検出不能故障に基づく
順序回路の簡単化手法に関する研究

１９９８年１月

四柳　浩之

検出不能故障に基づく
順序回路の簡単化手法に関する研究

１９９８年１月

四柳　浩之

内容梗概

近年の半導体技術の向上にともない，集積回路の大規模高集積化が進み，コンピュータを用いたＣＡＤシステムによる設計が行われている．同じ出力関数を満たす論理回路は無数に存在するため，大規模回路，特に順序回路においては回路の入出力数や状態数が多くなり，最適な回路を得ることは非常に困難な問題である．したがって準最適解を求めてから，一旦得られた論理回路をより最適に近い回路に変換する再合成手法が研究されている．再合成は回路面積の縮小や回路動作の高速化，回路のテスト容易化などの目的で行われるが，回路面積の縮小を目的とし，回路の信号線数およびゲート数，フリップフロップ数を削減する再合成のことを回路簡単化と呼ぶ．

設計された回路に設計者の意図しない冗長な部分が含まれている場合がある．そのような冗長部分を見つけることができれば，その部分の除去により同じ出力系列をもたらす面積の小さい回路を得ることができる．論理回路の冗長部分を発見するために，縮退故障モデルを用いた故障の検出可能性に着目した研究が行われている．縮退故障モデルとは信号線の論理値が固定されるという故障モデルであり，テスト系列の生成においては最もよく用いられているモデルである．ある入力系列を印加したときに，故障を仮定した故障回路がどの状態においても正常回路では得られない出力系列をもたらすならば，故障回路は正常回路と区別できるため，その入力系列がその縮退故障を検出するテスト系列となる．テスト系列が存在しない故障を検出不能故障という．組合せ回路においては検出不能故障に対応する信号線は，信号値をその縮退故障の固定値に置き換えても入力系列に対する出力系列の関係が変化しないため，冗長であることが知られている．一方順序回路では，同じ入力系列を与えても回路の内部状態に応じて得られる出力系列が異なるため，検出不能故障であっても内部状態によっては正常回路では得られない出力系列が故障回路で得られる場合があり，検出不能故障に対応するすべての信号線が冗長であるとは限らない．順序回路の冗長除去手法はいくつか提案されているが，検出不能故障に対応する信号線が冗長であるかどうかを正しく判定しているものは少なく，場合によっては再合成によって得られる回路が元の回路と異なる出力系列を生み出す恐れがあることが指摘されている．

本論文では順序回路の検出不能故障について，対応する信号線が冗長であるかどうかを判定し，再合成において必ず元の回路の出力系列が得られることを保証した簡単化手法をいくつか提案する．

第１章では本研究の背景と論文の構成について述べている．

第２章では，研究の対象となる順序回路についての諸定義と，提案手法に共通に用いられる縮退故障の検出可能性に基づく冗長除去による論理回路の簡単化手法について述べている．

第３章では，順序回路の再合成手法の１つであるリタイミング手法を用いた回路簡単化手法について新たな手法を提案している．リタイミングとは，回路の入力系列に対する出力系列の関係を変えずに，フリップフロップの配置を変えることで再合成を行う手法である．ここでは，従来のリタイミング手法がフリップフロップのリセット端子などを用いて初期化される回路への適用に問題があることを説明し，その問題点を解決するために，特定の初期状態が与えられたとき，その状態における出力系列が必ず再合成後の回路でも得られることを保証するリタイミング手法を提案する．提案手法により初期状態の制限のもとでリタイミングを有効に行うことができることを実験結果から示す．

第４章では，冗長除去手法においてリタイミングを効果的に用いる手法について提案している．リタイミングにより，組合せ回路部で判定することのできない検出不能故障が，組合せ回路部で検出不能であると判定できるようになる場合がある．組合せ回路部においては，検出可能性の判定は比較的容易であり，また検出不能と判定された故障に対応する信号線は除去可能であるため，リタイミングによるこの故障の検出可能性の変換は回路簡単化に有効である．どの状態からも設定できない状態である到達不能状態が存在する部分のフリップフロップをリタイミングにより再配置すると，故障の検出可能性が変換される場合があることを示し，組合せ回路部での冗長部分が増えるようなリタイミング手法を提案する．提案手法により組合せ回路部で判定が可能になる検出不能故障が多く得られることを実験結果から示す．

第５章では，到達不能状態に着目した冗長除去手法について提案している．ここでは縮退故障の検出可能性と到達不能状態の関係を考察する．到達不能状態が故障検出に必要となる縮退故障は検出不能故障であることを示し，さらにその到達不能状態が故障回路においても到達不能であればその検出不能故障が除去可能故障であることを証明する．ここで提案する冗長除去手法では回路の状態数が多い場合でも到達不能状態にのみ着目することで扱う状態数が少なくなり，大規模回路に対しても適用可能である．実験結果より提案手法が多くの除去可能故障を発見できることを示す．

第６章では，第５章で用いられている到達不能状態をフリップフロップの入力関数から求める手法を提案している．設定できないフリップフロップの論理値の組合せをフリップフロップの入

力関数から調べることで到達不能状態が得られる.また記憶容量などの問題から一度にすべてのフリップフロップを扱うことができない場合について,扱うことのできる範囲内で到達不能状態を有効に求めることができるように,フリップフロップ集合の分割を行う手法についても提案している.得られた到達不能状態に基づく冗長除去の実験結果からフリップフロップ集合の分割法の違いによる冗長除去の効果の違いについて述べている.

第7章では,本論文の提案手法についてまとめ,今後の課題について述べている.

目次

第1章　序論

集積回路の技術向上により回路の大規模高集積化が進むに伴い，設計自動化のためのＣＡＤ（Computer Aided Design）システムが必要不可欠となっている．ＣＡＤを用いた設計では，まずハードウェア記述言語（Hardware Description Language, HDL）などにより回路の機能を記述し，それを論理合成ツールにより論理回路に変換する．この処理を論理合成という．論理合成においては，与えられた機能を実現し，回路面積がなるべく小さくなる論理回路を得ることが求められるが，大規模回路に対しては最適解を求めることは困難である．

論理回路は，入力に対する出力が一意に定まる組合せ回路と，入力に対する出力が入力のみならず回路の以前の状態にも関係する順序回路の，２つに分けることができる．順序回路の論理合成では，回路の状態割当を行った後，組合せ部分の論理合成を行う．状態割当では与えられた状態に２進符号を割り当て記憶回路に対応させる．なるべく小さい回路を得るために状態数の最小化などの手法が提案されているが [1,2,3]，状態数が多い場合は最適な状態割当を求めるのは困難である．組合せ部分の論理合成ではまず与えられた仕様を二段論理回路に変形し，その後ゲートや信号線数の少ない回路が得られるように多段論理回路に変換する．しかし特に大規模の多段論理回路を最適化することは本質的に困難な問題であり，まだ確立したアルゴリズムは存在していない．大規模回路については分割して処理を行う必要があり，いくつかの手法を併用して合成が行われている．そのようにして得られた回路の信号線数やゲート数を削減し，さらに最適に近い回路になるように再合成することが研究されている．

論理回路の最適化問題に対しては様々な取り組みがなされている．特に組合せ回路の簡単化についてはよく研究されており，多くの手法がこれまでに提案されている[4]．その代表的手法としては，ESPRESSO[5]などの二段回路の簡単化手法，多段論理回路の簡単化としては許容関数を用いたトランスダクション法[6,7]や論理式の割り算を用いた MIS[8]などの論理関数を用いた手法がある．

さらに論理合成により得られたゲートレベル記述の回路を簡単化する手法として，テスト生成手法を応用した縮退故障の検査可能性に基づく回路簡単化手法が用いられている [9,10,11,12,13,14]．縮退故障とはゲートレベルにおける故障モデルで，各論理ゲートの入力もしくは出力線の論理値が０または１に固定されてしまうという故障モデルである[9,15,16,17,18]．縮退故障モデルは簡単なモデルであるが，縮退故障を対象として生成されたテスト系列を回路に印加するこ

とで実際の物理的欠陥の多くが検出できることから,テスト生成において最もよく用いられている故障モデルである.ある縮退故障を仮定した故障回路と正常回路の入力に対する出力の関係が,あらゆる入力においてもそれぞれ等しくなるならば,テスト生成においてその故障は検出不能であると判定される.

組合せ回路では,検出不能な縮退故障を仮定した故障回路と正常回路の入出力関係は完全に等しいため,故障を仮定した信号線を縮退値に固定しても回路の出力は変わらない.したがって,その信号線を定数と置き換えることで,より信号線数およびゲート数の少ない回路を得ることができる.本論文では縮退故障に対応する信号線が除去できるとき,その故障を除去可能な故障であるという.このような縮退故障の検出可能性に基づく冗長信号線の除去手法を冗長除去手法という.冗長除去手法の応用として,回路に冗長信号線を付加してより多くの冗長信号線の除去を行う手法も提案されている[19,20].

順序回路においても,組合せ回路部分に対しては上記の組合せ回路に対する手法を適用することで簡単化が可能であるが,状態遷移を考慮すると,さらに簡単化を行うことができる場合がある.状態遷移表が与えられている場合の手法としては,すべての状態遷移を考慮して冗長判定を行う手法[21,22]や,特定の初期状態を持つ回路について遷移できない状態を数え上げてそれらの状態を用いて冗長除去を行う手法[23]が提案されている.状態遷移表が与えられていない場合の手法としては,順序回路の組合せ部分を数時刻分つなぎあわせた時間展開回路に対して組合せ回路の冗長除去手法を適用する手法が提案されている[24].

順序回路用のテスト生成を用いた簡単化手法も提案されているが,それにはいくつかの問題点がある.順序回路の検出不能故障は,組合せ回路部分のみで判定できる組合せ回路的検出不能故障と,状態も考慮しなければ判定できない順序回路的検出不能故障の2つに分類される.順序回路では,同一の入力系列に対する出力系列が状態に応じて複数存在するため,検出可能故障と検出不能故障の選別を行うことは本質的に困難な問題である.順序回路のテスト生成手法はいくつか提案されているが[25,26,27],全ての検出不能故障の判定と全ての検出可能故障に対するテスト系列の生成を実用的な時間で行う手法はまだ提案されていない.さらに,テスト生成において検出不能であると判定された故障のすべてが冗長であるとは限らない.順序回路的検出不能故障の中には,故障回路の一部の状態における出力系列が正常回路の出力系列と区別できるものがある.それらの故障に対応する信号線を除去すると,特定の状態においては回路が仕様と異なる出

力系列をもたらすことになる．したがって，検出不能であると判定された故障であっても，除去を行うには，さらにその故障が除去可能であるかどうかを判定する必要がある．文献[28]には，順序回路の検出不能故障の除去可能性についての理論的考察が述べられている．ところが，大規模回路を対象として除去可能性を考慮した簡単化を行う手法については，まだあまり多く研究されていない．そのうちの1つとして，テストを比較的容易にするために回路のフィードバックループを切断した回路に対してテスト生成を行い冗長判定および除去を行う手法が提案されている[29]が，この手法を用いると除去してはならない信号線を誤って除去してしまう場合があることが指摘されている[30]．本論文では除去可能である検出不能故障のみを判定し，除去する手法を提案する．

また組合せ回路とは異なる再合成手法として，リタイミングと呼ばれる順序回路の再合成手法が提案されている[31-41]．リタイミングとは出力系列を変えずに回路内のフリップフロップの再配置を行う手法である．リタイミングは回路動作の高速化[32,33,34,35]やゲート数やフリップフロップ数の削減[36,37]，テスト容易性の向上[38,39,40,41]などに用いられている．リタイミングにより組合せ回路部分が変化するため，リタイミングを行った回路の組合せ部分に簡単化手法を適用する手法が提案されている．

リタイミングを適用することで，回路の特定の初期状態からの出力系列が保存されない場合があることが報告されている[42]．リタイミングを行った後で，元の回路の初期状態に等価な状態を求めることが提案されているが，回路の状態の等価判定は記憶素子の多い，つまり状態数の多い回路に対しては実用的ではない．本論文では，元の回路の初期状態に等価な状態がリタイミングを適用した後の回路で必ず得られるよう保証した新たなリタイミング手法を提案し，回路簡単化に応用する．

さらにリタイミングを行うことで，元の回路の検出不能故障の一部が組合せ回路的検出不能故障に変換されることが報告されている[39]．組合せ回路的検出不能故障は順序回路的検出不能故障に比べ容易に判定できるため，多くの故障を組合せ回路的検出不能故障に変換することができれば検出不能故障の判定に有効となる．本論文では順序回路的検出不能故障が組合せ回路的検出不能故障に変換される可能性の高くなるリタイミング手法を提案する．

順序回路の状態には，いかなる状態からも遷移させることのできない状態が含まれている場合がある．そのような状態を到達不能状態という．順序回路的検出不能故障には，到達不能状態に

設定することが故障検出の必要条件になり，したがって検出不能であるものが存在する．全ての検出不能故障を求めるには全ての状態について考慮しなければならないが，回路内のフリップフロップ数をnとすると状態数は2^nとなり，フリップフロップ数が多い回路では全状態を考慮するのは実質的に不可能となる．本論文では到達不能状態にのみ着目し検出不能故障が除去可能であるか否かを判定する手法を提案する．到達不能状態を用いることで，除去可能な故障の一部が容易に判定できる．

本論文は以下のように構成されている．第2章では本手法の基礎となる冗長除去による簡単化手法について述べる．第3章では，リタイミング手法を特定の初期状態を持つ回路に適用するときの問題点について考察し，特定の初期状態を持つ回路にも適用可能なリタイミング手法を提案し，冗長除去手法と併用した簡単化の結果を示す．第4章では，冗長除去手法と併用した簡単化を行うのに有効なリタイミング手法について提案する．リタイミングにより組合せ回路的検出不能故障に変換される縮退故障の特徴について考察し，なるべく多くの故障が組合せ回路的検出不能故障に変換されるようなリタイミング手法を提案する．第5章では縮退故障の除去可能性について考察し，到達不能状態を用いた冗長除去による回路簡単化手法を提案する．第6章では到達不能状態の判定に論理関数を直接扱うことのできるＢＤＤ（Binary Decision Diagram）を用いた到達不能状態の判定法とその判定法により得られた到達不能状態を用いた簡単化の結果について示す．第7章で本論文のまとめと今後の課題について述べる．

第２章　冗長除去による順序回路の簡単化

２．１　まえがき

図 2-1はLSIの設計工程の一例である．システム設計では仕様を実現するための機能ブロックを構成し，その動作を決定する．機能設計では機能ブロック内部の論理動作を記述するレジスタトランスファレベルの設計を行う．論理設計では論理ゲートへの展開を行うゲートレベルの設計を行う．論理設計で得られたゲートレベル回路に対して回路の検査に用いるテスト系列が生成される．回路設計ではトランジスタレベルの設計を行う．レイアウト設計ではマスクパターンを定めるレイアウトレベルの設計を行う．

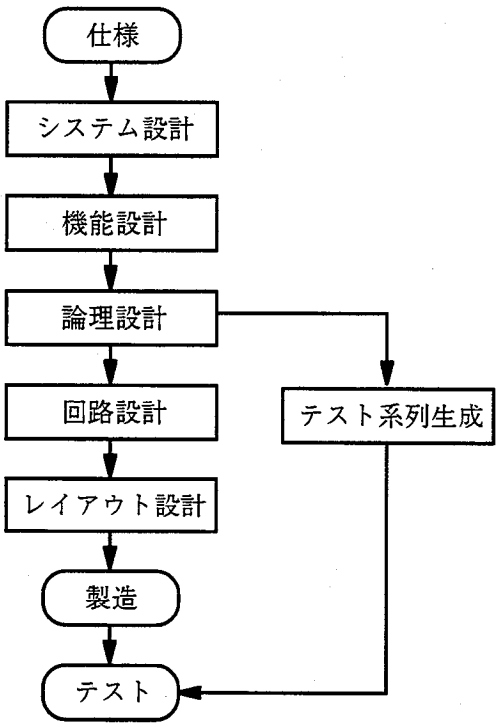


図 2-1　LSIの設計工程

各レベルの設計はLSIの大規模化にともない，計算機を利用したCADツールが用いられるようになったが，人手による設計においてもCADツールによる設計でも，仕様を完全に満たす設計を得るまでには，通常何度か設計を変更することが要求される．したがって，各レベルにおいて設計の検証を行うが，仕様を満たしていても設計者の意図しない冗長部分が含まれている場合がある．そのような冗長部分を除去することでＬＳＩチップの回路面積を削減できる．回路面積

の削減は消費電力の低減や歩留まりの向上に役立つ．また，回路に冗長な部分が存在するとテスト系列生成の際に存在しないテスト系列を生成しようとする無駄な時間が生じ，テストコストの増大につながる．論理設計で得られたゲートレベル回路における冗長部分の除去は主に縮退故障モデルを用いた研究が行われている[9-14]．本章では縮退故障モデルに基づく冗長除去の基本的な操作について述べる．

2．2 順序回路

順序回路は図 2-2に示すように論理ゲートからなる組合せ回路部とフリップフロップなどの記憶素子からなる記憶回路部で構成される．記憶素子に蓄えられている情報のことを内部状態という．順序回路のうち，クロックに同期して内部状態が遷移するものを同期式順序回路と呼び，それ以外のものを非同期式順序回路と呼ぶ．本論文では同期式順序回路を対象とし，記憶素子としてはDフリップフロップを用いることとする．以後，順序回路とは同期式順序回路のことを指し，フリップフロップとはDフリップフロップのことを指すこととする．

順序回路 C は5項組 $C=(X_C, S_C, Z_C, \delta_C, \lambda_C)$ で表すことができる．X_C は入力集合，S_C は状態集合，Z_C は出力集合，δ_C は次状態関数，λ_C は出力関数である．また入力系列の集合を X_C^+，出力系列の集合を Z_C^+ とし，入力系列に対する状態遷移関数と出力系列関数をそれぞれ δ_C^+，λ_C^+ と表す．

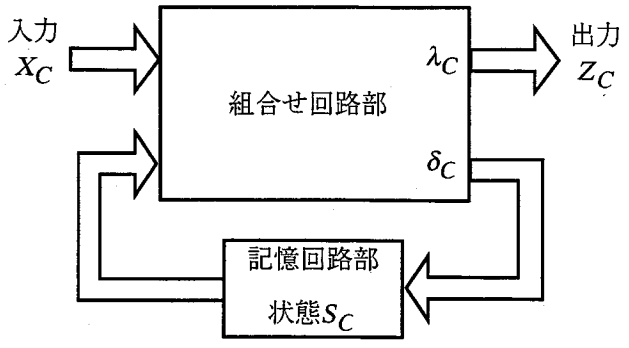


図 2-2 順序回路

順序回路にはいくつかの表現形式が存在する．状態遷移関数と出力関数をグラフの形式で図示したのが状態遷移図である．状態遷移図の例を図 2-3に示す．各ノードは内部状態を表す．ノード間の有向枝は状態間の遷移を示し，枝のラベルは枝の始点の状態における入力と出力の関係を

示している．ラベル中のXはdon't careを表す．図 2-3では s_1 から s_2 へラベル X,1/1 をもつ有向枝があるが，これは状態 s_1 において入力 $(x_1, x_2)=(0, 1)$ または $(1, 1)$ を印加したとき，出力は $z=1$ となり，次状態は s_2 になることを示している．同じ内容を表形式で表したものが状態遷移表である．図 2-3の状態遷移図を表にしたのが表 2-1である．

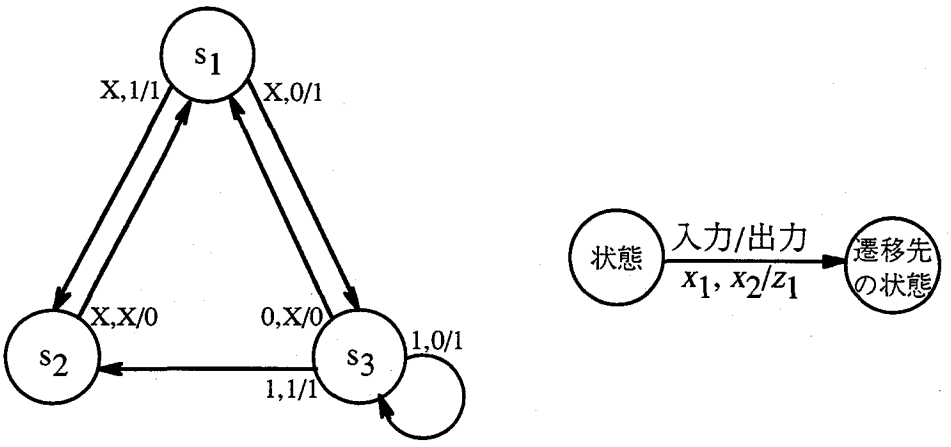


図 2-3　状態遷移図

表 2-1　状態遷移表

現状態	次状態，出力値			
	$(x_1, x_2)=(0, 0)$	$(x_1, x_2)=(0, 1)$	$(x_1, x_2)=(1, 0)$	$(x_1, x_2)=(1, 1)$
s_1	s_3 ,1	s_2 ,1	s_3 ,1	s_2 ,1
s_2	s_1 ,0	s_1 ,0	s_1 ,0	s_1 ,0
s_3	s_1 ,0	s_1 ,0	s_3 ,1	s_2 ,1

記号で表された状態をフリップフロップで構成される記憶回路として実装するために，各状態を二値ベクトルに対応させる必要がある．この操作を状態割当という．状態割当が行われたら与えられた入出力関係から組合せ回路部分の論理合成を行う．表 2-1の状態遷移表を論理合成システム SIS[43]で合成して得られた回路が図 2-4である．この論理合成では状態 s_1 が状態 $(FF_1, FF_2)=(0, 0)$ に，s_2 が $(0, 1)$ に，s_3 が $(1, 1)$ にそれぞれ割り当てられている．

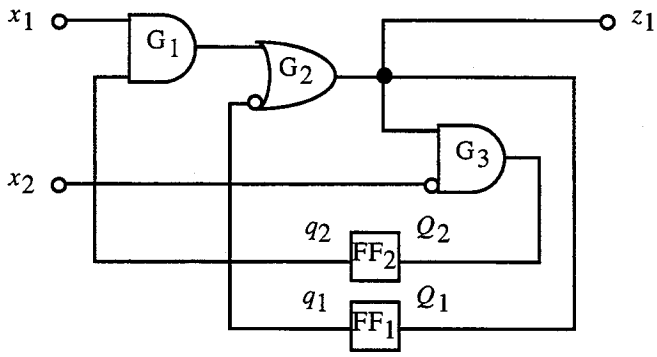


図 2-4　合成された回路

二値論理関数を表現する方法としては図 2-4のような各ゲートとゲート間の接続を記述するゲートレベル記述や，二分決定グラフ（Binary Decision Diagram : BDD [44,45,46,7,47,48]）で表現する方法がある．ＢＤＤは有向二分木で表現されたグラフであり，グラフの非終端ノードは入力変数に対応し，終端ノードは関数の論理値に対応する．ＢＤＤの根のノードが論理関数を表している．非終端ノードには２つの枝が存在するが，各枝に接続するノードはシャノン展開を行って得られる関数を表現している．図 2-5は図 2-4の回路の出力関数のＢＤＤである．根のノードは回路の出力関数 $\delta(x_1, q_1, q_2) = \overline{q_1} \vee (q_2 \wedge x_1)$ を表している．この関数を根のノードに対応する変数 q_1 でシャノン展開すると $\delta(x_1, q_1, q_2) = (\overline{q_1} \wedge \delta(x_1, 0, q_2)) \vee (q_1 \wedge \delta(x_1, 1, q_2))$ となるが，根のノードの負（０）の枝に接続するノードは関数 $\delta(x_1, 0, q_2) = 1$ を表し，正（１）の枝に接続するノードは関数 $\delta(x_1, 1, q_2) = q_2 \wedge x_1$ を表していることが分かる．ＢＤＤの各ノードはそのノードの入力変数，負の枝に接続するノード，正の枝に接続するノードの３つの要素で表現することができる．表 2-2に図 2-5のＢＤＤの各ノードを３つの要素で表現したものを示す．ここで，FALSE とは論理値０を表す終端ノードを，TRUE とは論理値１を表す終端ノードを示す．

ＢＤＤにおいて変数の順序を固定し，グラフの同型部分の共有による簡略化を行うと論理関数は一意的に表現できる．そのようにして論理関数を表現したＢＤＤのことを既約順序付きＢＤＤ（ROBDD[7,46,47,48]）という．ROBDD を用いると，論理関数の等価判定や論理演算を容易に行うことができる．通常この ROBDD のことをＢＤＤとよぶ．本論文においても以降 ROBDD のことを簡単にＢＤＤと表記する．

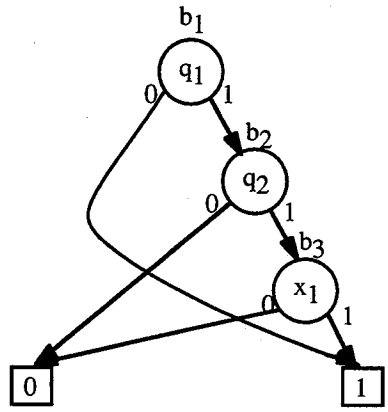


図 2-5　出力関数のＢＤＤ

表 2-2　ＢＤＤの３要素による表現

ノード＝（入力変数，負の接続ノード，正の接続ノード）
b_1 = (q_1, TRUE, b_2)
b_2 = (q_2, FALSE, b_3)
b_3 = (x_1, FALSE, TRUE)

２．３　初期化と到達不能状態

順序回路の初めの内部状態は不明である．したがって入力系列に対する出力系列は状態に応じて異なり一意に知ることができない．そのため，回路を使用する際にはまず内部状態を既知の状態に設定する必要がある．内部状態を既知の状態に設定することを回路の初期化という．初期化するためには，セット／リセット端子を持つフリップフロップを使用し外部から直接フリップフロップの状態を設定する方法と，入力系列により未知の内部状態から既知の内部状態に遷移させる方法がある．入力系列による初期化を行うには，内部状態がいかなる状態であっても，ある一定の入力系列によりある特定の状態に遷移させることが可能でなければならない．この特定の状態を初期状態と呼び，初期状態へ遷移させる入力系列を同期化系列という．

