動作合成における順序深度削減指向バインディング法のテスタビリティ評価

日大生産工(学部)　○長 孝昭　　日大生産工　細川 利典

1. はじめに

近年，半導体技術の進展に伴い，大規模集積回路（Large Scale Integration: LSI）の規模や複雑度が飛躍的に増大している. これにより，設計される LSI が急速に大規模化しており，設計が困難になってきている. これを解決するためには, LSI の設計生産性を向上させる必要があり, より高位レベルでの設計が提案されてきた. 高位レベルで設計された回路をレジスタ転送レベル（Register Transfer Level: RTL）に変換する技術として動作合成[2]が存在する. また， LSI 大規模化によって，テストが困難になってきているという問題も生じている.

動作合成では, 主に面積や性能の最適化のためにアルゴリズムが数多く提案されている. 面積や性能の最適化に加えて, テスト容易化も考慮に入れた動作合成アルゴリズムとしてバインディング時に順序深度を削減することにより, テスト容易化を実現する方法が提案されている[1].

本稿では, if 文や while 文などの制御文のない C 言語から，[1]の方法を考慮した動作合成アルゴリズムの実装を検討し, 小規模回路によるテスト容易性の評価を行ったので報告する.

2. 動作合成

動作合成とは，C 言語や SystemC などのプログラミング言語でハードウェアの動作を表現した動作記述を，VHDL や Verilog-HDL といったハードウェア記述言語（Hardware Description Language:HDL）によって記述された RTL 回路記述に変換するものであり，その処理内容は大きくわけると四つのフェーズにわかれる[2]．各フェーズはそれぞれ，CDFG（Control Data Flow Graph）生成，スケジューリング，バインディング，RTL 回路記述生成となる（図 2-1）．CDFG 生成というフェーズでは, 与えられた動作記述をグラフで表現する．グラフにおける頂点は演算を、辺は変数を表す. スケジューリングというフェーズでは, 各演算の依存関係を保ちつつ, 与えられた条件の下，各時刻に演算操作を割り当てる．また，スケジューリングを行うにあたって，CDFG に現れる演算を実現するために, どのような種類の演算器をどれだけ使うかを決める. バインディングでは，スケジューリングされた DFG（Scheduled Data Flow Graph:SDFG）を元にして, 各演算操作や変数に具体的な演算器やレジスタを割り当てる．RTL 回路記述生成では, 結線を実現したデータパスと制御信号を生成するコントローラを生成する．本稿では，if 文や while 文などの制御文のない C 言語を入力とするので, CDFG ではなく DFG を取り扱う.

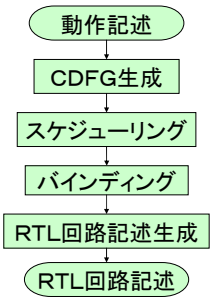

図 2-1.動作合成の流れ

Testability evaluation of binding method based on sequential depth reduction in behavioral synthesis

Takaaki CHO, Toshinori HOSOKAWA

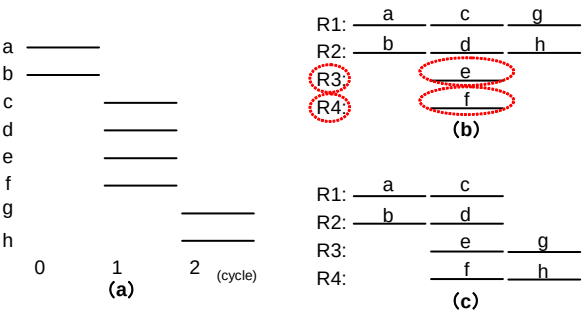

図 4-1.可観測性および可制御性の強化例

```
BBLEA(V1, V2){
1.  foreach(v ∈ V1)
2.      Ri ← v;
3.  foreach(v ∈ V2){
4.      Rpartial = closest_register_to(v)
5.      if(Rpartial == φ ) continue;
6.      VS = find_search_space(Rpartial);
7.      valloc = branch_and_bound(VS);
8.      Rpartial ← valloc;
9.      V2 = V2 - {valloc}; }
10. return(ΠR, V2); }
```

