社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－127（5）
2006／11／28

平衡構造に基づく階層テストにおけるテストプラン生成法

川原 侑大† 市原 英行†† 井上 智生††

† 広島市立大学大学院 情報科学研究科
†† 広島市立大学 情報科学部
〒731–3194 広島市安佐南区大塚東3丁目4-1
E-mail: †kawahara@dsgn.im.hiroshima-cu.ac.jp, ††{ichihara,tomoo}@im.hiroshima-cu.ac.jp

あらまし 大規模集積回路に対するテスト生成を効率良く行う方法として，階層テスト生成[2],[4],[5],[7]-[10]がある．従来の階層テスト生成では，レジスタ転送レベルデータパスのモジュールごとにテスト生成を行うのが一般的であった．本論文では，階層テスト生成をより効率良く行うために，平衡構造となる部分回路を階層の単位とした階層テスト生成を提案する．これにより，テストプラン生成が容易になり，またテスト実行時間の削減が期待できる．本論文では，この利点をいかし，テスト実行時間を効果的に削減するテストプランを生成するためのヒューリスティックアルゴリズムを提案する．また，実験結果では，提案手法がテスト実行時間を削減できることを示す．

キーワード 階層テスト生成，テストプラン，データパス，平衡構造，テスト実行時間

A Method of Test Plan Generation in Hierarchical Test Based on Balanced Structure

Yudai KAWAHARA†, Hideyuki ICHIHARA††, and Tomoo INOUE††

† Graduate School of Information Sciences, Hiroshima City University
†† Faculty of Information Sciences, Hiroshima City University
3-4-1, Ohzuka-higashi, AsaMinamiku, Hiroshima-shi , 731-3194 Japan
E-mail: †kawahara@dsgn.im.hiroshima-cu.ac.jp, ††{ichihara,tomoo}@im.hiroshima-cu.ac.jp

Abstract Hierarchical test generation is an efficient method of test generation for VLSI circuits. Traditional hierarchical test generators perform test generation for each module in the register-transfer level circuits. In this paper, we present a hierarchical test generation method which generates test-patterns for each balanced sub-circuit. Accordingly, the method can reduce the cost of test plan generation as well as the test application time. We propose a heuristic algorithm for generating test plans which can reduce the test application time based on our hierarchical test generation method. Experimental results show that our method can effectively generate test plans with small test application time.

Key words Hierarchical test generation, test plan, datapath, balanced structure, test application time

1. はじめに

大規模集積回路 (VLSI) に対するテスト費用の削減は重要な課題の1つである．今日のVLSI設計は，論理合成技術の発展により，レジスタ転送レベル (RTL) で行うことが一般的になってきた．RTLの情報を利用したテスト生成法として階層テスト生成[2],[4],[5],[7]-[10]がある．階層テスト生成は，(1) データパスを構成する各モジュールに対するテスト生成，(2) 各モジュールに対して (1) で生成されたテストパターンを，外部入力からモジュールの入力に伝搬し，その出力応答をモジュールの出力から外部出力まで伝搬するための制御信号系列（テストプラン）の生成，の2つのステップからなる．

上述した階層テスト生成は，ゲートレベル上のテスト生成に比べて，小さい時間でテスト生成，テスト実行をすることが期待できる．しかし，テストプラン生成は，回路の構造によっては多くの時間を要する場合がある．そこで，テストプラン生成を容易に行うための階層テスト容易化設計法[5]が提案されている．しかし，一般的にRTLデータパスには，モジュールが多数存在し，そのモジュールごとにテストプランを生成するので，テストプラン数の増加を招き，結果的にテスト実行時間が増加する．

本論文では，平衡構造を満たす部分回路（ブロック）を第1ステップの対象とした階層テスト生成を提案する．平衡構造を満たすブロックは，組合せ回路用ATPGを適用可能なだけでなく，テストプラン生成も容易になると考えられる．さらに，テスト実行時間の削減も期待できる．本論文では，これらの平衡ブロックの利点をいかして，テスト実行時間を効果的に削減するためのテストプランを生成するヒューリスティックアルゴリズムを提案する．

2. 平衡構造に基づく階層テスト生成

本論文では，RTL のデータパス回路を対象とする．データパス部の制御信号，状態信号は，それぞれデータパス外部から直接制御，観測可能とする．

図 1 に，対象とする RTL データパスの例を示す．この図において，$C1$,$C2$ などは組合せ回路部を表す．これらは，演算器やマルチプレクサ (MUX) などの組合せ回路のみの接続からなり，レジスタなどの記憶素子は含まない．また，1,2 などの矩形はレジスタを表す．この例の外部入力は $PI1$,$PI2$, 外部出力は $PO1$,$PO2$ である．

2.1 従来の階層テスト生成とその課題

階層テスト生成は，一般的に 2 つのステップからなる．第 1 ステップでは，演算器などの構成要素を対象とした論理レベルでのテストパターン生成を行う．一般的には，加算器，乗算器などの演算回路や MUX が対象となる．以下では，この第 1 ステップでのテストパターン生成対象回路を，階層の基本単位 (あるいは単に階層単位) という．第 2 ステップでは，各階層単位について，生成されたテストパターンを RTL データパスの外部入力から階層単位の入力に伝搬し，その出力応答を階層単位の出力からデータパスの外部出力まで伝搬するための制御信号系列を生成する．この制御系列をテストプランという．

