary outputs are not affected with simultaneously inverting output values of nodes in V' for i, then $ODCS$ is called **a compatible combination of ODC sets (CODC)** [1], [9]. More exact definition of CODC is the following.

Onset $ON(F)$ for a Boolean function F is the set of input vectors defined with $ON(F) = \{i | F(i) = 1\}$. Let IV be a subset of $ODCs(v)$. Then, the following $PF(v, IV, ODCs(v))$ represents a permissible function of v which is obtained with making the output value of v to be 1 for input vectors in IV and making that to be 0 for the input vectors in $ODCs(v) - IV$.

$$PF(v, IV, ODCs(v))(x) = \begin{cases} 1 & \text{if } x \in (ON(F_v) - ODCs(v)) \cup IV \\ 0 & \text{otherwise} \end{cases}$$

The following $SPF(v, ODCs(v))$ represents a set of all the permissible functions of v which are obtained with making the output value to be 0 or 1 for each $i \in ODCs(v)$.

$$SPF(v, ODCs(v)) = \bigcup_{IV \in Power(ODCs(v))} \{PF(v, IV, ODCs(v))\}$$

$Power(S)$ denotes the power set of S. Then, $ODCS$ which denotes a combination of ODC sets for all the nodes is a CODC if and only if the following condition is met for any set $V' \subseteq V(G)$.

- Replacing each node $v \in V'$ with a node v' whose global function is selected from $SPF(v, ODCs(v))$ arbitrarily. Please notice that the global function of each node $w \notin V'$ does not depends on the local function of v'. Let G' be a Boolean network obtained with the above multiple replacing, and let $po : PO(G) \rightarrow PO(G')$ be a bijection function which represents correspondences of primary outputs between G and G'. Then, the condition $\forall o \in PO(G), F_o \equiv F_{po(o)}$ is met.

Multiple CODCs can exist for a Boolean network G and a given set of input vectors I. $ODCs(v)$ for a node v in a CODC is a subset of the MODC of v [9].

Table 1 shows that the MODCs and an example of CODC for

Table 1 MODCs and an example of CODC for Fig. 1.

	h			m			n		
abcd	F_h	MODC	CODC	F_m	MODC	CODC	F_n	MODC	CODC
0000	0	1	1	0	1	0	0	0	0
0001	0	0	0	1	1	1	0	0	0
0010	0	1	1	0	1	0	0	0	0
0011	0	0	0	1	1	1	0	0	0
0100	0	1	1	0	1	0	0	0	0
0101	0	0	0	1	1	1	0	0	0
0110	1	0	0	1	0	0	1	0	0
0111	1	0	0	1	0	0	1	1	1
1000	1	1	1	0	0	0	0	0	0
1001	1	0	0	1	0	0	0	1	1
1010	1	1	1	0	0	0	0	0	0
1011	1	0	0	1	0	0	0	1	1
1100	1	1	1	0	0	0	0	0	0
1101	1	0	0	1	0	0	0	1	1
1110	1	0	0	1	0	0	1	0	0
1111	1	0	0	1	0	0	1	1	1

h, m, n in Fig. 1. "MODC" in Table 1 is 1 if and only if an input vector is included in the MODC. Similarly, "CODC" in Table 1 is 1 if and only if an input vector is included in the set of ODCs in the CODC. Since there is no ODC of k, it is omitted. Input vector 0000 and 0100 is included in the MODC of h and the MODC of m. If the output values of h and m are inverted simultaneously, the output value of k is also inverted. Thus, in any CODC, 0000 and 0100 cannot be included in both the ODC set of h and that of m. 0000 and 0100 are not included in $ODCs(m)$ in this example.

