高位合成時のモジュール分割における バッファコスト最小化問題とその解法

大場 諒介[1,a)]　川村 一志[1]　田宮 豊[2]　柳澤 政生[1]　戸川 望[1]

概要：従来 CPU 上で実行していた大規模演算を高速に処理する目的から，ハードウェアアクセラレータ設計に対する需要が高まっている．現在は Xilinx 社 Vivado-HLS 等の商用高位合成ツールが実用段階にあり，ソフトウェアからハードウェアを効率的に合成できる．一方，ツールが一度に高位合成可能なソフトウェアの規模には限界があることから，通常は複数のモジュールに分けてハードウェアを設計する．このような設計方法においては，個々に設計されたハードウェアモジュールを統合する際にタイミング調整用の遅延バッファが必要となるため，挿入されるバッファのコストを考慮してモジュール分割することが求められる．本稿では，高位合成時のモジュール分割によって挿入されるバッファコストを定量化し，バッファコスト最小化問題を定式化する．さらに，バッファコスト最小化問題に対する優良解を効率的に探索する部分結合法を提案する．計算機シミュレーションの結果，貪欲法を用いた探索に比べ，部分結合法は平均 23.5%バッファコストを削減することに成功した．

キーワード：高位合成, FPGA, アクセラレータ, モジュール分割

1. はじめに

従来 CPU 上で実行していた大規模演算を高速に処理する目的から，FPGA を対象としたアクセラレータ設計に対する需要が高まっている [1]．近年の高位合成技術の進歩に伴い，現在は高位合成（HLS: High Level Synthesis）を用いてアクセラレータを設計する場面が増えてきた．FPGA を対象とする場合は特に，Xilinx 社 Vivado-HLS [2] をはじめとする商用高位合成ツールが実用段階にあることから，ソフトウェアコード（動作記述）を効率的にハードウェアコード（RTL 記述）へと変換し FPGA 実装することが可能となっている．

一方，商用ツールを用いて大規模動作記述を高位合成し FPGA 実装する場合，以下のような課題が存在する．

- 演算処理の効率化のためにはハードウェア全体をパイプライン化することが求められるが，動作記述全体をパイプライン化の対象とした場合，対象範囲が大きくなりすぎることで演算処理の実行間隔（II: Initiation Interval）を小さくできず，スループットの低下を招く．
- 大規模動作記述を高位合成することで得た大規模回路を対象にレイアウト合成を実施すると，配線混雑による未配線箇所が多数発生し，レイアウトに失敗するケースが頻発する．

[1] 早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻
Dept. of Computer Science and Communications Engineering, Waseda University
[2] 富士通研究所
Fujitsu Laboratories Ltd.
[a)] ryosuke.oba@togawa.cs.waseda.ac.jp

上記の課題を解決するためには，高位合成の際に入力として与える動作記述のサイズに予め制限を設けておく必要がある．すなわち，大規模動作記述を入力として高位合成を実行する場合，その前工程として動作記述を適当なサイズへと切り分ける「モジュール分割」が必要となる．モジュール分割を導入したアクセラレータ設計の流れを図 1 に示す．図 1 では，大規模動作記述をモジュール分割工程で N 個の小規模な動作記述へと分割した後，各動作記述に対して個別に高位合成を実行している．その後，高位合成により得た N 個の RTL 記述をレイアウト合成時に統合し，FPGA に搭載するハードウェアを得る．図 1 のフローを採用する際に重要な点として，ハードウェア全体でのパイプライン化を前提とし，各モジュールをパイプラインハードウェアとして合成する点がある．

近年の高位合成の研究では，アクセラレータの性能向上を目的としてアクセラレータ設計フローにモジュール分割が取り入れられている．[3] では，C で記述されたソフトウェアコード全体に対する高位合成の結果をもとにモジュール分割を検討している．モジュール分割は動作記述中の潜在的な分割ポイントの探索により実行され，個別の動作記述に対する高位合成の結果得られる各パイプラインハードウェアの II を可能な限り一致させることでスループットの向上を図っている．