これまでの階層テスト生成の多くは，演算器や MUX などの組合せモジュールを階層単位としている．同じ型の演算器が複数使用されている場合，そのテストパターン集合は共有できるので，テストパターン生成時間の短縮になる．しかしながら，データパスの規模が大きくなると階層単位の数は増加する．テストプランは階層単位ごとに生成する必要があるので，テストプラン生成時間が増加するだけでなく，テスト実行時間も増加する．複数の階層単位に対するテストの平行実行によるテスト実行時間の短縮のためには，テストプラン圧縮 [7]- [9] が必要となる．

2.2 提案手法のポイント

このような問題を解決する方法として，階層の基本単位を大きくすることが考えられる．すなわち，複数のモジュールをからなる部分回路を 1 つの階層単位とする．階層単位を大きくすれば階層単位数は小さくなり，テストプランの数も小さくなる．その結果，総テスト実行時間の削減も期待できる．単純な階層単位の拡大は各階層単位のテストパターン生成時間を増大させるだけでなく，テストパターン集合の共有もできなくなるため，第 1 ステップでのテスト生成時間の増大を招く可能性がある．また，テストパターン数の増加によるテスト実行時間の増加も考えられる．したがって，テストパターン生成時間，テストプラン生成時間の増加を抑えながら，テスト実行時間を最小にする最適な階層の基本単位が存在すると考えられる．

本論文では，階層の基本単位として平衡構造 [1] を考える．平衡構造はテスト生成が容易な順序回路のクラスの一つであり，順序回路でありながら組合せ回路としてテスト生成アルゴリズムを適用可能である．また，平衡構造に対するテスト実行は，1 つのテストパターンを平衡構造の入力で d_b 時刻ホールドすることでテスト実行可能 (d_b: 平衡構造の順序深度) であり，テストプラン生成が容易になると考えられる．平衡構造の定義は後述する．

図 1 のデータパスを例として説明する．部分回路 B_2 は平衡構造である．この平衡構造 B_2 の入力に対応するレジスタは 2,3,8,9,10,13, 出力レジスタは 9,10,11,14 である．この平衡構造 B_2 に対するテスト系列は，図 2 に示す組合せ等価 $C(B_2)$ に対してテストパターンを求めることで得られる．$C(B_2)$ に対するテストパターン (a,b,c,d,e,f) と，対応する出力応答 $(z1,z2,z3,z4)$ について，平衡構造 B_2 で考えると，入力レジスタ $(2,3,8,9,10,13)$ にそれぞれ (a,b,d,e,c,f) を設定し，B_2 の順序深度 $d_b=2$ と同じ時間をホールドすることで，3 サイクル後に出力レジスタ $(9,10,11,14)$ に $(z3,z2,z1,z4)$ が得られる．

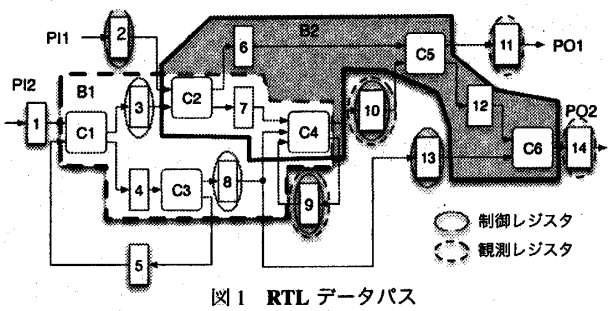

図 1 RTL データパス

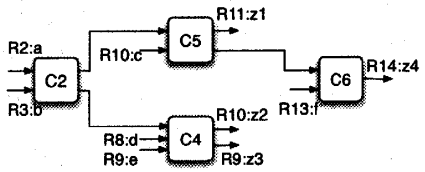

図 2 図 1 の B_2 における組合せ等価

	PI		PO		レジスタ										
t	1	2	1	2	1	2	3	4	7	8	9	10	11	13	14
-4	c	f													
-3	e	d			f	c									
-2		b			d	e		f	c						
-1	a				b			d	e	f		c			
0						a	b			d	e	c		f	
1						a	b			d	e	c		f	
2						a	b			d	e	c		f	
3			z1	z4							z3	z2	z1		z4
4			z2									z3	z2		
5			z3										z3		

図 3 図 1 の B_2 におけるテストプラン実行例

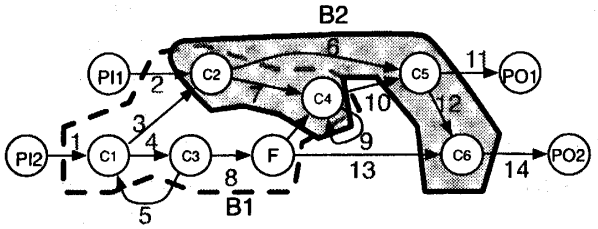

図 4 図 1 に対する構造グラフ

図 1 のデータパスにおいて，平衡構造となる部分回路は，B_2 のほかに B_1 が考えられる．これら 2 つの平衡構造を階層の基本単位としてテスト生成，テストプラン生成を行うことで，データパス全体のテスト生成が可能となる．図 1 のように，階層単位は重複があってもよい．重複部分の回路の故障は，いずれかの階層単位の故障として検出されれば十分である．

2.3 諸 定 義

平衡構造に基づく階層テスト生成法を考察するために以下の準備

を行う.

