社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－136（10）
2008 / 10 / 7

組み合わせ論理回路におけるソフトエラーの論理マスク効果の正確な見積り手法について

松永　裕介†

† 九州大学大学院システム情報科学研究院
〒 819–0395 福岡市西区元岡 744
E-mail: †matsunaga@c.csce.kyushu-u.ac.jp

あらまし　中性子線などの影響で組み合わせ回路中の論理値に一時的な誤りが発生することをソフトエラーと呼ぶ．組み合わせ回路内部のソフトエラーはそのすべてが外部出力まで到達するわけではなく，いくつかの要因でマスクされ回路の動作に影響を与えない場合もある．本稿ではそのようなマスク要因の一つである論理マスクの影響を正確に見積もる手法について述べる．このアルゴリズムは他の故障に関する計算結果を再利用することで処理の高速化を図っている．

キーワード　ソフトエラー, 組み合わせ論理回路, 論理マスク, 故障シミュレーション

On exact estimation of logic masking effect for soft erros on combinational circuits

Yusuke MATSUNAGA†

† Faculty of Information Science and Electorical Engineering, Graduate School of Kyushu University
744 Motooka, Nishi-ku, Fukuoka, 819–0395 Japan
E-mail: †matsunaga@c.csce.kyushu-u.ac.jp

Abstract　Soft-error is a phenomena that the output value of a logic gate flips transiently because of neutron particle strike, etc. In combinational circuit, not every soft error affects the behavior of the external outputs because of some masking effects. This paper proposes an efficient algorithm which exactly estimates the logic masking effect. This algorithm utilizes the previously computed results effectively to accelerate the entire computation.

Key words　soft error, combinational circuit, logic masking, fault simulation

1. はじめに

近年，LSI の信頼性において中性子起因のソフトエラーが問題として認識されつつある．従来は特に SRAM のようなメモリ素子の値が反転してしまう現象について多くの解析がなされ，その対策として多重化やエラー検出・訂正符号の導入などが提案されている．一方，ランダムロジックに対するソフトエラーはメモリに比べると確率が小さいことなどからあまり重点的には研究されておらず，また，メモリに比べると規則性がないことから単純で効果的な解析手法や対策は知られていない．

LSI の微細化はいくつかの点でソフトエラー耐性の低下を招いている．一つは CMOS ゲートが駆動している負荷容量が低下することによって，論理反転を起こすのに必要な電荷量 (臨界電荷量) が小さくなっているというものである．もう一つは微細化に伴ってクロック周波数が上がると，単位時間あたりにフリップフロップが値を取り込む回数が多くなり，誤ったデータを取り込む確率が上がるというものである．このようなことからランダムロジックに対するソフトエラーの確率も上昇傾向にある．SRAM などのメモリ素子はエラー訂正などの対策を施すことで見かけのエラー確率を下げることでできているので，近い将来は組み合わせ回路におけるソフトエラーの影響も無視できないと言われている [1], [2].

ランダムロジックにおけるソフトエラーとは，論理ゲートの出力が一定期間だけ，本来の論理値と異なる値を出力するものである．このような論理反転を Single Event Upset(SEU) または Single Event Transition(SET) と呼ぶ．

組み合わせ回路中のゲートにおける論理反転がすべて LSI の動作に影響を与えるわけではなく，以下の 3 つの要因によってエラーの影響の伝搬が阻害される．

論理的マスク (logic masking)　例えば AND ゲートの 1 つ

の入力が0で固定していたら，残りの入力にエラーのパルスが乗っていてもANDゲートの出力には伝搬しない．

電気的マスク (electrical masking) 厳密にはこの現象はマスク効果ではなく，減衰効果である．中性子起因のパルス波形は通常のCMOSゲートの出力波形と異なり，急峻なピークを持っているので，論理反転が起こったゲートから数段のゲート出力波形はだんだんとなまっていく．

