日本大学大学院 生産工学研究科

2014 年度

博士学位申請論文

スキャンテストにおけるコスト及び消費電力削減のためのテスト生成法に関する研究

学位申請者　山崎 紘史

本論文は日本大学大学院生産工学研究科に博士（工学）授与の要件として提出した博士論文である.

山崎 紘史

審査委員： 細川 利典 教授
三井 和男 教授
角田 和彦 教授
井口 幸洋 教授（明治大学）

論文要旨

近年，半導体の微細化技術の進歩に伴い，超大規模集積回路(Very Large Scale Integrated circuits：VLSI)の集積度が増大している．また，設計自動化技術の進歩により，大規模なデジタルシステムを VLSI 上に実装することが可能となった．VLSI 回路は，社会においても幅広く利用されており，製造された VLSI に故障(物理的な欠陥)が無いことを保証しなければならない．このため，VLSI のテスト設計が非常に重要になっており，その自動化は必要不可欠になっている．VLSI のテスト設計にはテスト生成(Automatic Test Pattern Generation：ATPG)とテスト容易化設計(Design For Testability：DFT)の 2 つが挙げられる．テスト生成とは VLSI 製造後の出荷検査に用いるテストパターンを生成することをいう．また，テスト容易化設計とはテストパターンの生成を容易にするために VLSI の回路構造を変更することをいう．

VLSI のテストでは，製造された VLSI に対してテストパターンを外部入力(Primary Input：PI)に印加し，その出力応答を外部出力(Primary Output：PO)で観測する．このとき，外部出力の観測値とテストパターンによる出力期待値を比較することで VLSI 内部の故障の有無を判定する．そのため，テスト生成では，VLSI 内部に欠陥をモデル化した故障モデルを仮定し，その故障モデルの故障影響を外部出力で観測できるテストパターンを生成する．このとき，故障検出効率という概念を導入し，出荷した VLSI の市場不良率をできるだけ低減するには，故障検出効率が高い(例，99%以上)テストパターンを用意する必要がある．

VLSI の微細化技術の進歩により，VLSI の集積度が増加している．また，テストパターン数はゲート数の 0.5 乗から 1.5 乗に比例して増加すると報告されている．VLSI の微細化により従来の縮退故障モデルのテストパターンでは検出困難なタイミング遅延を伴う欠陥が存在する．そのため，縮退故障モデルのテストパターンの他に遷移故障モデルやパス遅延故障モデルなどのテストパターンが必要である．テストパターン数の増加に伴いテスト実行時間が増加し，テストコストの増大につながる．このことから，VLSI におけるテスト実行時間の削減が重要である．

一方，VLSI の低消費電力化設計に伴い，実速度スキャンテストにおけるテスト時消費電力の増大が問題となっている．実速度スキャンテスト特有の消費電力として，キャプチャ時消費電力とシフト時消費電力が挙げられる．キャプチャ時消費電力は，テスト応答をフリップフロップ（Flip-Flop：FF）へ格納するキャプチャ動作時に発生する．シフト時消費電力は，スキャン FF へのテストパターンの印加とテスト応答の観測を行うシフト動作時に発生する．過度なキャプチャ時消費電力による問題として，電圧降下(IR ドロップ)による誤テストが挙げられる．また，過度なシフト時消費電力による問題として，発熱による回路の熱破壊が挙げられる．そのため，VLSI のテスト時消費電力の増大は歩留まり低下の原因の一つとして挙げられる．したがって，歩留まりの損失を抑制するために VLSI のテスト時消費電力の削減が重要である．

本研究は，テストパターン数削減のためのドントケア判定法と，キャプチャ時消費電力削減のためのマルチサイクルキャプチャ・テスト生成法を提案する．

本論文は序論および結論を含め 7 つの章から構成される．

第 1 章は序論である．本研究の目的と意義および背景について述べ，本論文の概説を行っている．

第 2 章では，故障モデル，テスト生成，スキャン設計，スキャン設計回路における遷移故障のテスト方式などの，VLSI のテストに関する技術について概説する．

第 3 章では，相補型金属酸化膜半導体(Complementary Metal Oxide Semiconductor：CMOS)回路の消費電力と，VLSI のテスト時消費電力その影響として，キャプチャ時消費電力とシフト時消費電力についてまとめている．また VLSI の消費電力の見積り手法として広く用いられている重み付き信号遷移(Weighted Switching Activity：WSA)について概説する．

第 4 章では，ドントケア，故障シミュレーション，テスト圧縮などの，VLSI のテストコスト削減に関する技術を概説する．テスト圧縮には，テスト生成中にテスト圧縮を行う動的圧縮と，テスト生成後のテスト集合に対してテスト圧縮が存在する．本章では，故障シミュレーションに

基づく静的圧縮手法と，テストパターン中のドントケアに基づく静的圧縮手法について概説する．

第 5 章では，テスト圧縮を考慮したドントケア判定法を提案する．一般に，生成されたテストパターンの各外部入力値は，全て 0 または 1 の論理値(ケアビット)に設定されている．しかしながら，生成されたテストパターンの中には，逆の値に変更しても故障検出率が低下しない外部入力値が存在する．このような外部入力値をドントケアという．テスト集合からドントケアを判定する技術には，ドントケア判定法が提案されている．

過去に提案されたドントケア判定法では，テスト集合中のドントケアの最大化や，テスト時消費電力の分野を考慮したドントケア判定法が提案されている．

しかしながら，過去に提案されたドントケア判定法では，各外部入力に対してはドントケア数の均一化を考慮していない．そのため，テスト圧縮の分野において，特定の外部入力にケアビットが集中するとテスト圧縮効率が低下する可能性が存在する．本論文では，各外部入力のドントケア数を均一にし，ドントケア数を最大化するテスト圧縮に効果的なドントケア判定法を提案し，ISCAS’89，ITC’99 ベンチマーク回路に対して実験を行った．

テスト圧縮を未適用な初期テスト集合に対する実験結果では，従来のドントケア判定法と比較して，ドントケア判定率は平均約 1%の増加，外部入力のドントケア分散は平均約 30%の削減を達成した．また，テスト圧縮後のテストパターン数に関しては，従来のドントケア判定法と比較して，平均約 12%の削減を達成した．また，テスト圧縮を適用済みの初期テスト集合に対する実験結果では，従来のドントケア判定法と比較して，ドントケア判定率は平均約 1%の増加，外部入力のドントケア分散は平均約 10%の削減を達成した．テスト圧縮後のテストパターン数に関しては，従来のドントケア判定法と比較して，平均約 3%の削減を達成した．

第 6 章では，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法を提案する．フルスキャン設計をした回路では，一

般的なテスト生成は機能動作を考慮せず，テスト生成の容易性を優先してテスト生成を行う．このため，生成されたテストパターンは，スキャン FF の状態が機能動作では起こりえない状態(無効状態)となる可能性がある．無効状態では，回路内の多くの信号線に遷移を発生させ WSA を増加させている可能性がある．

一方，過去の文献において，一般的なテスト生成で生成した遷移故障モデルのテストパターンを印可した後に，キャプチャ動作を 20 サイクル行うことで WSA が減少することが報告されている．また，テスト生成時に，時間展開モデル(マルチサイクルキャプチャ・テスト生成モデル)を用いてテスト生成を行うことで，テスト不可能故障を同定する手法も提案されている．

本論文では，この現象に着目し，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法を提案し，ISCAS'89，ITC'99 ベンチマーク回路に対して実験を行った．提案手法の 20 時間展開モデルにおいて WSA 閾値 50%では最大 78%(平均 45%)，WSA 閾値 60%では最大 85%(平均 62%)，WSA 閾値 70%では最大 100%(平均 72%)，WSA 閾値 80%では最大 100%(平均 75%)のアンセーフ故障数の削減ができた．また，他のテスト生成法との比較実験として，平均約 22%から 40%のアンセーフ故障数の削減が確認できた．

第 7 章は結論であり，以上の研究成果を述べるとともに，今後の研究課題について議論している．

関連発表一覧

● 学術論文誌

1. Hiroshi Yamazaki, Motohiro Wakazono,Toshinori Hosokawa and Masayoshi Yoshimura, " A Test Compaction Oriented Don't Care Identification Method Based on X-bit Distribution," IEICE TRANS.INF. & SYST., VOL.E96-D, NO.9 SEPTEMBER 2013, pp.1994-2002, Sep.2013

● 国際会議（査読付き）

1. Hiroshi Yamazaki, Motohiro Wakazono,Toshinori Hosokawa and Masayoshi Yoshimura, "A Test Compaction Oriented Don't Care Identification Method," The 12th Workshop on RTL and High Level Testing, pp.69-76, Nov. 2011.
2. Hiroshi Yamazaki, Motohiro Wakazono,Toshinori Hosokawa and Masayoshi Yoshimura, " A Don't Care Identification Method for Test Compaction," 2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, pp.215-218, Apr. 2013.

● 研究会報告

1. 山崎紘史, 細川利典, 吉村正義, "ケアビット分布制御ドントケア抽出　～ キャプチャ消費電力削減への適用 ～," 信学技報, vol. 111, no. 100, pp. 23-28, 2011 年 6 月.
2. 山崎紘史, 細川利典, 吉村正義, "ケアビット分布制御ドントケア抽出法," 第 44 回日本大学生産工学部学術講演会, 2011 年 12 月.
3. 山崎紘史, 細川利典, 吉村正義, "テスト圧縮指向ドントケア判定法の解析," 第 4 回 VLSI テストセミナー, 2012 年 2 月.
4. 山崎紘史, 細川利典, 吉村正義, "ケアビット数を考慮したテスト圧縮指向ドントケア抽出法," 第 67 回 FTC 研究会, 2012 年 7 月.
5. 山崎紘史, 細川利典, 吉村正義, "テスト圧縮指向ドントケア抽出を用いた静的テスト圧縮の評価," 日本大学生産工学部第 45 回学術講演会, 2012 年 12 月.
6. 川連裕斗，平井淳士，高橋慶安，山崎紘史，細川利典，吉村正義, "マルチサイクルキャプチャテスト生成を用いた低消費電力指向テスト生成法," 日本大学生産工学部第 46 回学術講演会, 2013 年 12 月.
7. 山崎紘史，川連裕斗，西間木淳，平井淳士，細川利典，吉村正義，山崎浩二, "マルチ

サイクルキャプチャテスト生成を用いた低消費電力指向遷移故障テスト生成法,” 信学技報, vol. 113, no. 430, pp. 61-66, 2014 年 2 月.

8. 山崎紘史，西間木淳，細川利典，吉村正義，山崎浩二, “マルチサイクルキャプチャテストの消費電力評価,” 第 71 回 FTC 研究会, 2014 年 7 月.
9. 山崎紘史, 細川利典, 吉村正義, “キャプチャ消費電力削減のためのマルチサイクルキャプチャテスト生成法,” デザインガイア 2014 -VVLSI 設計の新しい大地-, pp.191-196, 2014 年 11 月.

目次

Study on Test Generation Method to Reduce Costs and Power Dissipation in Scan Testing

Hiroshi Yamazaki

In recent years, the growing density and complexity of very-large-scale integration circuits (VLSI) has caused an increase in the numbers of test patterns and test power dissipation. Test patterns for not only stuck-at faults but also transition faults are required for VLSI testing. Test cost is generally proportional to the number of test patterns. It is important to reduce the number of test patterns for test cost reduction.

In at-speed scan testing of deep submicron era, high power dissipation can occur when the response to a test pattern is captured by flip-flops, resulting in excessive IR drop, which may cause significant capture-induced yield loss.

A test compaction technique is one to reduce the number of test patterns. A don't care based test compaction method reduces the number of test patterns by merging a test pattern with other compatible test patterns. Some of specified primary input (PI) and pseudo primary input (PPI) values in a test set may be able to be changed to opposite logic values without losing fault coverage. Such PI and PPI values can be regarded as X-bits. X-identification methods to identify many don't care inputs of test patterns in a given test set have been proposed. However, conventional X-identification techniques are less effective for application-specific fields such as test compaction because the X-bits concentrate on particular primary inputs and pseudo primary inputs.

Test generation methods for transition faults based on broadside testing have been proposed. It is considered that structural test generation without considering functional operations of VLSI, causes high capture power dissipation in broadside testing. Automatic test pattern generator (ATPG) generally generates test patterns without considering functional operations for full scan circuits. Thus, test patterns generated by ATPG cause transitions on many lines in circuits. On the other hand, it was reported that capture behavior for cycles more than four could drastically reduce capture power dissipation. It is important to generate test patterns with sequential operations fo multi cycles.

This dissertation proposes an X-identification method for test compaction and a test generation method using multi-cycle model to reduce capture power dissipation.

第 1 章

序論

近年，半導体の微細化技術の進歩に伴い，超大規模集積回路(Very Large Scale Integrated circuits：VLSI)の集積度が増大している．また，設計自動化技術の進歩により，大規模なデジタルシステムを VLSI 上に実装することが可能となった．VLSI 回路は，社会においても幅広く利用されており，製造された VLSI に故障(物理的な欠陥)が無いことを保証しなければならない．このため，VLSI のテスト設計が非常に重要になっており，その自動化は必要不可欠になっている．VLSI のテスト設計にはテスト生成[1](Automatic Test Pattern Generation ： ATPG)とテスト容易化設計[1](Design For Testability：DFT)の 2 つが挙げられる．テスト生成とは VLSI 製造後の出荷検査に用いるテストパターンを生成することをいう．また，テスト容易化設計とはテストパターンの生成を容易にするために VLSI の回路構造を変更することをいう．

VLSI のテストでは，製造された VLSI に対してテストパターンを外部入力(Primary Input ： PI)に印加し，その出力応答を外部出力(Primary Output：PO)で観測する．このとき，外部出力の観測値とテストパターンによる出力期待値を比較することで VLSI 内部の故障の有無を判定する．そのため，テスト生成では，VLSI 内部に欠陥をモデル化した故障モデル[1]を仮定し，その故障モデルの故障影響を外部出力で観測できるテストパ

ターンを生成する．このとき，故障検出効率という概念を導入し，出荷したVLSIの市場不良率をできるだけ低減するには，故障検出効率が高い(例，99%以上)テストパターンを用意する必要がある．

VLSIの微細化技術の進歩により，VLSIの集積度が増加している．また，テストパターン数はゲート数の0.5乗から1.5乗に比例して増加すると報告されている[2]．VLSIの微細化により従来の縮退故障モデル[1]のテストパターンでは検出困難なタイミング遅延を伴う欠陥が存在する[3][4]．そのため，縮退故障モデルのテストパターンの他に遷移故障モデル[1]やパス遅延故障モデル[1]などのテストパターンが必要である．テストパターン数の増加に伴いテスト実行時間が増加し，テストコストの増大につながる．このことから，VLSIにおけるテスト実行時間の削減が重要である．

一方，VLSIの低消費電力化設計に伴い，実速度スキャンテストにおけるテスト時消費電力の増大が問題となっている[5]．実速度スキャンテスト特有の消費電力として，キャプチャ時消費電力[6]とシフト時消費電力[6]が挙げられる．キャプチャ時消費電力は，テスト応答をフリップフロップ（Flip-Flop：FF）へ格納するキャプチャ動作時に発生する．シフト時消費電力は，スキャンFFへのテストパターンの印加とテスト応答の観測を行うシフト動作時に発生する．過度なキャプチャ時消費電力による問題として，電圧降下(IRドロップ)による誤テスト[7][8]が挙げられる．また，過度なシフト時消費電力による問題として，発熱による回路の熱破壊[9]が挙げられる．そのため，VLSIのテスト時消費電力の増大は歩留まり低下の原因の一つとして挙げられる．したがって，歩留まりの損失を抑制するためにVLSIのテスト時消費電力の削減が重要である．

本研究は，テストパターン数削減のためのドントケア判定法と，キャプチャ時消費電力削減のためのマルチサイクルキャプチャ・テスト生成法を提案する．

本論文は序論および結論を含め7つの章から構成される．

第1章は序論である．本研究の目的と意義および背景について述べ，本論文の概説を行っている．

第2章では，故障モデル，テスト生成，スキャン設計，スキャン設計回路における遷移故障のテスト方式などの，VLSIのテストに関する技術に

ついて概説する．

第 3 章では，相補型金属酸化膜半導体(Complementary Metal Oxide Semiconductor：CMOS)回路の消費電力と，VLSI のテスト時消費電力とその影響についてまとめている．また VLSI の消費電力の見積り手法として広く用いられている重み付き信号遷移(Weighted Switching Activity：WSA)[10]について概説する．

第 4 章では，ドントケア，故障シミュレーション，テスト圧縮などの，VLSI のテストコスト削減に関する技術を概説する．テスト圧縮では，テスト生成中にテスト圧縮を行う動的圧縮[11-15]と，テスト生成後のテスト集合に対してテスト圧縮を行う静的圧縮[11,13,16,17]について述べる．

第 5 章では，テスト圧縮を考慮したドントケア判定法を提案する．一般に，生成されたテストパターンの各外部入力値は，全て 0 または 1 の論理値(ケアビット)に設定されている．しかしながら，生成されたテストパターンの中には，逆の値に変更しても故障検出率[1]が低下しない外部入力値が存在する．このような外部入力値をドントケア(X)という．テスト集合からドントケアを判定する技術には，ドントケア判定法[18]が提案されている．しかしながら，文献[18]のドントケア判定法は，各テストパターンで検出を保証する故障数が異なる．そのため，ドントケア数が少ないテストパターンが生成される可能性がある．このため，テスト時低消費電力化の分野において，ドントケアへのケアビット再割当ての効果がほとんどないテストパターンが存在する[19]．この問題を解決する手法として，各テストパターンで検出を保証する故障数を均一化するドントケア判定法が提案されている[19]．この手法により，各テストパターンのドントケア数が均一になり，テスト時低消費電力化に対し，より効果のあるテストパターンが生成可能となった[19]．

しかしながら，文献[19]のドントケア判定法では，各外部入力に対してはドントケア数の均一化を考慮していない[20-22]．そのため，テスト圧縮の分野において，特定の外部入力にケアビットが集中するとテスト圧縮効率が低下することが報告されている[20]．本論文では，各外部入力のドントケア数を均一にし，テスト圧縮に効果的なドントケア判定法を提案する．また，ISCAS’89，ITC’99 ベンチマーク回路に対して提案手法の有効性を

評価する．

第 6 章では，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法を提案する．フルスキャン設計を施した回路では，一般的なテスト生成は機能動作を考慮せず，テスト生成の容易性を優先してテスト生成を行う．このため，生成されたテストパターンは，スキャン FF の状態が機能動作では起こりえない状態(無効状態)となる可能性がある．無効状態では，回路内の多くの信号線に遷移を発生させ WSA を増加させている可能性がある[23][24]．

一方，文献[25]では，一般的なテスト生成で生成した遷移故障モデルのテストパターンを印可した後に，キャプチャ動作を 20 サイクル行うことで WSA が減少することが報告されている．また，文献[26]では k(≧2)時間展開モデルを利用して遷移故障を検出するテスト生成法として，マルチサイクルキャプチャ・テスト生成が提案されている．文献[26]では，順序回路的にテスト不可能な故障が数多く同定できることが報告されている．本論文では，文献[25]で報告されている現象と，文献[26]のテスト生成法に着目し，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法を提案する．また，ISCAS’89，ITC’99 ベンチマーク回路に対して提案手法の有効性を評価する．

第 7 章は結論であり，以上の研究成果を述べるとともに，今後の研究課題について議論している．

第 2 章

VLSI のテストとスキャン設計

2.1　VLSI のテスト

VLSI のテストとは, VLSI の製造時に偶発的に混入する不良品を選別するための作業のことである. VLSI のテストには, VLSI テスターと呼ばれる装置を用いる. VLSI テスターは，製造された VLSI の外部入力にテストパターンを印加し，外部出力の応答値を観測する. このとき，あらかじめ求めておいた外部出力の期待値と，VLSI テスターの応答値を比較し，VLSI に欠陥が存在するか検査を行う. このとき，期待値と応答値が一致しない場合，テスト対象の VLSI に欠陥が存在すると判定できる.

2.2　故障モデル

製造された VLSI の回路内には，断線や短絡，不純物の混入による抵抗値の増大など，さまざまな欠陥が生じる可能性がある．その欠陥が存在するか否かを調べるためにテストを行うが，これら全ての欠陥を直接取り扱うことはテスト生成時間やテスト実行時間の面から考えても非現実的である．そこで，欠陥をその振る舞いかたによって分類し，故障モデルとして表現することが一般的に行われている．これにより，異なる欠陥でも同じ故障モデルで表現できれば，同一の故障として扱うことが可能であり，扱う故障の個数を少なくすることができる．故障は論理故障とタイミング故障の二つに分類できる．

論理故障は，論理回路の論理機能が故障により別な論理機能に変化してしまう故障である．論理故障をモデル化したものに，縮退故障モデル，ブリッジ故障モデル[1]，トランジスタ故障モデル[1]，PLA(Programmable Logic Array)故障モデル[1]，メモリ故障モデル[1]，機能故障モデル[1]などがある．タイミング故障は，ゲートのスイッチング時間や配線の信号伝搬時間が増大し，本来起きるはずの FF の値の変化が規定の時間内に起こらない故障である．タイミング故障をモデル化したものに，遷移故障モデル，パス遅延故障モデル，ゲート遅延故障モデル[1]などがある．VLSI のテストには，縮退故障が回路内の一か所に存在すると仮定する単一縮退故障モデル[1]が広く用いられている．しかしながら，近年，単一縮退故障のテスト集合だけでは不良品と判定できない欠陥 VLSI がある[3][4]．そのため，市場不良率を低減するためには，遷移故障モデルなどのテストパターンが必要であると報告されている[3][4]．

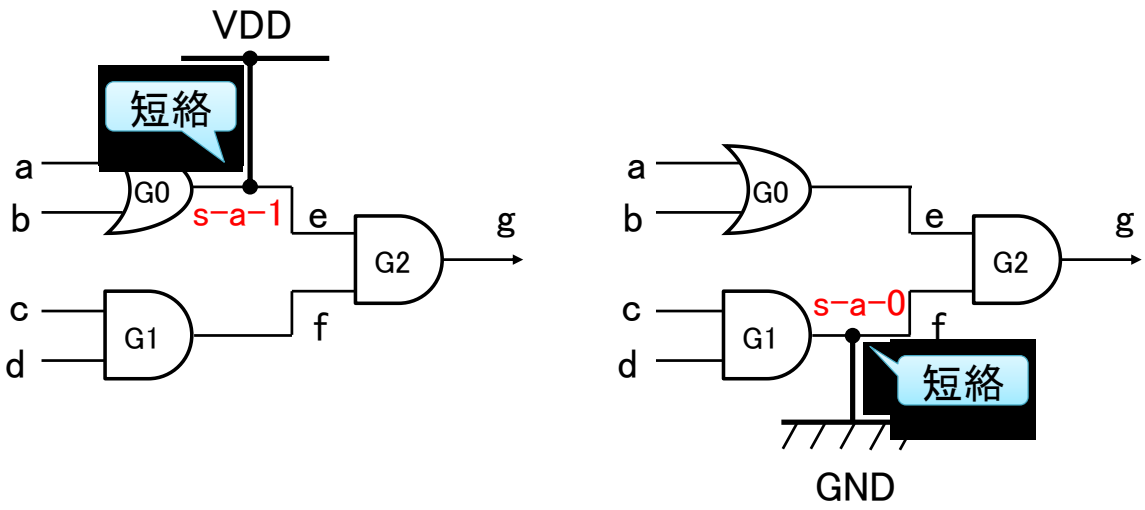
本節では，本論文で取り扱う縮退故障モデルと遷移故障モデルについて説明する．

2.2.1　縮退故障モデル

現在，最も広く用いられている故障モデルが縮退故障モデルである．縮退故障モデルは，論理ゲートの入力または出力の信号値が一定の論理値に固定される故障である．信号線の値が 0 に固定される場合を 0 縮退故障(stuck-at-0：s-a-0)といい，1 に固定される場合を 1 縮退故障(stuck-at-1：s-a-1)という．図 2.2.1(a)に 1 縮退故障の例を，図 2.2.1(b)に 0 縮退故障の例を示す．

図 2.2.1(a)において，VDD は電源を示す．図 2.2.1(a)では信号線 e と VDD(電源)とが短絡し，常に信号線 e の論理値が 1 に固定される 1 縮退故障を示している．ここで，s-a-1 は 1 縮退故障を示す．

図 2.2.1(b)において，GND はグランドを示す．図 2.2.1(b)では信号線 f と GND(グランド)とが短絡し，常に信号線 f の論理値が 0 に固定される 0 縮退故障を示している．ここで，s-a-0 は 0 縮退故障を示す．



（a）1 縮退故障　　（b）0 縮退故障

図 2.2.1：縮退故障の例

2.2.2　遷移故障モデル

遷移故障モデルは，ある一つの論理ゲートの入力または出力の信号線の伝搬遅延が増大する遅延故障モデルである．遷移故障モデルには立ち上がり遷移故障(slow-to-rise ： STR)と，立ち下がり遷移故障(slow-to-fall ： STF)の 2 種類に分類できる．立ち上がり遷移故障は，ある信号線値が 0 から 1 に遷移するのが遅延する故障である．立ち下がり遷移故障は，ある信号線値が 1 から 0 に遷移するのが遅延する故障である．遷移故障モデルの遅延は，経路の長短にかかわらず FF で観測される．そのため，遅延が VLSI の動作周期より大きいと考えてよい．

図 2.2.2 に立ち上り遷移故障の例を示す．図 2.2.2 において，信号線 b の(0,1)は，1 時刻目の信号線値が 0，2 時刻目の信号線値が 1 を示す．図 2.2.2 では，信号線 c の立ち上がり遷移故障の例である．信号線 c で立ち上がり遷移故障が発生し，FF に伝搬される値が次のクロックのキャプチャタイミングに間に合わず，正常値とは異なる 1 が取り込まれる．

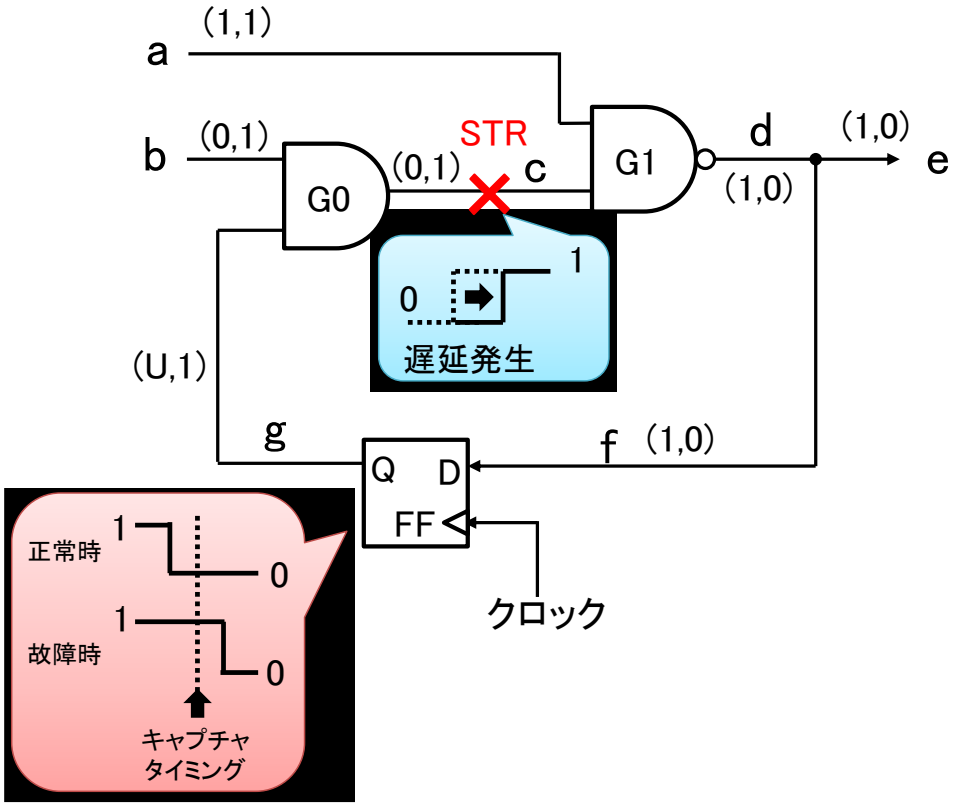


図 2.2.2：立ち上り遷移故障の例

図 2.2.3 に，図 2.2.2 の回路における立ち上り遷移故障のタイミングチ

ャート例を示す．図 2.2.3 は，図 2.2.2 の信号線 c が立ち上がり遷移故障している例である．Tはクロック周期を示す．T_gは正常時の c を通過する FF 間までの経路の最小伝搬時間を示す．遅延時間が T-Tg を越える時，遷移故障が発生する．

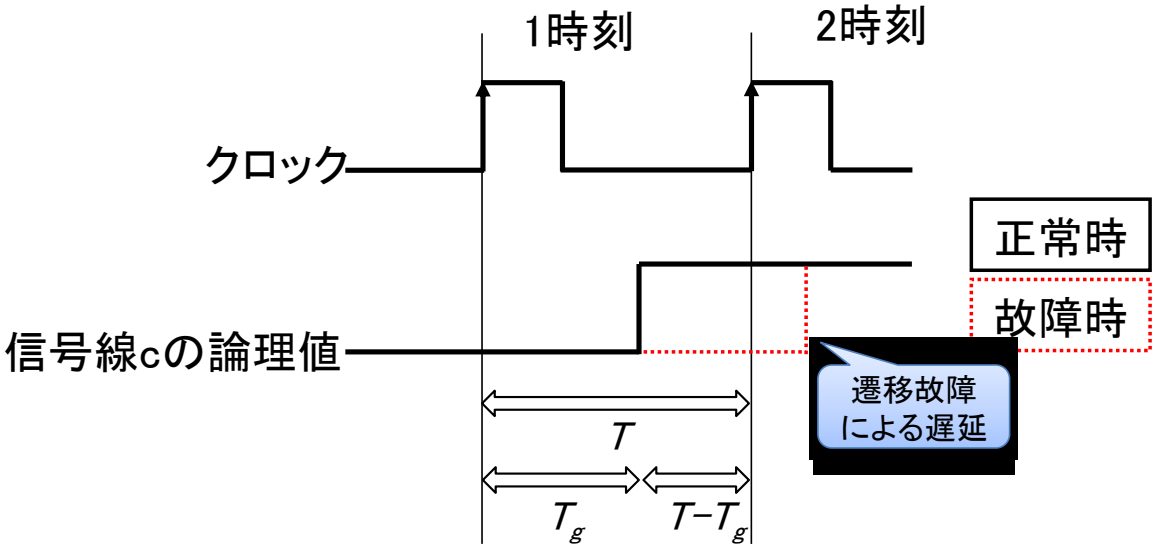


図 2.2.3：立ち下り遷移故障のタイミングチャート

2.3　テスト生成

テスト生成は，与えられた故障に対して冗長であるか否かを判定する．冗長でない故障の場合，その故障の影響を外部出力で観測可能な外部入力値の組合せを求める．このような外部入力値の組合せをテストパターンという．また，テストパターンの集合をテスト集合と呼ぶ．テスト生成には回路構造に基づき分枝限定法で解を求める経路活性化法[27-31]と，充足可能性問題(Satisfiability Problem ： SAT)を用いて解を求める手法[32-36]がある．

