縮退故障とパス遅延故障のためのプロセッサの命令レベル自己テスト法

井上美智子† 神戸 和子† ヴィレンドラ シン† 藤原 秀雄†

Software-Based Self-Test of Processors for Stuck-at Faults and Path Delay Faults

Michiko INOUE†, Kazuko KAMBE†, Virendra SINGH†, and Hideo FUJIWARA†

あらまし プロセッサの命令レベル自己テストは遅延やハードウェアオーバヘッドを伴わずに実動作テストを実現する手法として注目を集めている．本論文では，縮退故障，パス遅延故障に対し，自己テストのためのテストプログラム自動生成法を提案する．提案手法では，対象故障を含む部分回路に対するゲートレベルテスト生成とレジスタ転送レベル記述や命令セットアーキテクチャといった高位情報を利用する命令列選択を組み合わせてテストプログラムを合成する．部分回路に対するゲートレベルテスト生成には入出力に関する制約を考慮する必要がある．本論文では，命令実行グラフ，テストプログラムテンプレートを用いた効率の良い制約抽出法，制約抽出過程での冗長故障判定法を提案し高い故障検出効率を達成する．

キーワード プロセッサ，自己テスト，縮退故障，パス遅延故障，テストプログラム

1. まえがき

高性能高機能なハイエンドプロセッサには実動作テストが不可欠となっている．スキャン設計は高品質なテストを短いテスト生成時間で実現するが，遅延及びハードウェアオーバヘッドを伴う，実動作速度テストが困難であるといった問題点がある．ハードウェア BIST（組込み自己テスト）は実動作テストを可能にするが，面積オーバヘッドが大きい，回路の設計変更が必要な場合がある，更に，擬似ランダムテストパターンによる過剰な消費電力などが問題となっている．

プロセッサの命令レベル自己テストは，実動作テスト可能なテスト手法として注目されている．この手法では，プロセッサは命令列であるテストプログラムを実行することによって自己テストを行う．テスト容易化設計を必要としないため，遅延オーバヘッド，面積オーバヘッドを伴わず，過剰な消費電力も必要としない．プロセッサの命令レベル自己テストのための多くの手法が提案されている [1]〜[16].

最近では，モジュール単体に対するテストパターンを利用してテストプログラムを生成することで縮退故障や遅延故障といった構造故障モデルを対象としたテストプログラム生成法が提案されている．

Krantis ら [6] は，ALU や加算器など規則的な構造をもつ演算モジュールに対して有効なテストパターン集合，及びそのテストパターン集合を利用するテストプログラムを提案している．

プロセッサの各モジュールに対し，ゲートレベル ATPG（自動テストパターン生成ツール）を用いて生成したテストパターンを利用することで高いテスト品質を得る手法も提案されている [7]〜[11]．命令レベル自己テストでは命令列であるテストプログラムを用いてプロセッサにテストを印加する．よって，モジュールに印加可能な値若しくは値の系列や，外部出力で観測可能となるモジュールの出力タイミングなどには制限がある．そのため，これらの手法ではこの制限をモジュール単体へゲートレベル ATPG を適用するときの制約としてとらえ，モジュール単体に対し制約のもとでゲートレベルテスト生成を行う．最後に，生成されたテストパターンをテストプログラムに変換する．

Lai ら [7]〜[9] はパス遅延故障を対象とした手法を提案している．提案手法では，まず，すべての命令対に対して，それらが連続して実行される状況での制御

† 奈良先端科学技術大学院大学情報科学研究科，生駒市
Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–0192 Japan

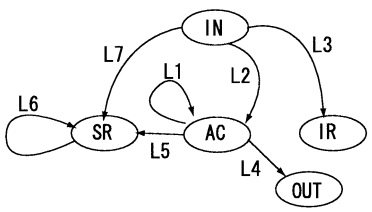
信号及びデータ信号を RTL 記述から求め，連続するサイクルでの値の対を制約として抽出する．次に，テスト対象パスを含むゲートレベル組合せ回路に対して，制約に基づく冗長故障判定及びテスト生成を行い，最後に，テスト生成によって得られたテストパターンの正当化及びテスト応答の観測を行うための命令列を生成する．しかし，制約抽出プロセスでは 2 命令しか考慮していないために制約が正確とはいえず，生成されたテストパターンに対し命令列の生成に失敗する場合があり，十分な故障検出率を達成しているとはいえない．この手法では，データパス部とコントローラ部の双方に対する手法が提案されているが，実験では，データパス部だけしか評価されていない．