［定義 2-1：同期化系列］　回路 C が式(2.1)を満たすならば，回路は同期化可能であるという．また，このときの入力系列 x_s のことを同期化系列という．

$$\exists s_r \in S_C, \exists x_S \in X_C^+, \forall s \in S_C, \delta_C^+(x_s, s) = s_r \tag{2.1}$$

同期化系列は回路の状態をある特定の1状態に設定する系列である.同期化系列を印加したあとの入力系列に対する出力系列は一意に定まる．回路の初期化に用いる入力系列としては，同期化系列の条件をもう少し緩くした,以下に定義する弱同期化系列と呼ばれる系列が用いられる場合もある．弱同期化系列は，未知の内部状態を，入力系列に対する出力系列の関係が等しい状態（等価な状態）のある集合に含まれる，いずれかの状態へと設定することのできる入力系列である．同期化系列は弱同期化系列に含まれる．

［定義 2-2：状態の等価性］　回路 C の2状態 $s_a, s_b \in S_C$ が式(2.2)を満たすならば，s_a と s_b は等価であるといい，$s_a \sim s_b$ と表す．

$$\forall x \in X_C^+, \lambda_C^+(x, s_a) = \lambda_C^+(x, s_b) \tag{2.2}$$

［定義 2-3：弱同期化系列］[28,49]　回路 C が式(2.3)を満たすとき，回路 C は弱同期化可能であるという．このときの入力系列 x_w を弱同期化系列という．

$$\exists s_r \in S_C, \exists x_w \in X_C^+, \forall s \in S_C, \delta_C^+(x_w, s) \sim s_r \tag{2.3}$$

例 2-1：弱同期化系列の例を示す．図 2-6は図 2-4の回路の状態遷移図である．この回路では入力系列 $x_1 = 11, x_2 = 00$ を印加することで，どの状態も状態 (1, 1) へ遷移する．したがって，入力系列 $x_1 = 11, x_2 = 00$ は弱同期化系列である．この系列は1つの状態 (1, 1) へと回路の状態を遷移させるため，同期化系列になっている．❑

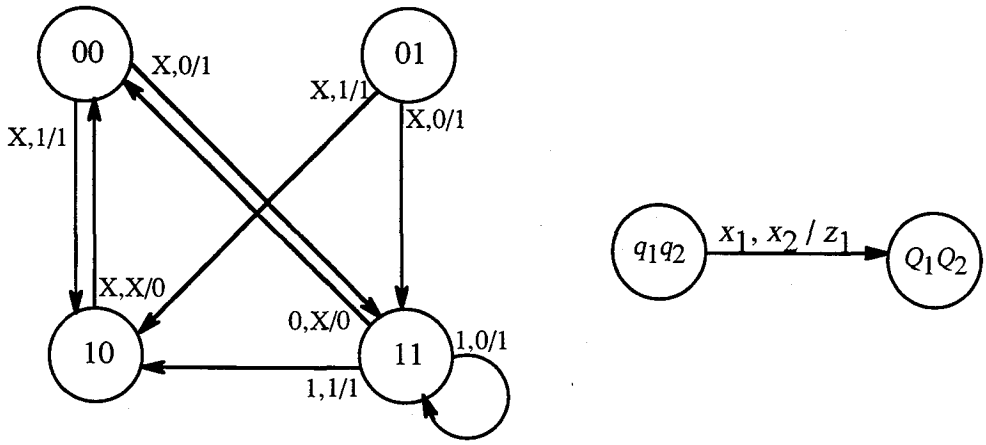


図 2-6　図 2-4の回路の状態遷移図

同期化系列，弱同期化系列を印加すると回路の状態が既知の状態へ遷移するため，それらの系列印加後の入力系列に対する出力系列は一意に定まる．テスト系列に同期化系列を用いると正常回路の出力系列は状態に依存せず1つの出力系列だけを故障回路の出力系列と比較すればよくなるため，同期化系列はテスト生成においても有用である．このことから同期化系列が与えられていない回路に対しても同期化系列を求める手法がいくつか提案されている[50,51]．

回路のフリップフロップ数をnとすると回路の状態数は 2^n となるが，実際に使用される状態数は 2^n 個よりも少ないことが多い．弱同期化系列を印加した後に遷移させることのできない状態を無効状態といい，また，その中でどの状態からも遷移させることができない状態を到達不能状態と呼ぶ．到達不能状態は以下のように定義される．

［定義 2-4：到達不能状態］　順序回路 M_C において状態 s_u が式 (2.4) を満たすとき（つまり，状態 s_u への遷移がないとき），状態 s_u を到達不能状態という．

$$\forall_{x \in X_C}, \forall_{s \in S_C}, s_u \neq \delta_C(x, s) \tag{2.4}$$

図 2-6の状態遷移図においては状態 (0, 1) が到達不能状態である．到達不能状態は，回路の組合せ回路部分のみで判定できるため，比較的容易に求められる．本論文では到達不能状態に着目した冗長除去手法を提案する．

2．4　順序回路の等価性

回路を再合成する際には，その回路の入力系列に対する出力系列の関係が，再合成後の回路でも同じでなければならない[52]．再合成前の回路と再合成後の回路の満たすべき条件として，以下に2つの回路の等価性および弱等価性を定義する．

［定義 2-5：2回路間の状態の等価性］　入出力数の等しい回路 $A = (X, S_A, Z, \delta_A, \lambda_A)$ と回路 $B = (X, S_B, Z, \delta_B, \lambda_B)$ の状態 $s_a \in S_A$, $s_b \in S_B$ が式 (2.5) を満たすとき，s_a と s_b は等価であるといい，$s_a \sim s_b$ と表す．

$$\forall_{x \in X^+}, \lambda_A^+(x, s_a) = \lambda_B^+(x, s_b) \tag{2.5}$$

［定義 2-6：2回路の等価性］　入出力数の等しい回路 $A=(X, S_A, Z, \delta_A, \lambda_A)$ と回路 $B=(X, S_B, Z, \delta_B, \lambda_B)$ が式(2.6)を満たすとき，回路Aと回路Bは等価であるという．

$$\left(\forall s_a \in S_A, \exists s_b \in S_B, s_b \sim s_a\right) \wedge \left(\forall s_b \in S_B, \exists s_a \in S_A, s_a \sim s_b\right) \tag{2.6}$$

等価な2回路においては，片方の回路で得られる出力系列は必ずもう一方の回路においても得られることが保証されている．したがって，再合成で得られた回路が元の回路と等価であれば，その回路を元の回路と交換することが可能である．さらに再合成の対象となる回路が，初めに必ず弱同期化系列を印加して用いられ，弱同期化系列印加後の出力系列のみが使用されるときは，再合成後の回路と元の回路が次の弱等価の条件を満たしていれば，等価ではなくても交換することが可能である．

［定義 2-7：2回路の弱等価性］　入出力数の等しい回路 $A=(X, S_A, Z, \delta_A, \lambda_A)$ と回路 $B=(X, S_B, Z, \delta_B, \lambda_B)$ が式(2.7)を満たすとき，回路Aと回路Bは弱等価であるという．

$$\exists w_a \in X^+, \exists w_b \in X^+, \forall s_a \in S_A, \forall s_b \in S_B, \delta_A^+(w_a, s_a) \sim \delta_B^+(w_b, s_b) \tag{2.7}$$

式 (2.7) は回路Aの入力系列 w_a 印加後の状態が，回路Bの入力系列 w_b 印加後の状態に等価であることを示している．このことから，回路Aの入力系列 w_a 印加後の入力系列に対する出力系列の関係は，回路Bの入力系列 w_b 印加後の入力系列に対する出力系列の関係に等しいことがわかる．したがって，回路Aの弱同期化系列 w_a に対して，回路Bに式 (2.7) を満たす入力系列 w_b が存在していれば，w_b を弱同期化系列として用いることで回路Bは回路Aと交換できる．

2．5　縮退故障の検出可能性による分類

縮退故障モデルとは，信号線の値がある一定の論理値（0または1）に固定される故障モデルである．縮退故障モデルは実際の故障の影響の多くを包含するため，テスト系列を生成する対象として使われている．正常回路を $G=(X_G, S_G, Z_G, \delta_G, \lambda_G)$，縮退故障 f の存在する回路を $F=(X_F, S_F, Z_F, \delta_F, \lambda_F)$ とすると，縮退故障は以下のように分類される [22]．正常回路の入力集合と故障回路の入力集合は等しいので，ここでは $X_G=X_F=X$ と表記する．

まず，縮退故障はテスト系列が存在する検出可能故障と存在しない検出不能故障に分けられる．

［定義 2-8：検出可能故障］　正常回路 G と故障回路 F が式(2.8)を満たすとき，故障 f は検出可能故障であるという．式(2.8)を満たす入力系列 x_T は故障 f のテスト系列である．

$$\exists x_T \in X^+, \forall s_g \in S_G, \forall s_f \in S_F, \lambda_G{}^+(x_T, s_g) \neq \lambda_F{}^+(x_T, s_f) \tag{2.8}$$

［定義 2-9：検出不能故障］　検出可能でない故障を検出不能故障という．

さらに検出不能故障は，組合せ回路部において正常回路と故障回路の出力関数，次状態関数がそれぞれ等しい組合せ回路的検出不能故障と，順序回路的検出不能故障の2つに分けられる．

［定義 2-10：組合せ回路的検出不能故障］　正常回路 G と故障回路 F が式(2.9)を満たすならば，その故障を組合せ回路的検出不能故障という．

$$\forall x \in X, \forall s \in S, (\lambda_G(x,s) = \lambda_F(x,s)) \wedge (\delta_G(x,s) = \delta_F(x,s)) \tag{2.9}$$

［定義 2-11：順序回路的検出不能故障］　組合せ回路的検出不能故障以外の検出不能故障を順序回路的検出不能故障という．

例 2-2：組合せ回路的検出不能故障と順序回路的検出不能故障の例を示す．

図 2-7に示した回路の信号線 r の1縮退故障を考える．故障回路と正常回路の関数が異なるのは信号線 r の論理値が0になるときのみである．このとき，一意的に入力 x_2 の論理値は1，ゲート G_1 の出力値は0，ゲート G_3 の出力値は1に定まり，出力 z_1 の論理値が1となる．故障回路においても同様に出力 z_1 の論理値が1となり，またこの故障の影響はフリップフロップの入力線の値にも影響しないので，正常回路の出力関数，次状態関数と故障回路の出力関数，次状態関数はそれぞれ等しい．よって信号線 r の1縮退故障は組合せ回路的検出不能故障である．

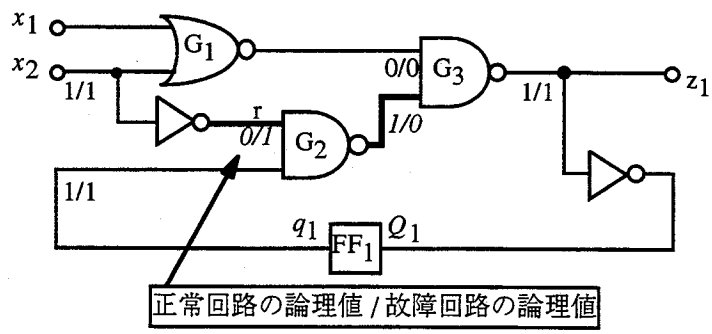


図 2-7　組合せ回路的検出不能故障の例

次に順序回路的検出不能故障の例を示す．図 2-8は図 2-4の回路である．ここで信号線ｒの１縮退故障を考える．信号線ｒの論理値が０になるとき入力x_2=0により故障の影響がフリップフロップ FF_2 の入力線 Q_2 に伝播する．また，信号線ｒの論理値を０とするには信号線ａの論理値は必ず０であるので，フリップフロップ FF_1 の入力線 Q_1 の論理値は０となる．そして，次の時刻において， q_1 の論理値が０であり，ゲート G2 の出力ａの論理値が故障の有無に関わらず一意的に１となるため，故障の影響はゲート G_2 で打ち消される．したがって，故障の影響が外部出力まで伝播できないため，信号線ｒの１縮退故障は検出不能であることが分かる．故障回路の状態遷移関数は正常回路の状態遷移関数と異なるため，この故障は組合せ回路的検出不能故障ではなく，順序回路的検出不能故障である． ❑

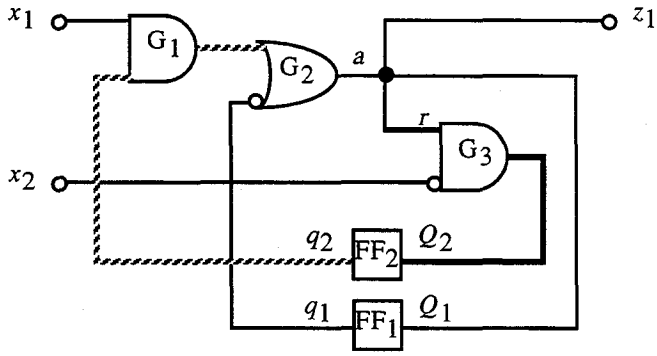


図 2-8　順序回路的検出不能故障の例（図 2-4の回路）

組合せ回路的検出不能故障は組合せ回路のテスト生成手法を用いた判定法などで容易に判定することができるが，順序回路的検出不能故障の判定は困難であり，特に大規模回路についてはすべての順序回路的検出不能故障の判定を実用的な時間で行うことのできる手法は提案されていない．

２．６　冗長除去による回路簡単化

ある縮退故障について故障回路が正常回路と等価であれば故障回路の出力系列は正常回路の出力系列と等しくなる．このとき故障を仮定した信号線は冗長な信号線であり，縮退故障の縮退値に対応した信号線と取り替えることで回路内の信号線数を削減することが可能である．故障の

検出不能性を利用して回路の冗長な信号線の除去を行う手法を冗長除去手法という[4,9]. また冗長除去の対象にできる縮退故障を*除去可能な縮退故障*という.

組合せ回路的検出不能故障は,故障回路の出力と状態遷移がすべて正常回路の出力と状態遷移に等しいため, 除去可能である. ところが順序回路的検出不能故障には除去可能であるものと除去可能でないものが存在する. 弱同期化系列印加後の出力系列のみ用いられると仮定すると, 故障回路が正常回路と弱等価になる場合に順序回路的検出不能故障が除去可能となる.

縮退故障が除去可能になる信号線が存在すれば,冗長除去により信号線数を少なくとも1つ削減できる. 冗長除去を行うことで, 入力系列に対する出力系列の関係が等しく信号線数の少ない回路を得ることができる.

信号線rのα縮退故障（$\alpha \in \{0, 1\}$）が除去可能であるときの冗長除去の手順を以下に示す.

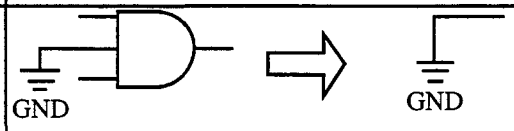
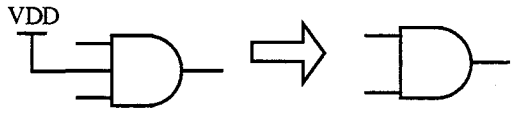
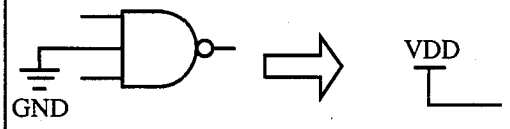
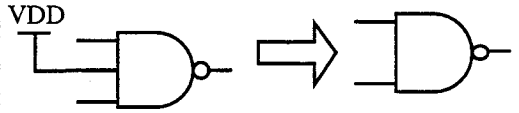
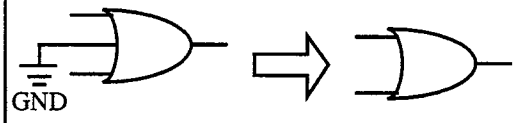
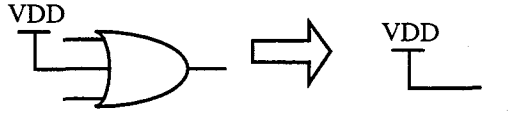
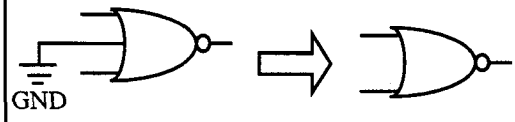
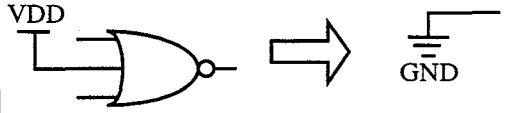
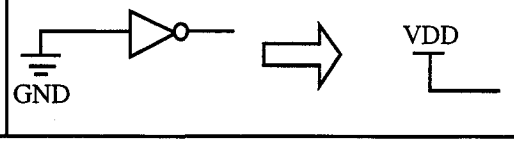
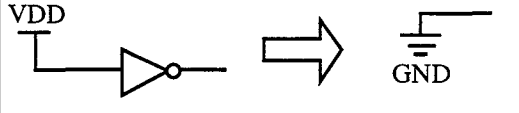
[手順：信号線rのα縮退故障に基づく冗長除去]

1）信号線rを固定値αを供給する信号線と置き換える. 固定値αを供給する信号線とは, $\alpha = 0$ならばＧＮＤ端子に接続する信号線, $\alpha = 1$ならばＶＤＤ端子に接続する信号線である.

2）固定値を供給する信号線を入力に持つゲートに対して表 2-3に示す処理を行う. 表の処理を行うことのできるゲートが存在しなくなるまでこの処理を繰り返す.

3）出力信号線がすべて除去されたゲートを入力線とともに除去する. この処理を出力がすべて除去されたゲートがなくなるまで繰り返す.

表 2-3　冗長除去の手順

ゲート	入力が固定値 0	入力が固定値 1
AND		
NAND		
OR		
NOR		
NOT		

第3章　特定の初期状態を考慮したリタイミング手法

3．1　まえがき

順序回路の再合成手法の1つとして,入力系列に対する出力系列の関係を保ったままフリップフロップの再配置を行う手法である，リタイミングが提案されている[31-41]．リタイミングにより回路動作の高速化[32-35]や，ゲート数やフリップフロップ数の削減[36,37]，テスト容易性の向上[38-41]などが実現されている．

リタイミング前後の回路は弱等価であることが知られている[53]．しかし，フリップフロップのリセット端子等を用いて特定の初期状態に初期化される回路では,リタイミングを行うときに初期状態と等価な状態がリタイミング後の回路に存在することを保証しなければならない．

本章では特定の初期状態が与えられている回路について,リタイミング後の回路がその初期状態と等価な状態を必ず持つようにリタイミングを行う提案手法[54]について述べる．

3．2　リタイミング

リタイミングとは順序回路の再合成手法の1つで,回路内のフリップフロップの位置を再配置する手法である．リタイミングは以下に示すような4つの操作の組合せによって行われる．

・ゲート部の前方への再配置

・ゲート部の後方への再配置

・分岐部の前方への再配置

・分岐部の後方への再配置

ここで，前方への再配置とは，入力側にあるフリップフロップを出力側へ再配置する操作を表し，後方への再配置とは，その逆の操作を表す．図 3-1(a) のように，あるゲートの全入力線上にフリップフロップが配置されているときには，それらのフリップフロップを取り除き，ゲートの出力線上にフリップフロップを配置することができる.この操作をゲート部の前方への再配置という．逆に図 3-1 (b) のようにフリップフロップがゲートの出力に配置されているとき，そのフリップフロップをすべての入力線上に再配置できる.この操作をゲート部の後方への再配置という．

また，図 3-1 (c) のように分岐のすべての枝にフリップフロップが配置されているときは，それらを幹へと再配置できる．この操作を分岐部の後方への再配置という．逆に図 3-1 (d) のよう

にフリップフロップが分岐の幹に配置されているときは，それをすべての枝に再配置することができる．この操作を分岐部の前方への再配置という．

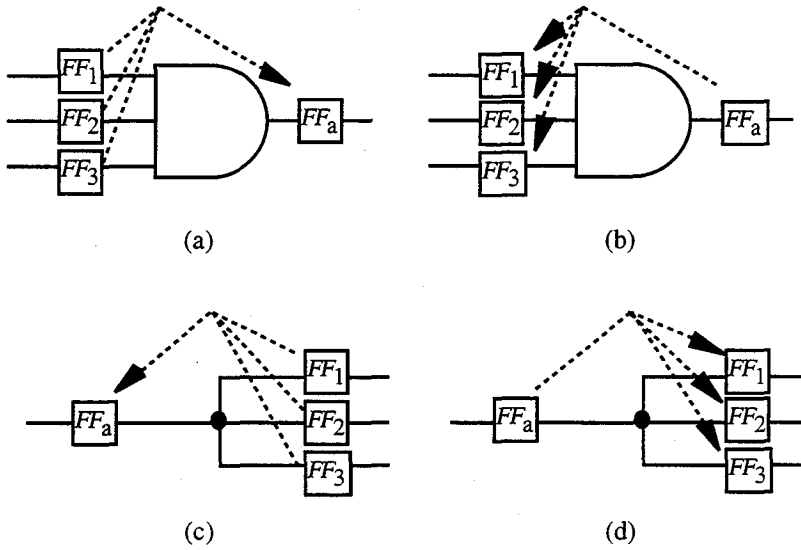


図 3-1　リタイミングの基本操作

(a)ゲート部の前方への再配置　　(b)ゲート部の後方への再配置

(c)分岐部の後方への再配置　　(d)分岐部の前方への再配置

リタイミングを行った後の回路は，元の回路と完全には等価ではないことがある．ゲート部分の前方および後方への再配置のみを行った場合は，リタイミング前後の回路は等価である．図3-1 (a)，(b) において状態 $(FF_1, FF_2, FF_3)=(1, 1, 1)$ は状態 $(FF_a)=(1)$ と等価であり，その他の状態はすべて $(FF_a)=(0)$ と等価である．

一方で，分岐部の後方への再配置を行うと，どの状態からも遷移できない到達不能状態は等価な状態を持たないことがある．図 3-1 (c) において状態 $(FF_1, FF_2, FF_3)=(1, 1, 1)$ は状態 $(FF_a)=(1)$ と等価であり，状態 $(FF_1, FF_2, FF_3)=(0, 0, 0)$ は $(FF_a)=(0)$ と等価であるが，(FF_1, FF_2, FF_3) に示されるその他の状態に等価な状態が存在するとはいえない．例えば，$(FF_1, FF_2, FF_3)=(1, 1, 0)$ と等価な状態は存在しない．到達不能状態に依存する出力は初期化系列印加中にしか現れないため，正常回路の通常動作には影響しない．しかし，故障の検出可能性に変化を与えるため，冗長除去に有効に用いることができる．

分岐部の前方への再配置を行うと，元の回路の各状態にそれぞれ等価な状態が再配置を行った後の回路に必ず存在している．しかし，再配置後の回路のある状態が，元の回路に等価な状態を

持たない場合がある．図 3-1 (d) において状態 (FF_a)=(1) は状態 (FF_1, FF_2, FF_3)=(1, 1, 1) と等価であり，状態 (FF_a)=(0) は (FF_1, FF_2, FF_3)=(0, 0, 0) と等価である．しかし，その他の状態 ((FF_1, FF_2, FF_3)=(0, 1, 1) など) は必ずしも状態 (FF_a)=(0), (1) のどちらかに等価であるとはいえない．このときは元の回路に等価な状態を持たない状態がリタイミングにより付け加わっていることになる．

3．3　初期状態依存問題

従来のリタイミング手法では同期化系列のような初期化系列を印加した後に回路を動作させ，初期化系列印加中の出力系列は無視することが前提となっていた．リタイミング前後の回路は弱等価であるので，初期化が終了した後は必ず元と同じ出力系列をもたらすことが保証される．しかしながら，初期化系列を用いて回路を初期化するのではなく，リセット機能付きのフリップフロップ等を用いた初期化を行う回路にリタイミングを行うと，リタイミング後の回路で元の回路の出力系列が得られない場合がある．図 3-2にその例を示す．

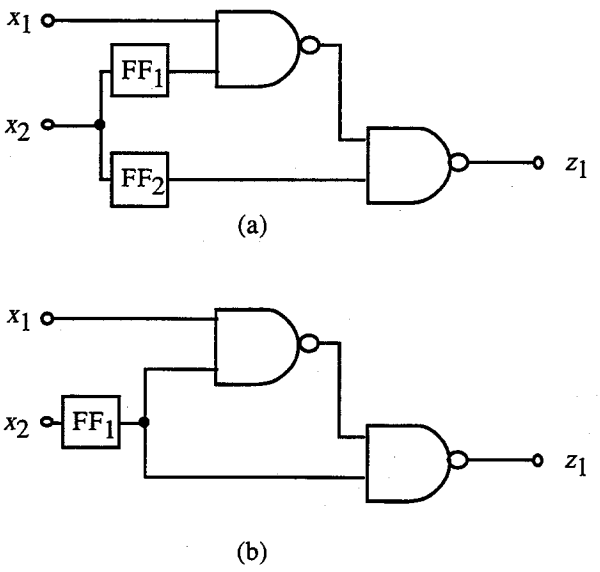


図 3-2　リタイミングと回路の等価性

図 3-2 (a)の回路からリタイミングにより図 3-2 (b)の回路が得られたとする．図 3-2 (b)では，入力 x_1 の値が 1 ならば，出力 z_1 の値は常に 1 となる．しかし図 3-2 (a)の元の回路では初期状態(FF_1, FF_2) = (0, 1)のとき x_1 = 1 で出力の値は 0 になる．従って，フリップフロップのセット機

