ISSN 2186-5647

2-30

レジスタ転送レベルのテスト容易化設計情報を用いたテスト生成法

日大生産工（院）　○中村　健太　日大生産工（院）　石山　悠太
日大生産工　細川　利典

1. はじめに

超大規模集積回路(Very Large Scale Integrated circuits : VLSI)のテスト生成を容易にするためのテスト容易化設計(Design For Testability : DFT)手法としてフルスキャン設計[1]が用いられる．しかしながら，フルスキャン設計では，面積，遅延，消費電力などのハードウェアオーバーヘッドの増加や，テスト実行時間の増加が問題点として挙げられる．

これらの問題点を解決するために，レジスタ転送レベル(Register Transfer Level : RTL)回路に対するDFT 手法が提案されている[2,3]．これらの DFT 手法では，データパスとコントローラから構成される RTL 回路を対象としている．文献[3]では，データパスの構造からテスト容易化機能的時間展開モデル(Easily Testable Functional Time Expansion Model : ETFTEM)と ETFETM を用いたテスト生成手法が提案されている．ETFTEM を用いたテスト生成では，制御信号の入力系列，状態信号の出力系列，時間展開数 k を制約としてテスト生成を行う．ETFTEM はデータパスの構造に基づいて，演算器に対する入出力順序深度[4]が小さくなるように生成される．文献[3]では，ETFTEM の動作を制御可能にするために，状態遷移を追加し，コントローラの拡大を行う．この手法では，追加の状態遷移を用いて演算器をテストするためのテスト系列を正確に生成可能である．しかしながら，演算器以外のモジュールのテスト生成モデルについては提案されていない．

コントローラ拡大[2,3,5,6]とパーシャルスキャン設計[7]に基づく DFT 手法が提案されている[5]. 文献[5]の DFT 手法では，演算器だけでなく回路全体をテストの対象としているため，回路全体で高い故障検出効率を実現している．すべてのハードウェア要素をテストするための ETFTEM が生成され，ETFTEM の動作を実行するためのテスト動作制御・状態信号系列(Test operation Control-Status signal Sequence : TCSS)を生成する．TCSS は，データパスの信号を制御するために必要となる．コントローラの有効状態[6]と無効状態[6]の状態遷移で，ETFTEM に対応する TCSS がデータパスに供給されるようにコントローラ拡大を行う．さらに，コントローラの状態レジスタと，コントローラの入力である状態信号レジスタをスキャン化する．ここで，データパスの制御信号に TCSS を供給するための状態遷移が設計されている状態をテスト活性化状態[8]と呼ぶ．文献[9]では，コントローラ拡大のための面積オーバーヘッドを削減するために，テスト活性化状態の圧縮手法が提案されている．文献[5]では，パーシャルスキャン設計とコントローラ拡大を適用した RTL 回路を論理合成した論理回路に対し一般的なテスト生成ツール(Auto Test Pattern Generator : ATPG)を用いてテスト生成を行っている．しかしながら，コントローラ拡大の情報を用いない一般的な ATPG では，データパスの制御信号に TCSS を供給するための状態遷移を見つけることが難しいため，常に故障検出効率の高いテスト系列が生成されるとは限らない．

本論文では, RTL における RTL-DFT 情報から ATPG 制約を抽出する方法を提案し，抽出された制約に基づく制約付きテスト生成手法を提案する．本手法では，TCSS を用いて RTL データパス回路の構造的記号シミュレーションが実行されると，ETFTEM に対応する時間展開数，対象故障，観測点，観測時刻を ATPG 制約として抽出する．また，実験結果では RTL ベンチマーク回路[10]に対する本手法の有効性を示す．

2. DFT から ATPG までの設計フロー

本章では，RTL-DFT から ATPG までの設計フローについて説明する．図 1 に設計フローを示す． RTL-DFT は，文献[5]で提案されているパーシャルスキャン設計とコントローラ拡大に基づく DFT 手法である．データパスとコントローラで構成される RTL 回路を入力し, TCSS 集合と DFT 適用 RTL 回路を出力する．RTL-DFT の詳細については 3 章で説明する．論理合成では，DFT 適用 RTL 回路から，論理合成を行い，論理回路を生成する．ATPG 制約抽出では，TCSS, DFT 適用 RTL 回路，論理回路から ATPG 制約を抽出する．ATPG 制約抽出の詳細については，4 章で説明する．最後に，故障集合，抽出された ATPG 制約，論理回路を用いて専用 ATPG が実行され，テスト集合が生成される．専用 ATPG の詳細については，5 章で説明する．