Chen ら [11] は，テストプログラムテンプレートを利用して制約を求める手法を提案している．テストプログラムテンプレートとは，オペランドの値が未決定のテストプログラムであり，テスト対象モジュールに対し，テストパターンの正当化及びテスト応答の観測を行う命令列からなる．この手法では，テンプレートに対し，いくつかのランダムパターンをオペランドに与えたシミュレーション結果から回帰解析により制約を抽出し，抽出した制約のもとでモジュール単体に対してテスト生成を行う．最後に，得られたテストパターンからテンプレートのオペランドを求めてテストプログラムを生成する．提案手法では，テンプレート生成法にも触れているが，テスト対象モジュールの入力空間として，テストパターンをモジュールに印加する命令のオペランドのみを考慮するといった単純なテンプレート生成法を採用している．そのため，先行する命令によって設定されたレジスタの値など，命令のオペランドには陽に現れないが命令の実行で用いられる値などを考慮することができない．また，制約抽出に回帰解析を利用するため正確な制約を抽出しているともいえない．実験結果では，組合せモジュールに対しては良い結果が得られているが，順序モジュールに対するテストプログラム生成は行われていない．

本論文では，縮退故障とパス遅延故障に対し，プロセッサの命令レベル自己テストのためのテストプログラム生成法を提案する．縮退故障に対しては，各モジュール単体に対するテストパターンを求めテストプログラムに変換する．モジュールの入出力を考慮したテストプログラムテンプレート生成法，テンプレートからの制約抽出法，テストプログラム合成法を提案する．制約抽出法は，組合せモジュール及び順序モジュールに対しそれぞれ提案し，順序モジュールに対しても効率良くテストプログラムが合成できることを示す．パス遅延故障に関しては，テスト対象パスを含む組合せ回路に対してテストパターンを求めテストプログラムに変換する．テスト生成の対象となる組合せ回路に対する制約抽出法を提案する．また，ゲートレベル情報を必要としない冗長故障判定法も提案する．提案法は，故障モデルにかかわらず，レジスタ間のデータ転送と命令の関係を表す命令実行グラフ（Instruction Execution グラフ，IE グラフ）を利用する効率の良い探索を行う．実験結果では，縮退故障，遅延故障それぞれに対し，高い故障検出効率が得られることを示す．

2. 命令実行グラフ

縮退故障とパス遅延故障に対するテストプログラム生成では命令実行グラフ（Instruction Execution グラフ，以降，IE グラフ）[13], [14] を用いる．IE グラフはレジスタ間のデータ転送と命令及びコントローラの状態との関係を表す有向グラフで，レジスタ間のデータ転送を表す S グラフ [1], [2] を拡張したものである．頂点はレジスタ及び外部入出力に対応し，辺は頂点間のデータ転送を表す．ただし，レジスタファイル内のレジスタのように命令に対して同じ振舞いをするレジスタは等価なレジスタとして一つの頂点で表す．各辺は，そのデータ転送がどの命令のどの状態で起こるかを表すラベルをもつ．図 1 に IE グラフの例を示す．IE グラフは，命令セットアーキテクチャと RTL （レジスタ転送レベル）記述からなる高位情報から生成され，後述するテストプログラムテンプレート生成，パ



L1: {[s_3, I_{13-14}], [s_3, I_{16-17}], [s_6, I_{2-4}]}
L2: {[s_6, I_{1-4}]}
L3: {[s_2, I_{1-23}]}
L4: {[s_6, I_6]}
L5: {[s_3, I_{16-17}], [s_6, I_{2-4}]}
L6: {[s_3, I_{15}]}
L7: {[s_6, I_{1-4}]}

図 1 IE グラフ
Fig. 1 IE graph.

ス遅延故障に対する制約抽出及び冗長判定などに利用する．

3. 縮退故障に対するテストプログラム生成法

3.1 概　　要

RTL 記述では，プロセッサは複数のモジュールから構成される．縮退故障に対しては，モジュール単体に対してゲートレベル ATPG を適用し，得られたテストパターンを利用してテストプログラムを合成する．本論文では，モジュールとして ALU などの組合せモジュール，及びコントローラなどの順序モジュールを考える．モジュール単体に対するテストパターン（順序モジュールの場合はテスト系列）及びテスト応答は，命令列を用いて正当化及び観測する．よって，モジュール単体へテストパターンとして用いることができる値，テスト応答として観測できる値には制約がある．一般に，モジュール単体への入出力に関する制約を正確に求めるのは困難な問題である．制約が正確でない場合，モジュールに対して生成したテストパターンがテストプログラムに変換できない場合がある．この場合，テストプログラムが存在しない故障，すなわち，命令レベル自己テストの下で冗長である故障にモジュール単体へのテストパターンを求めてしまうだけでなく，実際には検出可能である故障に対してテストプログラムに変換できないテストパターンを求めてしまい故障検出効率が低下するなどの問題が生じる．