能またはリセット機能により，回路の状態が $(FF_1, FF_2) = (0, 1)$ に初期化されているときは，このリタイミングは無効である．

リタイミング前後で回路が異なる出力系列をもたらすのは，以下の条件が同時に満たされるときである．

[条件]

i) リタイミング前の回路が到達不能状態を持ち，その中の一つの状態 s_i に初期化されて回路が使われる．

ii) リタイミング後の回路に状態 s_i と等価な状態が存在しない．

ゲートの前後にフリップフロップを再配置するリタイミングでは，変換前の回路のどのような状態に対しても，その状態に等価な状態が変換後の回路に必ず存在する．すなわち，変換前の回路がどのような初期状態をとったとしても，変換後の回路は同じ出力系列を与えることができる．一方で，分岐の枝にあるフリップフロップを分岐の前方に再配置するリタイミングでは，図 3-2に示したように，変換後の回路が変換前の到達不能状態と等価な状態を持たない場合が起こりうる．この場合には，変換後の回路は異なる出力系列を与えることになる．文献[42]では初期状態と等価な状態がリタイミング後の回路に存在するかどうか調べ，等価な状態がなければその再合成を破棄する手法が提案されている．しかし回路の状態数が多い場合は全ての状態について等価性の判定を行うことは実用的ではない．次章では，変換前の回路の初期状態に相当する状態を変換後の回路が必ず持つように分岐点におけるリタイミングに制限をつける手法を提案する．

3．4　初期状態を考慮したリタイミング手法

本手法では，各フリップフロップに5値の論理値 $\{0, 1, X, X_0, X_1\}$ を持たせて，与えられた初期状態に相当する状態を変換後の回路で必ず実現できるようにリタイミングを行う．まず，初期状態として各フリップフロップに強制的に制御される初期値を与える．

ゲートの出力のフリップフロップを入力へ再配置するときは，ゲート出力の信号線の値が元のフリップフロップの値と等しくなるように，リタイミング後のフリップフロップに値を与えることにより，初期状態と等価な状態を必ず保つことができる．例えばANDゲートの出力のフリップフロップを入力に再配置する場合には，もとのフリップフロップの値が1であれば入力へ再配置したフリップフロップの値をすべて1とし，0であれば入力へ再配置したフリップフロップの

いずれか１つを０にすれば再配置の前と等価な状態が保たれる．再配置後の初期値が一意的に定まらない場合のために論理値 X_0 と X_1 を次のように定義する．論理値 X_0 は AND/NAND ゲートの入力へ再配置したフリップフロップに与えられる．この論理値は他の入力のフリップフロップの値が１つでも ０ として用いられれば X に，他の全ての入力のフリップフロップの値が１またはXとして用いられれば元の初期状態を保つため ０ に変更される．論理値 X_1 も同様に，OR/NOR ゲートの入力へ再配置したフリップフロップに与える．

同様にゲートの入力から出力への再配置では初期状態におけるゲートの出力値を再配置したフリップフロップの初期値とし，分岐の出力方向への再配置ではもとのフリップフロップの値を各枝へ再配置したフリップフロップの値とする．

分岐の入力方向への再配置では，各枝のフリップフロップの値が初期値と矛盾した値を持たない場合にのみ再配置を行う．図 3-3に例を示す．

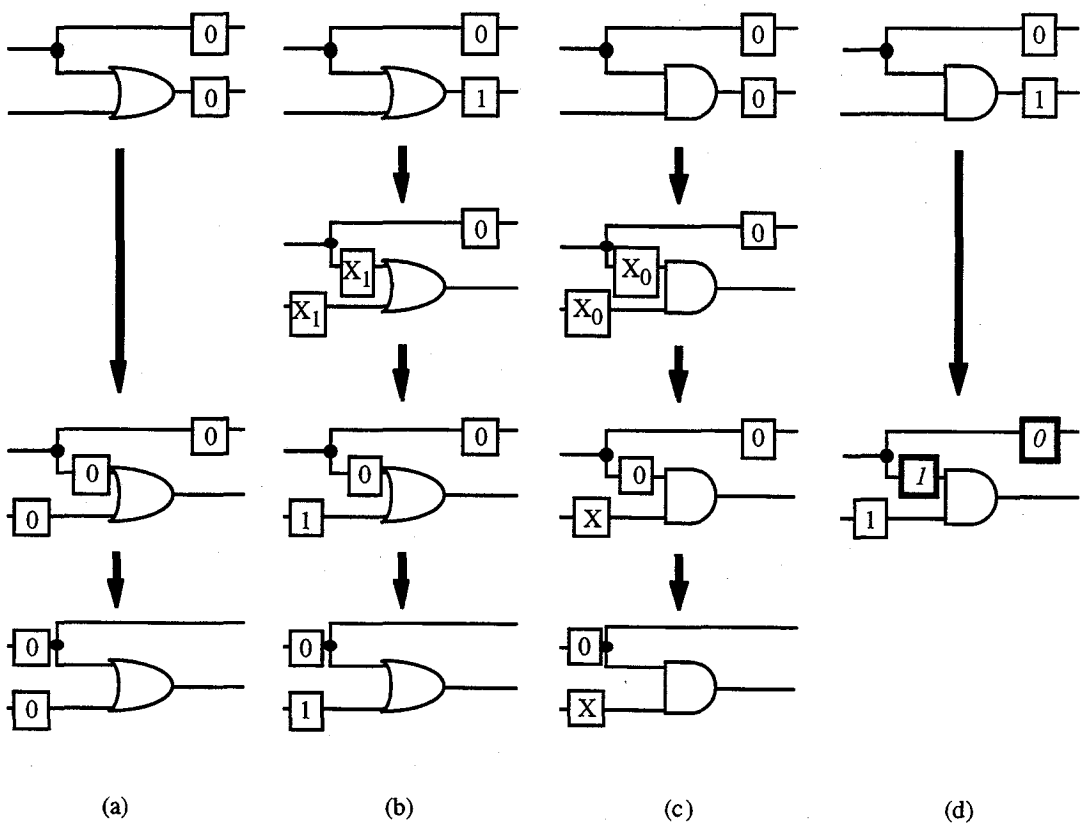


図 3-3　５値の論理値を用いたリタイミング

図 3-3 (a)の回路では分岐部へ再配置したフリップフロップの値が0であり，分岐の枝のフリップフロップの値がすべて等しいため，分岐部の後方への再配置ができる．再配置されたフリップフロップには0を与える．図 3-3(b)の回路ではORゲートの入力へ再配置された2つのフリップフロップのどちらかが論理値1を持つように再配置すればよい．このことを示すために論理値 X_1 を用いる．分岐部の後方への再配置を行うときに，分岐の枝にあるフリップフロップの論理値を X_1 から0に変える．このとき，ORゲートの入力にある論理値 X_1 を持つフリップフロップが1つになるため，そのフリップフロップの論理値を X_1 から1に変える．リタイミング後の回路における元の回路の初期状態 $(FF_1, FF_2) = (0, 1)$ と等価な状態は $(FF_1, FF_2) = (0, 1)$ である．図 3-3(c)の回路ではANDゲートの入力へ再配置されたフリップフロップのどちらかが論理値0を持つようにするため，ANDゲートの入力へ再配置したフリップフロップには論理値 X_0 を与えている．分岐部の後方への再配置を行うときには，分岐の枝のフリップフロップの論理値を X_0 から0に変える．これによりANDゲート入力の他方のフリップフロップの論理値をXとできる．この場合も分岐部の後方への再配置を行うことができ，リタイミング後の回路における元の回路の初期状態 $(FF_1, FF_2) = (0, 0)$ と等価な状態は $(0, X)$ である．図 3-3(d)の回路では分岐の枝のフリップフロップの値が0と1である．従って分岐部の後方への再配置は初期状態と等価な状態が失われる可能性があるため，再配置は行わない．

これらの処理により，元の回路の初期状態と等価な状態をリタイミング後の回路で必ず得ることができる．

以下に初期状態を考慮したリタイミングの基本操作を示す．

1）各フリップフロップに初期状態の値を以下のように与える．

・リセットされるフリップフロップ　　論理値0

・セットされるフリップフロップ　　論理値1

・セット／リセットされないフリップフロップ　論理値X

2）リタイミングは以下の4つの基本操作の組合せで行う．

a）ゲート後方への再配置

再配置を行うフリップフロップを FF_j とする．FF_j の値が X_0 であるとき，FF_j の出力側のAND（またはNAND）ゲートの入力に接続しているフリップフロップが FF_j の他に一つしかなければ，そのフリップフロップの値を0とする．同様に，FF_j の値が X_1 である

とき，FF_j の出力側のOR（またはNOR）ゲートの入力に接続しているフリップフロップが FF_j の他に一つしかなければ，そのフリップフロップの値を1とする．この処理を行った後，FF_j をゲート入力へ再配置し，表 3-1に示す値を与える．

表 3-1　ゲート入力へ再配置されたフリップフロップの値

ゲート	FF_j の値が0	FF_j の値が1	FF_j の値がX, X_0, または X_1
AND	X_0	1	X
NAND	1	X_0	X
OR	0	X_1	X
NOR	X_1	0	X
NOT	1	0	X

b）ゲート前方への再配置

ゲート入力のフリップフロップの値に応じたゲートの出力値を，出力へ再配置したフリップフロップの値とする．

c）分岐前方への再配置

分岐のすべての枝へフリップフロップを再配置し，再配置する前のフリップフロップの値をそれらのフリップフロップの値とする．

d）分岐後方への再配置

分岐の各枝のフリップフロップの値を調べ，0と1が少なくとも1つづつ含まれていれば再配置は行わない．論理値0を持つフリップフロップが存在しない場合は，分岐の枝のフリップフロップの論理値をすべて1に変更する．このとき，分岐の枝のフリップフロップが論理値 X_0 を持ち，かつそのフリップフロップの出力側にあるAND（またはNAND）ゲートの入力に接続しているフリップフロップが他に一つしかない場合は，再配置の対象ではない方のフリップフロップの値を X_0 から 0 に変更する．また，分岐の枝のフリップフロップが論理値 X_1 を持っていれば，そのフリップフロップの出力側にあるOR（またはNOR）ゲートの入力に接続している他のフリップフロップの値を X_1 から X に変更する．この処理の後，フリップフロップを分岐の枝から幹へ再配置し，その値を1とする．

同様に，論理値1を持つフリップフロップが存在しない場合は，分岐の枝のフリップフロ

ップの論理値をすべて0に変更する．このとき，分岐の枝のフリップフロップが論理値 X_0 を持っていれば，そのフリップフロップの出力側にある AND（または NAND）ゲートの入力に接続している他のフリップフロップの値を X_0 から X に変更する．また，分岐の枝のフリップフロップが論理値 X_1 を持ち，かつそのフリップフロップの出力側にある OR（またはNOR） ゲートの入力に接続しているフリップフロップが他に一つしかない場合は，再配置の対象ではない方のフリップフロップの値を X_1 から1に変更する．この処理の後，フリップフロップを分岐の枝から幹へ再配置し，その値を0とする．

3．5 初期状態を考慮した冗長除去手法

初期状態の考慮はリタイミングにおいてのみならず，冗長除去に基づく回路簡単化においても必要となる．組合せ回路的検出不能故障があった場合，その信号線を縮退値に固定させて冗長な信号線の除去ができる．フリップフロップの入出力線をそれぞれ疑似外部出力線，疑似外部入力線とみなした組合せ回路で冗長除去を行った場合，疑似外部出力線が固定値をとる場合がある．初期状態を考慮しない場合は，その疑似外部出力線に対応するフリップフロップの出力線も冗長信号線であり，対応するフリップフロップが削除できる[55]．しかし初期状態を考慮するときに，対応するフリップフロップが削除できるか否かは，初期状態によって決まる．

図 3-4(a) において信号線 r の1縮退故障が冗長であるとする．このときフリップフロップの出力に信号値1を固定して得られた回路が図 3-4 (b) の回路である．しかし，この変換が無効になる場合がある．元の回路のフリップフロップがリセットにより初期化される場合，初期状態 (0) における入力値 $(a, b) = (1, 1)$ に対する出力値は $z = 0$ となる．ところが図 3-4 (b) では同じ入力に対する出力は $z = 1$ となる．この場合は，フリップフロップの入力部分の除去のみ可能であるが，フリップフロップそのものとフリップフロップの出力から接続された部分の除去はできなくなる．すなわち，フリップフロップがセット端子により初期化される（フリップフロップの初期値が1）場合はフリップフロップの入出力線はともに冗長であるが，リセット端子により初期化される（初期値が0）場合はフリップフロップの出力線は冗長ではない．

このように，フリップフロップの入力線の固定値が初期値と違う値であれば，フリップフロップの入力線を外部入力線とする．図 3-4 (c) のようにその外部入力線は固定値が常に供給されるよう VDD，あるいは GND 端子とつなぐようにする．こうすることによって簡単化後の回路の同

一動作が保証される．

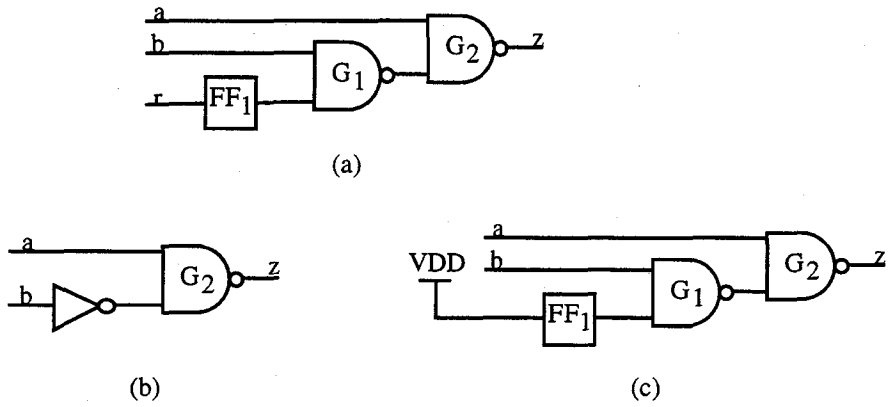


図 3-4　初期状態と冗長除去

3．6　簡単化手順

前節までに述べた初期状態を考慮したリタイミング手法と初期状態を考慮した冗長除去手法を用いた，特定の初期状態が与えられた回路の簡単化手法について述べる．

リタイミングを用いてフリップフロップ数を削減することができる．ここでは，各分岐点において分岐の後方への再配置を用いて，フリップフロップ数を削減する．

図 3-5の例を用いて，分岐の後方への再配置によるフリップフロップ数の削減法について示す．

各分岐点において，次に定義する分岐からフリップフロップまでの距離を計算し，その分岐の後方への再配置により，フリップフロップ数が削減できるかどうかを判断する．

［定義 3-1：分岐からフリップフロップまでの距離］　分岐の枝 i から他の分岐点を通らずにフリップフロップに接続する経路があるとき，一番近いフリップフロップまでの経路上の２入力以上のゲート数に１を加えたものを i からフリップフロップまでの距離 d_i とする．それ以外の場合の距離 $d_i = 0$ とする．

図 3-5の分岐部では，枝 b_5 以外は他の分岐を通らずに接続するフリップフロップが存在する．距離 d_i はそれぞれ，$d_{b1} = 2$，$d_{b2} = 2$，$d_{b3} = 1$，$d_{b4} = 1$，$d_{b5} = 0$，$d_{b6} = 3$ である．この d_i の数は，分岐の枝から一番近いフリップフロップを分岐点まで後方へ再配置する際のフ

リップフロップ数の変化を表している．例えば，図のフリップフロップ f_1 を分岐の枝 b_1 まで後方への再配置を行うと，ゲート G_1 の入力線上にフリップフロップが配置され，この部分におけるフリップフロップ数は1から2 $(=d_{b1})$ になる．このとき，フリップフロップと分岐の枝との間に3入力以上の入力を持つゲートが存在していると，再配置により，フリップフロップ数が d_i よりも多くなるが，図 3-6に示すように入力に2つ以上のフリップフロップを持つゲートについては，ゲートを付加してやることで複数のフリップフロップを1つにすることができる．この処理をゲート付加を用いたゲート前方への再配置とよぶ．この処理をすることで，フリップフロップを分岐点まで再配置したときのフリップフロップ数が，d_i の数に等しくなる．

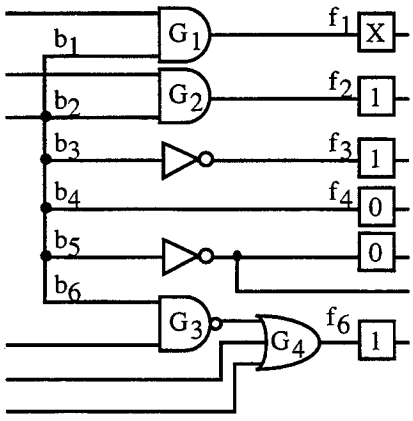


図 3-5　回路例

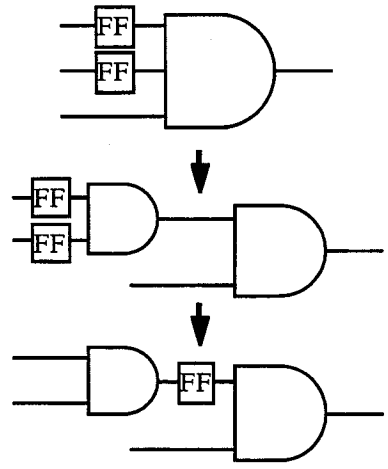


図 3-6　ゲート付加を用いた前方への再配置

分岐点の後方への再配置を行う際には，フリップフロップの初期値を考慮する必要がある．ここでは，分岐の枝に再配置したときの各フリップフロップの値を表 3-1から求める．図においては，フリップフロップ f_1 を分岐の枝 b_1 に再配置するとその値は X となり，同様に f_2 は1，f_3 と f_4 は0，f_6 は X となることが分かる．ここでは，値が0となるフリップフロップと1となるフリップフロップが両方存在しているため，分岐の枝に再配置されたフリップフロップのすべてを分岐の後方へ再配置することはできない．そこで，今値が0となるフリップフロップの数の方が多いので，分岐の後方へ0の値を持つフリップフロップを再配置することを考えると，再配置の対象となるフリップフロップは，f_1, f_3, f_4 および f_6 である．再配置の対象となるフリップフロップの数を n とし，再配置の対象とならない分岐の枝の d_i を0とすると，分岐の後方

への再配置によりフリップフロップ数が増えるかどうかは，次の式 (3.1) により判断できる．

$$1 - n + \sum d_i \leqq n \tag{3.1}$$

右辺は再配置を行う前のフリップフロップ数,左辺は再配置を行った後のフリップフロップ数を示している．この式 (3.1) を満たせば，フリップフロップ数を増やすことなく再配置を行うことができる．図 3-5においては，この式 (3.1) を満たすので，分岐の後方への再配置を行う．このとき，図 3-7のように再配置の対象となる分岐の枝 b_1, b_3, b_4 および b_6 のみを枝とする分岐を作り，分岐後方への再配置を行う．この分岐点での処理を，分岐の2段化を用いた後方への再配置とよぶ．図 3-5の分岐 b_6 へフリップフロップ f_6 を再配置した後の経路の様子を図 3-8に示す．b_i の信号値を0にすると，ゲート G_4 の出力は1に定まるのでゲート G_4 の入力にあるフリップフロップの値は X_1 から X に変更される．このようにして得られる回路を図 3-9に示す．

この図では回路内のフリップフロップ数が6から7に増加しているが,ゲート付加を用いたゲート前方への再配置により，図 3-10に示すようにフリップフロップ数6の回路に変換される．

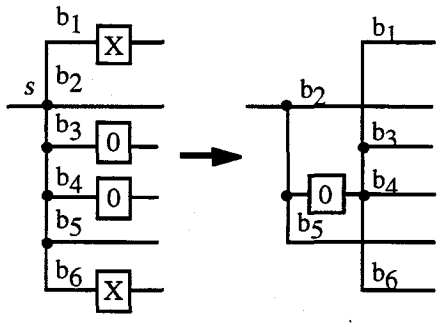


図 3-7　分岐の2段化を用いた再配置

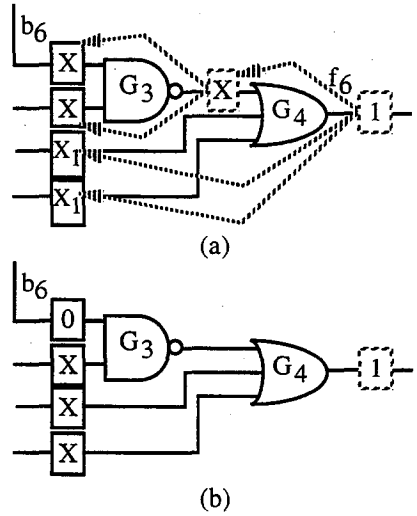


図 3-8　フリップフロップの再配置

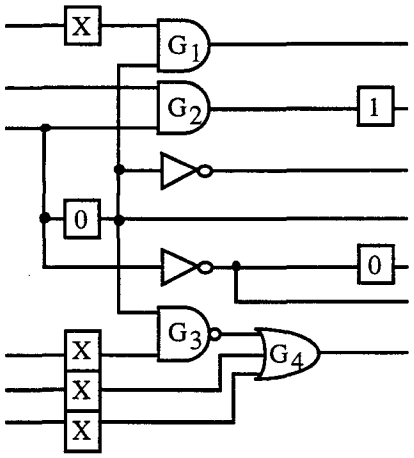


図 3-9　分岐後方への再配置後の回路

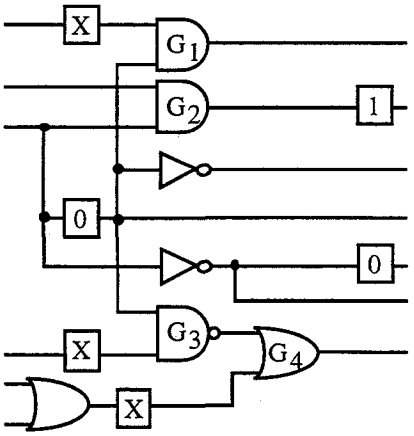


図 3-10　ゲート前方への再配置後の回路

図 3-11に，フリップフロップ数を削減するリタイミングの手順Retiming_RFIの疑似コードを示す．この手順では以下に示す処理が行われる．

［手順 Retiming_RFI (Retiming to Reduce the number of Flip-flops with a specified Initial state)］

Step-rfi0）各フリップフロップに以下のように論理値を与える．

・セット端子により初期化されるフリップフロップ　：論理値0

・リセット端子により初期化されるフリップフロップ：論理値1

・セットおよびリセットされないフリップフロップ　：論理値X

Step-rfi1）入力のすべてにフリップフロップを持つゲートがなくなるまで，ゲート前方への再配置を行う．

Step-rfi2）フリップフロップ数が削減できなくなるまで，各分岐点 s に対し以下の処理を行う．

rfi2-1）分岐の各枝 i について，定義 3-1 に示したフリップフロップまでの距離 d_i を求める．

rfi2-2）$d_i \geqq 1$ となる枝 i において，i からの経路が存在する一番近いフリップフロップ f_i の初期値と等しい値を f_i に与えるとき，i の論理値が一意的に0もしくは1に定まるならば，その論理値を V_i とする．i の論理値が一意的に定まらない場合や，フリップフロップ f_i の初期値がXであるならば，$V_i = \mathrm{X}$ とする．

rfi2-3）$V_i = 0$ となる枝 i の数と $V_i = 1$ となる i の数を比較し，大きい方の V_i の値を V_s とする．そして，V_i が V_s の反転値である d_i の値を0とする．

rfi2-4)　nを $d_i \geqq 1$ となる枝 i の数として式(3.1) ($1 - n + \sum d_i \leqq n$) を計算する．この式を満たさない場合は，他の分岐点 s についてStep-rfi2 を行う．

rfi2-5)　$d_i \geqq 1$ となる枝 i へゲート後方の再配置によりフリップフロップ f_i を再配置する．再配置した各フリップフロップの初期値は，表 3-1を用いて与える．その後，図 3-8の例のように，i の信号値を V_s に設定すると出力値が一意的に定まるゲートの，入力に再配置させたフリップフロップの値をすべてXにする．

rfi2-6) 図 3-7の例のように，$d_i \geqq 1$ となる枝 i をまとめて一つの分岐点とし，分岐後方への再配置を行う．

Step-rfi3) ２つ以上の入力にフリップフロップを持つゲートがなくなるまで，ゲート付加を用いた，ゲート前方への再配置を行う．

この処理を行うことにより初期状態に対応する状態を常に保ったままフリップフロップ数を削減することができる．

手順 Retiming_RFI を組合せ回路の冗長除去手法と組み合わせて，回路簡単化を行った．簡単化の手順を以下に示す．

［簡単化手順］

Step-s0) 初期状態に対応する値を各フリップフロップに与える．

Step-s1) 組合せ部分の冗長除去を行う．

Step-s2) 冗長除去後の回路にリタイミングを行う．（手順 Retiming_RFI)

