設計自動化 62－27
(1992. 5.28)

論理関数処理による組合せ論理回路に対する最小テスト集合の生成

樋口 博之　　濱口 清治　　矢島 脩三
京都大学工学部

あらまし

共有二分決定グラフを用いた論理関数処理により組合せ論理回路に対する最小テスト集合を生成する方法を2つ提案する。最小テスト集合生成問題は故障表に対する最小カバー問題に帰着できるが、本手法は、最小カバー問題の解法として知られているペトリック関数を用いる方法を論理関数処理により行なう場合に比べ、多くの場合処理する関数の変数の数が少なく、より効率的に最小テスト集合の生成が可能である。実験により、本手法では、ゲート数100程度の回路に対しても最小テスト集合が生成できる場合があることがわかった。

Generation of Minimum Test Sets for Combinational Logic Circuits Based on Boolean Function Manipulation

Hiroyuki HIGUCHI　　Kiyoharu HAMAGUCHI　　Shuzo YAJIMA
Faculty of Engineering, Kyoto University

Abstract

We propose two methods of generating minimum test sets for combinational logic circuits based on Boolean function manipulation using Shared Binary Decision Diagrams(SBDD's). The minimum test set generation problem can be reduced to the minimum covering problem for fault tables. As a method of solving the minimum covering problem based on Boolean function manipulation, the method using the Petrick functions has been proposed. Since our algorithms require fewer Boolean variables than the conventional method, minimum test sets can be generated more efficiently. Experimental results show that our methods can generate minimum test sets for the circuits composed of about 100 gates in some cases.

1 はじめに

近年の集積回路技術の進歩により、集積回路はより広範な分野で利用されるようになっている。これらの回路の信頼性確保のために、故障検査の技術は益々重要性を増している。一般に、集積回路の故障検査は、外部入力端子にテスト系列を印加したときの外部出力端子での応答を観測することにより行われる。故障検査に必要なテスト系列を生成する手続きをテスト生成という。テスト生成問題はNP完全であることが知られているが [1] 、組合せ回路のテスト生成アルゴリズムは、[2][3] など数多く考案され、大部分の故障に対しては現実的な時間でテストベクトルを求めることが可能となっている。

故障検査では、一般に、テスト系列を1ベクトルずつ外部入力端子に印加するため、故障検査のコストはテスト系列の長さに比例する。スキャン設計の下では、テストベクトルをスキャン・パスに直列に入力する必要がある。そのため、1ベクトル当たりに要する故障検査に要する時間が増大し、テスト集合のサイズの、故障検査のコストに対する影響はさらに大きくなる [4]。従って、できるだけ小さなテスト集合を生成することが望まれる。

多くのATPG(Automatic Test Pattern Generation)においては、テスト生成の高速化を図ることが主たる目標とされ、テスト集合のサイズの削減はテスト生成の効率を損なわない範囲で行なわれてきた。小さなテスト集合を生成することを第一の目標とした方法としては、[5] が提案されている。[5] では、独立故障集合という概念を用いて最小テスト集合のサイズの下界を求めている。さらに、独立故障集合をもとに小さなテスト集合の生成アルゴリズムが提案されているが、最小テスト集合が生成されるとは限らない。最小のテスト集合のサイズを求めることは、テスト集合の圧縮法の評価を行なうためにも重要な問題である。そこで、本稿では、論理関数処理により最小なテスト集合を求める方法を2つ提案する。実験を行なった結果、ゲート数100程度の回路に対しても最小テスト集合が生成できる場合があることが分かった。

以下、2章では、まず準備として、本稿で対象とする回路と故障モデルについて述べ、テスト集合最小化に関する諸定義を行う。さらに、論理関数の内部表現方法として用いる共有二分決定グラフについて述べる。3章では、記号故障シミュレーションによる故障表の生成とその簡約化手法について述べる。4章では、故障表からテスト系列の集合を表す特徴関数を生成することにより、最小テスト集合を求める方法について述べる。5章では、テスト系列を受理するオートマトンの状態遷移グラフ上の探索により、最小テスト集合を生成する方法について述べる。6章では、4、5章に基づく2つの最小テスト集合生成プログラムの実現とその評価について述べる。7章では、結びとして、本研究の成果と今後の展望を述べる。

2 準備

2.1 対象回路と故障モデル

テスト生成において、対象とする回路の種類、回路記述のレベルには様々なものがあり、必要に応じた選択が行なわれる。本稿ではゲートレベル、零遅延でモデル化された組合せ回路のみを対象とする。回路に仮定される故障に関しても、種々のモデルが考えられるが [6]、実際に起こる故障の多くは縮退故障モデルに帰着できるため、縮退故障を考えることが多い。縮退故障とはゲートの入出力端子の信号値が0または1に固定される故障であり、それぞれ0縮退故障、1縮退故障という。本稿では、単一縮退故障を仮定する。