3. DFT 手法

本章では, DFT 手法, ETFTEM, コントローラ拡大, RTL-DFT アルゴリズムについて説明する．

3.1. パーシャルスキャン設計

文献[5]の DFT 手法では，コントローラの状態レジスタと，コントローラの入力であるデータパスの状態信号レジスタをスキャン化する．

状態レジスタをスキャン化することで，スキャンテスト時のシフト動作によりコントローラの無効状態を含むすべての状態を初期状態として自由に設定可能となる．ETFTEM の動作は，シフト動作によって到達した初期状態から k サイクルの状態遷移を実行することによって制御可能である．状態信号レジスタをスキャン化することで，接続されているハードウェア要素の可観測性とコントローラの可制御性が向上する．コントローラの入力は状態レジスタから制御可能であるが，出力はデータパスの入力に接続されているため，外部出力でコントローラの故障影響を観測することが困難である．コントローラの出力である制御信号線に EXOR ツリーを挿入し，EXOR ツリーの出力にスキャンレジスタを挿入することで可観測性を向上する．その結果，コントローラ

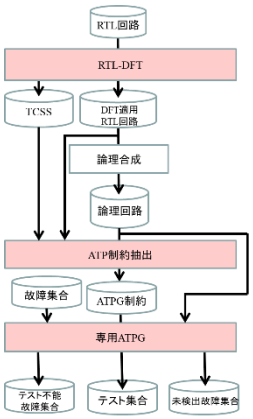

図 1. DFT から ATPG までの設計フロー

A Test Generation Method Using Information of Design for Testability at Register Transfer Level

Kenta NAKAMURA, Yuta ISHIYAMA and Toshinori HOSOKAWA

のテスト容易性は，フルスキャン設計と同等となる．

3.2. テスト容易化機能的 *k* 時間展開モデル

本論文では，時間展開数が *k* の場合，ETFTEM は ETF*k*-TEM と呼ぶ. 系列内の各時刻での制御信号線と状態信号線の値を *k* 時間展開モデルの制約として与えられるとき, そのモデルはテスト容易化機能的 *k* 時間展開モデルと定義する． また，TCSS は以下のように定義する．

＜定義 1：TCSS＞

RTL 回路のテスト動作の制御信号線の値(∈{0、1、*X*})と状態信号線の値(∈{0、1、*b*})の系列を定義する．TCSS において，”*X*”はドントケアを表し，”*b*”は 1 と 0 の両方を表す. TCSS の時刻数は系列の長さとなる．

図 2，表 1，図 3 に，データパス，TCSS，ETF3-TEM の例を示す．図 2 において，*i1* と *i2* は外部入力，*o1* は外部出力，*R1*～*R3* はホールドレジスタ，*R4* はホールド機能のない状態信号レジスタ，*ADD0* は加算器，*SUB0* は減算器，*CMP0* は比較器，*M1*～*M5* はマルチプレクサを示す．また，*m1*～*m5*，*r1*～*r3* は制御信号線を示し, *status* は状態信号線を示す. 表 1 において，1 列目の 0～2 は時刻を示し，2 列目以降は制御信号線と状態信号線の値を示す．図 3 は表 1 に示す TCSS によって動作可能なハードウェア要素で構成される ETF3-TEM を示す．図 3 の左の数字は時刻を表す．文献[3]では，動作可能なハードウェア要素は以下のように定義する．

＜定義 2：TCSS によって動作可能なハードウェア要素＞

ETFTEM において，外部入力か初期時刻のスキャンレジスタからハードウェア要素 *f* の入力に値を伝搬し，*f* の出力から外部出力か最終時刻のスキャンレジスタに値を伝搬するパスが存在する場合， *f* は *TCSS* によって実行可能なハードウェア要素 *f* として定義する．

図 3 では, *ADD0*, *R1*, *R2*, *M1* の *inpput_0* と *input_1*, *M2* の *input_0*，*M3* の *input_0*，*M4* の *input_1* は，表 1 に示す TCSS によって動作可能なハードウェア要素である．ここで，マルチプレクサに *n* 個の入力が存在する場合，マルチプレクサの入力信号線は，左側から順に *input_0*，*input_1*，…，*input_n-1* と表す．