本論文では，テストプログラムテンプレートを用いて正確な制約を抽出する．テンプレート生成を利用するテストプログラム生成のフローを図 2 に示す．

提案手法では，まず，各レジスタと命令の対に対し，以下の可制御性，可観測を表す述語の評価を行う．

- $C_g(i,r)$：命令 i で終わる命令列でレジスタ r に任意の値を設定できる．
- $C_p(i,r)$：命令 i で終わる命令列でレジスタ r の一部のビットに任意の値を設定できる．
- $O_g(i,r)$：命令 i で始まる命令列でレジスタ r の値を観測できる．

次に，各モジュールに対し，テンプレートを生成する．テンプレートとは，オペランドの値が未決定の命令列であり，テスト対象モジュールの入力を正当化し出力を観測する命令列からなる．次に，生成されたテンプレートに対しモジュールの入出力に関する制約を求める．制約は，制約回路と呼ぶ回路記述で表現する [15]．モジュールの入出力にそれぞれ制約回路を付加し ATPG を適用する．得られたテストパターンからテンプレートのオペランドの値を求めテストプログラムに変換する．最後にテストプログラムを用いてプロセッサ全体を対象に故障シミュレーションを行い故障検出率を評価する．これらの処理を，各モジュールに対して，あらかじめ設定した故障検出率に到達するかまたはテンプレートが生成できなくなるまで繰り返す．

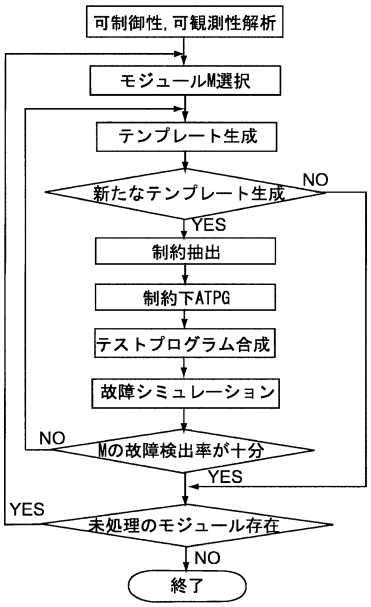


図 2　テストプログラム生成法
Fig. 2　Test program generation.

3.2 テンプレート生成

テストプログラム生成に有効なテンプレートを生成するために，モジュールの隣接レジスタの入力空間，出力タイミングを考慮してテンプレートを生成する．ここで，隣接レジスタとはテスト対象モジュールに組合せ回路だけを介して接続するレジスタである（図 3 参照）．入力隣接レジスタの値は対象モジュールの入力空間を決定し，モジュールの出力を観測するには，出力隣接レジスタに値を取り込むタイミングを考慮する必要がある．

テンプレート生成は以下の順序で行う．(1) テスト対象モジュールにテストパターンを印加する命令を選択する，(2) 入力隣接レジスタに値を正当化するための命令列を選択する，(3) 出力隣接レジスタの値を外部出力に伝搬する命令列を選択する．命令の選択には，

IE グラフ及び先に評価した可制御性，可観測性を利用する．テスト対象モジュールにテストパターンを印加する命令は，少なくとも一つの出力隣接レジスタに値を取り込む命令から選択する．このとき，可観測性の高いレジスタに値を取り込む命令の選択を優先する．次に，出力隣接レジスタに値を取り込むサイクルでの入力隣接レジスタの値空間を考慮して正当化のための命令列を選択する．一般に，1 命令の実行では複数のレジスタへのデータ転送を行う，しかし，1 命令の実行ですべての入力隣接レジスタにデータ転送を行えるとも限らない．提案手法では，まず，各入力隣接レジスタへデータ転送を行う命令を選択する．このとき，あるレジスタへ設定した値を，他のレジスタにデータ転送を行う命令が上書きしてしまう可能性がある．すなわち，命令の実行順序には依存関係が生じる．また，入力隣接レジスタにデータ転送を行う命令が別のレジスタの値を必要とするかもしれない．すなわち，ある命令に先行して別の命令を実行する必要がある．提案手法では，選択した複数の命令の実行順序に関する依存関係を半順序関係として抽出し，外部入力から入力隣接レジスタの値を正当化するために必要なすべての命令の選択後に，トポロジカルソートによって半順序関係を全順序関係に変換することで命令列を生成する．最後に，出力隣接レジスタに取り込まれた値を外部出力まで伝搬する命令列を選択する．

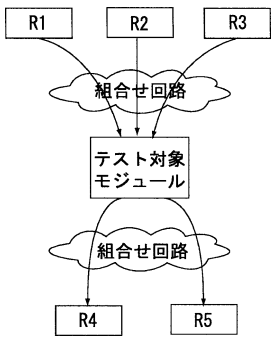


図 3　テスト対象モジュールと隣接レジスタ
Fig. 3　Module under test and adjacent registers.