Step-s3) リタイミング後の回路の組合せ部分の冗長除去を行う．

まず，Step-s1 では，回路の初期状態を考慮した上で，組合せ部分の冗長除去を行う．Step-s2 では，フリップフロップ数の削減を目的に手順 Retiming_RFI を適用する．リタイミングにより，順序回路的な冗長が組合せ回路的な冗長として現れることがあるので，それらを除去するため，Step-s3 でもう一度組合せ部分の冗長除去を行う．

```
Procedure: Retiming_RFI(C, s)
Input: circuit C, initial state s
Output: retimed circuit C'
set initial values for each flip-flops from s                    /* Step-rfi0 */
for each flip-flop f                                             /* Step-rfi1 */
    set g = output of f
    while g is an output of a flip-flop and is an input of a logic gate
        set g to the output of the gate whose input is g
        if gate g has flip-flops at all its inputs, then
            apply Gate_forward_rearrangement(g)
set flg = 1
while flg == 1                                                   /* Step-rfi2 */
    set flg = 0
    for each fanout stem s
        set n = 0, n0 = 0, n1 = 0
        for each fanout branch i of s
            set di = Calculate_distance(i)
            if di ≥ 1, then
                set n = n + 1
                calculate value Vi necessary for setting the initial value to the nearest flip-flop
                if Vi == 0, then set n0 = n0 + 1
                else if Vi == 1, then set n1 = n1 + 1
        if n0 = 0 and n1 = 0, then set Vs = X
        else if n1 > n0 , then
            set Vs = 1, n = n - n0
            set di = 0 whose Vi = 0
        else
            set Vs = 0, n = n - n1
            set di = 0 whose Vi = 1
        if n ≥ 2 and 1 - n + Σ di ≤ n, then
            for each di
                if di ≥ 1, then apply Gate_backward_rearrangement toward branch i
            apply Fanout_backward_rearrangement_with_fanout_partitioning(s)
            set flg = 1
for each flip-flop f                                             /* Step-rfi3 */
    set g = output of f
    while g is an output of a flip-flop but neither fanout stem nor primary output
        set g to the output of the gate whose input is g
        if every fan-in line of g is an output of a flip-flop, then apply Gate_forward_rearrangement(g)
        else if two or more fan-in lines of g are outputs of a flip-flop, then
            apply Gate_forward_rearrangement_with_gate_addition(g)

Procedure: Calculate_distance(i)
Input: fanout branch i
Output: distance di
if there is no flip-flops on the path from i to any other fanout stem or primary output, then set di = 0
else set di = 1 + the number of multiple-input gates on the path from i to the nearest flip-flop
```

図 3-11 手順 Retiming_RFI の疑似コード

3．7　実験結果

本手法をC言語で記述し，富士通 S-4/CL ワークステーション上で実験を行った．使用した回路はISCAS89のベンチマーク回路[56]である．

表 3-2に使用した回路の詳細を示す．LN, GT, FFの欄はそれぞれ回路中の信号線数，ゲート数とフリップフロップ数を示している．回路には多くの連続するNOTゲートや1入力のAND/ORゲートが含まれている．それらを除いた信号線数とゲート数をそれぞれLNR, GTRの欄に示している．以下では本手法による簡単化の効果を見るために，信号線数，ゲート数の比較にはLNR, GTRに示されている数を用いる．

表 3-2　ベンチマーク回路

回路名	LN	GT	LNR	GTR	FF
s5378	5344	2779	3947	1625	179
s9234	9256	5597	5923	2583	228
s13207	13300	7951	8396	3179	669
s15850	15934	9772	10422	4470	597
s38417	38445	22179	27402	12217	1636
s38584	38710	19253	34937	15783	1452

表 3-3から表 3-5に提案手法による簡単化の結果を示す．表 3-3はすべてのフリップフロップが論理値0に初期化される場合，表 3-4はすべてのフリップフロップが論理値1に初期化される場合の結果である．比較のため初期値をどのフリップフロップにも与えない場合（すべてのフリップフロップの初期値をXとした）の結果を表 3-5に示す．各表において，簡単化手順の各手順Step-s1，Step-s2，Step-s3 を適用した後の信号線数，ゲート数，フリップフロップ数をそれぞれStep-s1, Step-s2, Step-s3の欄に示す．削減率のDLNの欄にはベンチマーク回路から信号線がどれだけ削減されたかを表 3-2のLNRの値とStep-s3の欄のLNの値から式DLN = 100 * (LNR - LN)/LNR を用いて計算した値を示している．ゲート，フリップフロップの削減率も同様に計算し，得られた値をそれぞれDGT, DFFの欄に示している．

初期状態を考慮するとリタイミングや冗長除去に制限が加わり，簡単化の能力が低下すると考

えられる．しかし，表 3-5の削減率と表 3-3，表 3-4の削減率の差を平均すると，信号線数，ゲート数，およびフリップフロップ数の削減率の差がそれぞれ 0.6%，0.6%，0.4% 程度と小さく，この実験においては初期状態を考慮しない場合とほぼ同等の簡単化が行われていた．

表 3-3　簡単化の結果（初期状態 0）

回路名	Step-s1			Step-s2			Step-s3			削減率(%)		
	LN	GT	FF	LN	GT	FF	LN	GT	FF	DLN	DGT	DFF
s5378	3845	1582	179	3811	1579	161	3564	1470	160	9.7	9.5	10.6
s9234	5003	2151	228	4967	2151	208	4921	2130	208	16.9	17.5	8.8
s13207	7248	2750	667	7079	2753	554	6815	2661	554	18.8	16.3	17.2
s15850	9255	3949	594	9252	3949	585	9250	3948	585	11.2	11.7	2.0
s38417	27085	12077	1636	26986	12069	1579	26900	12027	1579	1.8	1.6	3.5
s38584	30924	13755	1435	30918	13755	1426	30918	13755	1426	11.5	12.8	1.8

表 3-4　簡単化の結果（初期状態 1）

回路名	Step-s1			Step-s2			Step-s3			削減率(%)		
	LN	GT	FF	LN	GT	FF	LN	GT	FF	DLN	DGT	DFF
s5378	3841	1581	178	3805	1578	160	3548	1465	159	10.1	9.8	11.2
s9234	5003	2151	228	4967	2151	208	4921	2130	208	16.9	17.5	8.8
s13207	7531	2883	668	7364	2886	560	7104	2795	560	15.4	12.1	16.3
s15850	9261	3949	597	9258	3949	588	9256	3948	588	11.2	11.7	1.5
s38417	27085	12077	1636	26987	12069	1579	26901	12027	1579	1.8	1.6	3.5
s38584	31545	14077	1436	31539	14077	1427	31536	14075	1427	9.7	10.8	1.7

表 3-5　簡単化の結果（初期状態X）

回路名	Step-s1			Step-s2			Step-s3			削減率(%)		
	LN	GT	FF	LN	GT	FF	LN	GT	FF	DLN	DGT	DFF
s5378	3819	1574	176	3783	1571	158	3526	1458	157	10.7	10.3	12.3
s9234	5003	2151	228	4967	2151	208	4921	2130	208	16.9	17.5	8.8
s13207	7248	2750	667	7078	2753	554	6814	2661	554	18.8	16.3	17.2
s15850	9255	3949	594	9252	3949	585	9250	3948	585	11.2	11.7	2.0
s38417	27085	12077	1636	26982	12067	1578	26894	12024	1578	1.9	1.6	3.5
s38584	30924	13755	1435	30918	13755	1426	30918	13755	1426	11.5	12.8	1.8

表 3-6に表 3-3から表 3-5までの結果が得られたときの各手順ごとの処理時間を示す．Step-s1の欄は初めの組合せ回路部分の冗長除去に要した時間を示し，Step-s2 の欄はリタイミングに要した時間，Step-s3 の欄はリタイミング後の回路の組合せ部分の冗長除去に要した時間を示している．組合せ回路部分の冗長除去に要する時間(Step-s1)に比べ，リタイミングに要する時間はかなり少ないことが分かる．また，リタイミング後の回路に対する冗長除去に要する時間も，Step-s1で要する時間よりも少なくなっている．これはStep-s1 で組合せ回路部の冗長部分が除去されていることに加えて，Step-s2 においてフリップフロップ数が削減され，組合せ回路部の入力数が少なくなっているためであると考えられる．この結果から，本手法では従来から行われている組合せ回路部分の冗長除去に要する時間の１．１倍から１．７倍の時間で，より多くの冗長部分を削除できることがわかる．

表 3-6　処理時間（sec.）

回路名	初期状態 0				初期状態 1				初期状態 X			
	Step-s1	Step-s2	Step-s3	total	Step-s1	Step-s2	Step-s3	total	Step-s1	Step-s2	Step-s3	total
s5378	87.1	0.3	46	133.6	86.5	0.28	48.5	135.4	89.5	0.32	50.4	140.5
s9234	3320	0.45	1476	4798	3332	0.45	1482	4815	3433	0.45	1532	4966
s13207	4943	1.05	68.1	5626	5372	1.13	672	6046	5079	1.08	590	5671
s15850	3469	0.77	216	3686	3450	0.75	207	3658	3536	0.72	217	3754
s38417	2604	2.38	1607	4216	2604	2.45	1784	4393	2643	2.33	1619	4267
s38584	28924	5.15	1379	30311	29323	5.2	1523	30853	29258	5.03	1381	30646

初期状態の影響について表 3-7に実験結果を示す．これはs5378の回路においてフリップフロップの初期値の一つを1とし，他を0として初期状態を与えたときの結果である．s5378の179個あるフリップフロップのそれぞれに対して実験を行い簡単化の結果の違いを調べた．表中の回路数とは同じ簡単化が行われた回路の数を示している．表の最下段は全体の平均の値を示している．表より全体の約8割の133個では，簡単化で得られた回路のゲート数，信号線数，およびフリップフロップ数がそれぞれ等しくなっている．これより初期状態の値により結果は異なるが，その影響を受ける場合は少ないことがわかる．

表 3-7　s5378の初期状態に依存する結果

回路数	LN	GT	FF	DLN(%)	DGT(%)	DFF(%)
1	3538	1462	157	10.4	10.0	12.3
2	3557	1466	160	9.9	9.8	10.6
2	3560	1470	160	9.8	9.5	10.6
4	3563	1470	160	9.7	9.5	10.6
133	3564	1470	160	9.7	9.5	10.6
1	3565	1470	160	9.7	9.5	10.6
1	3565	1471	160	9.7	9.5	10.6
22	3566	1470	161	9.7	9.5	10.1
1	3566	1472	159	9.7	9.4	11.2
2	3567	1470	161	9.6	9.5	10.1
1	3567	1473	159	9.6	9.4	11.2
1	3571	1473	161	9.5	9.4	10.1
4	3626	1496	161	8.1	7.9	10.1
4	3721	1543	160	5.7	5.0	10.6
179(平均)	3569	1472	160	9.6	9.4	10.5

３．８　あとがき

フリップフロップのセット端子およびリセット端子を用いて初期化される回路に対して，定められた特定の初期状態に等価な状態をリタイミング後の回路が必ず持つように保証したリタイミング手法を提案した．リタイミングの問題点として，分岐点の入力方向へのフリップフロップの再配置により到達不能状態に等価な状態が失われる場合があることが知られている．本手法では初期状態でのフリップフロップの論理値を記憶し，初期状態と等価な状態が失われる場合には分岐点の入力方向への再配置を行わないという制限をリタイミングに加えた．本手法により，初期状態を考慮した上で回路簡単化は考慮しない場合とほぼ同等に行うことができた．さらに各フリップフロップの初期値の相互関係を考慮して処理を行えば簡単化の能力低下を防ぐことができると考えられる．また，提案したリタイミング手法は回路簡単化のみならず，回路高速化などの目的に用いられるリタイミングにも応用が可能である．

第4章 リタイミングによる順序回路的検出不能故障の冗長除去手法

4．1　まえがき

リタイミングを行うことで順序回路的検出不能故障が組合せ回路的検出不能故障に変換される場合があることが知られている[39]. 順序回路的検出不能故障に比べて組合せ回路的検出不能故障は判定が容易であることから,リタイミングで検出不能故障が組合せ回路的検出不能故障として判定できれば，テスト容易化や回路簡単化を効率よく行うことができる．本章では，順序回路的検出不能故障が組合せ回路的検出不能故障に変換される可能性が高くなるようなリタイミング手法として,設定ができない状態である到達不能状態を削除するリタイミング手法を提案する.設定ができないフリップフロップの値の組合せがなくなるようにリタイミングを行うことで,到達不能状態を故障検出に必要とする検出不能故障が組合せ回路的検出不能故障に変換される確率は高くなると考えられる.本手法を適用することで組合せ回路的検出不能故障に変換される故障が多く得られることを，ベンチマーク回路に対する実験結果より示す．

4．2　リタイミングと除去可能な縮退故障

リタイミングによる再合成を行う際に,元の回路で検出可能である故障は変換後においても検出可能であり，検出不能故障は変換後の回路に対しても検出不能であることが知られている[53]. ところが検出不能故障が組合せ回路的検出不能であるか,順序回路的検出不能であるかという特性は，リタイミングにより変化することがある[30,38]. 例として図 4-1の回路を考える．信号線 r の1縮退故障は順序回路的検出不能故障である.この回路に対してゲート G_3 の後方への再配置を行い,分岐部を2つに分けて得られる回路と,この回路の組合せ回路部分を,それぞれ図 4-2,図 4-3に示す．図 4-1の信号線 r に対応する信号線を r' とすると，図 4-3の組合せ回路の信号線 r' の1縮退故障は，故障の影響をゲート G_2 より先に伝播させることができないため，検出不能である．したがって，図 4-2の順序回路の信号線 r' の1縮退故障は組合せ回路的検出不能故障である．

このようにリタイミングにより組合せ回路的冗長に変換される故障があれば,組合せ回路用の冗長除去手法を用いてそれらを除去することが可能になる．

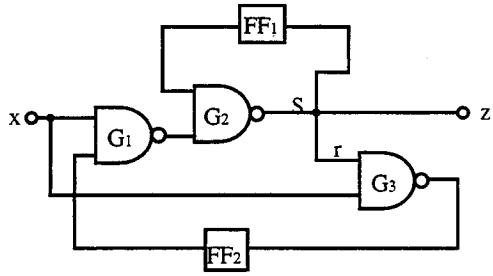


図 4-1　順序回路的検出不能故障の例

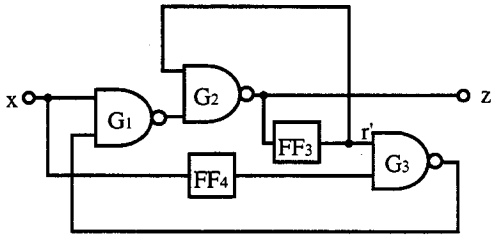


図 4-2　リタイミング後の回路

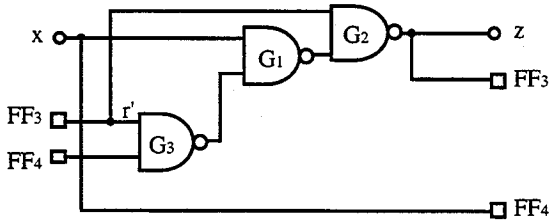


図 4-3　図 4-2の回路の組合せ部分

4．3　到達不能状態の削除

組合せ回路用のテスト生成では，フリップフロップの値はすべて独立に制御できることを仮定している．そのため，組合せ回路部分では検出可能な故障でも順序回路的には検出不能故障となる可能性がある．例えば，組合せ回路のテストパターンで到達不能状態に相当するフリップフロップの値が必要である故障は，順序回路的検出不能故障である．なぜなら，任意の状態から到達不能状態へ遷移させることのできる入力系列は存在しないからである．

3．2節で述べたように到達不能状態のいくつかは分岐部の後方へのフリップフロップの再配置により対応する状態が存在しなくなる．本論文ではこのことを*リタイミングによる到達不能状態の削除*という．もし組合せ回路で検出するのに必要な状態に到達不能状態があり，それがリタ

イミングにより削除されたならば，その故障は組合せ回路的検出不能故障へと変換される．

以下に組合せ回路的検出不能に変換できる故障をなるべく多く変換するためのリタイミング手法について述べる．この手法では元の回路に含まれる一部の到達不能状態の削除を目的とする．

本手法では到達不能状態を見つけるためにある1つの分岐の幹での信号値とフリップフロップの信号値との関係に注目する．ある分岐の幹 s において以下のような記号を用いる．$F(s)$ を s から他の分岐を通らずに接続している経路のあるフリップフロップの集合とする．そのうち，s の信号値が0（1）であるときに一意的に値の定まるフリップフロップの集合を $F_0(s)$（$F_1(s)$）とする．集合 $F_0(s), F_1(s)$ の要素数をそれぞれ $|F_0(s)|, |F_1(s)|$ とすると，次の定理が成り立つ．

[定理 4-1]　ある分岐の幹sにおいて，集合 $F_0(s), F_1(s)$ が次式を満たすならば，少なくとも1つの到達不能状態が存在する．

$$(|F_0(s)| \geq 1) \wedge (|F_1(s)| \geq 1) \wedge (|F_0(s) \cup F_1(s)| \geq 2) \tag{4.1}$$

証明：分岐の幹 s の信号値を0にしたときに一意的に定まるフリップフロップ $f_0 \in F_0(s)$ の入力値を $\alpha \in \{0,1\}$ とする．まず $|F_1(s)| \geq 2$ のときを考える．このとき，集合 $F_1(s)$ 内に $f_1 \neq f_0$ であるフリップフロップ f_1 が存在する．分岐の幹 s の信号値を1にしたときに一意的に定まる f_1 の入力値を $\beta \in \{0,1\}$ とする．フリップフロップ f_0 は $F_0(s)$ に含まれているため，f_0 の入力値が $\bar{\alpha}$ のときにはsの値は1となる．このとき f_1 の入力値は β となり，$(f_0, f_1) = (\bar{\alpha}, \bar{\beta})$ という値の組合せは設定できない．したがって，$(f_0, f_1) = (\bar{\alpha}, \bar{\beta})$ という到達不能状態が存在する．$|F_1(s)| \geq 2$ ではない（つまり $|F_1(s)| = 1$ である）場合は，$|F_0(s) \cup F_1(s)| \geq 2$ から $|F_0(s)| \geq 2$ が成り立っている．この場合も到達不能状態があることが上と同様にして証明される．□

ある分岐点 s において，式(4.1)が満たされれば，$F_0(s)$ と $F_1(s)$ に含まれるフリップフロップを s に配置されるようにリタイミングを行うことで，到達不能状態を削除することができる．

図 4-4に例を示す．図 4-4 (a) の回路には到達不能状態 $(FF_1, FF_2) = (1, 0)$ がある．分岐 s について $F(s) = \{FF_1, FF_2\}$ である．s の値を0にすると FF_2 の値は一意的に1に定まるので，$F_0(s) = \{FF_2\}$ となる．s の値を1にすると FF_1, FF_2 の値がともに一意的に0に定まるので，$F_1(s) = \{FF_1, FF_2\}$ となる．$F_0(s)$ と $F_1(s)$ が式 (4.1) を満たすため，到達不能状態があることがわかる．そこで，$F_0(s)$ と $F_1(s)$ に含まれるフリップフロップを再配置して，s にフリップフロップを配置する．再配置を行ったあとの回路を図 4-4 (b) に示す．図 4-4 (a) の回路の到達不能状態 (1, 0)

はリタイミングにより削除されるため，図 4-4 (b) の回路では対応する状態がない．このリタイミングにより順序回路的検出不能故障であった図 4-4 (a)の回路の信号線 r の 1 縮退故障が，組合せ回路的検出不能故障に変換される．

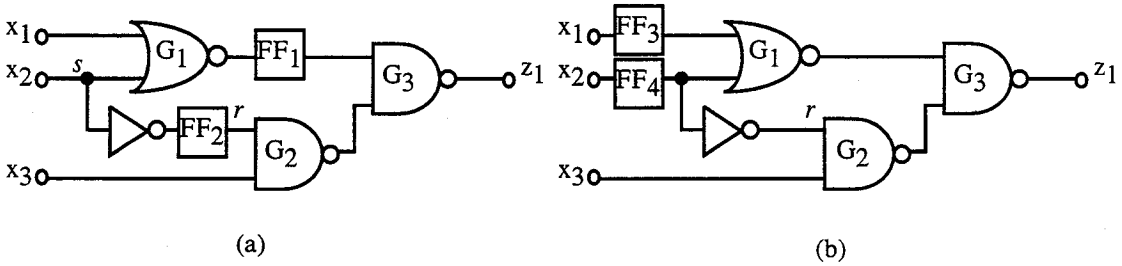


図 4-4　到達不能状態の削除

到達不能状態を削除するためのリタイミングの手順を手順 Retiming_DU とする．手順 Retiming_DU の概略を示す．

［手順 Retiming_DU (Retiming for Deleting Unreachable states) ］

Step-du1）回路内の分岐の幹の集合を S とする．

Step-du2）S から未処理の分岐の幹 s を取り出す．

Step-du3）$F(s)$ を求める．

Step-du4）$F(s)$ 内のフリップフロップのうち，s の値を 0 （1）にして一意的に値の定まるフリップフロップの集合 $F_0(s)$ （ $F_1(s)$ ）を求める．

Step-du5）$F_0(s)$ と $F_1(s)$ が式(4.1)を満たしていれば Step-du6 へ．満たさなければ Step-du8 へ．

Step-du6）$F_0(s)$ と $F_1(s)$ に含まれるフリップフロップをゲートの後方への再配置を繰り返して分岐の枝へ配置する．

Step-du7）フリップフロップのある枝のみが枝となる分岐をつくり，その幹へフリップフロップを再配置する．

Step-du8）$S \neq \phi$ ならば Step-du2 へ．$S = \phi$ ならば終了．

このリタイミングを行うと回路内のフリップフロップ数は増加することがあるが，多くの到達不能状態を削除することができる．それゆえ多くの故障を組合せ回路的検出不能故障に変換することができる．

4．4　フリップフロップ数の削減

リタイミングを行うことによりフリップフロップ数を削減できることがある．図 3-1(a) のように，あるゲートの全入力にフリップフロップが配置されているときは，ゲート後方への再配置を行うことで，フリップフロップ数を削減できる．また，図 3-1 (c) のように，ある分岐についてすべての枝にフリップフロップが配置されているときは，分岐前方への再配置によりフリップフロップ数を削減できる．

ゲート出力のフリップフロップをその入力へ再配置するとフリップフロップ数は増加するが，分岐部でフリップフロップ数が削減されるならばフリップフロップ数の総数は削減できる場合がある．分岐部で効率的にフリップフロップ数を削減するため，分岐の枝 i からフリップフロップまでの距離 d_i を次のように定義する．

［定義 4-1：分岐からフリップフロップまでの距離］　分岐の枝 i から他の分岐点を通らずにフリップフロップに接続する経路があるとき，一番近いフリップフロップまでの経路上の 2 入力以上のゲート数に 1 を加えたものを i からフリップフロップまでの距離 d_i とする．それ以外の場合の距離 $d_i=0$ とする．

ある分岐点において各枝での距離を求めることで，後方への再配置を行ってフリップフロップ数の削減が可能かどうかを判定できる．$d_i>0$ となる i の数を n とすると，後方への再配置を繰り返すことで，n 個のフリップフロップを分岐の枝に配置できる．この再配置により，フリップフロップは (Σd_i) 個に増加する．その後，分岐部の後方への再配置を行うことで，$(n-1)$ 個のフリップフロップが削減できる．したがって n 個のフリップフロップが，リタイミングにより $(1-n+\Sigma d_i)$ 個となる．よってこの数の比較を行い，フリップフロップ数が増加しないときにはリタイミングを行うことにすればよい．

図 4-5に例を示す．図 4-5 (a) の回路の分岐 s の各枝での距離を求めると，枝 b_1 からフリップフロップへの経路上には 2 入力 NOR ゲートがあるので距離 $d_{b1}=2$，枝 b_2 と b_3, b_4 ではフリップフロップまでの経路上に 2 入力以上のゲートはないので $d_{b2}=d_{b3}=d_{b4}=1$ である．枝 b_5 からは別の分岐点までにフリップフロップがないので $d_{b5}=0$ である．よって $n=4$．このとき，$1-n+\Sigma d_i=2 \leqq n$ であるので，リタイミングによりフリップフロップ数が増加しないことがわかる．リタイミングを行った後の回路が図 4-5 (b) の回路である．このリタイミングによりフ

リップフロップ数は5から3に削減される.

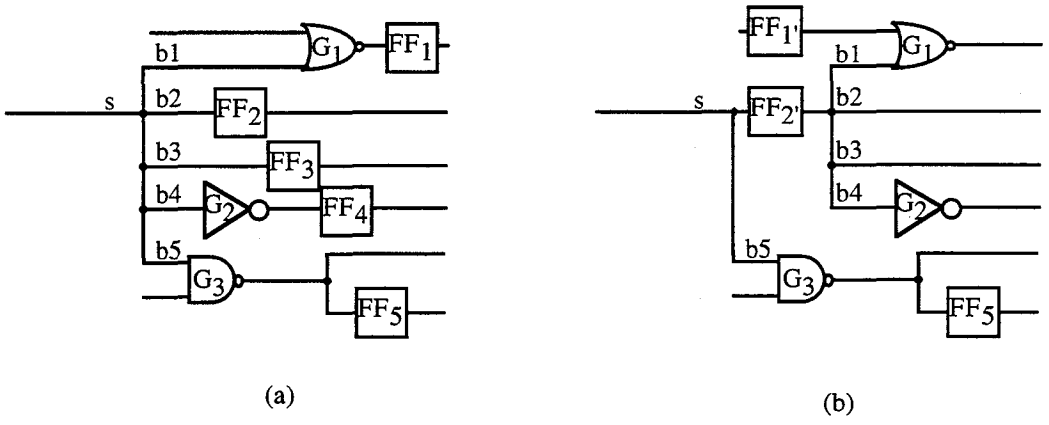


図 4-5　フリップフロップ数を削減するリタイミングの例

フリップフロップ数の総数を削減するためのリタイミングの手順を，手順 Retiming_RF とする．手順 Retiming_RF の概略を以下に示す．また図 4-6に，この手順の疑似コードを示す．

［手順 Retiming_RF(Retiming for Reducing the number of Flip-flops) ］

Step-rf1）入力のすべてにフリップフロップを持つゲートがなくなるまで，ゲート前方への再配置を行う.

Step-rf2）各分岐点において，分岐の各枝の距離 d_i を計算し，$1-n+\sum d_i \leqq n$ を満たしていれば，その分岐点への後方への再配置を行う.

Step-rf3）2つ以上の入力にフリップフロップを持つゲートがなくなるまで，ゲート付加を用いた，ゲート前方への再配置を行う.

