社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

高位合成における可変スケジューリングの近似手法

曽根　康介†　　石浦菜岐佐†

† 関西学院大学 理工学部
〒 669-1337 兵庫県三田市学園 2-1

あらまし　本稿では, 高位合成における可変スケジューリングの近似手法を提案する. オペランドの値に依存して実行サイクル数が変動する演算がある場合, 従来の固定的なスケジューリングでは無駄な待ちが生じる. これに対し, 可変スケジューリングは, 演算器の完了信号を基に各演算の実行タイミングを動的に変更することにより, 効率的なスケジューリングを可能とする. しかし, 可変スケジューリングの結果は, 起こり得る全ての制御ステップから成る状態遷移グラフとして表現され, その状態数が膨大なものとなることがあるため, 合成される回路の規模や性能が損なわれる恐れがあった. 本稿では, 可変スケジューリングにおいて平均サイクル数を犠牲にして回路規模および遅延の増加を抑制する近似手法として, 平均サイクル数の短縮効果に着目した方法と, 状態独立なバインディングに基づく方法を提案する. 評価実験を行った結果, 従来の可変スケジューリングと比較して平均サイクル数は増加するが, 回路規模の指標となる LUT 数と遅延を削減することができた.
キーワード　高位合成, 不定サイクル演算, 可変スケジューリング, 近似計算法

Approximated Variable Scheduling for High-Level Synthesis

Kousuke SONE† and Nagisa ISHIURA†

† School of Science and Technology, Kwansei Gakuin University
2-1 Gakuen, Sanda, Hyogo, 669-1337, Japan

Abstract　This article presents approximated variable scheduling methods for high-level synthesis. In the presence of indefinite cycle operations, which complete their tasks in different cycles depending on the values of their operands, conventional static scheduling often results in inefficient execution. *Variable scheduling* enables efficient computation by adjusting the execution steps of each operation dynamically based on the completion signal from the functional unit. However, the size of the state transition graphs, which are the results of variable scheduling, often grow so large that the area and the delay of the synthesized circuits may not be acceptable. For the purpose of relaxing this problem, we propose two approximate methods which curve the area and the delay of the synthesized circuits at the cost of the average execution cycles in variable scheduling. The first method is based on deletion of the states that do not contribute to the reduction of the execution cycles. The second one is based on state independent binding of operations to functional units, which reduces both the state transition graph size and the datapath complexity. Experimental results show that the size and the delay of the circuits are reduced as compared with the conventional variable scheduling, although the average number of execution cycles is increased.
Key words　high-level synthesis, indefinite cycle operation, variable scheduling, approximate method

1. はじめに

近年, VLSI の設計効率化のため, C 言語などによるハードウェアの動作記述からレジスタ転送レベルの回路を自動合成する高位合成技術 [1] に関する研究が多くなされ, 実用化が進められている.

高位合成の処理過程の中で, 生成されるハードウェアの品質を大きく左右するのがスケジューリングとバインディングである. スケジューリングでは, 各演算を実行する制御ステップを演算間の依存関係に基づき決定し, バインディングでは, 演算と演算の入出力の値に対してそれぞれ演算器とレジスタの割り当てを行う. スケジューリングとバインディングよって, 実行に必要なサイクル数や生成回路の規模, 遅延は大きく異なってくる.

従来のスケジューリングでは, 各演算の実行に要するサイク

ル数は一定であると仮定して演算を制御ステップに割り当てていた. しかし, 演算のサイクル数は必ずしも一定ではなく, メモリアクセス演算や乗除算等では, オペランドの値に依存して実行に必要なサイクル数が変動する場合がある. このような演算がある場合, 仮定したサイクル数よりも早く演算が完了すれば無駄な待ち時間が生じ, 仮定したサイクル数以下で演算が完了しなければ回路全体を停止 (ストール) させる必要がある. これに対し, 可変スケジューリング [2] は, 演算器からの完了信号を基に各演算の実行タイミングを動的に変更することによって効率的な実行を可能にする.

可変スケジューリングの結果は 1 つの制御ステップを 1 つの状態とする状態遷移グラフで表現されるが, 起こり得る全ての制御ステップを網羅するため状態数が著しく増加することがある. これに伴い, 合成されるハードウェアの回路規模や遅延が許容できないほど大きくなる恐れがあった.