時間的マスク (temporal masking, latch window masking) 論理反転がフリップフロップの入力にたどり着いた時にちょうど，フリップフロップが値を取り込むタイミングでない限り，誤った値はフリップフロップに取り込まれない．

これらの振る舞いを見積もる手法はいくつか提案されている[1]〜[6]．しかし，見積り精度と計算時間，大規模回路への適用性などの問題から，実用的な手法が確立しているとは言い難い．

本稿では，これらの要因のうち特に論理的マスクの効果を正確に見積もるための手法について考察する．さらにDisjoint Separator Setと呼ばれる概念を用いて故障シミュレーションを高速化する手法および，Disjoint Separator Setの効率的な計算方法を提案する．以降，2.章で論理マスク効果の見積り手法について説明を行い，3.章でDisjoint Separator Setを用いた故障シミュレーションの高速化について述べる．4.で実験結果を示し考察を行う．

2. 論理マスク効果の見積り手法

2.1 基本定義

本稿では組み合わせ回路をDAG(directed acyclic graph：非巡回有向グラフ)で表す．DAG $G(V_G, E_G)$ はノード $v \in V_G$ および枝 $e \in E_G$ を持つ．以降，特にことわりのないかぎり回路を表すDAGは与えられているものとし，特に表記しない．

ノード v の**ファンイン (fan-ins)** はノード v の入力元となっているノードの集合であり $FI(v)$ で表される．ノード v の**ファンアウト (fan-outs))** とはノード v が出力しているノードの集合であり $FO(v)$ で表される．

外部入力 (primary inputs) とはファンインを持たないノードの集合で PI で表される．**外部出力 (primary outputs)** とはファンアウトを持たないノードの集合で PO で表される．

ノード v の**推移的ファンイン (transitive fan-ins)** とは次式で定義されるノードの集合であり，$TFI(v)$ で表される．

$$TFI(v) = FI(v) \cup \bigcup_{u \in FI(v)} TFI(u)$$

ノード v の**推移的ファンアウト (transitive fan-outs)** とは次式で定義されるノードの集合であり，$TFO(v)$ で表される．

$$TFO(v) = FO(v) \cup \bigcup_{u \in FO(v)} TFO(u)$$

ノード v の推移的ファンアウトである外部出力を $PO(v)$ で表す．

2.2 論理マスク効果の計算に関する諸性質

[**定義** 1] 外部入力に対する入力パタン $m = (m_1, m_2, \ldots)$ が与えられたとき，ノード v における論理反転がノード $u \in TFO(v)$ における論理反転を引き起こすとき，**パタン m でノード v のエラーがノード u に伝搬する**と定義する．また，パタン m および，ノード v，ノード u を入力として，v のエラーが u に伝搬する可能性を表す論理関数を**論理的脆弱性関数 (logic vulnerability function)** と呼び，$LV(m, v, u)$ で表す．

ノード v における論理反転が $PO(v)$ のいずれかにおける論理反転を引き起こす可能性を表す関数をノード v の論理的脆弱性関数と呼び，$LV_{PO}(m, v)$ で表す． □

$LV_{PO}(m, v)$ は次式で計算できる．

$$LV_{PO}(m, v) = \vee_{u \in PO(v)} LV(m, v, u)) \quad (1)$$

[**定義** 2] ノード v における論理反転がノード u における論理反転を引き起こす確率をノード v からノード u への**論理的脆弱性指標 (logic vulnerability factor: LVF)** と呼び，$LVF(v, u)$ で表す．

ノード v における論理反転が $PO(v)$ のいずれかにおける論理反転を引き起こす確率をノード v の論理的脆弱性指標と呼び，$LVF_{PO}(v)$ で表す． □

