CC-004

遅延テストのための高位合成ツールに関する研究
Studies on a High Level Synthesis Tool for Delay Testing

灰谷 亮† 吉川 祐樹†
Ryo Haitani Yuki Yoshikawa

1. はじめに

近年，半導体集積回路(LSI)の微細化により動作速度やその性能はますます向上している．それに伴い LSI 回路の設計はますます複雑化しており，設計に必要な時間など製造コストの増加が問題視されている．

一般に SoC や FPGA など LSI の設計工程は，上流の仕様設計から始まりその仕様を満たす機能設計を行う．そして高位合成の処理を経てその機能を実現するハードウェアをブロック図レベルで構成する RTL 設計を行う．続いて論理合成によって各ブロックを AND や NAND など論理回路で構成し，レイアウト設計によってトランジスタの配置配線を決定する．LSI 設計は，下流の工程になるほど回路の構成部品数は増加し，人手での設計は時間的にも能力的にも困難となる．そこで下流工程から順にコンピュータによる自動支援設計（CAD）が開発されてきた経緯がある．

最近では動作記述を C 言語や SystemC などで記述する C 言語ベース設計の研究が進み，それに伴って高位合成技術が実用化されるなど上流設計が主流になってきている．高位合成は設計の効率化を行う以外に，回路の高性能化，低消費電力化，再利用性の向上などが期待できる．更に信頼性の向上について，高位合成の段階から高信頼設計を考慮する研究も行われている．

回路の信頼性を保証する技術の 1 つとして，製造された LSI に対して故障の有無を確認するテストがある． LSI のテストを考慮した設計は，産業界で主流となっているスキャン設計[3,4] のように，一般にゲートレベルで行われる．一方，研究レベルでは，ゲートレベルより上位の行程でテスト容易性を考慮する設計手法が提案されている[6-11]．文献 [1]は，高位合成の段階から遅延故障のテスト容易性を考慮したバインディング法を提案している．ここで遅延故障とは，回路のある信号線上の電気信号（0→1，もしくは 1→0 の信号変化）の伝搬が遅れる故障で，その結果システムの誤動作につながる可能性がある．また，我々はバインディングより更に上位行程のスケジューリングの段階から遅延故障のテスト容易性を考慮した設計を提案した[2]．高位合成の段階から設計効率だけでなくテスト容易性も考慮して設計することで，後工程での負担を減らし全体の設計コストを削減することができることを示した．

上述したように，これまで遅延故障のテスト容易性を考慮したバインディングやスケジューリング手法は提案されているが，コントロールデータフローグラフを入力として RTL 回路を出力とする一連の設計フローの実装はまだされていない．本研究では，遅延故障のテスト容易性向上を目的とした高位合成ツールの実装を行う．ただし 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扱わない．チェイニングやマルチサイクル演算を含んだ高位合成ツールは今後の課題とする．

2. 遅延テスト容易化高位合成

高位合成は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．図 1 に一般的な高位合成の手続きを示す．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）を作成する．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，各演算を実際に実行する演算器に，各変数を記憶素子である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 節では遅延テストのためのスケジューリングについて説明し，2.2 節ではバインディングを説明する．続いて 2.3 節ではレジスタ転送レベル回路の生成について述べる．

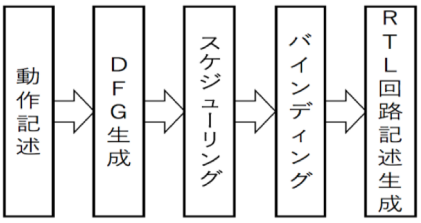

図 1 高位合成の流れ

2.1 遅延テスト容易化スケジューリング

スケジューリングの行程は，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）などがある．

中谷らが提案した遅延テストのためのスケジュール法[2]は，与えられた DFG について 2 パタンテスト可能な演算ペア（Test Operation Pair 以下，TOP） 数が最大の SDFG を出力することを最適化の目標としている．ここで TOP とは，ある SDFG の 2 つの演算 (op1, op2) について，op1 と op2 が同じ型の演算であり，かつ op1 の実行時刻と op2 の実行時刻が 2 時刻連続しており，更に op1 と op2 の全ての入力が制御可能で op2 の出力が観測可能ある演算ペアを言う[1]．TOP について図 2 の SDFG で例を説明する．図 2 の 2 つの演算 OP1 と OP2 を考える．OP1 は加算であり OP2 も加算であるため，OP1 と OP2 は同じ型の演算である．また OP1 の実行時刻は 2 で OP2 の実行時刻は 1 であるため，OP1 と OP2 は 2 時刻連続で実行される．更に OP1 の入力 (v1,v2) と OP2 の入力 (v5,v6) は全て外部入力変数であるため制御可能であり，演算時刻が遅い OP1 の出力は v3=1，v0=0 とすることで観測できる．これより OP1 と OP2 は TOP であると言える．このスケジューリング法によって TOP 数を最大化することで，次のバインディングの行程時に 2 パタンテス