2.2 諸定義

n 入力 m 出力組合せ回路において、各外部入力の信号値を $\vec{x} \stackrel{\text{def}}{=} (x_1, x_2, \cdots, x_n)$ とする。i 番目の出力 z_i は入力ベクトル $\vec{x}$ の論理関数として $z_i = f_i(\vec{x})$ と表すことができる。いま、ある故障 α により i 番目の出力の論理関数が f_i^α になるとする。このとき、次の関数 F^α を故障 α の**故障差関数**という。

$$F^\alpha \stackrel{\text{def}}{=} \bigvee_{i=1}^{m} (f_i \oplus f_i^\alpha). \tag{2.1}$$

故障差関数 F^α は次の性質を満たす論理関数である、

$$F^\alpha(\vec{x}) = 1 \Leftrightarrow \exists i. f_i(\vec{x}) \neq f_i^\alpha(\vec{x}).$$

$F^\alpha(\vec{x}) = 1$ を満たす入力ベクトル $\vec{x}$ は故障 α を検出しうる入力ベクトルであり、これを故障 α の**テストベクトル**と呼ぶ。また、入力ベクトルを行に、故障を列に対応させ、入力 i が故障 j を検出するとき i 行 j 列を1とし、検出できないとき0とした表を**故障表**という。

故障 α に対する故障時の出力関数が正常時の出力関数に等しい、すなわち、故障差関数 $F^\alpha = 0$(恒偽関数) のとき、この故障は検出不可能である。このとき、故障 α を**冗長故障**という。また、二つの故障 α と β に対し、$F^\alpha = F^\beta$ となるとき、故障 α と故障 β は**等価故障**であるという。等価故障が存在する場合、その中の一つの故障だけを対象とすればよい。等価故障の中から選ばれた一つの故障を**代表故障**という。故障 α, β を検出する入力ベクトルの集合をそれぞれ T_α, T_β としたとき、$T_\beta \supseteq T_\alpha$ ならば、故障 β は故障 α を**支配する**という。このとき故障 α の故障差関数 F^α は故障 β の故障差関数 F^β を含意する (以下、$F^\alpha \rightarrow F^\beta$ と書く)。故障 β が故障 α を支配するとき、α を検出するテストベクトルは β も検出することができるので、α が検出可能ならば β を考えずに α だけを考えてテスト生成を行なえばよい。

入力ベクトル $\vec{x}_2$ が検出できる故障はすべて入力ベクトル $\vec{x}_1$ でも検出できるとき、入力ベクトル $\vec{x}_1$ は入力ベクトル $\vec{x}_2$ を**支配する**という。さらに、$\vec{x}_1$ で検出できる故障と $\vec{x}_2$ で検出できる故障が同一の場合、これらの入力ベクトルは**等価である**という。最小テスト集合を1つ選ぶ場合には、被支配入力ベクトル $\vec{x}_2$ を故障表から削除できる。

いま、p 個の故障 $\alpha_1, \alpha_2, \cdots, \alpha_p$ を仮定し、これらに対する故障差関数を $F^{\alpha_1}, F^{\alpha_2}, \cdots, F^{\alpha_p}$ とする。故障集合 P を $P = \{\alpha_1, \alpha_2, \cdots, \alpha_p\}$ としたとき、P に対する**テスト集合** T は $\{0,1\}^n$ の部分集合であり、かつ、

$$\forall \alpha \in P \ s.t. \ F^\alpha \neq 0, \exists \vec{x} \in T, F^\alpha(\vec{x}) = 1$$

を満たすものと定義される。本稿で議論する最小のテスト集合を生成する問題は、上記の条件を満たすテスト集合のなかでテストベクトル数が最小のテスト集合 T_{min} を求める問題である。

2.3 共有二分決定グラフ

本稿では、論理関数の内部表現として、共有二分決定グラフ (Shared Binary Decision Diagrams, SBDD's) [7] を採用する。共有二分決定グラフは、二分決定グラフ (Binary Decision Diagrams, BDD's) [8] を拡張し、複数の関数を表すグラフの間においてもサブグラフの共有を行ったものである。二分決定グラフは、シャノンの展開を再帰的に繰り返すことにより得られる二分木のグラフに対し、冗長なノードの削除と、同型なサブグラフの共有を行うことにより、簡約化したものである。共有二分決定グラフは以下の特長を持つ。

