A-01

組み合わせ回路におけるソフトエラー発生確率の効率的計算手法

An Effective Calculation Method for Soft-Error Probability Analysis in Combinatorial Circuits

津島 雅俊 † 山下 茂 †
Masatoshi Tsushima Shigeru Yamashita

1. はじめに

回路内のゲートは，外部からの影響によって一時的な誤作動をすることがある．このようなエラーはソフトエラーと呼ばれ [1]，主に中性子線などの宇宙線によって引き起こされる．これまでの回路の微細化や低電圧化に伴い，ソフトエラーの発生は顕著になってきている．将来的には，このエラーの発生を考慮した設計が求められるようになると考えられている [2]．

ソフトエラーの発生は，必ずしも回路全体に致命的な影響を与えるわけではない．例えば，2 入力の AND ゲートに本来 $(00)_2$ の入力を与えるべき状況において，二つの入力のうち一つが反転したとしても最終的な結果が変化することはない．このように，ゲートの論理によってソフトエラーの伝搬が阻害される効果を論理マスクと呼ぶ．一方で，入力が $(11)_2$ の際には，どちらの入力も反転することは許されない．さらに，この例から回路全体に対するエラーの影響は入力パターンに依存することがわかる．すなわち，回路の入力数 n に対して 2^n パターンの場合の回路の故障確率が考える必要がある．

ソフトエラーの発生による誤動作の確率を見積るためには，既にいくつかの手法が知られている．Probabilistic Transfer Matrix (PTM) を用いて入力パターンに対する出力の発生確立を求める手法 [2] や，論理的脆弱性指標 (logic vulnerability factor: LVF) を用いた論理マスクの影響を見積もる手法 [3] などである．しかし，いずれの方法においてもソフトエラーによる回路の誤動作の厳密な確率の計算は困難であり，大規模回路には適用できていない．そのため，現実的な時間で回路の検証を行うためには，いくつかの限られた入力パターンのみを試すなどのヒューリスティックが用いられている．

PTM による手法では，回路の形状によっては計算時間が増大する．本稿では PTM による計算手法とは異なるアプローチによりソフトエラーの発生による回路の誤動作の確率を効率的に計算する手法を提案する．第 2 章で既存の PTM による計算手法について説明したのち，第 3 章で提案手法，第 4 章で実験結果について述べる．さらに第 5 章では提案手法の改善について考察する．

2. 既存手法

2.1. Probabilistic Transfer Matrix

PTM は，ゲートや回路の入力に対する出力の確率を表す行列である．各ゲートや結線の分岐を個々の PTM で表し，それらを用いて回路全体の PTM を生成していく．

† 立命館大学大学院情報理工学研究科, Graduate School of Information Science and Engineering,Ritsumeikan University

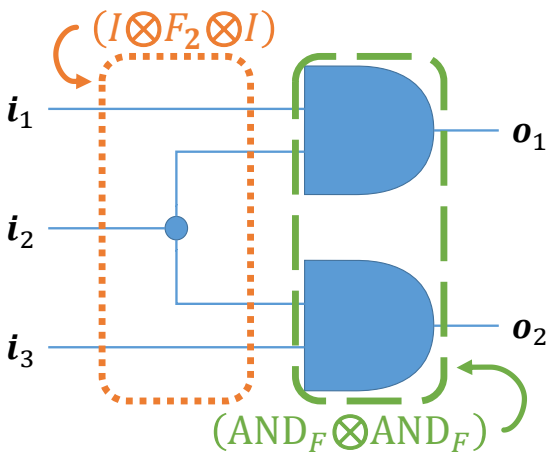

図 1　組み合わせ回路 C の例

定義 **1.** m 入力， n 出力の部分回路を考える．部分回路の入力に長さ m ののビット列 $\boldsymbol{i}$ を与えた時，出力が長さ n のビット列 $\boldsymbol{o}$ となる確率が p ならば，この部分回路に対応する PTM は 2^m 行 2^n 列の行列で，M_F の $\boldsymbol{i}+1$ 行 $\boldsymbol{o}+1$ 列目の値を p とする．このとき， $\boldsymbol{i}, \boldsymbol{o}$ は，ビット列を 2 進数と解釈したときの数値である．