3.3. コントローラ拡大

コントローラ拡大は，コントローラに状態や状態遷移を追加することを示す．コントローラには，機能動作のリセット状態から到達できない無効状態が存在する可能性がある．スキャンテストでは，コントローラはテスト時のシフト動作で無効状態に自由に遷移することが可能である．したがって，文献[5]では，無効状態での状態遷移を設計するコントローラ拡大を行う． 無効状態での状態遷移は，コントローラが TCSS を出力して ETFTEM の動作を実現するように設計する．TCSS には多数のドントケアが存在するため，複数の TCSS をマージすることが可能である．さらに，コントローラのリセット状態から到達可能な有効状態の状態遷移で設計された制御信号にも多数のドントケアが存在するため，有効状態の状態遷移から供給される系列と TCSS をマージできる可能性がある．その結果，TCSS は無効状態の状態遷移だけでなく，有効状態の状態遷移からも供給可能となる．

3.4. RTL-DFT アルゴリズム

図 1 に示す設計フローにおける RTL-DFT のアルゴリズムを図 4 に示す. データパス *DP* とコントローラ *CR* を入力とし，DFT 適用データパス *DP_DFT*，DFT 適用コントローラ *CR_DFT*，TCSS 集合 *TCSS* を出力する．*DP* からターゲットハードウェア要素集合 *MD* を生成する(4 行目)．*MD* は，演算器，外部入力，外部出力，マルチプレクサの入力，マルチプレクサの制御信号線，レジスタの入力，レジスタの出力，レジスタの制御信号線を要素とする．*TCSS* を空集合で初期化する(5 行目)．*MD* が空集合になるまで 7 行目から 11 行目の処理を繰り返す(6 行目)．*MD* からターゲットモジュール tm_i を選択する(7 行目)．tm_i をテストするための ETFTEM $etftem_i$ を生成する(8 行目)．ETFTEM 生成では，時間展開数，再収斂構造数，定数の数が少なくなるように生成する．$etftem_i$ からテスト TCSS $tcss_i$ を生成する(9 行目)．*TCSS* は，*TCSS* と $tcss_i$ の和集合で更新される(10 行目)．$etftem_i$ の実行可能なハードウェア要素を $tcss_i$ から識別する(11 行目)．*TCSS* の一部は *CR* の有効状態の状態遷移とマージされ，残りは無効状態の状態遷移に設計し，*CR* が拡大され *CR_DFT* を生成する(13 行目)．コントローラの入力であるデータパスの状態信号レジスタをスキャン化し，*DP_DFT* を生成する(14 行目)．*CR_DFT* の状態レジスタをスキャン化し, EXOR ツリーとスキャンレジスタを観測点として挿入し, *CR_DFT* を更新する(15 行目)．最後に，*DP_DFT*，*CR_DFT*，*TCSS* を返す(16 行目)．

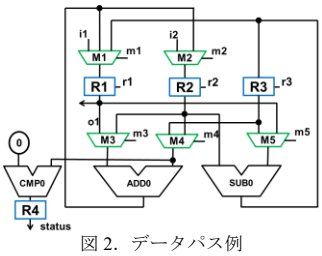

図 2．データパス例

表 1．TCSS

time	r1	r2	r3	m1	m2	m3	m4	m5	status
0	1	1	X	00	0	X	X	X	b
1	1	X	X	01	X	0	1	X	b
2	X	X	X	XX	X	X	X	X	b