RTL データパスは，構造グラフ $G=(V,E,w)$ で表す．ここで，$v\ (\in V)$ は組合せ回路部 (組合せ回路のみの接続からなる部分回路，$v\in V_C$)，外部入力 ($v\in V_I$)，外部出力 ($v\in V_O$)，または分岐点 ($v\in V_F$) のいずれかを表す (それぞれ，$V=V_C\cup V_I\cup V_O\cup V_F$)，辺 $(u,v)\ (\in E)$ は，組合せ回路部などの接続を表し，$w(u,v)=1$ は，組合せ論理部 u から v へはレジスタを通して接続されていることを表す．$w(u,v)=0$ は，u,v 間にレジスタがない (信号線のみの) 接続を表す．例えば，図 1 のデータパスを表す構造グラフ G は図 4 のようになる．辺 $(F,C4)$ に対してのみ，$w(F,C4)=0$ となり，その他の各辺 (u,v) について $w(u,v)=1$ である．

構造グラフ G が以下の条件を満たすとき，対応するデータパスは平衡であるという [1].

- G は無閉路である (フィードバックループを持たない).
- 任意の頂点対 $u,v\ (\in V)$ をそれぞれ始点，終点とする経路集合 $P(u,v)$ について，$\forall p,q\in P(u,v), l(p)=l(q)$, ただし，$l(p)=\sum_{(u,v)\in p} w(u,v)$.

階層の基本単位として考える平衡構造 (平衡な部分回路) を平衡ブロックという．平衡ブロックは，データパスを表す構造グラフ G の部分グラフ $G_B=(V_B,E_B)$ $(V_B\subseteqq V, E_B\subseteqq E)$ として表される．例えば，図 1 のデータパスにおける平衡ブロック B_2 の構造グラフ G_{B_2} は，$V_{B_2}=\{C2,C4,C5,C6\}$, $E_{B_2}=\{(C2,C4),(C2,C5),(C5,C6)\}$ となる．平衡ブロックの外部入出力は，その境界にあるレジスタに対応し，テストプランにおける制御対象，観測対象のレジスタ (それぞれ (2,3,8,9,10,13), (9,10,11,14)) になる．以下では単にそれぞれを制御レジスタ，観測レジスタといい，E_B^j, E_B^p $(\subseteqq E-E_B)$ と表す.

以下の条件を満たす経路 $p=(v_1,v_2,\ldots,v_n)$ と写像 $t:V\to Z$ (Z は整数の集合) の組を，(外部入力から) 制御レジスタ $e\in E_B^j$ への制御プランといい，$\pi^j(e)=(p,t)$ と表す.

$$v_1\in V_I$$
$$(v_{n-1},v_n)=e$$
$$\forall 1\leq i<n\ [w(v_i,v_{i+1})>0\Rightarrow t(v_i)<t(v_{i+1}),\ w(v_i,v_{i+1})=0\Rightarrow t(v_i)=t(v_{i+1})]$$

ここで，写像 $t(v)$ は，組合せ回路部 v のスケジュール (v を利用する時刻) を表している．同様に，以下の条件を満たす経路 $p=(v_1,v_2,\ldots,v_n)$ と写像 $t:V\to Z$ の組を，観測レジスタ $e\in E_B^p$ から (外部出力への) 観測プラン といい，$\pi^p(e)=(p,t)$ と表す.

$$v_n\in V_O$$
$$(v_1,v_2)=e$$
$$\forall 1\leq i<n\ [w(v_i,v_{i+1})>0\Rightarrow t(v_i)<t(v_{i+1}),\ w(v_i,v_{i+1})=0\Rightarrow t(v_i)=t(v_{i+1})]$$

ここでは，組合せ回路部 v の入力から出力の値の伝搬はスルー機能等によって実現されるものとする.

平衡ブロック G_B に対するテストプラン Π_B は，次の条件を満たす制御プラン集合と観測プラン集合の和集合である．つまり，$\Pi_B(=\Pi_B^j\cup\Pi_B^p)$ で表される．ただし，$\Pi_B^j=\bigcup_{e\in E_B^j}\pi^j(e)$, $\Pi_B^p=\bigcup_{e\in E_B^p}\pi^p(e)$ であり，それぞれは以下の条件を満たす.

$$\forall e\in E_B^j, \exists\pi^j(e)\in\Pi_B^j,\ head(p)=e$$
$$\forall(p,t_p)\neq(q,t_q)\in\Pi_B^j,\ \forall u\in p-\{head(p)\},\forall v\in q-\{head(q)\},\ u=v\Rightarrow t_p(u)\neq t_q(v)$$
$$\forall e\in E_B^p, \exists\pi^p(e)\in\Pi_B^p,\ tail(p)=e$$
$$\forall(p,t_p)\neq(q,t_q)\in\Pi_B^p,\ \forall u\in p-\{tail(p)\},\forall v\in q-\{tail(q)\},\ u=v\Rightarrow t_p(u)\neq t_q(v)$$