$$M_F(\boldsymbol{i}+1, \boldsymbol{o}+1) = p$$

M の行と列の数は，それぞれ入力と出力のパターン数だけ存在する．

例えば，確率 0.05 で反転する AND ゲートの PTM AND_F は以下のようになる．

$$AND_F = \begin{bmatrix} 0.95 & 0.05 \\ 0.95 & 0.05 \\ 0.95 & 0.05 \\ 0.05 & 0.95 \end{bmatrix}$$

この例の 1 行目は，入力が $\boldsymbol{i} = (00)_2$ となる時，出力は 0.95 の確率で $\boldsymbol{o} = (0)_2$ となるが，0.05 の確率で反転して $\boldsymbol{o} = (1)_2$ となることを表している．一方 4 行目は，入力が $\boldsymbol{i} = (11)_2$ となる時，出力は 0.95 の確率で $\boldsymbol{o} = (1)_2$ となるが，0.05 の確率で反転して $\boldsymbol{o} = (0)_2$ となることを表している．また， PTM では図 1 の回路にあるような分岐を以下のように表す．

$$F_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ゲートの直列接続に対応する PTM は，対応する PTM 同士の積に対応する．また，並行したゲートに対応する PTM は，

対応する PTM 同士のテンソル積（*Kronecker* 積，$\otimes$）に対応する．これらの計算を組み合わせることで，目的とする回路の PTM を得ることができる．

例えば，確率 0.05 で反転する AND ゲートを用いて図 1 の回路を構成した時，この回路に対応する PTM は式 1 のように求められる．

$$M_F = (I \otimes F_2 \otimes I)(AND_F \otimes AND_F) \quad (1)$$
$$= \begin{bmatrix} 0.9025 & 0.0475 & 0.0475 & 0.0025 \\ 0.9025 & 0.0475 & 0.0475 & 0.0025 \\ 0.9025 & 0.0475 & 0.0475 & 0.0025 \\ 0.0475 & 0.9025 & 0.0025 & 0.0475 \\ 0.9025 & 0.0475 & 0.0475 & 0.0025 \\ 0.9025 & 0.0475 & 0.0475 & 0.0025 \\ 0.0475 & 0.0025 & 0.9025 & 0.0475 \\ 0.0025 & 0.0475 & 0.0475 & 0.9025 \end{bmatrix}$$

よって， $M_F(4,2) = 0.9025$ という計算結果から，$\boldsymbol{i} = (011)_2$ の時に 0.9025 の確率で $\boldsymbol{o} = (01)_2$ が得られるとわかる．

回路に故障が発生しない理想的な状況に対して PTM を計算すると，確率 p は 0 もしくは 1 となる．このような PTM を，特に Ideal Transfer Matrix (ITM) と呼ぶ．図 1 の回路に対応する ITM は，式 2 のようになる．

$$M = (I \otimes F \otimes I)(AND \otimes AND) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

回路の最終的な信頼性は，正しい出力が得られる確率を，入力パターンが与えられる確率によって重み付き平均をとる (式 3) ことによって評価する．入力パターンが与えれれる確率はベクトル $\boldsymbol{v}$ で表現し，$\boldsymbol{i}+1$ 番目の要素は，$\boldsymbol{i}$ を 2 進数としたときの入力パターンが与えられる確率を表している．

$$\mathit{fidelity}(\boldsymbol{v}, M, M_F) = |\boldsymbol{v}(M_F . * M)|_1 \quad (3)$$

ここで， $.*$ は要素ごとの乗算を表し， $|\boldsymbol{v}|_1$ は，ベクトル $\boldsymbol{v}$ の l_1-ノルム を表す．

