LC-3　テストパターン中の特定ビットにおけるドントケア判定法について

宮瀬紘平[1], 梶原誠司[1,2], ポメランツ イリス[3], レディ スダーカ[4]
1: 九州工業大学大学院情報工学研究科
2: 九州工業大学マイクロ化総合技術センタ
3: 電気コンピュータ工学科, パデュー大学
4: 電気コンピュータ工学科, アイオワ大学

1. はじめに

論理回路に対するATPGにおいて、ある故障を対象に生成されたテストベクトルは、一般に未設定の入力値（以下Xと表す）を含んでいる．しかし、ATPGで対象とした故障でなくても故障シミュレーションにより検出できる場合があるので、Xにはランダムに値が設定されるか、静的・動的圧縮手法等を用いて値が割当てられる．したがって生成されたテスト集合の全ての入力値は0または1に決まっている場合が多く、一旦生成されたテスト集合に別の特性を持たせることは難しい．一方で、もしテスト集合がXを含むなら、Xへの論理値の割当を通じて、そのテスト集合を例えば、テスト時の消費電力を抑える[1]、テストデータ量を削減する[2]など、近年のLSIテストの問題を緩和するようにできる．

テスト集合の全ての入力値が0または1に決まっている場合でも、入力値の中には逆の論理値に変えても故障検出率に影響を及ぼさないものがある．そのような入力値はドントケアであり、未設定入力値のXと同様に扱うことができる．テスト集合中のX（ドントケア）を判定する手法は[3]に提案されている．一般に、テスト集合に含まれるXの位置については組合せが数多く存在するが、[3]ではXの数が多くなるようなものをみつけている．しかしながら、テスト時の消費電力削減やテスト圧縮などのXへの値の再割当てを考えるとき、テスト集合中のXには、その目的に対して有効なものとそうでないものが存在する．

そこで本論文では、全ての入力値が 0 または 1 に設定されたテスト集合 T に対して、以下のような特徴を持った X を含むテスト集合 T'を求める手法を提案する．

(1) T'は Tを被覆する．
(2) T'と Tの縮退故障検出率は等しい．
(3) T'は指定されたビット上にできるだけ多く X を含む．

ここで、テスト集合Tは組合せ回路の単一縮退故障に対して生成されたものとし、Xとなりうるビットについては、テスト集合中の任意のビットをあらかじめ指定可能とする．

提案手法において、Xの判定には、故障シミュレーションとATPGアルゴリズムの含意操作・正当化操作と同様の処理を用いるが、ATPGで行われるような探索を行わないので、高速に計算できる．以下では、まずテスト集合中のXについて、[3]の手法とともに概説し、次に提案手法の特徴と処理手順について述べる．最後にISCASベンチマーク回路に対する実験により、提案手法の有効性を示す．

2. テスト集合中のドントケア

まず、図1と表1、2を用いてXの例を示す．表1は、回路Cに対して生成されたテスト集合Tである．テストベクトルt_1は、故障a/0、b/0、c/1を検出する．ここで、s/vは信号線sのv縮退故障を表している．a/0は、t_1によって必ず検出されなければならないのに対して、c/1は、t_3でも検出されるのでt_1で検出する必要はない．それ故、t_1の入力cはXとすることができる．表2で示すテスト集合T'は、回路Cの全ての故障を検出できる．以下、本論文では、元のテスト集合をT、提案手法によって得られたXを含むテスト集合をT'とする．また、t_iとt_i'は、それぞれテスト集合TとT'に含まれるテストベクトルを表す．

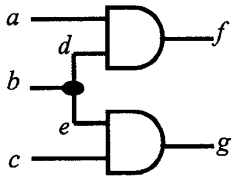

図 1: 回路 C

表1: 初期テスト集合 T

	abc
t_1	110
t_2	101
t_3	010
t_4	011

表2: Xを含むテスト集合 T'

	abc
t_1'	11x
t_2'	101
t_3'	x10
t_4'	011

表3: 表2と異なる T' の例

	abc
t_1'	110
t_2'	101
t_3'	01x
t_4'	x11

表4: マスク集合 M

	abc
t_1	xff
t_2	fxx
t_3	fxx
t_4	xxf

図２に回路のネットリストとテスト集合Tから、Xを含むテスト集合T'を求める手順を示す．最初に、故障シミュレーションにより、必須故障に関する情報を収集する．必須故障とは、テスト集合中のある一つのテストベクトルによってのみ検出でき、その他のテストベクトルでは検出できない故障のことである．次に、各t_iの必須故障を検出するためのテストベクトルの入力値を固定し、その他の入力値をXとする．こうして得られたテスト集合をT'とする．このT'によって検出されない故障も存在するので、それらを検出

```
---------------------------------------------------
Basic procedure X-search(C,T)
    Circuit C; Test set T;
{
  fault_simulation(T);
  clear_fault_list();
  for each test pattern t_i in T {
      F=collect_essential_fault(t_i);
      t_i'=find_value(F);
  }
  fault_simulation(T');
  for each test pattern t_i in T {
      G=collect_undetected_fault(t_i);
      t_i'+=find_value(G);
      fault_simulation(t_i');
  }
  return T' composed of t_i';
}
---------------------------------------------------
```

図 2: 基本処理手順

するような入力値をさらに固定する．結果として、故障の検出に必要なかった入力がXとなる．ベンチマーク回路に対する実験では、圧縮技術を用いて生成したテスト集合でも約47%入力値をXにできることが報告されている[3].