図 1 のデータパスにおける平衡ブロック B_2 について，そのテストプランについて考える．例えば，制御レジスタ 9 (図 4 における辺 $(C4,C4)$) の制御について，制御プラン $((PI1,-3),(C2,-2),(C4,-1),(C4,0))$ によって制御可能となる．また，制御レジスタ 10 (辺 $(C4,C5)$) の制御プランでは，$((PI1,-4),(C2,-3),(C4,-2),(C5,0))$ のように，制御レジスタ 10 のホールドを用いても，制御することが可能である．さらに，観測レジスタ 10 (辺 $(C4,C5)$) の出力応答の伝搬について考える．これは，観測プランを $((C4,2),(C5,3),(PO1,4))$ とすることで，伝搬が可能になる．このように，全ての制御・観測レジスタに対するテストプランを生成することでテスト実行が可能となる.

平衡構造に基づく階層テストは，平衡ブロックの集合 $\boldsymbol{G_B}=\{G_{B_1},G_{B_2},\ldots,G_{B_N}\}$ が データパス G を被覆する，すなわち，$V=\bigcup_{i=1}^N V_{B_i}$ を満たすように階層単位を求め，各平衡構造 G_{B_i} に対するテストプランを得ることで実行できる．つまり，図 1 に示したデータパスの B_1, B_2 がそれぞれ G_{B_1}, G_{B_2} に対応し，それぞれのテストプランを得ることで図 1 のテスト実行が可能となる．平衡ブロック G_{B_i} に対して単一パターン系列の適用に要するテスト実行時間は以下の式で表すことができる.

$$T_{B_i}=\max_{(p,t)\in\Pi_{B_i}^j}(t(head(p))-t(tail(p)))+d_{B_i}+1+\max_{(p,t)\in\Pi_{B_i}^p}(t(head(p))-t(tail(p))) \quad (1)$$

ここで，d_{B_i} は平衡ブロック G_{B_i} の順序深度を示す.

平衡構造に基づく階層テストは，平衡ブロック G_{B_i} ごとに実行する．G_{B_i} に対するテスト実行時間は，G_{B_i} のテストプラン長に，G_{B_i} のテストパターン数を乗算することで得られる．したがって，平衡構造の集合 $\boldsymbol{G_B}$ に対する総テスト実行時間は，以下の式で表される.

$$T_B=\sum_{G_{B_i}\in\boldsymbol{G_B}} T_{B_i}N_{B_i} \quad (2)$$

ただし，N_{B_i} は平衡ブロック G_{B_i} に対して生成されたテストパターン数を示す.

3. テストプラン生成アルゴリズム

本論文では，与えられた平衡ブロック集合 $\boldsymbol{G_B}$ に対する総テスト実行時間を削減するために，各平衡ブロック G_{B_i} に対するテストプラン長 T_{B_i} の削減を考える．式 (1) より，T_{B_i} を小さくするためには各平衡ブロックの制御プラン集合 (観測プラン集合) におけるプラン長の最大値，つまり $\max_{(p,t)\in\Pi_{B_i}^j}(t(head(p))-t(tail(p)))$ $(\max_{(p,t)\in\Pi_{B_i}^p}(t(head(p))-t(tail(p))))$ を小さくすることが必要である．以下ではプラン長を最小化することを目的とした平衡ブロックの制御プラン集合生成アルゴリズムについて説明する．なお，観測プラン集合の生成アルゴリズムについては，紙面の都合上割愛するが，制御プラン集合生成アルゴリズムと類似したアルゴリズムで生成可能である.

3.1 基本方針

平衡ブロック G_{B_i} に対する制御プラン $\Pi_{B_i}^j$ の長さを小さくするために，制御レジスタ e の制御プランを決定する際，他の制御レジスタの制御プランのことも考えて探索を行う.

図 5 は，ある平衡ブロックの構造グラフにおいて，外部入力か

らそれぞれの制御レジスタ {(4,8),(9,8),(7,11),(13,14)} までに到達可能な部分構造グラフを示している．これらの制御レジスタをそれぞれ a,b,c,d と呼ぶ．各頂点 v の横に書かれた数値は最短順序深度 $d(v)$ を表す．図 6 は，図 5 の各制御レジスタが利用可能な制御プランをすべて示した上で，制御プラン集合のサイズが最小となる場合（長さ 4）の制御プラン集合を線で囲んで示している．なお，各制御レジスタに値が設定される時刻を 0 としている．また，頂点 8 は 2 つの制御レジスタ a,b の終点であるため，説明の都合上制御レジスタ a の終点として，新たに頂点 $8'$ を追加して表記している．図 6 に示したとおり，各制御レジスタの利用可能な制御プランは複数存在する．制御レジスタ b の制御プラン $((PI1,-4),(2,-3),(5,-2),(9,-1),(8,0))$ は，b の制御プランとして最短ではないが，制御プラン集合としては最小であることがわかる．