Saboj proposed a method to calculate a CODC [9]. Let $CODC(v, i)$ be a Boolean function that returns 1 if and only if an input vector i is included in $ODCs(v)$ in a CODC. Furthermore, ODC is also defined for each edge in paper [9]. Let $CODC((v, w), i)$ denotes a Boolean function which returns 1 if and only if i is included in the ODC set of edge (v, w) in a CODC. In topological order from primary outputs to primary inputs, for each node v and each edge (w, x), $CODC(v, i)$ and $CODC((w, x), i)$ is calculated. For any primary output o, $CODC(o, i) = 0$. $CODC(v, i)$ is calculated with the following expression for $v \notin PO(G)$ and $i \in I$.

$$CODC(v, i) = \prod_{w \in FO(v)} CODC((v, w), i) \quad (2)$$

The detailed notation of $CODC((v, w), i)$ is shown in Ref. [9]. In this paper, $CODC((v, w), i)$ is shown where the local function of w is AND. Let $FI(w)$ be $\{v_0, ..., v_p, ..., v_n\}$ and the priority order of nodes in $FI(w)$ be $v_0 \prec ... \prec v_p \prec ... \prec v_n$. If there is a fanin $v_q \in FI(w) - \{v_p\}$ where $F_{v_q}(i) = 0$, then $F_w(i)$ does not depend on the value of F_{v_p}. Even if the above condition is met, if $F_{v_p}(i) = 0$ and if there is no fanin $v_q \in \{v_0, ..., v_{p-1}\}$ where $F_{v_q}(i) = 0$, $F_{v_p}(i)$ must be guaranteed to be 0. Thus, $CODC((v_p, w), i)$ is calculated with the following equation.

$$CODC((v, w), i)$$
$$= CODC(w, i) + (\prod_{v_q \in FI(w) - \{v_p\}} \overline{F_{v_q}(i)})(F_{v_p}(i) + \prod_{q=0}^{p-1} \overline{F_{v_q}(i)}) \quad (3)$$

Any node whose local function is not AND can be converted into a tree constructed with ANDs and inverters. Let G' be a Boolean network obtained with converting all the nodes whose local function is not AND in G into trees constructed with ANDs and inverters. Let $ROOT : V(G) \rightarrow V(G')$ be a function which returns the root of a tree which is generated from a node $v \in V(G)$. Then, a CODC of G is obtained with setting the ODC set of $ROOT(v)$ as the ODC set of v for each $v \in V(G)$, where the ODC set of $ROOT(v)$ is in a CODC computed for G' using Eqs. (2) and (3).

If $|FI(v)|$ is assumed to be constant at most, the computing complexity of $CODC(v, i)$ for all the nodes and the computing complexity of $CODC((w, x), i)$ for all the edges are $O(|V(G)|)$ per an input vector i. Thus, the computing complexity of a CODC is $O(|I||V(G)|)$. As shown in the above, a CODC depends on the priority orders of fanins for each node.

3. A Robust Heuristic to Approximate MODC Based on Multiple CODCs

3.1 Overview

Given a Boolean network G and a set of input vectors I, an algorithm to calculate approximate MODCs for all the nodes is proposed. The framework of the algorithm is the following.

(1) Multiple CODCs are calculated with variant priority orders of fanins of each node.

(2) The union of the ODC sets for all the CODCs is calculated for each v as an approximate MODC of v.

Let $\{CODC^0, CODC^1, ..., CODC^{X-1}\}$ be the set of the calculated CODCs where X CODCs are calculated. Let $ODCs^x(v)$ be the ODC set of v in $CODC^x$. Then, the approximate MODC of v is calculated with the following equation.

$$AppMODC_v = \bigcup_{0 \le x < X} ODCs^x(v) \quad (4)$$

Because $ODCs^x(v)$ is a subset of the MODC of v [9], $AppMODC_v$ is also a subset of the MODC of v. Thus, the approximate susceptibility $AppSCP(v, I)$ shown at the following expression is guaranteed to be larger than $SCP(v, I)$ for any node v in any Boolean network.

$$AppSCP(v, I) = \frac{|I - AppMODC_v|}{|I|} \quad (5)$$

Figure 2 shows a pseudo code of the algorithm to calculate $AppMODC_v$ for all the nodes. The codes from line 8 to line 11 is a process to calculate $CODC(v, i)$ for each node v. If $CODC(v, i)$ is 1, i is added to $AppMODC[v]$.