入力隣接レジスタに値を正当化するための命令選択時には，選択した命令による入力隣接レジスタの値空間を考慮し，より広い値空間を被覆する命令の組合せを選択する．また，一つのテンプレートだけでは十分な値空間を被覆できない場合は，テンプレートを複数生成してより広い値空間を被覆できるようにする．提案手法では，レジスタの可制御性を考慮して，効率良く複数のテンプレートを生成する．テンプレート生成手法の詳細については，文献 [15] を参照されたい．

3.3　制約抽出とテストプログラム合成

生成されたテンプレートに対し ATPG のための制約を抽出する（図 4 参照）．テンプレートを固定するとプロセッサの動作を限定したことになる．例えば，図 4 の例では，テンプレートの先頭の命令 LDA は外部のメモリからアキュムレータ AC に値をロードする命令で，この命令の実行時には，ALU は左の入力端子の値をそのまま出力する．提案手法では，プロセッサの RTL 記述からテンプレートで指定した動作を抽

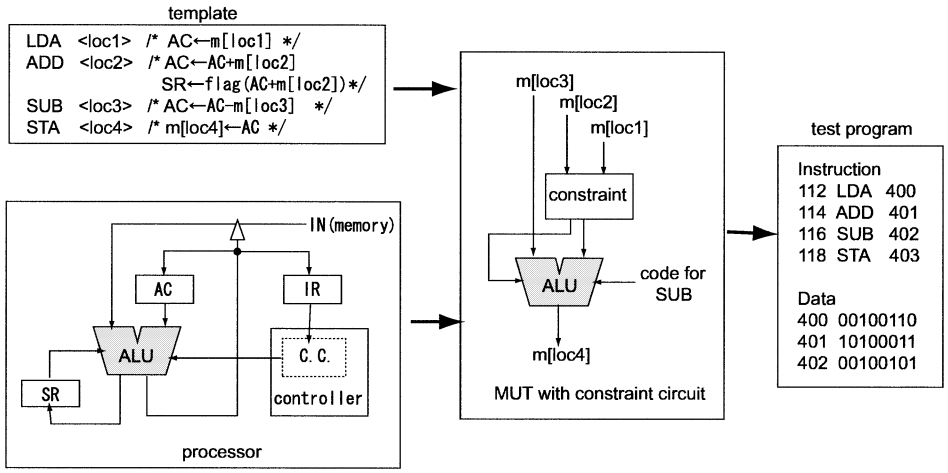


図 4　制約抽出とテストプログラム合成
Fig. 4　Constraint extraction and test program synthesis.

出して制約回路を生成する．テスト対象モジュールが組合せ回路の場合，制約回路では，テストパターンを印加するサイクルでのテスト対象モジュールの入出力のみを表せばよいので，制約回路は組合せ回路となる．組合せモジュールの場合，モジュールの入力制約は入力空間のみであるので，この制約を入力空間制約と呼ぶ．

テスト対象モジュールが順序回路の場合は，順序ATPG を用いてモジュール単体のテスト生成を行う．順序モジュールに対しては，組合せモジュール同様にテンプレートを生成するが，テンプレート生成手順 (1) で選択した命令を実行する全サイクルに対し制約を抽出する．すなわち，モジュールの制約は入力空間制約の系列となる．これを，入力時相空間制約と呼ぶ．入力時相空間制約を表す制約回路は順序回路となる．

テスト対象モジュールに制約回路を付加した回路にATPG を適用して，モジュール単体へのテストパターンを求める．制約回路の入出力は外部入出力または定数値であるので．モジュール単体に対するテストパターン（順序モジュールの場合はテスト系列）として外部入力の値が求まる．これらの値はテンプレートの各命令で用いるオペランドに対応する．よって，生成されたテストパターンから容易にテストプログラムが合成できる．

3.4 冗長故障判定

コントローラに対して，入力隣接レジスタとコントローラ及び両者を結ぶ組合せ回路からなる回路を用いて冗長故障を判定する [16]．この回路では，隣接レジスタのロード信号はコントローラから制御される．この冗長故障判定のための回路は，コントローラ本来の動作を包含した振舞いが可能であるので，この回路にATPG を適用して冗長と判定された故障は，テストプログラムが存在しない冗長故障であると判定できる．

3.5 評 価 実 験

Parwan プロセッサ [17] に対し提案法を適用し手法の有効性を示す．Parwan はアキュムレータベースの 8 ビットアーキテクチャで，12 ビットのアドレスバスと 23 の命令をもつ．プロセッサの RTL-VHDL 記述を Design Compiler（Synopsys）で論理合成し，モジュール単体のテスト生成には TestGen（Synopsys）を用いた．また，得られたテストプログラムとプロセッサの RTL 記述に対し Scirocco（Synopsys）を用い RTL 論理シミュレーションを行いプロセッサに対する入力系列を求め，ゲートレベルで故障シミュレーションを行った．