本節では，2.3.1 で組合せ回路のテスト生成，2.3.2 で順序回路のテスト生成について述べる．

2.3.1　組合せ回路のテスト生成

組合せ回路の回路構造に基づくテスト生成アルゴリズムとして，D アルゴリズム[27]，PODEM(Path Oriented Decision Making)アルゴリズム[28]，FAN(Fan-out Oriented Test Generaion Algorithm)アルゴリズム[29]，SOCRATES[30]，SPIRIT[31]などが提案されている．また，充足可能性問題に基づくテスト生成アルゴリズムとして，NEMESIS[32][33]，TEGUS[34]，TG-GRASP[35]，PASSAT[36]などが提案されている．

図 2.3.1 に組合せ回路に対するテスト生成の例を示す．図 2.3.1 は，信号線 a の 0 縮退故障に対するテスト生成例である．図 2.3.1 において，1/0 は正常値 1，故障値 0 を示す．同様に 0/1 は正常値 0，故障値 1 を示す．まず，信号線 a に故障励起のために正常値 1 を割当てる．次に，信号線 d に故障影響を伝搬させるために，信号線 b に AND ゲートの非制御値である 1 を割当てる．さらに，故障影響を信号線 e に伝搬させるために，信号線 c に 0 を割当てる．すると外部出力 e で故障が検出できることを確認できる．図 2.3.1 の回路において，信号線 a の 0 縮退故障を検出可能なテストパターンは(a, b, c) = (1, 1, 0)であることがわかる．また，誤りが伝搬した経路(パス)である a, d, e を活性化経路と呼ぶ．

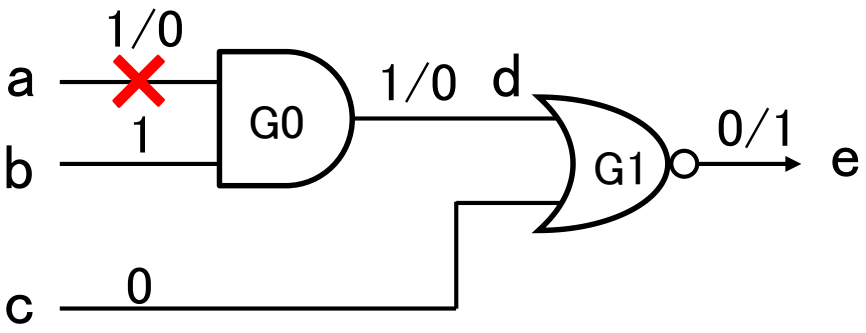


図 2.3.1：組合せ回路に対するテスト生成例

2.3.2　順序回路のテスト生成

スキャン設計を用いない順序回路に対するテスト生成は，順序回路から組合せ回路モデルの一種である時間展開モデル[1]を用いる手法がよく利用される．しかしながら，展開時間数が大きいと回路モデルの規模が増大する．回路モデルの規模が増大すると，テスト生成の複雑度が増し高い故障検出効率を得ることが困難となる．順序回路のテスト生成アルゴリズムとして，拡張 D アルゴリズム[37]などが提案されている．

図 2.3.2 に，順序回路のテスト生成例を示す．順序回路のテスト生成アルゴリズムは回路中の組合せ回路部を図 2.3.2 に示すように時間軸に展開した時間展開モデルを生成して行う．図 2.3.2 において各回路 C(i)（以後セルと呼ぶ）は，時刻 i に対応する順序回路の組合せ回路部に対応し，X(i)はその時刻の入力を表す．Y(i), Y(i+1)はそれぞれ，時刻 i, i+1 の FF の状態に対応している．一般に FF の初期状態は未知であるので，Y(0) = (U, U, U, …, U)とする．ここで，U は未知の値(不定値)を示す．図 2.3.2 では，時刻 q において故障箇所に誤りが発生し，p 時刻後に誤りが外部出力 Z(q+p)に伝搬する例を示している．

順序回路のテスト生成法のうち，時間展開モデルを利用してテスト系列を生成する手順例は次のようになる．

Step 1：

p と q に適当な非負の整数を選ぶ．最初は p=q=0 とおく．

Step 2：

q 番目のセル C(q)において，故障箇所の正常値と故障値が異なるような入力パターン X(q)と Y(q)を求める．

Step 3：

故障の影響を q+p 番目のセルの出力 Z(q+p)にまで伝搬させる．伝搬できなければ p を 1 つ増加して繰り返す. 伝搬に成功した場合は, 入力 X(q), X(q+1), ..., X(q+p)および Y (q), Y (q+1), ..., Y (q+p)の値を決定する．これらの値を決定する際に何らかの矛盾が発生し，値の決定に失敗したら，再び q 番目のセルにおいて別の X(q)と Y(q)を求める．

Step 4：

Y(q)が求められた値をとり，かつ初期状態が任意状態，すなわち Y(0) = (U,U,U,...,U)になるように，X (0), X (1), ..., X (q-1)の値を求める．求められない場合，q の値を 1 つ増し，繰り返す.

Step 4 で注意すべき点は, 故障は単一故障であっても時間展開した回路では多重故障になることである.

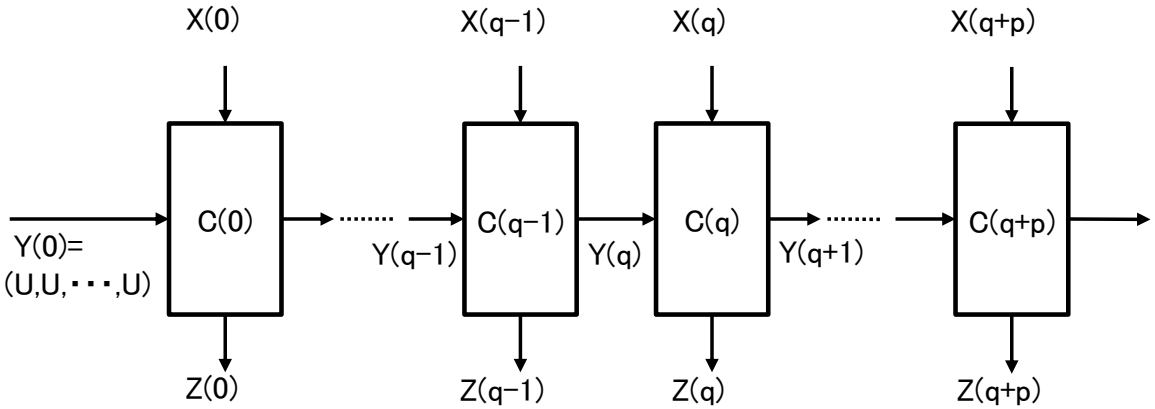


図 2.3.2：順序回路の時間展開とテスト系列

図 2.3.3 に順序回路の例を示す．図 2.3.4 と図 2.3.5 に，図 2.3.3 の順序回路の信号線 d に 1 縮退故障を仮定して，時間展開モデルを用いてテスト生成をした例を示す．図 2.3.4 は p=q=1 の例であり，図 2.3.5 は p=1, q=2 の例である．

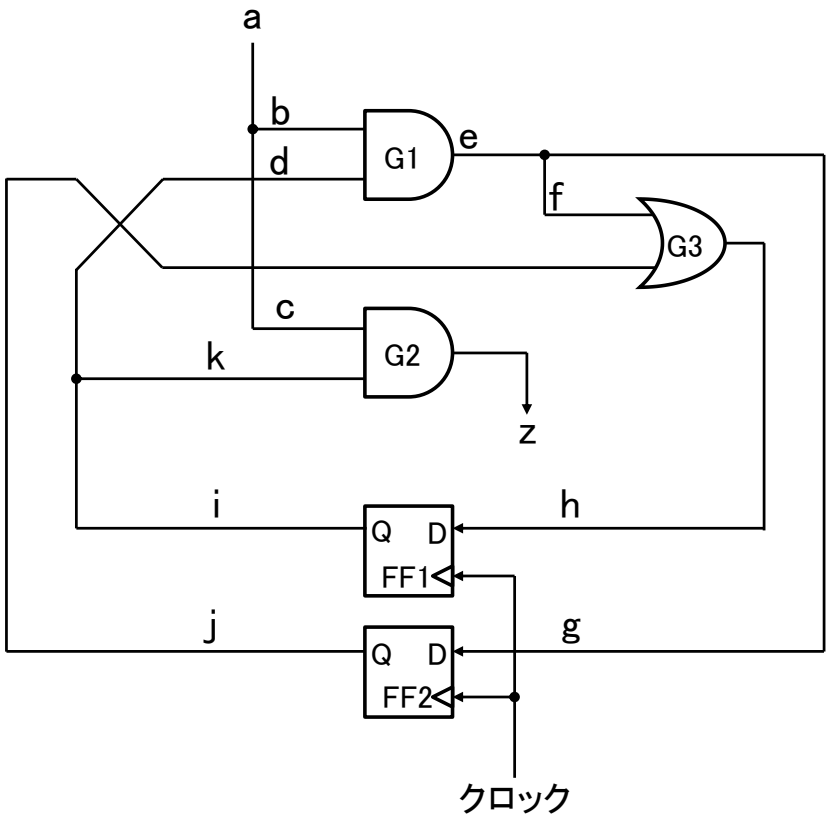


図 2.3.3：順序回路例

まず，p=0, q=1 とおき，d の 1 縮退故障を検出するテストパターンを求める．セル C(1)における信号線 d_1 の値を 0/1 とする．セル C(1)の外部出力 z_1 に故障影響を伝搬することが出来ないので，p=1 と増やす．この時の時間展開モデルの回路図は，図 2.3.4 となる．図 2.3.4 では，セル C(1)の信号線 d_1 の故障値 0/1 がセル C(2)へ伝搬して，外部出力 z_2 に伝搬する．このとき，入力は a_1=1，a_2=1 で，内部状態は g_0=h_0=0 である．h_0=0 とするためには，セル C(0)の OR ゲート G3 の入力信号線 f_0 と i_0 を両方とも 0 に割当てる必要がある．したがって，j_0=0 となり，j_0=U とできない．そこで，q を 1 つ増加させ q=2 とする．この場合の時間展開モデル例を図 2.3.5 に示す．

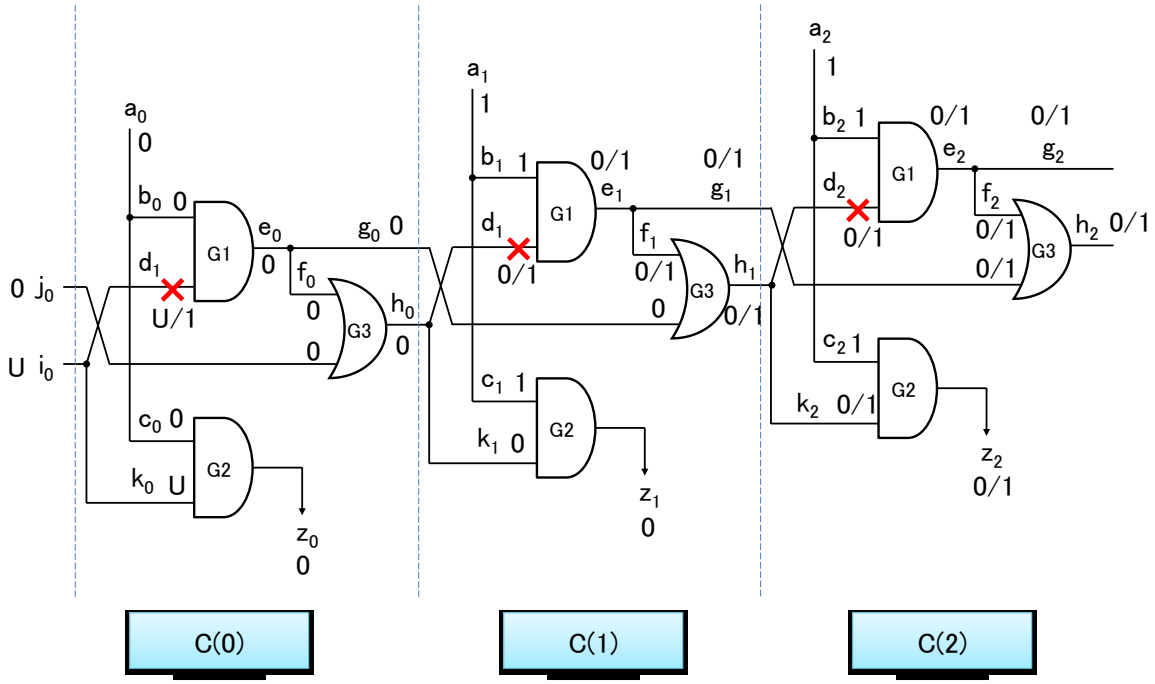


図 2.3.4：時間展開モデルを用いたテスト生成例(p=q=1 の場合)

図 2.3.5 おいて，セル C(1)，C(2)，C(3)は図 2.3.4 のセル C(0)，C(1)，C(2)に対応する．図 2.3.5 のセル C(1)の状態 g_1=h_1=0 とするには，a_1=0，g_0=0 とすればよい．さらに，セル C(0)の状態 g_0=0 とするには，a_0=0 とすればよい．この時，セル C(0)の状態 j_0 と i_0 はどのような値でも良く，不定値 U となる．結果として，図 2.3.3 の順序回路の信号線 d の 1 縮退故障を検出可能なテスト系列は，(a_0，a_1，a_2，a_3) = (0，0，1，1)となる．

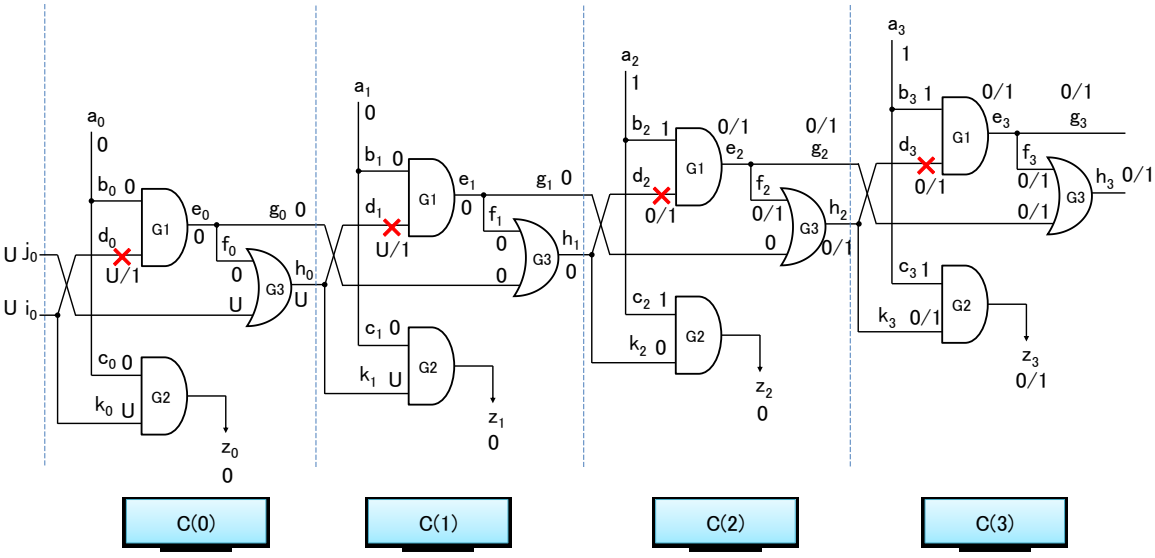


図 2.3.5：時間展開モデルを用いたテスト生成例(p=1, q=2 の場合)

2.4　テスト品質評価尺度

一般にテストパターンの生成はテスト生成ツールを用いて行う．テスト生成は，故障モデルを定義し，その故障モデルを検出するテストパターンの生成を試みる．その結果，故障に対するテストパターンを求めるか，テスト不可能な故障であるかを判定する．ここでいうテスト不可能な故障とは，任意の入力に対して故障が回路の出力応答に影響を与えないため，検出が不可能な故障をいう．特に組合せ回路のテスト不可能な故障を冗長故障[1]という．テスト生成の際にテスト不能な故障の判定が困難なため，処理の途中でバックトラック数の制限値などにより打ち切られる場合がある．このような故障を打ち切り故障[1]と呼ぶ．

テストコストを示す評価尺度には，テスト生成時間やテスト実行時間がある．テスト品質の評価尺度には，故障検出率がある．故障検出率は，テスト生成で生成されたテストパターン集合が仮定した故障をどれだけ検出できるかを示す比率である．式(2.4.1)に故障検出率の計算式を示す．

$$\text{故障検出率}[\%] = \frac{\text{検出故障数}}{\text{総仮定故障数}} \times 100 \qquad (2.4.1)$$

テストパターン集合としては，故障検出率 100%が望ましい．しかしながら，冗長故障が含まれる場合は故障検出率 100%の達成は不可能である．そのため，テスト品質を評価する別の評価尺度として故障検出効率[1]がある．故障検出効率は，テスト生成が検出可能と判定した故障数とテスト不可能と判定した故障数の比率である．式(2.4.2)に故障検出効率の計算式を示す．

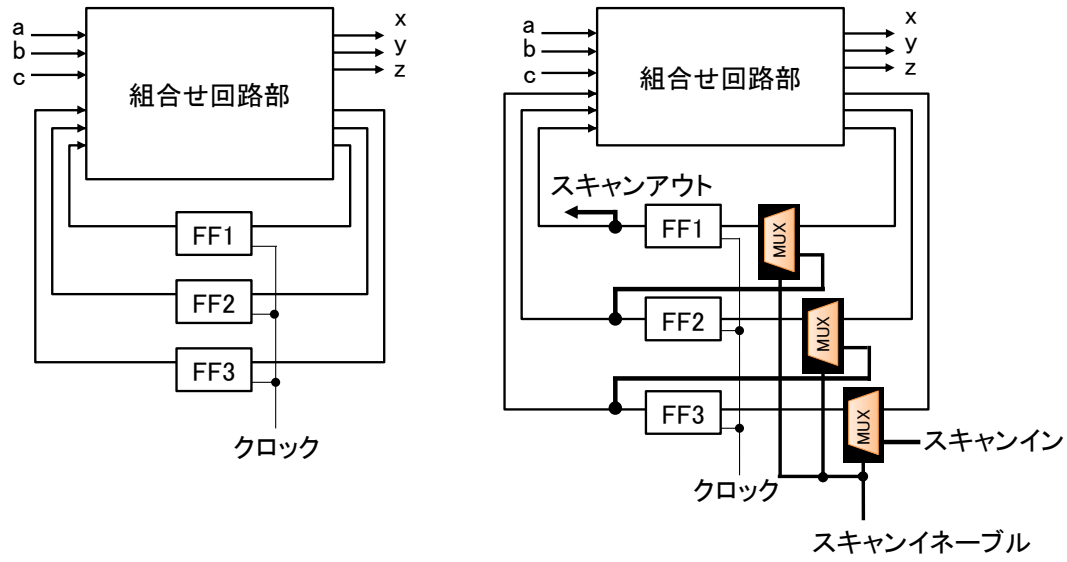
$$\text{故障検出効率}[\%] = \frac{\text{検出故障数} + \text{テスト不可能故障数}}{\text{総仮定故障数}} \times 100 \qquad (2.4.2)$$

故障検出効率が 100%達成されることを完全故障検出効率と呼ぶ．また，故障検出効率 100%のテストが最良のテストといえる．

2.5　スキャン設計

組合せ回路に対するテスト生成は，高速で高い故障検出効率を得る手法が提案されている[27-36]．一方，順序回路に対するテスト生成は問題の解空間が膨大になり，現実的な時間で高い故障検出効率を得られない[1]．そこで，回路をテスト生成が容易になるように変更する DFT が行われる．広く普及している DFT 方法に回路中の FF を可制御可観測にするスキャン FF に置換するフルスキャン設計[1]がある．

図 2.5.1 に順序回路をフルスキャン設計した前後の回路を示す．図 2.5.1(a)はフルスキャン設計前の順序回路である．図 2.5.1(b)は，図 2.5.1(a)の順序回路に対してフルスキャン設計を施した回路例である．フルスキャン設計では，各 FF に外部から直接，値を与えることが可能になるようにスキャンイン端子を追加する．さらに，マルチプレクサ(Multiplexer：MUX)を追加し，通常動作時のデータ入力とスキャンイン動作の切替えを可能にする．一方，各 FF の値を直接，外部出力で観測することが可能になるようにスキャンアウト端子を追加する．さらに，各 FF を一列に連結することにより，シフトレジスタとして動作可能にする．通常動作とスキャンイン動作(スキャンアウト動作)の切替えはスキャンイネーブル信号線で制御する．



（a）順序回路例　　（b）フルスキャン設計

図 2.5.1：フルスキャン設計の例

フルスキャン設計により，スキャン FF への入力は，スキャンイン入力より外部から自由に制御できるので組合せ回路に対する外部入力と等価であるとみなせる．これを，擬似外部入力(Pseudo Primary Input：PPI)という．同様にスキャン FF の出力は，スキャンアウト出力より自由に観測できるので組合せ回路に対する外部出力と等価であるとみなせる．これを擬似外部出力(Pseudo Primary Output：PPO)という．フルスキャン設計された順序回路は，組合せ回路に対するテスト生成技術を適用してテストパターンの生成が可能になり，実際の VLSI に対して高い故障検出効率を得ることができる．しかしながら，フルスキャン設計方法は，回路にテスト用の回路要素を追加するため，面積，遅延，消費電力などのオーバーヘッドが大きいという問題点がある．

2.6　実速度スキャンテスト

タイミング遅延を伴う欠陥の故障モデルとして，遷移故障モデルやパス遅延故障モデルなど存在する．これらの遅延故障モデルを検出するには，変化前の信号値を設定するテストパターンと変化後の信号値を確かめるテストパターンの 2 種のテストパターンを連続して印加する必要がある．このテスト方法を 2 パターンテストという．本節では，スキャン設計回路における 2 パターンテストを実現するテスト方法として，ブロードサイド方式[38]とスキュードロード方式[39][40]について説明する．

2.6.1　ブロードサイド方式

ブロードサイド方式では，1 パターン目はシフトイン動作によってテストパターンが印加され，2 パターン目では，1 パターン目のテストパターンの回路応答によってテストパターンが印加される．また 1 パターン目と 2 パターン目の外部入力値には同じ値を印加する．

図 2.6.1 にブロードサイド方式でテストするときのクロック信号線とスキャンイネーブル信号線のタイミングチャートを示す．図 2.6.1 において，キャプチャ動作時の 1 サイクル目(スキャン FF に 2 パターン目を設定)と 2 サイクル目(スキャン FF に故障の影響を取り込む)の間を実速度で動作させる．

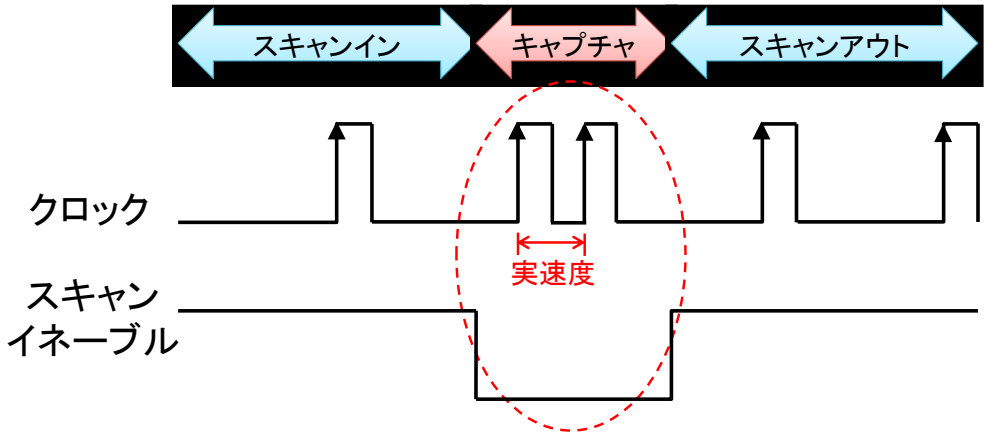


図 2.6.1：ブロードサイドにおけるタイミングチャート

図 2.6.2 にブロードサイド方式における各時刻の回路動作を時間展開したものを示す．ブロードサイド方式の特徴は 2 パターン目の FF の入力が通常動作と同じなので設計時のタイミング制約が少ない．また無効状態を用いた過剰テストによる歩留まりの低下の危険性もスキュードロード方式と比較して少ないことが挙げられる．しかしながら，2 パターン目は 1 パターン目の回路応答を使用するため，解の探索空間が広くなり，高い故障検出効率を得るための効率の良いテスト生成が必要とされる．

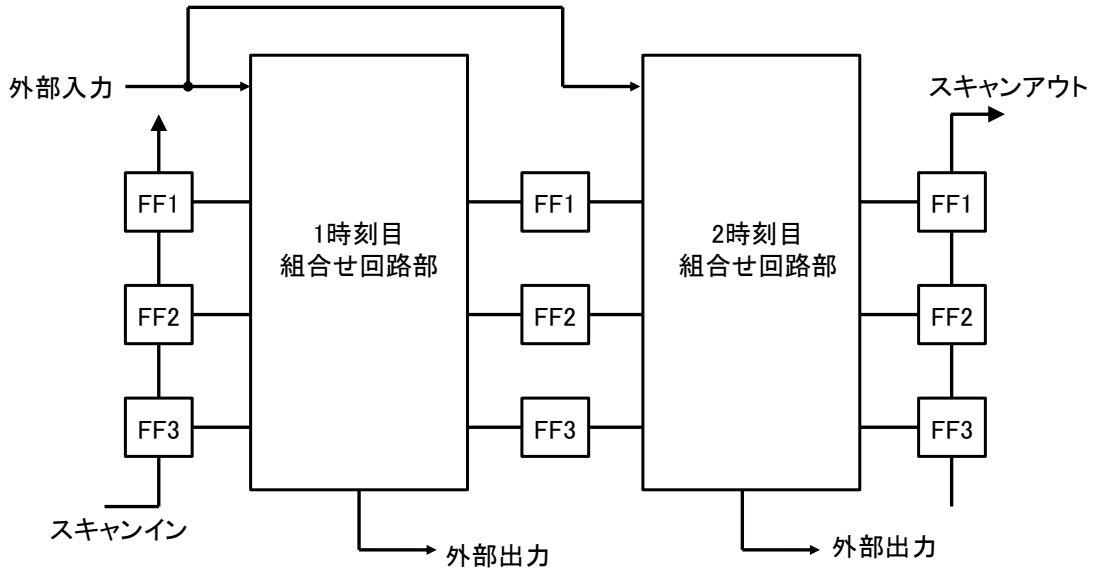


図 2.6.2：ブロードサイド方式の回路動作の時間展開例

図 2.6.3 にブロードサイドテストの例を示す．図 2.6.3 は，図 2.3.3 の順序回路の信号線 h の立ち下がり遷移故障に対するブロードサイドテスト例である．図 2.6.3 において，信号線 a は外部入力，信号線 j_0，i_0 は擬似外部入力，信号線 g_1, h_1 は擬似外部出力を示す．制御可能な信号線は a, j_0, i_0 であり，観測可能な信号線は g_1, h_1 ある．まずスキャンイン動作により，1 時刻目の信号線 h_0=1 を割当て可能なテストパターンを 1 パターン目として印加する．図 2.6.3 では，(a, j_0, i_0) = (0, 1, 1)を印加する．次の時刻でキャプチャ動作を行い，1 パターン目の回路応答を取り込み，2 時刻目の信号線 h_1 の 1 縮退故障を検出可能な (a, h_0, g_0) = (0, 1, 0)が 2 パターン目として印加されることにより，信号線 h_1 において 1 から 0 へ遷移が発生し，次の時刻で擬似外部出力 h_1 において故障を検出できる．

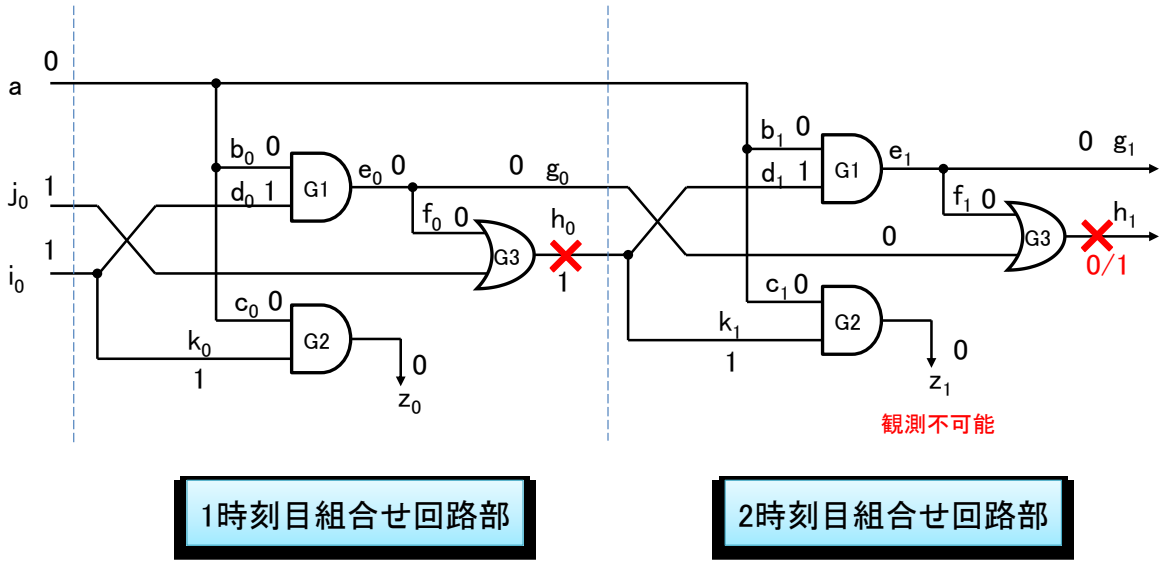


図 2.6.3：ブロードサイドテストにおける遷移故障検出例

2.6.2　スキュードロード方式

スキュードロード方式では，スキャンイン動作で 1 パターン目を設定した後，スキャンイン動作のまま次のサイクルで 2 パターン目の設定を行う．スキャンイン動作の最終サイクルとキャプチャ動作のサイクル間を実速度で動作させる．また 1 パターン目と 2 パターン目の外部入力値には同じ値を印加する．