LVFは回路内部の構造および，回路に与えられる入力パタンの生起確率に依存する．今，回路に与えうる全入力パタンの集合を $\mathcal{M}$ とし，各パタン $m \in \mathcal{M}$ の生起確率 $P_{PI}(m)$ とすると，$LVF(v, u)$ は次式で計算できる．

$$LVF(v, u) = \sum_{m \in \mathcal{M}} P_{PI}(m) \times LV(m, v, u) \quad (2)$$

また，$LVF_{PO}(v)$ は次式で計算できる．

$$LVF_{PO}(v) = \sum_{m \in \mathcal{M}} P_{PI}(m) \times LV(m, v) \quad (3)$$

[**補題** 1] ノード v からノード u へ至るすべての経路が必ずノード w を含むとき，次式が成り立つ．

$$LV(m, v, u) = LV(m, v, w) \wedge LV(m, w, u) \quad (4)$$

証明: 省略 □

ノード v(に対応する組み合わせ回路中のゲート)においてソフトエラー(論理反転)が起こる確率を $P_{seu}(v)$ とすると，ノード v においてソフトエラーが起こって，それが外部出力まで到達する確率 $P_{err}(v)$ は次式で与えられる．

$$P_{err}(v) = P_{seu}(v) \times LVF_{PO}(v) \quad (5)$$

$P_{seu}(v)$ はそのゲートの種類などによって求めることのできる値である．一方，$LVF_{PO}(v)$ は同一の論理関数を実現する組み合わせ回路であっても，その内部構造によって異なる値を持つ．論理設計，もしくは論理設計においては $LVF_{PO}(v)$ を低減させることがソフトエラー対策の目標となる．

2.3 故障シミュレーションによるLVFの計算手法

上記のLVFを計算する単純なアルゴリズムは以下のようになる(図1)

```
calc_lv_naive() {
    foreach m ∈ M {
        正常な回路の全ゲートの論理値を計算する.
        foreach ノード v
            ノード v の論理値を反転した故障回路の論理値を計算する.
            foreach u ∈ PO(v) {
                if u に論理反転の影響が現れている
                LV_PO(m, v) ← 1
            }
        }
    }
}
```

図 1 LV の計算アルゴリズム

通常の場合，全入力パタンの集合 $\mathcal{M}$ および，その要素の生起確率 $P_{PI}(m)$ が列挙不可能であるので，サンプルとなる入力パタン集合 $\mathcal{M}'$ を用意し，そのサンプルのみを用いて LVF の近似値を計算することが広く行われている (いわゆるモンテカルロ法)．また，全ゲートに対して故障シミュレーションを行わずに，Fanout Free Region (FFR) と呼ばれる領域の性質に着目して，2 つ以上のゲートにファンアウトしているゲートのみで故障シミュレーションを行う工夫が広く使われている [7].

3. Disjoint Separator Set を用いた故障シミュレーションの高速化

3.1 Disjoint Separator Set の定義と性質

2 つのノード v_1 と v_2 が推移的ファンアウトを共有しないとき ($TFO(v_1) \cap TFO(v_2) = \phi$)，$v_1$ と v_2 は **TFO-disjoint** であると言う．

ノード v の**セパレータ (seperator)** とは，ノード v から $PO(v)$ へ至るいかなる経路も，そのセパレータの要素のノードを 1 つ以上含むようなノードの集合のことである．一般に，1 つのノードに対するセパレータ集合は多数存在する．

[**定義** 3] ノード v のセパレータ集合のうち，その要素が互いに TFO-disjoint であるようなものを **Disjoint Separator Set: DSS** と呼ぶ． □

[**補題** 2] ノード v および，ノード v に対する Disjoint Separator Set S を考える．このとき，ノード v から $PO(v)$ へ至るいかなる経路上にも，S の要素は必ず 1 つだけ含まれる．