図 5-1.可制御性強化アルゴリズム

```
ralloc(){
1.  (ΠR, Vremain) = BBREA(VO, VM);
2.  (ΠrempR) = BBLEA(VO, VM);
3.  ΠR = merge(ΠR, ΠtempR);
4.  if(Vremain ≠ φ )
5.      ΠR = ΠR ∪ rallocmin(Vremain);
6.  return(ΠR); }
```

図 5-2.レジスタ割り当てアルゴリズム

3. 戦略

順序深度削減指向テスト容易化バインディングの戦略として, 可制御性および可観測性の強化を行う. まず, 可能な限り内部変数を入力レジスタ[1]か出力レジスタ[1]に割り当てる. 入力レジスタとは, 外部入力に直接繋がっているレジスタのことで, このレジスタに内部変数を割り当てると制御可能となる. 出力レジスタとは, 外部出力に直接つながっているレジスタのことで, このレジスタに内部変数を割り当てると観測可能となる. 順序深度[1]とは, 入力レジスタから出力レジスタまでの深さと定義する.

本稿では, 可制御性と可観測性を強化し, 順序深度が削減されるようにバインディングすることを検討する. これにより, 順序回路のテスト生成が容易になり, テスト生成時間を削減されると予測する.

4. 可制御性および可観測性強化の例

図 4-1（a）は, 縦軸は変数名を, 横軸は時刻を表しており, 全体として変数のライフタイムを示している. ライフタイムとは, その変数が値を保持している時間のことであり, ライフタイムが重なっている変数同士はレジスタを共有することができない. よって, レジスタが 4 つ必要であることがわかる. このレジスタに変数を割り当てる際, 何も考慮せず, レフトエッジアルゴリズムを用いて単純に図の上から順に割り当てていくと, R1＝ {a, c, g}, R2＝{b, d, h}, R3＝ {e}, R4＝ {f} となる（**図 4-1 (b)**）. これのバインディング法では, R3 に割り当てられた e と R4 に割り当てられた f が制御も観測も不能となってしまう. しかし, 可制御性および可観測性を強化するようにバインディングをすると, R1＝ {a, c,}, R2＝ {b, d,}, R3＝ {e, g}, R4＝ {f, h} となる. このように変数をレジスタに割り当てることにより, 全変数が制御可能ないしは観測可能ということになる.

5. 順序深度削減指向バインディングアルゴリズム

バインディングを行う前処理として, 動作合成の手順通り, 動作記述をスケジューリングのフェーズまで済ませ, SDFG とする. ここでは, レジスタを R, レジスタの集合を ΠR, 変数を v, 変数の集合を V とする. 順序深度削減指向バインディングの全体のアルゴリズムを ralloc(), 順序深度削減の核となるアルゴリズムを BBLEA（BBREA）とする（図 5-1, 図 5-2）. また, VI を入力変数, VO を出力変数, VM を内部変数とする. 全体の流れとしては, 可能な限り可観測性を強化し, 次に可能な限り可制御性を強化する. また, この過程で順序深度を考慮した割り当てを行う. アルゴリズムを, 例を用いて説明する.

g = (a + b) + (c + d)という動作記述を例に順序深度を考慮しないバインディングと順序深度削減指向バインディングを各々説明する（図 5-3）. 今回は加算器を 2 個使用してよいという制約下で動作合成しているものとする.