図 2.6.4 にスキュードロード方式におけるタイミングチャートを示す．

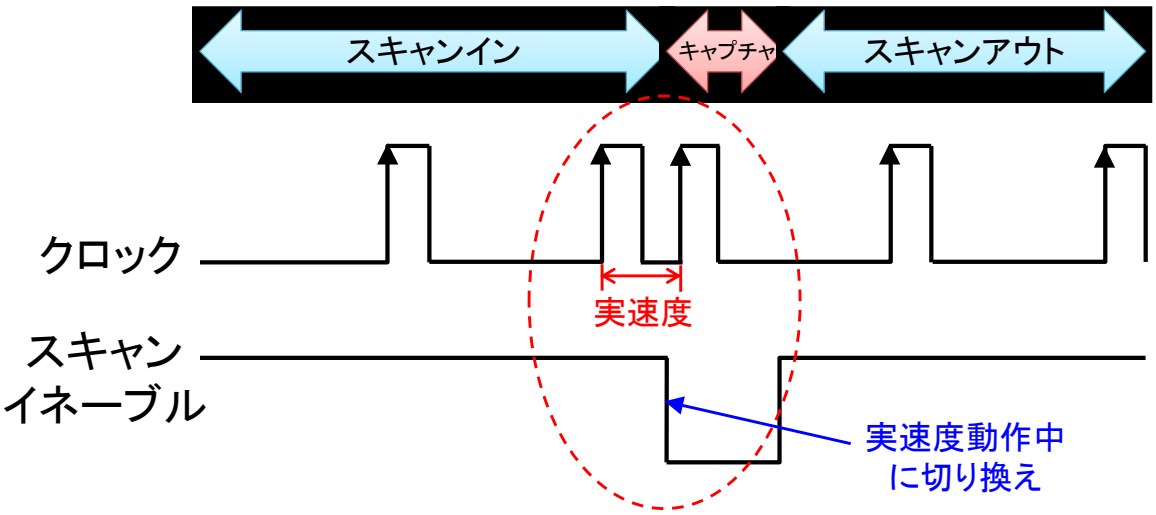


図 2.6.4：スキュードロード方式におけるタイミングチャート

図 2.6.5 にスキュードロード方式における各時刻の回路動作を時間展開した図を示す．スキュードロード方式で遷移故障を検出する際には，スキャンインのための最終クロックとキャプチャクロックの間隔で実速度の動作が要求されるため，短時間でスキャンシフト動作からキャプチャ動作へとスキャンイネーブル信号を切り替える必要がある．また，スキュードロード方式では，スキャンチェイン中の FF の並びに依存したテストパターンを生成する．また，ブロードサイド方式では検出不可能，または検出困難な故障をスキュードロード方式では検出できる可能性がある．しかしながら，通常動作において使用しない状態遷移に基づく動作でテストを行う場合が増加し，過剰テストにつながる．また，実動作中にシフト動作からキャプチャ動作への切替えを行うため，物理的な設計制約が厳しくなるという問題点が存在する．

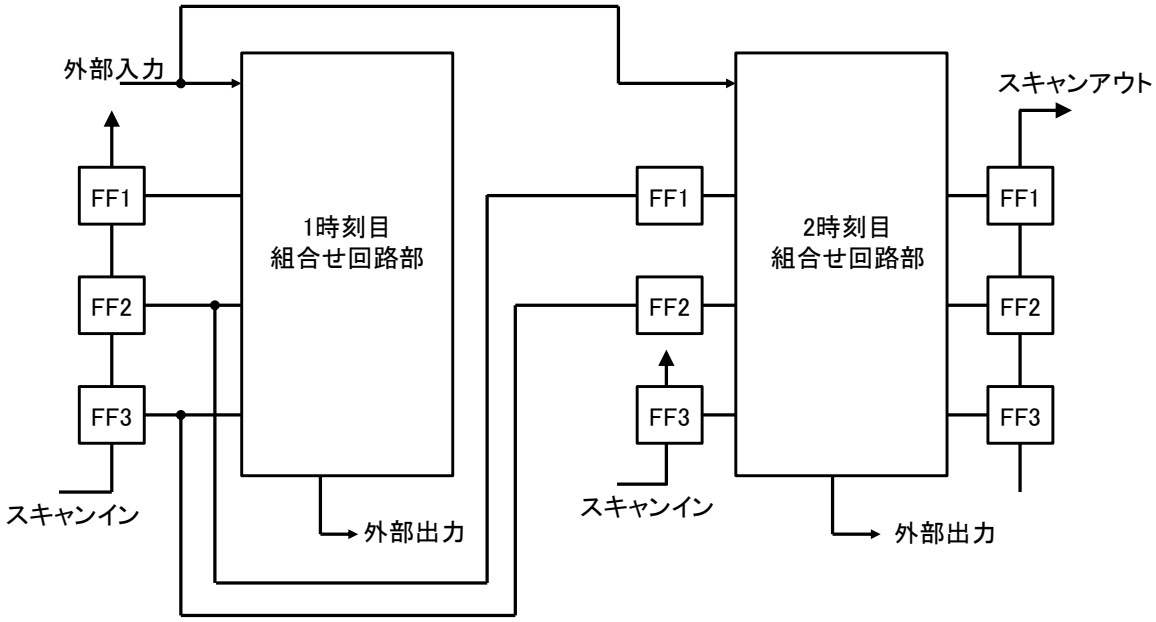


図 2.6.5：スキュードロードの回路動作の時間展開例

図 2.6.5 にスキュードロードテストの例を示す．図 2.6.5 は，図 2.3.3 の順序回路の信号線 j の立ち下がり遷移故障に対するスキュードロードテスト例である．図 2.6.5 において，信号線 a は外部入力，信号線 j_0，i_0，i_1 は擬似外部入力，信号線 g_1，h_1 は擬似外部出力を示す．制御可能な信号線は a, j_0，i_0，i_1 であり，観測可能な信号線はで g_1，h_1 ある．また，信号線

i_1 は 2 時刻目のスキャン FF への 2 パターン目印加のための，スキャンイン動作を示す．まずスキャンイン動作により，1 時刻目の信号線 h_0=1 を割当て可能なテストパターンを 1 パターン目として印加する．図 2.6.6 では，$(a, j_0, i_0) = (0, 1, 1)$を印加する．

次にシフト動作により 2 時刻目の信号線 h_1 の 1 縮退故障を検出可能なテストパターン$(a, i_0, i_1) = (0, 1, 0)$を 2 パターン目として印加する．最後にキャプチャ動作を行い，擬似外部出力 h_1 に伝搬している故障の影響をスキャンアウトにおいて故障を検出する．

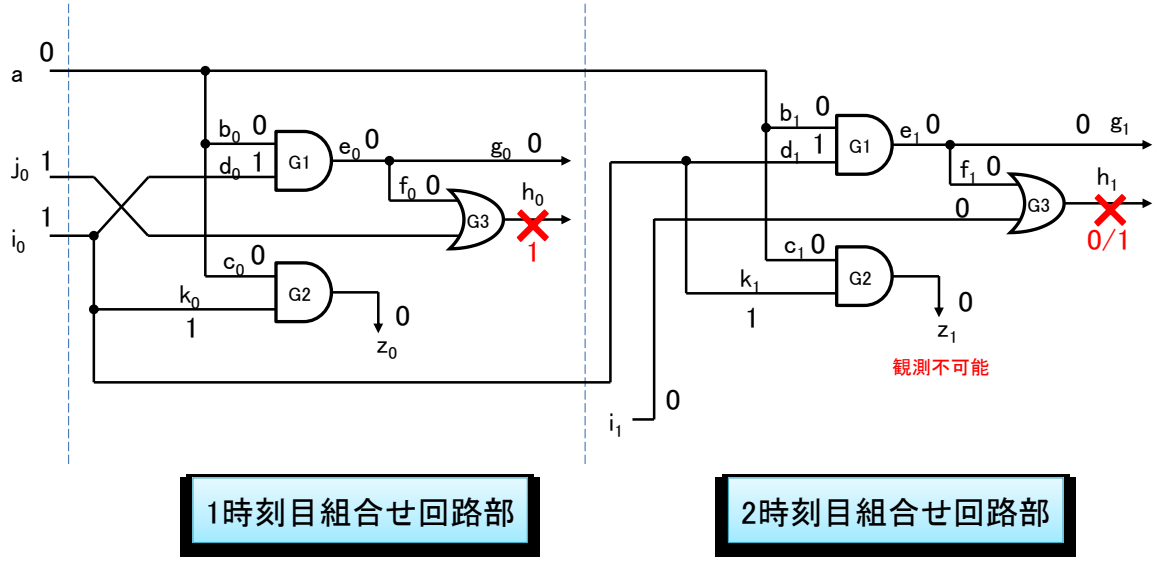


図 2.6.6：スキュードロードにおける故障検出例

第 3 章

VLSI の消費電力

3.1　CMOS 論理回路の消費電力

現在の VLSI の大多数は，CMOS から構成される．本節では，CMOS 論理回路の消費電力について述べる．

図 3.1.1 に NOT ゲートの CMOS 論理回路の例を示す．図 3.1.1 において，VDD は電源電圧を示し，GND はグランドを示し，IN は NOT ゲートの入力信号線を示し，OUT は NOT ゲートの出力信号線を示す．

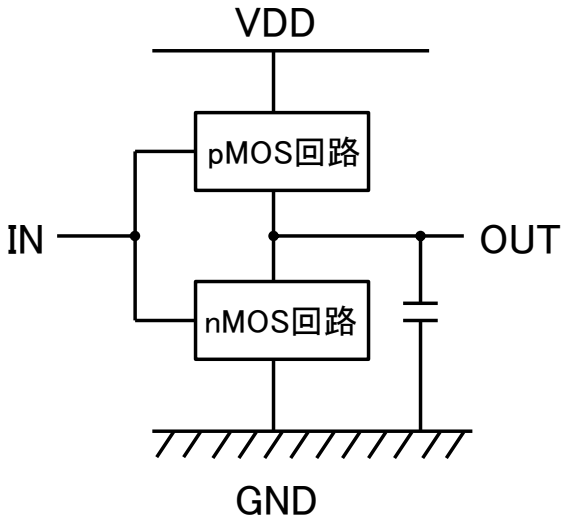


図 3.1.1：CMOS 論理回路例

CMOS 論理回路の消費電力は，スイッチング電流による消費電力，貫通電流による消費電力，直流電流による消費電力，リーク電流による消費電力の 4 つに分類できる[41].

式(3.1.1)に CMOS 論理回路の消費電力の計算式[41]を示す．式(3.1.1)において，Pは CMOS 論理回路の消費電力，$P_{dynamic}$はスイッチング電流による消費電力，P_{SC}は貫通電流による消費電力，P_{DC}は直流電流による消費電力，P_{Leak}はリーク電流による消費電力を示す.

$$P = P_{dynamic} + P_{SC} + P_{DC} + P_{Leak} \tag{3.1.1}$$

式(3.1.2)にスイッチング電流による消費電力の計算式[41]を示す．式(3.1.2)において，p_tはスイッチング確率，f_{CLK}は動作周波数，C_Lは負荷容量，V_Sは信号振幅，V_{DD}は電源電圧を示す．スイッチング電流による電力とは，CMOS 論理回路が負荷の充放電に要する消費電力である.

$$P_{dynamic} = p_t \cdot f_{CLK} \cdot C_L \cdot V_S \cdot V_{DD} \tag{3.1.2}$$

式(3.1.3)に貫通電流による消費電力の計算式[41]を示す．式(3.1.3)において，I_{SC}は貫通電流の平均値，Δt_{sc}は貫通電流が流れる時間を示す．CMOS 論理回路では，pMOS と nMOS のオンとオフが高速に変化する際，物理構造的に，両方がオンまたは両方がオフになる瞬間が存在する．その際に，電源からグランドに向け，貫通電流が発生する.

$$P_{SC} = p_t \cdot I_{SC} \cdot \Delta t_{SC} \cdot V_{DD} \cdot f_{CLK} \tag{3.1.3}$$

式(3.1.4)に直流電流による消費電力の計算式[41]を示す．式(3.1.4)において，I_{DC}は直流電流を示す．直流電流による消費電力は，メモリのセンスアンプやアナログ回路で流れるバイアス電流が主な原因である.

$$P_{DC} = I_{DC} \cdot V_{DD} \tag{3.1.4}$$

式(3.1.5)にリーク電流による消費電力の計算式[41]を示す．式(3.1.5)において，I_{Leak}はリーク電流を示す．リーク電流は微細化によるゲート絶縁膜の絶縁性の低下や，閾値電圧の低下により発生する電流である．リーク電流の主な原因は，ソースやドレイン接合の逆方向電流や，MOS トランジスタのサブスレッショルド電流[41]である．

$$P_{Leak} = I_{Leak} \cdot V_{DD} \tag{3.1.5}$$

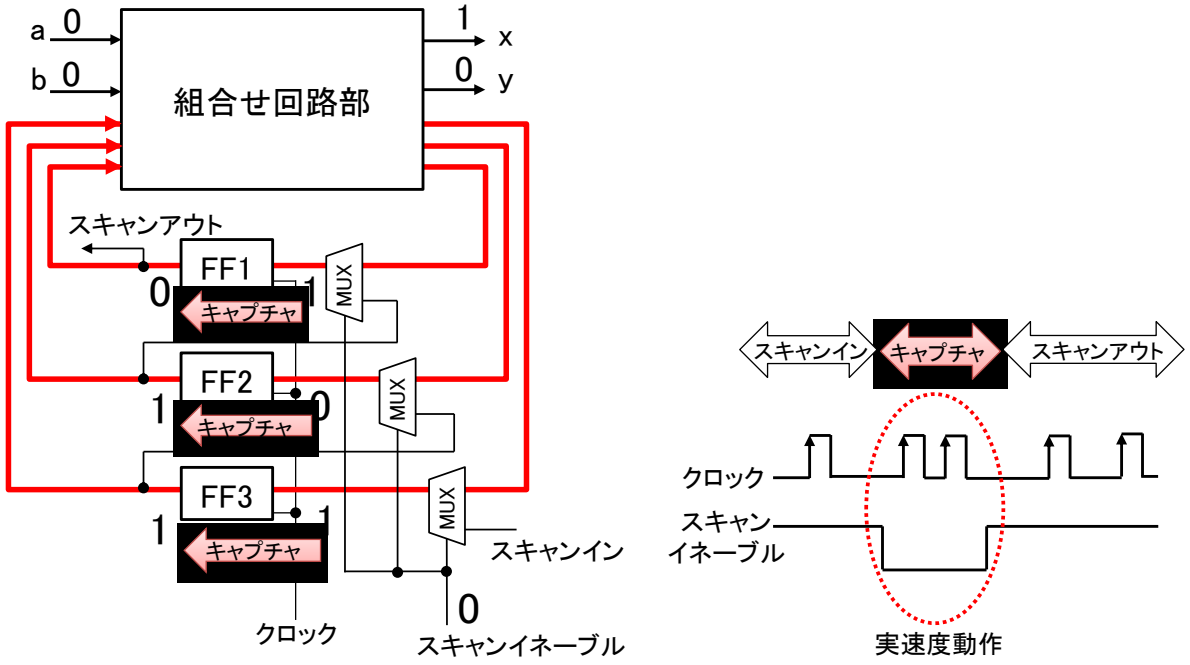
3.2　VLSI のテスト時消費電力

CMOS 論理回路の消費電力は，スイッチング電流による消費電力，貫通電流による消費電力，直流電流による消費電力，リーク電流による消費電力の 4 つに分類できる．VLSI のテスト時消費電力の削減には，テストパターンのスイッチング動作(0 から 1，または 1 から 0 への値の遷移)の削減によるスイッチング電流と貫通電流による消費電力削減技術[25][42-47]が多く提案されている．本節では，VLSI の実速度スキャンテスト時特有の消費電力として，キャプチャ時消費電力とシフト時消費電力について概説する．

3.2.1　キャプチャ時消費電力

実速度スキャンテスト特有の消費電力として，キャプチャ時消費電力が挙げられる．キャプチャ時消費電力は，テスト応答を FF へ格納するキャプチャ動作時に発生する．キャプチャ動作時において，値の遷移が発生する FF が多いほど，過度な消費電力が発生する．過度なキャプチャ時消費電力による問題として，IR ドロップによる誤テストが挙げられる．

図 3.2.1 にブロードサイド方式のおける，キャプチャ時消費電力の例を示す. 図 3.2.1(a)は(FF1, FF2, FF3) = (0, 1, 1)から(FF1, FF2, FF3) = (1, 0, 1)にキャプチャ動作する例を示している．図 3.2.1(a)において，FF1 が 0 から 1 へ，FF2 が 1 から 0 へ遷移が発生している．図 3.2.1(b)にブロードサイド方式のタイミングチャートを示す．キャプチャ時消費電力は図 3.2.1(b)のタイミングチャートにおいて，スキャンイネーブルが 0 の状態でクロックが動作するキャプチャ動作時に発生する．



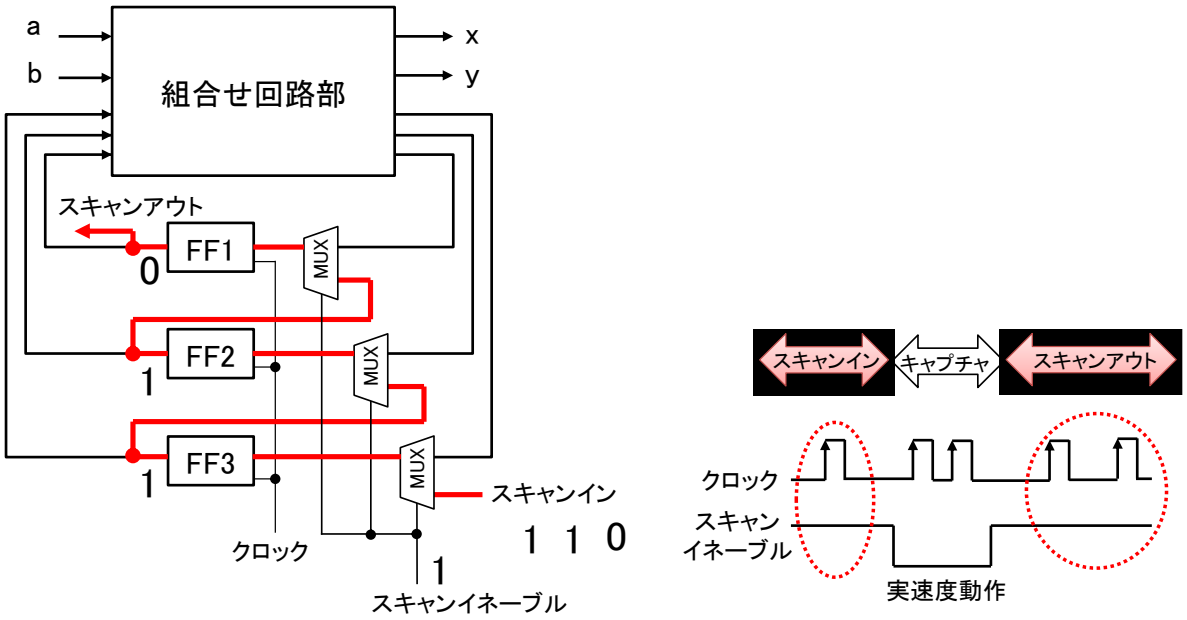
(a)キャプチャ動作例　　(b)タイミングチャート

図 3.2.1：キャプチャ時消費電力例

3.2.2　シフト時消費電力

実速度スキャンテスト特有の消費電力として，シフト時消費電力が挙げられる．シフト時消費電力は，スキャン FF へのテストパターンの印加(スキャンイン動作)とテスト応答の観測(スキャンアウト動作)を行うシフト動作時に発生する．シフト動作時において，値の遷移が発生する FF 数が多いほど，過度な消費電力が発生する．過度なシフト時消費電力による問題として，発熱による回路の熱破壊が挙げられる．

図 3.2.2 にシフト時消費電力の例を示す．図 3.2.2(a)は(FF1, FF2, FF3) = (0, 1, 1)から(FF1, FF2, FF3) = (1, 1, 0)にシフト動作する例を示している．図 3.2.2(a)において，内部状態(FF1, FF2, FF3) = (0, 1, 1)に対してスキャンイン動作より 1 を印加すると，(FF1, FF2, FF3) = (1, 1, 1)に値が変化する．この時，FF1 に 0 から 1 の遷移が発生する．同様にスキャンイン動作より 1 を印加すると，(FF1, FF2, FF3) = (1, 1, 1)のままで，各 FF に値の遷移は発生していない.同様にスキャンイン動作より 0 を印加すると，(FF1, FF2, FF3) = (1, 1, 0)に値が変化する．この時，FF3 において，1 から 0 の遷移が発生する．図 3.2.2(b)にブロードサイド方式のタイミングチャートを示す．シフト時消費電力は図 3.2.2(b)のタイミングチャートにおいて，スキャンイネーブルが 1 の状態でクロックが動作するシフト動作時に発生する．また，シフト時消費電力は，スキャンイン動作時とスキャンアウト動作時のどちらでも発生する．



(a) シフト動作例　　(b)タイミングチャート

図 3.2.2：シフト時消費電力例

3.3　WSA

VLSI の消費電力の見積り手法として，消費電力と相関が高く，計算が容易な WSA が広く用いられている[23-25]．WSA は電源電圧，動作周波数，負荷容量などを考慮した厳密な消費電力計算を行わず，ゲートの出力信号線の遷移数から消費電力の見積り値を計算する．式(3.3.1)に WSA の計算式を示す．

$$WSA(t_j) = \sum_{i=1}^{G} (tran(g_i) \times (1 + fanout(g_i))) \qquad (3.3.1)$$

式(3.3.1)において，t_j はテストパターン，G は回路内に含まれる総ゲート数である．tran(g_i)はゲート g_i の出力値に遷移が発生している場合は 1 を返し，遷移が発生してない場合は 0 を返す関数である．fanout(g_i)はゲート g_i の分岐信号線数を返す関数である．1 は遷移が発生したゲートの重み付けのために足し込まれる．

図 3.3.1 に，WSA の計算例を示す．図 3.3.1 の回路において，括弧内の数値は(1 時刻目信号線値，2 時刻目信号線値)を示す．図 3.3.1 では，ゲート G0，G1，FF の出力値に遷移が発生している．また，各ゲートの分岐信号線数は G0 が 0 本，G1 が 2 本，FF が 0 本となっており，式(3.3.1)より WSA は 5 と計算できる．

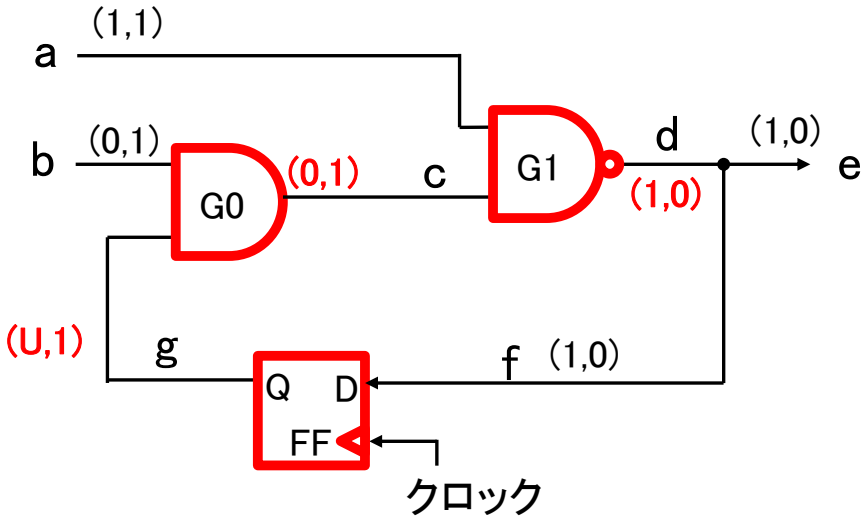


図 3.3.1：WSA 計算例

第 4 章

テストコスト削減技術

4.1　ドントケア

ドントケアとは，あるテストパターンで故障を検出するときに，故障検出に必要のない論理値である．そのため，ドントケアと判定された外部入力値は 0 または 1 のどちらに値割当てをしても，故障検出率が低下しない．ドントケアは，X または x と表記される．また，0 または 1 の論理値をケアビットという．与えられたテスト集合からドントケアを判定する手法として，ドントケア判定法[18-22]が提案されている．

図 4.1.1 にドントケアの例を示す．図 4.1.1 は，信号線 d の 1 縮退故障がテストパターン(a, b, c) = (0, 1, 0)で検出可能な例を示している．図 4.1.1 において，信号線 d の故障励起のために，信号線 a または b に 0 を割当てる必要がある．図 4.1.1 では，信号線 a=0 が割当てられており，信号線 b=1 が割当てられている．このとき，信号線 b の値は 0 または 1 のどちらでも，信号線 d の 1 縮退故障の検出に影響しない．よって，信号線 b の値はドントケアと判定できる．

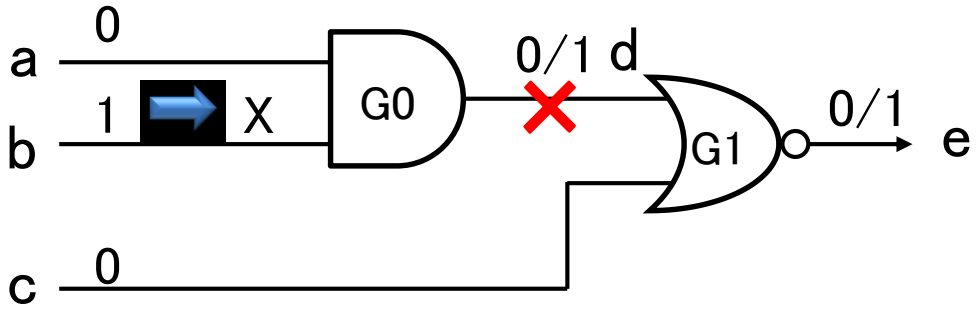
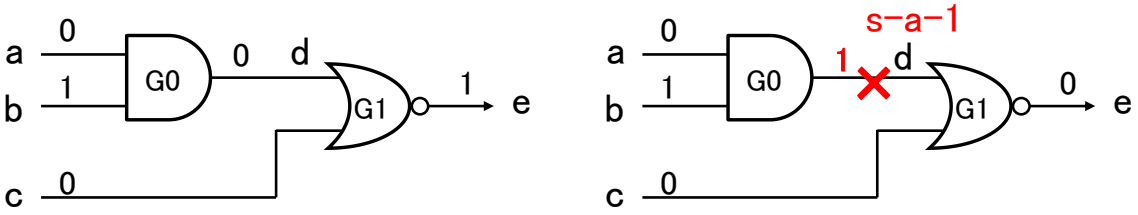


図 4.1.1：ドントケアと判定される外部入力例

4.2　故障シミュレーション

故障シミュレーションとは，回路，テスト集合，故障リスト(故障の集合)が与えられたとき，与えられたテスト集合で検出可能な故障を判定することである．故障シミュレーションの結果から，故障検出率が計算可能である．故障シミュレーションでは，故障を想定した回路(故障回路)と故障を想定していない回路(正常回路)を用いて，同一のテストパターンでシミュレーションを行う．シミュレーションの結果，正常回路と故障回路で異なる値を出力している外部出力が 1 つ以上存在している場合，想定した故障が検出可能と判定できる．また，正常回路と故障回路の外部出力値が全て一致する場合は，印加したテストパターンでは想定した故障は検出不可能と判定できる．同様の処理を各テストパターンに対して，入力された全ての故障について実行する．全てのテストパターンに対して処理が終了すると，各テストパターンで検出可能な故障集合(故障辞書)を出力する．

図 4.2.1 にテストパターン(a, b, c) = (0, 1, 0)における，信号線 d の 1 縮退故障の故障シミュレーションの例を示す．図 4.2.1(a)は，正常回路のシミュレーション例，図 4.2.1(b)は故障回路のシミュレーション例を示す．図 4.2.1 において，外部出力の値が正常回路では e=1，故障回路では e=0 となっているため，テストパターン(a, b, c) = (0, 1, 0)では信号線 d の 1 縮退故障が検出可能と判定できる．



(a) 正常回路例　　(b)故障回路例

図 4.2.1：信号線 d の 1 縮退故障の故障シミュレーション例

表 4.2.1 に，図 4.2.1 の回路に対する故障辞書の例を示す．表 4.2.1 は，テストパターン t_1，t_2，t_3，t_4，t_5，t_6 の 6 個のテストパターンに対する故障辞書の例である．表 4.2.1 において，e/1 は信号線 e の 1 縮退故障を示しており，他も同様である．表 4.2.1 において，テストパターン t_1 では，信号線 e の 1 縮退故障が検出可能と判定された．他のテストパターンに対しても，同様に検出故障情報が記述されている．

また，与えられたテスト集合内の 1 つのテストパターンしか検出できない故障を必須故障[13]という．表 4.2.1 において，信号線 c の 0 縮退故障はテストパターン t_2 でしか検出できないため，必須故障である．同様に，信号線 d の 0 縮退故障，信号線 b の 0 縮退故障，信号線 a の 0 縮退故障，信号線 a の 1 縮退故障，信号線 b の 1 縮退故障も必須故障である．

表 4.2.1：故障辞書例

テストパターンID	テストパターン(a, b, c)	検出故障リスト
t_1	1 1 1	e/1
t_2	0 1 1	e/1, **c/0**
t_3	1 1 0	e/1, **d/0**, **b/0**, **a/0**
t_4	0 0 0	e/0, d/1, c/1
t_5	0 1 0	e/0, d/1, c/1, **a/1**
t_6	1 0 0	e/0, d/1, c/1, **b/1**

4.3　テスト圧縮

VLSI のテストコスト削減技術の 1 つにテスト圧縮がある．テスト圧縮技術はテストパターン数を削減する技術で，それを用いることでテストコストを削減する．テスト圧縮には，ドントケアを利用してテストパターンを併合する手法[14][16]や，故障シミュレーションを用いて故障の被覆関係を解析して圧縮する手法[13][17]などが提案されている．

4.3.1 節でドントケアに基づくテスト圧縮，4.3.2 節で故障シミュレーションに基づくテスト圧縮について概説する．

4.3.1　ドントケアに基づくテスト圧縮

テスト圧縮手法にテストパターン中のドントケアを利用して，複数のテストパターンを 1 つのテストパターンに併合する手法[14][16]が提案されている．

4.3.1.1　テスト圧縮問題

式(4.3.1.1)に，テストパターン中の各外部入力の値を返す関数を示す．式(4.3.1.1)において，xt_iはテスト集合に含まれる i 番目のテストパターン，p_kはテストパターン xt_iの k 番目の外部入力を示す．式(4.3.1.1)は，テストパターン xt_iに対して，外部入力 p_kの値を返す関数である．式(4.3.1.1)は，テストパターン xt_iの外部入力 p_kが 0 の場合は 0 を返し，1 の場合は 1 を返し，それ以外の場合は X を返す．

$$V(xt_i, p_k) = \begin{cases} 0 & \text{if } p_k \text{ value of } xt_i \text{ is } 0 \\ 1 & \text{if } p_k \text{ value of } xt_i \text{ is } 1 \\ \mathrm{X} & \text{otherwise (X-bit)} \end{cases} \qquad (4.3.1.1)$$