上述の通り，モジュール分割によりレイアウト合成が容易となり，ハードウェアのスループットを向上させることができる．一方で，複数のパイプラインハードウェアを統合し所望のスループットを達成するためには，レイアウト合成時に各モジュールの実行タイミングを調整することが求められる．すなわち，モジュール分割を導入した図 1 の

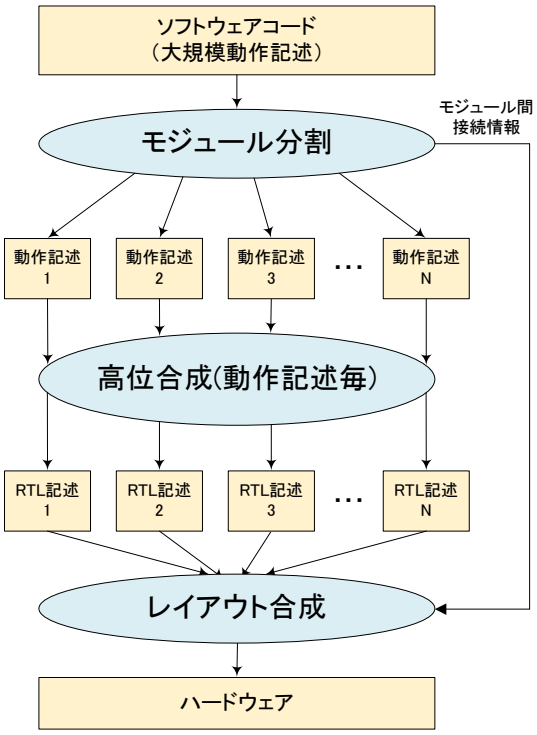

図 1: アクセラレータ設計の流れ.

ような設計フローではレイアウト合成段階でタイミング調整用の遅延バッファが挿入されることになる．このとき挿入される遅延バッファは回路面積や消費電力に影響を与えることから，モジュール分割工程では後々挿入される遅延バッファ量を抑えつつ適切なサイズのモジュールとなるよう動作記述を分割することが重要となる．

以上の背景のもと，本稿では，高位合成時のモジュール分割によって挿入される遅延バッファ量を「バッファコスト」として定量化し，バッファコスト最小化問題を定式化する．さらに，バッファコスト最小化問題に対する優良解を効率的に探索する部分結合法を提案する．計算機シミュレーションの結果，貪欲法を用いた探索に比べ，部分結合法は平均 23.5%のバッファコストを削減することに成功した．

2. バッファコスト最小化問題の定式化

高位合成の前工程としてのモジュール分割を検討するにあたり，まず，高位合成の入力として与えられる大規模動作記述を解析し，動作記述全体を最小単位演算をノードとする演算グラフ G_o で表現する．本稿では，演算グラフ G_o からモジュールグラフ G_m を生成することで，モジュール分割解を表現する．続いて，モジュール分割によって挿入される遅延バッファ量をモジュールグラフ G_m にもとづいて見積り，バッファコスト C として定量化する．

2.1 演算グラフ G_o

演算グラフ $G_o = (\mathbb{N}_o, \mathbb{E}_o)$ はノード集合 $\mathbb{N}_o$ とノード間のデータの流れを表すエッジ集合 $\mathbb{E}_o$ で構成される．ノード集合 $\mathbb{N}_o$ は演算ノード v_i，入力ノード I_i，出力ノード O_i を含む．以降，入力ノードの集合を $\mathbb{I}$，出力ノードの集合を $\mathbb{O}$ と表す．一方，エッジ集合 $\mathbb{E}_o$ はエッジ $E_o(x_i, y_j)$ を含む（$x_i, y_j \in \mathbb{N}_o$）．演算ノード $v_i \in \mathbb{N}_o$ には実行にかかるクロックサイクル数 $c(v_i)$ が，エッジ $e_o = E_o(x_i, y_j) \in \mathbb{E}_o$ には入出力にかかるクロックサイクル数 $c(e_o)$ が予め付与される．簡単のため，本稿ではすべてのエッジの配線幅を同一であると仮定する．

