RC-003

2パタンテスト可能な演算ペアに基づく遅延テストのためのスケジューリング法
A Scheduling method for Delay testing based on two-pattern testable operation pair

中谷 夏主政† 吉川 祐樹†
Kazumasa Nakatani Yuki Yoshikawa

1. はじめに

近年，製造プロセスの微細化による VLSI の動作速度や性能の向上により，回路のタイミング不良の原因となる欠陥が問題となっている．このような欠陥を検出するために遅延故障のテストは重要な課題である．遅延故障のモデルには，遷移故障，セグメント故障，パス遅延故障などがある．中でも遷移故障はゲート上(一般にはゲートの入出力の信号線間)の過度な伝搬遅延を表現する故障モデルで，遷移故障に対するテストを行うことで信号遅延の原因となる欠陥を見つけることができる[1].

一般に順序回路に対するテスト生成は膨大な時間を要するため，テスト容易化設計が行われる．現在は，テスト容易化設計としてスキャン設計[2]-[5]が主流である．しかし，実動作速度(at-speed)テストの制御が困難，スキャンシフトによるテスト実行時間の増大，面積オーバヘッドなど様々な問題を抱えている．それに対し，設計の上位からテストのことを考慮した設計法が提案されている．設計の上位段階では，回路の通常動作だけでなくテスト時の動作も併せて設計することができ，動作速度や面積の制約を満たし，かつテスト容易な回路を設計できる．文献[6]，[7]では，階層テスト生成[8]のためのレジスタ転送レベル(RTL)でのテスト容易化設計法が提案されている．この手法は，RTL 回路に対してテストパタンの正当化と伝搬に機能パスを利用することで，面積オーバヘッドとテスト実行時間の問題を緩和している．また，更に上位の動作レベルの段階からテスト動作を考慮したテスト容易化高位合成法[9]-[12]も提案されている．特に文献[12]では，RTL モジュール(演算器)に対する 2 パタンテストを保証する演算器/レジスタバインディング法を提案している．

本研究では，演算器のテストをスケジュールの段階から考える．具体的には，演算操作の流れをグラフで表現したデータフローグラフ(DFG) に対し，2 時刻連続で実行される演算をそれぞれ同じ演算器に割り当て RTL データパス中の対応する演算器が 2 時刻連続で活性化することに着目したスケジューリング法を提案する．

2. 高位合成

高位合成は回路の動作記述をレジスタ転送レベル回路に変換する操作である．高位合成の手続きは，まず動作記述から処理の流れを表すデータフローグラフ（以下，DFG）を生成する．続いてその DFG に対して，各演算の実行時刻を決定するスケジューリングを行い，スケジュール済み DFG（以下，SDFG）を生成する．更に SDFG に対して，各演算を実際に演算する演算器に，各変数をレジスタに割り当てるリソースバインディングを行い，コントローラの生成と併せて RTL 回路を生成する．

2.1 スケジューリング

本節ではスケジューリングについて具体的に説明する．スケジューリングは DFG を入力とし，SDFG を出力する．一般にスケジューリングでは，制約と最適化の目標を決め，その与えられた制約下で最適なスケジューリングの解を求める．制約や最適化の目標としては時間（実行サイクル数）や面積（リソース数）がある．例えば面積を制約として使用する演算器数を決めた場合，その制約下で実行サイクル数が最小となるスケジューリングを求める．また反対に，実行サイクル数を制約とする場合には，その制約下で使用する演算器数が最小になるようにスケジューリングを行う．

例として，使用するリソース制約が乗算器数 2 つと加算器数 1 つの場合のスケジューリングを図 1 に示す．左のグラフが DFG であり，右のグラフはスケジュール済み DFG となる．このスケジュールでは，乗算器数の制約が 2 つであるため，時刻 1 には 2 つの乗算を配置し，残りを時刻 2 に配置している．加算については時刻 2 と 3 に配置している．