ト可能な演算器をより多く生成することができる．2 パンテスト可能な演算器については 2.2 節で詳細を述べる．

以下に遅延テストのためのスケジューリング法の手順を説明する．まず始めに DFG の各演算が制御可能演算(control operation)であるかを確認する．制御可能演算 (以下, CO)とは，演算の入力変数が外部入力変数もしくは他の制御可能演算の出力であり，かつその演算の入力変数は個々に制御可能である（それぞれの入力変数へのコーンが互いに重ならない）演算を言う．

続いて各演算の処理の優先度を決めるラベルの設定を行う．外部出力変数につながっている演算のラベルを 1 とし，その演算から親にたどるごとに 1 ずつ増加するように設定する．ラベルが複数設定された場合は，よりラベルの大きい方をその演算のラベルとする．

次にラベルの設定と CO の判定，リソース制約をもとに演算の計算時刻を決定する．まず対象とする演算タイプの時刻 t=1 に配置可能な演算の集合 A を候補として選択する．続いてその集合 A の中で，ラベルが最大の演算の集合 B を選択する．このとき集合 B が，対象とする演算器のリソース制約より大きい場合は，時刻 t に配置する演算を集合 B の中から決定する．この時，集合 B の各演算を時刻 t と時刻 t+1 にスケジューリングした時の TOP が最も多くなる組み合わせを見つける．この組み合わせに従って時刻 t に配置する演算を決定する．

集合 B と対象の演算器のリソース制約が等しい場合，集合 B をそのまま時刻 t に配置する．

集合 B が対象の演算器のリソース制約より小さい場合，B をそのまま時刻 t に配置する．その後集合 B の値を除いた，集合 A のラベルが最大の演算を時刻 t と時刻 t+1 にスケジューリングし，その中で TOP が最も多くなる組み合わせを見つける．その結果，時刻 t に配置した方が TOP の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 t に配置せずスケジュールを終える．

まだ時刻 t の演算を行なっていない演算タイプがある場合，対象とする演算を変えて同様の処理を行う．時刻 t における全ての演算タイプの処理を終えている場合，時刻 t+1 の処理へ移り，同様の処理を行う．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．

遅延テストのためのスケジューリング例として，リソース制約が乗算器数 2 つと加算器数 2 つの場合のスケジューリングを図 2 に示す．左のグラフが DFG であり，右のグラフが SDFG となる．この結果を得る手順を説明する．

まず，CO の判定を行う．DFG 中で CO ではないものは op7 のみで，これは v3 を op7 の入力である op3 と op5 が共に必要としているため，op3 と op5 を同時に制御できない．そのため，op7 は CO のための条件を満たせない．次にラベルの設定を行う．外部出力変数につながっている演算のラベルを 1 とし，その演算から親にたどると，外部出力変数につながっている op6，op7 のラベルが 1，op3，op5 のラベルが 2，op1，op4 のラベルが 3，op2 のラベルが 4 になる．続いて演算の計算時刻を決定する．まず，加算器を対象に考える．1 時刻目に計算できる加算器は op1 と op2 である．その中で最大のラベルは op 2 のラベル 4 である．最大ラベルを持つ演算の集合がリソース制約より少ないため，op2 は時刻 1 にスケジューリングされる．次に，op1 を時刻 1 にスケジューリングする場合と時刻 2 にスケジューリングする場合の TOP 数を考える．この場合，時刻 2 にスケジューリングを行う方が op1 と op2 で TOP が組める．すなわち，TOP が増えることがわかる．そのため，時刻 1 ではスケジューリングをしない．次に，乗算器を対象に同様に 1 時刻目のスケジューリングを行う．この時，1 時刻目に計算できる乗算器はないため，スケジューリングをしない．対象になる演算タイプは乗算器，加算器のみなので，時刻 1 のスケジューリングを終了し，時刻 2 のスケジューリングを考える．この手順を繰り返し，全てのスケジューリングを行った結果が図 2 の右図である．TOP は加算 op1 と加算 op2，乗算 op3 と乗算 op 4 の 2 組である．