- 複数の関数を、少ない記憶量で同時に表すことができる。
- 関数の一致判定をポインタの一致判定だけで行うことができる。

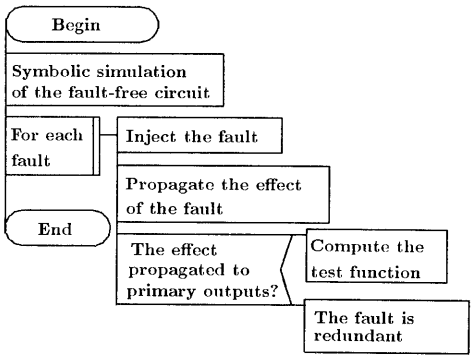

図 1: 記号単一故障伝播法 (SSFP)

- 関数同士の論理演算を、グラフのノード数にほぼ比例する時間で行うことができる。

3 記号故障シミュレーションによる故障表の生成とその簡約化

3.1 記号故障シミュレーションによる故障表の生成

組合せ回路の記号故障シミュレーション [9] は、1) 組合せ回路 (n 入力 m 出力とする) と 2)1 つの仮定故障 α を入力として、正常回路の出力関数 $f_1, f_2, \cdots, f_m$ と故障回路の出力関数 $f_1^\alpha, f_2^\alpha, \cdots, f_m^\alpha$ を計算するものである。本手法では、記号故障シミュレーションの実現手法として記号単一故障伝播法 (Symbolic Single Fault Propagation method, SSFP)(図 1) [10] を採用する。記号故障シミュレーションにより計算した正常回路の出力関数 $f_1, f_2, \cdots, f_m$ と故障 α の下での出力関数 $f_1^\alpha, f_2^\alpha, \cdots, f_m^\alpha$ から、式 (2.1) により故障差関数 F^α を求めることができる。

故障 α に対する故障差関数 F^α は、故障表の故障 α の列を真理値表とする関数である。従って、すべての故障に対し故障差関数を生成すると、故障表を作成できたことになる。このように、本手法では故障表を各故障の故障差関数の並びとして表現する。

3.2 論理関数処理による故障表の簡約化

最小テスト集合を生成するのに要する計算量を削減するため、故障表を簡約化することを考える。故障表の簡約化は以下に示す等価、支配故障および等価、被支配入力ベクトルの削除により行なう。

3.2.1 等価故障の削除

等価故障が存在する場合、その中の 1 つの故障だけを代表故障とすればよい。等価故障の削除は、故障差関数同士の等価性判定により行なうことができる。任意の 2 つの故障に対し、それらの故障差関数同士が等価であれば、それらの故障は等価であるので、いずれか一方の故障を削除する。

3.2.2 支配故障の削除

2 章で述べたように、故障 β が非冗長故障 α を支配するとき、支配故障 β は削除可能である。支配故障の削除は、故障差関数同士の包含性判定により行なうことができる。すなわち、任意の 2 つの故障 α と β に対し、

$$\begin{cases} F^\alpha \to F^\beta & \text{if } F^\alpha \wedge F^\beta = F^\alpha \\ F^\beta \to F^\alpha & \text{if } F^\alpha \wedge F^\beta = F^\beta \\ \text{支配性なし} & \text{上記以外のとき} \end{cases}$$

により支配性を判定し、支配性があれば支配故障を削除する。

3.2.3 等価入力ベクトルの削除

等価入力ベクトル $\vec{x}_1$ と入力ベクトル $\vec{x}_2$ の関係は、各故障 α の故障差関数 F^α を用いて、

$$「\vec{x}_1 \text{ と } \vec{x}_2 \text{ が等価である}」 \Leftrightarrow \forall \alpha, F^\alpha(\vec{x}_2) = F^\alpha(\vec{x}_1)$$

と表すことができる。すべての入力ベクトルの組に対して上式が成立するかどうかを調べれば、等価入力ベクトルを完全に削除することができるが、すべての入力ベクトルの組はテスト対象回路の外部入力数 n に対して 2^{2n} 個あり、非現実的である。そこで、本手法では、論理関数処理により効率的に等価入力ベクトルの削除を行なう。テスト対象回路の外部入力数を n、対象故障数を p とすると、その処理方法は以下のようになる。ここで、1) における論理関数 ER は、

$$ER(\vec{x}_1, \vec{x}_2) = 1 \Leftrightarrow 「\text{入力ベクトル } \vec{x}_1 \text{ と } \vec{x}_2 \text{ が等価である}」$$

を満足する論理関数である。4) における論理関数 E は、

$$E(\vec{x}) = 1 \Leftrightarrow 「\vec{x} \text{ は削除すべき等価入力ベクトルである}」$$

を満足する論理関数である。また、記法として、

$$\exists \vec{x}.F(\vec{x}) \stackrel{\text{def}}{=} \exists x_1 \exists x_2 \ldots \exists x_n.F(\vec{x}),$$
$$\exists x_i.F(\vec{x}) \stackrel{\text{def}}{=} F(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \vee F(x_1, x_2, \cdots, x_{n-1}, 1, x_{i+1}, \ldots, x_n).$$