Since the complexity of computing a CODC for I is $O(|I||V(G)|)$, the complexity of computing X CODCs is also $O(X|I||V(G)|)$.

```
ApproximateMODC(network G, input vector set I, integer X){
1:  for each v ∈ V(G) {
2:    AppMODC[v] ← {};
    }
3:  for each i ∈ I {
4:    for each v ∈ V(G) in topological order from PI to PO {
5:      Computing the output value of v;
      }
6:    for (int x = 0; x != X; ++x){
7:      for each v ∈ V(G) in topological order from PO to PI {
8:        Computing CODC(v, i);
9:        Setting a priority order of fanins of v;
10:       for each w ∈ FI(v) {
11:         Computing CODC((w, v), i);
          }
          // CODC((w, v), i) and CODC(v, i) are
          // described in section 2.3.
12:       if (CODC(v, i) = true) {
13:         AppMODC[v] ← AppMODC[v] ∪ {i};
          }
        }
      }
    }
14: return AppMODC;
}
```

Fig. 2 A pseudo code of the proposed algorithm.

3.2 Decision of Priority Order of Fanins

During a CODC calculation, to maximize the number of ODCs in the CODC, the proposed algorithm employs a heuristic technique to decide a priority order of the fanins of each node. Assuming xth CODC calculation, the priority order of the fanins of

a node v is decided with considering the following information.

- $CODC((w,t),i)$ which has already calculated in the current xth CODC calculation, where w is a fanin of v, and t is another node from v.
- A set of already calculated CODCs $\{CODC^0, CODC^1, ..., CODC^{x-1}\}$.

Let consider to set priority order of $\{(h,k),(m,k)\}$ where the output values of h, m, k are $0,0,0$ for an input vector i in Fig. 1. If an additional node w and an edge (h,w) exist, and if $CODC((h,w),i)$ has already decided to be 0, then i cannot be an ODC of h in the xth CODC. Thus, the priority of (h,k) is set to be lower than the other edge in this case. On the other hand, if i has already included in $AppMODC_h$ during the past CODC calculation, inducing i to be an ODC of h in the xth CODC is likely to make almost nothing. Thus, the priority of (h,k) is also set to be lower than the other edges in this case. For nodes with the same priority under the above conditions, a random order is set to them.

4. Related Works

4.1 An Exact Algorithm to Compute MODCs Using Fault Simulation

This section briefly introduces an exact algorithm to compute the MODC for each of all the nodes. The algorithm employs fault simulation. **Figure 3** shows a pseudo code of a naive algorithm. Both the computing complexity of line 4 and that of line 6 is $O(|V(G)|)$. The codes from line 6 to line 9 are executed for $|I||V(G)|$ times. Thus, the computing complexity of the algorithm is $O(|I||V(G)|^2)$. There is a technique to speed up fault simulation using Fanout Free Region (FFR) [10]. The computing complexity $O(|I||V(G)|^2)$, however, cannot be avoided with employing the technique in the worst case.

4.2 AnSER

AnSER is the state-of-the-art heuristic to analyze logic masking effects with approximate MODCs. The procedure to calculate approximate MODC [8] in *AnSER* is similar to the procedure to calculate CODC [9]. The difference is that *AnSER* sets the highest priority for each of all the fanins of a node, while CODC computation sets a priority order of fanins of a node.

```
ExactMODC(network G, input vector set I){
1:   for each v ∈ V(G) {
2:     MODC[v] ← {};
     }
3:   for each i ∈ I {
4:     Output values for all the nodes are computed;
5:     for each v ∈ V(G) {
6:       For a faulty circuit where v is inverted,
         output values for all the successors of v is computed;
7:       for each o ∈ PO(G) {
8:         if the value of o for the faulty circuit is
             different from that for the original circuit {
9:           MODC[v] ← MODC[v] ∪ {i};
             break;
           }
         }
       }
     }
10:  return MODC;
}
```

Fig. 3 A pseudo code of a naive algorithm to compute the exact MODCs.