この問題を解決するため, 本稿では, 可変スケジューリングにおける近似手法を提案する. これは, 平均サイクル数を犠牲にして状態数を削減することにより, 回路規模と遅延の増加を抑制するものである. 本稿では, 具体的な近似手法として, 平均サイクル数の短縮効果に基づく方法と, 状態独立なバインディングに基づく方法の 2 つを提案する. 前者は, 各状態から終了状態までの平均サイクル数の短縮効果が小さい場合に完了信号による分岐を行わないことにより状態数増加を抑えるものである. 後者は, 演算を実行する演算器を状態に依存せず一意に決定する演算器バインディングを行って, 状態遷移グラフの状態数増加を抑制するものである. これらの手法により, 状態数の増加に伴う制御回路の面積増加を抑制すると同時に, データパスの複雑度を抑制し回路の遅延の短縮を図る.

上記の近似可変スケジューリング・バインディングを実装し, 生成された Verilog HDL 記述を Xilinx ISE 12.3 で論理合成した. 評価実験の結果, 従来の可変スケジューリングと比較して平均サイクル数は増加するが, 回路規模の指標となる LUT 数と遅延を削減することができた.

2. 従来法の可変スケジューリングとバインディング

2.1 不定サイクル演算

実行サイクル数がアドレスやオペランドの値に依存して変化する不定サイクル演算の例として, メモリアクセス演算や乗除算などが挙げられる. メモリアクセス演算では, メモリセルアレイの同じ行に続けてアクセスする場合は高速であるが, そうでない場合にはアクセスに多くのサイクル数を要することがある. 乗除算では, オペランドの値によっては少ないサイクル数で計算を完了するものがある. また, エラー検出・回復方式による演算回路 [3] でもクロックサイクルやオペランドの値に依存して演算のサイクル数が変動する.

本稿では, 演算の実行サイクル数はリストで与えられるものとする. 例えば, $<1,2,3>$ は演算に 1, 2, または 3 サイクル要することを表す. (固定サイクル演算は要素数 1 のリストで表す.)

図 1 (a) の DFG において, f_2, f_3, f_4 は実行サイクル数が $<1,2>$ の不定サイクル演算であり, f_1, f_5 はサイクル数 1 の固定サイクル演算であるとする. この DFG に対し, 従来の静的なスケジューリングを行った結果を (b) と (c) に示す. (b) は実行サイクル数を最大としてスケジューリングしたもので, 演算が早く終わった場合には無駄な待ちが生じる. f_3 が 1 サイクルで完了するとわかっていれば, f_4, f_5 を 1 サイクル繰り上げることができ, 全体の実行サイクル数を 4 サイクルに減らせる. (c) は不定サイクル演算の実行サイクル数を最小としてスケジューリングを行ったものである. ただし, f_2 には 2 サイクル要した場合には, 図のようにハードウェア全体をストールさせる必要がある. f_2 にだけ 2 サイクル要することがわかっていれば, f_1 と f_3 に依存する f_4, f_5 を繰り上げ, 全体の実行サイクル数を 3 サイクルに短縮できる.

従来のスケジューリングでは, このように不定サイクル演算を含む場合に必ずしも効率的な実行を行うことができなかった.

2.2 可変スケジューリング

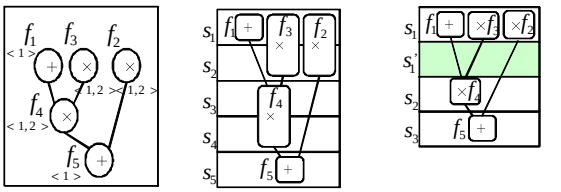
可変スケジューリングは, 演算器からの完了信号を基に各演算の実行タイミングを動的に変更することによって効率的なスケジューリングを行い, 全体の実行サイクル数を減少させる. これは, 予め状態遷移グラフを構築することにより実現する.

図 1 (a) の DFG に対する可変スケジューリングの例を図 2 (a) に示す. c_j は不定サイクル演算 f_j を実行する演算器からの f_j の完了信号を表す. f_j^t は演算 f_j の t サイクル目の実行を表す. 有向枝は状態 (制御ステップ) 間の遷移を表し, 枝のラベルは遷移が起こる条件を表す. 状態 s_1 において, f_2 が 1 サイクルで完了し f_3 が 1 サイクルで完了しない場合, $c_2\bar{c}_3$ の枝をたどって状態 s_7 に遷移する.

この可変スケジューリングにより DFG の実行に必要な平均サイクル数は減少する. しかし, DFG によっては状態数が大幅に増大する可能性がある. これによって合成後のデータパスのマルチプレクサ (MUX) や制御回路の面積も増加する恐れがある.