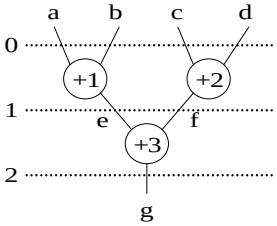

図 5-3.bex2（SDFG）

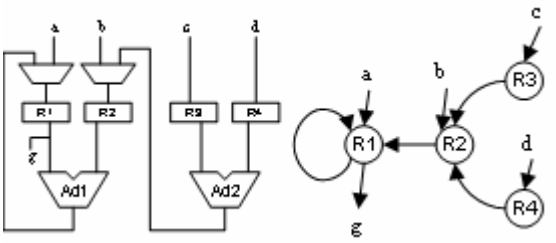

図 5-4.データパスおよび順序深度

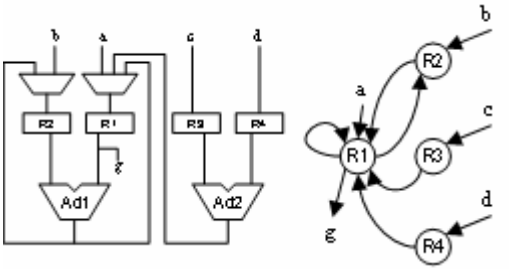

図 5-5.順序深度削減指向
データパスおよび順序深度

まずは, 順序深度を考慮しないバインディングを行う．Vi を i 時刻目の変数とすると，V0＝｛a, b, c, d｝，V1＝｛e, f｝，V2＝｛g｝となる. レジスタは同時刻に異なる値を保持することができないので，必要なレジスタ数は 4 となる. 特に順序深度は考慮しないので, 時刻の早い変数から順番にレジスタに割り当てていく．その結果，R1＝｛a, e, g｝，R2＝｛b, f｝，R3＝｛c｝，R4＝｛d｝となる．さらに，各演算を演算器に割り当てていくと，Add1＝｛+1, +3｝，Add2＝｛+2｝となる．以上のバインディング結果を結線し, データパスを作成した図を図 5-4 に示す.

次に, 順序深度削減指向バインディングを行う．変数はそれぞれ，VI＝｛a, b, c, d｝，VM＝｛e, f｝，VO＝｛g｝となる．可観測性強化のため，出力レジスタの作成を行う（図 5-2, 1 行目）．ここで，BBREA へ飛び，V1＝VO＝｛g｝，V2＝VM＝｛e, f｝とし，アルゴリズムを実行する．BBREA と BBLEA との違いは, 割り当てや探索を行う時のアルゴリズムにて, レフトエッジアルゴリズムを適用するかとライトエッジアルゴリズムを適用するかのみであり, 擬似コードに異なる処理はないため，ここでは BBLEA のアルゴリズムを参照しながら説明する．まず，V1 の信号線をそれぞれレジスタに割り当てる (図 5-1, 1,2 行目). なお, ここでの割り当ては, 仮に行っているだけである．この時点で，R1=｛g｝，V2＝｛e, f｝となっている．次に，closest_register_to という関数の引数 V に V2＝｛e, f｝の中を渡す（図 5-1，4 行目）．戻り値として，e が値を保持し終わる時間に値の保持を始めるレジスタを検索し, そのレジスタを返し Rpartial に代入する．ここで，Rpartial にレジスタが代入されていなければ v を選ぶ処理に戻るが, この例では Rpartial＝R1 となっているので，次の処理へ進む（図 5-1，5 行目）．find_search_space という関数に Rpartial＝R1=｛g｝を送り，戻し値として, Rpartial の R1 が値を保持し始める時間に値を保持しなくなる内部変数を検索し, それらを VS として返す（図 5-1，6 行目）．ここで VS＝｛e,f｝となっている．次に，branch_and_bound という関数に VS＝｛e,f｝を渡し，Rpartial＝R1=｛g｝に割り当てる最適の信号線を探索し，その内部変数を valloc として返す(図 5-1，7 行目)．なお，この例では valloc＝f となる．また，valloc を決定する過程で，演算子+1 と+3 は同じ演算器を共有したほうが順序深度を削減できるという情報を得ているとする．このあと，valloc＝f を Rpatial＝R1＝｛g｝に割り当て Rpatial＝R1＝｛f, g｝となり，V2＝｛e,f｝から valloc＝f を削除し V2＝｛e｝となる（図 5-1，8,9 行目）．さらに 3 行目の処理に戻り，同じ処理を繰り返すが，この例では，これ以上 BBREA において処理不可能なので，ralloc()に戻る．なお戻り値はΠR＝｛(f, g)｝，V2＝Vremain＝｛e｝となっている．次に，可制御性を強化するために，BBLEA アルゴリズムを実行する．ここで，V1＝VI＝｛a, b, c, d｝，V2＝Vremain＝｛e｝となっている．まず，BBREA と同様，V1 の変数をそれぞれレジスタに割り当てる（図 5-1，1,2 行目）．ここでの割り当ても，仮に行っているだけとする．この時点で， R1＝｛a｝，R2＝｛b｝，R3＝｛c｝，R4＝｛d｝，V2＝｛e｝とな