Let G and I be a Boolean network G and a given set of input vectors, respectively. Let $ODC_{AnS}(v,i)$ be a Boolean function that is 1 if and only if an input vector i is included in the ODC set of v. For any primary output o, $ODC_{AnS}(o,i) = 0$. $ODC_{AnS}(v,i)$ is calculated for $v \notin PO(G)$ and $i \in I$ by the following equation.

$$ODC_{AnS}(v,i) = \prod_{w \in FO(v)} ODC_{AnS}((v,w),i)$$

$ODC_{AnS}((v,w),i)$ denotes a Boolean function that represents whether i is an ODC of edge (v,w). $ODC_{AnS}((v,w),i)$ is 1 if the inversion of the value of (v,w) does not affect the output value of w. If $ODC_{Ans}(w,i)$ is 1, $ODC_{AnS}((v,w),i)$ is also 1. Otherwise, $ODC_{AnS}((v,w),i)$ is 0. If the local function of node w is AND, $ODC_{AnS}((v,w),i)$ is calculated with the following equation.

$$ODC_{AnS}((v,w),i) = ODC_{AnS}(w,i) + \prod_{q \in FI(w)-\{v\}} \overline{F_q(i)}$$

Please notice that $ODC_{Ans}((v,w),i)$ is different from $CODC((v,w),i)$ in Eq. (3). An example of $ODC_{AnS}((n,h),i)$ in Fig. 1 is shown where i is 1111. Then, all the values of nodes n, h, m, k are 1. Even if the value of (n,h) is inverted from 1 to 0, the output value of h is still 1. Thus, $ODC_{AnS}((n,h),i)$ is 1. On the other hand, $ODC_{Ans}((a,h),i)$ is also 1 in the same way. Both $ODC_{AnS}((a,h),i)$ and $ODC_{AnS}((n,h),i)$ are 1 in this case, while one of $CODC((a,h),i)$ and $CODC((n,h),i)$ must be 1.

AnSER might be significantly inaccurate to analyze the susceptibility of each node for circuits protected with spatial redundancy like partial TMR [7]. Partial TMR is a technique to mitigate soft error vulnerability of circuits with triplicating parts of circuits. At first, an original circuit is parted into two sub-circuits F_1, F_2. Then, F_2 is triplicated. A voter is inserted to calculate the majority vote of the outputs of the copies. **Figure 4** illustrates an example of a network which is protected with partial TMR, where k_0, k_1, k_2 are protected. The values of all the nodes for an input vector $i = 0110$ are 1. $ODC_{AnS}((k_0,voter),i)$, $ODC_{AnS}((k_1,voter),i)$ and $ODC_{AnS}((k_2,voter),i)$ are 1, since the voter masks the inversion of one of the input values. Then, $ODC_{AnS}(k_0,i), ODC_{AnS}(k_1,i)$ and $ODC_{AnS}(k_2,i)$ are 1. Since i is an ODC for each of all the fanouts of h, $ODC_{AnS}(h,i)$ is 1. In a similar way, $ODC_{AnS}(m,i)$ is also 1. Then, $ODC_{AnS}(n,i)$ is 1, since i is judged as an ODC for each of all the fanouts of n. In fact, however, i is not an ODC of n, h and m because an error at n, h or m affects the output of the voter. Thus, an approximate

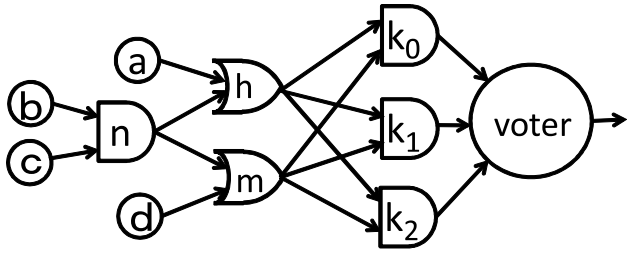

Fig. 4 A simple example of partial TMR.