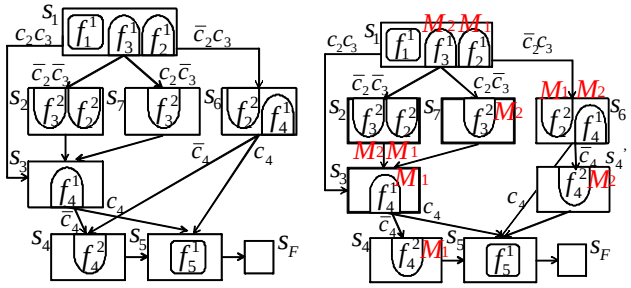
2.3 可変スケジューリングに対応したバインディング

バインディングでは, 演算と演算の入出力値に対してそれぞれ演算器とレジスタの割り当てを行う. 可変スケジューリングに対応したバインディングでは, 状態に依存して各演算に演算器を割り当てる必要がある. 図 2 (a) の状態遷移グラフに対する演算器バインディングの例を (b) に示す. 例えば, s_1 では f_2, f_3 にそれぞれ M_1(乗算器 1), M_2(乗算器 2) を割り当て, s_6 では, f_4 に利用可能な M_2 を割り当てる. また, s_3 では, f_4 に M_1 を割り当てる. このとき, s_6 で f_4 に M_2 を割り当てた場合, s_6 の遷移先である s_4 (f_4 の 2 サイクル目を行う制御ステップ) では, f_4 を M_1 に既に割り当てている. このように, s_4 で f_4 を M_1, M_2 で実行するという衝突が生じる. この場合, s_4 で f_4 の 2 サイクル目を M_1, M_2 のどちらでも行えるように状態を複製し f_4 を M_2 で実行する状態 s_4' を作る必要がある. この結果, 状態数はさらに増加すると同時に, 同じ演算を異なる演算器に割り当てることに伴ってマルチプレクサや遅延が増加する恐れがある.



(a) DFG　(b) サイクル数最大　(c) サイクル数最小

図 1: スケジューリングの結果



(a) スケジューリング　(b) バインディング

図 2: 可変スケジューリング・バインディングの結果

3. 実行サイクル数の短縮効果を考慮した可変スケジューリングの近似手法

本稿では, 近似可変スケジューリングの一手法として, 状態遷移グラフにおいて総サイクル数の改善に貢献しない場合に遷移の分岐を削除することにより, できるだけ性能を低下させずに状態数を削減する手法を提案する.

3.1 スケジューリングにおける状態数削減の着眼点

状態 s からの平均サイクル数 $\sigma(s)$ を s から最終状態までのサイクル数の平均値と定義する. 例えば, 図 3 (a) において, $\sigma(s_5) = 1.0$, $\sigma(s_4) = 2.0$ であり, $\sigma(s_3)$ は両状態 s_4, s_5 への遷移確率が等しいと仮定すると 2.5 となる. 初期状態 s_1 からの平均サイクル数 $\sigma(s_1)$ が全演算の実行に要する平均サイクル数となる. 本近似手法の着眼点は次の通りである.

(着眼点 1) 2 状態の遷移先が一致する場合

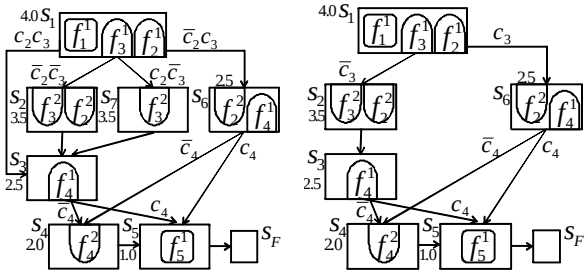
可変スケジューリングにより得られる状態遷移グラフにおいて, ある状態から完了信号 c により分岐した 2 状態に注目する. この 2 状態からの遷移先が全て一致していれば, c による分岐を削除 (遷移を $\bar{c}$ 側に統合) することにより, 平均サイクル数を維持したまま状態数を削減できる. 例えば, 図 3 (a) において, 状態 s_1 から ($\bar{c_3}$ の条件下で) c_2 により分岐する s_2, s_7 に着目する. 両者の遷移先は同じ s_3 なので, この分岐 (完了信号 c_2) による平均サイクル数の短縮効果は得られない. そこで, s_1 から s_7 への遷移を無くし s_2 への遷移に統合する. s_7 は初期状態から到達できない状態になるので削除する. 同様に, s_1 から (c_3 の条件下で) c_2 により分岐する s_3, s_6 に着目すると, 遷移を $\bar{c_2}c_3$ に統合できる. 以上の結果, 実行サイクル数を増加させることなく (b) のように状態数を削減できる.