表 4.3.1.1 に，式(4.3.1.1)を用いた圧縮演算$\cap_T$を示す．表 4.3.1.1 は，テストパターン xt_iと xt_jの外部入力 p_kの値が圧縮可能か否かを判定する．表 4.3.1.1 において，ϕ は圧縮不可能を示す．表 4.3.1.1 において，テストパターン xt_iと xt_jの外部入力 p_kの値が同一であれば圧縮可能である．また，テストパターン xt_iと xt_jの外部入力 p_kの値のどちらか一方に X を含む場合は，X でない値にすることで圧縮可能である．テストパターン xt_iと xt_jの外部入力 p_kの値が異なるケアビット(0 または 1)の場合，圧縮不可能である．

表 4.3.1.1：圧縮演算 $\cap_T$

$V(xt_i, p_k)$ \ $V(xt_j, p_k)$	0	1	X
0	0	φ	0
1	φ	1	1
X	0	1	X

式(4.3.1.2)に，表 4.3.1.1 より得られる圧縮演算式を示す．式(4.3.1.2)は，テストパターン xt_iと xt_jの外部入力 p_kの値に対して，表 4.3.1.1 の圧縮演算$\cap_T$を行った結果を示した式である．圧縮演算後の外部入力 p_kの値は $cxt_{pk} \in \{0, 1, X, \phi\}$となる．

$$cxt_{p_k} = V(xt_i, p_k) \bigcap\nolimits_T V(xt_j, p_k) \qquad (4.3.1.2)$$

式(4.3.1.3)に，テストパターン xt_i と xt_j の圧縮判定関数を示す．式(4.3.1.3)において，テストパターン xt_i と xt_j が圧縮可能な場合は 1 を，圧縮不可能な場合は 0 を返す．

$$COM(xt_i, xt_j) = \begin{cases} 1 & \text{if } xt_i \text{ and } xt_j \text{ are compatible} \\ 0 & \text{otherwise (incompatible)} \end{cases} \tag{4.3.1.3}$$

テストパターン xt_i と xt_j において，各外部入力に対して異なるケアビット(0 または 1)を持つ外部入力が 1 つでも存在する場合，圧縮不可能と判定できる．式(4.3.1.2)と式(4.3.1.3)より，テストパターン xt_i と xt_j が圧縮不可能な場合の式を示す．

$$\exists p_k (cxt_{pk} = \Phi) \Rightarrow COM(xt_i, xt_j) = 0 \tag{4.3.1.4}$$

テストパターン xt_i と xt_j において，全ての外部入力に対して異なるケアビット(0 または 1)を持つ外部入力が存在しない場合，圧縮可能と判定できる．式(4.3.1.2)と式(4.3.1.3)より，テストパターン xt_i と xt_j が圧縮可能な場合の式を示す．

$$\forall p_k (cxt_{pk} \neq \Phi) \Rightarrow COM(xt_i, xt_j) = 1 \tag{4.3.1.5}$$

図 4.3.1.1 にテストパターンの圧縮例を示す. 図 4.3.1.1(a)は圧縮不可能なテストパターンの例，図 4.3.1.1(b)は圧縮可能なテストパターンの例を示している．図 4.3.1.1(a)において，テストパターン t_1 と t_2 の外部入力 d の論理値が異なる．このため，テストパターン t_1 と t_2 は圧縮不可能と判定できる．図 4.3.1.1(b)では，全ての外部入力に対して異なるケアビット値を持つ外部入力が存在しない．このため，テストパターン t_1 と t_3 は，(a, b, c, d, e) = (1 0 1 1 0)に圧縮可能である．

テストパターンID	テストパターン (a, b, c, d, e)
t_1	1 X X 1 0
t_2	0 0 X 0 1

圧縮不可能

テストパターンID	テストパターン (a, b, c, d, e)
t_1	1 X X 1 0
t_3	X 0 1 X X

(a, b, c, d, e) = (1 0 1 1 0)に圧縮可能

(a)圧縮不可能なテストパターン例　　(b)圧縮可能なテストパターン例

図 4.3.1.1：テスト圧縮例

4.3.1.2　頂点彩色問題

ドントケアに基づくテスト圧縮手法に，頂点彩色問題に帰着させて解く貪欲アルゴリズム[16]が提案されている．頂点彩色問題とは，グラフの隣接する頂点が異なる色になるように色を塗り，最小の色数を求める問題である．

頂点彩色問題を利用したドントケアに基づくテスト圧縮を説明する．頂点をテストパターン，圧縮不可能なテストパターン同士を辺で結ぶことにより，テストパターンの圧縮不可能性を無向グラフとして表現することができる．これを圧縮不可能グラフという．

圧縮不可能グラフは無向グラフ G=(V,E)であり，頂点 v ∈ V はテストパターンを表す．また∀u, v ∈ V(u≠v)において，辺(u,v)∈ E は，u と v は圧縮不可能であることを表す．

表 4.3.1.2 にテスト集合例，図 4.3.1.2 に表 4.3.1.2 に対する圧縮不可能グラフの例を示す．表 4.3.1.2 において，頂点はテストパターン，辺は圧縮不可能であることを示す．表 4.3.1.2 において，テストパターン t_1 と t_2 は圧縮不可能なため，図 4.3.1.2 の頂点 t_1 と t_2 の間に辺が存在する．同様に，テストパターン t_2 と t_4，テストパターン t_3 と t_4，テストパターン t_3 と t_5，テストパターン t_4 と t_5 も圧縮不可能なため，対応する頂点間に辺が存在する．

表 4.3.1.2：テスト集合例

テストパターンID	テストパターン (a, b, c, d, e)
t_1	1XX10
t_2	00X01
t_3	X01XX
t_4	111XX
t_5	X00XX

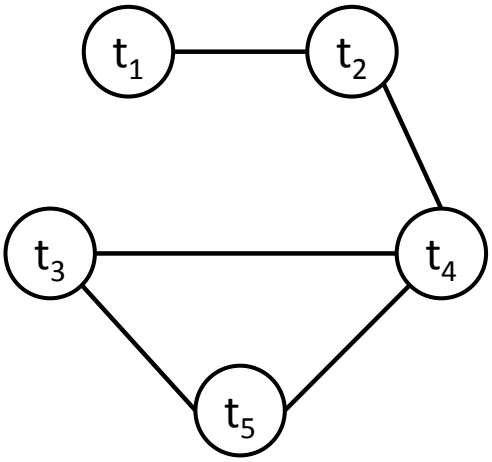


図 4.3.1.2：圧縮不可能グラフ例

図 4.3.1.3 に，図 4.3.1.2 に示したテスト集合と圧縮不可能グラフに対して，頂点彩色した例を示す．図 4.3.1.3(a)はテスト圧縮前とテスト圧縮後のテスト集合，図 4.3.1.3(b)は頂点彩色をした圧縮不可能グラフを示す．まず，t_1に青が彩色されたとして，隣接する頂点 t_2に対して青以外の色である赤に彩色をする．t_4に対しては，隣接した頂点 t_2が赤，t_3と t_5に対しては未彩色のため，赤以外の色である青に彩色する．t_3に対しては，隣接した頂点 t_4が青，t_5は未彩色であるため，青以外の色である赤に彩色する．t_5に関しては，隣接した頂点 t_3が赤，t_4が青に彩色されているため，赤と青以外の色である緑に彩色する．この例では 3 色に彩色されたため，同じ色のテストパターン t_1と t_4，t_2と t_3が圧縮され，5 個のテストパターンが 3 個に圧縮される．図 4.3.1.3(a)において，t_6は t_1と t_4が圧縮されたテストパターン，t_7は t_2と t_3が圧縮されたテストパターンを示す．

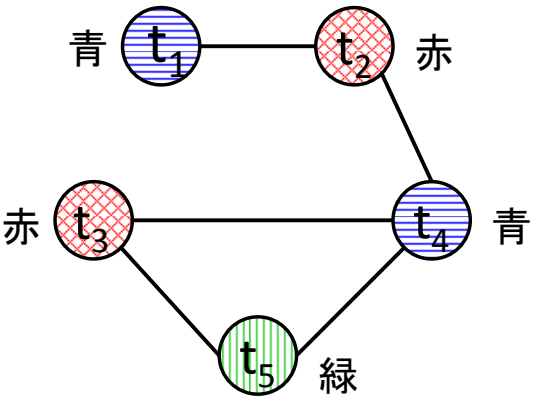
テスト圧縮前のテスト集合

テストパターンID	テストパターン (a, b, c, d, e)
t_1	1XX10
t_2	00X01
t_3	X01XX
t_4	111XX
t_5	X00XX

テスト圧縮後のテスト集合

テストパターンID	テストパターン (a, b, c, d, e)
t_6	11110
t_7	00101
t_5	X00XX

(a)テスト集合例



(b)頂点彩色問題例

図 4.3.1.3：頂点彩色問題を用いたテスト圧縮例

4.3.2　故障シミュレーションに基づくテスト圧縮

故障シミュレーションに基づくテスト圧縮法に，逆順故障シミュレーション[17]と二重検出法[13]が提案されている．4.3.2.1 節で逆順故障シミュレーション，4.3.2.2 節で二重検出法について概説する．

4.3.2.1　逆順故障シミュレーション

テスト生成により生成されたテスト集合には，冗長なテストパターンが含まれている可能性が存在する．逆順故障シミュレーションは，テスト集合内のテストパターンに対して逆順に故障シミュレーションを実行することで，冗長なテストパターンを削除するテスト圧縮手法である．テスト生成は，それまでに生成したテストパターンでは検出不可能な故障(未検出故障)に対してテストパターンを生成する．そのため，各テストパターンは生成された時点では冗長ではない．しかしながら，テスト生成の対象とした故障が，それ以降に生成されたテストパターンで偶発的に検出された場合，そのテストパターンは冗長である可能性が存在する．

表 4.3.2.1 に，t_1，t_2，t_3，t_4，t_5 のテストパターンに対する故障辞書の例を示す．表 4.3.2.1 において，t_1，t_2，t_3，t_4，t_5 はテストパターン，f_1，f_2，f_3，f_4，f_5，f_6 は故障を示す．表 4.3.2.1 において，テスト生成は t_1，t_2，t_3，t_4，t_5 の順にテストパターンを生成したとする．また，テスト生成は t_1 は f_1，t_2 は f_3，t_3 は f_4，t_4 は f_5，t_5 は f_6 を対象にテストパターンを生成したとする．t_1 の f_2，t_2 の f_2，t_3 の f_1，t_4 の f_3，t_5 の f_5 はテスト生成の対象とされて故障ではなく，故障シミュレーションにより偶発的に検出された故障である．また，f_4 と f_6 は，それぞれ t_3 と t_5 でのみ検出可能故障のため必須故障である．

表 4.3.2.1：故障辞書例

テストパターンID	検出故障リスト
t_1	f_1 ， f_2
t_2	f_3 ， f_2
t_3	f_4 ， f_1
t_4	f_5 ， f_3
t_5	f_6 ， f_5

表 4.3.2.2 に，逆順故障シミュレーションの例を示す．表 4.3.2.2(a)は通常の故障シミュレーションの例，表 4.3.2.2(b)は．逆順故障シミュレーションの例を示している．表 4.3.2.2 において，テストパターン ID は故障シミュレーションの対象となるテストパターン，故障シミュレーション対象故障リストは故障シミュレーションの対象となる故障　を示す．故障はどれか 1 個のテストパターンで検出されれば十分である．そのため，故障シミュレーションは，故障シミュレーションを実行するたびに故障シミュレーション対象故障リストを更新する．検出故障リストは，故障シミュレーションの結果，シミュレーションを実行したテストパターンで検出できた故障を示す．

表 4.3.2.2(a)は，t_1，t_2，t_3，t_4，t_5 の順に故障シミュレーションを実行する．t_1 に対する故障シミュレーションでは，f_1，f_2，f_3，f_4，f_5，f_6 が故障シミュレーションの対象となり，f_1，f_2 が検出された．t_2 に対する故障シミュレーションでは，　f_3，f_4，f_5，f_6 が故障シミュレーションの対象となり，　f_3 が検出された．t_3 に対する故障シミュレーションでは，　f_4，f_5，f_6 が故障シミュレーションの対象となり，　f_4 が検出された．t_5 に対する故障シミュレーションでは，　f_5，f_6 が故障シミュレーションの対象となり，f_5 が検出された．t_6 に対する故障シミュレーションでは，f_6 が故障シミュレーションの対象となり，　f_6 が検出された．t_1，t_2，t_3，t_4，t_5 の順に故障シミュレーションのを実行すると，全ての故障を検出するのに t_1，t_2，t_3，t_4，t_5 の 5 個のテストパターンが必要である．

表 4.3.2.2(b)は，表 4.3.2.2(a)の逆順である t_5，t_4，t_3，t_2，t_1 の順に故障シミュレーションを実行する．t_5 に対する故障シミュレーションでは，f_1，f_2，f_3，f_4，f_5，f_6 が故障シミュレーションの対象となり，f_6，f_5 が検出された．t_4 に対する故障シミュレーションでは，f_1，f_2，f_3，f_4 が故障シミュレーションの対象となり，f_3 が検出された．t_3 に対する故障シミュレーションでは，f_1，f_2，f_4 が故障シミュレーションの対象となり，f_4，f_1 が検出された．t_2 に対する故障シミュレーションでは，f_2 が故障シミュレーションの対象となり，f_2 が検出された．t_5 から t_2 までの故障シミュレーションで全ての故障が検出されたため，t_1 は冗長なテストパターンであることがわかる．逆順故障シミュレーションの結果から，t_1 を削除した t_2，t_3，t_4，t_5 の 4 個のテストパターンで全ての故障を検出可能と判定できる．

表 4.3.2.2：逆順故障シミュレーションの例

(a) 通常故障シミュレーション

テストパターンID	故障シミュレーション対象故障リスト	検出故障リスト
t_1	f_1，f_2，f_3，f_4，f_5，f_6	f_1，f_2
t_2	f_3，f_4，f_5，f_6	f_3
t_3	f_4，f_5，f_6	f_4
t_4	f_5，f_6	f_5
t_5	f_6	f_6

(b) 逆順故障シミュレーション

テストパターンID	故障シミュレーション対象故障リスト	検出故障リスト
t_5	f_1，f_2，f_3，f_4，f_5，f_6	f_6，f_5
t_4	f_1，f_2，f_3，f_4	f_3
t_3	f_1，f_2，f_4	f_4，f_1
t_2	f_2	f_2
t_1	なし	なし

4.3.2.2　二重検出法

二重検出法とは，各テストパターンの必須故障情報を基にしたテスト圧縮法である．二重検出法は，必須故障情報よりテスト集合に含まれる冗長なテストパターンを削除する．必須故障とは，テスト集合内の 1 個のテストパターンでしか検出できない故障である．また，必須故障を検出するテストパターンを必須テストパターン[13]という．それとは逆に，必須故障を検出しないテストパターンを冗長テストパターン[13]という．二重検出法により，ドントケアに基づくテスト圧縮や逆順故障シミュレーションでは削減できないテストパターンを削減可能である．

表 4.3.2.3 に，必須故障，必須テストパターン，冗長テストパターンの例を示す．表 4.3.2.3 では，故障 f_1，f_2，f_3，f_4，f_5，f_6 を検出するテストパターンとして，テストパターン t_1, t_2, t_3, t_4, t_5 を示している．表 4.3.2.3(a)は各故障に対する検出テストパターンリスト，表 4.3.2.3(b)は各テストパターンに対する検出故障リストを示している．表 4.3.2.3(a)において，故障 f_4，f_6 は，それぞれテストパターン t_3，t_5 でしか検出できない．したがって，故障 f_4，f_6 は必須故障である．また，テストパターン t_3，t_5 は必須テストパターンである．f_1，f_2，f_3，f_5 はそれぞれ 2 個のテストパターンで検出されるため，必須故障ではない．また，f_1，f_2，f_3，f_5 しか検出しないテストパターンは冗長テストパターンである．

表 4.3.2.3：必須故障と必須テストパターン例

(a) 検出テストパターンリスト

故障	検出テストパターンID
f_1	t_1 , t_3
f_2	t_1 , t_2
f_3	t_2 , t_4
f_4	t_3（必須）
f_5	t_4 , t_5
f_6	t_5（必須）

(b)検出故障リスト

テストパターンID	検出故障リスト
t_1	f_1 , f_2
t_2	f_3 , f_2
t_3	f_4 , f_1
t_4	f_5 , f_3
t_5	f_6 , f_5

二重検出法では，必須テストパターンと冗長テストパターンの判定を行う．故障シミュレーションにより，表 4.3.2.4 のようなテストパターンと故障リストの情報を作成する．

表 4.3.2.4(a)に check の計算例を示す．check とは，各故障に対して異なるテストパターンで検出できた回数を最大 2 回まで計算したものである．check の初期値は 0 であり，必須故障の場合は check が 1，複数のテストパターンで検出可能な故障の場合は check が 2，未検出故障の場合は check が 0 となる．したがって，check の値の取りうる範囲は 0，1，2 となる．二重検出法では，check が 2 の故障に対しては，故障シミュレーションの対象から除外する．表 4.3.2.4(a)では，故障 f_4,f_6 は必須故障であるため check はそれぞれ 1 と計算される．また，故障 f_1，f_2，f_3，f_5 はそれぞれ 2 個のテストパターンで検出されるため，check は 2 と計算される．

表 4.3.2.4(b)に one_check の計算例を示す．one_check とは，各テストパターンの必須故障検出数である．one_check の初期値は 0 である．one_check の値が 1 以上のテストパターンは必須テストパターンと判定できる．また，one_check の値が 0 のテストパターンは冗長テストパターンと判定できる．表 4.3.2.4(b)において，テストパターン t_3，t_5 はそれぞれ必須故障を 1 個検出する．そのため，テストパターン t_3，t_5 の one_check はそれぞれ 1 と計算できる．また，テストパターン t_1，t_2，t_4 は必須故障を 1 個も検出しないため，one_check は 0 と計算される．

表 4.3.2.4：必須故障と冗長テストパターンの判定例

(a) check 計算例

故障	第一検出 テストパターンID	check
f_1	t_1	2
f_2	t_1	2
f_3	t_2	2
f_4	t_3	1
f_5	t_4	2
f_6	t_5	1

(b) one_check 計算例

テスト パターンID	検出故障リスト	one_check
t_1	f_1 ， f_2	0
t_2	f_3 ， f_2	0
t_3	f_4 ， f_1	1
t_4	f_5 ， f_3	0
t_5	f_6 ， f_5	1

以下に二重検出法の処理手順を示す．二重検出法では，与えられたテスト集合に対して，以下の処理を適用する．

Step 1：

入力されたテスト集合に対して，故障シミュレーションを実行する．故障シミュレーションの結果から，各テストパターンの one_check と，各故障の check の値を計算する．

Step 2：

one_check の値が 1 以上のテストパターンを必須テストパターン，0 のテストパターンを冗長テストパターンと分類する．冗長テストパターン数が 0 個の場合は処理を終了する．冗長テストパターン数が 1 個の場合は，そのテストパターンを削除して終了する．冗長テストパターン数が 2 個以上の場合は，Step3 へ進む．

Step 3：

check と one_check の値を 0 に初期化し，必須テストパターンを用いて回路内の全ての故障を対象に故障シミュレーションを実行する．

Step 4：

冗長テストパターンを用いて，Step3 での未検出故障を対象に逆順故障シミュレーションを実行する．逆順故障シミュレーションの結果，未検出故障を 1 個も検出しない冗長テストパターンを削除する．Step2 へ戻る．

二重検出法の実行例を示す．まず，Step1 の処理で，表 4.3.2.4 の結果が得られたとする．Step2 で，one_check の値よりテストパターン t_3，t_5 が必須テストパターン，t_1，t_2，t_4 は冗長テストパターンであると判定できる．ここで，冗長テストパターン数が 3 個であるため Step3 へ進む．Step3 では，必須テストパターン t_3，t_5 に対して故障シミュレーションを実行すると，f_1，f_4，f_5，f_6，が検出可能であり，f_2，f_3 が未検出と判定できる．Step4 では，冗長テストパターン t_1，t_2，t_4 に対して，t_4，t_2，t_1 の順に逆順故障シミュレーションを実行する．このとき，t_4 は未検出故障 f_3 を検出し，t_2 は未検出故障 f_2 を検出する．t_1 の故障シミュレーションの時点で未検出故障は残っていないため，t_1 を削除する．この結果，テストパターンと故障リストは表 4.3.2.5 のようになる．表 4.3.2.3(b)において，冗長テストパターン t_4 が 1 個存在する．そのため，Step2 に戻り，t_4 を削除し処理が終了される．二重検出法の結果，t_2，t_3，t_6 で全故障を検出でき，テストパターン数が 5 個から 3 個に圧縮された．

表 4.3.2.5：二重検出法の適用

(a)Step4 時の check 計算例

故障	第一検出 テストパターンID	check
f_1	t_3	1
f_2	t_2	1
f_3	t_2	2
f_4	t_3	1
f_5	t_4	2
f_6	t_5	1

(b) Step4 時の one_check 計算例

テスト パターンID	検出故障リスト	one_check
t_2	f_3 ， f_2	1
t_3	f_4 ， f_1	2
t_4	f_5 ， f_3	0
t_5	f_6 ， f_5	1

第 5 章

テスト圧縮を考慮したドントケア判定

5.1　緒言

入力されたテスト集合からドントケアを判定する手法としてドントケア判定法[18][19]が提案されている．しかしながら，文献[18][19]で提案されたドントケア判定法は，各外部入力に対してドントケア数の均一化を考慮していない[20-22]．テスト圧縮の分野において，特定の外部入力にケアビットが集中するとテスト圧縮効率が低下することが報告されている[20]．そのため，文献[18][19]のドントケア判定法はテスト圧縮の分野において，効果的でない可能性がある．本章では，各外部入力のドントケア数を考慮したテスト圧縮に効果的なドントケア判定法を提案する．また，ISCAS'89，ITC'99 ベンチマーク回路に対して提案手法の有効性を評価する．

5.2 節でドントケア判定問題の定式化と，ドントケア判定法について概説する．5.3 節で提案手法である，テスト圧縮を考慮したドントケア判定法について説明する．5.4 節で結言を述べる．

5.2　ドントケア判定

本節では，ドントケア判定問題の定式化と，ドントケア判定アルゴリズムについて概説する．5.2.1 節でドントケア判定問題の定式化をし，5.2.2 節でドントケア判定アルゴリズムについて概説する．5.2.3 節でドントケア判定における，故障伝搬経路の決定規則を概説する．また，5.2.4 節で限定含意操作と限定正当化操作を概説し，5.2.5 節で見逃し故障を概説する．

5.2.1　ドントケア判定問題定式化

本節では，ドントケア判定問題の定式化を行う．ドントケア判定では初期テスト集合 T と回路 C が入力され，ドントケアを含むテスト集合 XT が出力される．本章で取り扱う初期テスト集合 T は，テスト生成で生成された単一縮退故障のテスト集合である．回路 C は組合せ回路またはフルスキャン設計を施した順序回路である．T と C が与えられたとき，XT は以下の問題の解を求めることにより導出される．

(1)　XT は T を被覆する
(2)　XT は可能な限り多くのドントケアを含む
(3)　XT と T の故障検出率は等しい

5.2.2　ドントケア判定アルゴリズム

本節では文献[18]で提案されたドントケア判定のアルゴリズムを概説する．図 5.2.2.1 にドントケア判定のアルゴリズムを示す．

```
1.  Procedure X-Identification(C,T,)
2.  C :  circuit , T : initial_test_set  ;
3.  {
4.      EXT = Φ;
5.      UXT = Φ;
6.      XT = Φ;
7.      D = fault_simulation (C, T ) ;
8.      for each test_pattern t_i in T {
9.          EXT += essential_X-filling (C, D, t_i ) ;
10.     }
11.     U = collect_undetected_fault (C, D, EXT ) ;
12.     for each test_pattern t_i in T {
13.         UXT += undetected_X-filling (C, D, t_i , ext_i ) ;
14.     }
15.     U = collect_undetected_fault (C, D, UXT ) ;
16.     for each test_pattern t_i in T {
17.         XT += missed_X-filling (C, D, t_i , uxt_i ) ;
18.     }
19.     return XT ;
20. }
```

図 5.2.2.1：ドントケア判定アルゴリズム

与えられた初期テスト集合 Tから，ドントケア判定後のテスト集合 XTを得るまでの処理手順を説明する．ここで入力は回路 Cと，初期テスト集合 Tである．まず，ドントケア判定後のテスト集合を保存する変数 EXT, UXT, XTをそれぞれΦに初期化する(行 4-6)．次に，初期テスト集合 Tに対して故障シミュレーションを実行し，故障辞書 D を算出する(行 7)．Dを用いて，Tに含まれる各テストパターン t_iに対して必須故障の検出に必要な外部入力値を求め，EXT に格納する．EXT は必須故障の検出のみを保証したドントケアを含むテスト集合である(行 8-10)．EXTは必須故障の検出のみを保証したテスト集合であるが，偶発的に他の故障も検出する可能性がある．そのため，EXTに対して故障シミュレーションを実行し，未検出故障リスト U を算出する(行 11)．T に含まれる各テストパターン t_iに対して行 13 の処理を実行する(行 12)．t_iで検出可能な未検出故障 Uに対して，ext_iと t_iから故障検出に必要な外部入力値を求め，ext_iを更新し，UXTに格納する．UXTは必須故障と未検出故障の検出を保証したドントケアを含むテスト集合である(行 13)．UXT を用いて故障シミュレーションを実行し，未検出故障リスト Uを更新する．文献[18]で提案されたドントケア判定法では，回路内の正常値のみを用いて故障検出に必要なケアビットを決定する．そのため，正常値のみでの計算では検出不可能な故障が存在する．このような故障を見逃し故障という(行 15)．見逃し故障については 5.2.4 節で述べる．行 15 で算出した未検出故障(見逃し故障)に対して，Tに含まれる各テストパターン t_iに対して未検出故障の検出に必要な外部入力値を求め，XTに格納する．XTは Tと故障検出率が等しいドントケアを含むテスト集合である(行 16-18)．XTを返し，終了する(行 19)．

5.2.3　故障伝搬経路の決定規則

ドントケア判定は，故障シミュレーションによる内部信号線値の計算後，外部出力から故障箇所まで故障伝搬経路を遡る．このとき，故障伝搬経路と故障伝搬に必要な信号線と信号線値をスタックに保存する．スタックに保存された信号線と信号線値は，限定含意操作[18]，限定正当化操作[18]，拡張含意操作[18]を用いて故障検出に必要な信号線値を決定する．このとき，故障の検出に必要のない外部入力値をドントケアとする．

本節では，故障伝搬経路の決定規則[18]を説明する．また，5.2.4 節で限定含意操作と限定正当化操作，5.2.5 節で見逃し故障および拡張含意操作を説明する．

故障伝搬経路の決定規則 1

規則 1 は，故障伝搬しているゲートの入力信号線に故障伝搬信号線が 1 本の時に適用する．このとき，故障伝搬しているゲートの入力信号線のみ遡り，他の入力信号線と信号線値をスタックに保存する．

図 5.2.3.1 に故障伝搬経路の決定規則 1 の例を示す．図 5.2.3.1 において，信号線 a，b，c，d，e，f は内部信号線，信号線 g は外部出力である．外部出力 g からの故障伝搬経路の決定を考える．ゲート G2 において，入力信号線 e は故障値が伝搬しており，f は故障値が伝搬していない．そのため，信号線 e は故障伝搬経路として遡り，信号線 f とその信号線値をスタックに保存する．

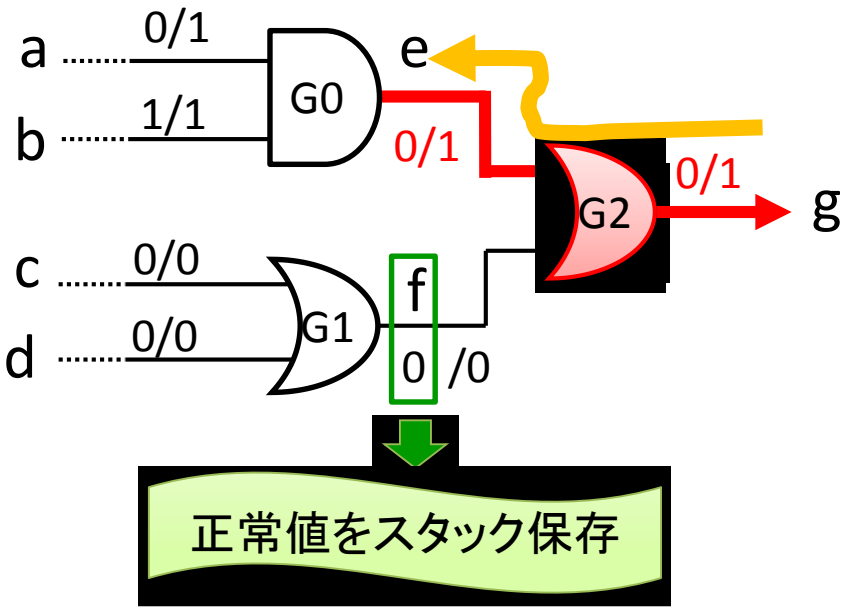


図 5.2.3.1：故障伝搬経路の決定規則 1 の例

故障伝搬経路の決定規則 2

規則 2 は，故障伝搬しているゲートの入力信号線に故障伝搬信号線が 2 本以上で，故障伝搬している入力信号線の正常値が非制御値のときに適用する．このとき，故障伝搬しているゲートの入力信号線のどれか 1 つを遡り，他の入力信号線と信号線値をスタックに保存する．これは，故障検出を保証するには少なくとも 1 つの故障伝搬経路を確保すればよいからである．また，遡る信号線はどれを選択しても構わない．

図 5.2.3.2 に故障伝搬経路の決定規則 2 の例を示す．図 5.2.3.2 において，信号線 a，b，c，d，e，f は内部信号線，信号線 g は外部出力である．外部出力 g からの故障伝搬経路の決定を考える．ゲート G2 において，入力信号線 e と f に故障値が伝搬しており，e と f の正常値は G2 の非制御値である．そのため，e か f のどちらか一方のみを故障伝搬経路として遡る．図 5.2.3.2 では，e を故障伝搬経路として遡り，f とその値をスタックに保存する．