このような制御プラン集合を生成するために，以下のようなヒューリスティックアルゴリズムを提案する．

(1) 制御レジスタをホールドする回数 g を 0（ホールドを用いない）に設定する．

(2) E_B^j から制御レジスタ e を 1 つ選択して，制御プラン (p_e,t_e) を決定する．この時以下の考え方に基づく．
- 各制御レジスタを g 時刻ホールドすることを条件に，できるだけ小さいプランを選択する．
- プランの探索は制御レジスタ e から入力方向に向かって行う．
- 他の制御レジスタが利用する可能性が高いプランはなるべく避ける．
- 利用したいプランがすでに他の制御レジスタのプランとして利用されている場合は，バックトラックを行い他のプランを探す．
- バックトラックを尽くしてもプランが見つからなければ，制御プラン探索を打ち切る．

(3) すべての制御レジスタの制御プランが決定すれば，終了．それ以外は g を 1 だけ増やして，(2) に戻る．

さらに，テストプラン実行を簡単にするために，制御レジスタ以外のレジスタはホールドしない．つまり，$\Pi^j_{B_i},\Pi^p_{B_i}$ は，以下の条件を満たす．

制御プラン集合 $\Pi^j_{B_i}$：

$$\forall((v_1,v_2,\ldots,v_n),t)\in\Pi^j_{B_i},$$

$$\begin{cases} t(v_i)+w(v_i,v_{i+1})=t(v_{i+1}) & (1\leq i<n-1)\\ t(v_{n-1})<t(v_n)\end{cases}$$

観測プラン集合 $\Pi^p_{B_i}$：

$$\forall((v_1,v_2,\ldots,v_n),t)\in\Pi^p_{B_i},$$

$$\begin{cases} t(v_i)+w(v_i,v_{i+1})=t(v_{i+1}) & (1<i\leq n-1)\\ t(v_1)<t(v_2)\end{cases}$$

以下，(2) の制御プランの決定方法について詳しく述べる．

3.1.1 制御レジスタの制御プラン決定法

提案する制御プランの決定手法では，各制御レジスタから入力方向に向かって，プランを探索する．全ての制御レジスタ $E^j_{B_i}$ に対して，小さいプラン長で競合のないテストプランを生成したいので，制御レジスタ $e\in E^j_{B_i}$ のプラン決定の際，小さいプラン長になるように制御プランを選択しつつ，e 以外の制御レジスタの制御プラン生成にできるだけ影響を与えないような制御プランの探索を行う．

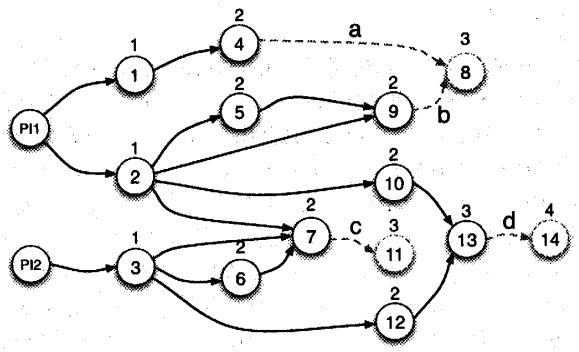

図 5 制御レジスタ $\{a.b.c.d\}$ に関する部分構造グラフ: G'

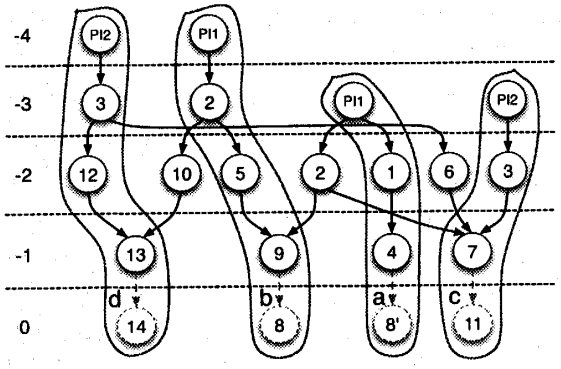

図 6 G' の制御レジスタが利用可能な制御プランと最適制御プラン集合

例えば，図 6 において，制御レジスタ c の制御プランを決定する際，時刻 -1 の頂点 7 において，時刻 -2 の頂点 3, 6, 2 のどの頂点を選ぶべきかを考える．プラン長だけを考えると，プラン長が 3 となる頂点 2 と頂点 3 のいずれかが候補となる．ここで，頂点 2 を考えると制御レジスタ b の制御プランとして利用される可能性があるが，頂点 3 は制御レジスタ c の制御プランでしか利用されない．よって，ここでは頂点 3 を選んだ方が良いと言える．

このように，他の制御レジスタの制御プランで利用される可能性の低い制御プランを選択したいので，各頂点をそれぞれの制御レジスタが利用しようとしている度合いを数値化し，それを利用して各プランの混雑度を求め，これを指標として各制御レジスタの制御プランを決定する．