一般に、テストベクトル中に判定されるXの位置の組合せは数多く存在し、一意に決定されない．表3に、表2とXの位置が異なるT'を示す．表3のテスト集合も、Tと故障検出率が同じであるが、表2のテスト集合とXのビット位置が異なる．Xになることが望まれる入力の位置は、Xへの値の再割当の目的に依存するが、[3]の手法はXを判定するビット位置が考慮されていない．そこで、テスト集合中のビット位置を特定した上で、Xを判定する手法を示す．

3. 特定入力に対するX判定手法

Xになることが望まれるビット位置を特定するため、本手法では表4に示す形式のマスクベクトルの集合であるマスク集合を使用する．マスク集合は、テスト集合の各入力値に対応しており、マスク集合中のxはXになることが望まれるビット位置を、fはそうでないビット位置を表す．

図3に、提案手法の処理手順を示す．まず、故障シミュレーションを行って、それぞれのテストベクトルの必須故障を求める．次に、初期テスト集合とマスク集合により生成された、Xとならない入力値のみを用いて構成したテスト集合T^*で故障シミュレーションを行い、検出できた故障を、故障リストから取り除く．これは、マスク集合に基づいて固定した入力値のみで検出できる故障があるための処置である．残りの故障数を少なくできるので、最終的に、Xと判定されるビットを増やすことができる．

以降、T^*に[3]のX判定手法を適用することにより、必須故障を検出するXを含んだテスト集合T^{**}を生成し、T^{**}で検出できない故障を検出する処理を続けることで、求めるテスト集合T'が得られる．例えば、図３に示す処理手順により、表1のTと表4のMから表3のT'が求められる．

```
---------------------------------------------------------
Procedure X-search on specific bits(C, T, M)
    Circuit C; Test set T; Mask file M;
{
  fault_simulation(T);
  clear_fault_list();
  T*=fixed_bit(T, M);
  fault_simulation(T*);
  F=collect_undetected_fault();
  for each test pattern t_i in T {
      G=collect_essential_fault(t_i)∩F;
      t_i**= (t_i*)+find_value(G);
  }
  fault_simulation(T**);
  for each test pattern t_i in T {
      H=collect_undetected_fault(t_i);
      t_i**+=find_value(H);
      fault_simulation(t_i**);
      t_i' = t_i**
  }
  return T' composed of t_i';
}
---------------------------------------------------------
```

図 3：提案手法の処理手順

4. 実験結果

提案手法をPentium3 700MHz, メモリ384Mの計算機上でC言語を使用して実装し、ISCASベンチマーク回路に対して実験を行った．表5と表6に、それぞれ圧縮されていないテスト集合と圧縮されたテスト集合に対する実験結果を示す．実験ではテスト集合中の50%のビットをランダムに特定し、Xを判定した．提案手法の有効性を示すために、Xとなるビットを特定しない[3]の手法で得られた結果を基準に比較を行った．表中の"#inc/#fix[3]"の欄は、特定したビットにおいて、[3]の手法ではXと判定されなかったが、提案手法によってXと判定したビット数を表す．"%" の欄は、[3]の結果に対する提案手法の改善率を示す．表中の最後の欄は、CPU時間を表している．

提案手法により、特定したビット中のXの数は増加させることができた．特定するビットをより限定すれば（マスク集合におけるxの割合を減らせば）、提案手法の効果はより大きくなることも確認している．また、特定したビットにおいて、[3]の手法ではXと判定されたが、提案手法によってXと判定されなかったビットも存在したが、その数はほとんどの回路で増加分の5%以下であり、最大でも13%であった．

5. まとめ

本論文では、テスト集合中の特定ビット上にできるだけ多くのXを判定する手法を提案した．実験では、従来手法と比較して、特定したビット上のXが増加することを示した．テスト集合中のXを利用してテストデータ量を削減する手法や、テスト時の消費電力を削減する手法はすでに発表されており、本論文で提案された手法を使用することで、さらに効果を高めることが期待できる．テストデータ量の削減については、予備実験で本手法を用いた場合従来手法の10%以上の改善が得られているが、これらについては今後別の機会で報告する予定である．

参考文献

[1] R. Sankaralingam, R. R. Oruganti, N. A. Touba, "Static Compaction Techniques to Control Scan Vector Power Dissipation," 18th VLSI Test Symposium, pp. 35-40, 2000.

[2] B. Koenemann, et. al., "A Smart BIST Variant Guaranteed Encoding," 10th Asian Test Symposium, pp. 325-330, Nov. 2001.

[3] S. Kajihara, K. Miyase, "On Identifying Don't Care inputs of Test Patterns for Combinational Circuits," ICCAD-2001, pp. 364-369, Nov. 2001.

表5：圧縮されていないテスト集合に対する実験結果

circuit	#tests	#inc / #fix[3]	%	time(sec)
s13207	586	908 / 8233	11.03	16.12
s15850	500	2375 / 9215	25.77	17.11
s35932	76	6085 / 10156	59.92	11.89
s38417	1243	1877 / 36703	5.11	128.44
s38584	854	844 / 23858	3.54	89.48

表6：圧縮されたテスト集合に対する実験結果

circuit	#tests	#inc / #fix[3]	%	time(sec)
s13207	235	493 / 6894	7.15	8.05
s15850	97	1679 / 7098	23.65	5.46
s35932	12	766 / 6938	11.04	9.13
s38417	87	641 / 20922	3.06	16.14
s38584	114	743 / 17335	4.29	18.73