この手順においては，Step-rf3 においてゲート付加による再配置を行うため，ゲート数は増える場合がある．しかしながら，ゲート付加による再配置では，ゲートが1つ増えたときには，フリップフロップが1つ以上削減されているため，ゲート付加による再配置が回路簡単化の能力を低下させることはない.

```
Procedure: Retiming_RF(C)
Input: circuit C
Output: retimed circuit C'

for each flip-flop f                                              /* Step-rf1 */
    set g = output of f
    while g is an output of a flip-flop and is an input of a logic gate
        set g to the output of the gate whose input is g
        if gate g has flip-flops at all its inputs, then
            apply Gate_forward_rearrangement(g)
set flg = 1
while flg = 1                                                     /* Step-rf2 */
    set flg = 0
    for each fanout stem s
        set n = 0
        for each fanout branch i of s
            set d_i = Calculate_distance(i)
            if d_i ≥ 1, then set n = n + 1
        if n ≥ 2 and 1 - n + Σ d_i ≤ n, then
            for each d_i
                if d_i ≥ 1, then apply Gate_backward_rearrangement toward branch i
            apply Fanout_backward_rearrangement_with_fanout_partitioning(s)
            set flg = 1
for each flip-flop f                                              /* Step-rf3 */
    set g = output of f
    while g is an output of a flip-flop but neither fanout stem nor primary output
        set g to the output of the gate whose input is g
        if every fan-in line of g is an output of a flip-flop, then
            apply Gate_forward_rearrangement(g)
        else if two or more fan-in lines of g are outputs of a flip-flop, then
            apply Gate_forward_rearrangement_with_gate_addition(g)

Procedure: Calculate_distance(i)
Input: fanout branch i
Output: distance d_i

if there is no flip-flops on the path from i to any other fanout stem or primary output, then
    set d_i = 0
else
    set d_i = 1 + the number of multiple-input gates on the path from i to the nearest flip-flop
```

図 4-6　手順 Retiming_RF の疑似コード

4．5　簡単化手順

4．3節と4．4節で述べたリタイミングの手順を組合せ回路用の冗長除去手法と組み合わせ，以下のように順序回路の簡単化に応用した．

処理の手順を以下に示す．また図 4-7にこの手順の疑似コードを示す．ここでは以下のような処理が行われる．なお，不要なゲートを取り除くために，1入力の AND および OR ゲートと連続する2つの NOT ゲートを各処理の前後に除去している．

［簡単化手順 Synthesis_RR(Synthesis of sequential circuits by Redundancy removal using Retiming)］

Step-s1）回路の組合せ部分の冗長除去を行う．

Step-s2）回路内のフリップフロップの位置を記憶し，到達不能状態の削除のためのリタイミングを行う．（手順 Retiming_DU）

Step-s3）組合せ部分の冗長除去を行う．

Step-s4）2で記憶した位置へフリップフロップを戻し，フリップフロップ数の削減のためのリタイミングを行う．（手順 Retiming_RF）

Step-s5）組合せ部分の冗長除去を行う．

まず Step-s1 では，与えられた回路の組合せ回路的検出不能故障を除去する．組合せ回路的検出不能故障の中に，リタイミングにより順序回路的検出不能故障に変換されてしまい，検出不能故障の判定ができなくなる故障があるためである．その後，Step-s2 では，多くの故障を組合せ回路的検出不能故障に変換するために，4．3節で述べたリタイミング手順 Retiming_DU を適用する．このリタイミングにより変換された組合せ回路的検出不能故障に対応する信号線の除去を Step-s3 で行う．Step-s2 では，一般にフリップフロップ数が増加するため，このリタイミングを行う前にフリップフロップの位置を記憶し，冗長除去後に元の位置へと戻す．更に Step-s4 では，フリップフロップ数を削減するために，4．4節で述べたリタイミング手順 Retiming_RF を行う．このリタイミングによっても新たに組合せ回路的検出不能故障となる故障がありうるため，最後にもう一度 Step-s5 において組合せ回路部分の冗長除去を行う．

```
Procedure: Synthesis_RR(C)
Input : circuit C
Output : modified circuit C'

set C' = Remove_unnecessary_inverter_chains_and_buffers(C)   /* Step-s1 */
set C' = Combinational_redundancy_removal(C')
mark the position of flip-flops                              /* Step-s2 */
set C' = Retiming_DU(C')
set C' = Remove_unnecessary_inverter_chains_and_buffers(C')
set C' = Combinational_redundancy_removal(C')                /* Step-s3 */
replace all flip-flops to its original position              /* Step-s4 */
set C' = Retiming_RF(C')
set C' = Remove_unnecessary_inverter_chains_and_buffers(C')
set C' = Combinational_redundancy_removal(C')                /* Step-s5 */
```

図 4-7　手順 Synthesis_RR の疑似コード

4．6　実験結果

提案手法をC言語で記述し，富士通 S-4/CL ワークステーション上で実験を行った．対象回路はISCAS89ベンチマーク回路[56]である．本手法のリタイミングにより組合せ回路的検出不能故障に変換された順序回路的検出不能故障のあった7つの回路に対しての結果を表に示す．表4-1のORIGINALの下のGT，FFの欄はベンチマーク回路のゲート数(GT)，フリップフロップ数(FF)をそれぞれ示している．REDUCEDの欄にはベンチマーク回路から連続する2つのNOTゲート，1入力のAND（またはOR）ゲートを除去した後のゲート数とフリップフロップ数を示している．表 4-2は手順 Synthesis_RR の各ステップごとの結果を示す．Step-s1，Step-s2&s3，Step-s4&s5 の各欄はそれぞれ初めの冗長除去，手順 Retiming_DU を行った後の冗長除去，手順 Retiming_RF を行った後の最後の冗長除去を行った後で得られた回路のゲート数(GT)・フリップフロップ数(FF)・除去された故障数(RF)を示している．

これらの結果から本手法により，平均30個程度の順序回路的検出不能故障が組合せ回路的検出不能故障に変換され，それらを除去したことで，ゲート数とフリップフロップ数が元の回路の組合せ回路の冗長除去を行った後にも削減されていることがわかる．

表 4-1 ベンチマーク回路のゲート数とフリップフロップ数

	ORIGINAL		REDUCED	
回路名	GT	FF	GT	FF
s386	133	6	133	6
s5378	2779	179	1625	179
s9234	5597	228	2583	228
s13207	7951	669	3179	669
s15850	9772	597	4470	597
s38417	22179	1636	12217	1636
s38584	19253	1452	15783	1452

表 4-2 各ステップごとの結果

	Step-s1			Step-s2&s3			Step-s4&s5		
回路名	GT	FF	RF	GT	FF	RF	GT	FF	RF
s386	133	6	0	138	25	12	130	6	0
s5378	1596	176	29	1633	198	33	1459	158	0
s9234	2387	228	258	2592	826	1	2151	200	16
s13207	3014	667	412	3483	1152	111	2665	533	0
s15850	4305	594	354	4808	1685	0	3951	579	1
s38417	12176	1636	137	13450	3446	29	12013	1578	0
s38584	14298	1435	455	15629	3580	16	13758	1426	0

表 4-3に各ステップの実行に要した処理時間を示す. 組合せ回路用のテスト生成を用いた冗長除去（Step-s1, Step-s3, Step-s5）が処理の多くの時間を占めている. リタイミングに要する時間（Step-s2, Step-s4）は，テスト生成に比べてかなり短いことがわかる. s386 の最後の冗長除去は，2回目のリタイミングで回路が変えられなかったため行っていない. 本手法ではテスト生成を用いた冗長除去を3回適用しているが，一度目の冗長除去に要する時間（Step-s1）の平均1.7倍程度の時間で処理ができることから, 大規模回路に対しても十分適用可能であると考えられる.

表 4-4は文献[57]での手法による結果と本手法での結果との比較を示す．文献[57]ではベンチマーク回路を修正して用いているものがあり，元のベンチマーク回路よりフリップフロップ数が少ない回路で実験を行っている．それらの回路については回路名に＊印を付けている．s38417 に対してはゲート数，信号線数，フリップフロップ数ともに本手法のほうが多く削減されている．また，s386 ではフリップフロップ数は変化していないが，ゲート数および信号線数は多く削減されていることが分かる．

表 4-3　処理時間（sec.）

回路名	Step-s1	Step-s2	Step-s3	Step-s4	Step-s5	total
s386	1.96	0.05	3.18	0.05		5.28
s5378	90.6	0.42	46.3	0.88	39.6	179
s9234	2900	0.93	94.3	0.58	1539	4534
s13207	4812	2.37	519	5.65	150	5491
s15850	3285	3.80	227	7.13	209	3736
s38417	2555	15.62	1570	37.42	1484	5701
s38584	28754	58.04	1904	89.88	1370	32195

表 4-4　文献[57]との比較

	本手法			文献[57]		
回路名	GT	LN	FF	GT	FF	FF
s386	130	352	6	159	359	6
s5378	1459	3528	158	1363	3096	119
s9234*	2151	4943	200	2644	5174	198
s13207*	2665	6803	533	2166	4227	318
s15850*	3951	9247	579	5023	10082	466
s38417	12013	26883	1578	18399	33692	1620
s38584	13758	30905	1426	N/A	N/A	N/A

4．7　あとがき

本章ではゲート数やフリップフロップ数の削減による簡単化を行うリタイミングと冗長除去を併用した手法を提案した．本手法においては2つの異なるリタイミングの手順を用いた．1つは到達不能状態を削除するために，分岐点の信号値とフリップフロップの入力値との関係からリタイミングにより削除可能な到達不能状態を調べ，分岐後方への再配置を行うリタイミング手順であり，もう1つは，フリップフロップ数を削減するために，分岐点においてリタイミングによるフリップフロップ数の増減を計算し，分岐点の後方への再配置を行うリタイミングの手順である．初めのリタイミングにより順序回路的検出不能故障が組合せ回路的検出不能故障へと変換され，その除去により回路内のゲート数とフリップフロップ数を削減することができた．しかし，リタイミングを行っても組合せ回路的検出不能故障にならない順序回路的検出不能故障は存在する．それらの故障について考察し，判定を行うことが今後の課題として残されている．

第5章　到達不能状態に基づく冗長除去手法

5．1　まえがき

前章においてリタイミングにより順序回路的検出不能故障が組合せ回路的検出不能故障に変換される原因の一つは到達不能状態にあることを考察した.到達不能状態の中にはリタイミングにより削除できないものも多く存在する.検出不能故障を求める手法として設定できない信号値の組合せからそれらを故障検出に必要とする検出不能故障を求める手法が提案されている[58].また文献[59]では設定できない信号値の組合せとして無効状態の値を与え，検出不能故障を求めている．この手法は大規模回路にも適用可能であるが，順序回路的検出不能故障の除去可能性についてまでは考察していない.そこで本章では到達不能状態の情報を用いて得られた検出不能故障について除去可能性を判定し，冗長除去を行う提案手法[60]について述べる．

5．2　到達不能状態と除去可能な検出不能故障

状態の中には，回路の起動時にしか現れない状態がありうるが，そのような状態を到達不能状態という．到達不能状態における出力と状態遷移のみが正常回路と異なり，他の状態における出力と状態遷移は正常回路と等しい縮退故障は検出不能故障である．

［補題 5-1］　正常回路 M_G の到達不能状態の集合を U_G とする．U_G のある部分集合 U' について，故障 f が存在すると仮定した故障回路 M_F と正常回路 M_G が式(5.1)を満たすならば，故障 f は検出不能故障である．

$$\forall x \in X, \forall s \in \overline{U'}, \left(\delta_F(x,s) = \delta_G(x,s)\right) \wedge \left(\lambda_F(x,s) = \lambda_G(x,s)\right) \tag{5.1}$$

証明：式 (5.1)より到達不能状態以外の状態における M_F と M_G の出力と状態遷移はすべて等しい．従って，ある到達不能ではない状態 s に対して $\lambda_G^+(x_T, s) \neq \lambda_F^+(x_T, s)$ を満たす $x_T \in X^+$ は存在しない．よって定義 2-9 より故障 f は検出不能故障である．□

例 5-1：図 5-1に到達不能状態から得られる検出不能故障の例を示す．図 5-1の回路の状態遷移図を図 5-2(a) に示す．この状態遷移図より状態 (0, 0) は到達不能状態であることが分かる．この回路の3つの縮退故障，信号線 a の1縮退故障（a/1 と表す），故障 b/1 ，故障 c/0 の故障回路の状態遷移図を図 5-2 (b)～(d) に示す．これらの故障はいずれも状態 (0, 0) においてのみ，出力または状態遷移が正常回路と異なっている．従って，補題5-1 よりこれらの故障は検出不能

である．

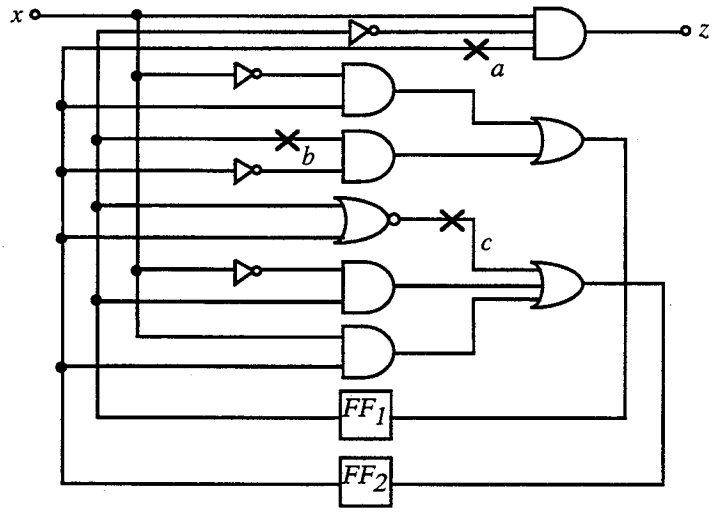


図 5-1　到達不能状態をもつ回路例

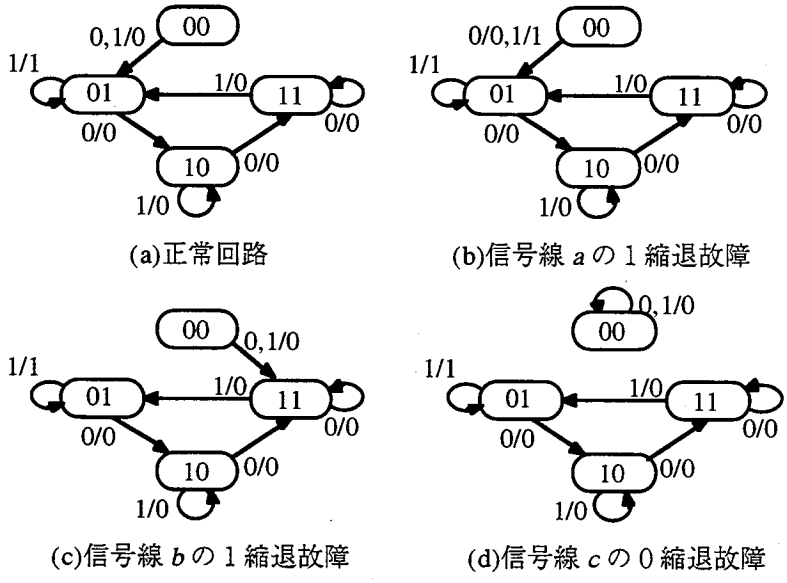


図 5-2　図 5-1の回路の状態遷移図

次にこれらの故障の除去可能性を考える．正常回路は入力系列 $x = 000$ により状態 (1, 1) に同期化可能である． $a/1$ と $b/1$ の故障回路は正常回路と同様に入力系列 $x = 000$ により (1, 1) へ同期化される．また，故障回路の状態 (1, 1) と正常回路の状態 (1, 1) は等価である．したがって故障回路と正常回路は弱等価であり，これらの故障は除去可能である．ところが，$c/0$ の故

障回路は状態 (0, 0) ではいかなる入力系列に対しても状態 (0, 0) のままであり，出力は常に1となり，正常回路では得られない出力系列が現れる．従って，故障 $c/0$ は除去可能ではない．❑

到達不能状態から得られる検出不能故障の除去可能性は，次の定理を用いて調べることができる．

［定理 5-1］正常回路 M_G と故障回路 M_F の到達不能状態の集合をそれぞれ U_G, U_F とする．U_G の部分集合 U' が式(5.1)を満たし，かつ $U' \subseteq U_F$ であるならば，故障 f は除去可能である．

証明：M_G の弱同期化系列を w とすると，M_G の任意の状態 s に対して $\delta_G^+(w, s) \sim s_r$ となる状態 s_r が存在する．M_F の状態 $s_f \notin U'$ における状態遷移は M_G と等しいので，$\delta_F^+(w, s_f) \sim s_r$ が成り立つ．そこで状態 $s_u \in U'$ における状態遷移について考える．

状態 s_u についても $\delta_F^+(w, s_u) \sim s_r$ が成り立つ場合は，M_G と M_F のすべての状態対 (s_g, s_f) において $\delta_G^+(w, s_g) \sim \delta_F^+(w, s_f) \sim s_r$ を満たす．この場合は M_F は w で弱同期化可能で，かつ M_F の w 印加後の出力系列が M_G の w 印加後の出力系列と等しくなり，除去可能である．

また，s_u が $\delta_F^+(w, s_u) \sim s_r$ を満たさない場合は，$s_u \in U' \subseteq U_F$ より任意の入力 $i \in X$ について $\delta_F(i, s_u) \neq s_u$ が成り立ち，よって $\delta_F^+(w, \delta_F(i, s_u)) \sim s_r$ が成り立つ．従って S_G と S_F のすべての状態対 (s_g, s_f) において $\delta_G^+(w, \delta_G(i, s_g)) \sim \delta_F^+(w, \delta_F(i, s_f)) \sim s_r$ であるから，入力 i を w の前に加えた系列を w' とすると $\delta_G^+(w, s_g) \sim \delta_F^+(w', s_f) \sim s_r$ となりこの場合も M_F は弱同期化可能である．また M_F の w' 印加後の出力系列が M_Gの w 印加後の出力系列と等しくなるため，故障 f は除去可能である．❑

冗長除去を行うことで，故障の除去可能性が変化することがある．例えば，図 5-1の回路から $a/1$, $b/1$ に基づき冗長除去を行ったとする．除去後の回路と状態遷移図を図 5-3に示す．この回路で故障 $c/0$ の影響を見ると状態遷移は図 5-3(c)に示すようになり，到達不能状態 (0, 0) は故障回路においても到達不能である．従って，図 5-1の回路では除去可能ではなかった $c/0$ が $a/1$, $b/1$ の冗長除去により除去可能になる．

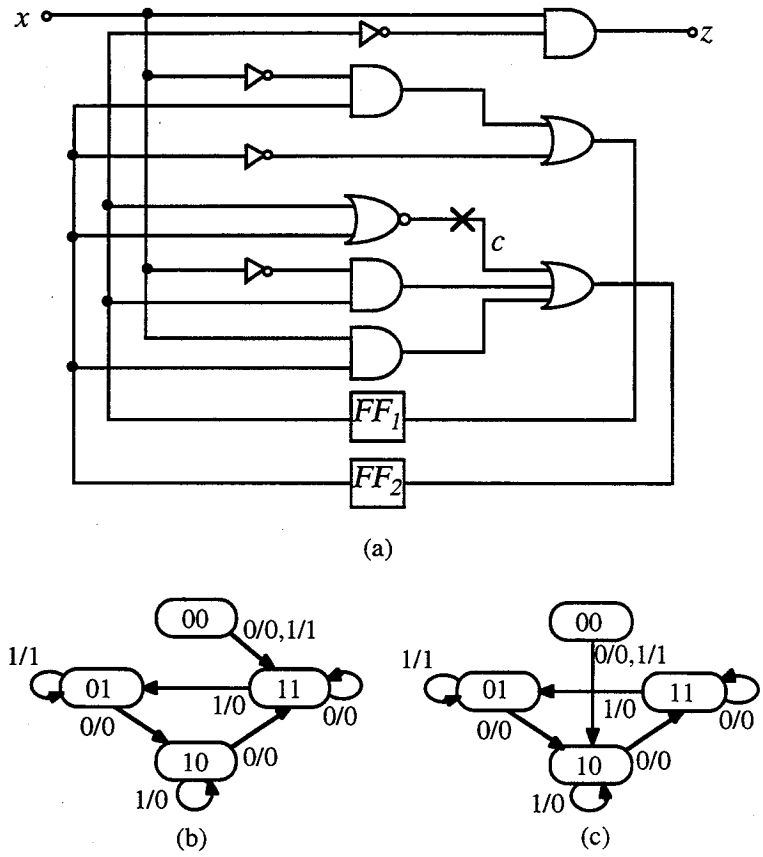


図 5-3　冗長除去により除去可能性が変化する故障の例

(a) 信号線 a, b を除去した回路

(b) (a)の回路の状態遷移図　(c) 故障 $c/0$ の故障回路の状態遷移図

5．3　到達不能状態の探索法

定理5-1に基づいて，一部の検出不能故障の除去可能性は全状態を考慮しなくても到達不能状態に着目することで容易に判定できる．到達不能状態を状態遷移関数により求める方法が提案されているが[61]，フリップフロップ数の多い回路に対しては実用的ではない．本章ではゲートレベル記述から到達不能状態を求める方法を提案する．まず，本手法における到達不能状態の探索について述べる．

回路のフリップフロップの入力（出力）線を疑似外部出力（入力）線とした組合せ回路を用いて到達不能状態を調べる．i 番目のフリップフロップの入力（出力）線を $Q_i(q_i)$ とし，状態 s の i 番目のフリップフロップの値を $s_{(i)}$ で表し，Q_i の論理関数を $\delta_{(i)}$ とする．また，i 番目のフリップフロップの入力値が $\alpha \in \{0, 1\}$ となるすべての $(x, s) \in X \times S$ の集合を $D_{(i)}(\alpha)$ とする，

すなわち $D_{(i)}(\alpha) = \{(x, s) \mid \delta_{(i)}(x, s) = \alpha\}$.

［定理 5-2］　$(..., s_{(i)}, ..., s_{(j1)}, ..., s_{(j2)}, ..., s_{(jp)}, ...) = (..., \alpha, ..., \beta_1, ..., \beta_2, ..., \beta_p, ...)$ なる状態 s が式(5.2)を満たすならば，状態 s は到達不能である．

$$D_{(i)}(\alpha) \subseteq \left(D_{(j_1)}(\overline{\beta}_1) \cup D_{(j_2)}(\overline{\beta}_2) \cup ... \cup D_{(j_p)}(\overline{\beta}_p) \right) \tag{5.2}$$

証明：式(5.2)より

$$D_{(i)}(\alpha) \cap \overline{\left(D_{(j_1)}(\overline{\beta}_1) \cup D_{(j_2)}(\overline{\beta}_2) \cup ... \cup D_{(j_p)}(\overline{\beta}_p) \right)} = \phi \tag{5.3}$$

従って

$$D_{(i)}(\alpha) \cap D_{(j_1)}(\beta_1) \cap D_{(j_2)}(\beta_2) \cap ... \cap D_{(j_p)}(\beta_p) = \phi \tag{5.4}$$

この式より，$\delta_{(i)}(x, s) = \alpha$ かつ $\delta_{(jk)}(x, s) = \beta_k$ $(1 \le k \le p)$ のすべてを満たす (x, s) は存在しない．従って，$(..., s_{(i)}, ..., s_{(j1)}, ..., s_{(j2)}, ..., s_{(jp)}, ...) = (..., \alpha, ..., \beta_1, ..., \beta_2, ..., \beta_p, ...)$ で表される状態は到達不能である．❑

例5-2：図 5-4に簡単な回路例を示す．図の部分回路には3つのフリップフロップ FF_1, FF_2, FF_3 がある．各フリップフロップの入力線をそれぞれ Q_1, Q_2, Q_3 とする．

$v[Q_2] = 1$ のとき，必ず $v[Q_1] = 1$ であるから，$D_{(2)}(1) \subseteq D_{(1)}(1)$ が成り立つ．よって定理5-2より状態ベクトル (0, 1, X) で表される到達不能状態が得られる．

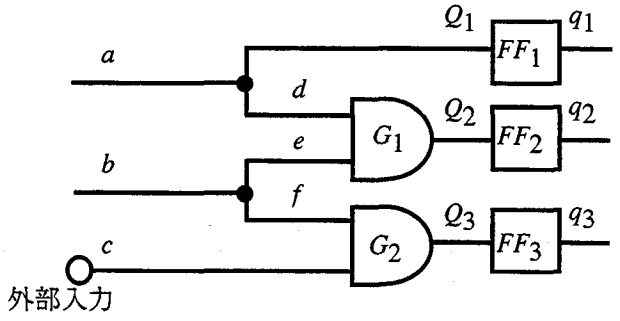


図 5-4　到達不能状態の例

次に，$v[Q_2] = 0$ の場合を考える．このとき，ゲート G_1 は入力が一意には決まらない．このように，出力値が定まっているが入力値の定まっていないゲートのことを未充足ゲートと呼ぶ．

ここで未充足ゲート G_1 の取りうる入力値として，$v[d]=0$ と $v[e]=0$ の2通りの入力値を考えると，$v[d]=0$ ならば $v[Q_1]=0$ となり，$v[e]=0$ ならば $v[Q_3]=0$ となる．よって，$D_{(2)}(0) \subseteq (D_{(1)}(0) \cup D_{(3)}(0))$ の関係が得られるので，定理2より (1, 0, 1) も到達不能状態である．□

このようにして1つのフリップフロップの入力値から到達不能状態を調べることができる．しかし，すべての未充足ゲートについてゲート入力値を割り当てることは無駄が多い．例えば，未充足ゲートの入力に他のフリップフロップと外部入力を共有しない信号線があれば，入力値を割り当てても他のフリップフロップの値は決まらず，到達不能状態は見つからない．