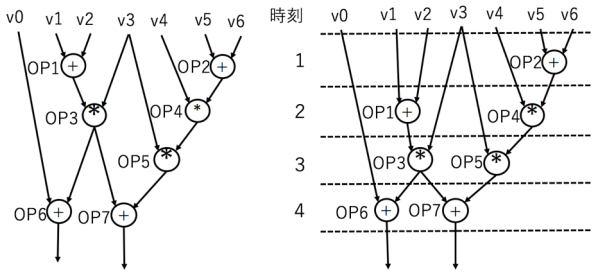

図 2 スケジューリング例

2.2 遅延テスト容易化バインディング

バインディングでは 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．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，実行時間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生成する回路の面積を小さくすることができる．

Wang らが提案した遅延テスト容易な RTL 回路を生成するためのバインディング法 [1] は，TOP を同じ演算器に割り当てることで，その演算器が 2 パタンテスト可能（つまり，その演算器内の遅延故障がテスト可能）となることに着目している．文献[1]では，遅延故障に対する階層テストを考えており，ある演算器について任意の 2 パタンテストのペア（v1, v2）を 2 時刻連続で入力でき，かつその入力 v2 に対する出力応答を外部出力で観測できるなら，その演算器は 2 パタンテスト可能と定義している．つまり，もし演算器が 2 パタンテスト可能であるなら，任意の 2 パタンテストを外部入力からその演算器の入力へ制御し，かつその出力応答を外部出力へ伝搬する遅延テスト環境（Delay Test Environment ）と呼ばれる制御系列が存在すると言える．

提案されたバインディング法は，まず入力の SDFG に対してどの演算ペアが TOP であるかの判定を行い，TOP と判定された演算ペアを各演算器に割り当てる．全ての演算器に対して TOP が割り当てられた場合は，残りの演算についてレフトエッジアルゴリズムを適用する．もし全て演算器に対して TOP を割り当てられない場合には，ダミー演算を SDFG に追加することで TOP を構成し，新たに構成された TOP を演算器に割り当てる処理をする．全ての演算器に

TOP を割り当て 2 パタンテスト可能にすることで，遅延テスト容易な RTL 回路を生成している．

例として図 2 の SDFG を入力としたバインディング結果を図 3 に，その時の演算器とレジスタへの各変数の割り当てを図 4 に示す．リソース制約は加算器が 2 個，乗算器が 2 個とする．この SDFG に関して TOP は OP1 と OP2（加算），OP3 と OP4（乗算）の 2 つ存在する．リソース制約は加算器が 2 個，乗算器が 2 個であるため，リソースに対して TOP が加算と乗算で 1 つずつ不足する．そこで加算と乗算の TOP をそれぞれ 1 つずつ追加する．ここではダミーの演算として OP8 と OP9 を追加する．乗算 OP8 は OP5 と TOP を構成し，加算 OP9 は OP6 と TOP を構成できる．これにより，全ての演算器に対して TOP を割り当てることができる．この時，TOP の 2 時刻目の演算の出力変数である v7,v10,v11 は出力レジスタにバインディングすることとなる．まだ割り当てが決まっていない演算と変数についてはレフトエッジアルゴリズムが適用される．

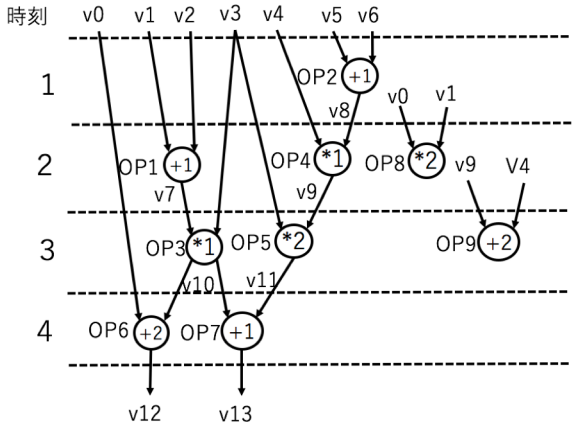

図 3　バインディング例

加算器 1 : op1,op2,op7　　R1: v0
加算器 2 : op6,op9　　R2: v1,v9,v10
乗算器 1 : op3,op4　　R3: v2,v12
乗算器 2 : op5,op8　　R4: v3
　　R5: v4
出力 1　: v11,v12　　R6: v5,v8,v7,v13
出力 2　: v7,v10,v13　　R7: v6,v11