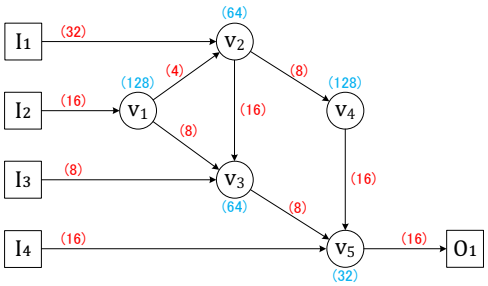

図 2: 演算グラフ G_o の例.

例 1. 演算グラフ G_o の例を図 *2* に示す．図 *2* にあるように演算グラフ G_o は *DAG*（*Directed Acyclic Graph*）で表される．演算ノード v_1〜v_5 は最小単位演算を表し，各々に演算の実行に必要なサイクル数が付与されている．また，ノード間を結ぶエッジはデータの入出力関係を表し，各々にデータの入出力に必要となるサイクル数が付与されている． □

2.2 モジュールグラフ G_m

モジュールグラフ $G_m = (\mathbb{N}_m, \mathbb{E}_m)$ はノード集合 $\mathbb{N}_m$ とノード間のデータの流れを表すエッジ集合 $\mathbb{E}_m$ で構成される．ノード集合 $\mathbb{N}_m$ はモジュールノード V_i の他，$\mathbb{I}$ と $\mathbb{O}$ に含まれるすべてのノードを含む．一方，エッジ集合 $\mathbb{E}_m$ はエッジ $E_m(x_i, y_j, idx)$ を含む（$x_i, y_j \in \mathbb{N}_m$）．[*1]

モジュールグラフ G_m に含まれるモジュールノードのひとつを $V_k \in \mathbb{N}_m$ とすると，V_k は内部に 1 個以上の演算ノードを持つ．モジュールノード V_k 内の演算ノードの集合を $\mathbb{V}_k$ と表すと，演算ノード $v_i \in \mathbb{V}_k$ は V_k 以外のモジュール内部に存在してはならない．また，V_k が内部に持つ演算ノードの個数 $|\mathbb{V}_k|$ は，$1 \leq |\mathbb{V}_k| \leq M$ の範囲で制限される．ここで，M は一度に高位合成可能な演算ノードの最大個数を意味する．

$V_k \in \mathbb{N}_m$ は M 個以下の任意の演算ノードを持つことができる一方，レイアウト合成時にはモジュールの実行順序を明確にする必要があることから，最終的なモジュールグラフの構造は DAG となる必要がある．モジュールグラフ $G_m = (\mathbb{N}_m, \mathbb{E}_m)$ において，エッジ $e_m \in \mathbb{E}_m$ に付加するサイクル数 $c(e_m)$ は演算グラフ G_o 内で対応するエッジ $e_o \in \mathbb{E}_o$ のサイクル数 $c(e_o)$ に等しい．一方，モジュールノード $V_k \in \mathbb{N}_m$ に付加するサイクル数 $c(V_k)$ は別途議論が必要となる（詳細は次節）．

例 2. 図 *3*，図 *4* にモジュールグラフ G_m の例を示す．図 *3* は図 *2* の演算グラフにおいてすべての演算ノードを各々個別モジュールとした場合のモジュールグラフを表す．一方，図 *4* は図 *2* の演算グラフにおいて $\mathbb{V}_1 = \{v1, v2, v3\}$，$\mathbb{V}_2 = \{v4, v5\}$ とした場合のモジュールグラフを表す． □