証明: S はノード v のセパレータであるから，その性質より，$PO(v)$ へ至るいかなる経路上にも S の要素が最低 1 つ以上含まれている．

また，S の各々の要素は互いに TFO-disjoint であるので，互いの推移的ファンアウトを共有しない．もしも，同一の経路上に S の 2 つの要素 $s, t \in S$ が含まれていたとすると，どちらかは他方の推移的ファンアウトに含まれることになるので矛盾が生じる．よって同一の経路上にはたかだか 1 つしか S の要素は含まれないことになる．

以上によりノード v から $PO(v)$ へいたるいかなる経路上にも S の要素が必ず 1 つ含まれることが証明された． □

ノード v に対する任意の Disjoint Separator Set を S とする．S の要素 s を用いて，LV_{PO} の計算を以下の様に変更することができる．

[**定理** 1] S をノード v に対する任意の Disjoint Separator Set とするとき，次式が成り立つ．

$$LV_{PO}(m, v) = \vee_{s \in S} LV(m, v, s) \wedge LV_{PO}(m, s) \qquad (6)$$

証明: S はノード v のセパレータ集合なので，$PO(v)$ の各要素は各々が素な集合 $PO(s)(s \in S)$ に分割できる (つまり，$PO(v) = \cup_{s \in S} PO(s)$)．すると，式 (1) は以下の様に変形される．

$$LV_{PO}(m, v) = \vee_{s \in S} \vee_{u \in PO(s)} LV(m, v, u) \qquad (7)$$

Disjoint Separator Set の定義より，ノード v から $PO(s)$ に至るすべての経路は s を含んでいる．よって補題 1 より，

$$\forall u \in PO(s), LV(m, v, u) = LV(m, v, s) \wedge LV(m, s, u) \qquad (8)$$

が成り立つ．式 (8) を式 (7) に代入すると，

$$\begin{aligned} LV_{PO}(m, v) &= \vee_{s \in S} \vee_{u \in PO(s)} LV(m, v, s) \wedge LV(m, s, u) \\ &= \vee_{s \in S} (LV(m, v, s) \wedge (\vee_{u \in PO(s)} LV(m, s, u))) \\ &= \vee_{s \in S} LV(m, v, s) \wedge LV_{PO}(m, s) \end{aligned}$$

となり式 (6) が成り立つ． □

定理 1 の意味するところは，ノード v における LV_{PO} の計算を行うのに，ノード v の論理反転が外部出力まで到達するかを調べる必要はなく，ノード v の Disjoint Separator Set の各々の要素までエラーが伝搬するかを計算し，あとは $LV_{PO}(s)$ の値を利用すれば良いということである．通常は，回路全体のノード v に対して $LV_{PO}(v)$ の計算を行うので，回路の外部出力に近いノードから LV_{PO} の計算を行って行けば，ノード v の LV_{PO} の計算を行う際には，Disjoint Separator Set の要素である s の $LV_{PO}(s)$ の値は計算済みである．

各 $LV_{PO}(v)$ の計算量を減らすためには，なるべくノード v に近い Disjoint Separator Set を利用したほうが有利である．そこで以下のような定義を行う．

[**定義** 4] ノード v に対する 2 つの Disjoint Separator Set S と T が与えられたとき，ノード v から $PO(v)$ へ至るいかなる経路上でも $s \in S$ のノードが $t \in T$ のノードよりも先に現れるか $s = t$ であるとき S **は** T **よりも** v **に近い** と定義し，$S \prec T$ で表す． □

もちろん，ある経路では S の要素のノードが先に現れ，他の経路では T の要素のノードが先に現れる場合も考えられる．この場合，2 つの集合 S と T の間に「近い」という関係は定義できない．

ノード v に対する任意の 2 つの Disjoint Separator Set S と T から以下の様に新しいノード集合 $C(S, T)$ を作る処理を考える (図 2).