(着眼点 2) 2 状態からの平均サイクル数が等しい場合

次に (着眼点 1) を一般化し, 分岐の遷移先にある 2 状態 s_x, s_y が $\sigma(s_x) = \sigma(s_y)$ を満たすとき, 両状態からの遷移先が一致していなくても分岐をせずに遷移を統合する. これにより, 平均サイクル数を増加させることなく状態数を削減することができる. 図 4 (a) において, s_8 ($\sigma(s_8) = 8.2$) から c の有無により s_9 と s_{10} に分岐するが, どちらの場合をたどっても, $\sigma(s_9)$, $\sigma(s_{10})$ は 7.2 で等しいので分岐を $\bar{c}$ に統合する. s_{10} が初期状態から到達できなければ s_{10} を削除する.

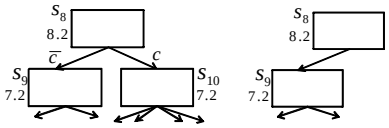
(着眼点 3) 平均サイクル数の短縮効果が小さい場合

さらに, (着眼点 2) の考え方を進め, 分岐の遷移先にある 2 状態 s_x と s_y の平均サイクル数 $\sigma(s_x)$, $\sigma(s_y)$ の差が小さければ平均サイクルの短縮効果が小さいと判断して分岐をせず遷移を統合する. 即ち, $0 \leqq \beta \leqq 1$ なる定数 β に対して, $\sigma(s_y) \geqq \beta \cdot \sigma(s_x)$ であれば分岐を無くし遷移を統合する. 例えば, 図 5 (a) の s_1 から s_2, s_6 への分岐に注目する. s_2, s_6 から終了までの $\sigma(s_2)$, $\sigma(s_6)$ はそれぞれ 3.5, 2.5 サイクルで, それらの比は 0.7 となる. $\beta = 0.7$ とすれば, 図 5 (b) のように s_6 への遷移を無くし s_6 を削除することができる. これにより, $\sigma(s_1)$ は 4.0 サイクルから 4.5 サイクルに増加するが, 状態数をさらに削減することができる.



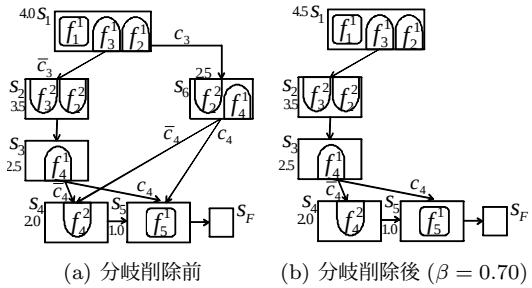
(a) 分岐削除前　(b) 分岐削除後

図 3: 遷移先が一致する場合の状態削除例



(a) 分岐削除前　(b) 分岐削除後

図 4: 平均サイクル数が等しい場合の状態削除例



(a) 分岐削除前　(b) 分岐削除後 ($\beta = 0.70$)

図 5: 平均サイクル数の短縮効果に注目した状態削除例

3.2 実行サイクル数の短縮効果を考慮した近似アルゴリズム

3.1 節の考え方に基づく可変スケジューリングの近似アルゴリズムを図 6 に示す.

01 ～ 05 行目の main では再帰関数 var_scheduling を呼び出して, 与えられた DFG の可変スケジューリングを行う. var_scheduling は, 未スケジューリングの演算の集合 O と, 実

行が完了した演算の集合 O_F を受け取って, 可変スケジューリングを行い, 得られる状態遷移グラフの先頭状態を返す. (ここでは, s_1 が初期状態となる.)

07 ～ 12 行目の var_scheduling では, 08 行目で新たな状態 s を作り, 09 行目で状態 s に実行可能な演算をスケジュールし, 10 行目で s 以降のスケジューリングを行う. SCHEDULE(s, O, O_F) は, O_F の実行が完了しているときに O のうち実行可能なものを s にスケジューリングする. O_S は s でスケジューリングした演算の集合, O_I は s で実行が完了する可能性と完了しない可能性の両方がある演算の集合, O'_F は s で実行が完了する演算の集合である. 関数 next は, O_I, 未スケジューリングの演算の集合 $O - O_S$, s までで完了した演算の集合 $O_F \cup O'_F$ を受け取り, s の次のスケジューリングを行って, s の次状態を返す.