[*1] モジュールグラフでは同一モジュール間に複数のエッジが存在する可能性があるため，区別のためのインデックス idx をエッジに付与する．

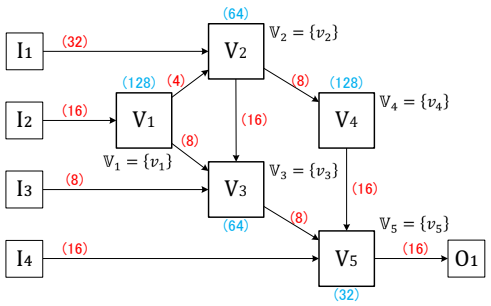

図 3: モジュールグラフ G_m の例 1：図 2 の演算グラフにおいてすべての演算ノードを各々個別モジュールとした場合．

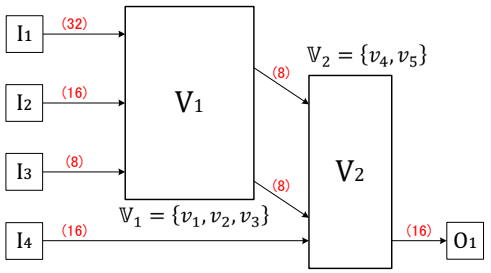

図 4: モジュールグラフ G_m の例 2：図 2 の演算グラフにおいて $\mathbb{V}_1 = \{v1, v2, v3\}$，$\mathbb{V}_2 = \{v4, v5\}$ とした場合．

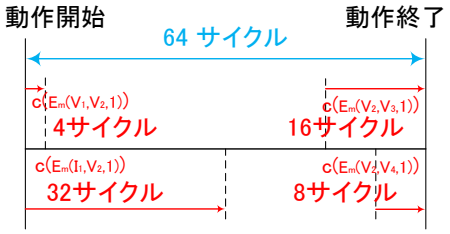

図 5: 図 3 のモジュールグラフにおけるモジュール V_2 の動作の様子．

2.3 モジュールの実行タイミング解析

前章で述べた通り，高位合成はモジュール単位で実行され，各モジュールはパイプラインハードウェアとして合成される．小さい II で動作するハードウェアを設計するため，本稿ではモジュール間のデータ入出力をストリーム形式と想定する．すなわち，モジュールからデータが出力されると同時に後続モジュールへとデータを入力可能である．このような前提のもと，本節では，モジュールグラフ $G_m = (\mathbb{N}_m, \mathbb{E}_m)$ にもとづいて各モジュール $V_k \in \mathbb{N}_m$ の実行タイミングを解析する方法について議論する．

まず，Vivado-HLS [2] を用いた高位合成においては，モジュールへのデータ入力タイミング及びモジュールからのデータ出力タイミングに関して以下の特徴を持つことが実験的に確認されている．

- モジュールの動作が開始すると同時に，モジュールへのデータ入力が開始される．
- モジュールの動作が終了すると同時に，モジュールからのデータ出力が完了する．

Vivado-HLS に則り，本稿ではモジュールが上記の通りに動作するものと想定して議論を進める．

例 3. 図 *3* に示すモジュールグラフ中のモジュール V_2 は図 *5* の通りに動作する．V_2 への入力エッジは $E_m(I_1, V_2, 1)$ と $E_m(V_1, V_2, 1)$ の *2* 本があり，これらのエッジに関わるデータの入力タイミングは V_2 の動作開始タイミングと一致する．また，V_2 からの出力エッジは $E_m(V_2, V_3, 1)$ と $E_m(V_2, V_4, 1)$ の *2* 本があり，これらのエッジに関わるデータの出力タイミングは V_2 の動作終了タイミングと一致する．これらのデータ入出力はストリーム形式であることから，モジュール V_2 からデータが出力されると同時に後続モジュール V_3，V_4 にデータを入力することが可能となる． □