を用いる。

[等価入力ベクトル削除の手続き]

1) 各故障 α の故障差関数 $F^\alpha(\vec{x})$ と入力ベクトルの変数 $\vec{x}$ を別の変数 $\vec{y}$ で置き換えた関数 $F^\alpha(\vec{y})$ について、以下の論理関数処理により 2n 変数論理関数 $ER(\vec{x}, \vec{y})$ を構成する。
$$ER(\vec{x}, \vec{y}) := \wedge_{i=1}^{p}(F^{\alpha_i}(\vec{x}) \to F^{\alpha_i}(\vec{y})).$$
2) 1) で得られた関数 ER が表す等価入力ベクトルの集合から、$\vec{x} = \vec{y}$ という自明な等価ベクトルを削除する。
$$ER(\vec{x}, \vec{y}) := ER(\vec{x}, \vec{y}) \wedge (\vee_{i=1}^{n} x_i \oplus y_i).$$
3) さらに入力ベクトル対 $(\vec{x}, \vec{y})$ が関数 ER が表す集合に含まれれば、$(\vec{y}, \vec{x})$ もまたその集合に含まれるので、一方を取り除く。
$$ER(\vec{x}, \vec{y}) := ER(\vec{x}, \vec{y}) \wedge (\vec{x} > \vec{y} \text{を表す論理関数}).$$
ただし、$\vec{x} > \vec{y}$ は、$\vec{x}, \vec{y}$ を 2 進数とみなしたときの大小関係を表す。
4) すべての削除すべき等価入力ベクトルの集合を表す関数 $E(\vec{x})$ は、$\vec{y} = (y_1, y_2, \ldots, y_n)$ とすると、
$$E(\vec{x}) := \exists \vec{y}.ER(\vec{x}, \vec{y}).$$
となる。
5) 等価入力ベクトルを削除した故障差関数を F'^α とすると、
$$F'^\alpha := F^\alpha \wedge \overline{E}.$$
により F'^α が求まる。 □

3.2.4 被支配入力ベクトルの削除

被支配入力ベクトルも等価入力ベクトルと同様に削除することができる。支配入力ベクトル $\vec{x}_1$ と被支配入力ベクトル $\vec{x}_2$ の関係は、

$$「\vec{x}_1 \text{ が } \vec{x}_2 \text{ を支配する}」 \Leftrightarrow \forall \alpha F^\alpha(\vec{x}_2) \to F^\alpha(\vec{x}_1),$$

と表せる。したがって、被支配入力ベクトルを削除する手続きは以下のようになる。ただし、論理関数 F'^{α_i} は、等価入力ベクトルを削除したあとの、故障 α_i に対する故障差関数である。

[被支配入力ベクトル削除の手続き]

1) $DR(\vec{x},\vec{y}) := \{\bigwedge_{i=1}^{p}(F'^{\alpha_i}(\vec{x}) \rightarrow F'^{\alpha_i}(\vec{y}))\wedge(\bigvee_{i=1}^{n} x_i \oplus y_i)$ により、支配関係にある入力ベクトルの組の集合を表す論理関数 DR を計算する。
2) DR に以下の演算をほどこし、被支配入力ベクトルの集合を表す論理関数 D を求める。
$$D(\vec{x}) = \exists\vec{y}.DR(\vec{x},\vec{y})$$
3) 各故障 α に対し、等価、被支配入力ベクトルを削除した故障差関数 F''^{α} を求める。
$F''^{\alpha} = F'^{\alpha}\wedge\overline{D}.$ □

4 テスト系列の集合を表す特徴関数を用いた最小テスト集合の生成

本章では、3章の方法により故障差関数の並びとして表現された、簡約化された故障表からテスト系列の特徴関数を得ることにより、最小のテスト集合を生成する方法について述べる。

本手法は、論理関数処理に基づいて最小テスト集合を生成する。また、**独立故障集合** (*independent fault set*) [5] を用いて、探索空間の削減を図る。

4.1 独立故障集合

故障 α と β が同一のテストベクトルで検出不可能なとき、故障 α と故障 β は**独立** (*independent*) であるという [5]。故障差関数を用いると、故障 α と故障 β が独立である条件は、それぞれの故障差関数を F^{α}, F^{β} とすると、

$$F^{\alpha}\wedge F^{\beta} = (\text{恒偽関数})$$

である。従って、本手法では支配故障の判定を行なう際に、すべての独立な故障の組を見つけることが可能である。

また、故障の集合 $P = \{\alpha_0,\alpha_1,\cdots,\alpha_q\}$ が独立であるとは、P に含まれるどの2つの故障 $\alpha_i,\alpha_j (i\neq j)$ も同一のテストベクトルで検出することができないことをいう。ある独立故障集合の大きさと、独立故障集合に含まれるすべての故障を検出する最小のテストベクトルの集合の大きさは等しい。従って、対象故障集合における最大独立故障集合 (*maximum independent fault set*) の大きさは最小なテスト集合の大きさの下界を与える。最大独立故障集合を求める問題はNP困難であることが知られている [11]。ここでは、論理関数処理により独立故障集合を求める発見的手法を示す。(ここで得られる独立故障集合は最大とは限らない。) その処理の流れは以下の通りである。