本論文では 1 つの時刻に複数の演算を配置するチェイニングや複数時刻にまたがって演算を行うマルチサイクル演算はないものとする．

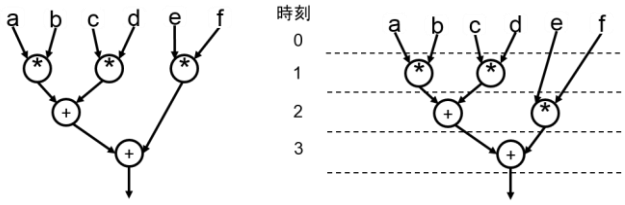

図 1 スケジューリング例

2.2 バインディング

バインディングは SDFG を入力とし，各演算について演算器への割り当てを決定し，各変数についてレジスタの割り当を決定する．SDFG の各演算を演算器に割り当てる演算器バインディングでは，実行時刻が重なっていない演算を同じ演算器に割り当てることができる．また各変数をレジスタに割り当てるレジスタバインディングでは，ライフタイムが重なっていない変数を同じレジスタに割り当てることができる．同じリソースを共有することで回路の面積を小さくすることができる．

例として図 1 の右側の SDFG を対象にバインディングを行った結果を図 2 に示す．このバインディングでは，乗算器*1 に対して乗算 op1 を割り当て，乗算器*2 に対して乗算 op2 と op4 を割り当てている．また加算器+1 については op3 と op5 を割り当てている．

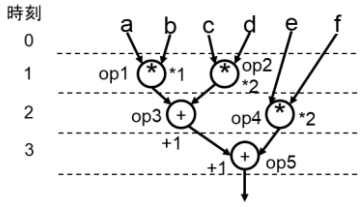

図 2　バインディング例

2.3 レジスタ転送レベル回路生成

バインディングの処理を行った後，各リソースの入力ポート間のインターコネクトを決定することでレジスタ転送レベル（RLT）回路を生成できる．この時，異なる複数のポートから同じポートへデータを転送する経路がある場合には適宜マルチプレクサを挿入する．また各レジスタのロードイネーブルとマルチプレクサの選択信号は，コントローラで制御される．

図 3 に図 2 から生成した RTL 回路のデータパス部を示す．

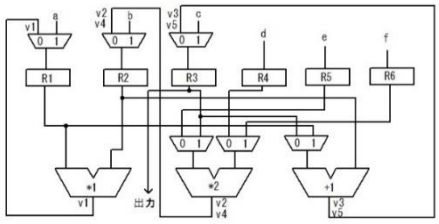

図 3　RTL データパスの例

3. 遅延故障とそのテスト

遅延故障のテストは，与えられた動作速度で回路が正しく動作するかどうかを検証するために行われる．回路に欠陥が存在する場合の影響をモデル化したものとして遷移故障やパス遅延故障などがある[1]．遷移故障は故障数が回路中のゲート数に対して線形であること，テストパタンが ATPG によって比較的容易に生成できることからよく使われているモデルである．本研究では遷移故障を対象とする．

3.1 遷移故障

遷移故障は，信号の立ち上がり遅延（Slow-to-Rise）と立ち下がり遅延（Slow-to-Fall）の 2 種類の遅延欠陥をモデル化している．遷移故障を検出するには，テストパタンのペア V=(v1,v2)を 2 時刻連続で回路に入力する必要がある．パタン v1 は回路を初期化するためのパタンであり，パタン v2 は故障箇所に立ち上がり（もしくは立ち下がり）の信号遷移を起こし，その遷移を観測点まで伝搬される活性化パタンである．

図 4 は遷移故障をテストするための 2 パタンテストの例を示している．この例では信号線 e で立ち上がりの信号遷移が遅れる遷移故障 f が存在すると仮定する．故障 f を検出するためには，初期パタンとして v1=（0,x,1,1），活性化パタンとして v2=（1,1,1,1）を入力することで，信号線 e に立ち上がりの遷移を起こし，故障 f によって遅れた信号を信号線 g で観測することができる．これにより故障 f を検出することができる．