表 1 に，モジュール単体に対するテスト生成結果，表 2 にテストプログラムの故障シミュレーション結果を示す．モジュール単体に対するテスト生成は，二つの組合せモジュール ALU，SHU と二つの順序モジュール CTRL, PC に対して行った．表 1 には，生成したテンプレート数，故障検出率に貢献したテンプレート数，テストパターン数，故障数，検出故障数，及び故障検出効率を示す．組合せモジュールに対しては，高い故障検出効率を得ることができ，順序モジュールに対しても 90%前後の故障検出効率を達成した．これら 4 モジュールに対するテストパターンからテストプログラムを合成し，全モジュールを対象として故障シミュレーションを行った．表 2 の冗長故障は，組合せ冗長故障にコントローラに対する冗長故障判定の結果を加えたものである．SHU, CTRL, PC の 3 モジュールに対しては，単体テストに比べ高い故障検出効率を達成したが，ALU に対しては単体テストに比べ低い故障検出効率となった．これは，ALU に対するテストパターンの正当化及びテスト応答の観測のためのデータ転送が ALU を通過することにより起こる故障のマスクが原因ではないかと考えられる．また，モジュール単体にはテスト生成していないモジュール

表 1 モジュール単体テスト生成（縮退故障，Parwan）
Table 1 Test generation for modules. (stuck-at faults, Parwan)

モジュール	テンプレート		テストパターン	故障	検出故障	故障検出効率 (%)
	生成	検出				
ALU	276	12	77	886	881	99.44
SHU	276	11	49	264	258	97.73
CTRL	299	19	33	654	606	92.73
PC	559	3	50	494	436	88.26

表 2 テストプログラムの故障シミュレーション（縮退故障，Parwan）
Table 2 Fault simulation of test program. (stuck-at faults, Parwan)

モジュール	故障	検出故障	冗長判定故障	故障検出率 (%)	故障検出効率 (%)
ALU	886	856	2	96.61	96.84
SHU	264	257	2	97.35	98.11
CTRL	654	540	75	82.57	94.04
PC	494	490	1	99.19	99.39
AC	126	123	0	97.62	97.62
IR	126	123	0	96.09	96.09
MAR	176	164	12	93.18	100.00
SR	66	63	1	95.45	96.97
その他	414	283	97	68.36	91.79
計	3,208	2,899	190	90.37	96.29

に対しても，高い故障検出効率を達成し，プロセッサ全体で 96.29% という高い故障検出効率を達成した．

4. パス遅延故障に対するテストプログラム生成法

4.1 概　　要

遅延故障に対しては，最も一般的な遅延故障のモデルであるパス遅延故障を対象とする．パス遅延故障は，フリップフロップ（または外部入力）からフリップフロップ（または外部出力）までのパスの遅延にかかわる故障で，テスト対象パスは複数のモジュールを通過する場合もある．パス遅延故障に対しては，テスト対象パスを含む組合せ回路に対してゲートレベル ATPG を適用し，得られたテストパターンからテストプログラムを合成する．パス遅延故障のテスト生成には，一般に以下の二つの問題がある．(1) テスト対象パスの個数が膨大でテスト生成時間が大きい，(2) 検出可能な故障が少なく，故障検出と同様に冗長故障判定が重要である．提案手法では，RTL でパスを考慮することで，効率良く ATPG のための制約を抽出する．また，制約が抽出できないパスを冗長故障と判定することでゲートレベル ATPG の対象となる故障数を削減する．

パス遅延故障に対するテストプログラム生成法では，データ転送を行うパスと制御信号を転送するパスをそれぞれ考慮する．RTL 記述では，プロセッサはデータパスとコントローラからなる．データ転送を行うパスとは，データパス内部のパスを指し，制御信号を転送するパスとは，コントローラ内部及びコントローラ及びデータパスの双方にかかわるパスを指す．

4.2 データ転送を行うパス

データ転送を行うパスに対しては，IE グラフを用いて ATPG に対する制約を抽出する．IE グラフの有向辺は RTL 記述でのレジスタ間のデータ転送を表し，ゲートレベル記述における複数のパスに対応する．提案法では，IE グラフの各有向辺に対し，ATPG のための制約を抽出しテスト生成を行う．

あるパスをテストプログラムを用いてテストするには，あるサイクルでのパスの始点での値の変化をパスを通じて伝搬させ次のサイクルでパスの終点で取り込む必要がある．まず，IE グラフの有向辺に対応するパスがテストプログラムでテスト可能であるための必要条件を示す．以下の補題において，IE グラフの有向辺がラベル $[s, I]$ をもつとは，命令 I の状態 s で対応するデータ転送が起こることを意味する．