[独立故障集合生成の手続き]

対象とする故障の集合を $\{\alpha_1,\alpha_2,\cdots,\alpha_p\}$、求める独立故障集合を I、I に含まれる故障を少なくとも1つ検出できる入力ベクトルの集合を表す特徴関数を V とする。

1) 故障をその故障差関数の真理値表密度の小さい順に並べる。このときの故障の順を $\alpha_{i_1},\alpha_{i_2},\cdots,\alpha_{i_p}$ とする。
2) $V := false$, $I := \phi$.
3) 故障 $\alpha_{i_1},\alpha_{i_2},\cdots,\alpha_{i_p}$ に対し、順に以下の処理を行なう。
   3.1) if $V\wedge F^{\alpha_{i_k}} = false$
   then $I := I\cup\{\alpha_{i_k}\}, V := V\vee F^{\alpha_{i_k}}$. □

このアルゴリズムは、$O(p)$ 回 (ただし、p は対象故障数) の故障差関数同士の論理演算を要する。

4.2 最小テスト集合生成アルゴリズム

最小テスト集合生成問題は最小カバー問題に帰着できる。最小カバー問題を解く方法としてはペトリックの方法 [12] などが知られているが、この方法を論理関数処理により行ない、最小テスト集合生成問題を解こうとすると、入力ベクトル数 2^n(ただし n は外部入力数) だけ変数が必要になり非現実的である。そこで本章では、最小テスト集合の各ベクトルに外部入力数分の変数を割り当て、論理関数処理により、最小テスト集合を割り当てる方法を考える。従って、本手法では

$$(\text{外部入力数})\times(\text{最小テスト集合のサイズ})$$

だけ変数が必要である。以下がその処理の流れである。

[最小テスト集合生成の手続き]

1) 故障 $\alpha_i (i = 1,\cdots,p)$ に対し故障差関数 F^{α_i} を求める。
2) 故障差関数が恒偽関数である故障を冗長故障と同定する。
3) 削除できる故障および入力ベクトルがなくなるまで、3章で述べた方法により等価、支配故障、および等価、被支配入力パタンの削除を繰り返す。この結果得られる簡約化された故障差関数を $F_1,F_2,\cdots,F_r$ とする。
4) その故障を検出する入力ベクトルがただ一つしか存在しない故障を同定し、その入力ベクトルをテストベクトルとする。このような入力ベクトルを必須テストという。必須テストをもつ故障の同定は、故障差関数の真理値表密度が $\frac{1}{2^n}$(ただし、n はテスト対象回路の外部出力数) である故障を見つけることにより行なう。
5) 独立故障集合 $I = \{i_1,i_2,\cdots,i_q\}$ を 4.1 節で述べた手続きにより求める。
6) 5) で求めた最小テスト集合の大きさの下界 $low = q$ が下限であるかを、以下の論理式 (4.1) の充足可能性判定により判定する。ここで、F_j^{α} は故障 α に対する故障差関数 F^{α} を $\vec{x}_j$ という変数ベクトルで表現したものとする。すなわち、F_i^{α} と F_j^{α} $(i\neq j)$ は変数のみ異なる論理関数である。充足可能であればその割り当てが最小テスト集合である。
$$\bigwedge_{i=1}^{r}\{\bigvee_{j=1}^{low} F_j^i\} \qquad (4.1)$$
充足不可能であれば、$low := low + 1$ として式 (4.1) に対し充足可能な割当が見つかるまで繰り返す。 □

1)、3) は3章で述べた手続きで行なう。4) において、ある故障の故障差関数の真理値表密度が $\frac{1}{2^n}$ であることは、その故障を検出するテストベクトルがただ1つしか存在しないことを示す。6) の論理関数 (4.1) はテストサイズが low のテスト系列の集合を表す特徴関数である。式 (4.1) の計算では、先に述べた独立故障集合 $I = \{i_1,i_2,\cdots,i_q\}$ に対し、故障 i_k を k 番目のテストベクトルで検出するとあらかじめ決めておくことにより、探索空間の削減を図ることができる。式 (4.1) のかわりに以下の式 (4.1)' を用いる。

$$[\bigwedge_{i\in I}\{F_i^i\vee\bigvee_{j\notin I, j\leq low} F_j^i\}]\wedge[\bigwedge_{i\notin I}\{\bigvee_{j=1}^{low} F_j^i\}] \qquad (4.1)'.$$