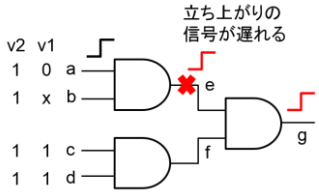

図 4　遷移故障の例

3.2 RTL モジュールの 2 パタンテスト

3.1 節で述べたように，遷移故障はゲートレベルで表現される故障モデルである．本研究では遅延テスト容易な回路設計をスケジューリングの段階から提案するため，スケジューリング，バインディングを経て生成される RTL 回路の遅延テスト容易性について評価する．そこで RTL モジュールの単位で遅延故障のテスト容易性を考える．

文献[12]では，遅延故障に対する階層テストを考えており，著者の Wang らは RTL モジュールの 2 パタンテスト容易性について述べている．その文献では，ある RTL モジュールについて，任意の 2 パタンテストのペア（v1,v2）を 2 時刻連続で RTL モジュールに入力でき，更にそのモジュールへの入力 v2 に対する出力応答を外部出力で観測できるならその RTL モジュールは 2 パタンテスト可能であると示されている．つまり，もし RTL モジュールが 2 パタンテスト可能であるならば，遅延テスト環境（Delay Test Environment）と呼ばれる制御系列が存在すると言える．本研究では，RTL モジュールの 2 パタンテスト容易性について同様に考える．

RTL モジュールの 2 パタンテスト容易性について図 3 の RTL データパス回路を例に説明する．ここでは RTL モジュールとして乗算器*2 を考える．乗算器*2 に対する 2 パタンテストを V=（v1,v2）とし，v1 の乗算器*2 の左ポートへの入力を v1L，右ポートへの入力を v1R とする．また v2 についても同様に左ポートへの入力を v2L とし，右ポートへの入力を v2R とする．ある時刻 t で入力 c に v1L を入力 d に v1R を入力する．続いて次時刻 t+1 で入力 e に v2L を入力 f に v2R を入力する．これにより時刻 t で乗算器*2 に v1 が入力され次時刻 t+1 で v2 が入力される．入力変数である c, d，e，f はそれぞれ外部入力となるため，v1 と v2 はそれぞれ任意のパタンを構成できる．また v2 の出力応答の観測については，レジスタ R2 に取り込まれた応答を加算器+1 の右ポートを通して R3 へ転送し，最後に外部出力で観測する．加算器+1 で値を変化させずに通過させるには，その時刻に左ポートの入力を 0 とすることでスルーさせることができる．そのためには，時刻 t で a=1，b= -(c*d)とすることで加算器+1 の左ポートに 0 を入力できる．このことから乗算器*2 は 2 パタンテスト可能であると言える．

3.3 2 パタンテスト可能な演算ペア

前節では 2 パタンテスト可能な RTL モジュールについて述べた．本節では 2 パタンテスト可能な RTL モジュール（演算器）について SDFG に対するバインディングの視点から考える．演算器の入力に任意のパタンを制御することは，SDFG において演算の入力変数を制御することである．以下に制御可能演算を定義する．

定義 1：制御可能演算（*Controllable operation*）
与えられた SDFG のある演算 op について，その入力変数が以下の条件を満たすなら，その演算 op は制御可能である．
条件 1　演算 op の入力変数は外部入力変数である，もしくは他の制御可能演算の出力である．
条件 2　演算 op が 1 つ以上の入力変数を持つなら，それらは個々に制御可能である．つまりそれぞれの入力変数へのコーンが互いに重ならない．

ある SDFG について，2 つの演算 op1 と op2 を考える．この op1 と op2 について，両演算は制御可能演算（以下 CO 呼ぶ）であり，かつ両演算同士に入力変数のデータ依存がないこと，同じ演算の型で 2 時刻連続で実行される，更に op2 の出力は観測できるとき，op1 と op2 を同じ演算器に割り当てることで，その演算器は 2 パタンテスト可能となることが文献[12]に示されている．この op1 と op2 はテスト演算ペア（*Test operation pair*）と呼ばれる．