14 ～ 29 行目が関数 next の処理過程である. 15 ～ 17 行目は, 分岐の可能性がない場合の処理であり, var_scheduling を呼び出して次状態以降のスケジューリングを行う. 18 行目以降が, 遷移の分岐を作成する処理である. 19 ～ 21 行目で, O_I の 1 つの要素 f_1 に対して, next を再帰的に呼び出して遷移の分岐を作成する. 20 行目は f_1 が完了しなかった場合の遷移先を, 21 行目は f_1 が完了した場合の遷移先をそれぞれ返す. 22 ～ 27 行目では, 分岐の統合判定を行う. 22 行目において, $\sigma(x_1) \geqq \beta \cdot \sigma(x_0)$ を満たせば, 分岐を作らず, f_1 が完了しなかった場合の遷移 x_0 を返す. そうでなければ, 24 ～ 26 行目において分岐節点 x を作ってこれを返す. 24 行目の BRANCH は, f_1 の完了信号 $c(f_1)$ が得られないときには x_0 に, 得られるときには x_1 に分岐する分岐節点 x を作る. 25 行目では, x の平均サイクル数 $\sigma(x)$ を計算する. $p(\overline{c(f_1)})$ は $c(f_1)$ が得られない確率, $p(c(f_1))$ は $c(f_1)$ が得られる確率を表す.

```
01:  main() {
02:    O = { 全演算 };
03:    O_F = φ;
04:    s_1 = var_scheduling(O, O_F);
05:  }
06:
07:  State var_scheduling(O, O_F) {
08:    s = new State;
09:    (O_S, O_I, O'_F) = SCHEDULE(s, O, O_F);
10:    s.next = next(O_I, O − O_S, O_F ∪ O'_F)
11:    return s;
12:  }
13:
14:  State next(O_I, O, O_F){
15:    if (O_I == φ) {
16:      return var_scheduling(O, O_F);
17:    }
18:    else {
19:      f_1 ∈ O_I;
20:      x_0 = next(O_I − {f_1}, O, O_F);
21:      x_1 = next(O_I − {f_1}, O, O_F ∪ {f_1});
22:      if (σ(x_1) ≧ β · σ(x_0)) { return x_0; }
23:      else {
24:        x = new BRANCH(c(f_1), x_0, x1);
25:        σ(x) = p(c(f_1)) · σ(x_0) + p(c(f_1)) · σ(x_1);
26:        return x;
27:      }
28:    }
29:  }
```

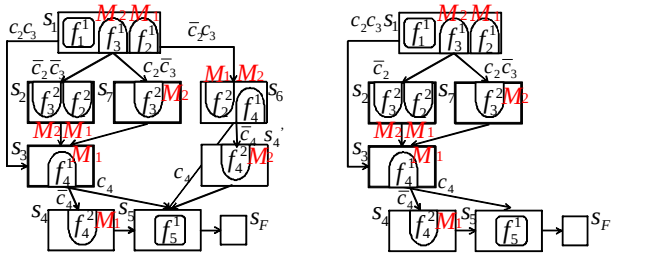
図 6: 可変スケジューリングの近似アルゴリズム

4. 状態独立なバインディングによる可変スケジューリングの近似手法

4.1 バインディングにおける状態数削減の着眼点

2.3 節で述べた通り, 演算の演算器へのバインディングは状態に依存して変化するが, これが状態数の増加だけでなく, データパスの複雑化を引き起こす. 可変スケジューリングに対応したバインディングの結果を図 7 (a) に示す. s_4 では f_4 を M_1, M_2 の両方で実行する可能性があるので (f_4 を s_3 では M_1 に, s_6 では M_2 に割り当てるため), s_4, s'_4 のように状態分割が生じ状態数が増加する. 図 7 (a) から生成されるデータパスは図 8 (a) のようになる. レジスタ $R1$ と $R3$ の値を使う f_4 は状態に依存して M_1, M_2 の両方で行うので, これを切り換えるためのマルチプレクサが必要になり, 回路面積および遅延が増加する.