この回路に全ての入力パターンが等確率で与えられる場合，$\boldsymbol{v}$ は，各要素は 0.125 で要素数 8 のベクトルとなり，式 3 を用いて 0.9025 という値が得られる．この結果は，ソフトエラーによって 0.05 の確率で反転する AND ゲートを用いて図 1 の回路を構成した時，全ての入力パターンが等確率で与えられることを仮定すると， 0.9025 の確率で正常に動作することを意味している．

2.2. ADD の利用

第 2.1 節の手法において，計算の途中で発生する PTM のサイズは，指数的に大きくなる可能性がある．与えられた二つの行列 M_1, M_2 のサイズが $k \times l$ と $m \times n$ のとき，$M_1 \otimes M_2$ のサイズは $km \times ln$ となる．そのため，回路の中で並行した部分の入力と出力の数を n', m' とすると，$2^{n'} \times 2^{m'}$ の PTM が生成される．この課題に対して，Krishnaswamy らは *Algebraic Decision Diagram* (ADD) [4] を用いた [2]．PTM を ADD で表現することにより，圧縮された状態で計算することができる．ただし， ADD のノード数は行列のサイズに相関する．そのため，この改善を行ったとしても，現実的な時間やメモリで解くことができるのは小規模な回路に限られる．大規模な回路に適用するためには，より効率的な計算方法が必要となる．

3. 提案手法

第 2 章で述べたように，PTM のサイズが計算を制限する可能性がある．そこで，提案手法ではこの行列を用いずに，$M_F . * M$ と等価な値を得る別のアプローチを検討する．

$M_F . * M$ は，正しい出力以外を 0 にする計算とみなすことができる．$M_F . * M$ から，行ごとに 0 以外の要素を抜き出したベクトル $\boldsymbol{s}$ を用いると，式 3 の値は内積を用いて式 4 で求めることができる．

$$\mathit{fidelity}(\boldsymbol{v}, M, M_F) = \boldsymbol{v} \cdot \boldsymbol{s} \quad (4)$$

$\boldsymbol{s}$ の $\boldsymbol{i}+1$ 番目の要素は，2 進数として表したビット列 $\boldsymbol{i}$ を入力したときに正しい出力が得られる確率である．第 2.1 節の例の場合， $\boldsymbol{s}$ は各要素が 0.9025 で要素数 8 のベクトルとなり，$\boldsymbol{v} \cdot \boldsymbol{s} = 0.9025$ となる．

回路の外部入力と外部出力をそれぞれ PI, PO で表す．回路に $\boldsymbol{i}$ を入力した時のゲート g が反転する確率 $\mathit{fail}(\boldsymbol{i}, g)$ とする．全ての外部出力が反転しなければ良いので，$\boldsymbol{s}$ の各要素は，式 5 のように求められる．

$$\boldsymbol{s}(\boldsymbol{i}+1) = \prod_{g \in PO} (1 - \mathit{fail}(\boldsymbol{i}, g)) \quad (5)$$

$\boldsymbol{i}$ を入力した時にゲート g が反転する確率 $\mathit{fail}(\boldsymbol{i}, g)$ は，ゲートのファンインで考えられる全てのエラーパターンについて考える．ゲート g のファンインを $FI(g)$ で表すと，式 6 のように計算できる．

$$\mathit{fail}(\boldsymbol{i}, g) = \sum_{\boldsymbol{e} \in \{0,1\}^{|FI(g)|}} O(\boldsymbol{i}, g, \boldsymbol{e}) \times \mathit{to\_fail}(\boldsymbol{i}, g, \boldsymbol{e}) \quad (6)$$

$$O(\boldsymbol{i}, g, \boldsymbol{e}) = \prod_{\substack{h \in FI(g), \\ e_h \in \boldsymbol{e}}} \begin{cases} \mathit{fail}(\boldsymbol{i}, h) & : e_h = 1 \\ 1 - \mathit{fail}(\boldsymbol{i}, h) & : e_h = 0 \end{cases} \quad (7)$$

$$\mathit{to\_fail}(\boldsymbol{i}, g, \boldsymbol{e}) = \begin{cases} \mathit{err}(g) & : \mathit{out}(\boldsymbol{i}, g) = \mathit{act}(\boldsymbol{i}, g, \boldsymbol{e}) \\ 1 - \mathit{err}(g) & : \mathit{out}(\boldsymbol{i}, g) \neq \mathit{act}(\boldsymbol{i}, g, \boldsymbol{e}) \end{cases} \quad (8)$$