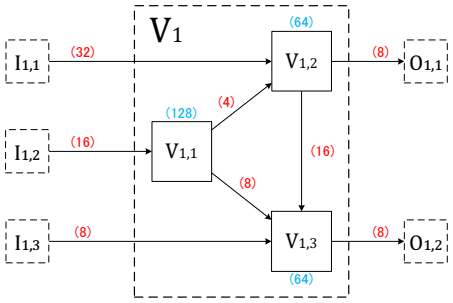

図 6: 図 4 のモジュールグラフにおけるモジュール V_1 内部のサブモジュールグラフ $G_{m,1}$．

以上を踏まえ，モジュール $V_k \in \mathbb{N}_m$ の実行開始タイミング $t_s(V_k)$ と実行終了タイミング $t_f(V_k)$ を式 (1)，式 (2) に従って解析する．

$$t_s(V_k) = \max_{V_l \in \mathbb{V}_{k(in)}} \left(\max_{idx} \left(t_f(V_l) - c(E_m(V_l, V_k, idx)) \right) \right) \quad (1)$$

$$t_f(V_k) = t_s(V_k) + c(V_k) \quad (2)$$

ここで，$\mathbb{V}_{k(in)}$ は V_k に直接エッジで接続された先行モジュールの集合を表し，$\mathbb{V}_{k(in)} = \emptyset$ の場合は $t_s(V_k) = 0$ とする．なお，本タイミング解析では，すべてのモジュール $V_k \in \mathbb{N}_m$ のサイクル数 $c(V_k)$ が必要となる．

本稿では $c(V_k)$ を簡易的に見積もるため，V_k 内部にサブモジュールグラフ $G_{m,k} = (\mathbb{N}_{m,k}, \mathbb{E}_{m,k})$ を仮定する．$G_{m,k}$ は V_k 内部の演算ノードを各々個別モジュールとした場合のモジュールグラフを表し，すべての演算ノード $v_i \in \mathbb{V}_k$ をサブモジュール $V_{k,i}$ に置換して構成する．サブモジュールグラフ $G_{m,k}$ を構成する際には，モジュール V_k への入力にあたる箇所に仮入力ノード $I_{k,i}$ を付加し，モジュール V_k からの出力にあたる箇所に仮出力ノード $O_{k,i}$ を付加する．エッジ集合 $\mathbb{E}_{m,k}$ はエッジ $E_{m,k}(x_i, y_j)$ を含む $(x_i, y_j \in \mathbb{N}_{m,k})$．$G_{m,k}$ では，各サブモジュール $V_{k,i} \in \mathbb{N}_{m,k}$ のサイクル数 $c(V_{k,i})$ が v_i のサイクル数 $c(v_i)$ に等しいものと想定する．$G_{m,k}$ に対して上記と同様の実行タイミング解析を実施し，サブモジュール $V_{k,i} \in \mathbb{N}_{m,k}$ の実行開始タイミング $t_s(V_{k,i})$ と実行終了タイミング $t_f(V_{k,i})$ を計算する．その後，式 (3) を用いてモジュール V_k のサイクル数 $c(V_k)$ を見積もる．

$$c(V_k) = \max_{V_{k,i} \in \mathbb{V}_k^m} (t_f(V_{k,i})) \quad (3)$$

例 4. 図 *4* に示すモジュールグラフ中のモジュール V_1 は内部に *3* 個の演算ノード v_1，v_2，v_3 を持つ．V_1 の実行サイクル数 $c(V_1)$ を見積もるため，V_1 内部に図 *6* のようなサブモジュールグラフ $G_{m,1} = (\mathbb{N}_{m,1}, \mathbb{E}_{m,1})$ を仮定する．$G_{m,1}$ は内部に *3* 個のサブモジュールノード $V_{1,1}$，$V_{1,2}$ $V_{1,3}$ を持ち，式 *(1)* 及び式

(2) から各モジュールの実行終了タイミングは $t_f(V_{1,1}) = 128$, $t_f(V_{1,2}) = 188$, $t_f(V_{1,3}) = 236$ と計算される．従って，式 *(3)* から $c(V_1) = 236$ と見積もられる． □