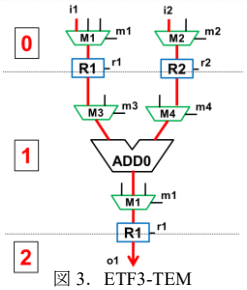

図 3．ETF3-TEM

4. ATPG 制約抽出

本章では，ATPG 制約を抽出する手法を提案する．

4.1. ATPG 制約抽出アルゴリズム

図 1 に示す設計フローにおける ATPG 制約抽出のアルゴリズムを図 5 に示す．DFT 適用データパス *DP_DFT*，DFT 適用コントローラ *CR_DFT*，論理合成後の回路 *LC*，TCSS 集合 *TCSS* を入力とし，ATPG 制約の集合 *Const* を出力する．*Const* を空集合で初期化する(4 行目)．RTL とゲートレベル間の対応表 *corres_table* を生成する(5 行目)．*corres_table* は，各 RTL モジュールに対応するゲートレベルの信号線情報である．モジュール内の信号線の名前が RTL モジュールの名前を部分的に継承するように論理合成を

```
1.  Input: data-path DP, controller CR;
2.  Output: data-path with DFT DP_DFT, controller CR_DFT, a set of test operation control-status
    signal sequences TCSS;
3.  Procedure RTL_DFT (DP, CR) {
4.      MD = collect_module (DP);
5.      TCSS = φ;
6.      while (MD ≠ φ) {
7.          tm_i = select_target_module (MD);
8.          etftem_i = generate_ETFTEM (tm_i);
9.          tcss_i = generate_test_operation_control_status_signal _sequence (etftem_i);
10.         TCSS = TCSS ∪ tcss_i;
11.         MD = delete_concurrent_testable_module ( MD, etftem_i, tcss_i);
12.     }
13.     CR_DFT = controller_augmentation (CR, TCSS);
14.     DP_DFT = datapath_status_signal_register_scan (DP);
15.     CR_DFT = controller_state_register_scan (CR_DFT);
16.     return (DP_DFT, CR_DFT, TCSS);
17. }
```

図 4．RTL-DFT アルゴリズム

行う．例えば，図 2 の ADD0 の場合，ゲートレベルの ADD0 の信号線の名前には「ADD0」が含まれる．したがって，*corres_table* を容易に生成可能である．*TCSS* の各 $tcss_i$ に対し，7 行目と 8 行目の処理を繰り返す(6 行目)．$tcss_i$ を用いて，DP_DFT の構造的記号シミュレーションを実行し，テストのターゲットモジュールを識別する．$tcss_i$ の ATPG 制約 $constraints_i$ は，シミュレーション結果と *corres_table* から抽出する(7 行目)．構造的記号シミュレーションについては，4 章 2 節で説明する．*Const* は，*Const* と $constraints_i$ の和集合で更新する(8 行目)．最後に，*Const* を返す(10 行目)．

4.2. 構造的記号シミュレーション

構造的記号シミュレーションは，ATPG 制約を抽出するために実行する．表 1 に示す TCSS を用いて，図 2 に示すデータパスに対して構造的記号シミュレーションを行った場合のシミュレーション結果を図 6 に示す．図 6 の左側の数字は時刻を表し，水平の破線は時刻の境界を表す．赤色の信号線と黄色の信号線は，制御可能なシンボル(C シンボル)が信号線に割当てられていることを表す．この例では，*i1* と *i2* は外部入力であり，*R4* はスキャンレジスタである．最初に，すべての時刻の *i1*，*i2*，定数 0 と時刻 0 の *R4* に C シンボルを割当て，伝搬する．C シンボルの伝搬規則を以下に示す．

(規則 1)時刻 *t* で演算器のすべての入力信号線に C シンボルが割当てられている場合，時刻 *t* で演算器の出力に C シンボルが割当てられる．

(規則 2)時刻 *t* でマルチプレクサの制御信号線に論理値が割当てられ，かつ制御信号線の論理値に対応するマルチプレクサの入力信号線に C シンボルが割当てられている場合，時刻 *t* でマルチプレクサの出力に C シンボルが割当てられる．

(規則 3)時刻 *t* でレジスタの制御信号線に 1(ロードモード)が割当てられ，かつレジスタの入力信号線に C シンボルが割当てられている場合，時刻 *t*+1 のレジスタの出力に C シンボルが割当てられる．

(規則 4)時刻 *t* でレジスタの制御信号線に 0(ホールドモード)が割当てられ，かつレジスタの出力信号線に C シンボルが割当てられている場合，時刻 *t*+1 のレジスタの出力に C シンボルが割当てられる．