［補題 1］ IE グラフの有向辺 (R_i, R_o) に対応するパスがテストプログラムを用いてテスト可能であるとき，以下のいずれかが成り立つ．

（1） R_i は外部入力である．

（2） R_i を終点とする有向辺がラベル $[s_1, I_1]$ を，有向辺 (R_i, R_o) がラベル $[s_2, I_2]$ をもち，命令 I_1 の状態 s_1 と命令 I_2 の状態 s_2 は連続した 2 サイクルで起こり得る． □

本手法では，まず，IE グラフの各有向辺に対し，補題 1 を満たすかどうかを判定する．有向辺が補題を満たさない場合，対応するすべてのパスをテストプログラムでテストできない冗長故障であると判定する．補題 1 の判定は，IE グラフ及び RTL 記述のみから判定可能である．すなわち，本手法では，高位レベルの記述のみから冗長故障を識別することが可能である．また，補題を満たす IE グラフの有向辺に関しては，補題を満たす命令及び状態の組合せに対してのみ ATPG のための制約を抽出すればよく，効率の良い制約抽出を行うことができる．

図 1 の IE グラフでは，便宜上有向辺のラベルを記号 $L_1, L_2, \ldots, L_6$ で表示し，完全なラベルを図の下に示している．AC から AC への有向辺はラベル $[s_3, I_{13}]$,$[s_3, I_{14}]$,$[s_3, I_{16}]$,$[s_3, I_{17}]$,$[s_6, I_2]$,$[s_6, I_3]$,$[s_6, I_4]$ をもつ．また，コントローラの状態遷移より，命令 $I_{13}, I_{14}, I_{16}, I_{17}$ の状態 s_3 は同じ命令の状態 s_2 に続いて起こり，命令 I_2, I_3, I_4 の状態 s_6 は同じ命令の状態 s_4 または s_5 に続いて起こることが分かる．ところが，AC を終点とするどの有向辺もこれらのラベルをもたない．すなわち，AC から AC へのパスはテストプログラムを用いてテスト可能ではない冗長故障と識別できる．

一方，外部入力である IN から AC への有向辺はラベル $[s_6, I_1], [s_6, I_2], [s_6, I_3], [s_6, I_4]$ をもつ．また，コントローラの状態遷移により，命令 I_1, I_2, I_3, I_4 の状態 s_6 は同じ命令の s_4 または s_5 に続いて起こることが分かる．よって，IN から AC へのパスをテストするには，命令 I_1, I_2, I_3, I_4 の状態 s_4, s_6 または s_5, s_6 の連続する 2 サイクルを用いてテスト対象パスに遅延故障を検出するための 2 パターンを印加する必要がある．そこで，各命令の該当する 2 サイクルに対し組合せ ATPG への制約を抽出しテスト生成を行う．表 3 に命令 I_3 の状態 s_4, s_6 に対して抽出した制約を示す．命令 I_3 は，IN, AC の値をオペランドとして ALU を用いて演算を行い，結果を AC に取り込

表 4　テスト生成結果（パス遅延故障）
Table 4　Results on test generation. (path delay fault)

	Parwan						DLX					
	データ転送パス			制御信号転送パス			データ転送パス			制御信号転送パス		
	Robust	NR	FS	Robust	NR	FS	Robust	NR	FS	Robust	NR	FS
総故障	10434			348724			529812			1468822		
冗長故障 (RTL)	3902			162812			89848			0		
冗長故障 (ATPG)	6376	4879	3914	185505	183495	182392	420610	405040	395640	1453676	1426527	1356087
検出故障	156	1653	2618	407	2417	3520	19354	34924	44324	15146	42295	112735
故障検出率 (%)	1.50	15.84	25.09	0.12	0.69	1.01	3.65	6.59	8.37	1.03	2.88	7.68
故障検出効率 (%)	100	100	100	100	100	100	100	100	100	100	100	100
CPU (ATPG)	3 min. 41 sec.			3 hours 24 min.			1 hour 32 min.			15 hours 18 min.		

表 3　IN から AC へのパスに対する制約（命令 I_3）
Table 3　Constraints for a path from IN to AC. (instruction I_3)

状態	$ALUctrl$	IN	AC
s_4	000	xxxxxxxx	xxxxxxxx
s_6	101	xxxxxxxx	hhhhhhhh

む命令である．そのため，ALU への制御信号である $ALUctrl$ の値が制約として抽出される．表 3 において，x は任意の値を取り得ることを表し，h は直前のサイクルと同じ値であることを表す．

制約を抽出した IE グラフの有向辺に対し，各制約を組合せ ATPG に適用し，有向辺に対応するゲートレベルのパス集合を故障リストとして，遅延故障を検出するための 2 パターンテストを求める．制約に対応する命令から，テスト対象パスに 2 パターンを印加する命令が求まる．更に，生成された 2 パターンテストから 2 パターンを印加する命令のオペランドの値が求まる．これらオペランドの値を正当化する命令列，及び，テスト応答を観測するための命令列を求め，テストプログラムを生成する．