そこで，前処理として未充足ゲートの入力値割り当ての対象となるゲートを次のように定める．回路の各分岐点について，その分岐点から他のフリップフロップを通らない経路の存在するフリップフロップが2つ以上あるものを調べ，印をつけておく．そして，ゲートの入力線の中に，外部入力線からのどの経路上にも印のついた分岐点が存在していない入力線があれば，そのゲートを入力値割り当ての対象から除く．なぜなら，この条件を満たす入力線に値を割り当てても，他のフリップフロップの値には関係がなく，到達不能状態は得られないためである．

この処理を図 5-4の回路に対して行うとゲート G_2 は対象から除かれる．この前処理により無駄な入力値割り当てを減らすことができる．

未充足ゲートに入力値を割り当てることで，他のゲートが新たに未充足ゲートとなる場合がある．ゲート数の多い回路では，前処理により対象ゲート数を減らしても，未充足ゲートに対する入力割り当てにおいて，次々に未充足ゲートが現れ，処理時間が非常に長くなる．この場合には入力値割り当ての対象を，始めに現れた未充足ゲートだけに限定し，未充足ゲートに対して入力値の割り当てを行ったときに現れる未充足ゲートには入力値を割り当てないことにする．

到達不能状態を求める手順を手順 Find_URS とする．手順 Find_URS の疑似コードを図 5-5に示す．この手順では，到達不能状態を以下のようにして求めている．

信号線 a の信号値を $v[a]$ と表す．まず $v[Q_i]=\alpha$ を与え，一意的に定まる信号値を求める．このとき，他のフリップフロップの入力値 $v[Q_j]$ が β に定まるとすると，$D_{(i)}(\alpha) \subseteq D_{(j)}(\beta)$ が成り立ち，$s_{(i)}=\alpha$ かつ $s_{(j)}=\overline{\beta}$ なる到達不能状態 s が得られる（手順 f_urs_line）．

さらにゲートの出力値が定まっているが入力値の定まっていない未充足ゲートに対する処理を行う．ある未充足ゲート G の入力線を $g_k\ (1 \le k \le p)$ とすると，$v[g_k]=c$ （c はゲートの制御値で，AND, NAND ゲートは0，OR, NOR ゲートは1）を与えたときに値が一意的に $\beta_k \in \{0,$

1} に定まるフリップフロップの入力 Q_{jk} が各 g_k に対して存在するならば，$D_{(i)}(\alpha) \subseteq (D_{(j1)}(\beta_1) \cup D_{(j2)}(\beta_2) \cup ... \cup D_{(jp)}(\beta_p))$ が成り立ち，$(..., s_{(i)}, ..., s_{(j1)}, ..., s_{(j2)}, ..., s_{(jp)}, ...) = (..., \alpha, ..., \overline{\beta}_1, ..., \overline{\beta}_2, ..., \overline{\beta}_p, ...)$ なる到達不能状態が得られる（手順 f_urs_gate）．

ゲート数が多く処理に時間がかかる場合には，手順 Find_URS 内の *MODE* の値を *SKIP_MODE* とし，初めに f_urs_line が適用されるときに現れる未充足ゲートのみに入力割り当ての処理を行うように，未充足ゲートの処理を制限する．この制限により，到達不能状態の一部しか得られない可能性があるが，全ての到達不能状態が得られなくても，求められた到達不能状態に対して次の除去可能故障の判定法は適用できる．

Procedure: Find_URS(Q_i, α)
Input: Input line of i-th flip-flop Q_i
Input: logic value α
Output: set of unreachable state vectors U
set s be a state vector such that $s_{(i)} = \alpha$ and $s_{(j \neq i)}$ = don't care
U = f_urs_line(Q_i, α, s, 0)

Procedure: f_urs_line(*lin*, *val*, *s*, *mode*)
Input: line *lin*
Input: logic value *val*
Input: state vector *s*
Input: option *mode*
Output: set of state vector *U*
set $U = \phi$
for each Q_j
　　if v[Q_j] is determined to value β by setting v[*lin*] = *val*, then
　　　　set $s' = s$, $s'(j) = \beta$, and add s' to U
if *mode* ≠ *SKIP_MODE*
　　for each unjustified gate G
　　　　set U' = f_urs_gate(G, s), and add U' to U

Procedure: f_urs_gate(G, s)
Input: gate G
Input: state vector s
Output set of state vector U
set $U' = \{s\}$
set c = controlling value of G
for each input g_k of G
　　set U_k = f_urs_lin(g_k, c, s, *MODE*)
　　if $U_k = \phi$, then set $U' = \phi$
　　else set $U' = \{s_u | s_u = (s_0 \cap s_1), s_0 \in U', s_1 \in U_k\}$

図 5-5　到達不能状態探索アルゴリズム

5．4 除去可能な検出不能故障の判定法

前節で述べたアルゴリズムにより得られた到達不能状態を用いた,除去可能故障の判定法について述べる.

まず,回路の状態を到達不能状態に設定しなければ故障検出ができない故障を求める.それらの故障については,故障回路の到達不能状態以外の状態における出力系列が正常回路と等しくなるため,定理5-1を用いて除去可能故障の判定ができる.ここでは文献[59]に基づき次の判定法を用いる.

得られた到達不能状態はベクトルで表される.そのうちの1つを s_u とし,そのi番目の要素を $s_{u(i)}$ で表す.$s_{u(i)} \in \{0, 1\}$ である各 i について,$v[q_i] = s_{u(i)}$ を設定しなければ故障検出ができない故障の集合 F_i を求めると,状態を s_u に設定しなければ故障検出ができない故障は F_i の積集合で求められる.

$v[q_i] = s_{u(i)}$ を設定しなければ故障検出ができない故障の集合 F_i は以下のように求められる.

まず,$v[q_i] = \overline{s}_{u(i)}$ を与え一意的に定まる信号値を求める.このとき信号線 a の値が一意的にβに定まるならば,故障 a/β の検出には $v[q_i] = s_{u(i)}$ が必要である.従ってこのような故障 a/β はすべて F_i に含まれる.次に $v[q_i] = \overline{s}_{u(i)}$ により出力値が定まったゲートについて,ゲート入力線から外部入力か分岐の枝までの入力部分に存在する値の定まっていない信号線を求める.ここで求められた信号線の0縮退故障,1縮退故障のどちらも故障検出に $v[q_i] = s_{u(i)}$ が必要である.従って求められた信号線の0,1縮退故障も F_i に含まれる.

例 5-3：図 5-1の回路について例を示す.状態 (0,0) が到達不能状態である.図 5-6の信号線上の 0, 1 は, $v[q_1] = 1$ を与えたときに一意的に定まる信号値を示している.例えば $v[c] = 0$ であるから,故障 $c/0$ は F_1 に含まれる.また,信号値の定まったゲートの入力部分に存在する信号線を＊印で示している.例えば,信号線 a はゲート G の出力値が0であるため,故障検出に $v[q_1] = 0$ が必要であり,故障 $a/0, a/1$ は F_1 に含まれる.同様に $v[q_2] = 1$ について F_2 を求めると, F_1 と F_2 に共通に含まれる故障は, $a/1, b/1, c/0$ の3つである.これらの故障は到達不能状態 (0, 0) のときにのみ影響が現れる検出不能故障である. ❑

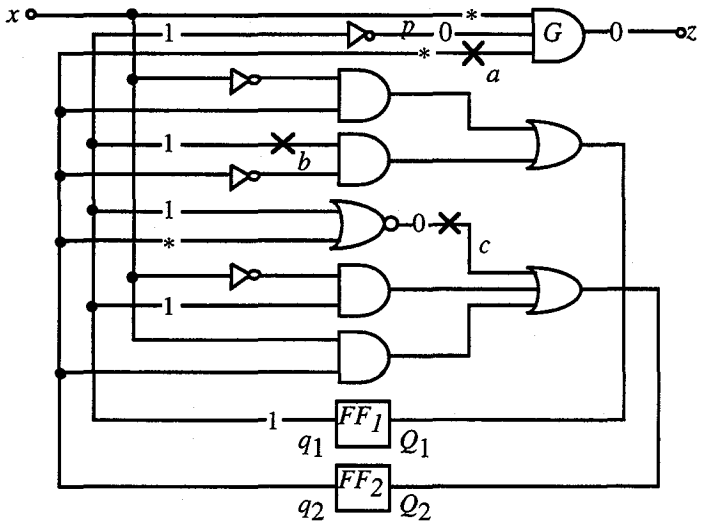


図 5-6　到達不能状態に基づく検出不能故障の判定

次に到達不能状態ベクトル s_u から得られた検出不能故障のうち1つについて，その故障の除去可能性を判定する．定理 5-1 より，状態 s_u が故障回路においても到達不能ならば除去可能である．状態 s_u が故障回路においても到達不能であるかどうかは，次の処理を行って調べることができる．$s_{u(i)}=\alpha \in \{0,1\}$ であるすべての i に対して，故障回路の疑似外部出力に $v[Q_i]=\alpha$ を割り当てる．含意操作を行い矛盾が起きれば s_u は故障回路においても到達不能であり，故障は除去可能である．

除去可能である故障については除去処理を行う．除去可能でないときは，s_u によって得られた他の検出不能故障について除去可能性の判定を行う．

本手法では冗長除去の後，検出不能故障の判定およびその除去を繰り返す．冗長除去によりフリップフロップが除去されなければ，用いた到達不能状態 s_u は到達不能のままである．従って繰り返し s_u を用いて検出不能故障の判定およびその除去を行うことができる．状態 s_u により除去可能な故障が見つからなくなれば，他の到達不能状態について処理を行う．

アルゴリズムの概略を図 5-7に示す．

Procedure: Find_RMF(s_u)
Input: unreachable state vector s_u
Output: set of removable faults F
set F be a set of all stuck-at faults
for each $s_{u(i)}$ which is 0 or 1
　　set F_i be a set of faults which cannot be detected without setting v[q_i] = $s_{u(i)}$
　　set $F = F \cap F_i$
for each fault $f \in F$
　　if s_u is not an unreachable state of the faulty circuit, then remove f from F

図 5-7　除去可能故障判定アルゴリズム

5．5　簡単化の手順

本手法の冗長除去のアルゴリズムを図 5-8に示す.

まず組合せ回路的検出不能故障に基づく冗長除去を行う．ここでは文献[14]のテスト生成を用いた手法を適用する．次に，手順Find_URSにより各フリップフロップの入力値から到達不能状態を求め，得られた到達不能状態を用いて検出不能故障を求める．検出不能故障のうち除去可能であるものを1つ除去し，繰り返し到達不能状態を用いた検出不能故障の判定および除去を行う．すべての到達不能状態について処理が終われば，再び組合せ回路的検出不能故障の除去を行う．

Procedure: RESURS(C)
Input: original circuit C
Output: modified circuit C'
set C' = Combinational_redundancy_removal(C)
REPEAT:
　　for each Q_j and value $\alpha \in \{0, 1\}$
　　　　set U = Find_URS(Q_j, α)
　　　　for each $s_u \in U$
　　　　　　do
　　　　　　　　set F=Find_RMF(s_u)
　　　　　　　　if $F \neq \phi$, then remove $f \in F$ from circuit C'
　　　　　　while $F \neq \phi$
　　　　　　if circuit is reduced based on s_u, then goto REPEAT
set C' = Combinational_redundancy_removal(C')

図 5-8　冗長除去アルゴリズム

5．6　実験結果

提案手法をC言語で記述し，Sun-SS/Classic ワークステーション上で実験を行った．対象回路はISCAS89ベンチマーク回路 [56] である．

表 5-1に得られた到達不能状態の総数を調べた結果を示す．s208 は状態の総数 2^8=256 状態のうち，到達不能状態が 225 状態存在した．s1423 と s5378 以降の回路についてはフリップフロップ数が多く総数は計算できなかったが,得られた到達不能状態のベクトルから少なくとも2の（フリップフロップ数－1）乗以上の到達不能状態が存在することはわかった．また s9234 より大きい回路に対しては，手順 Find_URS 内の *MODE* の値を *SKIP_MODE* とし，未充足ゲートの入力割り当てを制限した．したがって，それらの回路では到達不能状態の一部しか得られていない可能性がある．

表 5-1　到達不能状態数

回路名	FF数	到達不能状態数	回路名	FF数	到達不能状態数
s208	8	225	s820	5	0
s298	14	7,680	s832	5	0
s344	15	0	s838	32	4,294,641,825
s349	15	4,224	s953	29	536,867,834
s382	21	1,863,680	s1196	18	222,592
s386	6	51	s1238	18	220,672
s400	21	1,789,952	s1423	74	$>2^{73}$
s420	16	64,800	s1488	6	0
s444	21	1,789,952	s1494	6	0
s510	6	0	s5378	179	$>2^{178}$
s526n	21	983,040	s9234	228	$>2^{227}$
s526	21	983,040	s13207	669	$>2^{668}$
s641	19	512,512	s15850	597	$>2^{596}$
s713	19	512,512	s38417	1636	$>2^{1635}$

表 5-2にベンチマーク回路のゲート数，信号線数，フリップフロップ数と，組合せ回路的検出不能故障の冗長除去により除去された故障数と除去後の回路のゲート数，信号線数，フリップフロップ数およびそれらの削減率を示す．GT, LN, FF の各欄は，それぞれゲート数，信号線数，フリップフロップ数を示している．RF の欄は除去された故障数を示す．DGT, DLN, DFF の各欄は，それぞれゲート数の削減率，信号線数の削減率，フリップフロップ数の削減率を示す．ISCAS89 のベンチマーク回路には表に示す以外の回路も含まれているが，到達不能状態に基づく冗長除去が行われなかった回路は結果より除いた．s953 より小さい回路には組合せ回路的検出不能故障は存在しなかった．

表 5-3に組合せ回路的検出不能故障の冗長除去のあと到達不能状態に基づく冗長除去を行って得られた回路についての結果を示す．到達不能状態に基づく冗長除去により除去された故障数を RF の欄に示している．GT, LN, FF の欄にはそれぞれゲート数，信号線数，フリップフロップ数を，DGT, DLN, DFF の欄には元のベンチマーク回路に対しての削減率を示している．組合せ回路的検出不能故障の冗長除去も含めた処理時間を CPU の欄に示す．s420, s838 は本手法を適用することでフリップフロップ数が大幅に減少した．この実験では，冗長除去の対象となる順序回路的検出不能故障が平均４０個程度得られ，それらの故障に対応する信号線の除去により，最大８割，平均で２割程度ゲート数，信号線数，およびフリップフロップ数を削減できた．

表 5-2 ベンチマーク回路と組合せ回路的検出不能故障の冗長除去の結果

	ベンチマーク回路			組合せ回路的検出不能故障の冗長除去						
回路名	GT	LN	FF	RF	GT	DGT	LN	DLN	FF	DFF
s208	96	210	8	0	96	(0.0%)	210	(0.0%)	8	(0.0%)
s386	159	393	6	0	159	(0.0%)	393	(0.0%)	6	(0.0%)
s420	196	432	16	0	196	(0.0%)	432	(0.0%)	16	(0.0%)
s838	390	840	32	0	390	(0.0%)	840	(0.0%)	32	(0.0%)
s953	395	976	29	0	395	(0.0%)	976	(0.0%)	29	(0.0%)
s5378	2779	5344	179	29	2709	(2.5%)	5197	(2.8%)	176	(1.7%)
s9234	5597	9256	228	189	4881	(12.8%)	7957	(14.0%)	228	(0.0%)
s13207	7951	13300	669	406	7611	(4.3%)	12243	(7.9%)	667	(0.3%)
s15850	9772	15934	597	330	9486	(2.9%)	14988	(5.9%)	594	(0.5%)
s38417	22179	38445	1636	137	22179	(0.0%)	38221	(0.6%)	1636	(0.0%)

表 5-3　本手法の適用結果

回路名	RF	GT	DGT	LN	DLN	FF	DFF	CPU(sec.)
s208	13	66	(31.3%)	129	(38.6%)	5	(37.5%)	0.8
s386	34	155	(2.5%)	353	(10.2%)	6	(0.0%)	3.5
s420	13	73	(62.8%)	136	(68.5%)	5	(68.8%)	1.9
s838	13	67	(82.8%)	130	(84.5%)	5	(84.4%)	5.8
s953	2	395	(0.0%)	974	(0.2%)	29	(0.0%)	235.6
s5378	105	2547	(8.3%)	4768	(10.8%)	165	(7.8%)	178.9
s9234	8	4874	(12.9%)	7932	(14.3%)	228	(0.0%)	4002.7
s13207	63	7586	(4.6%)	12026	(9.6%)	667	(0.3%)	6073.8
s15850	11	9443	(3.4%)	14908	(6.4%)	591	(1.0%)	899.2
s38417	108	21959	(1.0%)	37731	(1.9%)	1616	(1.2%)	6527.2

次に冗長除去で得られた回路に対してテスト生成を行った結果について表 5-4に示す．使用したのは文献[25] の順序回路用テスト生成手法FASTESTである．ベンチマーク回路の欄にはベンチマーク回路の故障数(faults)とテスト生成により得られたテストパターン長(len.), 検出可能と判定された故障数(det.), および検出率(cov.)を示している．本手法適用後の回路に対しても同様に結果を示している．ほとんどの回路で検出率の向上が得られた．s953 では検出率が減少しているが, 得られたテストパターン長を比べると本手法適用後はパターン長が短くなっている．本手法適用後の回路に対してベンチマーク回路で得られたテストパターンを用いると本手法適用前の結果と同じ６０個の故障が検出可能であった．これより検出率の減少はテスト生成手法に依存した結果であり，実際には総故障数のうちの検出可能故障数の占める割合は減少しないと考えられる．

5．7　あとがき

到達不能状態に基づく除去可能な検出不能故障の判定法について述べた.除去可能な故障の判定には故障回路が正常回路と弱等価であるかどうかを判定しなければならないが,すべての状態を扱うことは実用的ではない．そこでどの状態からも設定できない到達不能状態のみに着目し,到達不能状態を故障検出に必要とする検出不能故障の除去可能性について考察した.ある到達不能状態を故障検出に必要とする故障は,その到達不能状態が故障回路においても到達不能になっていれば除去可能であることを示し,その判定を行う手法を提案した.実験結果より組み合わせ回路的検出不能故障の存在しない回路に対しても,本手法の冗長除去によりゲート数およびフリップフロップ数が削減されることがあることを示した.

表 5-4　テスト生成の結果

	ベンチマーク回路				本手法適用後			
回路名	length	faults	det.	cov.	length	faults	det.	cov.
s208	89	195	114	58%	69	109	96	88%
s386	108	358	258	72%	127	318	269	85%
s420	100	394	147	37%	86	109	94	86%
s838	115	789	208	26%	71	107	79	74%
s953	13	975	60	6.2%	9	971	53	5.5%
s5378	906	4147	3253	78%	821	3582	3094	86%
s9234	4	6427	12	0.2%	4	5359	12	0.2%

第６章　ＢＤＤを用いた到達不能状態の探索法

６．１　まえがき

従来テスト生成の観点から，正常回路において設定できない状態を求めるため，無効状態と呼ばれる状態を求める手法がいくつか提案されている[21, 62, 63, 64]．無効状態とは，弱同期化系列を印加した後には遷移させることができない状態である．本論文で着目している到達不能状態は，どの状態からも遷移させることができない状態であり，無効状態の一部である．第５章では，到達不能状態に着目することで，除去可能である故障を容易に判定する手法を提案したが，無効状態を故障の除去可能性の判定に有効に用いることはまだ考えられていない．したがって，無効状態ではなく，到達不能状態のみを判定する必要がある．到達不能状態は，フリップフロップの設定できない信号値の組合せとして得ることができる．第５．３節ではゲートレベル記述の回路から到達不能状態を得る手法について述べたが，厳密にすべての到達不能状態を求めるためにはゲートの入力値の組合せを数多く調べることが必要とされる．本章では，到達不能状態をフリップフロップの入力関数から求める手法について提案する．２つ以上のフリップフロップの入力関数を用いて，設定できない入力値の組をすべて数え上げることで到達不能状態を求めることができる．提案手法では，フリップフロップの入力関数を表現するために，論理関数を扱うのに適したＢＤＤを用いる．大規模回路に対しては，記憶容量や処理時間の問題から，すべてのフリップフロップの関数を扱うことは実用的ではない．そこで，ＢＤＤで同時に扱うことができるようにフリップフロップ集合を分割する手法についても述べる．

６．２　ＢＤＤによる探索

本手法ではまず回路のフリップフロップの入力線の論理関数をＢＤＤで表現し，その関数間の関係を調べることで到達不能状態の判定を行う．論理関数の処理には，文献[47]をもとに神戸大学で開発されたＢＤＤパッケージ[48]を用いた．判定には論理関数の制限を求めるＢＤＤの処理手順 bdd_cofact を利用する．手順 bdd_cofact は入力として２つのＢＤＤ b_1, b_2，出力として b_1 の論理関数の定義域を b_2 の論理関数が真となる領域に制限した論理関数のＢＤＤを返す．例として b_1 が論理関数（$x_1 \wedge x_2$）を表すＢＤＤ，b_2 が論理関数 x_2 を表すＢＤＤであるときを考える．このとき b_2 が真となる領域は $x_2 = 1$ である．$x_2 = 1$ の領域における b_1 は $b_1 = (x_1 \wedge x_2) = (x_1 \wedge 1) = x_1$ となり，この関数が bdd_cofact(b_1, b_2) で得られる関数である．表 6-1に bdd_cofact をいくつかの論理

関数に適用して得られる関数の例を示す．

表 6-1　bdd_cofact を適用した例（入力変数を x_1, x_2 とする）

b_1	b_2	bdd_cofact(b_1, b_2)
TRUE	b_2（任意の関数）	TRUE
FALSE	b_2（任意の関数）	FALSE
b_1（定数以外の任意の関数）	TRUE	b_1
b_1（定数以外の任意の関数）	FALSE	FALSE
x_1	x_2	x_1
$x_1 \wedge x_2$	x_2	x_1
$x_1 \vee x_2$	x_2	TRUE
$x_1 \wedge x_2$	$\overline{x_2}$	FALSE

もし b_2 が真となる領域で b_1 が真となれば，$b_2=1$ のときに常に $b_1=1$ であり，したがって $b_1=0$ かつ $b_2=1$ とする入力が存在しないことがわかる．したがって，手順 bdd_cofact をフリップフロップの入力関数に適用して，到達不能状態の判定を行うことができる．もし，i 番目と j 番目のフリップフロップの入力関数 $\delta_{(i)}$, $\delta_{(j)}$ に対して bdd_cofact($\delta_{(i)}$, $\delta_{(j)}$) = TRUE となれば，$\delta_{(j)}=1$ となるとき必ず $\delta_{(i)}=1$ であることがわかる．よって，$D_{(j)}(1) \subseteq D_{(i)}(1)$ であり，定理 5-2 より $s_{(i)}=0$ かつ $s_{(j)}=1$ なる状態 s は到達不能状態であることがわかる．表 6-2 に bdd_cofact の結果と到達不能状態との関係を示す．

表 6-2　bdd_cofact を用いた到達不能状態の判定

b_1	b_2	bdd_cofact(b_1, b_2)	得られる到達不能状態
$\delta_{(i)}$	$\delta_{(j)}$	TRUE	$s_{(i)}=0$ かつ $s_{(j)}=1$
$\delta_{(i)}$	$\delta_{(j)}$	FALSE	$s_{(i)}=1$ かつ $s_{(j)}=1$
$\delta_{(i)}$	$\overline{\delta}_{(j)}$	TRUE	$s_{(i)}=0$ かつ $s_{(j)}=0$
$\delta_{(i)}$	$\overline{\delta}_{(j)}$	FALSE	$s_{(i)}=1$ かつ $s_{(j)}=0$
$\delta_{(i)}$	$\delta_{(j)} \wedge \delta_{(k)}$	TRUE	$s_{(i)}=0$ かつ $s_{(j)}=1$ かつ $s_{(k)}=1$
$\delta_{(i)}$	$\delta_{(j)} \wedge \overline{\delta}_{(k)}$	TRUE	$s_{(i)}=0$ かつ $s_{(j)}=1$ かつ $s_{(k)}=0$

例 6-1：　手順 bdd_cofact によって到達不能状態が求められる例を示す．第５．２節の図 5-1の回路のフリップフロップの入力関数 $\delta_{(1)}, \delta_{(2)}$ をＢＤＤで表現したものを，図 6-1に示す．図中の黒丸は，否定エッジ[47]を示す．図 5-1の回路では状態 (0, 0) が到達不能状態であるが，それは手順 bdd_cofact を用いて次のように求められる．

手順 bdd_cofact では，各入力値に対する２関数の値を比べることで，制限された関数を求める．ここで，bdd_cofact($\overline{\delta}_{(2)}, \delta_{(1)}$) により得られる関数 b_r を考える．関数 $\overline{\delta}_{(2)}$ のＢＤＤは， $\delta_{(2)}$ の根のノードの否定エッジを除いたものである．入力値が x＝0 かつ $q_1 = 0$ である場合，双方のＢＤＤは同じノード (q_2, 0, 1) を示す．したがって，今 $\overline{\delta}_{(2)} = 1$ になる入力値に定義域を制限するので，関数 b_r の入力値 x=0 かつ q_1=0 に対する値は１である．また，入力値が x=0 かつ $q_1 = 1$ である場合，関数 $\delta_{(1)}$ の値が１となるので，関数 b_r の入力値 x=0 かつ $q_1 = 1$ に対する値も１である．入力値が x＝1 の場合，関数 $\overline{\delta}_{(2)}$, $\delta_{(1)}$ がともに同じノードを示すので，入力値 $x = 0$ かつ $q_1 = 0$ の場合と同様，関数 b_r の入力値 $x = 1$ に対する値も１となる．したがって，関数 $\delta_{(1)}$ の定義域を $\overline{\delta}_{(2)} = 1$ になる入力値に制限した関数 b_r は図 6-2(a) のようなＢＤＤで表される．このＢＤＤを整理すると，図 6-2(b) に示すように関数 $b_r = 1$ が得られる．