[**補題** 3] $C(S, T)$ は Disjoint Separator Set である．

証明: $C(S, T)$ の定義より，ノード v から $PO(v)$ に至るいかな

```
merge_dss(DSS S, DSS T) {
    foreach ノード v から PO(v) へ至る経路 P {
        s ← P に含まれる S の要素
        t ← P に含まれる T の要素
        if s が t よりも先に現れる
            C(S,T) ← C(S,T) ∪ {s}
        else
            C(S,T) ← C(S,T) ∪ {t}
    }
}
```

図 2　Disjoint Separator Set のマージ処理

る経路にも 1 つの $C(S,T)$ の要素が含まれる．よって $C(S,T)$ はノード v のセパレータである．

また，S の各要素，および T の各要素は互いに TFO-disjoint であったので，$C(S,T)$ に含まれるもともと S であった要素どうし，および，もともと T であった要素どうしは互いに TFO-disjoint である．今，$C(S,T)$ に含まれるもともと S であった要素を s，もともと T であった要素を t とする．もしも，s と t が TFO-disjoint でないとすると，s と t の推移的ファンアウトに共通に含まれるノード u が最低一つ存在することになる．$C(S,T)$ の定義より，v から $PO(v)$ へいたる各経路上に含まれる $C(S,T)$ の要素の数は 1 つであり，s と t が同一の経路に含まれることはない．すると v から u にいたる 2 つの経路があって，一つは s を経由し，もう一つは t を経由することになる．すると，v から t を経由して u へ至る経路上に s とは異なる S の要素が含まれなければならない (セパレータの性質)．そのような S の要素を s' とすると，s と s' はその推移的ファンアウトに共通なノード u を含むことになり TFO-disjoint ではなくなり矛盾が生じる．よって，s と t が TFO-disjoint であることが示された．

以上により $C(S,T)$ がノード v に対するセパレータ集合であり，その要素が互いに TFO-disjoint であるので Disjoint Separator Set であることが示された．　□

[補題 4]　$C(S,T) \prec S$ および $C(S,T) \prec T$ である．
証明:　$C(S,T)$ の構成法および $\prec$ の定義より明らか．　□

ちなみに $S \prec T$ であった場合には $C(S,T) = S$ となる．

[定義 5]　ノード v の Disjoint Separator Set のうち，自身の要素のノードよりもノード v に近いノードを要素として持つような他の Disjoint Separator Set が存在しないものを **First Disjoint Separator Set** と呼び，$FDSS(v)$ で表す．　□

[定理 2]　First Disjoint Separator Set は常に唯一つ存在する．
証明:　ノード v に対するすべての Disjoint Separator Set の集合のうち，自身よりも「近い」Disjoint Separator Set の存在しないものの集合を $\mathcal{D}$ とする．自明に $|\mathcal{D}| \geq 1$ である．

今，$\mathcal{D}$ が 2 つ以上の Disjoint Separator Set を含んでいると仮定し，その 2 つを S と T とする．すると，補題 3 によって S と T から新たな Disjoint Separator Set $C(S,T)$ を生成することが可能であり，同時に，補題 4 から $C(S,T)$ は S および T よりもノード v に近いことが分かる．すると，もともと S および T は自身よりもノード v に近い Disjoint Separator Set を持つことになり仮定に反する．よって $\mathcal{D}$ の要素数がたかだか 1 つ以下であることが示された．

以上により $\mathcal{D}$ は常に唯一つの要素のみを持つことがわかる．$\mathcal{D}$ の定義により，この要素 $s \in \mathcal{D}$ が First Disjoint Separator Set である．　□

以上の考察の結果を用いると，回路の各ゲートにおける LV_{PO} の計算を以下のようにして行うことができる．

```
calc_lv() {
    foreach ノード v (外部出力からのトポロジカル順) {
        v の First Disjoint Separator Set を求める．
    }
    foreach m ∈ M {
        foreach ノード v (外部出力からのトポロジカル順) {
            foreach s ∈ FDSS(v) {
                LV(m,v,s) を計算
            }
            式 (6) を用いて LV_PO(m,v) を計算
        }
    }
}
```