図 4　演算器・各変数の割り当て

2.3 レジスタ転送レベル回路生成

前節で述べた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 5 にバインディング結果から生成した RTL 回路のデータパス部を示す．

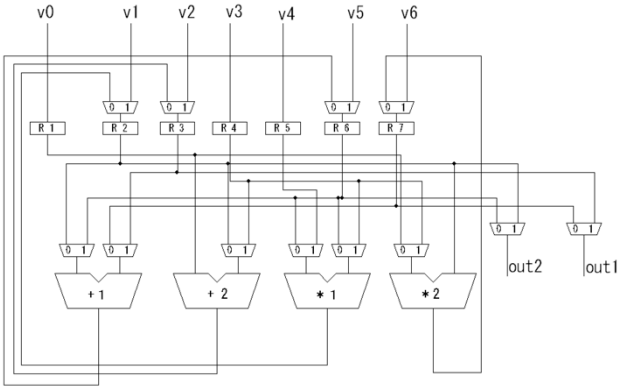

図 5　RTL 回路の例

3. 遅延テスト容易性を考慮した CAD ツールの実装

本研究で実装する CAD ツールは 4 つのパートで構成する．1 つ目は DFG を作成し，その DFG をテキストデータに変換するパートで 3.1 節に述べる．2 つ目は DFG のデータを入力とし遅延テスト容易化スケジューリングを適用するパートで 3.2 節に述べる．3 つ目はそのスケジューリング結果から遅延テスト遅延テスト容易化バインディングを適用するパートで 3.3 節に述べる．ただし，このバインディング法は Wang らによる提案であり，まだ自動化までは実装しておらず現在は手作業となっている．4 つ目は，バインディング結果から RTL 回路のデータパスを生成するパートで 3.4 節に述べる．

これ以降は図 2 に示す DFG から図 5 に示す RTL 回路生成まで，我々が実装したツールで高位合成の処理を行う際の各行程を説明する．図 6 は実装した CAD ツールの画面で，右側が全体図，左側がメニュー部分を拡大した図である．

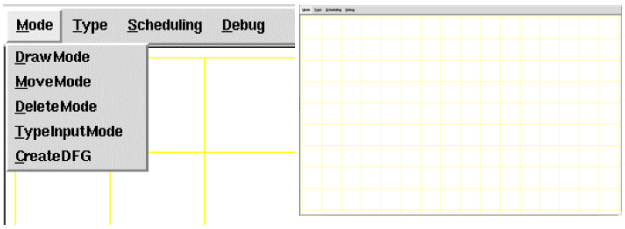

図 6　実装した CAD の UI (左:拡大　右:全体)

3.1 DFG データ作成

DFG の作成は CAD ツール左上のメニュー「Mode」と「Type」の 2 つを使用し，マウス操作でグラフを作成する．「Mode」内の「DrawMode」では，頂点とそれらを接続する辺でグラフを作成する．ここで各頂点は外部入力，外部出力，演算であり，各辺は変数となる．「MoveMode」はすでに配置した頂点の移動ができ，「DeleteMode」では配置した頂点の削除を行うことができる．「TypeInputMode」では，配置した各頂点の種類（外部入力 から I，外部出力 なら O，加算演算なら A，乗算演算なら M など）を設定することができる．これらの機能を使い，DFG を作成する．DFG を作成した後に「Mode」メニュー内の「CreateDFG」を選択することで DFG を確定し，それのテキスト形式データを出力することができる．

実際に図 2 の DFG を 実装した CAD により作成したものが図 7 になる．頂点の種類について外部入力が青，加算や乗算などの演算は赤，外部出力は黒と種類によって色分けされる．また，TypeInputMode で設定した，入出力や演算の種類などの情報と，各頂点に自動的に振り分けられた頂点番号がその右側に表示される．