式 7 の $O(\boldsymbol{i}, g, \boldsymbol{e})$ は，エラーパターン $\boldsymbol{e}$ の発生率である．$\boldsymbol{e}$ は，要素数 $|FI(g)|$ のビット列で，i 番目のビットが 1 の時，ゲートの i 番目の入力が反転していることを表している．また， e_h は，ゲート h からの入力が，ソフトエラーによって反転しているときに $e_h = 1$ とする．すなわち，全てのファンインについて反転が発生する確率を調べ，エラーパターン $\boldsymbol{e}$ が同時に発生する確率を求めている．

式 8 の $\mathit{to\_fail}(\boldsymbol{i}, g, \boldsymbol{e})$ は，ゲートの出力が最終的に反転する確率である．回路に $\boldsymbol{i}$ を入力した時のゲート g の出力を $\mathit{out}(\boldsymbol{i}, g)$ とし，$\boldsymbol{i}$ を入力してエラーパターン $\boldsymbol{e}$ が発生した時の

Algorithm 1 fail: 与えられたノードの出力が反転する確率を求める

Require: g: 出力のエラー率を求めるノード

1: **if** g が終端 **then**
2: **return** 0
3: **end if**
4: $res = 0$
5: **for all** $\boldsymbol{e} \in \{0,1\}^{|FI(g)|}$ **do**
6: $O \leftarrow 1$
7: **for all** $h \in FI(g), e_h \in \boldsymbol{e}$ **do**
8: **if** $\boldsymbol{e}_h = 1$ **then**
9: $O \leftarrow O \times fail(h)$
10: **else**
11: $O \leftarrow O \times (1 - fail(h))$
12: **end if**
13: **end for**
14: $res \leftarrow res + O \times to\_fail(g, \boldsymbol{e})$
15: **end for**
16: **return** res

ゲート g の出力を $act(\boldsymbol{i}, g, \boldsymbol{e})$ とする．to_fail は，影響が無い（$out(\boldsymbol{i}, g) = act(\boldsymbol{i}, g, \boldsymbol{e})$）場合はゲート g の出力が反転する確率 $err(g)$ を返し，影響がある（$out(\boldsymbol{i}, g) \neq act(\boldsymbol{i}, g, \boldsymbol{e})$） 場合は反転しない確率を返す．

従って，式 6 の $O(\boldsymbol{i}, g, \boldsymbol{e})to\_fail(\boldsymbol{i}, g, \boldsymbol{e})$ は，エラーパターン e が発生し，かつ，そのエラーパターンの時に最終的なゲート g の出力が正しい値から反転する確率を求めている．

実際の計算では，各入力を終端ノードに設定してから Algorithm 1 のように再帰的に計算を行う．6-13 行目が式 7 の計算に相当する．to_fail の計算に用いる out の値は，入力の設定時に事前に計算することができる．また，act の値は， out と $\boldsymbol{e}$ を用いて計算できる．この計算は，全ての出力線から開始しなくてはならない．

関数 $fail$ の結果は，入力パターンとノードの組に対して一意に定まるので，一度求めた結果は再利用することができる．具体的には，Algorithm 1 に加えて， g に対する値を計算していないか調べ，存在すればその値を返すような処理行なえばよい．また，入力パターンを変更する毎に計算結果をクリアしている．したがって，一つの入力パターンに対しては，式 5 のように $|PO|$ 回だけ再帰を開始しなくてはならないが，パターンを設定してから二つ目以降の外部出力に対する結果は，過去に計算した値を利用できる場合があるため比較的少ない時間で求められると期待できる．