(規則 5)時刻 *t* で分岐元の信号線に C シンボルが割当てられている場合，時刻 *t* で分岐先のすべての信号線に C シンボルが割当てられる．

各時刻の外部出力または最終時刻のスキャンレジスタの入力信号線に C シンボルが割当てられているかを確認し，テストの観測点を判別する．この例では，C シンボルは時刻 1 と 2 で *o1* に割当てられている．したがって，時刻 1 と 2 での *o1* は，テストの観測点として判別する．C シンボルが割当てられた信号線が構造的および機能的に観測点に到達できる場合，その信号線はテスト可能である．図 6 において，赤色の信号線はテスト可能であることを表し，黄色の信号線はテスト不能であることを表す．時刻 1 の *M3* の *input_1* は，時刻 2 の *o1* に構造的に到達可能であるが，機能的に到達不可能である．したがって，*M3* の *input_1* はテスト不能である．テスト可能な信号線で構成される経路上のハードウェア要素もテスト可能となる．図 6 では，*i1*，*i2*，*o1*，*M1* の *input_0*，*M4* の *input_1*，*R1*，*R2*，*ADD0*，*M3* の制御信号線の 1 縮退故障，*M1* の制御信号線の左から 2 番目の bit の 0 縮退故障，*R1* の制御信号線の 0 縮退故障がテスト可能であり，ターゲットハードウェア要素となる．

```
1.  Input: data-path with DFT DP_DFT, controller with DFT CR_DFT, logic circuit LC,
    a set of test operation control-status signal sequences TCSS;
2.  Output: a set of ATPG constraints Const;
3.  Procedure ATPG_Constraints_Extraction (DP_DFT, CR_DFT, LC, TCSS) {
4.      Const = φ;
5.      corres_table = make_correspondence_table (DP_DFT, CR_DFT, LC)
6.      for (each tcss_i ∈ TCSS) {
7.          constraints_i = structual_symbolic_sim (DP_DFT, tcss_i, corres_tbl);
8.          Const = Const ∪ constraints_i;
9.      }
10.     return (Const);
11. }
```

図 5．ATPG 制約抽出アルゴリズム

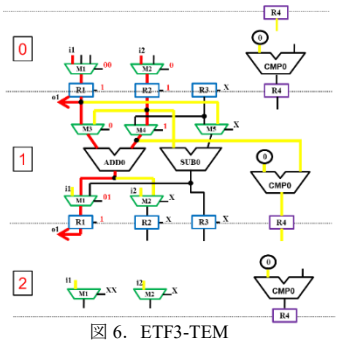

図 6．ETF3-TEM

5. 専用 ATPG

本章では，抽出した制約を使用する専用の ATPG 手法を提案する．まず，ATPG 全体アルゴリズムについて説明し，1 つの ETFTEM に対する制約付き ATPG アルゴリズムについて説明する．

5.1. 専用 ATPG アルゴリズム

図 1 に示す設計フローにおける専用 ATPG のアルゴリズムを図 7 に示す．論理回路 *LC*，抽出された ATPG 制約集合 *Const*，特定の故障集合 *F* を入力とし，生成されたテスト集合 *T*，テスト不能故障集合 *UTF*，未検出故障集合 *UDF* を出力する．まず，*F* と *LC* から組合せ回路用 ATPG を実行し，テスト不能故障を識別し *UTF* を生成する(4 行目)．*UTF* の故障を *F* から削除し，*F* を更新する(5 行目)．*T* を空集合で初期化する(6 行目)．*Const* の制約ごとに，8，9 行目の処理を繰り返す(7 行目)．*F* と *LC* から $constraint_i$ を用いた制約付き ATPG を実行し，テスト系列の集合 T_i を生成する．T_i によって検出された故障を *F* から削除し，*F* を更新する(8 行目)．*T* は，*T* と T_i の和集合で更新する(9 行目)．*F* と *LC* からコントローラの ATPG を実行し，テスト系列の集合 T_c を生成する．T_c によって検出された故障を *F* から削除し，*F* を更新する(11 行目)．*T* は，*T* と T_c の和集合で更新する(12 行目)．残った *F* を *UDF* に代入する(13 行目)．最後に *T*，*UTF*，*UDF* を返す(14 行目)．