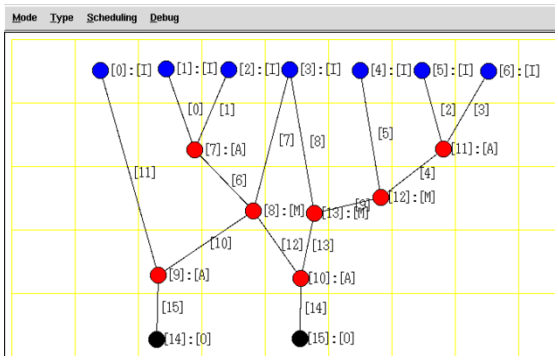

図 7　CAD で作成した DFG

3.2 スケジューリングの実行

前節で作成した DFG に対してスケジューリングを行う．スケジューリングの実行は CAD ツール左上のメニューの「Scheduling」を選択する．この時，プログラム起動時のコンソールからリソース制約の入力が要求されるため，コンソールの指示に従い各演算器のリソース制約を入力する．リソース制約を入力した後，遅延テスト容易化スケジューリングに従って得られたスケジューリング結果を出力する．際に図 7 の DFG をリソース制約として加算器 2，乗算器 2 を与えてスケジューリングした結果，図 8 のような SDFG を得ることができる．

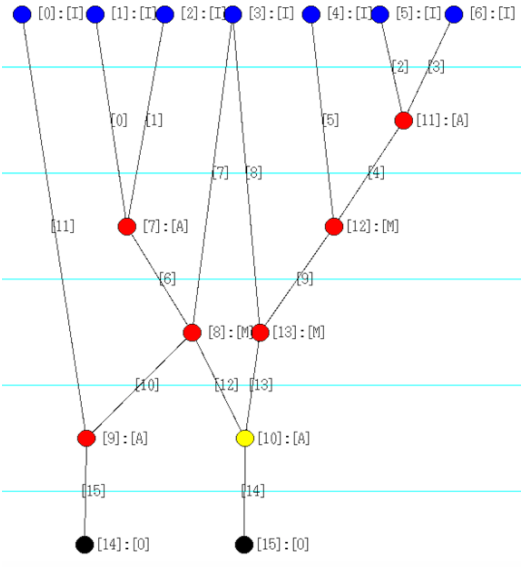

図 8　スケジューリングの結果

3.3 バインディングの実行

バインディングは Wang らの手法を適用する．2.3 節で説明した通り，このバインディング法は TOP を各演算器に割り当てることで，その演算器が 2 パタンテスト可能となる．それにより生成される RTL 回路の遅延故障に対するテスト容易性が向上する．我々が実装したスケジューリングは TOP 数の最大化を最適化目標としているため，このバインディング手法を適用することで，遅延故障のテスト容易性の向上に関して効果が期待できる．ただし，我々のグループで提案した手法ではないため，バインディングに関して自動化はできておらず，現在は手作業で処理をしている．

図 8 のスケジューリング結果を入力としてバンディングを行った結果は図 3 および図 4 となる．バインディング結果の説明は 2.3 節で行っているため，ここでは説明を省略する．

3.4 RTL 回路の自動生成

RTL 回路の自動生成ツールは，前節で出力した各演算の演算器割り当ておよび各変数のレジスタ割り当ての結果と 3.1 節で生成した DFG のテキストデータを入力とする．バインディングによる演算および変数の割り当て結果だけでなく，DFG のテキストデータを入力する理由は，RTL 回路を生成する際に演算器とレジスタの接続関係の情報が必要となるためである．

図 9 に自動生成した RTL 回路を示す．DFG のデータから演算器への入力変数とその出力変数を決定することができる．2.3 節で説明したように，バインディングデータは，SDFG をもとに決定される図 4 のようなデータであり，このデータにはリソース制約や演算の演算器への割り当て，各変数のレジスタへの割り当て，レジスタの個数などの情報を得ることができる．そのため，バインディングデータから，レジスタと演算器の個数，外部入力や演算器の出力先のレジスタを決定することができる．これらのデータから RTL 回路の接続関係が完全に決定する．異なる演算器からの出力が同じレジスタの入力に接続される場合や，複数のレジスタの出力が演算器の同じ入力ポートに接続される場合には，適宜マルチプレクサを配置する．これらの処理をすることで演算器とレジスタ間のインターコネクトを決定し，演算器とレジスタ，マルチプレクサが重ならないように配置配線することで RTL 回路を生成することができる．