図 3　DSS を用いた LV の計算アルゴリズム

3.2　FDSS の計算アルゴリズム

残る問題は効率よく First Disjoint Separator Set を計算することである．単純なアルゴリズム (Algorithm 1) は図 4 のようになる．

```
algorithm1(ノード v) {
1:  D ← {}
2:  S ← FO(v)
    forever {
3:      D1 ← {s|s ∈ S, ∀s' ∈ S, s と s' は TFO-disjoint}
4:      S ← S \ D1
5:      D ← D ∪ D1
6:      if S = {}
            break
7:      t ← TFI(t) ∩ S = φ であるような S の任意の要素 t
8:      S ← S ∪ FO(t) \ {t}
    }
}
```

図 4　単純な FDSS の計算アルゴリズム

この手続きで正しく First Disjoint Separator Set を計算できることを示す．

[補題 5]　Algorithm 1 において $S \cup D$ は常に v のセパレータ集合となっている．
証明:　数学的帰納法を用いて証明する．まず，ステップ 2 において S は v のファンアウトで初期化されている．明かに $FO(v)$

は v のセパレータになっている．

次に，前回の繰り返しまでは $S \cup D$ が v のセパレータになっていると仮定する．ステップ 3 で選ばれたノード集合 $D1$ がなんであれ，ステップ 4 およびステップ 5 で行っていることは S の部分集合を D に移しているだけなので $S \cup D$ は不変であり v のセパレータになっている．ステップ 8 において S から t を取り除き，代わりに $FO(t)$ を加えているが，この処理によっても $S \cup D$ が v のセパレータになっているという性質は変化しない．

よって，常に $S \cup D$ が v のセパレータになっていることが示された． □

［補題 6］ Algorithm 1 のステップ 7 において，t は $FDSS(v)$ よりも v に近い位置にある．

証明： もしも t が $FDSS(v)$ の要素だとすると，DSS の定義より，ノード v から $TFO(t)$ のいずれのノードへ至る経路も t を経由することになる．ところが，ステップ 3 により t と TFO-disjoint でないノードが S に含まれていることになっている．そのようなノードを s' とすると，t の定義より，s' は $TFO(t)$ に含まれることになる．一方，s' が S に含まれているためにはいずれかの繰り返しでステップ 8 において追加されているはずであるが，s' が追加されるまえに，t がすでにそのファンアウトに置き換えられている必要がある．これは矛盾である．よって，t は $FDSS(v)$ の要素ではない．

次に，t が $FDSS(v)$ よりも遠い位置にある可能性を考える．先ほどの s' に関する議論と同様に，t が S に追加されるためにはそのファンイン側のノード t' が一度 S に追加され，それがステップ 7 で選ばれて，ステップ 8 で t に置き換えられる必要がある．そのような一連の繰り返しのなかで，必ず一回は $FDSS(v)$ の要素を選んでそのファンアウトと置き換える処理が必要となるが，上記の議論で，$FDSS(v)$ の要素はステップ 7 では選ばれない，よって，$FDSS(v)$ よりも遠い位置にあるノードは S に追加されないので，ステップ 7 で選ばれることもない． □

［定理 3］ Algorithm 1 は必ず停止し，その時の D はノード v の First Disjoint Separator Set になっている．

証明： ステップ 8 により S に含まれるノードの 1 つがそのファンアウトノードに置き換えられている．また，補題 6 によりステップ 7 で選ばれるノードは常に $FDSS(v)$ よりもノード v 側にあるノードである．ノード v と $FDSS(v)$ の間に存在するノードは有限であり，ステップ 8 で一度置き換えられたノードは二度と S に追加されることはないので Algorithm 1 の繰り返しは有限回で終了する．

補題 5 により $S \cup D$ は常に v に対するセパレータセットになっている．また，D の各要素は互いに TFO-disjoint である．Algorithm 1 の終了時には $S = \phi$ であり，D がセパレータセットかつ TFO-disjoint ということになるので，D はノード v の Disjoint Separator Set である．