4.3　制御信号を転送するパス

制御信号を転送するパスは，コントローラ内部のパス及びコントローラとデータパス双方にかかわるパスである．コントローラは命令レジスタなどのレジスタ及び外部入力を入力とし，状態レジスタと合わせて次状態及びデータパスへの出力を決定する．テストプログラムを用いてこれらのパスに印加できる 2 パターンはコントローラの状態遷移によって制限される．本手法では，コントローラの状態遷移とパスの関係から，冗長故障判定及び制約抽出を行う．制御信号を転送するパスの始点及び終点となるレジスタのとり得る値は，命令やその状態によって特定の値をとる場合が多い．まず，レジスタのあるビットから別のレジスタのあるビットまでのパスがテストプログラムによってテスト可能である必要条件を考える．

［補題 2］　レジスタ R_1 のビット i からレジスタ R_2 のビット j までのパスがテストプログラムを用いてテスト可能であるとき，コントローラの状態遷移 (s_m, s_n), (s_n, s_q) が存在して，s_m から s_n への状態遷移時にビット i の値が変化し，s_n から s_q への状態遷移時にビット j の値が変化する．　□

制御信号を転送するパスに対しては，コントローラの状態遷移，及び，状態割当情報から，補題 2 を満たす状態遷移が存在するパスを抽出し，それらのパスのみをゲートレベルテスト生成の対象とする．テスト生成の対象となるパスに対しては，コントローラの入力及び状態遷移を制約として，ゲートレベル ATPG を適用する．終点がデータパスのレジスタであるパスに対しては，更に IE グラフからデータ転送を行うパスと同様に制約を抽出する．補題を満たさないパスは冗長故障であると識別できる．補題 2 では，更にパスの終点での値の変化が立上りであるか立下りであるかを考慮することによって，テスト可能なパス遅延故障を限定できる．2 パターンテストを生成したパスに対しては，生成したパターンから 2 パターンを印加する命令及びそのオペランドが求まる．更に，オペランドの値を正当化する命令列，及び，テスト応答を観測するための命令列を求め，テストプログラムを生成する．

4.4　評 価 実 験

Parwan プロセッサ [17]，及び DLX プロセッサ [18] に対して，提案法を適用し手法の有効性を示す．論理合成には DesignCompiler（Synopsys）を用い，パス遅延故障に対し制約付きテスト生成を行うゲートレベル APTG を実装しテスト生成を行った．パス遅延故障として，ロバストテスト可能故障（Robust），ノンロバストテスト可能故障（NR），機能的活性化可能故障（FS）[19] に対しテスト生成を行った結果を表 4 に

示す．これら 3 種類の故障に対しては，NR は Robust を含まないと定義するものがあるが，ここでは，NR は Robust を含み，FS は Robust, NR を含む故障であるとして冗長故障数，検出故障数を示す．

図 4 において，冗長故障（RTL）は制約抽出時に高位情報のみから冗長と判定した故障数で，ATPG の対象とならない故障数である．また，冗長故障（ATPG）は，ATPG が冗長と判定した故障数を示す．多くの故障を高位情報のみから冗長と判定でき，また，ATPG によるテスト生成ではすべての故障に対し処理を打ち切ることなく 100%の故障検出効率を達成した．

5. むすび

縮退故障とパス遅延故障に対して，プロセッサの命令レベル自己テストのためのテストプログラム生成法を提案した．提案法は，命令セットアーキテクチャ，プロセッサの RTL 記述といった高位情報を利用して効率良く自己テストのためのテストプログラムを生成する．縮退故障，パス遅延故障に対しともに高い故障検出効率が得られることを実験で評価した．

謝辞 本研究は一部，半導体理工学研究センター（STARC）との共同研究，及び日本学術振興会科学技術研究費補助金 基盤研究 B(2) 15300018 の研究助成による．

文　献

[1] S.M. Thatte and J.A. Abraham, “Test generation for microprocessors,” IEEE Trans. Comput., vol.C-29, no.6, pp.429–441, 1980.

[2] D. Brahme and J.A. Abraham, “Functional testing of microprocessors,” IEEE Trans. Comput., vol.33, no.6, pp.475–485, 1984.

[3] J. Shen and J. Abraham, “Native mode functional test generation for processors with applications to self-test and design validation,” Proc. International Test Conference 1998, pp.990–999, 1998.

[4] K. Batcher and C. Papachristou, “Instruction randomization self test for processor cores,” Proc. 17th VLSI Test Symposium, pp.34–40, 1999.

[5] F. Corno, C. Cumani, M.S. Reorda, and G. Squillero, “Fully automatic test program generation for microproessor cores,” Proc. Design, Automation & Test in Europe, 2003, pp.1006–1011, 2003.