5 テスト系列を受理するオートマトンを用いた最小テスト集合の生成

本章では、4章で述べた方法とは別のアプローチによる最小テスト集合生成アルゴリズムとして、テスト系列を受理する有限オートマトンの状態遷移グラフ上での最短経路の探索により最小テスト集合を生成する方法について述べる。

5.1 テスト系列を受理するオートマトン

外部入力数 n の組合せ回路に、p 個の故障 $\alpha_1, \alpha_2, \ldots, \alpha_p$ を仮定し、各故障 α_i の故障差関数を F^{α_i} とする。ただし、これらの故障は冗長故障ではないものとする。このとき、以下のような有限オートマトン (S, I, δ, s_0, F) を考える。

状態集合 $S : S \subseteq \{0,1\}^p$,
入力集合 $I : I = \{0,1\}^n$,
状態遷移関数 $\delta : \delta = (f_1(\vec{s},\vec{x}), f_2(\vec{s},\vec{x}), \ldots, f_p(\vec{s},\vec{x}))$,
ただし、$\vec{s} = (s_1, s_2, \ldots, s_p) \in S$,
$\vec{x} = (x_1, x_2, \ldots, x_n) \in I$,
$f_i(\vec{s},\vec{x}) = s_i \vee (\overline{s_i} \wedge F^i(\vec{x})) = s_i \vee F^i(\vec{x})$.
初期状態 $s_0 : s_0 = \vec{0}$,
受理状態集合 $F : F = \{\vec{1}\}$.
ただし、$\vec{1}$ は、p 個の 1 からなるベクトルを表す。

この有限オートマトンは、テスト対象回路の入力ベクトルを入力とする。状態は、状態ベクトルの各ビットが各故障に対応し、その値の意味は以下の通りである。

$$\begin{cases} s_i = 1 \Leftrightarrow \text{故障 } \alpha_i \text{ が検出済み.} \\ s_i = 0 \Leftrightarrow \text{故障 } \alpha_i \text{ が未検出.} \end{cases}$$

上記のように、初期状態を、どの故障も検出されていない状態、すなわち $\vec{0}$ とし、受理状態を、すべての故障が検出された状態、すなわち $\vec{1}$ とすると、この有限オートマトンは入力されたベクトルの系列がすべての故障を検出するとき、そのときのみ受理する有限オートマトンと考えることができる。

5.2 最小テスト集合生成アルゴリズム

上記のオートマトンが受理する長さ最小の系列を求めることにより、テスト集合の最小化を行なうことができる。本手法も論理関数処理により最小テスト集合を求めるものであるが、必要な変数の数は、

$$(\text{外部入力数}) + (\text{対象故障数}) + 1$$

であり、前章の方法よりも、多くの場合少ない変数で最小化を行なうことができる。最小テスト系列を求めることは、初期状態から、オートマトンの状態遷移グラフを幅優先探索により受理状態を探索していくことにより行なう。外部入力数 n、仮定故障数 p の回路に対し最小テスト集合サイズを求めるアルゴリズムは以下のようになる。ただし、$\vec{x}, \vec{s}, \vec{s'}$ はそれぞれ入力変数ベクトル、現状態変数ベクトル、次状態変数ベクトルである。

[最小テスト集合のサイズを求める手続き]

1) 故障 α_i $(i = 1, \cdots, p)$ に対し、故障差関数 F^{α_i} を生成する
2) 故障差関数が恒偽関数である故障を冗長故障と同定する。
3) 削除できる故障および入力ベクトルがなくなるまで、等価、支配故障および等価、被支配入力ベクトルを削除する。
4) $test\_size := 0$, $from := \bigwedge_{i=1}^{p} \overline{s_i}$, $reached := \bigwedge_{i=1}^{p} \overline{s'_i}$.
5) $reached|_{\vec{s'}=\vec{1}} = 1$ になるまで、4.1)、4.2) を繰り返す。
   5.1) $reached := \exists\vec{x}.\exists\vec{s}.[from \wedge (\bigwedge_{i=1}^{p}(s'_i \equiv f_i(s_i,\vec{x}))]$
   ただし、
   $$f_i(s_i,\vec{x}) = s_i \vee F^{\alpha_i}(\vec{x}),$$
   5.2) $from := (reached|_{\vec{s'}=\vec{s}}) \vee from$,
   $test\_size := test\_size + 1$ □