そこで, 第 2 の近似手法では, 状態に依存せずに各演算を一意に演算器にバインディングし, 矛盾が生じる状態を削除する. これにより, 制御回路の縮小化および演算器への入力部のマルチプレクサの削減による生成回路の面積の削減や遅延の短縮を狙う. 例えば, 図 7 (a) において, f_2, f_4 を M_1 に, f_3 を M_2 に一意 (状態に依存せず) に割り当てるとする. すると, s_6 では f_2, f_4 を同一演算器 M_1 で実行するという矛盾が生じるので s_6 を削除する. ここで, s_6 は s_1 から ($\bar{c}_2$ の条件下で) c_3 により分岐する状態の 1 つなので, 図 7 (b) のように遷移を $\bar{c}_3$ に統合する. これによって, f_4 は状態に依存せず M_1 で実行されるので, MUX3, MUX4 が不要になり, 回路面積と遅延が削減できる. ただし, これによって平均サイクル数は増加する可能性がある.



(a) 可変バインディング　　(b) 状態に依存しないバインディング

図 7: バインディングの結果

4.2 状態独立なバインディングによる可変スケジューリングの近似アルゴリズムの流れ

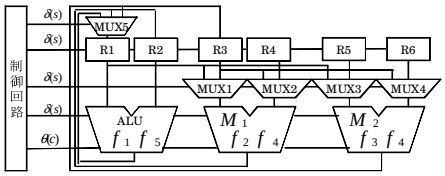
状態独立なバインディングによる可変スケジューリングの状態遷移グラフのサイズの削減は, 次の手順により行える.

(Step 1): 可変スケジューリングを行う.

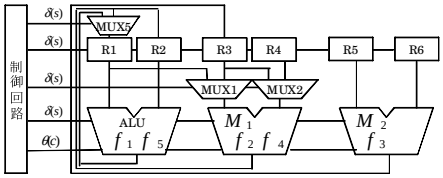
(Step 2): 可変スケジューリングの初期状態から終了状態へ至る最長のパス (遷移の条件が全て完了信号の否定リテラルの積である状態の系列) に対してバインディングを行う.

(Step 3): (Step 2) のバインディングで矛盾が生じる状態 (矛盾状態) を全て削除する. 矛盾状態とは, その状態にスケジューリングされた複数の演算に同一の演算器がバインディングされているような状態である.

(Step 4): 削除された状態に, ある状態 s から完了信号 c をラベ



(a) 可変バインディングによる生成データパス



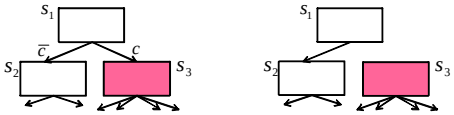
(b) 状態に依存しないバインディングによる生成データパス

図 8: データパス

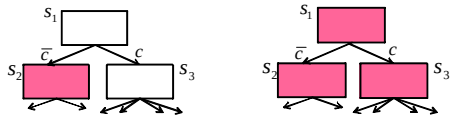
ルに持つ遷移があった場合には, それを s から $\bar{c}$ をラベルに持つ遷移に統合する. 逆に, 削除された状態に, ある状態 s から完了信号 $\bar{c}$ をラベルに持つ遷移があった場合には, s を削除する. (Step 5): (Step 3), (Step 4) によって, 初期状態から到達できなくなる状態を全て削除する.

上記 (Step 4) の例を図 9 に示す. s_1 は c により分岐を持ち, s_2 または s_3 に遷移するものとする. (a) は矛盾状態 s_3 を持つとする. (a) においては, s_3 は削除されているので (b) のように $\bar{c}$ の遷移に統合する. また, (c) は矛盾状態 s_2 を持つとする. (c) においては, s_2 は削除されているので遷移を c の遷移に統合することができない. ゆえに, (d) のように s_1, s_3 も矛盾状態として削除する.

(Step 1) 〜 (Step 5) によって, 図 7 (b) の状態遷移グラフを得る過程を図 10 に示す. スケジューリング結果 (a) に対して, まず, パス $s_1s_2s_3s_4s_5$ にバインディングを行う. その結果, (b) のように, f_2, f_4 は M_1 に, f_3 は M_2 に割り当てたとする. (c) では, s_6 において, f_2 と f_4 の両方を同時に M_1 を割り当てることになるので, s_6 は矛盾状態とし削除する. (d) では, s_1 から ($\bar{c_2}$ の条件下で) $\bar{c_3}$ に遷移を統合する. 結果, 図 7 (b) と同じ状態遷移グラフを得る.