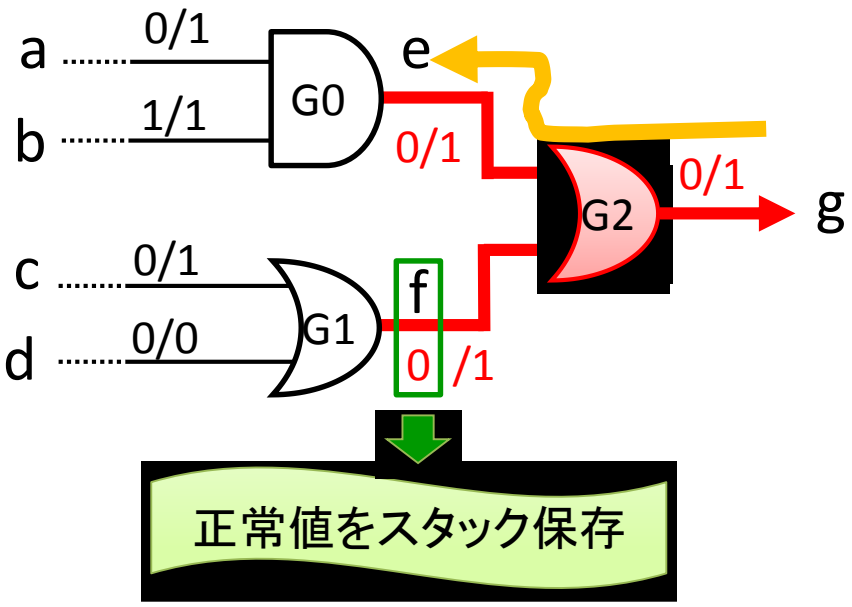


図 5.2.3.2：故障伝搬経路の決定規則 2 の例

故障伝搬経路の決定規則 3

規則 3 は，故障伝搬しているゲートの入力信号線に故障伝搬信号線が 2 本以上で，故障伝搬している入力信号線の正常値が制御値のときに適用する．このとき，故障伝搬しているゲートの入力信号線の全てを遡り，故障伝搬していない入力信号線と信号線値をスタックに保存する．故障伝搬している入力信号線の正常値が制御値の場合は，故障伝搬している全ての入力信号線が故障の検出に必要な経路となる．

図 5.2.3.3 に故障伝搬経路の決定規則 3 の例を示す．図 5.2.3.3 において，信号線 a，b，c，d，e，f は内部信号線，信号線 g は外部出力である．外部出力 g からの故障伝搬経路の決定を考える．ゲート G2 において，入力信号線 e と f に故障値が伝搬しており，e と f の正常値は G2 の制御値である．そのため，e と f の両方を故障伝搬経路として遡る．

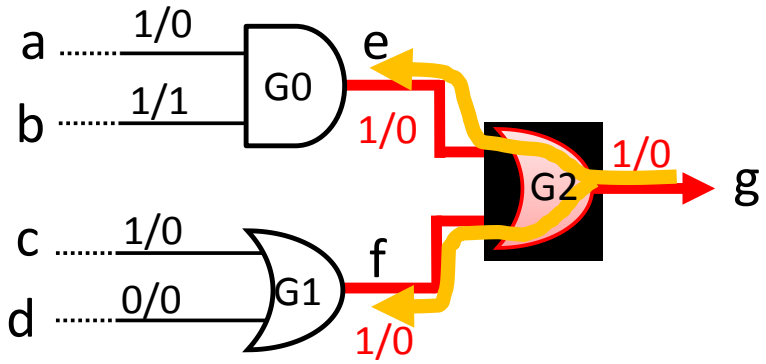


図 5.2.3.3：故障伝搬経路の決定規則 3 の例

図 5.2.3.4 に，規則 1，2，3 に従い故障伝搬経路を求める例を示す．図 5.2.3.4 では，信号線 b の 1 縮退故障が外部出力 r まで伝搬している．このとき，外部出力 r から信号線 b へ故障伝搬経路を遡りながら，内部信号と内部信号値をスタックに保存する．まず，ゲート G6 の入力信号線の内，故障伝搬しているのは信号線 q のみであるため，規則 1 を適用する．これにより，故障伝搬経路である信号線 q を遡り，外部入力 f とその正常値をスタックに保存する．次にゲート G5 では，信号線 n と p の両方に故障伝搬されており，n と p の正常値はゲート G5 の非制御値であるため規則 2 が適用され， n か p のどちらか一方を選択して遡る．この例では n を選択して遡り，p とその正常値をスタックに保存する．ゲート G4 では信号線 k と m の両方に故障伝搬しており，k と m の正常値はゲート G4 の制御値であるため規則 3 が適用され，信号線 k と m の両方を遡る．この時，スタックには何も保存されない．ゲート G1 と G2 において，それぞれ信号線 h と i から故障伝搬しているため，規則 1 が適用される．このとき故障伝搬信号線 h と i の分岐元信号線 g まで遡り，スタックには外部入力 a と e とその正常値が保存される．ゲート G0 において，外部入力 b から故障伝搬しているため，規則 1 が適用され，外部入力 b を遡り，外部入力 c とその正常値をスタックに保存する．ここで故障箇所である外部入力 b に到達したため，最後に外部入力 b とその正常値をスタックに保存して終了する．この例の場合，最終的にスタックに保存される信号線は a，b，c，e，f，p の 6 本であり，それぞれの信号線値は 0，0，1，0，0，1 である．

また，スタックに保存された信号線と信号線値については，限定含意操作，限定正当化操作，拡張含意操作を用いて外部入力まで正当化処理を行う．

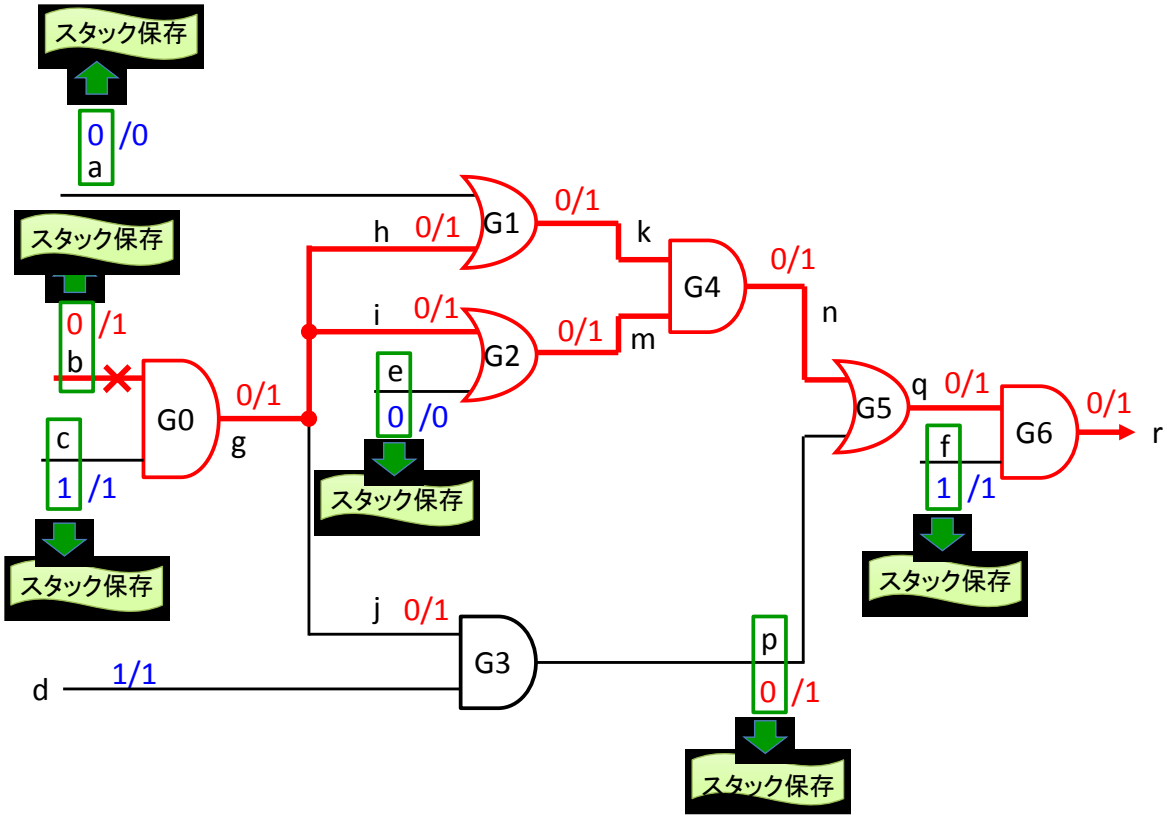


図 5.2.3.4：故障伝搬経路の決定例

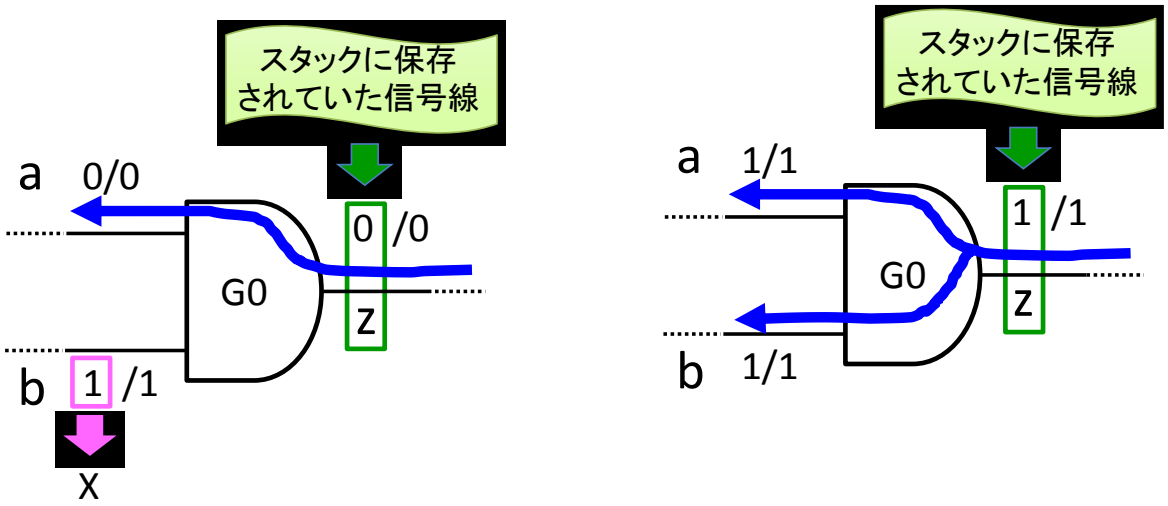
5.2.4　限定含意操作と限定正当化操作

ドントケア判定において，故障伝搬経路の決定後，スタックに保存されている信号線とその信号線値を外部入力まで遡り，故障検出に必要な外部入力を特定する．このとき，テスト生成で用いられる含意操作と正当化操作に類似した処理を適用する．この処理を限定含意操作と限定正当化操作という．限定含意操作と限定正当化操作では，ドントケア判定前の論理値と矛盾しないように設定していくため，設定する論理値に制限がある．また，限定含意操作と限定正当化操作は外部入力に到達したら終了となる．本節では，限定含意操作と限定正当化操作について説明する．

5.2.4.1　限定含意操作

限定含意操作は，スタックに保存されている信号線値を入力方向に遡るとき，入力信号線の正常値に制御値をもつ信号線が 1 本の場合，または全ての入力信号線の正常値が非制御値の場合に適用する．入力信号線の正常値に制御値をもつ信号線が 1 本の場合は，その信号線を遡り，他の信号線を X と判定する．全ての入力信号線の正常値が非制御値の場合は，全ての入力信号線を遡る．

図 5.2.4.1 に限定含意操作の例を示す．図 5.2.4.1 において信号線 a, b, z は内部信号線を表し，z は故障伝搬経路の決定時にスタックに保存された信号線である．図 5.2.4.1(a)は z から入力信号線を遡るとき，入力信号線 a しか制御値をもつ信号線が存在しない．そのため，限定含意操作により a を遡り，b を X と判定する．図 5.2.4.1(b)は，入力信号線 a，b の両方とも正常値に非制御値をもつ信号線である．そのため，限定含意操作により a，b の両方の信号線を遡る．



(a)制御値入力の場合　　(b)非制御値入力の場合

図 5.2.4.1：限定含意操作の例

5.2.4.2　限定正当化操作

限定正当化操作は，スタックに保存されている信号線値を入力方向に遡るとき，入力信号線の正常値に制御値をもつ信号線が 2 本以上存在する場合に適用する．限定正当化操作では，正常値に制御値をもつ信号線を 1 本選択し，その信号線を遡る．また，それ以外の信号線を X と判定する．

図 5.2.4.2 に限定正当化操作の例を示す．図 5.2.4.2 において信号線 a，b，c，z は内部信号線を表し，z は故障伝搬経路の決定時にスタックに保存された信号線である．信号線 a の正常値は非制御値であるが，b と c は制御値が割当てられている．このとき，b と c のどちらの信号線を遡ってもかまわない．例では信号線 b を選択し遡る．選択されなかった信号線 a と c は X と判定される．

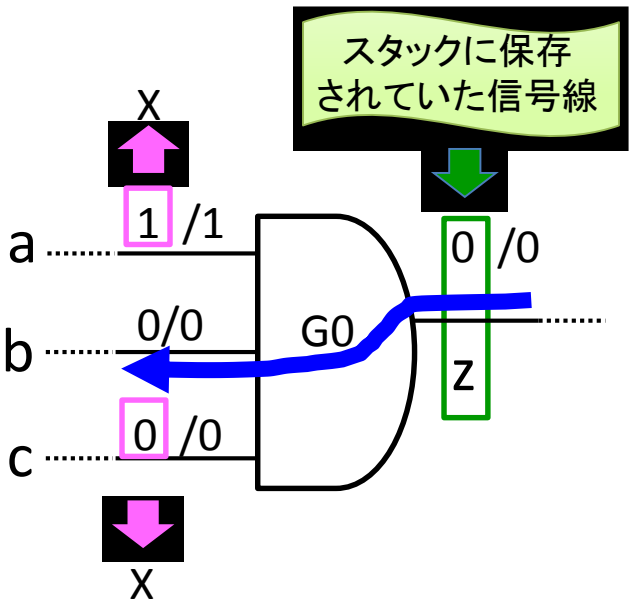


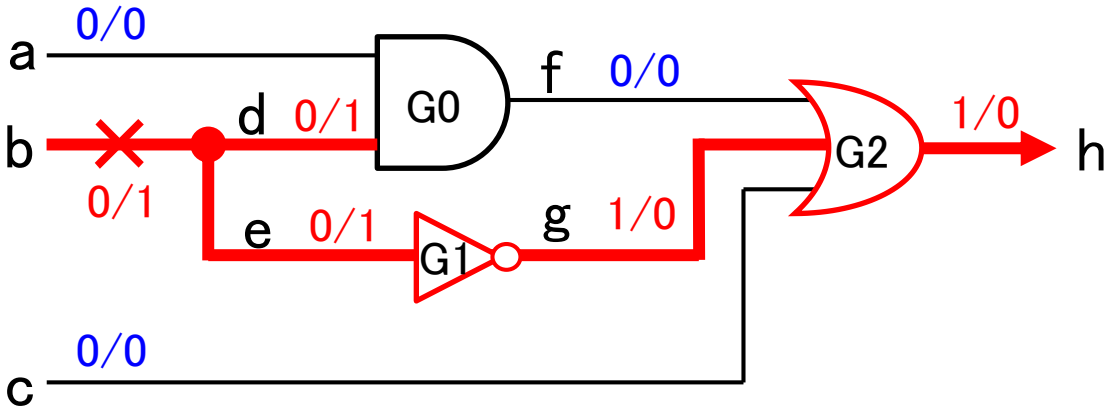
図 5.2.4.2：限定正当化操作の例

5.2.5　見逃し故障

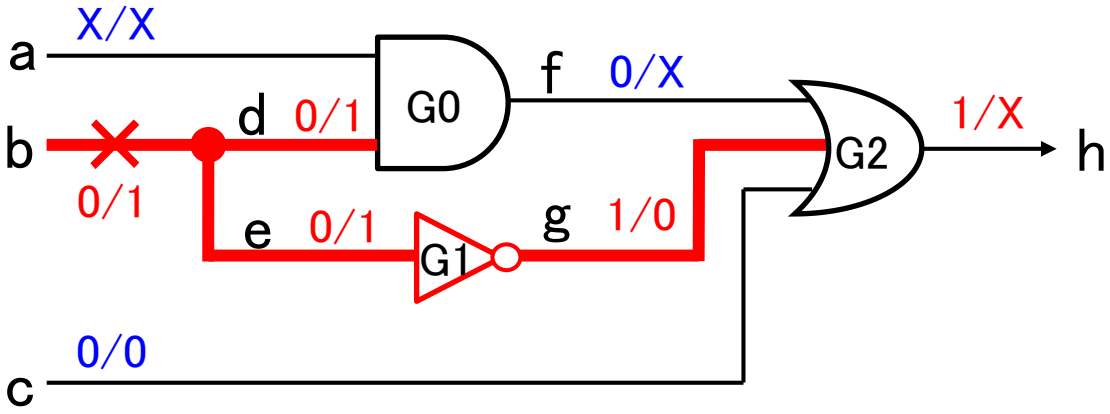
限定含意操作と限定正当化操作は正常値のみを考慮し，故障値を考慮していない．そのため，限定含意操作と限定正当化操作ではドントケア判定後のテストパターンで検出不可能な故障が存在する．このような故障を見逃し故障という．

図 5.2.5.1 に見逃し故障の例を示す．図 5.2.5.1 (a)はドントケア判定前の初期テストパターンの回路状態である．初期テストパターン(a，b，c)＝(0，0，0)は外部入力 b の 1 縮退故障を検出する．この時，故障伝搬経路を見ると b→e→g→h となっている．ここで故障励起のための内部信号値の割当てを行うと，故障励起化のために b=0 が割当てられ，かつ故障伝搬のために f=0，c=0 が割当てられ，スタックに保存される．ここで，信号線 a の値を X と見なすことができる．しかしながら，信号線 a の値を X としたテストパターンを印加した場合，図 5.2.5.1 (b)のように，初期テストパターンでは検出することができた信号線 b の 1 縮退故障を検出することができなくなる．

このように見逃し故障が発生する理由は，限定含意操作と限定正当化操作が正常値のみを計算し，故障値を考慮してないためである．図 5.2.5.1 (b)の場合，信号線 b の 1 縮退故障を検出するためには f=0 という条件が必要となってくるが，b=0(正常値)によって f=0(正常値)が含意され，限定含意操作によって信号線 f の故障値は X になる．このような故障検出の見逃しを避けるために，拡張含意操作を行う．



(a)初期テストパターン



(b)ドントケア判定後のテストパターン

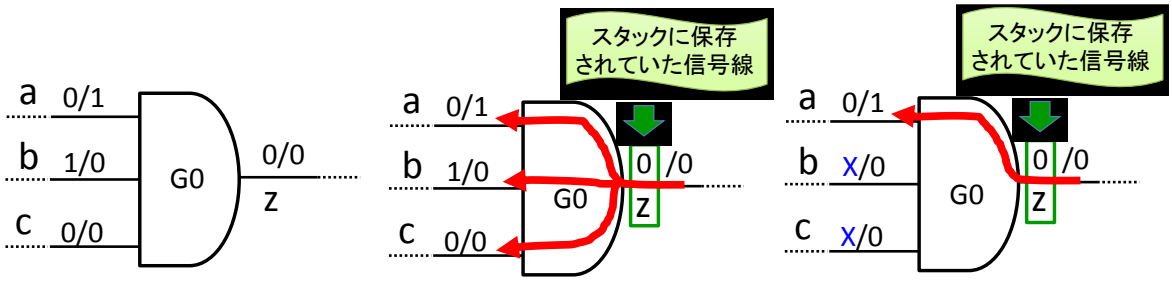
図 5.2.5.1：見逃し故障の例

拡張含意操作

拡張含意操作は，スタックに保存されている信号線値を入力方向に遡るとき，信号線値に関係なく全ての信号線を遡る．

図 5.2.5.2 に拡張含意操作の例を示す．図 5.2.5.2(a)は初期テストパターンの内部状態を表している．ゲート G0 の正常入力値が(a，b，c)＝(0，1，0)であり，信号線 a，b にそれぞれ 0/1，1/0 の故障が伝搬してきていたとする(図 5.2.5.2 (a))．この場合，限定含意操作はどの値も決定することができず，限定正当化操作は信号線 a，もしくは信号線 c のどちらかの値を決定するだけである(図 5.2.5.2 (c))．拡張含意操作はこの例の場合，信号線 a，b，c 全ての値を(a，b，c)＝(0，1，0)と決定する(図 5.2.5.2 (b))．

拡張含意操作を用いると，確実に元のテストパターンで検出した故障を検出することが可能である．しかしながら，この操作を用いることで故障の検出に関係のない値まで固定してしまう可能性があるため，ドントケアを判定するという本来の目的から外れてしまう．そのため，拡張含意操作は多くの故障に対して適用するのではなく，見逃し故障に対してのみ行う．



(a)初期テストパターン　(b)拡張含意操作　(c)限定含意操作

図 5.2.5.2：拡張含意操作の例

5.3　テスト圧縮指向ドントケア判定

文献[18][19]で提案されたドントケア判定法は，各外部入力に対してドントケア数の均一化を考慮していない[20-22]．そのため，テスト圧縮の分野において，特定の外部入力にケアビットが集中するとテスト圧縮効率が低下することが報告されている[20]．本節では，テスト圧縮指向ドントケア判定について説明する．5.3.1 節で，予備実験として外部入力のドントケア分散とテスト圧縮について述べる．5.3.2 節で，提案手法であるテスト圧縮指向ドントケア判定について述べる．5.3.3 節で，実験結果を述べる．

5.3.1　外部入力のドントケア分散とテスト圧縮

本節では，予備実験として外部入力のドントケアの偏りとテスト圧縮の関係を解析する．外部入力のドントケアの偏りの評価は分散を利用する．

式(5.3.1.1)にテストパターン中のある外部入力値がドントケアか否かを判定する関数を示す．式(5.3.1.1)において，xt_iはドントケアを含むテスト集合の i 番目のテストパターン，p_kは xt_iの k 番目の外部入力を示す．式(5.3.1.1)では，xt_iの p_kの値がドントケアの場合 1 を，それ以外の場合 0 を返す．

$$X(xt_i, p_k) = \begin{cases} 1 & \text{if } p_k \text{ value of } xt_i \text{ is an X-bit} \\ 0 & \text{otherwise (care bit)} \end{cases} \qquad (5.3.1.1)$$

式(5.3.1.2)にテスト集合中の各外部入力に含まれるドントケア数の平均値を算出する関数を示す．式(5.3.1.2)において，XT はドントケアを含むテスト集合，N_{PI}は外部入力数，xt_nは XTに含まれる n 番目のテストパターン，p_mは m 番目の外部入力を示す．

$$A_X(XT) = \frac{1}{N_{PI}} \sum_{m=1}^{N_{PI}} \left(\sum_{n=1}^{N_{TP}(XT)} X(xt_n, p_m) \right) \qquad (5.3.1.2)$$

式(5.3.1.3)にテスト集合中の外部入力のドントケア分散を算出する関数を示す.

$$s^2(XT) = \frac{1}{N_{PI}} \sum_{i=1}^{N_{PI}} (A_X(XT) - \sum_{j=1}^{N_{TP}(XT)} X(xt_j, p_i))^2 \qquad (5.3.1.3)$$

予備実験内容について説明する．予備実験では，テスト生成で生成した初期テスト集合 Tに対して，ランダムでテストパターンの論理値をドントケアに置き換えたテスト集合 RXTを 1000 種類作成した．各 RXTのドントケア数は等しく，外部入力のドントケアの分散はそれぞれ異なるテスト集合である．また，予備実験のため RXTは Tと等しい故障検出率は保証していない．その後，各 RXTに対して，ドントケアに基づくテスト圧縮[16]を適用し，外部入力のドントケアの分散とテスト圧縮の関係を解析した．外部入力のドントケア分散は式(5.3.1.3)を用いて算出した.

図 5.3.1.1 に，ITC'99 ベンチマーク回路の b14 の予備実験結果を示す．初期テスト集合 Tは Synopsys 社の TetraMAX ATPG を用いて生成した．また，TetraMAX ATPG によるテスト生成時に動的圧縮を適用している．RXT 生成時におけるドントケアの割合は，テスト集合中の 70%，80%，90%の 3 種類を用いて，計 3000 種類のテスト集合を生成した．図 5.3.1.1 において，縦軸は各 RXTのテスト圧縮後のテストパターン数，横軸は各 RXTの外部入力のドントケア分散を表している．また X-bit 70%は RXTに含まれるドントケアの割合が 70%, X-bit 80%は RXTに含まれるドントケアの割合が 80%, X-bit 90%は RXTに含まれるドントケアの割合が 90%であることを示している．図 5.3.1.1 の X-bit 90%のプロットに着目すると，外部入力のドントケア分散が 30000 以下の場合，分散値が低いほど圧縮後のテストパターン数が大きく削減されてる．しかしながら，外部入力のドントケア分散が 30000 以上の場合は, 圧縮後のテストパターン数に変化が無い．X-bit 80%のプロットに関しても，分散値が 20000 を境に同様の傾向が見られる．しかしながら，X-bit 70%のプロットに関しては，分散値の変化に対して圧縮後のテストパターン数の差はほとんど見られな

かった．また，X-bit 90%，X-bit 80%，X-bit 70%の 3 つのプロットについて比較すると，外部入力のドントケア分散が同じ値でも，ドントケア数が多い X-bit 90%のプロットが最も圧縮後のテストパターン数が少ないことがわかる．予備実験結果より，ドントケアに基づくテスト圧縮は，外部入力のドントケアの分散とテスト集合中のドントケア数によって，圧縮後のテストパターン数に差が生じることが確認できた．

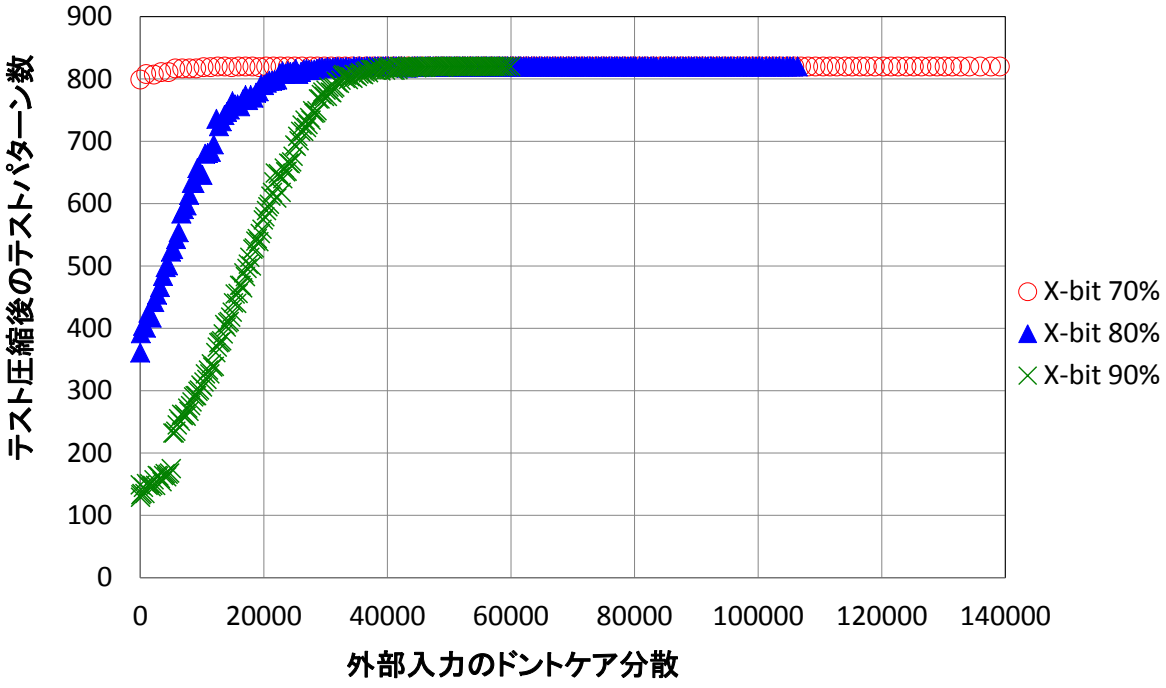


図 5.3.1.1：外部入力のドントケア分散とテスト圧縮の関係(ITC'99 b14)

5.3.2　テスト圧縮指向ドントケア判定

本節では，提案手法であるテスト圧縮指向ドントケア判定[20]を説明する．5.3.1 節の予備実験結果より，ドントケアに基づくテスト圧縮は，外部入力のドントケアの分散とテスト集合中のドントケア数によって圧縮後のテストパターン数に差が生じることが確認できた．これより，テスト圧縮指向ドントケア判定は以下のように問題定式化ができる．

Inputs：初期テスト集合 T
Outputs：ドントケアを含むテスト集合 XT
Constraint：テスト集合中のドントケアの割合 ≧ n (%)
Minimization：$s^2(XT)$, subject to $D(T) = D(XT)$　　(5.3.2.1)

式(5.3.2.1)において，$s^2(XT)$はドントケアを含むテスト集合 XT の外部入力のドントケア分散を示す．また，$D(T)$は初期テスト集合 Tの故障検出率，$D(XT)$はドントケアを含むテスト集合 XT の故障検出率を示す．n はテスト集合に含まれるドントケアの割合$(0 \leqq n \leqq 100)$を示す．

図 5.3.2.1 に，テスト圧縮ドントケア判定の全体アルゴリズムを示す．

```
1.  Procedure test_compaction_oriented_X-Identification(C,T,)
2.  C :  circuit , T : initial_test_set  ;
3.  {
4.        EXT = Φ;
5.        UXT = Φ;
6.        XT = Φ;
7.        D = fault_simulation (C, T) ;
8.        for each test_pattern t_i in T {
9.            EXT += essential_X-filling (C, D, t_i ) ;
10.       }
11.       U = collect_undetected_fault (C, D, EXT ) ;
12.       UXT = distribution_X-filling (C, D, U, T, EXT ) ;
13.       U = collect_undetected_fault (C, D, UXT ) ;
14.       for each test_pattern t_i in T {
15.           XT += missed_X-filling (C, D, t_i , uxt_i ) ;
16.       }
17.       return XT ;
18. }
```

図 5.3.2.1：テスト圧縮指向ドントケア判定アルゴリズム

与えられた初期テスト集合 Tから，ドントケア判定後のテスト集合 XT を得るまでの処理手順を説明する．ここで入力は回路 Cと，初期テスト集合 Tである．まず，ドントケア判定後のテスト集合を保存する変数 EXT, UXT, XTをそれぞれΦに初期化する(行 4-6)．次に，初期テスト集合 Tに対して故障シミュレーションを実行し，故障辞書 D を算出する(行 7)．D を用いて，Tに含まれる各テストパターン t_iに対して必須故障の検出に必要な外部入力値を求め，EXT に格納する．また，EXT は必須故障の検出のみを保証したドントケアを含むテスト集合である．(行 8-10)．EXTは必

須故障の検出のみを保証したテスト集合であるが，偶発的に他の故障も検出する可能性がある．そのため，EXTに対して故障シミュレーションを実行し，未検出故障リスト Uを算出する(行 11)．未検出故障リスト Uに対して，外部入力のドントケア分散とドントケア数を考慮したテスト圧縮指向ドントケア割当てを実行し，テスト集合 UXTを生成する．UXTは必須故障と未検出故障の検出を保証し，外部入力のドントケア分散とドントケア数を考慮したドントケアを含むテスト集合である．詳細なアルゴリズムについては，図 5.3.2.2 で述べる(行 12)．UXTを用いて故障シミュレーションを実行し，未検出故障リスト Uを更新する(行 13)．行 13 で算出した見逃し故障に対して，Tに含まれる各テストパターン t_iに対して未検出故障の検出に必要な外部入力値を求め，XTに格納する．XTは Tと故障検出率が等しく，ドントケアを含むテスト集合である．(行 14-16)．XTを返し，終了する(行 17)．

図 5.3.2.2 に，外部入力のドントケア分散とドントケア数を考慮したテスト圧縮指向ドントケア割当てのアルゴリズムを示す．

```
1.  Procedure distribution_X-filling (C, D, U, T, EXT) ;
2.  C : circuit , D : fault_dictionary ,
3.  U : undetected_fault_set , T : initial_test_set ,
4.  EXT : essential_fault_detectable_test_set ;
5.  {
6.      XT = EXT ;
7.      MCT = ∞ ;
8.      for each undetected_fault f_i in U {
9.          DT_fi = collect_f_i_detectable_test_pattern( f_i, D) ;
10.         for each detectable test pattern t_j for f_i in DT_fi {
11.             xt'_j = find_value(f_i , t_j ) ;
12.             CT = caluculation_cost( XT, xt'_j ) ;
13.             if ( MCT > CT) {
14.                 MCT = CT ;
15.                 t_mct = xt'_j ;
16.                 k = j ;
17.             }
18.         }
19.         XT = merge_test_pattern ( xt_k in XT , t_mct , XT) ;
20.         U = fault_simulation ( C, XT, U) ;
21.     }
22.     return XT ;
23. }
```

図 5.3.2.2：外部入力のドントケアの分散を考慮した
ドントケア判定アルゴリズム

外部入力のドントケア分散とドントケア数を考慮したテスト圧縮指向ドントケア割当ての処理手順を説明する．入力は回路 C と，初期テスト集合 T の故障辞書 D と，未検出故障リスト U と，初期テスト集合 T と，必須故障の故障検出を保証したドントケアを含むテスト集合 EXT である．まず，XT に EXT を代入する(行 6)．次に，外部入力のドントケア分散と

ドントケア数の最小コスト MCT を無限大に初期化する(行 7). 次に，未検出故障リスト U に含まれる未検出故障 f_i に対して，行 9 から行 21 の処理を適用する(行 8). f_i を検出可能な初期テストパターンを故障辞書 D から算出し，DT_{fi} に格納する(行 9). DT_{fi} 含まれる各テストパターン t_j に対して，行 11 から行 18 の処理を適用する(行 10). t_j に対して，f_i の検出に必要な外部入力値を算出し，ドントケアを含むテストパターン xt'_j を生成する. ここで, xt'_j は f_i の検出のみを保証したテストパターンである(行 11). XT と xt'_j を用いて, xt'_j で f_i を検出したときのテスト集合での外部入力のドントケア分散とドントケア数のコスト CT を算出する. コスト CT の計算方法については 5.3.3 節で詳細に述べる (行 12). MCT と CT を比較し，CT のほうが小さい場合は行 14 から行 17 の処理を適用する(行 13). MCT に CT を代入し，MCT を更新する(行 14). 最小のコストをもつテストパターン xt'_j を t_{mct} に代入する(行 15). 最小のコストをもつテストパターンの ID(j)を k に保存する(行 16). 最小のコストを持つテストパターン t_{mct} と，同じテストパターン ID を持つテストパターン xt_k をドントケアに基づくテストを適用し，テスト集合 XT を更新する(行 19). XT を用いて U に対して故障シミュレーションを実行し，U を更新する(行 20). テスト集合 XT を返し，終了する(行 22). ここで，XT は未検出故障リスト U に含まれる全ての故障を検出可能なテスト集合である.