2.4 バッファコストの算出

実行タイミング解析結果のもと，モジュール分割によって挿入される遅延バッファ量を算出する．

2.4.1 モジュール間遅延バッファ量 B_E の算出

モジュールの実行タイミング調整のためにレイアウト合成時に挿入される遅延バッファをモジュール間遅延バッファと呼ぶ．モジュール間遅延バッファ量 B_E はモジュールグラフ $G_m = (\mathbb{N}_m, \mathbb{E}_m)$ をもとに算出される．遅延バッファが挿入されうる箇所として，(a) 入力ノードとモジュールノードを結ぶエッジ，及び (b) モジュールノード間を結ぶエッジがある．以下，G_m に含まれるエッジのうち (a) に該当するエッジの集合を $\mathbb{E}_m^{(a)}$，(b) に該当するエッジの集合を $\mathbb{E}_m^{(b)}$ と表し，議論を進める．

先述の通り，各モジュールは動作の開始とともにデータ入力を開始する．そのため，各モジュールへのデータ入力タイミングを当該モジュールの動作開始タイミングに一致させなければならず，その調整のために遅延バッファが必要となる．このとき必要な遅延バッファ量 B_E は遅延させる総サイクル数で表され，式 (4) を用いて導出される．

$$\begin{aligned} B_E = & \sum_{e=E_m(I_i,V_j,idx)\in\mathbb{E}_m^{(a)}} (t_s(V_j)) \\ & + \sum_{e=E_m(V_i,V_j,idx)\in\mathbb{E}_m^{(b)}} (t_s(V_j) - t_f(V_i) + c(e)) \end{aligned} \tag{4}$$

2.4.2 モジュール内遅延バッファ量 B_I の算出

モジュール内部の演算の実行タイミング調整のため各モジュールを高位合成する際にはモジュール内に遅延バッファが発生するものと考えられる．本稿では，高位合成時に発生する遅延バッファをモジュール内遅延バッファと呼び，モジュール間遅延バッファと区別する．モジュール内遅延バッファ量 B_I を算出するためには，モジュール毎の遅延バッファ量を見積もる必要がある．

モジュール $V_k \in \mathbb{N}_m$ 内に発生する遅延バッファ量 B_I^k はサブモジュールグラフ $G_{m,k} = (\mathbb{N}_{m,k}, \mathbb{E}_{m,k})$ をもとに見積もられる．遅延バッファが挿入されうる箇所として，(a) 仮入力ノードとサブモジュールノードを結ぶエッジ，(b) サブモジュールノード間を結ぶエッジ，(c) サブモジュールノードと仮出力ノードを結ぶエッジがある．以下，$G_{m,k}$ に含まれるエッジのうち (a) に該当するエッジの集合を $\mathbb{E}_{m,k}^{(a)}$，(b) に該当するエッジの集合を $\mathbb{E}_{m,k}^{(b)}$，(c) に該当するエッジの集合を $\mathbb{E}_{m,k}^{(c)}$ と表し，議論を進める．

Vivado-HLS を用いた高位合成におけるデータ入出力タイミングの性質から，各モジュールは動作の開始とともにデータ入力を開始し，動作の終了とともにデータ出力を完了する．本稿では V_k 内部の演算ノードを各々個別モジュールとして考えるため，$V_{k,i} \in \mathbb{N}_{m,k}$ へのデータ入力タイミングを $V_{k,i}$ の動作開始タイミングに一致させなければならず，その調整のために遅延バッファが必要となる．加えて，モジュール V_k からのデータ出力タイミングを V_k の動作終了タイミングに一致させる必要があることから，(c) に該当するエッジにおいても遅延バッファが発生する可能性がある．以上から，モジュール V_k 内に発生する遅延バッファ量 B_I^k は式 (5) を用いて導出される．

$$\begin{aligned} B_I^k = & \sum_{e=E_{m,k}(I_{k,i},V_{k,j})\in\mathbb{E}_{m,k}^{(a)}} (t_s(V_{k,j})) \\ & + \sum_{e=E_{m,k}(V_{k,i},V_{k,j})\in\mathbb{E}_{m,k}^{(b)}} (t_s(V_{k,j}) - t_f(V_{k,i}) + c(e)) \\ & + \sum_{e=E_{m,k}(V_{k,i},O_{k,j})\in\mathbb{E}_{m,k}^{(c)}} (c(V_k) - t_f(V_{k,i})) \end{aligned} \tag{5}$$