再度，補題 6 より $FDSS(v)$ よりも外部出力側のノードは S に追加されないので，結果として D にも含まれない．よって D が First Disjoint Separator Set となる． □

定理 3 で示したようにこのアルゴリズムは正しく動くが，一つのノード対する $FDDS(v)$ を計算するのに回路規模に比例した計算の手間を必要とする．実際に実験結果で示すように，この計算量は莫大で場合によっては 1,000 パタン程度のシミュレーション時間を遥かに越える計算時間を必要とする場合もある．そこで，上記の Algorithm 1 のステップ 8 を改良した Algorithm 2 を示す (図 5.

```
algorithm2(ノード v) {
1:  D ← {}
2:  S ← FO(v)
    forever {
3:      D1 ← {s|s ∈ S, ∀s' ∈ S, s と s' は TFO-disjoint}
4:      S ← S \ D1
5:      D ← D ∪ D1
6:      if S = {}
            break
7:      t ← TFI(t) ∩ S = φ であるような S の任意の要素 t
8':     S ← S ∪ FDSS(t) \ {t}
    }
}
```

図 5 改良版 FDSS の計算アルゴリズム

つまり，t をそのファンアウトで置き換える代わりに $FDSS(t)$ で置き換えるというものである．このアルゴリズムにおいても補題 5 と同様の性質が成り立つことは自明である．

［補題 7］ ノード v から $FDSS(v)$ のいずれかのノードへ至る経路上のノードを t としたとき，$FDSS(t)$ のすべてのノードはノード t から $FDSS(v)$ のいずれかのノードへ至る経路上にある．

証明： $FDSS(v) \cap TFO(t)$ は $FDSS(v)$ の部分集合なので互いに TFO-disjoint であり，また，t に対するセパレータセットになっていることから t の Disjoint Separator Set である．よって，t の First Disjoint Separator Set は $FDSS(v) \cap TFO(t)$ よりも外部出力側には存在しない． □

［補題 8］ Algorithm 2 のステップ 7 において，t は $FDSS(v)$ よりも v に近い位置にある．

証明： スペースの都合で省略 □

［定理 4］ Algorithm 2 は必ず停止し，その時の D はノード v の First Disjoint Separator Set になっている．

証明： 補題 5，補題 7，および補題 8 と定理 3 より自明 □

アルゴリズムの変更点は些細であり，最悪の計算複雑度は回路規模に比例しているが，この改良版アルゴリズムは実行時間においては大幅な改善を達成している．

4. 実験および考察

本稿で提案した FDSS の計算法およびそれを利用した LVF

の計算法をプログラム実装し，計算時間の評価を行った．実験の手順は以下のとおりである．

（1） ISCAS85, ISCAS89, ITC99 のベンチマーク回路を入力データとする．順序回路の場合はフリップフロップの出力を擬似的な外部入力とみなし，フリップフロップの入力を擬似的な外部出力とみなす．