まず，各制御レジスタ e の頂点 v への時刻 τ での要求度 $r(e,v,\tau)$ を定義する．時刻 0 の制御レジスタ e の終点 $v\,(=head(e))$ の要求度は，必ず必要なので 1 とする．つまり，$\forall e\in E^j_{B_i}, r(e,head(e),0)=1$. また，時刻 τ の頂点 v の要求度は，時刻 τ または時刻 $\tau-1$ で，その頂点 v に直接繋がる親集合（$pre(v)$）に分配することで，頂点 v がその親集合の各頂点をどのぐらい要求しているかを表現する．ここでは分配比は，頂点 v が $pre(v)$ をプランとして利用したときの順序深度の逆数の比としている．また，制御レジスタ e からのプランが頂点 v で収れんしている場合は，制御レジスタの要求度がそれぞれのプランを通って頂点 v に集まると考えられるため，それらを全て足し合わせることとする．以上をまとめると，制御レジスタ e の頂点 v への時刻 τ での要求度 $r(e,v,\tau)$ は，頂点 v の子の集合を $suc(v)$ とすると，

$$r(e,v,\tau)=\sum_{u\in suc(v)}\alpha(v,u)\times r(e,u,\tau+w(v,u))$$

ただし，

$$\alpha(v,u)=\frac{1/(d(v)+w(v,u))}{\sum_{q\in pre(u)}1/(d(q)+w(q,u))}$$

である．

次に，それぞれの制御レジスタから，頂点 v への時刻 τ での要求度の総和を考える．つまり，

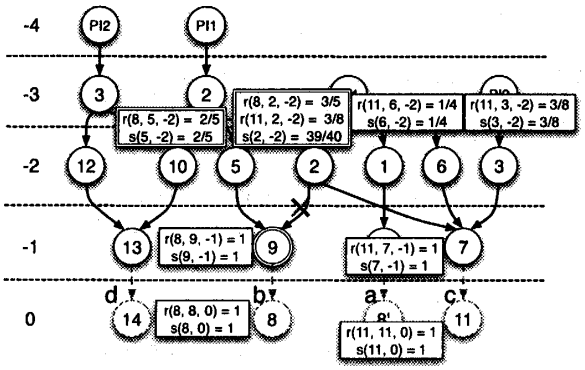

図 7　要求度と要求度和の計算例

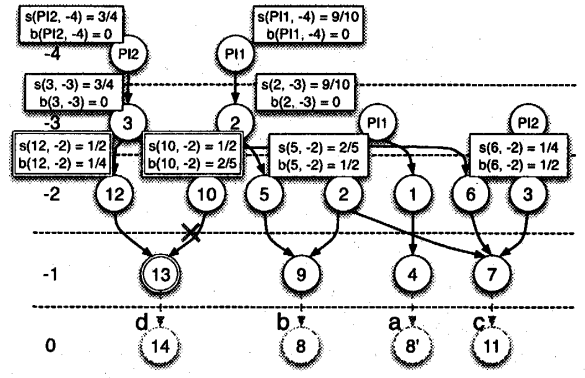

図 8　混雑度の計算例

$$s(v,\tau) = \sum_{e \in E^j_{B_i}} r(e,v,\tau).$$

この頂点 v の時刻 τ での要求度和は，時刻 τ の頂点 v に対して，各頂点がどれだけ要求しているかを表す指標となる．

時刻 -2 から 0 で，制御レジスタ b と c から，頂点 5, 2, 6, 3, 9, 7, 8, 11 に対する要求度と，要求度和を図 7 に示す．時刻 -2 において，頂点 5 の要求度和は $s(5,-2) = 2/5$，頂点 2 の要求度和は $s(2,-2) = 39/40$ となる．

次に，要求度和をもとに，時刻 τ における頂点 v から外部入力までの混雑度 $b(v,\tau)$ を考える．ここで混雑度 $b(v,\tau)$ とは，時刻 τ の頂点 v から外部入力までに，その頂点 v を通る制御プラン以外の制御プランでどれだけの要求があるのかを表した指標である．この混雑度が高いほど，外部入力までの制御プラン生成が困難であるということを示すことができ，また同時に，制御プランを生成した時に，他の制御レジスタの制御プラン生成に与える影響の強さも示すことができる．今，頂点 v から親頂点 u(時刻 $\tau-1$) を通り，外部入力までのプランの混雑度を考える．このとき，頂点 u の混雑度 $b(u,\tau-1)$ に，頂点 u への頂点 v から以外の要求度和 $s(u,t-1)-s(v,t)$ を加えれば，頂点 v を通る制御プラン以外の要求度和，つまり混雑度を表すことができる．なお，定義から，v が外部入力ならば，$b(v,\tau)$ は 0 となる．また，v の親頂点が複数ある場合は，混雑度を楽観的に考え，最も小さい $b(u,\tau-1), u \in pre(v)$ をもとに v の混雑度を計算する．以上をまとめると，

$$b(v,t) = \min_{u \in pre(v)} (s(u,t-w(u,v)) - s(v,t) + b(u,t-w(u,v)))$$

となる．

図 8 に混雑度の計算例を示す．図では，時刻-4 から-2 の頂点 PI2, PI1, 3, 2, 12, 10, 5, 6 の混雑度を示している．例えば，時刻 -2 の頂点 10 の混雑度は，$b(10,-2) = s(2,-3) - s(10,-2) + b(2,-3) = 2/5$ となる．$s(2,-3) - s(10,-2)$ によって，時刻 -3 で頂点 2 は，制御レジスタ b の制御プランとして要求されていること，また，$b(2,-3)$ を加算することで，時刻 -3 以前に，他の制御レジスタの制御プランとして要求されていないことがわかる．