本式でモジュールグラフ G_m に含まれるモジュール毎の遅延バッファ量 B_I^k を導出し，それらの総和（式 (6)）をとることでモジュール内遅延バッファ量 B_I とする．

$$B_I = \sum_k B_I^k \tag{6}$$

2.4.3 バッファコスト C の算出

モジュール間遅延バッファはレイアウト合成時に挿入されるバッファを指すのに対し，モジュール内遅延バッファは高位合成時にツールが自動で発生させるバッファを指す．そのため，モジュール内遅延バッファはモジュール内の演算と合わせて最適化される可能性が高く，実際のバッファ量は 2.4.2 項の方法で見積もった B_I に比べて小さくなると予想される．本稿ではパラメータ p $(0 \le p \le 1)$ を導入し，バッファコスト C を式 (7) で定義する．

$$C = B_E + p \times B_I \tag{7}$$

2.5 バッファコスト最小化問題

これまでの議論をふまえ，バッファコスト最小化問題を以下に定義する．

定義 1. バッファコスト最小化問題とは，演算グラフ $G_o = (\mathbb{N}_o, \mathbb{E}_o)$，モジュール内演算ノードの最大数 M，パラメータ p を入力したとき，バッファコスト C が最小となるようなモジュールグラフ $G_m = (\mathbb{N}_m, \mathbb{E}_m)$ を探索する問題である． □

3. 部分結合法

本章では，バッファコスト最小化問題に対する優良解を効率的に探索する「部分結合法」を提案する．

3.1 方針

バッファコスト最小化問題では，モジュールをどのように構成するかによってモジュール間遅延バッファ量とモジュール内遅延バッファ量が変化し，それに伴ってバッファコストも変化する．最初に，例題を通してモジュール構成とバッファコストの関係について考察する．

例 5. 図 *3* 及び図 *4* に示したモジュールグラフはともに図 *2* の演算グラフをもとに生成されている．まず，図 *3* のモジュールグラフに着目してバッファコストを調査する．モジュールの実行タイミング解析の結果，

$$\begin{aligned} & t_s(V_1) = 0,\ t_f(V_1) = 128,\ t_s(V_2) = 124,\ t_f(V_2) = 188, \\ & t_s(V_3) = 172,\ t_f(V_3) = 236,\ t_s(V_4) = 180,\ t_f(V_4) = 308, \\ & t_s(V_5) = 292,\ t_f(V_5) = 324 \end{aligned}$$

を得る．この結果のもと，式 *(4)* を用いてモジュール間遅延バッファ量を算出すると，$B_E = 704$ となる．続いて，図 *4* のモジュールグラフに着目してバッファコストを調査する．モジュール V_1，V_2 はともに内部に *2* 個以上の演算ノードを持つことから，各々についてサブモジュールグラフを考え，タイミング解析及びモジュール内バッファ量の算出が必要となる．例 *4* にも示したように，モジュール V_1 に対してタイミング解析を実施すると，

$$t_s(V_{1,1}) = 0,\ t_f(V_{1,1}) = 128,\ t_s(V_{1,2}) = 124,\ t_f(V_{1,2}) = 188$$
$$t_s(V_{1,3}) = 172,\ t_f(V_{1,3}) = 236,\ c(V_1) = 236$$