5.2. 制約付き ATPG アルゴリズム

図 7 に示す Dedicated_ATPG によって呼び出される 1 つの FTEM に対する制約付き ATPG アルゴリズムを図 8 に示す．論理回路 *LC*，故障集合 *F*，*i* 番目の ATPG 制約 $constraints_i$ を入力とし，生成したテスト系列の集合 T_i，更新した故障集合 *F* を出力する．T_i を空集合で初期化する(4 行目)．制約の固定値によって *LC* に対して含意が実行された後，*LC* のテスト容易性尺度を計算する(5 行目)．故障集合 F_c は，*F* と $constraints_i$ の対象故障の積集合から生成する(6 行目)．*F* から F_c の故障を削除し，更新する(7 行目)．制約下においてテスト不可能な故障も F_c に含まれるが，このような故障は 5 行目で計算されたテスト容易性尺度を用いて制約下のテスト不可能な故障として識別される．F_c が空集合でなく，ATPG が行われていない

```
1.  Input: logic circuit LC, a set of ATPG constraints Const, a fault set F;
2.  Output: a test set T, an untestable fault set UTF, an undetected fault set UDF;
3.  Procedure Dedicated_ATPG (LC, Const, F) {
4.      UTF = Combinational_ATPG (LC, F);
5.      F = F − UTF;
6.      T = φ;
7.      for (each constraints_i ∈ Const) {
8.          (T_i, F) = Constrained_ATPG (LC, F, constraints_i);
9.          T = T ∪ T_i;
10.     }
11.     (Tc, F) = Controller_ATPG (LC, F);
12.     T = T ∪ Tc;
13.     UDF = F;
14.     return (T, UTF, UDF);
15. }
```

図 7．専用 ATPG アルゴリズム

故障が存在する場合，9 行目から 16 行目の処理を繰り返す(8 行目)．F_c から ATPG が行われていない故障 f を選択する(9 行目)．$constraints_i$ の制約のもと f に対し ATPG を実行し，テスト系列 t を生成する(10 行目)．t が存在する場合(11 行目)，F と F_c に対して t ごとに，故障シミュレーションを実行し，検出された故障が F と F_c から削除し，F と F_c を更新する(12 行目)．T_i は，T_i と t の和集合で更新する(13 行目)．F は，F と F_c の和集合で更新する(14 行目)．最後に，T_i と F を返す(17 行目)．

6. 実験結果

本手法は，C 言語で実装され，Intel Xeon E5-1660 v4(3.2FHz)と 32GB メモリを搭載したコンピュータを用いて，RTL ベンチマーク回路[10]に対し実験を行った．制約なし ATPG 手法と，抽出した制約あり ATPG 手法を比較し評価する．制約なし ATPG の場合，時間展開数のみを指定し，テスト生成は時間展開数に対し昇順で段階的に実行した．論理合成は Synopsys 社の DesignCompiler を用いて DFT 手法を適用した RTL 回路に対し実行した．パーシャルスキャン設計には，Synopsys 社の DFTCompiler を使用した．テスト生成には内製の ATPG を用いて故障ごとのバックトラックを 100 として実行した．

表 2 は，DFT 手法の面積オーバーヘッドを示す．この表で，「#SFF」はスキャン FF の数，「FF」はすべての FF の数，「#ETFTEM」は ETFTEM の数，「#TSS」は文献[9]で提案された状態圧縮を使用して設計されたテスト活性化状態の数，「Area」は回路面積，「AOH」は元の回路面積に対する DFT の面積の比率を示す．

表 3 は，提案した ATPG 手法のテスト生成の実験結果を示す．この表で，「k」は生成された ETFTEM の時間展開数，「FTL」は総故障数，「UF」はテスト不能故障数，「ATPG without constraints」は制約なし ATPG の実験結果，「ATPG with constraints」は制約あり ATPG の実験結果，「#DET」は検出故障数，「FC(FE)」は故障検出率(効率)，「#TS」は時刻 k ごとのテスト系列数，「TAT」はテスト実行時間，「CPU」はテスト生成時間を示す．

提案手法では，Sehwa，Maha，Kim の検出故障数がそれぞれ 104，105，70 個増加し，Sehwa と Maha のテスト生成時間をそれぞれ，21%，28%削減することを達成した．しかしながら，本手法では，テスト実行時間が 41～99%増加した．これは，生成された ETFTEM 内のターゲットハードウェア要素数が少ないため，すべてのハードウェア要素をテストするには多くの ETFTEM が必要となる．そのため，テスト実行時間が増加したと考えられる．Kim の ETFTEM の数は，Sehwa，Maha に比べ多いため，テスト実行時間とテスト生成時間が大幅に増加した．ETFTEM をマージすることにより，テスト実行時間の短縮が可能であると考えられる．Sehwa，Maha，Kim には，それぞれ 20，9，47 個の未検出故障が存在した．未検出故障のほとんどはコントローラの故障であり，コントローラの出力の観測点の構造を変えることで，コントローラの未検出故障を検出可能であると考えられる．

```
1.  Input: logic circuit LC, a set of ATPG constraints Const, a fault set F;
2.  Output: a test set T, an untestable fault set UTF, an undetected fault set UDF;
3.  Procedure Dedicated_ATPG (LC, Const, F) {
4.      UTF = Combinational_ATPG (LC, F);
5.      F = F – UTF;
6.      T = φ;
7.      for (each constraints_i ∈ Const) {
8.          (T_i, F) = Constrained_ATPG (LC, F, constraints_i);
9.          T = T ∪ T_i;
10.     }
11.     (Tc, F) = Controller_ATPG (LC, F);
12.     T = T ∪ Tc;
13.     UDF = F;
14.     return (T, UTF, UDF);
15. }
```