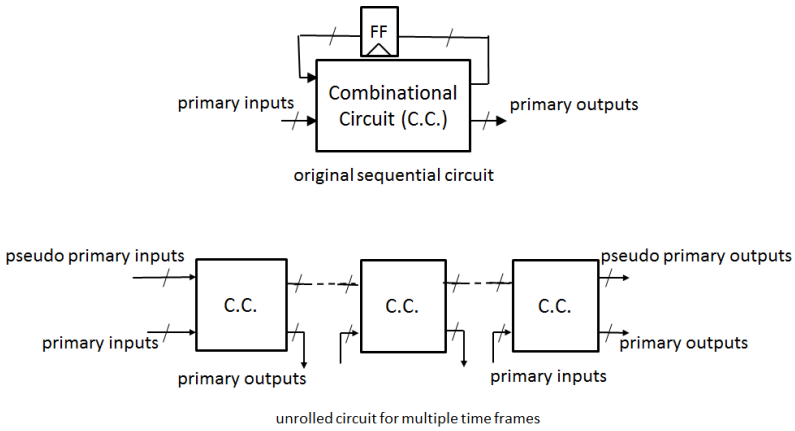

Fig. 5 A sequential circuit and the unrolled circuit for multiple time frames.

MODC of a node v computed with $AnSER$ may smaller than the MODC of v, which may results in underestimation of the susceptibility of v. On the other hand, the proposed algorithm does not make such mistakes because an approximate MODC computed with the proposed algorithm is a subset of MODC. In the above example, two of $CODC((k_0, voter), i)$, $CODC((k_1, voter), i)$ and $CODC((k_2, voter), i)$ must be 0. That is because inverting two or more values of k_0, k_1 and k_2 simultaneously makes value inversion of $voter$, which violates the condition to be a CODC. Then, $CODC(h, i)$, $CODC(m, i)$, and $CODC(n, i)$ is 0.

5. Experiments

5.1 Settings

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

- ex : an exact algorithm to compute MODCs with fault simulation. The speed-up technique [10] is employed.
- ans : $AnSER$.
- $codcs$: the proposed algorithm.

Using the above algorithms, average susceptibility (ASCP) denoted with the following expression is calculated, where C and I denote a circuit and a set of input vectors.

$$ASCP(C, I) = \frac{1}{|V(C)|} \sum_{v \in V(C)} SCP'(v, I)$$

$V(C)$ denotes the set of all the gates in C. $SCP'(v, I)$ for ex is equal to $SCP(v, I)$. $SCP'(v, I)$ for ans and $codcs$ is calculated with the following expression, where $AppMODC(v, I)$ denotes the approximate MODC for v.

$$SCP'(v, I) = \frac{|I - AppMODC(v, I)|}{|I|}$$

19200 input vectors are generated with random sampling to prepare I.

The benchmark circuits are the 14 largest circuits in ITC'99 benchmark set. The number of gates of the benchmark circuits ranges from about 10,000 to 90,000. Another 14 benchmark circuits are generated with protecting the above circuits with partial TMR. 20% gates of all the gates for each original circuit are selected in an order from primary outputs to primary inputs. The selected gates are triplicated, and voters are inserted. All the benchmark circuits are sequential circuits. Each circuit is unrolled for 2 time frames as shown in **Fig. 5** to be converted into a combinational circuit. The FFs immediately before the first time frame are considered as pseudo primary inputs, while the FFs immediately after the last time frame are considered as pseudo primary outputs.