(a) c 側に矛盾状態がある場合　(b) (a) の削除後

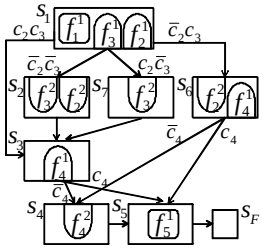


(c) $\bar{c}$ 側に矛盾状態がある場合　(d) (c) の削除後

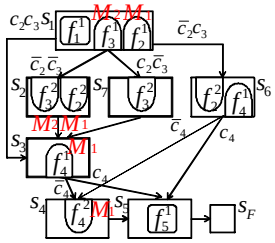
図 9: 状態遷移グラフの矛盾状態の削除例

5. 評価実験

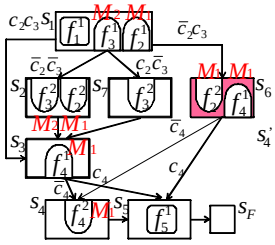
本稿で提案する可変スケジューリングの近似アルゴリズムを高位合成システム ACAP [4] 上に実装し, FPGA Xilinx Spartan3E 上に Xilinx ISE 12.3 で論理合成を行った. ACAP は Perl (5.10.1) で実装されており, Linux, Mac OSX 及び Windows 上の Cygwin 環境で動作する.



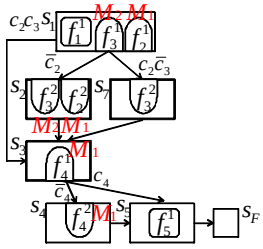
(a) 可変スケジューリングの結果



(b) 最長パスのバインディングの結果



(c) 矛盾状態の発生 (s_6)



(d) 遷移の統合

図 10: 状態に依存しないバインディングの流れ

合成結果を表 1 (a) 〜 (c) に示す. 論理合成で使用した ISE のオプションは表 2 の通りである. matrix41.c は 4 次正方行列と 4 次元ベクトルの積計算, matrix3_g1.c は 3 次正方行列同士の乗算, det3.c は 3 次元の行列式計算をそれぞれ行う C プログラムである. (a) において, 演算器には ALU 3 個, MUL (乗算器) 2 個, LDST (メモリアクセスユニット) 1 個を用いた. 乗算のサイクル数は, 等確率で 1 または 2 になるものとし, 加算とメモリアクセス演算は 1 サイクル固定とした. 従来法を含む 8 種類の手法について, 平均サイクル数, 状態数, LUT 数, 遅延を示している. 「max」は演算の実行サイクル数を最大と仮定としてスケジューリングを行ったものである. 「min」は演算の実行サイクル数を最小と仮定してスケジューリングを行い, そのサイクルで終了しなかったら回路全体をストールさせるものである. 「可変」は従来の可変スケジューリングとバインディングの結果である. 「近似 (1) 」は平均サイクル数を考慮した可変スケジューリングを行ったものであり, β は遷移の統合の基準値を示す. 「近似 (2)」は状態独立なバインディングによる可変スケジューリングの結果である. 「可変」は「max」より約 7% の平均サイクル数を減少させたが, 状態数, LUT 数がそれぞれ約 5.3 倍, 約 2.5 倍増加し, 遅延も約 9.6% も増加した. 近似 (1) において基準値 $\beta = 1.00$ のとき, 「可変」と比較すると LUT 数を 約 6.7% 削減した. β の値を徐々に下げることにより, 回路規模の抑制と遅延を短縮することができた. $\beta = 0.88$ では, 「可変」と比較すると, LUT 数は約 39.5% 削減した. β をある程度まで下げると, 「max」の回路と等価になる. β の設定方法は, LUT 数を許容範囲まで減少させ, 平均サイクル数の短縮効果を維持できる値まで下げることが必要となる. 「近似 (2)」は,「可変」よりも LUT 数を約 29.5% 削減し遅延を約 7.8% 短縮した.