制御レジスタ e の制御プランの決定アルゴリズムは各頂点の要求度和と混雑度に基づいて行われる．つまり，時刻 τ の頂点 v の親頂点が複数ある場合は ($|pre(v)| > 1$)，その親頂点の要求度和と混雑度を加えたもの $(u \in pre(v), s(u,\tau-w(u,v)) + b(u,\tau-w(u,v)))$ が小さい頂点を選ぶ．例えば図 8 で，制御レジスタ d の制御プランを決定する際，時刻 -1 の頂点 13 において，12 と 10 のどちらを選ぶかという選択がある．要求度和は $s(12,-2) = s(10,-2) = 1/2$ と同じであるが，混雑度が $b(12,-2) = 1/4$，$b(10,-2) = 2/5$ である．要求度和と混雑度を加えると，頂点 12 の方が小さいので，頂点 12 を選択する．

3.1.2　制御プラン決定の順番

ここでは，上記のような制御プランの決定を行う制御レジスタの順番について考察する．

まず，提案するアルゴリズムが全ての制御レジスタのテストプランを生成できることを保証するために，ある制御レジスタ u に対して，他のレジスタ u' が u の制御プラン上に存在する可能性がある場合，すなわちレジスタ u' から u に到達可能である場合は，レジスタ u' から制御プランを生成することにする．

また，これらの依存関係がない場合は，なるべく制御プラン生成が困難な制御レジスタから生成を行う．そのため，

- 制御プラン長 $d(head(e))$ が大きい制御レジスタ e ほど，制御プラン選択の時間的自由度が低いと考え，優先的に選ぶことにする．
- また，最短順序深度 $d(head(e))$ が同じ制御レジスタが複数あれば，前節で説明した混雑度の大きなレジスタから選択することにする．

という方針で制御レジスタの順番を決めて，制御プランを生成する．

図 5 の例では，それぞれの制御レジスタに依存関係はなく，また制御レジスタ a,b,c,d の終点の順序深度はそれぞれ 3,3,3,4 であるため，最初にレジスタ d の制御プランを決定する．さらに制御レジスタ a,b,c の混雑度はそれぞれ，39/40, -1/10, -5/8 であるため，a,b,c の順で制御プラン生成を行う．

3.2　適 用 例

図 8 において，制御レジスタ d の制御プラン決定について考える．時刻 -1 の頂点 13 において，12 か 10 のプランを選択できる．ここでは上述したとおり，頂点 12 を選択するが，もし，頂点 10 を選択した場合，d の制御プランに $(2,-3)$ が含まれる．その時，制御レジスタ b の制御プランは，$(2,-3)$ を利用できなくなり，$((PI1,-3),(2,-2),(9,-1),(8,0))$ となるので，$(PI1,-3)$ が利用する a の制御プランを生成できなくなる．一方，a の制御プランを $((PI1,-3),(1,-2),(4,-1),(8',0))$ と先に決定したら，b の制御プランを生成できなくなる．しかし，提案手法では d の制御プランを決める時に，他の制御レジスタのことも考慮して，時刻 -2 で頂点 12 を選択することができるので，図 6 に示した最小解を得ることができる．

図 7 の，制御レジスタ b の制御プラン決定について，時刻 -1 の頂点 9 からの探索を考える．時刻 -2 で頂点 5,2 の選択肢がある．前節で述べたとおり，制御レジスタ b の制御プランを探索する時には，制御レジスタ a のプラン $((PI1,-3),(1,-2),(4,-1),(8',0))$ を決定済みである．しかし，b のプラン探索において，もし頂点 2 を選択すると $(PI1,-3)$ を使用できないので，バックトラックが必要となる．本論文で提案したヒューリスティックでは，この場合，頂点 5 を選択するので，バックトラックを避けることができると思われる．

表1 実験結果

回路	文献[5]						提案手法					
	階層単位	単位数	#ptns.	TG[s]	Tapp.[cycle]	FCov.[%]	階層単位	単位数	#ptns.	TG[s]	Tapp.[cycle]	FCov.[%]
ex1	mux	13	12	0.03	876	100.00	B_1	1	88	0.70	968	100.00
(#PIs:2,	add	4	22	0.03	484	100.00	B_2	1	51	0.17	561	100.00
#Pos:2,	mult	2	66	0.27	792	100.00						
#regs:16)	total	19	*376(注1)	0.33	2152	100.00	total	**2**	**139**	**0.87**	**1529**	**100.00**
ex2	mux	12	12	0.03	792	100.00	B_3	1	107	0.80	1391	100.00
(#PIs:2	add	3	22	0.03	330	100.00	B_4	1	31	0.24	341	100.00
#POs:2,	mult	2	66	0.27	858	100.00						
#regs:14)	sub	1	20	0.03	120	100.00						
	total	18	*362	0.36	2100	100.00	total	**2**	**138**	**1.04**	**1732**	**100.00**
4thIIR	mux	3	12	0.03	276	100.00	B_5	1	103	0.89	1648	99.99
(#PIs:1,	add	2	22	0.03	374	100.00						
#POs:1,	mult	3	66	0.27	1320	100.00						
#regs:12)	total	8	*278	0.33	1970	100.00	total	**1**	**103**	**0.89**	**1648**	**99.99**