図 8．制約付き ATPG アルゴリズム

表 2．DFT の実験結果

Circuits	#SFF/#FF	#ETFTEM	#TSS	Area	AOH（%）
Sehwa	9 / 265	106	62	5929	12.65
Maha	9 / 201	105	67	5248	11.15
Kim	9 / 201	141	61	5906	11.96

7. まとめ

本論文では，RTL-DFT 情報から ATPG 制約を抽出する手法と，抽出した制約に基づき制約付きテスト生成手法を提案した．実験結果から，本手法が制約なし ATPG と比較し，故障検出率の増加とテスト生成時間の短縮を達成した．

今後の課題として，ETFTEM をマージし，テスト実行時間を短縮することが挙げられる．

参考文献

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

[2] L. M. FLottes, B. Rouzeyre, L. Volpe,"A Controller Resynthesis Based Methods for Improving Datapath Testabilty", IEEE International Symposium on Circuits and Systems, pp. 347 -350, May 2000.

[3] T. Masuda, J. Nishimaki, T. Hosokawa and H. Fujiwara, "A Test Generation Method for Data Paths Using Easily Testable Functional Time Expansion Models and Controller Augmentation," Proc. 24th Asian Test Symposium, pp. 37-42, Nov. 2015.

[4] M. Sato, T. Masuda, J. Nishimaki, T. Hosokawa, and H. Fujiwara, "A Binding Method to Generate Easily Testable Functional Time Expansion Models," Digest papers of 17th Workshop on RTL and High Level Testing, Nov. 2016.

[5] Y. Ishiyama, T. Hosokawa and H. Yamazaki," A Design for Testability Method for k-Cycle Capture Test Generation," IEEE 25th Int. Symp. on On-Line Testing and Robust System Design, pp. 40-43, July 2019.

[6] S. Ohtake, T. Masuzawa, and H. Fujiwara, "A non-scan approach to DFT for Controllers Achieving 100% Fault Efficiency," Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 16, No. 5, pp.553-566, Oct. 2000.

[7] S.T. Chakradhar, A. Balakrishnan, and V. D. Agrawal, "An exact algorithm for selecting partial scan flip-flops," Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 7, No. 1, pp.83-93, Aug. 1995.

[8] Y. Takeuchi, T. Hosokawa, H. Yamazaki, and M. Yoshimura, "A Controller Augmentation Method to Improve Transition Fault Coverage for RTL Data-Paths," IEEE 25th Int. Symp. on On-Line Testing and Robust System Design, pp. 293-298, July 2019.

[9] Y. Ikegaya, T. Hosokawa, Y. Ishiyama, and H. Yamazaki, "A Test Sensitization State Compaction Method on Controller Augmentation," IEEE 26th Int. Symp. on On-Line Testing and Robust System Design, 6-3, July 2020.

[10] M.T.-C. Lee, "High-Level Test Synthesis of Digital VLSI Circuits", Artech House Publishers, 1997.

表 3．テスト生成の実験結果

Circuits	k	#FLT	#UTF	ATPG without constraints						ATPG with constraints					
				#DET	FC(%)	FE(%)	#TS	TAT(cycle)	CPU(s)	#DET	FC(%)	FE(%)	#TS	TAT(cycle)	CPU(s)
Sehwa	1,2,3,4	16490	1	16365	99.24	99.25	14, 144, 158, 125	5281	15638.76	16469	99.87	99.88	161, 42, 339, 99	7463	12340.18
Maha	1,2,3,4	15300	2	15184	99.24	99.25	5, 163, 168, 63	4714	18805.89	15289	99.93	99.94	136, 44, 373, 90	7526	13467.34
Kim	1,2,3,4	17854	12	17725	99.28	99.34	5, 121, 145, 59	3924	20438.21	17795	99.67	99.74	166, 55, 394, 62	7835	55969.27