したがって，$\delta_{(2)} = 0$ のとき必ず $\delta_{(1)} = 1$ であることから，表 6-2より状態 (0, 0) は到達不能状態であることがわかる．❑

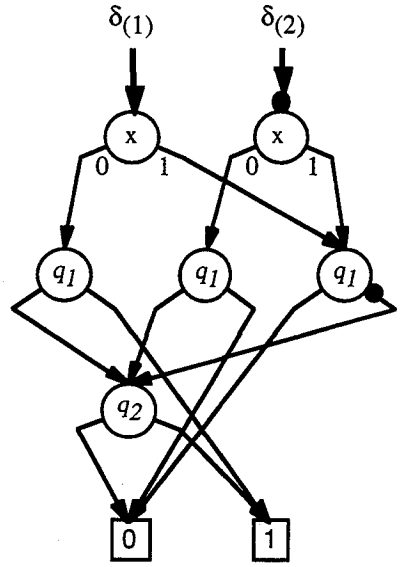


図 6-1　図 5-1の回路のフリップフロップの入力関数 $\delta_{(1)}$, $\delta_{(2)}$

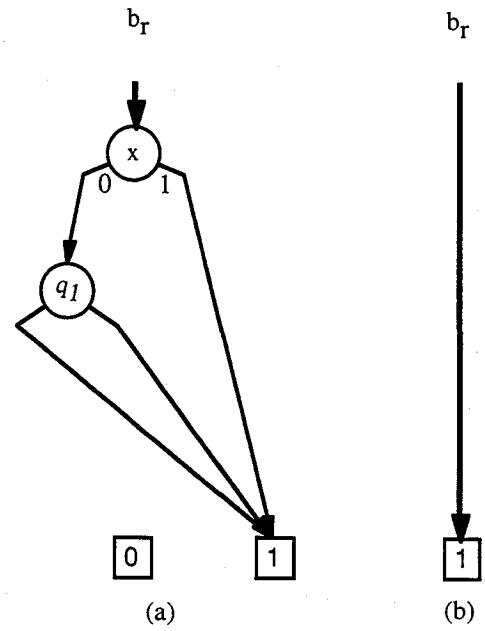


図 6-2　bdd_cofact($\overline{\delta}_{(2)}$, $\delta_{(1)}$) の結果

図 6-3に手順bdd_cofactの疑似コードを示す．手順bdd_cofactは，関数 b_1 の定義域を関数 b_2 が真となる入力値に制限して得られる関数 b_r を求める手順である．例 6-1 に示したように，ＢＤＤの根のノードの変数から順に入力値を割り当ててゆき，関数 b_1, b_2 について，その入力値を与えたときの部分関数を表すノードを比較する．この比較の結果を以下のように分類し，関数 b_r を生成する：

1）与えられた入力値における関数 b_2 の部分関数が，常に真(TRUE)となる場合：

この入力値における関数 b_1 の部分関数を，同じ入力値における b_r の部分関数とする．

2）与えられた入力値における関数 b_2 の部分関数が，常に偽(FALSE)となる場合：

この入力値における b_r の部分関数を常に偽(FALSE)とする．

3）１，２以外で，与えられた入力値における関数 b_1 の部分関数と，関数 b_2 の部分関数が等しい場合：

この入力値における b_r の部分関数を常に真(TRUE)とする．

4）１，２以外で，与えられた入力値における関数 b_2 の部分関数が関数 b_1 の部分関数を反転したものと等しい場合：

この入力値における b_r の部分関数を常に偽(FALSE)とする．

5）１～４以外の場合：

次の変数に入力値を割り当て，部分関数を比較する．

Procedure: bdd_cofact(b_1, b_2)
Input: BDD $b_1 = (i_1, b_{10}, b_{11})$, $b_2 = (i_2, b_{20}, b_{21})$
Output: BDD b_r
if b_1 is constant node (TRUE or FALSE), then set $b_r = b_1$
else if b_2 is TRUE, then set $b_r = b_1$
else if b_2 is FALSE, then set b_r = FALSE
else if b_1 represents the function that is identical to b_2, then set b_r = TRUE
else if b_1 represents the negative function of b_2, then set b_r = FALSE
else
　　if $i_1 = i_2$, then
　　　　set $b_r = (i_1,$ bdd_cofact(b_{10}, b_{20}), bdd_cofact(b_{11}, b_{21}))
　　else if $i_1 < i_2$, then
　　　　set $b_r = (i_2,$ bdd_cofact(b_{10}, b_2), bdd_cofact(b_{11}, b_2))
　　else set $b_r = (i_1,$ bdd_cofact(b_1, b_{20}), bdd_cofact(b_1, b_{21}))

図 6-3　bdd_cofactの疑似コード

フリップフロップの入力値の関数に bdd_cofact を用いて到達不能状態を得る手順を手順 FindURS_BDD とする．図 6-4に手順 FindURS_BDD の疑似コードを示す．この手順では，以下のような処理が行われる．

初めに各フリップフロップの入力関数のＢＤＤを生成する．そして，そのＢＤＤの１対１の関係を手順 bdd_cofact を用いて調べ，得られた到達不能状態を保存する．その次に３つ以上のフリップフロップの組に対して手順 bdd_cofact を用いて到達不能状態を探索する．この手法では最悪すべての状態について bdd_cofact を適用しなければならないが，なるべく適用回数を減らすために以下の２つの処理を行っている．

まず適用回数を減らすための処理の１つとして，得られた到達不能状態もＢＤＤの形で保存し，すでに到達不能と判定された状態について bdd_cofact を適用することがないようにした．ＢＤＤの入力変数を s_1, s_2, ..., s_n とし，それぞれが各フリップフロップの値を表すものとする．初めはこのＢＤＤの値を０（FALSE）とし，到達不能状態が得られたときにその値の組合せのときにＢＤＤの値が１（TRUE）になるようにＢＤＤを変形する．例えば１番目のフリップフロップと２番目のフリップフロップの値が共に０である状態が到達不能であることが分かったならば，ＢＤＤの関数を $s_1 = 0$ かつ $s_2 = 0$ のとき値が１になるようにする．このようにして得られた到達不能状態をＢＤＤの形で表現し，bdd_cofact の適用前にすでに到達不能であると判定されているかどうかを調べ，到達不能であると判定されている状態ならば bdd_cofact を適用しないようにした．

もう一つは，bdd_cofact で到達不能状態が得られたときになるべく到達不能状態のベクトルが大きくなるようにしている．bdd_cofact を適用するときには１番目から（ｉ－１）番目までの関数値を決めて，その関数値を満たす入力値の範囲内に制限したｉ番目の関数を bdd_cofact で求めている．したがって，到達不能状態の原因となるフリップフロップが１～ｉ番目の中の一部のフリップフロップであっても，１～ｉ番目のフリップフロップの値がすべて０か１に設定された状態を調べていることになる．このうち，到達不能状態の原因となるフリップフロップの部分だけ０か１に設定された状態を求めると，それに含まれる状態についての処理を省くことができる．具体的には，１～（ｉ－１）番目の値の組合せについてはそれ以前に処理を行っているので，ｉ番目から１番目へ逆順に bdd_cofact を適用し，設定不能な組合せになっているフリップフロップの値を求める．例えば，状態ベクトル（X, 0, 0, X, 1, X, X）が到達不能状態を表しているとき，処理中ではまず最初に状態（0, 0, 0, 0, 1, X, X）について到達不能であることが，bdd_cofact(f_5, $\overline{f_1}$

$\wedge\ \overline{f_2} \wedge\ \overline{f_3} \wedge\ \overline{f_4}$) =FALSE から得られる．このとき逆順に bdd_cofact(f_4, f_5), bdd_cofact(f_3, $\overline{f_4} \wedge f_5$), bdd_cofact($f_2$, $\overline{f_3} \wedge\ \overline{f_4} \wedge f_5$) を求める．すると bdd_cofact($f_2$, $\overline{f_3} \wedge\ \overline{f_4} \wedge f_5$)=TRUE となり，状態（X, 0, 0, 0, 1, X, X）が到達不能であることがわかる．この処理で状態（1, 0, 0, 0, 1, X, X）については bdd_cofact を適用しなくても良くなり，処理を減らすことができる．さらに詳しく調べることで状態（X, 0, 0, X, 1, X, X）が到達不能であることも分かるが，処理の複雑さからここでは行っていない．この処理では得られた到達不能状態の１～ｉ番目の値のうち，１～ｋ(k≤i)番目までの値がＸにできるかが分かる．

Procedure: FindURS_BDD
Input: a group of flip-flops GRP= { f_1, f_2, ..., f_n}
Output: a set of unreachable states U
set s = ($s_{(1)}$, $s_{(2)}$, ..., $s_{(n)}$) $s_{(1)}$ = X for all i
for each $f_i \in GRP$
　　calculate BDD, $\delta_{(i)}$, that represents the input function of f_i
set $U = \Phi$
for each f_i (i = 1, 2, ..., n)
　　for each f_j ($j = i$+1, i+2, ..., n)
　　　　calculate b_c = bdd_cofact($\delta_{(i)}$, $\delta_{(j)}$)
　　　　if b_c == TRUE, then Add_unreachable_state(U, s') such that $s'_{(i)} = 0$ and $s'_{(j)} = 1$
　　　　else if b_c == FALSE, then Add_unreachable_state(U, s') such that $s'_{(i)} = 1$ and $s'_{(j)} = 1$
　　　　calculate b_c = bdd_cofact($\delta_{(i)}$, $\overline{\delta_{(j)}}$)
　　　　if b_c == TRUE, then Add_unreachable_state(U, s') such that $s'_{(i)} = 0$ and $s'_{(j)} = 0$
　　　　else if b_c == FALSE, then Add_unreachable_state(U, s') such that $s'_{(i)} = 1$ and $s'_{(j)} = 0$
apply FindURS_BDDsub(s, 1, TRUE, U)

Procedure: FindURS_BDDsub(s, i, b_s, U)
Input: state s = ($s_{(1)}$, $s_{(2)}$, ..., $s_{(n)}$)
Input: index i
Input: BDD b_s which represents state s
Input/Output: a set of unreachable states U
if state s is already identified as unreachable (i.e. $s \in U$), then return
if bdd_cofact($\delta_{(i)}$, b_s) == FALSE, then set $s_{(i)}$ = 1 and Add_unreachable_state(U, s)
else if $i < n$, then
　　set $s_{(i)} = 0$
　　apply FindURS_BDDsub(s, i+1, $b_s \wedge\ \overline{\delta_{(i)}}$, U)
set $s_{(i)}$ = X
if bdd_cofact($\delta_{(i)}$, b_s) == TRUE, then set $s_{(i)} = 0$ and Add_unreachable_state(U, s)
else if $i < n$, then
　　set $s_{(i)}$ = 1
　　apply FindURS_BDDsub(s, i+1, $b_s \wedge \delta_{(i)}$, U)
set $s_{(i)}$ = X

図 6-4　ＢＤＤを用いた到達不能状態の探索

6．3　フリップフロップ集合の入力重複度による分割

前節ではＢＤＤを用いて到達不能状態を得る手法を提案したが，ＢＤＤで用いられる入力変数やノード数が多くなるとメモリが足りなくなったり実用的時間で処理できなくなる可能性がある．このため，本節では一度に判定の対象とするフリップフロップの数を減らすために，フリップフロップ集合の分割を考える．

図 6-5の回路の FF_1, FF_n のようにある２つのフリップフロップの入力部分回路が外部入力およびフリップフロップの出力を一つも共有しないならば，その２つのフリップフロップの値は完全に独立であり，その２つのフリップフロップ間で設定不能な論理値の組合せは存在しない．この例で，FF_1, FF_n を同時に処理すると，全く共通部分がない関数について無駄な処理を行うことになる．したがって，このようなフリップフロップを同時に到達不能状態の判定に用いることは無駄である．ここでは組合せ回路部を考え，各フリップフロップの入力線への経路のある外部入力とフリップフロップの出力線を調べ，その重複度の高いフリップフロップ同士を一つの集合に含めることを考える．以降，外部入力とフリップフロップの出力をあわせて回路の入力と呼ぶこととする．

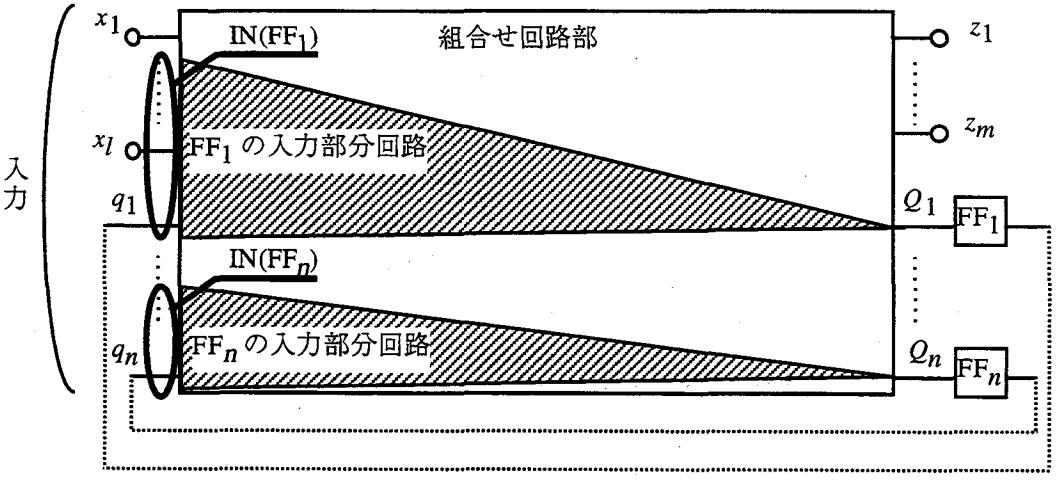


図 6-5　フリップフロップの入力部分回路を考慮した集合分割

ＢＤＤで扱うことのできるように同時に処理するフリップフロップの数と入力変数の数についての上限を与え，その制限に収まるようにフリップフロップ集合の分割を行う手順を手順 FF_partition とする．手順 FF_partition の疑似コードを図 6-6に示す．手順 FF_partition では以

下のような処理を行う．

まず，各フリップフロップ f_i の入力部分回路を調べ，フリップフロップ f_i に接続する経路の存在する入力の集合 $IN_{(i)}$ を求める．$IN_{(i)}$ の要素数がフリップフロップ f_i のＢＤＤに必要な入力変数の数である．この要素数が一番多いフリップフロップを，１つ目のフリップフロップ集合の最初の要素とする．次に，この集合に含まれるフリップフロップの IN との重複度が最も高い IN をもつフリップフロップを，入力変数とフリップフロップ数の上限を超えない限り，集合に加えてゆく．集合に加えることのできるフリップフロップがなくなれば，いずれの集合にも属していないフリップフロップのうち，$IN_{(i)}$ の要素数が一番多いフリップフロップを次のフリップフロップ集合の最初の要素とし，この集合に同様にしてフリップフロップを追加してゆく．この処理を全てのフリップフロップが集合に分けられるまで繰り返し行う．

得られたフリップフロップ集合の要素数が１つである場合は，到達不能状態に必要のない集合である．要素数が２つ以上である集合を用いて前節の手順 FindURS_BDD に適用し，到達不能状態を求める．

Procedure: FF_partition(*FF_limit*, *IN_limit*)
Input: limit of the number of FFs which are included in one group *FF_limit*
Input: limit of the number of inputs which are related to FFs in one group *IN_limit*
Output: the number of groups g
Output: the groups of FFs G_i $(1 \le i \le g)$

for each FF f_i
　obtain the set of inputs, $IN_{(i)}$, from which there exists a path to f_i
set $g = 1$
while ungrouped FF exists
　select f_i such that $IN_{(i)}$ is the largest among ungrouped FFs
　set $GI_g = IN_{(i)}$ and set $G_g = \{$ungrouped FF $f_k \mid IN_{(k)} \subseteq GI_g\}$
REPEAT:
　if $|G_g| >$ *FF_limit*, then
　　separate G_g to the groups G_g, G_{g+1}, ..., G_{g+k}, such that each groups contains less FFs than *FF_limit*
　　set $g = g+k$
　else if $(|G_g| <$ *FF_limit* $) \wedge (|GI_g| <$ *IN_limit*$)$, then
　　select f_j such that $GI_g \cap IN_{(j)}$ is the largest among ungrouped FFs
　　if $|GI_g \cup IN_{(j)}| <$ *IN_limit*, then
　　　set $GI_g = GI_g \cup IN_{(j)}$ and set $G_g = \{f_k \mid IN_{(k)} \subseteq GI_g\}$
　　　goto REPEAT
　　else set $g = g + 1$

図 6-6　フリップフロップのグループ分けアルゴリズム

６．４　フリップフロップ集合の構造解析による分割

前節ではフリップフロップをその依存する入力の重複度により分割を行った.しかしこのときに次のような問題点がある.

[問題点]依存する入力の重複度が低いフリップフロップの組においても到達不能状態が存在する可能性が高い場合がある.

この問題点は図 6-7に示すような回路例で明らかにすることができる.この回路においてはフリップフロップ FF_1 と FF_2 は共通する外部入力が１つしかないが，状態 $(FF_1,FF_2)=(1,1)$ が到達不能状態になっている.このような部分回路が存在しているときに６．３節の手法を適用すると，図中のフリップフロップ FF_1 と FF_2 は共通入力が少ないため，別々の集合に分割されることが考えられ，集合分割による到達不能状態の見落としが起きやすくなる.

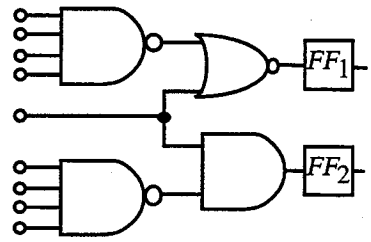


図 6-7　到達不能状態が存在するが入力重複度の低いフリップフロップ

これらの問題点を改善するために，外部入力の重複度と共に，回路の内部構造を考慮して，なるべく設定不能な値の組合せをもつフリップフロップの組を１つの集合内に含めることのできる方法を考える．また，１つのフリップフロップが複数の集合に含まれることを許し，到達不能状態を見つける可能性を高くするように集合を作成する.

まず，各分岐点に着目してフリップフロップの集合を生成する．１つの分岐点 s に着目し，その分岐点からの経路が存在しているフリップフロップを数え上げて１つの集合 $F(s)$とする．図6-8に示すように，この集合の要素数が２つ以上であるときは，その集合に含まれている複数のフリップフロップが分岐点 s を共有しているため，それらのフリップフロップは少なくとも１つの外部入力線またはフリップフロップの出力線を共有していることがわかる.したがってそのフリップフロップ間に設定不能な信号値の組合せが存在すれば，到達不能状態があることになる.

また，ここで得られた複数の集合が共通のフリップフロップを含んでいる場合がある．図 6-

8では，F(s1) と F(s2) の両方にフリップフロップ FF_3 が含まれている．この場合，F(s1) と F(s2) を１つの集合にできれば，より多くの到達不能状態が得られる可能性がある．したがって，フリップフロップ数などの制限値を超えないならば，これらの集合を１つの集合に統合する操作を行う．

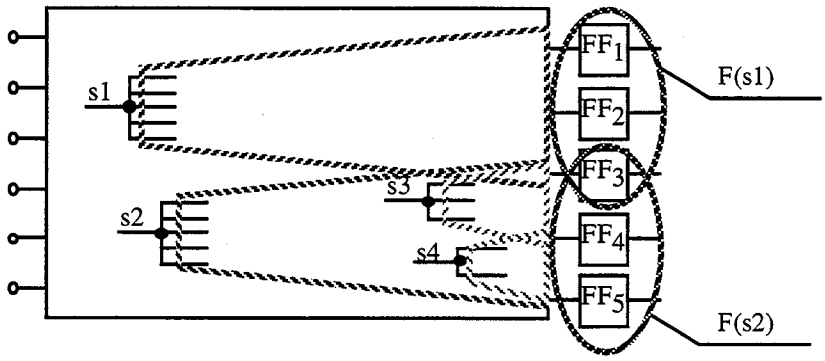


図 6-8　構造解析に基づくフリップフロップ集合の生成

このように回路の分岐点に着目し，分岐点からの経路の存在する複数のフリップフロップを１つの集合にする手順を手順 FF_groups とする．手順 FF_groups の疑似コードを図 6-9に示す．フリップフロップの集合を作成するときの制限値として，集合内のフリップフロップ数の上限値と，集合内のフリップフロップへの経路の存在する外部入力数（ＢＤＤの入力変数）の上限値，集合内のフリップフロップの入力関数の表現に必要なＢＤＤのノード数の総計の上限値を定めた．この制限内におさまるなるべく大きいフリップフロップ集合を以下のように生成する．

［手順 FF_groups］

Step-fg1）各フリップフロップのＢＤＤを１つづつ作成し，それぞれのＢＤＤで必要となる入力変数とＢＤＤのノード数を計算する．

Step-fg2）分岐点を外部入力に近い順に並べたリストを作り，その順に分岐点 s を選び，以下の処理を行う．

fg2-1）分岐点 s からの経路の存在するフリップフロップの集合 F(s) を作成する．

fg2-2）集合 F(s) が各上限値のなかにおさまるまで，集合 F(s) からフリップフロップを，ＢＤＤのノード数が一番多い順に，除いてゆく．

fg2-3）fg2-2 で一つもフリップフロップが除かれていないならば，分岐点 s からの経路

がある出力側の分岐点をすべてリストから除く．（例：図 6-8で集合 F(s2) が上限内に収まるならば，分岐点 s3 および s4 をリストから除く）

Step-fg3）得られた集合のうち，同じ要素をもつものや，他の集合に包含される集合があれば，それらを取り除く．

Step-fg4）得られた集合のうち入力に共通部分をもつ複数の集合を，集合内のフリップフロップ数の上限と入力変数の数の上限を超えない範囲内で，統合してゆく．

Procedure: FF_groups
Input: limit of the number of inputs which are related to FFs in one group, *IN_LIMIT*
Inout: limit of the number of nodes used in BDDs, *SIZE_LIMIT*
Output: the number of groups, *n_grp*
Output: the groups of FFs, *GRP*(i) ($1 \le i \le n\_grp$)

set S = {fanout stems$s$}
for each flip-flop f
　set *IN*(f) = {primary inputs and/or present state outputs from which a path exists to flip-flop f}
　set *BDD_SIZE*(f) = (the number of nodes used in the BDD of the function of f)
set $n\_grp = 0$
while $S \neq \phi$
　select stem s from S
　set F(s) = {flip-flops to which a path exists from stem s}
　if $\left(\sum_{f \in F(s)} BDD\_SIZE(f) \le SIZE\_LIMIT\right) \wedge \left(\left|\bigcup_{f \in F(s)} IN(f)\right| \le IN\_LIMIT\right)$, then
　　check the fanout stems to which a path exists from stem s and delete them from S
　else
　　while $\left(\sum_{f \in F(s)} BDD\_SIZE(f) > SIZE\_LIMIT\right) \vee \left(\left|\bigcup_{f \in F(s)} IN(f)\right| > IN\_LIMIT\right)$
　　select $f \in F(s)$ which is the largest BDD_SIZE, and delete f from $F(s)$
　　end
　if no *GRP*(i) exists such that *GRP*(i)　F(s), then
　　set *GRP*(n_grp) = $F(s)$, and $n\_grp = n\_grp + 1$
end
for each *GRP*(i) ($0 \le i \le n\_grp$)
　for each *GRP*(j) ($j \neq i$)
　　if $\left(\sum_{f \in GRP(i) \cup GRP(j)} BDD\_SIZE(f) \le SIZE\_LIMIT\right) \wedge \left(\left|\bigcup_{f \in GRP(i) \cup GRP(j)} IN(f)\right| \le IN\_LIMIT\right)$,
　　then *GRP*(i) = *GRP*(i) ∪ *GRP*(j), and delete *GRP*(j), and set $n\_grp = n\_grp - 1$
　end
end

図 6-9　構造解析によるフリップフロップ集合生成のアルゴリズム

６．５　実験結果

６．３節の手法と６．４節の手法をそれぞれＣ言語で記述し，ISCAS89 ベンチマーク回路に対して実行した．まず，６．３節で述べたフリップフロップの分割法を適用してフリップフロップ集合の分割を行い，６．２節の到達不能状態の探索法で得られた到達不能状態の数を表 6-3 に示す．ＦＦの欄は回路内のフリップフロップ数，到達不能状態数は手順 FindURS_BDD で得られた到達不能状態数を示す．

６．４節の手法によるフリップフロップ集合の分割では１つのフリップフロップが複数の集合に含まれているため，到達不能状態が重複して得られる可能性がある．したがって得られた到達不能状態数の総数を求めるのは困難なため，総数の計算は行っていない．

表 6-3　到達不能状態数

回路名	FF	到達不能状態数	回路名	FF	到達不能状態数
s208	8	225	s832	5	7
s298	14	10,584	s838	32	6*2^29
s344	15	9,536	s953	29	536,870,408
s349	15	9,536	s1196	18	259,492
s382	21	2,073,412	s1238	18	259,492
s386	6	51	s1423	74	3*2^71
s400	21	2,073,412	s1488	6	16
s420	16	64,800	s1494	6	16
s444	21	2,073,412	s5378	179	87*2^176
s510	6	3	s9234	228	0
s526n	21	1,695,692	s13207	669	372*2^666
s526	21	1,695,692	s15850	597	78*2^594
s641	19	517,625	s35932	1728	0
s713	19	517,625	s38417	1636	0
s820	5	7	s38584	1452	84*2^1449