1)、3) は、3 章で述べた手続きにより行なう。$test\_size$ は探索した状態の初期状態からの経路長を表し、テスト集合のサイズに対応する。$from$ は、既到達状態の集合を特徴関数として表したものである。すなわち、

$$from(\vec{s}) = 1 \Leftrightarrow \text{「}\vec{s}\text{が既到達状態である」}$$

を満足する関数である。5) では、オートマトンの状態遷移グラフ上の探索を行なう。5.1)、5.2) で、既到達状態集合 $from$ から 1 回以下の遷移で到達可能な状態集合 $reached$ を計算する。5.1)、5.2) の手続きを、状態集合 $reached$ に受理状態 $\vec{1}$ が含まれるまで繰り返し行なう。最初に受理状態が含まれたときの $test\_size$ の値が最小テスト集合のサイズを表す。5.2) の論理和演算は、実際には generalized cofactor を用いて行ない、関数 $from$ の記憶領域の削減を図る [13]。

最小テスト集合を生成するには、上記の最小テスト集合のサイズを求めるアルゴリズムの 5) において、1 遷移ごとに新たに到達した状態の集合 $new[i](i = 0, 1, \ldots, test\_size)$ を求めておき、以下の方法でテスト集合を生成する。

[最小テスト集合生成アルゴリズム]

1) $reached[test\_size] := \vec{1}, length := test\_size$.
2) $length < 1$ になるまで、2.1) ～ 2.3) を繰り返す。
   2.1) 状態 $reached[length]$ に 1 遷移で到達可能な状態集合を表す関数 $from$ を求める。
   $from := new[length] \wedge$
   $$[\exists\vec{x}.\bigwedge_{i=1}^{p}(s'_{0i} \equiv f_i(s_i,\vec{x})].$$
   ここで、s'_{0i} は状態 $reached[length]$ の第 i ビットの値である。
   2.2) $from$ が表す状態集合に含まれる状態を 1 つ選び、$reached$ とする。
   2.3) $length := length - 1$.
3) $reached[i](i = 1, \cdots, test\_size)$ は検出故障の系列であるから、故障差関数を用いてそれらの故障を検出するテストベクトルを生成する。 □

5.3 独立故障集合を利用した探索空間の削減

4 章で述べた独立故障集合の概念は、本章の最小化手法における探索空間削減にも用いることができる。5.2 節の最小化アルゴリズムは、状態遷移グラフ上の全遷移を受理状態に到達するまで幅優先で探索するものであった。独立故障集合内の各故障は、互いに相異なるテストベクトルでのみ検出可能であるから、ある独立故障集合 $\{\alpha_1, \alpha_2, \cdots, \alpha_q\}$ に対し、1 遷移目は故障 α_1 を、2 遷移目は故障 α_2 を、q 遷移目は故障 α_q を、それぞれ検出するような経路のみを探索しても、必ず最小のテスト系列を得ることが保証される。この場合、独立故障集合内の故障に対しては、テスト系列を受理するオートマトンの状態ベクトルのビットを割り当てる必要はない。さらに、支配故障解析の際に、任意の 2 つの故障に対する独立性が判定できるので、その情報を利用して、i 遷移目における検出対象である独立故障集合に含まれる故障 α_i と独立な故障については、i 遷移目の探索では、考慮しなくてよい。したがって、到達可能状態集合の計算をより高速に行なうことができる。

6 実験結果

4、5 章で述べたアルゴリズムに基づく最小テスト集合を生成する 2 つのプログラムを Sun SPARCstation2 上で C 言語を用いて実現した。論理関数の処理には SBDD パッケージ [7] を使用した。SBDD のノード数は最大 100 万ノード (約 20MB) とした。実験結果を表 1 に示す。表中の「circuit」は回路名である。回路名 adder$k$ は k ビット桁上げ伝播加算器、mult$k$ は k ビット配列型乗算器、dec8 は 8 ビットデータセレクタ、enc8 は 8 ビットプライオリティエンコーダ、74181 は 4 ビット ALU である。また、sao2 は MCNC ベンチマーク回路から、回路名が c で始まる回路は ISCAS85 ベンチマーク回路 [14] から、それぞれ選んだも

表 1: 実験結果

circuit	# of gates	# rep. faluts	lower bound	min. size	CPU(sec.) Sec.4	CPU(sec.) Sec.5
adder1	6	6	6	6	5.3	4.3
adder4	24	24	6	6	9.3	9.5
mult4	88	96	7	8	669.7	-
dec8	17	40	24	24	13.6	11.0
enc8	22	32	15	16	8.7	5.8
sao2	101	60	45	45	13.7	9.7
74181	58	81	9	-	-	-
c432	160	152	22	-	-	-