5.3.3 外部入力のドントケア分散とドントケア数のコスト

本節では，外部入力のドントケア分散とドントケア数のコスト[20][22]について説明する．このコストは，図 5.3.2.2 の行 12 で用いられる．また，このコストが小さいほどドントケア数が多く，外部入力のドントケア分散が小さいテスト集合である．そのため，コストが小さいほどテスト圧縮に効果的なテスト集合であるといえる．

式(5.3.3.1)にテストパターン中のある外部入力値がケアビットか否かを判定する関数を示す．式(5.3.3.1)において，xt_iはドントケアを含むテスト集合の i 番目のテストパターン，p_kは xt_iの k 番目の外部入力を示す．式(5.3.3.1)では，xt_iの p_kの値がケアビットの場合 1 を，それ以外の場合 0 を返す．

$$C(xt_i, p_k) = \begin{cases} 1 & \text{if } p_k \text{ value of } xt_i \text{ is a care bit} \\ 0 & \text{otherwise (X-bit)} \end{cases} \tag{5.3.3.1}$$

式(5.3.3.2)にテスト集合中のある外部入力のケアビット数を算出する関数を示す．式(5.3.3.2)において，XT はドントケアを含むテスト集合，p_nは n 番目の外部入力，$N_{TP}(XT)$はテスト集合 XT に含まれるテストパターン数を示す．

$$W(XT, p_n) = \sum_{j=1}^{N_{TP}(XT)} C(xt_j, p_n) \tag{5.3.3.2}$$

式(5.3.3.3)に外部入力のドントケア分散とドントケア数のコストを算出する関数を示す．式(5.3.3.3)において，xt_jはある未検出故障 f_iの検出がまだ保証されていないドントケア判定途中のテストパターン，xt'_j はある未検出故障 f_iのみの検出を保証したテストパターンを示す．

$$VX(xt_j, xt'_j) = \sum_{n=1}^{N_{PI}} W(XT, p_n) \times X(xt_j, p_n) \times C(xt'_j, p_n) \qquad (5.3.3.3)$$

表 5.3.3.1 にテスト集合 *XT* に含まれる各外部入力のケアビット数の計算例を示す．表 5.3.3.1 において，c はケアビットを示し，x はドントケアを示す．表 5.3.3.1 は，xt_1 から xt_5 の 5 個のテストパターン，p_1 から p_7 の 7 本の外部入力を持つテスト集合 *XT* である．各外部入力に対して，式(5.3.3.2)を用いて，各外部入力のケアビット数を算出すると，W(*XT*, p_1)=2, W(*XT*, p_2)=0，W(*XT*, p_3)=4，W(*XT*, p_4)=5，W(*XT*, p_5)=2，W(*XT*, p_6)=2, W(*XT*, p_7)=1 と計算される．

表 5.3.3.1：テスト集合 *XT* に含まれる
各外部入力のケアビット数の計算例

	p_1	p_2	p_3	p_4	p_5	p_6	p_7
xt_1	x	x	c	c	x	c	x
xt_2	c	x	x	c	c	x	c
xt_3	x	x	c	c	x	x	x
xt_4	c	x	c	c	c	x	x
xt_5	x	x	c	c	x	c	x
$W(XT, p_n)$	2	0	4	5	2	2	1

表 5.3.3.2 に外部入力のドントケア分散とドントケア数のコストの計算例を示す．表 5.3.3.2 において，c はケアビットを示し，x はドントケアを示す．また，W(*XT*,p_n)は表 5.3.3.1 のテスト集合のものを示す．表 5.3.3.2 では，未検出故障 f_i を検出可能な初期テストパターンは t_1，t_3，t_5 の 3 個であったとする．xt'_1，xt'_3，xt'_5 の 3 個のテストパターンは f_i のみの検出を保証したテストパターンである．また，xt'_1 の p_1，p_2，p_7，xt'_3 の p_2，

p_7, xt'_5の p_1, p_2は表 5.3.3.1 の時点ではドントケアであったが，f_iを検出するのに新たに増加したケアビットである．表 5.3.3.2 の xt'_3に対して，式(5.3.3.3)を用いて外部入力のドントケア分散とドントケア数のコストを計算すると以下のように計算される．

$$
\begin{aligned}
VX(xt_3, xt'_3) = {} & W(XT, p_1) \times X(xt_3, p_1) \times C(xt'_3, p_1) \\
& + W(XT, p_2) \times X(xt_3, p_2) \times C(xt'_3, p_2) \\
& + W(XT, p_3) \times X(xt_3, p_3) \times C(xt'_3, p_3) \\
& + W(XT, p_4) \times X(xt_3, p_4) \times C(xt'_3, p_4) \\
& + W(XT, p_5) \times X(xt_3, p_5) \times C(xt'_3, p_5) \\
& + W(XT, p_6) \times X(xt_3, p_6) \times C(xt'_3, p_6) \\
& + W(XT, p_7) \times X(xt_3, p_7) \times C(xt'_3, p_7) \\
= {} & (2\times1\times0) + (0\times1\times1) + (4\times0\times1) + (5\times0\times0) \\
& + (2\times1\times0) + (2\times1\times0) + (1\times1\times1) \\
= {} & 0 + 0 + 0 + 0 + 0 + 0 + 1 = 1
\end{aligned}
$$

同様に，xt'_1, xt'_5に対しても計算すると，$VX(xt_1, xt'_1)=3$, $VX(xt_5, xt'_5)=2$と計算される．外部入力のドントケア分散とドントケア数のコストが最も小さいテストパターンが未検出故障f_iを検出するテストパターンと選択される．そのため，表 5.3.3.2 の例では，テストパターン xt'_3が選択される．

表 5.3.3.2：外部入力のドントケア分散と
ドントケア数のコスト計算例

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	$VX(xt_j, xt'_j)$
xt'_1	c	c	x	x	x	x	c	3
xt'_3	x	c	c	x	x	x	c	1
xt'_5	c	c	x	x	x	c	x	2
$W(XT, p_n)$	2	0	4	5	2	2	1	

5.3.4　実験結果

本節では，提案手法であるテスト圧縮指向ドントケア判定と従来のドントケア判定法[17][18]に対してドントケア判定を行い，ドントケア判定率，外部入力のドントケア分散，テスト圧縮後のテストパターン数，ドントケア判定時間に対して比較評価した．実験対象回路は，ISCAS'89 ベンチマーク回路と ITC'99 ベンチマーク回路である．初期テスト集合は Synopsys 社の TetraMAX によって生成された縮退故障用のテスト集合を用いた．また，初期テスト集合はテスト生成時に動的圧縮を適用したテスト集合 T_c と，テスト圧縮を適用してないテスト集合 T_{uc} を用意した．

表 5.3.4.1 に，初期テスト集合 T_{uc} に対するドントケア判定結果を示す．

表 5.3.4.1：初期テスト集合 T_{uc} に対するドントケア判定結果

Circuits	N_{PI}	$N_{TP}(T_{uc})$	XID [17]			DC-XID [18]			PI_Vari [20]			VX(Proposed)			FC(%)
			%X-bit	$s^2(XT_{uc})$	Time(sec)	%X-bit	$s^2(XT_{uc})$	Time(sec)	%X-bit	$s^2(XT_{uc})$	Time(sec)	%X-bit	$s^2(XT_{uc})$	Time(sec)	
s13207	650	589	93.73	2252	**1.13**	95.51	1884	4.46	95.57	**1530**	5.27	**95.77**	1558	5.04	99.04
s15850	600	562	93.74	1040	**1.38**	94.24	1126	11.1	**94.62**	**689**	12.46	**94.62**	749	15.98	97.95
s35932	1763	80	86.03	15	**5.22**	87.70	12	73.4	**88.73**	**7**	85.87	88.51	8	116.57	90.04
s38417	1524	1185	96.94	1524	**9.04**	97.34	1131	66.5	97.50	**745**	81.63	**97.58**	783	97.37	99.73
s38584	1462	862	96.23	1818	**11.3**	96.49	1538	800.67	96.50	**931**	319.82	**96.67**	966	283.36	95.27
b14	277	1317	82.02	78512	**27.64**	80.74	78215	174.08	80.82	**54584**	199.8	**82.43**	56903	361.15	99.47
b15	485	909	90.75	14044	**25.57**	89.77	16639	307.13	**92.38**	**8715**	466.87	91.78	9454	497.1	97.26
b17	1452	3019	96.16	30017	**65.97**	95.96	29116	1699.44	**96.84**	**20184**	3866.55	96.63	20918	3550.99	97.83
b20	522	2138	86.94	60510	**65.9**	86.09	60007	629.02	87.17	**38127**	789.9	**87.64**	41266	514.68	99.48
b21	522	2316	87.94	68423	**54.67**	86.92	67539	679.45	87.78	**44401**	803.35	**88.29**	47875	544.77	99.44
b22	767	2770	91.24	57956	**96.29**	90.46	60311	1286.69	91.30	**36979**	843.53	**91.62**	40734	665.25	99.56

表 5.3.4.1 において，Circuits は回路名，N_{PI} は外部入力数，$N_{TP}(T_{uc})$ は初期テスト集合 T_{uc} に含まれるテストパターン数，%X-bit はドントケア判定後のテスト集合 XT_{uc} に含まれるドントケアの割合，$s^2(XT_{uc})$はドントケア判定後のテスト集合 XT_{uc} の外部入力のドントケア分散値，Time(sec)はドントケア判定の処理時間，FC(%)は故障検出率を示す． FC(%)は初期テスト集合とドントケア判定後のテスト集合で等しい故障検出率である．また，XID は文献[17]のドントケア判定法を適用した結果，DC-XID は文

献[18]のドントケア判定法を適用した結果，PI_Vari は外部入力のドントケア分散のみを考慮したドントケア判定法[20]を適用した結果，VX(Proposed)は提案手法である外部入力のドントケア分散とドントケア数を考慮したドントケア判定法[20]を適用した結果である．

ドントケア判定率(%X-bit)に着目すると，s13207，s15850，s38417，s38584，b14，b20，b21，b22 に対しては，提案手法である VX(Proposed)が最も高い結果となった．s35932，b15，b17 に対しては，PI_Vari が最も高い結果となった．VX(Proposed)と XID，DC-XID を比較すると，ドントケア判定率は VX(Proposed)のほうが約 1%高い結果となった．s13207，s15850，s38417，s38584，b14，b20，b21，b22 のドントケア判定率に関しては VX(Proposed)のほうが PI_Vari より約 0.4%高い結果となった．

外部入力のドントケア分散($s^2(XT_{uc})$)に着目すると，全ての回路に対して PI_Vari が最も低い結果となった．しかしながら，VX(Proposed)と PI_Vari を比較すると，$s^2(XT_{uc})$の差は非常に小さく，VX(Proposed)のほうが平均 6.9%高い結果となった．VX(Proposed)と XID，DC-XID を比較すると，VX(Proposed)のほうが 17%から 48%(平均 33%)低い結果となった．

ドントケア判定時間(Time(sec))に着目すると，全ての回路に対して XID が最も短い結果となった．VX(Proposed)と XID を比較すると，VX(Proposed)のほうが平均約 18 倍遅い結果となった．

表 5.3.4.2 に，初期テスト集合 T_c に対するドントケア判定結果を示す．

表 5.3.4.2：初期テスト集合 T_c に対するドントケア判定結果

Circuits	N_{PI}	$N_{TP}(T_c)$	XID [17]			DC-XID [18]			PI_Vari [20]			VX(Proposed)			FC(%)
			%X-bit	$s^2(XT_c)$	Time(sec)	%X-bit	$s^2(XT_c)$	Time(sec)	%X-bit	$s^2(XT_c)$	Time(sec)	%X-bit	$s^2(XT_c)$	Time(sec)	
s13207	650	267	87.86	1676	**2.66**	91.13	1384	10.66	91.23	**1302**	17.31	**91.66**	1315	17.5	99.04
s15850	600	130	79.02	430	**2.91**	79.74	451	27.14	80.96	**372**	32.17	**81.32**	383	30.9	97.95
s35932	1763	21	56.81	5	**8.29**	58.61	3	91.38	58.54	**2**	117.64	**59.09**	**2**	147.87	90.04
s38417	1524	104	75.54	401	**12.1**	76.78	387	297.48	76.49	**332**	218.72	**77.38**	351	185.31	99.73
s38584	1462	143	**83.65**	451	**12.33**	82.46	482	410.04	82.72	**394**	190.14	83.61	407	183.81	95.27
b14	277	749	**77.14**	21755	**20.46**	74.97	23272	188.99	75.46	**20450**	201.05	75.73	20961	220.77	99.47
b15	485	459	85.41	5086	**18.82**	84.29	5916	308.24	**87.31**	**4160**	276.53	86.22	4641	307.84	97.26
b17	1452	1056	91.33	13530	**145.81**	90.73	14229	3509.52	**92.18**	**11960**	3627.86	91.88	12302	2461.18	97.83
b20	522	754	**72.90**	18401	**41.48**	70.68	18921	737.28	71.45	**15886**	631.87	71.49	17296	624.06	99.48
b21	522	762	**72.92**	20468	**41.21**	70.90	20537	745.84	71.76	**17666**	628.82	71.80	19194	629.65	99.44
b22	767	640	**72.14**	18375	**61.92**	70.65	18694	1426.8	70.70	**16023**	1088.42	71.25	17415	1005.29	99.56

表 5.3.4.2 において，Circuits は回路名，N_{PI} は外部入力数，$N_{TP}(T_c)$は初期テスト集合 T_c に含まれるテストパターン数，%X-bit はドントケア判定後のテスト集合 XT_c に含まれるドントケアの割合，$s^2(XT_c)$はドントケア判定後のテスト集合 XT_c の外部入力のドントケア分散値，Time(sec)はドントケア判定の処理時間，FC(%)は故障検出率を示す． FC(%)は初期テスト集合とドントケア判定後のテスト集合で等しい故障検出率である．また，XID は文献[17]のドントケア判定法を適用した結果，DC-XID は文献[18]のドントケア判定法を適用した結果，PI_Vari は外部入力のドントケア分散のみを考慮したドントケア判定法[20]を適用した結果，VX(Proposed)は提案手法である外部入力のドントケア分散とドントケア数を考慮したドントケア判定法[20]を適用した結果である．

ドントケア判定率(%X-bit)に着目すると，s13207，s15850，s35932，s38417，に対しては，提案手法である VX(Proposed)が最も高い結果となった． VX(Proposed)と DC-XID を比較すると，全ての回路に対して VX(Proposed)のほうが約 1%高い結果となった.また，VX(Proposed)と PI_Vari を比較すると，b15 と b17 以外の回路に対しては VX(Proposed)

のほうが約 0.5%高い結果となった. VX(Proposed)と XID を比較すると，s13207，s15850，s35932，s38417，b15，b17 以外の回路に対しては VX(Proposed)のほうが約 2%高い結果となった. s13207, s15850, s35932, s38417，b15，b17 に対しては，VX(Proposed)のほうが XID より約 1%低い結果となった.

外部入力のドントケア分散($s^2(XT_c)$)に着目すると，全ての回路に対して PI_Vari が最も低い結果となった. しかしながら, VX(Proposed)と PI_Vari を比較すると，$s^2(XT_c)$の差は非常に小さく，VX(Proposed)のほうが平均 5.1%高い結果となった．VX(Proposed)と XID，DC-XID を比較すると，VX(Proposed)のほうが 3%から 60%(平均 13%)低い結果となった.

ドントケア判定時間(Time(sec))に着目すると, 全ての回路に対して XID が最も短い結果となった．VX(Proposed)と XID を比較すると，VX(Proposed)のほうが平均約 16 倍遅い結果となった.

表 5.3.4.3 に，初期テスト集合 T_{uc} にドントケア判定を適用した後のテスト集合 XT_{uc} に対するテスト圧縮結果を示す.

表 5.3.4.3：ドントケア判定後のテスト集合 XT_{uc} に対するテスト圧縮結果

Circuits	$N_{TP}(T_{uc})$	XID [17]		DC-XID [18]		PI_Vari [20]		VX(Proposed)		FC(%)
		#Dsatur	#DD	#Dsatur	#DD	#Dsatur	#DD	#Dsatur	#DD	
s13207	589	284	276	285	275	**271**	270	**271**	**269**	99.04
s15850	562	178	168	170	159	**150**	149	**150**	**147**	97.95
s35932	80	**37**	**36**	79	64	40	39	40	39	90.04
s38417	1185	187	185	157	155	**138**	**138**	140	140	99.73
s38584	862	210	209	193	188	174	172	**171**	**170**	95.27
b14	1317	1013	916	1009	896	918	870	**911**	**866**	99.47
b15	909	627	521	654	535	568	514	**552**	**504**	97.26
b17	3019	1315	1260	1294	1246	1228	1205	**1192**	**1174**	97.83
b20	2138	1283	1164	1300	1176	1133	**1080**	**1122**	1090	99.48
b21	2316	1390	1259	1398	1283	**1232**	**1194**	1250	1208	99.44
b22	2770	1152	1082	1147	1092	983	970	**964**	**957**	99.56

表 5.3.4.3 において，Circuits は回路名，$N_{TP}(T_{uc})$は初期テスト集合 T_{uc} に含まれるテストパターン数，#Dsatur はドントケア判定後のテスト集合 XT_{uc} に文献[16]のドントケアに基づくテスト圧縮を適用した後のテストパターン数,#DD は#Dsatur のテスト集合に対して二重検出法を適用した後のテストパターン数を示す．FC(%)はドントケア判定後のテスト集合の故障検出率である．また，XID は文献[17]のドントケア判定法を適用した結果，DC-XID は文献[18]のドントケア判定法を適用した結果，PI_Vari は外部入力のドントケア分散のみを考慮したドントケア判定法[20]を適用した結果,VX(Proposed)は提案手法である外部入力のドントケア分散とドントケア数を考慮したドントケア判定法[20]を適用した結果である.

#Dsatur に着目すると，s13207，s15850，s38584，b14，b15，b17，

b20，b22 の回路に対しては，VX(Proposed)が最もテストパターン数が少ない結果となった．VX(Proposed)と XID，DC-XID，PI_Vari を比較すると，VX(Proposed)は s13207，s15850，s38584，b14，b15，b17，b20，b22 の回路に対して最大 188 パターン(平均 71 パターン)のテストパターン数が削減できた. また, s35932 に対しては XID より 3 パターン, s38417 に対しては PI_Vari より 2 パターン，b21 に対しては PI_Vari より 18 パターン VX(Proposed)のほうが多い結果となった.

#DD に着目すると，s13207，s15850，s38584，b14，b15，b17，b22 の回路に対しては, VX(Proposed)が最もテストパターン数が少ない結果となった．VX(Proposed)と XID，DC-XID，PI_Vari を比較すると，VX(Proposed)は s13207，s15850，s38584，b14，b15，b17，b22 の回路に対して最大 135 パターン(平均 39 パターン)のテストパターン数が削減できた．また，s35932 に対しては XID より 3 パターン，s38417 に対しては PI_Vari より 2 パターン, b20 に対しては PI_Vari より 10 パターン，b21 に対しては PI_Vari より 14 パターン VX(Proposed)のほうが多い結果となった.

表 5.3.4.4 に，初期テスト集合 T_c にドントケア判定を適用した後のテスト集合 XT_c に対するテスト圧縮結果を示す．

表 5.3.4.4：ドントケア判定後のテスト集合 XT_c に対するテスト圧縮結果

Circuits	$N_{TP}(T_c)$	XID [17]		DC-XID [18]		PI_Vari [20]		VX(Proposed)		FC(%)
		#Dsatur	#DD	#Dsatur	#DD	#Dsatur	#DD	#Dsatur	#DD	
s13207	267	262	262	261	260	260	260	**259**	**259**	99.04
s15850	130	127	126	127	127	125	123	**124**	**121**	97.95
s35932	21	**21**	**21**	**21**	**21**	**21**	**21**	**21**	**21**	90.04
s38417	104	104	104	104	104	**103**	**103**	**103**	**103**	99.73
s38584	143	140	139	143	143	**135**	**134**	**135**	135	95.27
b14	749	728	710	737	708	716	705	**715**	**703**	99.47
b15	459	393	357	398	367	378	356	**372**	**352**	97.26
b17	1056	1033	1022	1028	1022	1016	1014	**1015**	**1012**	97.83
b20	754	733	729	728	727	**721**	719	**721**	**717**	99.48
b21	762	739	737	737	737	**725**	**722**	726	725	99.44
b22	640	**621**	**621**	638	638	624	624	625	624	99.56

表 5.3.4.4 において，Circuits は回路名， $N_{TP}(T_c)$は初期テスト集合 T_c に含まれるテストパターン数，#Dsatur はドントケア判定後のテスト集合 XT_c に文献[16]のドントケアに基づくテスト圧縮を適用した後のテストパターン数，#DD はドントケア判定後のテスト集合 XT_{uc} に対して二重検出法を適用した後のテストパターン数を示す． FC(%)はドントケア判定後のテスト集合の故障検出率である．また，XID は文献[17]のドントケア判定法を適用した結果，DC-XID は文献[18]のドントケア判定法を適用した結果，PI_Vari は外部入力のドントケア分散のみを考慮したドントケア判定法[20]を適用した結果，VX(Proposed)は提案手法である外部入力のドントケア分散とドントケア数を考慮したドントケア判定法[20]を適用した結果である．

#Dsatur に着目すると，s13207，s15850，s35932，s38417，s38584，b14，b15，b17，b20 の回路に対しては，VX(Proposed)が最もテストパターン数が少ない結果となった．VX(Proposed)と XID，DC-XID，PI_Vari を比較すると，VX(Proposed)は s13207，s15850，s35932，s38417，s38584，b14，b15，b17，b20 の回路に対して最大 26 パターン(平均 8 パターン)のテストパターン数が削減できた．また，b21 に対しては PI_Vari より 1 パターン，b22 に対しては XID より 4 パターン VX(Proposed)のほうが多い結果となった．

#DD に着目すると，s13207，s15850，s35932，s38417，b14，b15，b17，b20 の回路に対しては，VX(Proposed)が最もテストパターン数が少ない結果となった．VX(Proposed)と XID，DC-XID，PI_Vari を比較すると，VX(Proposed)は s13207，s15850，s35932，s38417，b14，b15，b17，b20 の回路に対して最大 15 パターン(平均 6 パターン)のテストパターン数が削減できた．また，s38584 に対しては PI_Vari より 1 パターン，b21 に対しては PI_Vari より 3 パターン，b22 に対しては XID より 3 パターン VX(Proposed)のほうが多い結果となった．

5.4 結言

本章では，ドントケア判定技術の説明，および予備実験として外部入力のドントケア分散とテスト圧縮の関係について解析した．予備実験結果より，ドントケアに基づくテスト圧縮は外部入力のドントケア分散とテスト集合中のドントケア数に依存することが確認できた．

また，予備実験結果よりテスト圧縮指向ドントケア判定法の提案と，提案手法と従来のドントケア判定法[17][18]との比較実験を行った．

テスト圧縮を未適用な初期テスト集合に対する実験結果では，従来のドントケア判定法[17][18]と比較して，ドントケア判定率は平均約 1%の増加，外部入力のドントケア分散は平均約 30%の削減を達成した．また，テスト圧縮後のテストパターン数に関しては，従来のドントケア判定法[17][18]と比較して，平均約 12%の削減を達成した．

テスト圧縮を適用済みの初期テスト集合に対する実験結果では，従来のドントケア判定法[17][18]と比較して，ドントケア判定率は平均約 1%の増加，外部入力のドントケア分散は平均約 10%の削減を達成した．また，テスト圧縮後のテストパターン数に関しては，従来のドントケア判定法[17][18]と比較して，平均約 3%の削減を達成した．

第 6 章

キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成

6.1　緒言

フルスキャン設計を施した回路では，一般的なテスト生成は機能動作を考慮せず，テスト生成の容易性を優先してテスト生成を行う．このため，生成されたテストパターンは，スキャン FF の状態が機能動作では起こりえない状態(無効状態)となる可能性がある．無効状態では，回路内の多くの信号線に遷移を発生させ消費電力を増加させている可能性がある．過去に提案されたテスト時消費電力の削減を考慮した手法として，FF の遷移数に制約のあるテスト生成法[42]や，FF の遷移数を抑えるドントケア割当て法[43-47]が提案されている．

一方，文献[25]では，一般的なテスト生成で生成した遷移故障モデルのテストパターンを印可した後に，キャプチャ動作を 20 サイクル行うことで WSA が減少することが報告されている．また，文献[26]では k(≧2)時間展開モデルを利用して遷移故障を検出するテスト生成法としてマルチサイクルキャプチャ・テスト生成が提案されている．本章では，文献[25]と文献[26]に着目し，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法[23][24]を提案する．

6.2 節で予備実験として，マルチサイクルキャプチャ動作と WSA の関

係について解析する．6.3 節で提案手法である，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法について説明する．6.4 節で結言を述べる．

6.2　マルチサイクルキャプチャ動作と WSA

ブロードサイドテストは，シフト動作によりテストパターンを印加した後，キャプチャ動作を 2 サイクル間実行する．しかしながら，文献[25]において，一般的なテスト生成で生成した遷移故障モデルのテストパターンを印可した後に，キャプチャ動作を 5 サイクル以上行うことで WSA が減少することが報告されている．本節では，文献[25]で報告されている内容を確認するために，予備実験として遷移故障モデルのテストパターンに対してキャプチャ動作を 50 サイクル行った際の WSA について解析した．また，本予備実験において 3 サイクル以降のキャプチャ動作に関しては，印加したテストパターンの故障検出率は保証されない．

図 6.2.1 に予備実験で用いたタイミングチャートを示す．図 6.2.1 において，S_L はテストパターン i における最後のスキャンイン動作を示す．C_1 は 1 サイクル目のキャプチャ動作を示す．また，C_2 から C_{50} は 2 サイクル目のキャプチャ動作から 50 サイクル目のキャプチャ動作を示す．S_1 はテストパターン i に対する C_{50} のキャプチャ動作の応答をスキャンアウトしつつ，テストパターン i+1 のスキャンイン動作を示す．外部入力の値は C_1 ではテストパターンを印加し，C_2 以降はランダムパターンを印加する．また，WSA は C_2 から C_{50} の 49 サイクル間で計測する．