[6] N. Krantis, A. Paschalis, D. Gizopoulos, and Y. Zorian, “Instruction-based self-testing of processor cores,” J. Electron. Test., Theory Appl., vol.19, pp.103–112, 2003.

[7] W.-C. Lai, A. Krstic, and K.-T. Cheng, “Test program synthesis for path delay faults in microprocessor cores,” Proc. International Test Conference 2000, pp.1080–1089, 2000.

[8] W.-C. Lai and K.-T. Cheng, “Instruction-level DFT for testing processor and IP cores in system-on-a-chip,” Proc. Design Automation Conference (DAC 01), pp.59–64, 2001.

[9] A. Krstic, L. Chen, W.-C. Lai, K.-T. Cheng, and Sujit Dey, “Embedded software-based self-test for programmable core-based designs,” IEEE Des. Test Comput., vol.19, no.4, pp.18–27, 2002.

[10] L. Chen and S. Dey, “Software-based self-testing methodology for processor cores,” IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.20, no.3, pp.369–380, 2001.

[11] L. Chen, S. Ravi, A. Raghunath, and S. Dey, “A scalable software-based self-test methodology for programmable processors,” Proc. 40th Design Automation Conference, pp.548–553, 2003.

[12] A. Almukhaizim, P. Petrov, and A. Orailoglu, “Faults in procesor control subsystems: Testing correctness and performance faults in the data prefetching unit,” Proc. 10th Asian Test Symposium, pp.319–324, 2001.

[13] V. Singh, M. Inoue, K.K. Saluja, and H. Fujiwara, “Instruction-based delay fault self-testing of processor cores,” Proc. 17th International Conf. on VLSI Design, pp.933–938, 2004.

[14] V. Singh, M. Inoue, K.K. Saluja, and H. Fujiwara, “Delay fault testing of processor cores in functional mode,” IEICE Trans. Inf. & Syst., vol.E88-D, no.3, pp.610–618, March 2005.

[15] K. Kambe, M. Inoue, and H. Fujiwara, “Efficient template generation for instruction-based self-test of processor cores,” Proc. 13th Asian Test Symposium, pp.152–157, 2004.

[16] M. Inoue, K. Kambe, N. Hoashi, and H. Fujiwara, “Instruction-based self-test for sequential modules in processors,” Digest of Papers 5th International Workshop on RTL and High-level Testing, pp.109–114, 2004.

[17] Z. Navabi, VHDL: Analysis and Modeling of Digital Systems, McGraw-Hill, New York, 1997.

[18] J.L. Hennessy and D.A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, San Francisco, 1996.

[19] A. Krstic and K.-T. Cheng, Delay Fault Testing for VLSI Circuits, Kluwer Academic Publishers, Boston, 1998.

（平成 16 年 11 月 30 日受付）

井上美智子　（正員）

昭 62 阪大・基礎工・情報卒．平元同大大学院博士前期課程了．同年（株）富士通研究所入社．平 7 阪大大学院博士後期課程了．同年奈良先端大・情報科学・助手．現在同助教授．分散アルゴリズム，グラフ理論，テスト容易化設計，高位合成に関する研究に従事．工博．IEEE，情報処理学会，人工知能学会各会員．

神戸　和子　（正員）

昭 55 広大・総合科学・総合科学卒．同年（株）シャープ入社．平 11 奈良女子大大学院修士課程了．平 14 同大学院博士後期課程了．現在奈良先端大・情報科学・産学官連携研究員．自動並列化コンパイラ，プロセッサの命令レベル自己テストに関する研究に従事．理博．情報処理学会会員．

ヴィレンドラ　シン

1994 マラビア工科大・電子通信工卒．1996 同大大学院修士課程了．現在 Central Electronics Engineering Research Institute（India）研究員．2002 より奈良先端大・情報科学・博士課程在学中．プロセッサの設計検証及びテスト，ネットワークプロセッサの設計に関する研究に従事．IEEE, VLSI Society of India, Institute of Electronics & Telecommunication Engineers of India 各会員．

藤原　秀雄　（正員：フェロー）

昭 44 阪大・工・電子卒．昭 49 同大大学院博士課程了．同大・工・電子助手，明大・工・電子通信助教授，情報科学教授を経て，現在奈良先端大・情報科学教授．昭 56 ウォータールー大客員助教授．昭 59 マッギル大客員準教授．論理設計論，フォールトトレランス，設計自動化，テスト容易化設計，テスト生成，並列処理，計算複雑度に関する研究に従事．著書「Logic Testing and Design for Testability」（MIT Press）など．大川出版賞，IEEE Computer Society Outstanding Contribution Award, IEEE Computer Society Meritorious Service Award など受賞．情報処理学会フェロー，IEEE Computer Society Golden Core Member，IEEE Fellow.