テスト演算ペア（以下 TOP と呼ぶ）について図 2 の乗算 op2 と乗算 op4 を例に説明する．乗算 op2 の入力は c と d であり，それぞれ外部入力変数であるため op2 は CO である．また同様に op4 の入力 e と f についても外部入力変数であるため op4 は CO である．さらにこれら 4 つの入力変数 c, d, e, f はデータ依存がなく互いに重複しない．乗算 op4 の出力変数については 3.2 節の例で示したように観測可能である．これら 2 つの演算 op2 と op4 は実行時刻が連続しており，演算の型は同じであることから，op2 と op4 を同じ乗算器*2 に割り当てた場合，その乗算器*2 は 2 パタンテスト可能となる．

4. テスト演算ペアを考慮したスケジューリング

前節で述べたとおり，TOP を同じ演算器に割り当てることでその演算器は 2 パタンテスト可能となる．そこで本節では，TOP の数を増やすことを目的としたスケジューリング手法を提案する．

4.1 提案するアルゴリズム

この手法は DFG を入力とし，各演算時刻を決定した SDFG を出力する．リソース制約は演算の種類ごとに使用できる演算器数であり，最適化目標は各演算の種類ごとに TOP 数を最大化することである．図 5 に提案するスケジューリングのアルゴリズムを示す．リソース制約 A については，例えば a_1 は乗算器の個数，a_2 は加算器の個数となっている．

この手法は，与えられた DFG の各演算について定義 1 に示した CO かどうかの判定を行う．演算が CO であるならばその演算の入力は任意の値に制御することができる．続いて各演算についてラベルの設定を行う．このラベルは子孫を持たない（つまり，外部出力変数にのみ直接つながっている）演算のラベルを 1 とし，その演算から親へたどるごとにラベルが 1 つ増加する．ただし，ラベルが複数の場合には最大のものをその演算のラベルとする．

まず始めは時刻 t=1 に配置する演算を考える．演算のタイプ k を決めたら，時刻 1 に配置可能なタイプ k の演算集合 $U_{1,k}$ を候補として選択する．続いてその $U_{1,k}$ の中でラベルが最大の演算の集合 S_k を選択する．もしこのとき$|S_k|$が

```
Proposed_schedule (DFG, Resource constraint A=(a1,a2,..)) {
CO judgment
Label setting
Schedule cycle t=1
repeat{
  for each resource type k=1,2,...n {
      Determine candidate operations U_t,k
      Determine S_k⊆U_t,k ,where S_k is a set of MAX label in U_t,k
      if |S_k| > a_k then TOP_count(S_k)
         Schedule operations at cycle t based on TOP_count(S_k)
      else if |S_k|=a_k then Schedule S_k at cycle t
      else Schedule S_k at cycle t
           TOP_count( U_t,k−S_k)
           Schedule operations at cycle t based on TOP_count( U_t,k−S_k)
  }
  t=t+1
  if all operations are scheduled then END
}

TOP_count( V ){
  Check data dependency of operations
  for each operation v in V
    Count #TOPs when v is located at t
    Count #TOPs when v is located at t+1
  }
  retrun a schedule with the largest #TOPs
}
}
```

図 5　TOP を考慮したスケジューリング

その演算器のリソース制約 a_k より大きい場合には，1 時刻目に配置する演算を S_k の中から決定する．この決定には評価尺度として TOP の数を用いる．関数 TOP_count は，S_k の中の各演算について時刻 t に配置した場合と t+1 に配置した場合の TOP の数をカウントし，TOP の数が最大となるスケジュール結果を返す．この結果に従って時刻 1 に配置する演算を決定する．もし$|S_k|=a_k$ となる場合にはそのまま S_k を 1 時刻目に配置する．また$|S_k| < a_k$ の場合には S_k を時刻 1 に配置し，更に $U_{1,k}$ から S_k を除いた中でラベルが最大の演算について関数 TOP_count を実行する．その結果，時刻 1 に配置した方が TOP 数が多くなる場合にはそのスケジュール結果を採用し，そうでない場合には時刻 1 に配置せずタイプ k 演算の時刻 1 へのスケジュールを終える．続いて同様に異なる演算タイプの時刻 1 への配置を行う．全ての種類の演算について時刻 1 への配置が終わったら時刻 t=2 の処理へ移る．