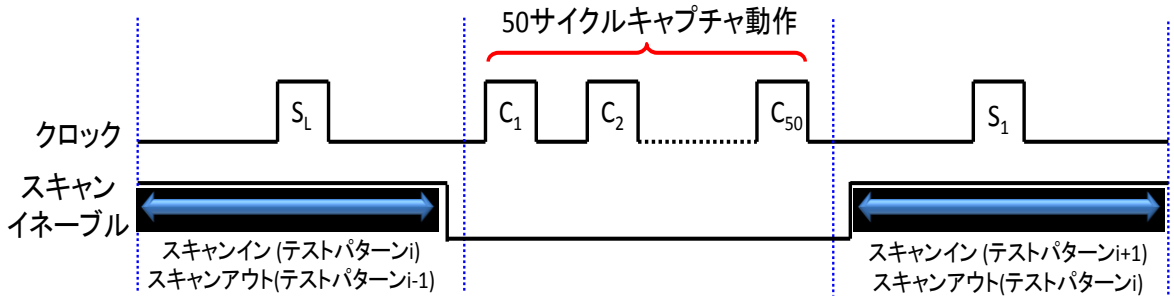


図 6.2.1：予備実験のタイミングチャート

図 6.2.2 に s15850 に対する予備実験結果を示す．テスト集合は Synopsys 社の TetraMAX ATPG で生成した．故障モデルは遷移故障モデルである．また，テストパターン数は 20 パターンである．図 6.2.2 において，縦軸は WSA 値，横軸はキャプチャサイクル数を示す．また，横軸は図 6.2.1 の C_2 から C_{50} に対応する．図 6.2.2 より，ほとんどのテストパターンにおいて 2 回目のキャプチャ動作(C_2)時の WSA が最も高い結果となった．また，5 サイクル目まではキャプチャ動作を繰り返すことにより WSA が大きく減少している．しかしながら，10 サイクル目以降はキャプチャ動作数を増やしても WSA に大きな変化はないことがわかる．また，キャプチャ動作を繰り返すことで，前時刻より WSA が増加するテストパターンが存在することが確認できた．他のベンチマーク回路においても同様の結果が得られた．

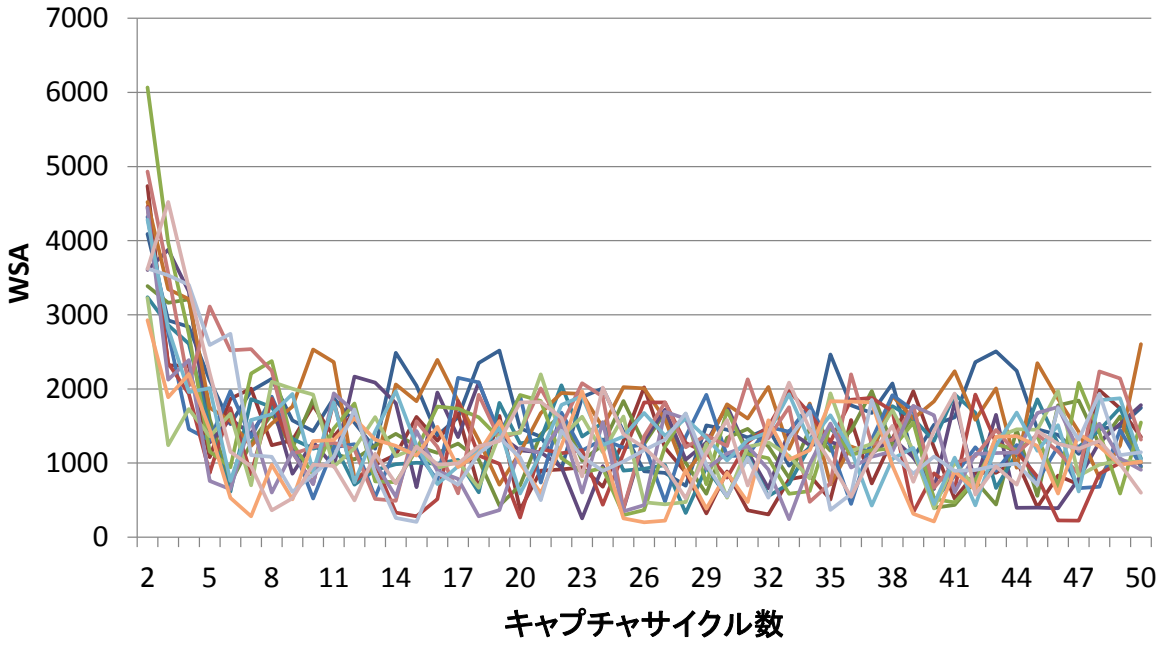


図 6.2.2：s15850 の 50 サイクル動作時の WSA

6.3　キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成

6.2 節より，テスト生成で生成したテストパターンに対して，10 サイクル以上キャプチャ動作を行うと WSA が減少することが確認できた．しかしながら，6.2 節の予備実験では 3 サイクル目以降のキャプチャ動作においては故障検出率が保証されていない．本節では，k 時間展開モデルを応用したキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法[23][24]を提案する．提案手法を用いることで，故障検出率が保証された低消費電力なテストパターンが生成可能である．

6.3.1 節でキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成のテスト生成モデル，6.3.2 節でキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成のアルゴリズム，6.3.3 節で実験結果を説明する．

6.3.1　キャプチャ時消費電力を考慮したマルチサイクルキャプチャ･テスト生成のテスト生成モデル

6.2 節より，テスト生成で生成したテストパターンに対して，10 サイクル以上キャプチャ動作を行うと WSA が減少することが確認できた．提案手法ではこの結果に着目し，テスト生成時に k サイクル動作を行いテストパターンを生成するキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成法を提案する．また提案手法はスキャン設計をした回路を対象とし，テストはブロードサイド方式を想定している．

2 章で説明したように，一般的な遷移故障モデルを対象としたテスト生成は 2 時間展開モデル(ブロードサイドモデルまたはスキュードロードモデル)を用いてテスト生成を行う．提案手法では，k 時間展開モデルを用いてテスト生成を行い，ブロードサイドテストのテストパターンを生成する．提案手法における k 時間展開モデルは，1 時刻目の FF の出力(D 端子)を擬似外部入力とし，k 時刻目の FF のデータ入力(Q 端子)を擬似外部出力として，k 時間分順序回路を時間展開した回路モデルである．また，時刻 k-1 と k にのみ故障を想定し，時刻 1～k-2 には故障を想定しない．これは，時刻 1～k-2 は状態正当化のみに利用し，回路内を低消費電力な状態に近づけることを目的としている．そのため，時刻 1～k-2 においては外部入力の値は常に同じである必要はなく，異なる外部入力値を制御可能なモデルとする．その後，時刻 k-1 と k においてブロードサイドモデルでテスト生成を行うことで，低消費電力なブロードサイドテストのテストパターンが生成可能と考えられる．そのため，時刻 k-1 と k においてはブロードサイドモデルと同様に同一の外部入力値が印加される回路モデルを使用する．このようなテスト生成モデルでテスト生成を行い，時刻 k-1 の FF の状態と外部入力の値をブロードサイドテストのテストパターンとして保存する．

図 6.3.1.1 に信号線 f の立上り遷移故障に対する，5 時間展開モデル(k=5)の場合のキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成のテスト生成モデルの例を示す．図 6.3.1.1 において，1 時刻目

の FF である $FF1_1$，$FF2_1$，$FF3_1$ の 3 個の FF は可制御である．また最終時刻+1 の FF である $FF1_6$，$FF2_6$ ，$FF3_6$ の 3 つの FF は可観測である．また，PI_1～PI_4 の各外部入力も可制御であり，4 時刻目と 5 時刻目の外部入力である PI_4 のみ同一の値が入力される．また 4 時刻目の組合せ回路部には，信号線 f に立上り遷移故障の初期値である 0 を割当てる．5 時刻目の組合せ回路部には，信号線 f に立上り遷移故障の故障値である 1/0 を割当てる．1 時刻目から 3 時刻目の組合せ回路部に関しては，故障は設定しない．テスト生成後，PI4 と $FF1_4$，$FF2_4$，$FF3_4$ に割当てられた値をブロードサイドテストのテストパターンとして保存する．このようなテスト生成モデルを用いることで，キャプチャ動作を複数サイクル間実行した後の FF の状態をテストパターンとするため，キャプチャ時消費電力が抑制されたテストパターンが得られることが期待できる．

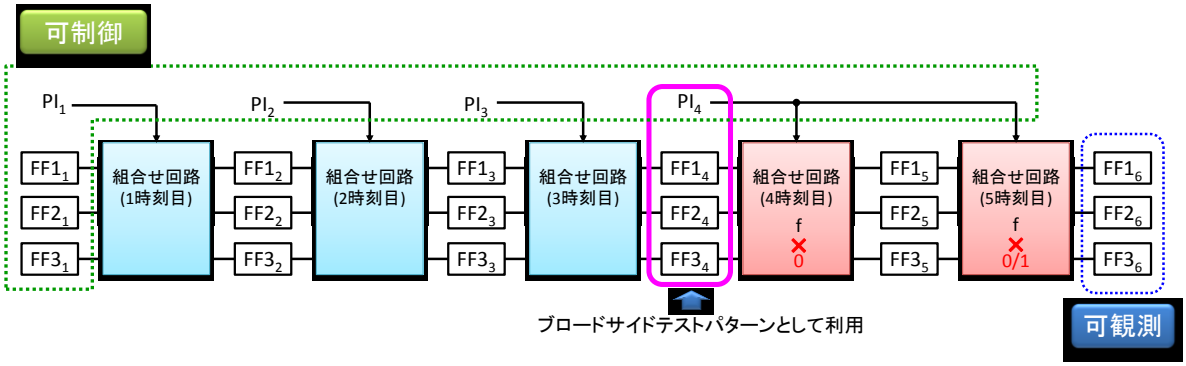


図 6.3.1.1：キャプチャ時消費電力を考慮した
マルチサイクルキャプチャ・テスト生成モデル例

6.3.2　キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成アルゴリズム

図 6.3.2.1 に，マルチサイクルキャプチャ・テスト生成のアルゴリズムを示す.

```
1.  Procedure  low_power_multi_cycle_test_generation(C, k , th);
2.  C : circuit ,     k : time_expansion ,     th : wsa_threshold
3.  {
4.          HPT = Φ;
5.          LPT = Φ;
6.          F = collect_all_fault( C );
7.          T = broadside_test_generation( C, F );
8.          D = broadside_fault_simulation(C, T );
9.          XT = x_identification( C, D, T );
10.         FT = low_power_x_filling( C, XT );
11.         for each test pattern ft_i in FT {
12.                 wsa_i = calc_wsa( C, ft_i );
13.                 if( wsa_i > th){
14.                         HPT = HPT ∪ ft_i ;
15.                 }
16.                 else{
17.                         LPT = LPT ∪ ft_i ;
18.                 }
19.         }
20.         HPF = broadside_fault_simulation( C, HPT , F );
21.         LPF = broadside_fault_simulation( C, LPT , F );
22.         HPF = HPF - LPF ;
23.         for each fault hpf_i in HPF {
24.                 mt_i = multi_cycle_capture_test_generation( C, k, hpf_i );
25.                 LPT = LPT ∪ mt_i ;
26.                 DM = broadside_fault_simulation( C, mt_i , HPF );
27.                 HPF = HPF - DM;
28.         }
29.         return (LPT );
30.}
```

図 6.3.2.1：キャプチャ時消費電力を考慮した
マルチサイクルキャプチャ・テスト生成のアルゴリズム

回路 C と時間展開数 k，高消費電力テストパターンを判定するための WSA 閾値 th を入力とする．まず高消費電力テスト集合 HPT と低消費電力テスト集合 LPT を Φ に初期化する(行 4，行 5)．次にテスト生成の対象となる全ての遷移故障を算出し故障集合 F に代入する(行 6)．行 6 で算出した F と C を基に，ブロードサイドモデルでテスト生成を行い，初期テスト集合 T を算出する(7 行)．T に対してブロードサイドモデルで故障シミュレーションを実行し，検出故障集合 D を算出する(行 8)．T に対してドントケア判定を実行し，ドントケアを含むテスト集合 XT を生成する(行 9)．XT に対してキャプチャ時消費電力を低減するドントケア割当てを実行し，テスト集合 FT を生成する(行 10)．FT に含まれる各テストパターン ft_i に対して行 12 から行 19 の処理を適用する．ft_i に対して WSA を算出し wsa_i を求める(行 12)．wsa_i が th より大きいか否かを判定する(行 13)．もし wsa_i が th より大きければ ft_i を HPT に加え，HPT を更新する(行 14)．それ以外の場合は，ft_i を LPT に加え，LPT を更新する（行 17)．HPT に対してブロードサイドモデルで故障シミュレーションを実行し，検出故障 HPF を算出する(行 20)．LPT に対してブロードサイドモデルで故障シミュレーションを実行し，検出故障 LPF を算出する(行 21)．HPF から LPF の差集合をとり，HPT でしか検出できないアンセーフ故障集合 HPF を更新する(行 22)．HPF に含まれる各故障 hpf_i に対して行 24 から行 28 の処理を適用する(行 23)．hpf_i に対して時間展開数 k でマルチサイクルキャプチャ・テスト生成モデルを用いた低消費電力指向テスト生成を行い，テストパターン mt_i を生成する(行 24)．mt_i を LPT に加え，LPT を更新する(行 25)．mt_i に対してブロードサイドモデルで故障シミュレーションを実行し，検出故障集合 DM を算出する(行 26)．HPF から DM の差集合をとり，HPF を更新する(行 27)．テスト集合 LPT を返す(行 29)．

6.3.3　実験結果

本節では，提案手法であるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成を用いてテスト生成を行い，生成したテスト集合を評価した．実験対象回路は，ISCAS'89 ベンチマーク回路と ITC'99 ベンチマーク回路である．故障モデルは遷移故障モデルである．また，初期テスト集合 T は充足可能性問題を用いたテスト生成で生成したブロードサイドモデルのテスト集合である．また，Tはテスト生成後に二重検出法[13]を適用した．テスト集合 XT の生成には，文献[19]のドントケア判定を適用した．テスト集合 FT の生成には，文献[43]のドントケア割当てを適用した．マルチサイクルキャプチャ・テスト生成モデルを用いた低消費電力指向テスト生成における時間展開数 k は 2，5，10，15，20 を用いて実験を行った．高消費電力テストパターンを判定するための WSA の閾値 th は，FT に含まれるテストパターンの最高 WSA の 50%，60%，70%，80%を設定した．

図 6.3.3.1 に実験フローを示す.

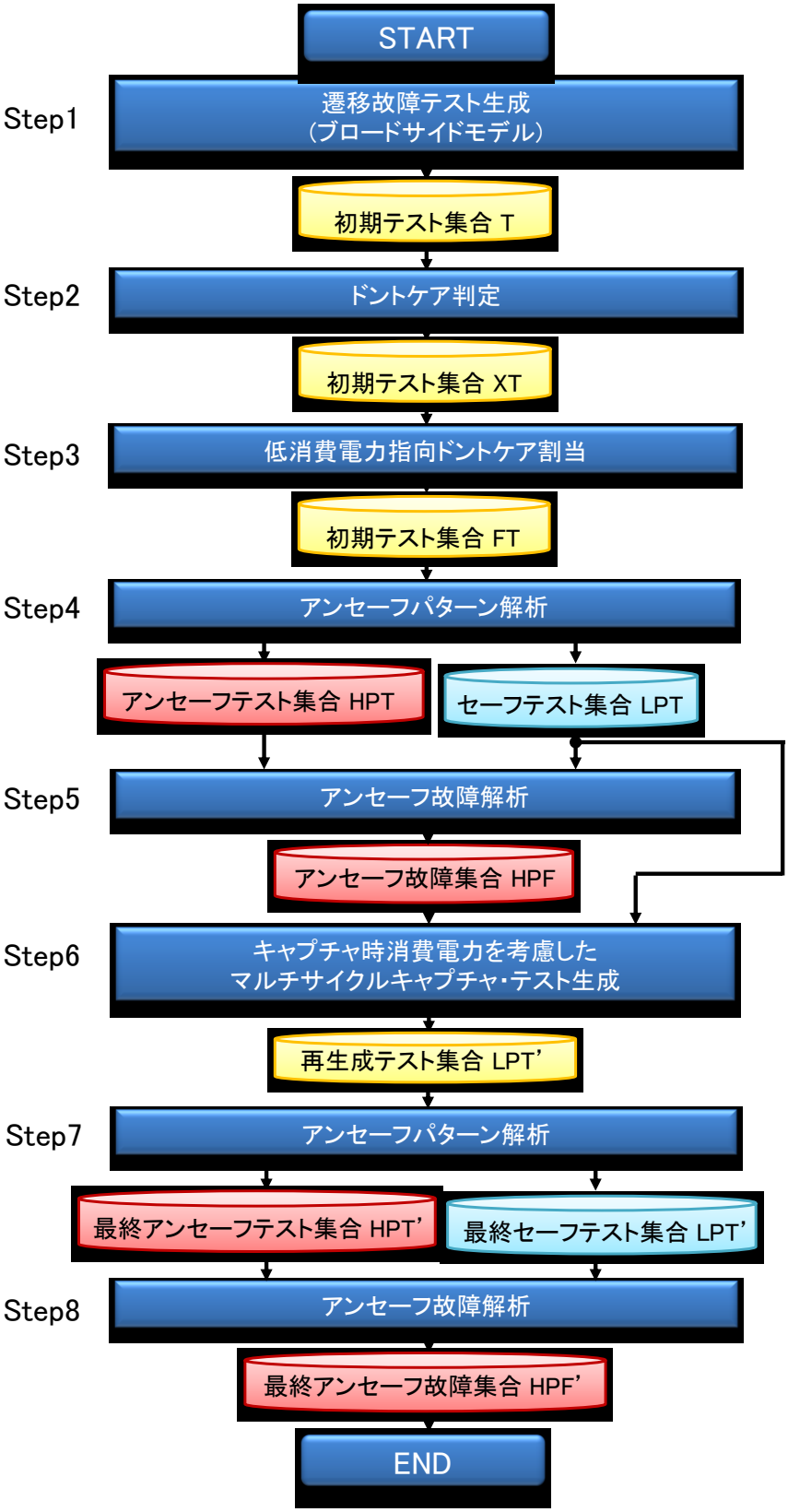


図 6.3.3.1 : 実験フロー

Step 1：

遷移故障を対象にブロードサイドモデルでテスト生成を行い，初期テスト集合 T を生成する．また初期テスト集合 T は，テスト生成後に二重検出法[13]を適用したテスト集合である．

Step 2：

Step1 で生成した初期テスト集合 T に対してドントケア判定を適用し，テスト集合 XT を生成する．

Step 3：

Step2 で生成した初期テスト集合 XT に対してキャプチャ時消費電力を考慮したドントケア割当てを適用し，テスト集合 FT を生成する．

Step 4：

Step3 で生成した初期テスト集合 FT に対して高消費電力テストパターンと低消費電力テストパターンを分類し，高消費電力テストパターン集合であるアンセーフテスト集合 HPT と，低消費電力テストパターン集合であるセーフテスト集合 LPT を生成する．高消費電力テストパターンを判定するための閾値は，FT に含まれるテストパターンの最高 WSA の 50%，60%，70%，80%を使用する．

Step 5：

Step4 で生成したアンセーフテスト集合 HPT とセーフテスト集合 LPT に対して，アンセーフテスト集合 HPT のみで検出可能な故障集合であるアンセーフ故障集合 HPF を生成する．

Step 6：

Step4 で生成したセーフテスト集合 LPT と，Step5 で生成したアンセーフ故障集合 HPF を用いて，提案手法であるキャプチャ時消費電力を考慮したマルチサイクル・キャプチャテスト生成を適用し，再生成テスト集合 LPT’を生成する．

Step 7：

Step6 で生成した再生成テスト集合 LPT'に対して高消費電力テストパターンと低消費電力テストパターンを分類し，高消費電力テストパターン集合であるアンセーフテスト集合 HPT'と，低消費電力テストパターン集合であるセーフテスト集合 LPT'を生成する．高消費電力テストパターンを判定するための閾値は，Step4 で使用した閾値を同じ値を使用する．

Step 8：

Step7 で生成したアンセーフテスト集合 HPT'とセーフテスト集合 LPT'に対して，アンセーフテスト集合 HPT'のみで検出可能な故障集合である最終アンセーフ故障集合 HPF'を生成する．

表 6.3.3.1 に, テスト集合 FT の実験結果を示す. 表 6.3.3.1 は, 図 6.3.3.1 の Step5 までの結果である.

表 6.3.3.1：テスト集合 FT の結果

回路名	WSA閾値	対象故障数	検出故障数	テスト不可能故障数	テストパターン数	アンセーフパターン閾値	アンセーフパターン数	アンセーフ故障数
s5378	50%	10590	6546	4044	223	794	43	543
	60%					952	22	285
	70%					1111	12	156
	80%					1270	6	71
s9234	50%	18468	13813	4655	581	1256	254	2781
	60%					1507	138	1224
	70%					1759	71	519
	80%					2010	32	210
s13207	50%	26358	19261	7097	627	1371	356	5113
	60%					1645	165	1746
	70%					1919	52	372
	80%					2193	12	85
s15850	50%	31694	20009	11685	498	1609	87	1120
	60%					1930	25	289
	70%					2252	8	120
	80%					2574	3	51
s35932	50%	71224	49278	21946	113	6243	64	21281
	60%					7491	30	8453
	70%					8740	22	4922
	80%					9988	13	2326
s38417	50%	76678	73736	2942	1307	5873	104	5047
	60%					7047	75	4258
	70%					8222	46	2688
	80%					9396	22	1448
s38584	50%	76864	50138	26726	1820	3214	181	4696
	60%					3856	47	2944
	70%					4499	13	1448
	80%					5142	7	1095
b14	50%	43250	40115	3135	1601	5126	640	8290
	60%					6151	445	4594
	70%					7177	258	2150
	80%					8202	112	775
b15	50%	40232	34298	5934	1674	2433	552	5626
	60%					2920	334	2654
	70%					3406	148	1107
	80%					3893	52	365

表 6.3.3.1 において, WSA 閾値はアンセーフパターンを判定するための閾値であり, テスト集合中の最大 WSA の 50%, 60%, 70%, 80%の 4 種類に対して実験行った. 対象故障数は, テスト生成の対象とした遷移故障数を示す. 検出故障数は, 対象故障のうちテスト生成によってブロードサイドモデルで検出可能と判定された遷移故障数である. 検出故障数は, 対象故障のうちテスト生成によってブロードサイドモデルで検出可能と判

定された遷移故障数である．テスト不可能故障数は，対象故障のうちテスト生成によってブロードサイドモデルではテスト不可能と判定された遷移故障数である．また，全ての回路において打ち切り故障数は 0 である．テストパターン数は,テスト集合 *FT* に含まれるテストパターン数を示す．アンセーフパターン閾値は，高消費電力テストパターンを判定するための閾値 *th* である．アンセーフパターン閾値は，テスト集合 *FT* に含まれるテストパターンの最大 WSA の 50%，60%，70%，80%から計算した．アンセーフパターン数は,テスト集合 *FT* において閾値 WSA(*th*)を超える WSA を持つテストパターン数を示す．これは，図 6.3.3.1 の Step4 で生成されるアンセーフテスト集合 *HPT* に含まれるテストパターン数に対応する．アンセーフ故障数は，アンセーフパターンでのみ検出可能な故障数を示す．これは，図 6.3.3.1 の Step5 で生成されるアンセーフ故障集合 *HPF* に含まれる故障数に対応する．アンセーフ故障数は，アンセーフパターンでのみ検出可能な故障数を示す．

表 6.3.3.2 に，WSA 閾値 50%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果を示す．表 6.3.3.2 は，図 6.3.3.1 における Step6 から Step8 までの結果である．

表 6.3.3.2：WSA 閾値 50%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果

回路名	時間展開数	対象故障数	検出故障数	テスト不可能故障数	テストパターン数	最終アンセーフパターン数	最終アンセーフ故障数	最終アンセーフ故障数削減率(%)
s5378	2	543	543	0	71	71	543	0
	5		399	144	67	65	368	32
	10		370	173	60	56	304	44
	15		353	190	63	61	327	40
	20		353	190	62	58	284	48
s9234	2	2781	2781	0	335	335	2781	0
	5		1403	1378	204	178	1146	59
	10		1392	1389	202	132	736	74
	15		1391	1390	210	132	630	77
	20		1391	1390	213	124	618	78
s13207	2	5113	5113	0	419	419	5113	0
	5		4535	578	393	384	3842	25
	10		3650	1463	372	280	2012	61
	15		3645	1468	384	274	1326	74
	20		3642	1471	383	277	1759	66
s15850	2	1120	1120	0	157	156	1113	1
	5		999	121	179	149	682	39
	10		933	187	163	101	521	53
	15		932	188	169	92	449	60
	20		927	193	164	75	346	69
s35932	2	21281	21281	0	97	97	21281	0
	5		21281	0	81	80	20612	3
	10		21281	0	67	64	17556	18
	15		21281	0	75	68	17156	19
	20		21281	0	75	72	19704	7
s38417	2	5047	5047	0	266	266	5047	0
	5		4980	67	335	325	3164	37
	10		4977	70	396	374	2773	45
	15		4976	71	448	402	2249	55
	20		4976	71	444	406	2475	51
s38584	2	4696	4696	0	413	413	4696	0
	5		4366	330	449	446	4083	13
	10		4357	339	464	456	3801	19
	15		4354	342	464	452	3620	23
	20		4354	342	441	424	3340	29
b14	2	8290	8290	0	955	874	7170	14
	5		8290	0	1046	992	6826	18
	10		8290	0	1030	869	6369	23
	15		8290	0	1062	908	6465	22
	20		8290	0	1055	907	6505	22
b15	2	5626	5626	0	779	690	4880	13
	5		5618	8	807	654	4177	26
	10		5618	8	836	656	4143	26
	15		5618	8	835	650	4009	29
	20		5618	8	815	632	3763	33

表 6.3.3.2 において，時間展開数はキャプチャ時消費電力を考慮したマルチサイクルキャプチャ･テスト生成における時間展開数 kを示す.また，k=2 はブロードサイドモデルでのテスト生成と同じである．対象故障数はキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の対象とした遷移故障数を示す．また，対象故障数は表 6.3.3.1 のアンセーフ故障数と同数であり，同じ故障である．検出故障数は，対象故障のうち提案手法の k 時間展開モデルでテスト生成を行い，検出可能と判定された故障数である．テスト不可能故障数は，対象故障のうち提案手法の k 時間展開モデルでテスト生成を行い，テスト不可能と判定された故障数である．これは，ブロードサイドモデルでは検出可能であるが，k 時間展開モデル(k>2)を用いることでテスト不可能と判定された故障を示す．そのため，k=2 においては，テスト不可能故障数は 0 となる．また，全ての回路において打ち切り故障数は 0 である．テストパターン数は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成で生成したテストパターン数を示す．これは，図 6.3.2.1 における行 24 のテストパターン mt_iの数に対応する．最終アンセーフパターン数は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成で生成したテスト集合のうちアンセーフパターン閾値を超えたテストパターン数を示す．アンセーフパターン閾値は，表 6.3.3.1 の WSA 閾値 50%と同じ値を使用した．最終アンセーフ故障数は，最終アンセーフパターンでのみ検出可能な故障数を示す．最終アンセーフ故障数削減率(%)は，対象故障からのアンセーフ故障数の削減率を示す.

表 6.3.3.2 より，時間展開数を増加させると検出故障数が減少していく傾向が確認できた．これは，時間展開数の増加とともにテスト不可能故障数の同定数が増加傾向にあるためである.

テストパターン数に対しては,k=5～k=20 に関しては差が小さいことが確認できる．しかしながら，k=2 と k=5～k=20 を比較すると，回路によっては大きく増減している．k=2 と比較して，テストパターン数が大きく減少している回路に対しては,k=5～k=20 においてテスト不可能故障数が多いことが確認できる.これは,テスト不可能故障が多く同定できたため,生成されるテストパターン数が少なったと考えられる.