（2） 1000 パタンの入力をランダムに生成する．これを入力集合 $\mathcal{M}$ とする．

（3） $\mathcal{M}$ に対する全ゲートの LVF を計算する．

その際に，以下のような異なる手法を用いて計算時間を比較した．計算される LVF の値そのものはすべて同一である．

DSS: DSS を用いた提案手法 (図 3) による故障シミュレーションにかかった計算時間．ここには FDSS を求める初期化は含まれていない．

Naive: DSS を用いない従来手法による計算時間．図 1 のアルゴリズムをベースに 2 つ以上のファンアウトを持つゲートのみでシミュレーションを行うもの．

初期化は以下の 2 通りを実装し比較した．

Alg1: Algorithm 1 (図 4) を用いた時の計算時間

Alg2: Algorithm 2 (図 5) を用いた時の計算時間

表 1 に実験結果を示す．

表 1 実験結果

回路名	DSS	Naive	Alg2	Alg1
C5315	0.65	1.08	0.00	0.01
C6288	8.95	8.89	0.00	0.24
C7552	1.18	2.12	0.00	0.10
S5378	0.39	0.73	0.00	0.01
S9234	0.70	1.31	0.00	0.03
S13207	0.66	1.33	0.00	0.07
S15850	1.13	3.21	0.01	0.15
S35932	1.46	4.78	0.07	0.67
S38417	3.22	7.42	0.03	0.14
S38584	2.78	5.04	0.02	0.70
b14	13.60	18.82	0.11	31.70
b15	15.80	16.90	0.45	32.09
b17	53.20	57.16	0.86	84.47
b18	310.14	449.55	1.25	152.08
b19	715.07	999.14	2.69	322.69
b20	34.83	47.64	0.25	71.21
b21	36.65	47.64	0.26	67.82
b22	59.15	81.24	0.39	105.37

いずれの回路に対しても DSS を用いた提案手法は従来の手法より高速である．C6288 や b15 のようにほとんど差が見られない場合もあるが，その他の回路に関しては提案手法は概ね，従来手法の 80% ～ 50% 程度の時間で処理を行っている．FDSS の計算に改良版の Algorithm 2 を用いた場合，初期化にかかる時間は 1000 パタンの故障シミュレーションの数%程度であり，無視できる．一方，Algorithm 1 は対象回路の規模が大きくなると急激に計算時間が増大し，なかには 1000 パタンの故障シミュレーション以上の時間を要する例もあり，このアルゴリズムが実用的でないことがわかる．

5. おわりに

本稿では，組み合わせ回路におけるソフトエラーの伝搬可能性を見積もる手法について延べ，その故障シミュレーションを高速化するために Disjoint Separator Set と呼ばれる概念を提案し，同時に Disjoint Separator Set を効率よく計算するアルゴリズムを提案した．Disjoint Separator Set を用いることで故障シミュレーションの計算時間は 80% ～ 50% 程度に短縮できている．DSS の計算も非常に高速に行えており，ソフトエラーの評価のように多くのパタンで，なんども同じ故障の影響をシミュレートする場合には非常に適しているといえる．

今後は他のマスク効果である electrical masking や latch window masking の影響も考慮した見積り手法について検討する予定である．

謝　辞

本研究は，独立行政法人科学技術振興機構 CREST-DVLSI の研究補助金の助成によるものである．

文　献

[1] S. Krishnaswamy, S. Plaza, I. Markov and J. Hayes: "Enhancing Design Robustness with Reliability-aware Resynthesis and Logic Simulation", Proceedings of IEEE International Conference on Computer Aided Design, pp. 149–154 (2007).

[2] Y. Lin and L. He: "Device and Architecture Concurrent Optimization for FPGA Transient Soft Error Rage", Proceedings of IEEE International Conference on Computer Aided Design, pp. 194–198 (2007).

[3] M. Zhang and N. Shanbhag: "A Soft Error Rate Analysis (SERA) Methodology", Proceedings of IEEE International Conference on Computer Aided Design, pp. 111–118 (2004).

[4] B. Zhang, W. Wang and M. Orshansky: "FASER: Fast Analysis of Soft Error Susceptibility for Cell-Based Designs", ISQED (2006).

[5] N. Miskov-Zivanov and D. Marculescu: "MARS-C: Modeling and Reduction of Soft Erros in Combinational Circuits", Proceedings of 43nd IEEE/ACM Design Automation Conference (2006).

[6] R. Rao, D. Blaauw and D. Sylvester: "An Efficient Static Algorithm for Computing the Soft Error Rates of Combinational Circuits", Proceedings of Design Automation and Test in Europe (2006).

[7] J. Waicukauski: "Fault Simulation of Structured VLSI", VLSI Systems Design (1985).