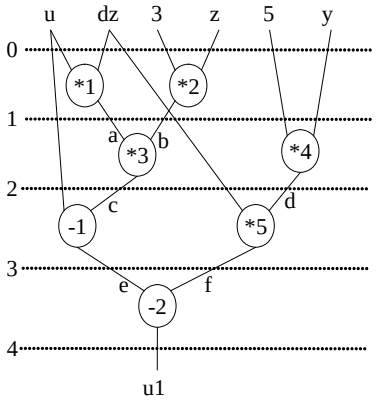

図 6-1.ex2（SDFG）

っている．ここから，BBREA と同様に，あるレジスタに割り当てるのに最適な変数を割り当てる（図 5-1，4-7 行目）．ここでは R2＝｛b｝に v＝｛e｝を割り当てたとする．割り当てた変数 v＝｛e｝を V2＝｛e｝から削除し，未割り当て変数がないか確認し，ralloc()に戻る．この時点で，ΠtempR＝｛(a)，(b, e)，(c)，(d)｝，Vremain＝φとなっている．次に，仮に割り当てられているだけだったレジスタΠR＝｛(f, g)｝とΠtempR＝｛(a)，(b, e)，(c)，(d)｝を merge 関数に送り，共有可能な信号線を保持しているレジスタは結合する．ここでは（f, g）と（a）を結合する．これにより，ΠR＝｛(a, f, g)，(b, e)，(c)，(d)｝となる．次に，まだレジスタに未割り当ての変数があったら，新たにレジスタを作成して割り当てる処理があるが，この例では未割り当て変数がすでにないので，この処理は行われない．この後，演算子を演算器に割り当てるのだが，演算子+1 と+3 を同じ演算器に共有化したほうが順序深度を削減できるという情報を得ているので，各演算を演算器に割り当てると，Add1＝｛+1, +3｝，Add2＝｛+2｝となる．以上のバインディング結果を結線し，データパスを作成した図を図 5-5 に示す．

各々の作成したデータパスから，順序深度を調べると，順序深度を考慮していないバインディングでは R1, R2, R3, R4 がそれぞれ 0, 1, 2，2，である．順序深度削減指向バインディングでは，R1, R2, R3, R4 がそれぞれ 0, 1, 1，1，であることがわかる．

このように，バインディングする際に，入力レジスタおよび出力レジスタに変数を割り当て，順序深度削減指向バインディング法を実行することによって，順序深度を削減することが可能である．

表 6-1.実験結果

circuit	allocation scheme	register allocation	module allocation	test coverage	fault coverage	total faults	ATPG CPU time
bex1	testability	R1=(a, c, f, u1) R2=(b, z) R3=(d, y) R4=(e, u) R5=(dz)	sub1=(−1,−2) mlt1=(*1, *4, *5) mlt2=(*2, *3)	95.95%	90.34%	1336	493.36
	normal	R1=(u, e, u1) R2=(b, c, z) R3=(d, y) R4=(a) R5=(dz, f)	sub1=(−1,−2) mlt1=(*1, *4, *5) mlt2=(*2, *3)	90.87%	85.10%	1134	583.73
ex2	testability	R1=(a, f, g) R2=(b, c) R3=(d) R4=(e)	add1=(+1, +3) add2=(+2)	94.86%	90.92%	3470	3314.67
	normal	R1=(a, c, g) R2=(b, f) R3=(d) R4=(e)	add1=(+1, +3) add2=(+2)	94.61%	90.72%	3504	4163.7

6. 実験結果

順序深度を考慮しないバインディング法を用いた回路と順序深度削減指向バインディング法を用いた回路とのテスト生成結果を比較した．その結果，テスト生成時間も故障検出効率もあげることができた（表 6-1）．

7. おわりに

本稿では，順序深度削減指向テスト容易化バインディングの実装を検討し，小規模回路によるテスト容易性の評価を行った．その結果，テスト容易性の向上が見られた．

今後の課題として今回検討したアルゴリズムを実装する予定である．さらに評価する回路の規模の拡大，また，テスト容易化のためのスケジューリング[3]の実装についても検討する．

参考文献

[1] Tien-Chien Lee, Wayne H. Wolf, Niraj K. Jha, John M. Acken: Behavioral Synthesis for Easy Testability in Data Path Allocation, ICCD, 1992

[2] Daniel D. Gajski : High-Level Synthesis, Kluwer Academic Pub, 1992

[3] Tien-Chien Lee, Wayne H. Wolf, Niraj K. Jha, John M. Acken: Behavioral Synthesis for Easy Testability in Data Path Scheduling, ICCD, 1992