最終アンセーフパターン数に対しては，k=2 ではほとんどの回路において，ほぼ全てのテストパターンがアンセーフパターンと判定された．k=5～k=20 においては，時間展開数を増加させた方がアンセーフパターン数が少ない傾向があることが確認できた．

最終アンセーフ故障数および最終アンセーフ故障数削減率に対しては，時間展開数を増加させた方が最終アンセーフ故障数が少なくなる傾向が確認できた．しかしながら，s35932 に関しては k=20 より k=15 のほうが最終アンセーフ故障数が大幅に少ないことが確認できた．また，最終アンセーフ故障数削減率は，k=5 で最大 59%(平均 28%)，k=10 で最大 74%(平均 40%)，k=15 で最大 77%(平均 44%)，k=20 で最大 78%(平均 45%)であった．

表 6.3.3.3 に，WSA 閾値 60%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果を示す．表 6.3.3.3 は，図 6.3.3.1 における Step6 から Step8 までの結果である．

表 6.3.3.3：WSA 閾値 60%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果

回路名	時間展開数	対象故障数	検出故障数	テスト不可能故障数	テストパターン数	最終アンセーフパターン数	最終アンセーフ故障数	最終アンセーフ故障数削減率(%)
s5378	2	285	285	0	38	38	285	0
	5		216	69	47	39	142	50
	10		198	87	45	40	140	51
	15		181	104	41	31	111	61
	20		181	104	38	30	112	61
s9234	2	1224	1224	0	214	214	1224	0
	5		529	695	121	84	292	76
	10		526	698	121	53	171	86
	15		526	698	126	55	162	87
	20		525	699	121	53	189	85
s13207	2	1746	1746	0	236	236	1746	0
	5		1584	162	262	242	1082	38
	10		1193	553	240	152	544	69
	15		1188	558	251	149	437	75
	20		1186	560	240	149	507	71
s15850	2	289	289	0	56	55	287	1
	5		240	49	56	37	131	55
	10		223	66	51	21	50	83
	15		223	66	50	19	90	69
	20		222	67	52	13	54	81
s35932	2	8453	8453	0	76	76	8453	0
	5		8453	0	75	74	7791	8
	10		8453	0	55	46	4904	42
	15		8453	0	53	49	7481	11
	20		8453	0	56	42	4032	52
s38417	2	4258	4258	0	239	239	4258	0
	5		4203	55	275	262	2316	46
	10		4200	58	352	286	1384	67
	15		4200	58	389	278	1005	76
	20		4200	58	397	284	1085	75
s38584	2	2944	2944	0	202	202	2944	0
	5		2868	76	250	220	2137	27
	10		2867	77	255	167	1596	46
	15		2867	77	271	185	1847	37
	20		2867	77	246	162	1564	47
b14	2	4594	4594	0	779	627	3180	31
	5		4594	0	856	595	2446	47
	10		4594	0	835	577	2486	46
	15		4594	0	866	582	2306	50
	20		4594	0	857	599	2566	44
b15	2	2654	2654	0	536	395	1930	27
	5		2650	4	550	390	1709	36
	10		2650	4	554	383	1581	40
	15		2650	4	572	381	1388	48
	20		2650	4	572	417	1564	41

表 6.3.3.3 において，各項目の示す内容は表 6.3.3.2 と同様である．表 6.3.3.2 より，WSA 閾値 60%において，k=5～k=20 では検出故障数にほとんど変化が無いことが確認できた．これは，WSA 閾値 50%と比較して対象故障数が減っており，これに伴いテスト不可能故障数も削減されたためと考えられる．

テストパターン数に対しては，k=2～k=20 で差が小さいことが確認できる．しかしながら，s9234 のみ k=2 と k=5～k=20 でテストパターン数の差が大きい．これは，s9234 では，テスト不可能故障数が他の回路より多く同定されたためと考えられる．

最終アンセーフパターン数に対しては，k=2 ではほとんどの回路において，ほぼ全てのテストパターンがアンセーフパターンと判定された．k=5～k=20 においては，時間展開数を増加させた方がアンセーフパターン数が少ない傾向があることが確認できた．

最終アンセーフ故障数および最終アンセーフ故障数削減率に対しては，時間展開数を増加させた方が最終アンセーフ故障数が少なくなる傾向が確認できた．しかしながら，s35932 に関しては k=10 より k=15 のほうが最終アンセーフ故障数が多いことが確認できた．また，最終アンセーフ故障数削減率は，k=5 で最大 76%(平均 42%)，k=10 で最大 86%(平均 59%)，k=15 で最大 87%(平均 57%)，k=20 で最大 85%(平均 62%)であった．

表 6.3.3.4 に，WSA 閾値 70%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果を示す．表 6.3.3.4 は，図 6.3.3.1 における Step6 から Step8 までの結果である．

表 6.3.3.4：WSA 閾値 70%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果

回路名	時間展開数	対象故障数	検出故障数	テスト不可能故障数	テストパターン数	最終アンセーフパターン数	最終アンセーフ故障数	最終アンセーフ故障数削減率(%)
s5378	2	156	156	0	32	32	156	0
	5		127	29	39	26	61	61
	10		109	47	33	22	51	67
	15		100	56	25	17	64	59
	20		100	56	28	19	58	63
s9234	2	519	519	0	129	128	516	1
	5		223	296	62	25	93	82
	10		222	297	63	20	58	89
	15		222	297	69	22	40	92
	20		221	298	66	22	56	89
s13207	2	372	372	0	95	95	372	0
	5		333	39	117	104	257	31
	10		232	140	96	55	107	71
	15		232	140	99	55	110	70
	20		232	140	94	51	119	68
s15850	2	120	120	0	17	16	119	1
	5		101	19	24	8	27	78
	10		95	25	20	4	15	88
	15		95	25	20	6	18	85
	20		94	26	20	1	0	100
s35932	2	4922	4922	0	71	71	4922	0
	5		4922	0	67	65	4363	11
	10		4922	0	50	40	2962	40
	15		4922	0	53	39	2460	50
	20		4922	0	49	38	2218	55
s38417	2	2688	2688	0	168	168	2688	0
	5		2652	36	221	198	1094	59
	10		2650	38	276	157	385	86
	15		2650	38	299	323	139	95
	20		2650	38	316	117	226	92
s38584	2	1448	1448	0	122	122	1448	0
	5		1417	31	134	68	804	44
	10		1417	31	125	42	747	48
	15		1417	31	133	42	725	50
	20		1417	31	143	48	645	55
b14	2	2150	2150	0	550	320	983	54
	5		2150	0	602	255	658	69
	10		2150	0	596	270	689	68
	15		2150	0	622	260	576	73
	20		2150	0	616	281	695	68
b15	2	1107	1107	0	287	182	683	38
	5		1105	2	338	184	529	52
	10		1105	2	336	166	456	59
	15		1105	2	324	165	488	56
	20		1105	2	323	172	504	54

表 6.3.3.4 において，各項目の示す内容は表 6.3.3.2 と同様である．表 6.3.3.4 より，WSA 閾値 70%において，k=5～k=20 では検出故障数にほとんど変化が無いことが確認できた．これは，WSA 閾値 50%と比較して対象故障数が減っており，これに伴いテスト不可能故障数も削減されたためと考えられる．

テストパターン数に対しては，k=5～k=20 に関してはほぼ同じテストパターン数であることが確認できた．また，回路によっては k=2 と k=5～k=20 のテストパターン数に差があることが確認できる．

最終アンセーフパターン数に対しては，k=2 ではほとんどの回路において，ほぼ全てのテストパターンがアンセーフパターンと判定された．k=5～k=20 においては，時間展開数を増加させた方がアンセーフパターン数が少ない傾向があることが確認できた．

最終アンセーフ故障数および最終アンセーフ故障数削減率に対しては，時間展開数を増加させた方が最終アンセーフ故障数が少なくなる傾向が確認できた．また，最終アンセーフ故障数削減率は，k=5 で最大 82%(平均 54%)，k=10 で最大 89%(平均 68%)，k=15 で最大 95%(平均 70%)，k=20 で最大 100%(平均 72%)であった．

表 6.3.3.5 に，WSA 閾値 80%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果を示す．表 6.3.3.5 は，図 6.3.3.1 における Step6 から Step8 までの結果である．

表 6.3.3.5：WSA 閾値 80%におけるキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の結果

回路名	時間展開数	対象故障数	検出故障数	テスト不可能故障数	テストパターン数	最終アンセーフパターン数	最終アンセーフ故障数	最終アンセーフ故障数削減率(%)
s5378	2	71	71	0	13	13	71	0
	5		60	11	23	11	19	73
	10		55	16	21	10	17	76
	15		55	16	16	10	40	44
	20		55	16	18	9	18	75
s9234	2	210	210	0	64	61	206	2
	5		84	126	30	8	14	93
	10		83	127	25	7	23	89
	15		83	127	27	5	8	96
	20		83	127	28	9	25	88
s13207	2	85	85	0	30	30	85	0
	5		67	18	28	23	46	46
	10		49	36	25	11	22	74
	15		49	36	26	11	17	80
	20		49	36	24	11	29	66
s15850	2	51	51	0	4	4	51	0
	5		40	11	8	2	3	94
	10		39	12	7	0	0	100
	15		39	12	6	0	0	100
	20		39	12	7	0	0	100
s35932	2	2326	2326	0	64	55	1111	52
	5		2326	0	55	53	2065	11
	10		2326	0	47	32	887	62
	15		2326	0	49	39	1448	38
	20		2326	0	43	30	946	59
s38417	2	1448	1448	0	110	110	1448	0
	5		1429	19	141	88	253	83
	10		1428	20	175	37	60	96
	15		1428	20	196	28	64	96
	20		1428	20	218	29	45	97
s38584	2	1095	1095	0	93	93	1095	0
	5		1080	15	97	34	577	47
	10		1080	15	82	29	640	42
	15		1080	15	92	35	709	35
	20		1080	15	97	35	620	43
b14	2	775	775	0	284	94	260	66
	5		775	0	311	62	118	85
	10		775	0	315	71	170	78
	15		775	0	314	76	154	80
	20		775	0	313	75	126	84
b15	2	365	365	0	125	53	149	59
	5		365	0	144	49	129	65
	10		365	0	142	48	111	70
	15		365	0	131	51	139	62
	20		365	0	131	50	124	66

表 6.3.3.5 において，各項目の示す内容は表 6.3.3.2 と同様である．表 6.3.3.5 より，WSA 閾値 80%において，k=2～k=20 では検出故障数にほとんど変化が無いことが確認できた．しかしながら，s9234 に関しては対象故障数の半分以上が k=5～k=20 でテスト不可能故障と判定された．そのため，k=2 と k=5～k=20 で検出故障数の差が大きいことが確認できた．

テストパターン数に対しては，k=2～k=20 で差が小さいことが確認できる．s9234 のみ k=2 と k=5～k=20 でテストパターン数の差が大きい．これは，s9234 では，テスト不可能故障数が他の回路より多く同定されたためと考えられる．

最終アンセーフパターン数に対しては，k=2 ではほとんどの回路において，ほぼ全てのテストパターンがアンセーフパターンと判定された．k=5～k=20 においては，時間展開数を増加させた方がアンセーフパターン数が少ない傾向があることが確認できた．

最終アンセーフ故障数および最終アンセーフ故障数削減率に対しては，時間展開数を増加させた方が最終アンセーフ故障数が少なくなる傾向が確認できた．また，最終アンセーフ故障数削減率は，k=5 で最大 94%(平均 66%)，k=10 で最大 100%(平均 76%)，k=15 で最大 100%(平均 70%)，k=20 で最大 100%(平均 75%)であった．

表 6.3.3.6 に，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成と Synopsys 社の TetraMAX ATPG の最終アンセーフパターン数の比較結果を示す．表 6.3.3.6 は，図 6.3.3.1 における Step6 から Step8 までの結果である．

表 6.3.3.6：キャプチャ時消費電力を考慮した マルチサイクルキャプチャ・テスト生成と TetraMAX ATPG の 最終アンセーフパターン数の比較結果

回路名	WSA閾値	対象故障数	テストパターン数						最終アンセーフパターン数					
			k=2	k=5	k=10	k=15	k=20	lp_tmax	k=2	k=5	k=10	k=15	k=20	lp_tmax
s5378	50%	543	71	67	60	63	62	91	71	65	56	61	58	91
	60%	285	38	47	45	41	38	49	38	39	40	31	30	49
	70%	156	32	39	33	25	28	32	32	26	22	17	19	32
	80%	71	13	23	21	16	18	24	13	11	10	10	9	24
s9234	50%	2781	335	204	202	210	213	404	335	178	132	132	124	404
	60%	1224	214	121	121	126	121	232	214	84	53	55	53	226
	70%	519	129	62	63	69	66	128	128	25	20	22	22	122
	80%	210	64	30	25	27	28	67	61	8	7	5	9	60
s13207	50%	5113	419	393	372	384	383	443	419	384	280	274	277	443
	60%	1746	236	262	240	251	240	249	236	242	152	149	149	249
	70%	372	95	117	96	99	94	102	95	104	55	55	51	102
	80%	85	30	28	25	26	24	32	30	23	11	11	11	32
s15850	50%	1120	157	179	163	169	164	149	156	149	101	92	75	149
	60%	289	56	56	51	50	52	51	55	37	21	19	13	51
	70%	120	17	24	20	20	20	22	16	8	4	6	1	21
	80%	51	4	8	7	6	7	9	4	2	0	0	0	7
s35932	50%	21281	97	81	67	75	75	80	97	80	64	68	72	80
	60%	8453	76	75	55	53	56	53	76	74	46	49	42	53
	70%	4922	71	67	50	53	49	48	71	65	40	39	38	48
	80%	2326	64	55	47	49	43	43	55	53	32	39	30	43
s38417	50%	5047	266	335	396	448	444	267	266	325	374	402	406	267
	60%	4258	239	275	352	389	397	234	239	262	286	278	284	234
	70%	2688	168	221	276	299	316	161	168	198	157	160	117	161
	80%	1448	110	141	175	196	218	105	110	88	37	28	29	95
s38584	50%	4696	413	449	464	464	441	371	413	446	456	452	424	371
	60%	2944	202	250	255	271	246	149	202	220	167	185	162	149
	70%	1448	122	134	125	133	143	72	122	68	42	42	48	72
	80%	1095	93	97	82	92	97	48	93	34	29	35	35	48
b14	50%	8290	955	1046	1030	1062	1055	748	874	992	869	908	907	668
	60%	4594	779	856	835	866	857	544	627	595	577	582	599	408
	70%	2150	550	602	596	622	616	361	320	255	270	260	281	189
	80%	775	284	311	315	314	313	181	94	62	71	76	75	34
b15	50%	5626	779	807	836	835	815	756	690	654	656	650	632	645
	60%	2654	536	550	554	572	572	488	395	390	383	381	417	371
	70%	1107	287	338	336	324	323	242	182	184	166	165	172	138
	80%	365	125	144	142	131	131	96	53	49	48	51	50	35

表 6.3.3.6 において，WSA 閾値はアンセーフパターンを判定するための閾値であり，テスト集合中の最大 WSA の 50%，60%，70%，80%の 4 種

類を用いた．対象故障数はテスト生成の対象とした遷移故障数を示す．また，対象故障数は表 6.3.3.1 のアンセーフ故障数と同数であり，同じ故障である．テストパターン数は，テスト生成により生成されたテストパターン数である．最終アンセーフパターン数は，テスト生成で生成したテスト集合のうちアンセーフパターン閾値を超えたテストパターン数を示す．アンセーフパターン閾値は，表 6.3.3.1 と同じ値を使用した．k=2，k=5，k=10，k=15，k=20 は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の時間展開数が 2，5，10，15，20 を示す．lp_tmax は図 6.3.2.1 の 24 行目の処理を，キャプチャ時消費電力を考慮した TetraMAX ATPG で実行したものである．lp_tmax の打ち切り制限は，バックトラック数 100 億回に設定した．また，k=2～k=20 の結果に関しては，表 6.3.3.2～表 6.3.3.5 と同じである．

lp_tmax と k=5～k=20 のテストパターン数を比較すると，平均約 16%から 19%の増加となった．また，lp_tmax と k=2 のテストパターン数を比較すると，平均約 12%の増加となった．

lp_tmax と k=5～k=20 のセーフテストパターン数を比較すると，平均約 52%から 68%の増加となり，多くのテストパターンをセーフテストパターンとして生成できることが確認できた．また，lp_tmax と k=2 のテストパターン数を比較すると，平均約 10%の増加となった．

表 6.3.3.7 に，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成と Synopsys 社の TetraMAX ATPG の最終アンセーフ故障数の比較結果を示す. 表 6.3.3.7 は, 図 6.3.3.1 における Step6 から Step8 までの結果である.

表 6.3.3.7：キャプチャ時消費電力を考慮した マルチサイクルキャプチャ・テスト生成と TetraMAX ATPG の 最終アンセーフ故障数の比較結果

回路名	WSA閾値	対象故障数	最終アンセーフ故障数						打ち切り故障数					
			k=2	k=5	k=10	k=15	k=20	lp_tmax	k=2	k=5	k=10	k=15	k=20	lp_tmax
s5378	50%	543	543	368	304	327	284	543	0	0	0	0	0	0
	60%	285	285	142	140	111	112	285	0	0	0	0	0	0
	70%	156	156	61	51	64	58	156	0	0	0	0	0	0
	80%	71	71	19	17	40	18	71	0	0	0	0	0	0
s9234	50%	2781	2781	1146	736	630	618	2781	0	0	0	0	0	0
	60%	1224	1224	292	171	162	189	1174	0	0	0	0	0	0
	70%	519	516	93	58	40	56	482	0	0	0	0	0	0
	80%	210	206	14	23	8	25	189	0	0	0	0	0	0
s13207	50%	5113	5113	3842	2012	1326	1759	5113	0	0	0	0	0	2
	60%	1746	1746	1082	544	437	507	1746	0	0	0	0	0	2
	70%	372	372	257	107	110	119	372	0	0	0	0	0	0
	80%	85	85	46	22	17	29	85	0	0	0	0	0	0
s15850	50%	1120	1113	682	521	449	346	1120	0	0	0	0	0	0
	60%	289	287	131	50	90	54	289	0	0	0	0	0	0
	70%	120	119	27	15	18	0	117	0	0	0	0	0	0
	80%	51	51	3	0	0	0	17	0	0	0	0	0	0
s35932	50%	21281	21281	20612	17556	17156	19704	21281	0	0	0	0	0	0
	60%	8453	8453	7791	4904	7481	4032	8453	0	0	0	0	0	0
	70%	4922	4922	4363	2962	2460	2218	4922	0	0	0	0	0	0
	80%	2326	1111	2065	887	1448	946	2326	0	0	0	0	0	0
s38417	50%	5047	5047	3164	2773	2249	2475	5047	0	0	0	0	0	6
	60%	4258	4258	2316	1384	1005	1085	4258	0	0	0	0	0	8
	70%	2688	2688	1094	385	139	226	2688	0	0	0	0	0	4
	80%	1448	1448	253	60	64	45	615	0	0	0	0	0	2
s38584	50%	4696	4696	4083	3801	3620	3340	4696	0	0	0	0	0	0
	60%	2944	2944	2137	1596	1847	1564	2944	0	0	0	0	0	0
	70%	1448	1448	804	747	725	645	1448	0	0	0	0	0	0
	80%	1095	1095	577	0	709	620	1095	0	0	0	0	0	0
b14	50%	8290	7170	6826	6369	6465	6505	7136	0	0	0	0	0	1038
	60%	4594	3180	2446	2486	2306	2566	4269	0	0	0	0	0	631
	70%	2150	983	658	689	576	695	1728	0	0	0	0	0	402
	80%	775	260	118	170	154	126	294	0	0	0	0	0	209
b15	50%	5626	4880	4177	4143	4009	3763	4686	0	0	0	0	0	425
	60%	2654	1930	1709	1581	1388	1564	1918	0	0	0	0	0	232
	70%	1107	683	529	456	488	504	618	0	0	0	0	0	119
	80%	365	149	129	111	139	124	169	0	0	0	0	0	42

表 6.3.3.7 において, WSA 閾値はアンセーフパターンを判定するための閾値であり，テスト集合中の最大 WSA の 50%，60%，70%，80%の 4 種類を用いた. 対象故障数はテスト生成の対象とした遷移故障数を示す. また,

対象故障数は表 6.3.3.1 のアンセーフ故障数と同数であり，同じ故障である．最終アンセーフ故障数は，表 6.3.3.6 の最終アンセーフパターンから算出した，アンセーフパターンでのみ検出可能な故障数を示す．また，最終案セーフ故障数には，打ち切り故障も含まれる．打ち切り故障数は，テスト生成時に打ち切り故障となった故障数を示す．k=2 は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の時間展開数 2，k=5 は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の時間展開数 5，k=10 は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の時間展開数 10，k=15 は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の時間展開数 15，k=20 は，キャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の時間展開数 20 を示す．lp_tmax は図 6.3.2.1 の 24 行目の処理を，キャプチャ時消費電力を考慮した TetraMAX ATPG で実行したものである．lp_tmax の打ち切り制限は，バックトラック数 100 億回に設定した．また，k=2～k=20 の結果に関しては，表 6.3.3.2～表 6.3.3.5 と同じである．

lp_tmax と k=5～k=20 の最終アンセーフ故障数を比較すると，平均約 22%から 40%の削減となった．また，lp_tmax と k=2 のテストパターン数を比較すると，平均約 1%の削減となった．

打ち切り故障に着目すると，lp_tmax では b14，b15 において多くの故障が打ち切り故障となっていることが確認できた．一方，提案手法の k=2～k=20 では打ち切り故障数は 0 である．

6.4　結言

本章では，予備実験としてマルチサイクルキャプチャ動作と WSA の関係について解析した．予備実験結果より，10 サイクル以上のキャプチャ動作を行うと WSA が大きく減少することが確認できた．

また，予備実験結果に着目しキャプチャ時消費電力を考慮したマルチサイクルキャプチャ・テスト生成の提案をした．提案手法の評価として，時間展開数 k=2，5，10，15，20 でテスト生成を行い，アンセーフパターン数とアンセーフ故障数を評価した．実験結果より，時間展開数を増加させるほどアンセーフ故障数が削減傾向にあることが確認できた．提案手法では k=20 において WSA 閾値 50%では最大 78%(平均 45%), WSA 閾値 60%では最大 85%(平均 62%), WSA 閾値 70%では最大 100%(平均 72%), WSA 閾値 80%では最大 100%(平均 75%)のアンセーフ故障数の削減ができた．

また，他のテスト生成法との比較実験として，キャプチャ時消費電力を考慮した TetraMAX ATPG とアンセーフ故障数の比較を行った．提案手法(k=5～20)とキャプチャ時消費電力を考慮した TetraMAX ATPG の比較実験の結果，キャプチャ時消費電力を考慮した TetraMAX ATPG と比較して，平均約 22%から 40%の最終アンセーフ故障数の削減が確認できた．

第 7 章

結論

本論文は，VLSI のテスト設計技術(テストコスト削減技術，テスト生成技術)について論じた．

近年，半導体の微細化技術の進歩に伴い，VLSI の集積度が増大している．また，設計自動化技術の進歩により，大規模なデジタルシステムを VLSI 上に実装することが可能となった．これに伴い，テストパターン数が増加傾向にある．また，VLSI の微細化により従来の縮退故障モデルのテストパターンでは検出困難なタイミング遅延を伴う欠陥が存在する．そのため，縮退故障モデルのテストパターンの他に遷移故障モデルやパス遅延故障モデルなどのテストパターンが必要である．このことから，テストパターン数の削減が重要である．

一方，VLSI の低消費電力化設計に伴い，実速度スキャンテストにおけるテスト時消費電力の増大が問題となっている．過度なキャプチャ時消費電力による問題として，電圧降下による誤テストが挙げられる．そのため，VLSI のテスト時消費電力の増大は歩留まり低下の原因の一つとして挙げられる．したがって，歩留まりの損失を抑制するために VLSI のテスト時消費電力の削減が重要である．

本論文では，テスト圧縮に効果的なドントケア判定法を提案した．提案手法は従来のドントケア判定を用いてテスト圧縮より，テストパターン数の削減が可能である．

さらに，本論文ではキャプチャ時消費削減のためのマルチサイクルキャ

プチャ・テスト生成法を提案した．提案手法は従来のキャプチャ時消費電力を考慮したテスト生成法と比較して，アンセーフ故障数の削減が可能である．

これらの研究成果は，今後さらに搭載する回路が大規模化，複雑化することが予想されるVLSIに対するテスト設計技術として有用なものである．

謝辞

本論文は，著者が日本大学大学院生産工学研究科数理情報工学専攻に在学中に日本大学生産工学部細川利典教授のご指導のもとで行ったものである．

本研究を進めるにあたり，終始ご指導を賜り，有益な議論，ご助言を頂きました日本大学生産工学部数理情報工学科の細川利典教授に心より感謝致します. 同じく本研究を進めるにあたり，ご指導を賜り，有益な議論，ご助言を頂きました京都産業大学コンピュータ理工学部コンピュータサイエンス学科の吉村正義准教授に心より感謝致します．

本論文の作成にあたり，様々なご教示を頂きました日本大学生産工学部数理情報工学科の三井和男教授，角田和彦教授，明治大学井口幸洋教授に深く感謝致します．

本研究の実験プログラムの作成に協力して頂きました日本大学大学院生産工学研究科数理情報工学専攻の西間木淳氏，平井敦士氏，高橋慶安氏，北尾隆志氏に深く感謝致します．

本研究の実験用回路の作成に協力して頂きました日本大学生産工学部数理情報工学科の佐藤護氏に深く感謝致します．

参考文献

[1] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT Press, 1985.

[2] Y.Sato, T.Ikeya, M.Nakao, and T.Nagumo, “A BIST Approach for Very Deep Sub-Micron (VDSM) Defects,” Proc. International Test Conference, pp.283-291, 2000.

[3] I.PARK, A.AI-Yamani, and E.J.McCluskey, “Effective TARO pattern generation,” Proc. 23rd VLSI Test Symposium, pp.161-166, 2005.

[4] E.J.McCluskey, A.AI-Yamani, J.C.-M Li, C-W.Tseng, E.Volkerink, F-F.Ferhani, E.Li, and S.Mitra, “ELF-Murphy data on defects and test sets,” Proc. 22nd VLSI Test Symposium, pp.16-22, 2004.

[5] T.Yoshida, and M.Watati, “A New Approach for Low Power Scan Testing,” Proc. International Test Conference, pp.480-487, 2003.

[6] A.Krstic, and K-T.Cheng, *Delay Fault Testing for VLSI Circuits*, Springer, 1998.

[7] P.Girard, N.Nicolici, and X.Wen, *Power-Aware Testing and Test Strategies for Low Power Devices*, Springer, 2009.

[8] C.P.Ravikumar, M.Hirech, and X.Wen, “Test Strategies for Low Power Testing,” Design, Automation and Test in Europe, pp.728-733, 2008.

[9] J.Song, H.Yi, D.Hwang, and S.Park “A Compression Improvement Technique for Low-Power Scan Test Data” IEEE Region 10 Conference TENCON, pp.1-4, 2006.

[10] S.Gerstendorfer, and H-J.Wunderlich, “Minimized power consumption for scan-based BIST,” Proc. International Test Conferance,

pp.77-84, 1999.

[11] N.K.Jha, and S.Gupta, *Testing of Digital Systems*, Cambridge University Press, 2002.

[12] I.Pomerantz, L.M.Reddy, and S.M.Reddy, "COMPACTEST: a method to generate compact test sets for combinational circuits," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.12, pp.1040-1049, 2002.

[13] S.Kajihara, I.Pomerantz, K.Kinoshita, and S.M.Reddy, "Cost-effective generation of minimal test sets for stuck-at faults in combinational logic circuits," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.14, pp.1496-1504, 1995.

[14] P.Goel, and B.C.Rosales, "Test Generation and Dynamic Compaction of Tests," International Test Conferance, pp.189-192, 1979.

[15] 細川利典, 吉村正義, 太田光保, "時間展開モデルを用いた無閉路順序回路の動的テスト圧縮手法の解析," 情報処理学会論文誌, Vol.41, No.4, pp.952-961, 2000.

[16] D.Brelaz, "New Methods to Color the Vertices of a Graph," Communications of the ACM, Vol.22, pp.251-256, 1979.

[17] L.N.Reddy, I.Pomeranz, and S.M.Reddy, "ROTCO: A Reverse Order Test Compaction Technique," Proc. Euro ASIC'92, pp.189-194, 1992.

[18] K.Miyase, and S.Kajihara, "XID: Don't Care Identification of Test Patterns for Combinational Circuits," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, Vol.23, No.2, pp.321-326, 2004.

[19] K.Miyase, K.Noda, H.Ito, K.Hatayama, T.Aikyo, Y.Yamato,

H.Furukawa, X.Wen, and S.Kajihara, “Effective IR-Drop Reduction in At-Speed Scan Testing Using Distribution-Controlling X-Identification,” IEEE/ACM International Conference on Computer-Aided Design, pp.52-58, 2008.

[20] H.Yamazaki, M.Wakazono, T.Hosokawa, and M.Yoshimura, “A Test Compaction Oriented Don’t Care Identification Method Based on X-bit Distribution,” Trans. IEICE TRANSACTIONS on Information and Systems, pp.1994-2002, 2013.

[21] H.Yamazaki, M.Wakazono, T.Hosokawa, and M.Yoshimura, “A Test Compaction Oriented Don’t Care Identification Method,” The 12th Workshop on RTL and High Level Testing, pp.69-76, 2011.

[22] H.Yamazaki, M.Wakazono, T.Hosokawa, and M.Yoshimura, “A Don’t Care Identification Method for Test Compaction,” 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, pp.215-218, 2013.

[23] 山崎紘史, 細川利典, 吉村正義, “キャプチャ消費電力削減のためのマルチサイクルキャプチャテスト生成法,” デザインガイア 2014 -VVLSI 設計の新しい大地-, pp.191-196, 2014.

[24] 山崎紘史, 川連裕斗, 西間木淳, 平井淳士, 細川利典, 吉村正義, 山崎浩二, “マルチサイクルキャプチャテスト生成を用いた低消費電力指向遷移故障テスト生成法,” 信学技報, vol.113, pp.61-66, 2014.

[25] E.K.Moghaddam, J.Rajski, S.M.Reddy, and M.Kassab, “At-Speed Scan Test with Low Switching Activity,” VLSI Test Symposium, pp.177-182, 2010.

[26] M.Yoshimura, H.Ogawa, T.Hosokawa, and K.Yamazaki, “Evaluation

of Transition Untestable Faults Using a Multi-Cycle Capture Test Generation Method," IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp.273-276, 2010.

[27] J.P.Roth, "Diagnosis of automata failures: A calculus and a method," IBM Journal of Research and Development, Vol.10, pp.278-291, 1966.

[28] P.Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," Trans. on Computers, vol.C-30, pp.215-222, 1981.

[29] H.Fujiwara, and T.Shimono, "On the acceleration of test generation algorithms," Trans. on Computers, vol.32, pp.1137-1144, 1983.

[30] M.Schulz, E.Trischler, and T.Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.7, pp.126-137, 1988.

[31] E.Gizdarski, and H.Fujiwara, "SPIRIT: A highly robust combinational test generation algorithm," Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.21, pp.1446–1458, 2002.

[32] T.Larrabee, "Efficient generation of test patterns using Boolean difference," Proc. International Test Conference, pp.795-801, 1989.

[33] T.Larrabee, "Test pattern generation using Boolean satisfiability," Trans on Computer-Aided Design of Integrated Circuits and Systems, vol.11, pp.4-15, 1992.

[34] P.Stephan, R.K.Brayton, and A.L.Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol.15,

pp.1167-1176, 1996.

[35] J.O.M.Silva, and K.A.Sakallah, “Robust search algorithms for test pattern generation,” 27th Annual International Symposium on Fault-Tolerant Computing, pp.152-161, 1997.

[36] J.Shi, G.Fey, R.Drechsler, A.Glowatz, F.Hapke, and J.Schloffel, “PASSAT: Efficient SAT-based test pattern generation for industrial circuits,” Computer Society Annual Symposium on VLSI, pp.212-217, 2005.

[37] H.Kubo, “A procedure for generating test sequences to detect sequential circuit failures,” NEC Res. Dev, pp.69-78, 1968.

[38] J.Savir, and S.Patil “On Broad-Side Delay Test” VLSI Test Symposium, pp.284-290 1994.

[39] J.Savir, “Skewd-Load Transition Test: Part 1: Calculaus.”, Proc. International Test Conference, pp.705-713, 1992.

[40] J.Savir, “Skewd-Load Transition Test: Part 2: Calculaus.”, Proc. International Test Conference, pp.714-722, 1992.

[41] 桜井貴康，岡本光正，瓜屋晋，A.Chandrakasan，R.Amirthara，黒田忠広，武藤伸一郎，水野正之，鈴木晃治朗，宇佐見公良，平木充，濱田基嗣，若林一敏，川口博，山品正勝，中込儀延，堀口真志，*低消費電力,高速LSI技術*，リアライズ理工センター, 1998.

[42] X.Wen, S.Kajihara, K.Miyase, T.Suzuki, K.K.Saluja, L-T.Wang, K.S.Abdel-Hafez, and K.Kinoshita, “A new ATPG method for efficient capture power reduction during scan testing,” Proc. VLSI Test Symposium, pp.6-65, 2006.

[43] X.Wen, K.Miyase, S.Kajihara, T.Suzuki, Y.Yamato, P.Girard,

Y.Ohsumi, and L-T.Wang, “A Novel Scheme to Reduce Power Supply Noise for High-Quality At-Speed Scan Testing,” International Test Conference, pp.1-10, 2007.

[44] X.Wen, Y.Yamashita, S.Kajihara, L-T.Wang, K.K.Saluja, and K.Kinoshita, “On Low-Capture-Power Test Generation for Scan Testing,” Proc. VLSI Test Symposium, pp.265-270, 2005.

[45] R.Sankaralingam, and N.A.Touba, “Controlling Peak Power During Scan Testing,” Proc. VLSI Test Symposium, pp.153-159, 2002.

[46] S.Remersaro, X.Lin, Z.Zhang, S.M.Reddy, I.Pomeranz, and J.Rajski, “Preferred Fill: A Scalable Method to Reduce Capture Power for Scan Based Designs,” International Test Conference, pp.1-10, 2006.

[47] K.M.Butler, J.Saxena, T.Fryars, G.Hetherington, A.Jain, and J.Lewis, “Minimizing Power Consumption in Scan Testing: Pattern Generation and DFT Techniques,” Proc. International Test Conference, pp.355-364, 2004.