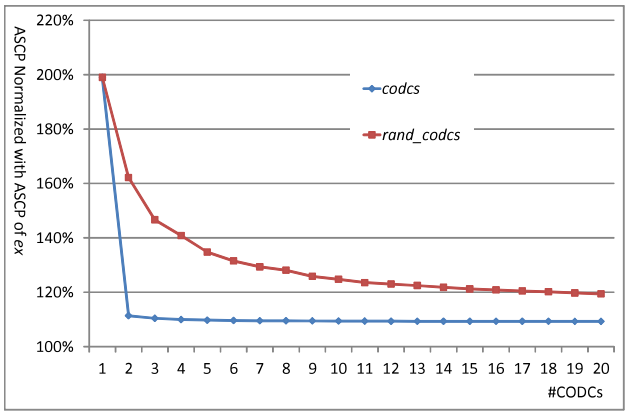

Fig. 6 Experimental results for original $b14$.

The CPU of the computing machine is *Intel Core i7 2.93 GHz*, and the memory size is 6 *GB*.

5.2 Impacts of the Heuristic Technique to Decide Priority Orders

At first, the heuristic technique employed in $codcs$ to decide priority orders of the fanins of each node is evaluated. An algorithm is implemented as a program $rand\_codcs$ which estimates the susceptibility of each gate in a similar way to $codcs$ except that priority orders are completely random.

Figures 6 and **7** show the results for original $b14$ and the results for $b14$ protected with partial TMR, respectively. The horizontal axis shows the number of CODCs. The vertical axis shows ASCP calculated with the algorithms divided by ASCP calculated with ex. Both the run-time of $rand\_codcs$ and that of $codcs$ are proportional to the number of CODCs. If the number of CODCs is the same, $rand\_codcs$ is tend to be slightly faster. This results show that $codcs$ can estimate ASCP of $b14$ more accurate than $rand\_codcs$ for each number of CODCs. The similar results are

Table 2 Experimental results for original circuits.

Circuit	#Node	ASCP					Run-time				
		ex	ans	codcs	ans/ex	codcs/ex	ex (sec.)	ans (sec.)	codcs (sec.)	ans/ex	codcs/ex
b14	13320	0.108	0.108	0.118	100.2%	109.4%	255.4	0.7	6.1	0.3%	2.4%
b14_1	8977	0.149	0.150	0.165	100.1%	110.1%	136.7	0.4	3.6	0.3%	2.6%
b15	11419	0.236	0.238	0.270	100.6%	114.4%	304.2	0.6	5.7	0.2%	1.9%
b15_1	17107	0.171	0.170	0.198	99.4%	115.7%	484.5	0.9	8.2	0.2%	1.7%
b17	49489	0.181	0.182	0.218	100.4%	120.5%	5219.9	3.1	27.4	0.1%	0.5%
b17_1	51795	0.170	0.170	0.194	99.9%	114.3%	5662.9	3.4	29.9	0.1%	0.5%
b18	92281	0.183	0.182	0.198	99.4%	108.2%	47473.1	9.6	84.5	0.0%	0.2%
b18_1	86937	0.193	0.192	0.209	99.3%	108.1%	42312.7	8.7	76.8	0.0%	0.2%
b20	26754	0.121	0.120	0.132	99.7%	109.8%	1445.6	1.7	15.0	0.1%	1.0%
b20_1	18567	0.165	0.164	0.182	99.7%	110.6%	778.4	1.1	9.2	0.1%	1.2%
b21	27246	0.121	0.120	0.132	99.7%	109.8%	1438.4	1.6	14.4	0.1%	1.0%
b21_1	18679	0.167	0.167	0.185	99.8%	110.8%	759.3	1.0	9.4	0.1%	1.2%
b22	39679	0.123	0.123	0.134	99.9%	109.2%	3078.0	2.4	21.8	0.1%	0.7%
b22_1	28136	0.164	0.164	0.181	99.7%	110.4%	2053.0	1.8	15.9	0.1%	0.8%
Average					99.8%	111.5%				0.1%	1.1%

Table 3 Experimental results for protected with partial TMR.