4. 実験

第 3 章で提案した手法についてプログラムを実装し，以下の手順で実験を行った．

1. ベンチマークには LGSynth'93 に含まれる BLIF フォーマットで用意された組み合わせ回路を用いる．
2. 入力以外の全てのゲートに $err(g) = 0.05$ を設定し， $\boldsymbol{s}$ を求める．
3. グラフ構造の生成から $\boldsymbol{s}$ の生成までを計算時間とする．回路データの入力や，結果の出力にかかる I/O の時間は含めない．
4. 1 時間以内に計算できない場合は計算を打ち切る

表 1　計算時間の比較結果

回路	入力数	提案手法	PTM
C17	5	0.000	0.212
decod	5	0.001	5.132
xor5	5	0.000	3.589
z4ml	7	TLE	3.849
9symml	9	0.060	89.145
x2	10	0.071	11.015
cu	14	0.801	23.700
parity	16	2.133	1.060
pm1	16	3.070	72.384
pcle	19	19.560	28.810
cc	21	123.58	57.400
mux	21	TLE	18.052

実験結果を表 1 に示す．**回路** は回路データのファイルイル名，**入力数**は回路に含まれる入力の数を表している．**提案手法**は，提案手法の計算時間（秒），PTM は既存手法の計算時間（秒）[2] である．表中の TLE は，時間内に計算を終えられなかったことを示している．

実験環境の CPU は Intel®Core™ i5-3570 CPU (3.40GHz)，メモリは 7.7 GB．ソフトウェアは Linux 上で g++ 4.8.3 を用いている．

既存手法の実験では，2GHz の Pentium®4 を用いている．従って，2 倍以上の高速化を達成しているケースに関しては，改善できたと考えられる．

5. 改善手法

5.1. ゲートの分割

第 5 章で紹介した結果の中で，mux と z4ml の 2 つのケースは計算を十分な時間で終えることができなかった．これは回路中に $|FI(g)|$ の大きなゲートが存在することが原因であると考えられる．

提案手法では，一つの入力パターンに対して全てのゲートについて $fail$ を計算すれば良い．このとき， $fail$ の計算にはゲート一つあたり $2^{|FI(g)|} \times |FI(g)|$ 回のループが必要である．多くのケースについては，最大の $|FI(g)|$ が 2 もしくは 3 程度であるが，mux には $|FI(g)| = 9$，z4ml には $|FI(g)| = 28$，となるようなゲートが存在する．

n 入力のゲートは， 2 入力のゲート $n-1$ 個で表すことができる．分割したゲートの中で，最後の結果を出力するゲートのみが分割前のゲートと同じ確率で反転し，それ以外は反転の発生しない理想的なゲートとみなすことで，分割前と等価な回路を得ることができる．このようにすると，理想的なゲートに対

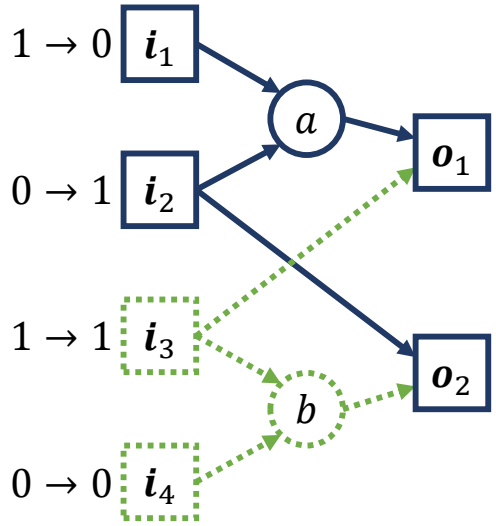

図 2　4 入力 2 出力の回路の例と再計算

する *fail* の値は，出力が反転するような入力が与えられる確率となる．

$|FI(g)| \geq 4$ のゲートに対して分割を適用することで，$2^{|FI(g)|} \times |FI(g)|$ 回のループが $2^3 \times (|FI(g)| - 1)$ 回の程度に改善されると期待できる．