引き続きまだ配置されていない演算について，時刻 t=2 に配置可能なタイプ k の演算集合 $U_{2,k}$ を候補として選択し，その中でラベルが最大の演算の集合 S_k を決める．これ以降は上述の説明と同様に進め，全ての演算が配置されたらスケジューリングを終了する．

4.2 提案手法の適用例

提案手法の適用例として図 6 の DFG について，リソース制約を乗算器 2 個，加算器 2 個でスケジューリングを行う．

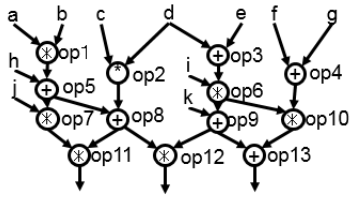

図 6 適用例で使用する DFG

まず始めに各演算が CO であるかの判定を行う．その結果 op11，op12，op13 が CO でないことがわかる．これは演算の入力コーンに重なりがあるためであり，例えば op11 であれば変数 a と b は op11 の両方の入力コーンに含まれてお

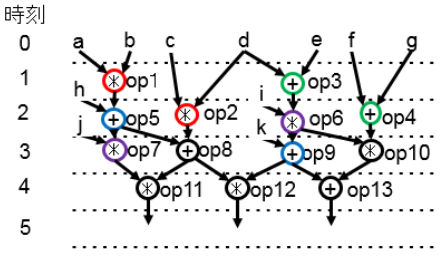

図 7　提案手法を適用した結果

り定義 1 の条件 2 に反する．その後，各演算のラベルを設定し時刻 1 への演算配置を行う．

ここではまず乗算の配置から考える．時刻 1 に配置できる候補としては op1 と op2 が挙げられる．ここでラベルを確認すると op1 のラベルは 4 で op2 のラベルは 3 となる．この場合，最大ラベルの乗算は op1 だけでありリソース制約の 2 個以下であるため時刻 1 に配置する．演算 op2 についてはラベルが 1 つ小さいため，次時刻に置くことも可能である．そこで op2 を時刻 1 に置いた場合と時刻 2 に置いた場合の TOP 数を計算すると，時刻 1 の場合は 0 個，時刻 2 の場合は 1 個（op1 と op2 が TOP）となる．そのため，op2 は時刻 1 に配置しないこととなる．続いて加算 op3 と op4 についても同様に進めると op3 だけが時刻 1 に配置される．

続いて時刻 2 の配置を考える．時刻 2 に配置可能な乗算の候補は op2 と op6 であり，それぞれラベルは 3 である．この場合最大ラベルの乗算は 2 つであり，リソース制約と同数であるため op2 と op6 はそのまま時刻 2 に配置する．同様に加算 op4 と op5 についても時刻 2 に配置する．このような流れで提案手法を最後まで行った結果を図 7 に示す．

提案手法は演算の配置に自由度がある場合に TOP が多くなるよう配置した結果，TOP 数は乗算で 2 つ，加算で 2 つとなった．

5. 評価実験

提案手法の有効性を確認するために，いくつかの DFG を作成し既存の手法であるリストスケジューリングによってスケジュールした SDFG と提案手法によってスケジュールした SDFG を比較した．評価項目は，SDFG 中の TOP 数と実行サイクル数を比較した．また，どちらのスケジューリングについても演算器数に関して同様のリソース制約下で行った．実験の結果は表 1 に示している．

今回の実験結果では，リストスケジューリングによって得られた結果と比較して提案手法はどの DFG でも TOP 数が多くなった．リストスケジュールはリソース制約を満たしながらラベル最大の演算から上から詰めて優先的に配置する手法である．そのため当然ながら偶発的にしか TOP は生まれない．一方，提案手法は配置時刻に自由度がある場合には TOP を積極的に増やすことに成功したと言える．このことから，提案手法で生成した SDFG は後段処理のバインディングにおいて，2 パタンテスト可能な演算器をより多く作ることができる可能性がある．