(b), (c) においても, 状態数や LUT 数の減少傾向は (a) と同様である. 特に, (b) では, 「近似 (1)」は,「可変」よりも平均サイクル数が約 6.6% 増加したが LUT 数を約 52.5% 削減で

表 1: 論理合成結果

(a) matrix41.c

	max	min	可変 [2]	近似 (1)				近似 (2)
				$\beta = 1.00$	$\beta = 0.96$	$\beta = 0.92$	$\beta = 0.88$	
平均サイクル数	39.00	40.53	36.14	36.14	36.69	37.08	38.00	37.11
状態数	39	34	206	154	131	78	45	81
LUT 数	2367	2589	6008	5605	5455	5292	3633	4236
遅延 (ns)	11.88	11.77	13.02	13.15	12.85	12.89	12.52	12.01

演算器数: ALU:3, MUL:2, LDST:1

実行サイクル数: $+ < 1 >, * < 1,\ 2 >$, M$< 1 >$ 確率: $+ < 1 >, * < 0.5,\ 0.5 >$, M$< 1 >$

(b) matrix3_g1.c

	max	min	可変 [2]	近似 (1)			近似 (2)
				$\beta = 1.00$	$\beta = 0.98$	$\beta = 0.96$	
平均サイクル数	44.00	45.56	39.39	39.39	39.41	41.98	41.11
状態数	44	35	804	410	408	110	222
LUT 数	3633	3649	13318	7925	8009	6326	6662
遅延 (ns)	12.10	12.07	13.69	13.74	14.50	12.95	13.97

演算器数: ALU:3, MUL:2, LDST:1

実行サイクル数: $+ < 1 >, * < 1,\ 2 >$, M$< 1 >$ 確率: $+ < 1 >, * < 0.5,\ 0.5 >$, M$< 1 >$

(c) det3.c

	max	min	可変 [2]	近似 (1)			近似 (2)
				$\beta = 1.00$	$\beta = 0.95$	$\beta = 0.90$	
平均サイクル数	25.00	21.66	19.08	19.08	19.24	20.4	20.38
状態数	25	16	114	81	69	47	76
LUT 数	1031	911	2319	2045	1925	1641	2025
遅延 (ns)	11.76	11.48	14.87	12.95	15.66	12.12	11.81

演算器数: ALU:1, MUL:2, LDST:1

実行サイクル数:$+ < 1 >, * < 1,\ 2 >$, M$< 1,\ 4 >$ 確率:$+ < 1 >, * < 0.5,\ 0.5 >$, M$< 0.8,\ 0.2 >$

きた. 「近似 (2)」は,「可変」よりも平均サイクル数が約 4.4% 増加したが LUT 数を約 50.0% 削減できた. (c) は,「近似 (2)」の効果があるもので, 遅延が「可変」よりも約 20.6% 短縮した. しかし, 遅延に関しては β を小さくしても, 「近似 (2)」でも改善されない (むしろ悪化する) 場合がある. この原因は判明していないが, 一因としては, 状態数の変化に伴う状態割り当ての変化が考えられる.

状態数と平均サイクル数は概ねトレードオフの関係にある. 生成回路の面積の指標となる LUT 数の増減は, 回路制御部分となる状態数とデータパス上のマルチプレクサ数, 配線数の影響が大きいと考えられる.

表 2: ISE のオプション

FSM Style	LUT
RAM Extraction	NO
ROM Extraction	NO
Mux Style	MUXCY
Mux Extraction	YES
Multiplier Style	AUTO
Automatic Register Balancing	YES
Optimization Goal	Speed

6. むすび

本稿では, 高位合成における可変スケジューリングの近似手法を 2 つ提案した. 本手法により, 平均サイクル数を維持したまま, あるいは犠牲にして, 状態数増加を抑制し, 合成される回路のサイズや遅延を削減できる可能性があることを確認した.

今後の課題として, 可変スケジューリングにおいて生成回路の規模の削減や性能を向上させるための状態符号化について検討する必要がある. また, マルチプレクサによる遅延を削減するための手法を考案する必要がある.

謝辞 本研究を進めるにあたり, 御助言・御討論を頂きました京都高度技術研究所の神原弘之氏, 立命館大学の冨山宏之教授, 元立命館大学の中谷嵩之氏, 元関西学院大学の戸田勇希氏, および関西学院大学石浦研究室の諸氏に感謝します.

文　献

[1] D. D. Gajski, N. D. Dutt, A. C-H Wu, and S. Y-L Lin: *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers (1992).

[2] Y. Toda, N. Ishiura, and K. Sone: "Static Scheduling of Dynamic Execution for High-Level Synthesis," in *Proc. SASIMI 2009,* pp. 107–112 (Mar. 2009).

[3] 右近, 井上, 高橋, 谷口: "エラー検出回復方式における加算器の性能評価," 信学技報, VLD2009-121, pp. 133–137 (Mar. 2010).

[4] 池上, 石浦: "MIPS アセンブリを中間表現とする高位合成," 情処関西支部大会 2008, A-03 (Dec. 2008).