Circuit	#Node	ASCP					Run-time				
		ex	ans	codcs	ans/ex	codcs/ex	ex (sec.)	ans (sec.)	codcs (sec.)	ans/ex	codcs/ex
b14	18946	0.0162	0.0030	0.0224	18.7%	137.8%	708.9	1.1	92.2	0.2%	13.0%
b14_1	12866	0.0357	0.0042	0.0403	11.7%	112.9%	334.1	0.7	59.3	0.2%	17.7%
b15	16504	0.1009	0.0042	0.1151	4.2%	114.1%	1170.4	1.1	90.7	0.1%	7.8%
b15_1	24468	0.0548	0.0029	0.0628	5.2%	114.7%	1635.1	1.5	123.7	0.1%	7.6%
b17	70793	0.0653	0.0014	0.0922	2.1%	141.1%	15464.2	4.8	390.2	0.0%	2.5%
b17_1	74024	0.0537	0.0013	0.0615	2.4%	114.6%	15691.2	5.3	426.2	0.0%	2.7%
b18	132531	0.0556	0.0002	0.0679	0.4%	122.1%	112794.0	14.6	1171.8	0.0%	1.0%
b18_1	125049	0.0625	0.0002	0.0756	0.3%	121.0%	100874.0	13.3	1082.5	0.0%	1.1%
b20	37965	0.0309	0.0007	0.0532	2.2%	172.1%	3186.4	2.6	214.2	0.1%	6.7%
b20_1	26505	0.0524	0.0008	0.0640	1.6%	122.2%	1971.2	1.6	142.1	0.1%	7.2%
b21	38655	0.0289	0.0007	0.0526	2.3%	181.8%	3009.6	2.5	203.8	0.1%	6.8%
b21_1	26661	0.0500	0.0008	0.0659	1.6%	131.7%	1853.2	1.7	140.8	0.1%	7.6%
b22	56305	0.0302	0.0004	0.0601	1.5%	198.8%	6698.9	3.7	310.3	0.1%	4.6%
b22_1	40147	0.0487	0.0005	0.0709	1.1%	145.6%	4635.2	2.8	235.4	0.1%	5.1%
Average					4.0%	137.9%				0.1%	6.5%

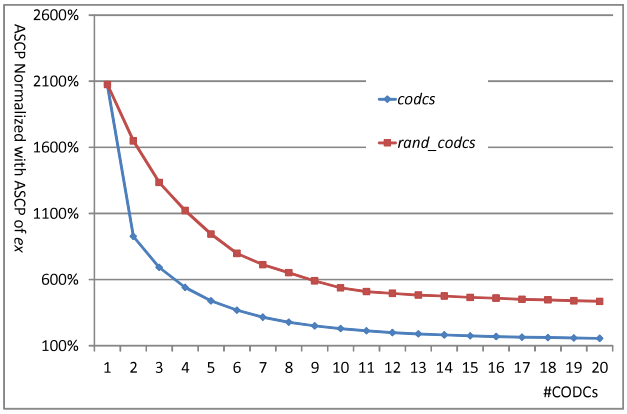

Fig. 7 Experimental results for $b14$ protected with partial TMR.

shown for each of all the other circuits. This results show that the heuristic technique is effective to estimate susceptibility accurately.

If the number of CODCs is small, adding a CODC leads to large improvement in accuracy of ASCP for both algorithms. On the other hand, if the number of CODCs is large, adding more CODCs makes just a little improvement in accuracy.

5.3 Comparisons of the Proposed Algorithm and *AnSER*

Table 2 shows experimental results for original circuits, where 10 CODCs are employed. *ans* runs significantly faster than *ex* for each of all the circuits. *ans* estimates ASCP with little error for the original circuits. On the other hand, *codcs* overestimates ASCP for all the circuits. *codcs* estimates ASCP about 11.5% larger than *ex* on average. The run-time of *codcs* ranges from 0.2% to 2.6% for the run-time of *ex*. *codc* runs about 91 times faster than *ex* on average.