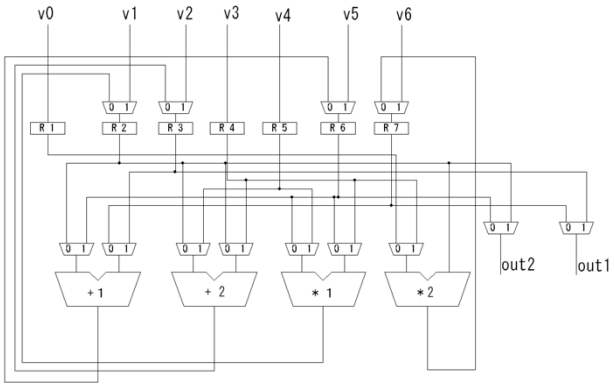

図 9　自動生成した RTL 回路

4. 評価・考察

実装した高位合成ツールの評価および考察を行う．表 1 は評価のために使用した DFG の特性で，1 列目から順に DFG 名，リソース制約，乗算演算数，加算演算数，減算演算数，比較演算数となっている．また表 2 は高位合成の結果生成された RTL 回路の特性を示しており，1 列目から順

に，回路名，乗算器数，加算器数，減算器数，比較器数，レジスタ数，MUX 数となっている．

表 3 は各 DFG を高位合成した時にかかった実行時間を示している．DFG 生成は CAD ツール上で頂点とその接続関係の辺をマウスで描画するため，作成に要する時間は個人差がある．またバインディングは今回まだ自動化できておらず手作業である．そのためこの 2 つの工程に関しては時間を記載していない．スケジューリングにかかった時間はどの DFG も 0.1 秒以下であり，この規模の DFG であれば高速に合成することができる．また RTL 回路の生成には数秒程度かかった．今回検証した DFG ではスケジューリング，RTL 回路生成のいずれも高速に結果を出すことができた．より大規模な DFG を使った検証は今後の課題である．

また，実装した高位合成ツールの使いやすさを評価するために，CAD 開発に関わっていない研究室の学生を対象に意見や感想を聞いた．設計手順に関しては，単純化を目指したこともあり，ツールの操作は直感的で使いやすいという意見があった．問題点として，現在はウィンドウサイズが一定のため大規模回路の設計をする際に，操作性や可読性が悪くなる点が挙げられる．そのため，拡大縮小などの機能の追加も行いたい．

5. まとめ

本研究では，遅延故障のテスト容易性向上を目的とした高位合成ツールの実装を行った．実装した高位合成ツールは，DFG の作成，遅延テスト容易化スケジューリング，遅延テスト容易化バインディング， RTL 回路生成の 4 つの工程からなる．バインディングの自動化は今回実現できておらず今後の課題であるが，それ以外の行程は自動化することができた．また，評価に使った DFG ではスケジューリング，RTL 回路生成の行程は数秒以内と高速に実行できた．より大規模回路での検証は今後の課題である．

表 1　評価のための DFG の特性

DFG名	リソース数	乗算演算数	加算演算数	減算演算数	比較演算数
ex1	乗算器2 加算器2	6	7	0	0
ex2	乗算器2	10	0	0	0
Diffeq	加算器1乗算器3 減算器1比較器 1	6	2	2	1

表 2　生成した RTL 回路の特性

回路名	乗算器数	加算器数	減算器数	比較器数	レジスタ数	MUX数
ex1	2	2	0	0	11	19
ex2	2	0	0	0	6	19
Diffeq	3	1	1	1	13	21

表 3　高位合成にかかった実行時間

回路名	DFG生成	スケジューリング	バインディング	RTL回路生成
ex1		0.026s		2.84s
ex2		0.033s		2.16s
Diffeq		0.016s		1.83s

参考文献

[1] S.J. Wang and T.H. Yeh, “High-level test synthesis with hierarchi- cal test generation for delay-fault testability,” IEEE Trans. on CAD, vol.28, no.10, pp.1583–1596, Oct. 2009.

[2] 中谷夏主政　吉川祐樹: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，情報科学技術フォーラム公演論文集，2016.9

[3] J. Savir and S. Patel, “Broad-side delay test,” IEEE Trans. on CAD, vol.13, no.8, pp.1057–1064, Aug. 1994.

[4] J. Savir and S. Patel, “Scan-based transition test,” IEEE Trans. on CAD, vol.12, no.8, pp.1232–1241, Aug. 1993.

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

[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.