4. 実験結果

提案手法の有効性を確認するために，実験用回路(16bit)を用いて文献[5]との比較実験を行った．実験結果を表1に示す．手法それぞれの項目は，左から階層の単位名，階層の単位数，テストパターン数，テストパターン生成時間[s]，テスト実行時間[cycle]，故障検出効率[%]を示す．なお，muxはマルチプレクサ，addは加算器，multは乗算器，subは減算器，B_1〜B_5は平衡ブロックを示す．また，論理合成にはSynopsys社のDesign Compiler，テスト生成にはTetra Maxを用いた．テスト実行時間の計算は，文献[5]については，各モジュールのテストパターン数に，そのモジュールのテストプラン長を乗算することで求めた．提案手法に関しては式(2)を用いて計算した．

階層単位について，文献[5]ではRTLモジュール，提案手法では平衡ブロックを示す．両手法ではこの階層単位に対して，テストパターン生成，テストプラン生成を行う．各回路の単位数のtotalに示したとおり，提案手法では階層単位数が小さくなる．

提案手法によって生成されるテストパターン数は，文献[5]に比べて小さいことがわかる．これは，提案手法はテストパターン生成の対象となる階層の単位を大きくすることで，1つのパターンに対して検出できる故障数が増加することに起因すると思われる．

テストパターン生成時間を比較すると，文献[5]に対して，提案手法は増加の傾向を示している．これはテスト生成の単位を大きくしたことが原因であると考えられる．また，文献[5]は，モジュール単位でテスト生成を行うので，同じ型のモジュールに対するテストパターンを共有できることが理由として考えられる．

テスト実行時間について，提案手法は，全ての回路において大きく削減できることがわかる．これは上述したとおり，階層の単位数を削減したことによって，テストプラン数を削減したこと，さらに3.で提案したアルゴリズムによってテストプラン長を効果的に削減したことが，原因であると考えられる．

回路4thIIRに対する結果に示すように，100%の故障検出効率が得られない場合もあった．このことより，たとえ平衡構造であっても，テスト生成が容易でない場合もあることがわかる．よって，平衡ブロックの選択法も重要であるといえる．

5. まとめと今後の課題

本論文において，提案手法は効果的にテスト実行時間を削減可能であることを示した．今後の課題について述べる．一般的に，あるRTLデータパスに対して，様々な平衡ブロックを生成可能である．(あるRTLデータパスに対して，平衡構造を満たす部分回路は一意に決まらない．)そこで，テスト実行時間のさらなる削減を実現するために，提案したアルゴリズムがより効果的に機能する平衡ブロックの性質の考察が考えられる．

参考文献

[1] Rajesh. Gupta, Rajiv Gupta, and Melvin A. Breuer, "The BALLAST Methodology for Structured Partial Scan Design," IEEE Trans. Com, Vol.39, No.4, April 1990.

[2] B.T. Murray and J.P. Hayes, "Hierarchical test generation using pre computed tests for modules," IEEE Trans. Computer-Aided Design, Integrated Circuits & Syst., vol.9, no.6, pp.594-603, June 1990.

[3] A. Balakrishnan, S. T. Chakradhar, "Sequential Circuits with Combinational Test Generation Complexity," 9th International Conference on VLSI Design, January 1996.

[4] I. Ghosh, A. Raghunathan, and N.K. Jha, "A design for testability technique for register-transfer level circuits using control/data flow extraction," IEEE Trans. Computer-Aided Design, vol. 17, pp. 706-723, Aug. 1998.

[5] 和田, 増澤, K.K. Saluja, 藤原, "完全故障検出効率を保証するRTLデータパスの非スキャンテスト容易化設計法," 信学論 (D-I), vol.J82-D-I, no.7, pp. 843-851, July 1999.

[6] 井上, 細川, 藤原, "組合せATPGに基づくRTレベル部分スキャン設計法," 信学技報 (FTS96-67), Feb. 1997.

[7] S. Ravi, G. Lakshminarayana and N. K. Jha, "High-level test compaction techniques," IEEE Trans. on Computer-Aided Design, Vol.21, No.7, pp.827-841, July 2002.

[8] T. Hosokawa, H. Date and M. Muraoka, "A test generation method using a compacted test table and a test generation method using a compacted test plan table for RTL data path circuits," in Proc. of VTS, pp.328-335, 2002.

[9] 永井, 大竹, 藤原, "テスト実行時間削減のためのデータパスの強可検査性に基づくテスト容易化設計法," 信学技報 (DC2002-84), Vol. 102, No. 658, pp.31-36, Feb. 2003.

[10] H. Ichihara, N. Okamoto, T. Inoue, T. Hosokawa, and H. Fujiwara, "An Effective Design for Hierarchical Test Generation Based on Strong Testability," Proc. IEEE Asian Test Symposium, pp. 288-293, Dec. 2005.

(注1)：表1の文献[5]において，#ptnsのtotalは，階層の各単位数と階層単位それぞれのテストパターン数を乗算し，その合計を示している．