のである。「# of gates」は、対象回路の外部入力数を示す。「# rep. faults」は、全仮定故障である、回路のすべての端子の 0/1 縮退故障に対し、3 章で述べた等価故障および支配故障の削除を行なった後の対象故障数を示す。「lower bound」は、4 章で述べたできるだけ大きな独立故障集合を求めるアルゴリズムを用いて得られた独立故障集合の大きさにより評価した最小テスト集合の大きさの下界である。「min. size」は、最小テスト集合の大きさである。「CPU」は、「Sec.4」は 4 章で述べた最小テスト集合生成手法の、「Sec.5」は 5 章で述べた最小テスト集合生成手法の、実行時間をそれぞれ示す。ただし、「Sec.5」では最小テスト集合の大きさを求めるのみで、テストベクトルの生成は行なっていない。表中の横線は、実行途中で二分決定グラフのノードあふれが起こり、結果が得られなかったことを示す。

表 1より、まず、独立故障集合の概念を用いた下界は最小サイズに非常に近いことがわかる。74181 や c432 については、最小解は得られていないが、他の圧縮されたテスト集合の大きさを評価するという点では、この下界をもとに評価を行なってもよいと思われる。最小化については、ゲート数 100、等価、および支配故障解析後の故障数 100 程度までの回路に対しては、最小テスト集合を生成できる場合があることが分かった。2 つの最小化手法については、テスト系列を受理するオートマトンを利用する方法ではテスト集合のサイズだけでテストベクトルの生成は行なっていないので直接比較はできないが、5 章のアルゴリズムの方が実験を行なったほぼすべての回路について高速であった。また、いずれの手法とも、独立故障集合の利用により必要な BDD ノード数が 1/100 程度になり、探索空間の削減手法として有効であることが分かった。

7 おわりに

論理関数処理による組合せ回路に対する最小テスト集合を生成する 2 つの方法を提案した。実験の結果、ゲート数 100、等価、および支配故障解析後の故障数 100 程度の回路に対しては、最小テスト集合を生成できる場合があることが分かった。5 章の手法については、独立故障集合による探索空間の削減は極めて有効であるといえる。今後は、5 章の手法でテストベクトルの生成まで行なうプログラムを実現し、さらに、独立故障集合以外の探索空間削減手法についても考えたい。また、5 章の方法では、故障の順序づけにより数倍程度 BDD の使用ノード数が変化することが簡単な実験から分かっているので、様々な故障の順序づけについて実験を行ないたい。さらに、最小テスト集合が求められた回路に対して考察を行ない、最小テスト集合内で同時に検出される故障の性質など、回路と最小テスト集合の関係を明らかにし、より高速に最小テスト集合を生成するための指針を得たいと考えている。

謝辞

共有二分決定グラフに基づく論理関数処理プログラムを提供して頂いた湊真一氏 (現在 NTT) に感謝致します。また、高木直史助教授をはじめ、御討論頂きました本学矢島研究室の皆様に感謝致します。

参考文献

[1] O. H. Ibarra and S. K. Sahni. Polynomially complete fault detection problems. *IEEE Trans. Comput.*, C-24(3), March 1975.

[2] H. Fujiwara. FAN: a fanout-oriented test pattern generation algorithm. In *Proc. Int. Symp. on Circuits and Systems*, pages 671–675, June 1985.

[3] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: a highly efficient automatic test pattern generation system. *IEEE Trans. Computer-Aided Design*, CAD-7(1):126–137, January 1988.

[4] E. B. Eichelberger and T. W. Williams. A logic design structure for LSI testability. In *Proc. 14th Design Automat. Conf.*, pages 28–33, June 1977.

[5] S. B. Akers, C. Joseph, and B. Krishnamurty. On the role of independent fault sets in the generation of minimal test sets. In *Proc. Int. Test Conf.*, pages 1100–1107, 1987.

[6] J. A. Abraham. Fault modeling in VLSI. In *Advances in CAD for VLSI, vol.5, VLSI Testing*, 1986.

[7] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proc. 27th Design Automat. Conf.*, pages 52–57, June 1990.

[8] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35(8):677–691, August 1986.

[9] K. Cho and R. E. Bryant. Test pattern generation for sequential MOS circuits by symbolic fault simulation. In *Proc. 26th Design Automat. Conf.*, pages 418–423, June 1989.

[10] K. Ioki, N. Ishiura, and S. Yajima. Test generation for combinational logic circuits using shared binary decision diagrams. In *Proc. 38th Annual National Conf. of IPSJ.*, March 1989. 2S-5 (in Japanese).

[11] B. Krishnamurthy and S. B. Akers. On the complexity of estimating the size of a test sets. *IEEE Trans. Comput.*, C-33(7):750–753, August 1984.

[12] S. R. Petrick. On the minimization of boolean funtions. In *Proc. International Conference on Information Processing 1959*, pages 422–423, 1960.

[13] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A.Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDD's. In *Proc. Int. Conf. Computer-Aided Design*, November 1990.

[14] F. Braglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. In *Proc. Int. Symp. on Circuits and Systems*, June 1985.