得られた到達不能状態を５章と同様の回路簡単化手法に応用した実験を行った．

まず，得られた到達不能状態ベクトルを二段回路の簡単化手法である espresso[5]を用いて，なるべく多くの要素の値が X になるよう変換した．そしてそのベクトルを５章の手順 Find_RMF に適用した．ベンチマーク回路に対する実験結果を示す．組合せ回路的検出不能故障の冗長除去を行った後の回路に対して，得られた到達不能状態を用いて冗長除去を行った．表 6-4に，冗長除去により簡単化することができたベンチマーク回路について，それらの元の回路と組合せ回路的検出不能故障の冗長除去後の回路のそれぞれに含まれるゲート数（GT），信号線数（LN），

およびフリップフロップ数（FF）を示す．

入力重複度により分割されたフリップフロップ集合を用いて到達不能状態を求めた場合の実験結果を表 6-5に，同様に，構造解析により分割された集合を用いた場合の実験結果を表 6-6に示す．ＲＦの欄は冗長除去で用いた除去可能な故障の数を示している．ほとんどの回路において同じような簡単化の結果が得られたが，いくつかの回路では違いが見られた．構造解析により分割されたフリップフロップ集合を用いて得られる到達不能状態は，入力重複度により分割されたフリップフロップ集合を用いて得られる到達不能状態よりも多いため，構造解析で得られた到達不能状態を用いたほうが，冗長除去をより多く行うことができると考えられる．実験結果ではs208, s420, s953, s838, s953, s5378, s9234 についてそのような傾向が見られる．ところがs13207, s38584 については，構造解析を用いた場合より入力重複度による分割を用いた場合の方がより多く冗長除去が行われていた．この理由としては５．２節で示したように，冗長除去を行うことで他の故障の除去可能性が変化することが考えられる．したがって，除去可能な故障が複数得られたときにどの故障に基づき除去を行うかを考慮すれば，より効率の良い冗長除去を行える可能性がある．

表 6-4　ベンチマーク回路と組合せ回路的検出不能故障の冗長除去の結果

	ベンチマーク回路			組合せ回路的検出不能故障の冗長除去			
回路名	GT	LN	FF	RF	GT	LN	FF
s208	96	210	8	0	96	210	8
s344	160	346	15	0	160	346	15
s349	161	351	15	2	160	346	15
s386	159	393	6	0	159	393	6
s420	196	432	16	0	196	432	16
s820	289	839	5	0	289	839	5
s832	287	851	5	13	281	831	5
s838	390	840	32	0	390	840	32
s953	395	976	29	0	395	976	29
s1488	653	1507	6	0	653	1507	6
s1494	647	1513	6	11	639	1493	6
s5378	2779	5344	179	29	2709	5197	176
s9234	5597	9256	228	189	4881	7957	228
s13207	7951	13300	669	406	7611	12243	667
s38584	19253	38710	1452	451	16797	34227	1435

表 6-5 冗長除去の結果（入力重複度によるフリップフロップの分割）

回路名	RF	GT	LN	FF	CPU(sec.)
s208	11	62	125	5	0.18
s344	1	160	344	15	2.48
s349	1	160	344	15	2.28
s386	34	151	349	6	0.25
s420	28	68	131	6	0.25
s820	22	280	798	5	0.33
s832	22	272	790	5	0.33
s953	2	395	974	29	267.01
s1488	11	650	1493	6	0.53
s1494	11	636	1479	6	0.53
s5378	52	2573	4891	169	17.95
s9234	3	4692	7762	228	7.47
s13207	7	7348	11967	667	30.38
s38584	8	16797	34219	1435	181.37

表 6-6 冗長除去の結果（構造解析によるフリップフロップの分割）

回路名	RF	GT	LN	FF	CPU(sec.)
s208	13	62	125	5	0.31
s344	1	160	344	15	5.23
s349	1	160	344	15	4.62
s386	34	151	349	6	1.11
s420	32	78	153	6	1.13
s820	22	280	798	5	4.15
s832	22	272	790	5	4.37
s838	72	73	148	6	136.43
s953	4	394	970	29	138.65
s1488	11	650	1493	6	8.65
s1494	11	636	1479	6	8.76
s5378	111	2517	4786	171	53.87
s9234	10	4685	7745	228	6761.75
s13207	4	7351	11977	667	6020.11
s38584	6	16796	34219	1435	10646.84

６．６　あとがき

本章では論理関数の表現形式であるＢＤＤを用いて到達不能状態を求める手法を提案した.提案手法ではフリップフロップの入力関数をＢＤＤで表現し,設定できないフリップフロップの値の組合せを求めることで到達不能状態を求める.ＢＤＤは論理関数の処理を高速に行うことができるが,大規模の回路に対しては記憶容量等の問題から一度に全てのフリップフロップの入力関数を扱うことができない.そこで,一度に扱うフリップフロップの数を制限した上で到達不能状態がより多く求められるようなフリップフロップ集合の分割手法を提案した.一つはフリップフロップの依存している外部入力線およびフリップフロップの出力線の重複度を考慮してフリップフロップの集合を生成し,もう一つは回路の分岐点に着目し分岐点から経路のあるフリップフロップの集合を生成した.実験結果から分岐点に着目した構造解析に基づくフリップフロップ集合の分割を行った方が冗長除去がより多く行われることが確認された.さらに今後の課題としては,除去可能な故障が複数あった場合,除去による他の部分の除去可能性が変化することを考慮すればより効率の良い冗長除去を行うことができると考えられる.

第7章　結論

本論文では順序回路の冗長除去による簡単化手法を提案した.

第3章では, 特定の初期状態を持つ回路を対象としたリタイミング手法を提案した. リタイミングとは, 回路のフリップフロップの配置を変える順序回路の再合成手法であり, 回路の動作速度の向上などに用いられているが, 従来のリタイミング手法が対象としている回路は, フリップフロップのリセット端子を使わないことを前提としていた. もしこれらの手法を, フリップフロップのリセット端子によりある特定の状態に初期化される回路に適用すると, その状態に固有の出力系列が失われる危険性があった. そこで, フリップフロップに初期状態に対応する5値の論理値を持たせて, 決められた初期状態と等価な状態がリタイミングを行った後の回路に必ず存在することを保証した, 新たなリタイミング手法を提案した. フリップフロップ数, ゲート数の削減による簡単化手法に適用した結果, 初期状態の制限による簡単化能力の低下は平均0.6% と小さく, 初期状態を考慮しない場合と同等の簡単化を行うことができた. 提案した再配置手法は簡単化以外を目的としたリタイミングにおいても適用可能である. したがって従来提案されている回路の高速化などを目的としたリタイミング手法も, 本手法の再配置の制限を加えることでリタイミングを適用することに問題のあった特定の初期状態をもつ回路に対しても適用できるようになると考えられる.

第4章では, 回路簡単化において多くの冗長部分を判定することを目的としたリタイミング手法を提案した. リタイミングにより順序回路的検出不能故障が組合せ回路的検出不能故障に変換される場合があることを利用し, 組合せ回路的検出不能故障に変換される可能性のある順序回路的検出不能故障をなるべく多く変換できるようなリタイミングを考察した. どの状態からも遷移させることのできない到達不能状態は分岐点の入力方向へのフリップフロップの再配置を含むリタイミングにより削除されることと, 到達不能状態がリタイミングにより削除される場合に組合せ回路的検出不能故障に変換される可能性が高いことを考慮し, 到達不能状態の削減を目的としたリタイミング手法を提案した. 実験により7つのベンチマーク回路に対して提案手法を適用すると, 平均30個程度の順序回路的検出不能故障が組合せ回路的検出不能故障に変換され, 効率の良い回路簡単化を行うことができることを確認した.

第5章では, 順序回路的検出不能故障の除去可能性を容易に判定できる手法を提案した. 順序回路的検出不能故障には除去可能であるものと除去可能でないものがあるが, 除去可能な故障を

判定する手法についてはほとんど研究されていない.提案手法では回路の到達不能状態に着目し,到達不能状態から得られる順序回路的検出不能故障の除去可能性について考察した.順序回路的検出不能故障が除去可能でないのは故障回路が弱同期化可能でない場合であり,到達不能状態から得られる順序回路的検出不能故障についてはその状態が故障回路においても到達不能であれば,除去可能であることを証明した.ベンチマーク回路に対する実験結果より,実際にかなりの数の到達不能状態が存在していることが分かり,また,組合せ回路的検出不能故障の存在しない5つの回路に対しても,除去可能な順序回路的検出不能故障を求めることができ,冗長除去が有効に行われることが確認された.提案手法では処理に用いる状態を到達不能状態に限定するため,状態数の多い回路に対しても実用的な時間で処理を行うことができた.しかしながら,除去可能な検出不能故障のすべてが本手法で判定できるとは限らないため,本手法で簡単化を行った回路についてもまだ除去可能な部分が存在していると考えられる.それらの除去可能な部分についての考察を行うことでさらなる回路簡単化ができる可能性がある.

第6章では,第5章で提案した到達不能状態に基づく冗長除去手法をより効率的に行うためにＢＤＤを用いた到達不能状態の探索法を提案した.提案手法ではフリップフロップの入力関数を用いて設定できないフリップフロップの論理値の組を調べ到達不能状態を求めた.ＢＤＤは論理関数の処理を高速に行うことができるが,反面記憶容量などの問題から扱うことのできる関数のサイズが制限されている.大規模回路については全てのフリップフロップの入力関数を同時に扱うことができないため,フリップフロップ集合の分割を行う2つの手法を提案した.フリップフロップ集合の分割を行う手法の一つとして,各フリップフロップの依存する入力を調べ,その重複度の高いものが1つの集合に含まれるような分割を行った.また回路構造から各分岐点に着目しその分岐点からの経路の存在するフリップフロップが1つの集合に含まれるような分割法も提案した.提案手法により得られた到達不能状態を第5章の冗長除去手法に応用した実験結果から,構造解析によるフリップフロップ集合の分割を行った方がより多くの冗長部分を判定できることが,7つの回路において確認された.しかしながら,冗長除去を行うと縮退故障の除去可能性が変化することから,除去可能な検出不能故障が複数存在する場合には,除去を行う順序についての考察を行うことで,より多くの冗長部分が発見できる可能性があることも考えられる.

本論文においてはどの状態からも遷移させることのできない状態である到達不能状態にのみ着目し,順序回路的検出不能故障の除去可能性について述べたが,順序回路的検出不能故障の中

には到達不能状態とは関係ないものも存在するため,それらの検出不能故障の判定法や除去可能性についても考察することが必要であると思われる.到達不能状態の定義よりも条件を広くした弱同期化系列印加後に遷移させることのできない状態である無効状態について考察することで提案手法と同様に除去可能故障の判定ができると思われる.しかしながら,それでもすべての検出不能故障を判定することは困難な問題であり,今後の課題として残されている.

謝辞

本研究は，大阪大学大学院工学研究科応用物理学専攻において，樹下行三教授の御指導のもとで行われたものである．本研究を遂行するにあたり，終始御指導賜り，また，有益な議論および御助言いただきました樹下教授に心より感謝いたします．

本論文の作成に関し，詳細な御検討，貴重な御教示を頂きました大阪大学大学院工学研究科応用物理学専攻 豊田順一教授，伊東一良教授に深く感謝いたします．同じく本論文の作成において貴重な御教示を頂きました大阪大学大学院応用物理学専攻 増原宏教授，志水隆一教授，河田聡教授，中島信一教授，八木厚志教授，石井博昭教授，川上則雄教授，後藤誠一教授，岩崎裕教授，萩行正憲教授および同研究科物質・生命工学専攻 一岡芳樹教授に深く感謝いたします．

大阪大学大学院工学研究科応用物理学専攻 小松雅治助教授には，本研究を遂行するにあたり有益な議論，貴重な御助言を頂きました．小松雅治助教授に深く感謝いたします．また，同専攻板崎徳禎博士には本研究において有益な議論，貴重な御助言を頂き，特に計算機実験を行う際に懇切な御指導いただきました．板崎徳禎博士に深く感謝いたします．プログラム実装および計算機実験において御指導御助言頂きました同専攻樋上喜信博士，広島市立大学情報科学部 上田祐彰博士に深く感謝いたします．

本研究を通じて，九州工業大学情報工学部 梶原誠司助教授には様々な御助言御指導いただきました．梶原誠司助教授に深く感謝いたします．また，論理関数の処理プログラムについて御指導いただきました神戸大学工学部 沼昌宏助教授に深く感謝いたします．

樹下研究室の諸氏には一方ならぬ御支援，御協力を頂きました．ここに記して感謝いたします．

参考文献

[1] P. Ashar, S. Devadas, and A. R. Newton: Sequential Logic Synthesis, Kluwer Academic Publishers, 1992.

[2] 笹尾勤：論理設計－スイッチング回路理論－, 近代科学社, 1995.

[3] H. Higuchi and Y. Matsunaga, "A Fast State Reduction Algorithm for Incompletely Specified Finite State Machines," Proc. 33rd Design Automation Conf., 30.3, 1996.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," Proc. of the IEEE, vol. 78, no. 2, pp. 264-300, Feb. 1990.

[5] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli: Logic Minimization Algorithms for VLSI Synthesis, Kluwer Academic Publishers, 1984.

[6] S. Muraga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The Transduction Method - Design of Logic Networks Based on Permissible Functions," IEEE Trans. Computers, vol. C-38, no. 10, pp. 1404-1424, Oct. 1989.

[7] H. Sato, Y. Yasue, Y. Matsunaga, and M. Fujita, "Boolean Resubstitution with Permissible Functions and Binary Decision Diagrams," Proc. 27th Design Automation Conf., pp. 284-289, June 1990.

[8] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A Multi-Level Logic Optimization System," IEEE Trans. Computer-Aided Design, vol. 6, no. 6, pp. 1062-1081, Nov. 1989.

[9] M. Abramovici, M. A. Brewer, and A. D. Friedman: Digital Systems Testing and Testable Design, Computer Science Press, 1990.

[10] D. Bryan, F. Brglez, and R. Lisanke, "Redundancy Identification and Removal," MCNC Workshop on Logic Synthesis, May 1989.

[11] P. R. Menon and H. Ahuja, "Redundancy Removal and Simplification of Combinational Circuits," Proc. VLSI Test Symp., pp. 268-273, April 1992.

[12] M. Abramovici and M. A. Iyer, "One-Pass Redundancy Identification and Removal," Proc. Int'l Test Conf., pp. 807-815, Sept. 1992.

[13] R. Jacoby, P. Moceyunas, H. Cho, and G. Hachtel, "New ATPG Techniques for Logic Optimization," Proc. Int'l Conf. Computer-Aided Design, pp. 548-551, Nov. 1989.

[14] S. Kajihara, H. Shiba, and K. Kinoshita, "Removal of Redundancy in Logic Circuits under Classification of Undetectable Faults," Proc. 22nd Int'l Symp. Fault-Tolerant Computing, pp. 263-270, July 1992.

[15] 樹下行三，藤原秀雄：ディジタル回路の故障診断（上），工学図書，1983.

[16] 樹下行三，浅田邦博，唐津修：VLSIの設計II，岩波書店，1985.

[17] 藤原秀雄：コンピュータの設計とテスト，工学図書，1990.

[18] 菅野卓雄，堀口勝治：ULSI設計技術，電子情報通信学会，1993.

[19] L. A. Entrena and K. -T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal," IEEE Trans. Computer-Aided Design, vol. 14, no. 7, pp. 909-916, July 1995.

[20] 梶原誠司，渡辺克吉，樹下行三，"含意操作に基づいた論理回路の簡単化手法について"，電子情報通信学会技術研究報告，FTS, 1994.

[21] S. Devadas, H-K. T. Ma, and A. R. Newton, "Redundancies and Don't Cares in Sequential Logic Synthesis," J. Electronic Testing: Theory and Applications, vol. 1, pp. 15-30, Feb. 1990.

[22] I. Pomeranz and S. M. Reddy, "Classification of Faults in Synchronous Sequential Circuits," IEEE Trans. Computers, vol. 42, no. 9, pp. 1066-1077, Sep. 1993.

[23] H. Cho, G. D. Hachtel, and F. Somenzi, "Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration," IEEE Trans. Computer-Aided Design, vol. 12, no. 7, pp. 935-945, July 1993.

[24] V. D. Agrawal and S. T. Chakradhar, "Combinational ATPG Theorems for Identifying Untestable Faults in Sequential Circuits," Proc. European Test Conf., pp. 249-253, 1993.

[25] E. Macii and A. R. Meo, "A Test Generation Program for Sequential Circuits," J. Electronic Testing: Theory and Applications, vol. 5, pp. 115-119, 1994.

[26] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An Efficient Algorithm for Sequential Circuit Test Generation," IEEE Trans. Computers, vol. 42, no. 11, pp. 1361-1371, Nov. 1993.

[27] I. Pomeranz and S. M. Reddy, "The Multiple Observation Time Test Strategy," IEEE Trans. Computers, vol. C-41, pp. 627-637, May 1992.

[28] I. Pomeranz and S. M. Reddy, "On Removing Redundancies from Synchronous Sequential Circuits with Synchronizing Sequences," IEEE Trans. Computer-Aided Design, vol. 45, no. 1, pp. 20-32, Jan. 1996.

[29] K. -T. Cheng, "Redundancy Removal for Sequential Circuits without Reset States," IEEE Trans. Computer-Aided Design, vol. 12, no. 1, pp. 13-24, Jan. 1993.

[30] M. A. Iyer, D. E. Long, and M. Abramovici, "Surprises in Sequential Redundancy Identification," Proc. European Design and Test Conf., 2C-3, 1996.

[31] C. E. Leiserson and J. B. Saxe, "Retiming Synchronous Circuitry," Algorothmica, vol. 6, pp. 5-35, 1991.

[32] G. De Michelli, "Synchronous Logic Synthesis: Algorithms for Cycle-Time Minimization," IEEE

Trans. Computer-Aided Design, vol. 10, no. 1, pp. 63-73, Jan. 1991.

[33] K. Bartlett, G. Borriello, and S. Raju, "Timing Optimization of Multiphase Sequential Logic," IEEE Trans. Computer-Aided Design, vol. 10, no. 1, pp. 51-62, Jan. 1991.

[34] I. Karkowski and R. H. J. M. Otten, "Retiming Synchronous Circuitry with Imprecise Delays," Proc. 32nd Design Automation Conf., pp. 322-326, 1995.

[35] N. Shenoy and R. Rudell, "Efficient Implementation of Retiming," Proc. Int'l Conf. Computer-Aided Design, pp. 226-233, 1994.

[36] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Techniques," IEEE Trans. Computer-Aided Design, vol. 10, no. 1, pp. 74-84, Jan. 1991.

[37] S. Leimi, B. Kaminska, and E. Wagneur, "Resynthesis and Retiming of Synchronous Sequential Circuits," Proc. Int'l Symp. Circuits and Systems, pp. 1674-1677, 1993.

[38] D. Kagaris and S. Tragoudas, "Partial Scan with Retiming," Proc. 30th Design Automation Conf., pp. 249-254, June 1993.

[39] S. Dey and S. T. Chakradhar, "Retiming Sequential Circuits to Enhance Testability," Proc. 12th VLSI Test Symp., pp. 28-33, May 1994.

[40] S. T. Chakradhar and S. Dey, "Resynthesis and Retiming for Optimum Partial Scan," Proc. 31st Design Automation Conf., pp. 87-93, 1994.

[41] T. E. Marchok, A. El-Maleh, W. Maly, and J. Rajski, "A Complexity Analysis of Sequential ATPG," IEEE Trans. Computer-Aided Design, vol. 15, no. 11, pp. 1409-1423, Nov. 1996.

[42] H. J. Touati and R. K. Brayton, "Computing the Initial States of Retimed Circuits," IEEE Trans. Computer-Aided Design, vol. 12, no. 1, Jan. 1993.

[43] E. M. Sentovich and K. J. Singh, "SIS: A System for Sequential Circuit Synthesis," U. C. Berkeley Memorandum No. UCB/ERL M92/41, May 1992.

[44] R. E. Bryant, "Graph Based Algorithms for Boolean function manipulation," IEEE Trans. Computers, vol. C-35, no. 8, pp. 677-691, Aug. 1986.

[45] 石浦菜岐佐, "BDD とは," 情報処理学会誌, vol. 12, no. 1, pp. 6-12, Jan. 1993.

[46] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," Proc. 27th Design Automation Conf., pp. 40-45, 1990.

[47] M. Fujita, H. Fujisawa, and Y. Matsunaga, "Variable Ordering Algorithm for Ordered Binary Decision Diagrams and Their Evaluation," IEEE Trans. Computer-Aided Design, vol. 12, no. 1, pp. 6-12, Jan. 1993.

[48] M. Numa, "nBDD Package," pp. 1-12, Dec. 1994.

[49] C. Pixley, "A Theory and Implementation of Sequential Hardware Equivalence," IEEE Trans. Computer-Aided Design, vol. 11, no. 12, pp. 1469-1478, Dec. 1992.

[50] J-K. Rho, F. Somenzi, and C. Pixley, "Minimum Length Synchronizing Sequences of Finite State Machine," Proc. 33th Design Automation Conf., pp. 463-468, 1993.

[51] J. A. Wehbeh and D. G. Saab, "Initialization of Sequential Circuits and its Application to ATPG," Proc. 14th VLSI Test Symp., pp. 246-251, 1996.

[52] C. Pixley, V. Singhal, A. Aziz, and R. K. Brayton, "Multi-level Synthesis for Safe Replaceability," Proc. Intl. Conf. Computer-Aided Design, pp. 442-449, Nov. 1994.

[53] V. Singhal, C. Pixley, R. Rudell, and R. K. Brayton, "The Validity of Retiming Sequential Circuits," Proc. 32nd Design Automation Conf., pp. 316-321, June 1995.

[54] H. Yotsuyanagi, S. Kajihara, and K. Kinoshita, "Resynthesis for Sequential Circuits Designed with a Specified Initial State," Proc. 13th VLSI Test Symp., pp. 152-157, May 1995.

[55] K.-T. Cheng, "On Removing Redundancy in Sequential Circuits," Proc. 28th Design Automation Conf., pp. 164-169, 1991.

[56] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," Proc. Int'l Symp. Circuits and Systems, pp. 1929-1934, May 1989.

[57] K. -T. Cheng, "Redundancy Removal for Sequential Circuits without Reset States," IEEE Trans. Computer-Aided Design, vol. 12, no. 1, pp. 13-14, Jan. 1993.

[58] M. A. Iyer and M. Abramovici, "Sequentially Untestable Faults Identified without Search," Proc. Int'l Test Conf., pp. 259-266, Oct. 1994.

[59] D. E. Long, M. A. Iyer, and M. Abramovici, "Identifying Sequentially Untestable Faults Using Illegal States," Proc. 13th VLSI Test Symp., pp. 4-11, May 1995

[60] 四柳浩之,梶原誠司,樹下行三, "到達不能状態に基づく順序回路の冗長除去手法," 電子情報通信学会論文誌 D-I, (採録決定)

[61] H. Cho, G. D. Hachtel, and F. Somensi, "Redundancy Identification/Removal and Test Generation for Sequential Circuits Using Implicit State Enumeration," IEEE Trans. Computer-Aided Design, vol. 12, no. 7, pp. 935-945, July 1993.

[62] J. Park, C. Oh, and M. R. Mercer, "Improved Sequential ATPG Using Functional Observation Information and New Justification Methods," Proc. European Design and Test Conf., 5C-2, 1995.

[63] I. Hartanto, V. Boppana, and W. K. Fuchs, "Identification of Unsettable Flip-flops for Partial Scan and

Faster ATPG," Proc. Int'l Conf. Computer-Aided Design, pp. 63-66, 1996.

[64] H-C. Liang, C. L. Lee, and J. E. Chen, "Invalid State Identification for Sequential Circuit Test Generation," Proc. 5th Asian Test Symp., pp. 10-15, Nov. 1996.

発表論文一覧

論文誌

1. Hiroyuki Yotsuyanagi, Seiji Kajihara, and Kozo Kinoshita
"Retiming for Sequential Circuits with a Specified Initial State and Its Application to Testability Enhancement"
IEICE Trans. Inf. & Syst, vol.E78-D, No.7, pp.861-867, July 1995.
2. Hiroyuki Yotsuyanagi, Seiji Kajihara, and Kozo Kinoshita
"Synthesis of Sequential Circuits by Redundancy Removal and Retiming"
Journal of Electronic Testing: Theory and Applications, vol. 11, no. 1, pp. 81-92, Aug. 1997.
3. 四柳浩之，梶原誠司，樹下行三
”到達不能状態に基づく順序回路の冗長除去手法”
電子情報通信学会論文誌（和文論文誌 DI）　（採録決定）

国際会議

1. Hiroyuki Yotsuyanagi, Seiji Kajihara, and Kozo Kinoshita
"Resynthesis for Sequential Circuits Designed with a Specified Initial State"
13th IEEE VLSI Test Symposium, pp.152-157, May 1995.
2. Hiroyuki Yotsuyanagi, Seiji Kajihara, and Kozo Kinoshita
"Synthesis for Testability by Redundancy Removal Using Retiming"
25th Annual International Symposium on Fault-Tolerant Computing, pp.33-40, June 1995.

研究会など

1. 四柳浩之，梶原誠司，樹下行三
”特定の初期状態を持つ順序回路のリタイミングによる簡単化”
第３１回ＦＴＣ研究会資料，July 1994.
2. 四柳浩之，梶原誠司，樹下行三
”特定の初期状態を持つ順序回路のリタイミングによる簡単化”
電子情報通信学会技術研究報告，FTS94-78, pp.33-40, Feb. 1995.
3. 四柳浩之，梶原誠司，樹下行三
”リタイミングと冗長除去を用いた順序回路の簡単化”
電子情報通信学会技術研究報告，FTS95-56, pp.39-46, Oct. 1995.
4. 四柳浩之，梶原誠司，樹下行三
”到達不能状態を用いた順序回路の冗長除去”
1996 年電子情報通信学会総合大会，SD-2-5, pp.325-326, Apr. 1996.
5. 四柳浩之，梶原誠司，樹下行三
”到達不能状態を用いて簡単化された順序回路のテスタビリティについて”
第３５回ＦＴＣ研究会資料，July 1996.