表 1　実験結果手法

DFG数	スケジュール方法	リソース数	テスト可能演算器数	サイクル数
DFG1 演算規模 加算 9 乗算 11	提案手法	加算器3 乗算器3	加算器3 乗算器3	7
	リスト		加算器2 乗算器3	7
DFG2 演算規模 加算 5 乗算6	提案手法	加算器2 乗算器2	加算器2 乗算器1	5
	リスト		加算器0 乗算器2	5
DFG3 演算規模 加算 7 乗算 8	提案手法	加算器2 乗算器2	加算器2 乗算器2	6
	リスト		加算器2 乗算器0	6
DFG4 演算規模 加算 6 乗算 7	提案手法	加算器2 乗算器2	加算器2 乗算器2	5
	リスト		加算器1 乗算器1	5

6. まとめ

本研究では，遅延故障のテスト容易性を考慮したスケジューリングについて提案した．評価実験では同様のリソース制約下でリストスケジューリングとの比較を行い，提案手法は積極的に TOP を作れることを確認した．これは提案手法で生成した SDFG が，バインディングにおいて 2 パタンテスト可能な演算器を多く生成できる可能性があることを示している．ただし，バインディングにおいて TOP が同じ演算器に割り当てられることは保証されていない．文献[12]のバインディング法は TOP を同じ演算器に割り当てることで遅延テスト容易性を高めることを目的としているため，この手法との接続性を検証することが今後の課題となる． また評価実験ではリストスケジュール以外の手法との比較や実用化されている回路の DFG を使った評価も今後の課題である．

参考文献

[1] A. Krstic and K.T. Cheng, Delay Fault Testing for VLSI Circuits, Kluwer Academic Publishers, 1998.
[2] J. Savir and S. Patel, “Broad-side delay test,” IEEE Trans. on CAD, vol.13, no.8, pp.1057–1064, Aug. 1994.
[3] J. Savir and S. Patel, “Scan-based transition test,” IEEE Trans. on CAD, vol.12, no.8, pp.1232–1241, Aug. 1993.
[4] B.I. Devadas and G.E. Stong, “Design for testability using scanpath techniques for path delay test and measurement,” Proc. International Test Conf., pp.365–374, 1991.
[5] T.J. Chakraborty, V.D. Agrawal, and M.L. Bushnell, “Design for testability for path delay faults in sequential circuits,” Proc. International Conf. DAC, pp.453–457, 1993.
[6] M.A. Amin, S.Ohtake, and H.Fujiwara, “Design for two-pattern testability of controller-data path circuits,” IEICE Trans.on Information and Systems, vol.E86-D, no.6, pp.1042–1050, Jun. 2003.
[7] Y. Yoshikawa, S. Ohtake, M. Inoue, and H. Fujiwara, “Design for testability based on single-port-change delay testing for data paths,” Proc. 14th IEEE Asian Test Symp., pp.254–259, 2005.
[8] B.T. Murray and J.P. Hayes, “Hierarchical test generation using pre computed tests for modules,” IEEE Trans. on Computer Aided Design, vol.9, no.6, pp.594–603, Jun. 1990.
[9] M.T.C. Lee, “High-level test synthesis of digital VLSI circuits,” Artech House Publishers, 1997.
[10] S. Bhatia and N. Jha, “Genesis: A behavioral synthesis system for hierarchical testability,” Proc. EDTC, pp.272–276, 1994.
[11] R.B. Norwood and E.J. McCluskey, “Orthogonal scan: Low-overhead scan for data paths,” Proc. International Test Conf., pp.659–668, 1996.
[12] S.J. Wang and T.H. Yeh, “High-level test synthesis with hierarchical test generation for delay-fault testability,” IEEE Trans. on CAD, vol.28, no.10, pp.1583–1596, Oct. 2009.