5.2. 変化した入力のみの再計算

ゲートに与えられる出力が反転する確率は，ゲートに与えられる入力パターンとそれらが反転する確率に依存している．したがって，それらが変わらない限り，出力が反転する確率を計算する必要はない．

第 3 章の手法では，出力側のノードから計算を開始するが，この手法では，入力側からの幅優先探索を用いた実装が考えられる．入力が変化したノードから探索を開始し，回路の出力や反転する確率が変化した場合は，出力先のノードをキューに追加する．

例えば、4 入力 2 出力の回路を図 2 のような閉路の無い有向グラフとして考える。この時、入力が変化した $\boldsymbol{i}_1, \boldsymbol{i}_2$ から始めて、実線で変化したノードのみを更新すればよい。最初に $\boldsymbol{i}_1$ の設定時に a，$\boldsymbol{i}_2$ の設定時に $\boldsymbol{o}_1$ を enqueue する．次に a を dequeue し，回路の出力と反転する確率を更新する．さらに，キューを操作して $\boldsymbol{o}_2, \boldsymbol{o}_1$ の順で更新する．このようにすると，変化した入力に依存するノードのみを再計算できる．

5.3. グレイコードの利用

第 5.2 節の手法は，順番によっては全ての入力線から再計算をおこなう必要が生じる場合がある．例えば $|PO| = 3$ の場合，$(000)_2, (001)_2, (010)_2, (011)_2$ の順で割り当てると，次の 100 の計算は三つの入力を更新し，全てのゲートについて値を更新しなくてはならない．一般に，入力値の割り当てを 1 ずつ増やした 2 進数で行う場合，2^n 回目の更新では， $n+1$ 個の入力を更新し，再計算しなくてはならない．

そこで，入力値の割り当てにグレイコードを用いる方法が考えられる．例えば 3 ビットのグレイコードを用いると，ビット列は $(000)_2, (001)_2, (011)_2, (010)_2$ のように常に 1 ビットしか変化しない．この手法を用いると，常に 1 カ所の入力だけを変化させることで全ての入力値の割り当てを計算することができる．よって，再計算しなくてはならない回数を減少させることができると考えられる．

6. おわりに

本稿では，ソフトエラーの発生による回路の誤動作の確率を効率的に計算する手法について提案した．実験では，従来に比べ 5000 倍以上の高速化を実現したケースも存在している．第 5 章で述べたいくつかの改善手法により，さらなる高速化が期待できる．今後は，本稿で検討した改善手法などを実装・検証し，より大きな規模の回路に対して適用可能な手法の開発を目指す．

参考文献

[1] Baumann, R.: Soft Errors in Advanced Computer Systems, *IEEE Des. Test*, Vol. 22, No. 3, pp. 258–266 (online), DOI: 10.1109/MDT.2005.69 (2005).

[2] Krishnaswamy, S., Viamontes, G. F., Markov, I. L. and Hayes, J. P.: Probabilistic Transfer Matrices in Symbolic Reliability Analysis of Logic Circuits, *ACM Trans. Des. Autom. Electron. Syst.*, Vol. 13, No. 1, pp. 8:1–8:35 (online), DOI: 10.1145/1297666.1297674 (2008).

[3] 裕介松永: 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 (プロセッサ, システム LSI の応用と要素技術, プロセッサ,DSP, 画像処理技術及び一般), 情報処理学会研究報告. SLDM, [システム LSI 設計技術]， Vol. 2008, No. 98, pp. 53–58（オンライン), 入手先 ⟨http://ci.nii.ac.jp/naid/110007081317/⟩ (2008).

[4] Bahar, R., Frohm, E., Gaona, C., Hachtel, G., Macii, E., Pardo, A. and Somenzi, F.: Algebraic decision diagrams and their applications, *Computer-Aided Design, 1993. ICCAD-93. Digest of Technical Papers., 1993 IEEE/ACM International Conference on*, pp. 188–191 (online), DOI: 10.1109/ICCAD.1993.580054 (1993).