Table 3 shows experimental results for circuits protected with partial TMR, where 100 CODCs are employed. *ans* estimates ASCP about 96% smaller than *ex* on average, which may be fatal error to evaluate soft error tolerance. On the other hand, *codcs* overestimates ASCP for all the circuits. *codcs* estimates ASCP about 37.9% larger than *ex* on average. The run-time of *codcs* ranges from 1.0% to 17.7% of the run-time of *ex*. *codcs* runs 15 times faster than *ex* on average. *codcs* is reasonable to estimate an upper bound of the susceptibility of each node quickly, while *ans* may underestimate it significantly.

6. Conclusions

This paper presents a robust algorithm to analyze logic masking effects pessimistically using multiple CODCs. Logic masking analysis with the proposed algorithm is helpful to judge whether required reliability has been achieved or not, since the logic masking analysis is guaranteed to be pessimistic, and it runs in reasonable time. Experimental results show that the proposed algorithm runs about 91 times faster than the algorithm which analyzes logic masking effects exactly with employing fault simulation. The proposed algorithm estimates average susceptibility about 11.5% larger than that estimates with the exact algorithm. For circuits protected with partial TMR, the proposed algorithm estimates av-

erage susceptibility with 37.9% overestimation, while *AnS ER* estimates it with 96% underestimation on average.

One of the future works is to study SAT (SATisfiability problem)-based approach to analyze logic masking effects. It is necessary to compare the proposed approach with SAT-based approaches to generate ODCs [3], [4], [5]. The future works also include evaluating soft error rate with considering not only logic masking effects but also electrical masking effects and temporary masking effects.

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

Reference

[1] Brayton, R.: Compatible Observability Don't Cares Revisited, *Proc. 2001 IEEE/ACM International Conference on Computer-aided design*, pp.618–623 (2001).
[2] Krishnaswamy, S., Plaza, S., Markov, I. and Hayes, J.: Signature-based SER Analysis and Design of Logic Circuits, *IEEE Trans. CAD of Integrated Circuits and Systems*, Vol.28, pp.74–86 (2009).
[3] Mishchenko, A. and Brayton, R.: SAT-Based Complete Don't-care Computation for Network Optimization, *Proc. Conference on Design, Automation and Test in Europe*, pp.412–417 (2005).
[4] Mishchenko, A., Brayton, R., Jiang, J.-H.R. and Jang, S.: Scalable Don't-care-based Logic Optimization and Resynthesis, *Proc. ACM/SIGDA International Symposium on FPGAs*, pp.151–160 (2009).
[5] Mishchenko, A., Zhang, J., Sinha, S., Burch, J., Brayton, R. and Chrzanowska-Jeske, M.: Using Simulation and Satisfiability to Compute Flexibilities in Boolean Networks, *IEEE Trans. CAD of Integrated Circuits and Systems*, Vol.25, No.5, pp.743–755 (2006).
[6] 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).
[7] 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).
[8] Plaza, S., hui Chang, K., Markov, I. and Bertacco, V.: Node Mergers in the Presence of Don't Cares, *Proc. Asia and South Pacific Design Automation Conference* (2007).
[9] Savoj, H. and Brayton, R.: The Use of Observability and External Don't Cares for the Simplification of Multi-level Networks, *Proc. 27th ACM/IEEE Design Automation Conference*, pp.297–301 (1990).
[10] Waicukauski, J.: Fault Simulation of Structured VLSI, *VLSI Systems Design* (1985).
[11] Zhang, B., Wang, W.-S. and Orshansky, M.: FASER: Fast Analysis of Soft Error Susceptibility for Cell-Based Designs, *Proc. 7th International Symposium on Quality Electronic Design*, pp.755–760 (2006).

Taiga Takata received his B.E., M.E. and Ph.D. degrees from Kyushu University, Fukuoka, Japan, in 2005, 2007 and 2010, respectively. He is currently a research fellow at the Department of Advanced Information Technology in Kyushu University. His current research interests include logic synthesis, technology mapping and formal verification for VLSI designs. He is a member of ACM and IPSJ.

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: *Toshinori Hosokawa*)