を得る．この結果のもと，式 *(5)* を用いてモジュール V_1 内に発生する遅延バッファ量を算出すると，$B_I^1 = 396$ となる．同様にして，モジュール V_2 に対してタイミング解析を実施すると $c(V_2) = 144$ となり，V_2 内部に発生する遅延バッファ量は $B_I^2 = 224$ と算出される．従って，式 *(6)* によりモジュール内遅延バッファ量 $B_I = 620$ が見積もられる．モジュール間遅延バッファ量については，上記で算出した $c(V_1) = 236$，$c(V_2) = 144$ から $B_E = 228$ と算出される．

以上から，図 *3* の場合のバッファコストを C_1，図 *4* の場合のバッファコストを C_2 とすると，式 *(7)* から

$$C_1 = \ 704$$
$$C_2 = \ 228 + p \times 620$$

となる．この結果は，モジュール内に *2* 個以上の演算ノードを含めることでモジュール間遅延バッファを減らし，モジュール内遅延バッファへと変換できることを示している． □

以降，演算グラフ中のすべての演算ノードを各々個別モジュールとすることで生成されるモジュールグラフを「基本モジュールグラフ」と呼び，モジュール内に複数の演算ノードをまとめる操作を「モジュール結合」と呼ぶ．例 5 にもあるように，遅延バッファの総量（$B_E + B_I$）は基本モジュールグラフの場合に最小となり，モジュール結合により増加する傾向にある．一方で，モジュール間遅延バッファ量 B_E を適切に削減するようなモジュール結合により，遅延バッファの総量を増加させながらもバッファコスト C を削減できる可能性がある．故に，バッファコスト最小化問題を解く際に考慮すべきポイントは，できるだけ多くのモジュール間遅延バッファをモジュール内遅延バッファへと変換しつつ，遅延バッファの総量の増加を抑えることにある．

モジュール間からモジュール内への遅延バッファの変換は，基本モジュールグラフにおいてモジュール間遅延バッファが発生する箇所で行われる．すなわち，遅延バッファの変換を検討する際にはモジュール間遅延バッファの発生箇所に着目したモジュール構成の探索が重要となる．ここで，モジュール間遅延バッファはデータ入力タイミングを調整するためのものであり，入力エッジを複数持つモジュールの前に挿入される．故に，与えられた演算グラフに対し，入力エッジを複数持つ演算ノードを起点としてモジュール結合を検討することにより，バッファコスト最小化問題を効率的に解くことができると考えられる．

一方，遅延バッファの総量が増加する原因として，モジュール結合により演算結果の出力タイミングが遅くなり後続のモジュールの動作開始タイミングを遅らせてしまう点がある．そのため，実行タイミングの近い演算ノード間を中心にモジュール結合を検討することで，遅延バッファの総量の増加を抑えることが期待される．

アルゴリズム 1 モジュール結合の探索．

入力: 位階が設定された演算グラフ $G_o = (\mathbb{N}_o, \mathbb{E}_o)$，$M$, p
1: $L_{max} \leftarrow \max_{v_j \in \mathbb{N}_o}(L(v_j))$
2: 空の演算ノードリスト *list* を用意する．
3: **for** $l \leftarrow 1$ to L_{max} **do**
4: $s \leftarrow$ *list* のサイズ．
5: $r(v_i) = l$ を満たす演算ノード $v_i \in \mathbb{N}_o$ のうち，入力エッジを複数持つものを *list* に挿入する．
6: $s' \leftarrow$ *list* のサイズ．
7: **if** $s' > s$ **then**
8: *list* 内の演算ノード間で M 個以下のモジュール結合の組み合わせを全通り検討し，それぞれの場合でバッファコストを算出する．ただし，モジュール結合がなされていない演算ノードは各々が個別モジュールであるとしてバッファコストを算出する．
9: 現状でバッファコストが最小となる場合のモジュール結合の組み合わせにおいて，複数ノード間でモジュール結合がなされた演算ノードの組をひとつのモジュールとして確定する．
10: 確定したモジュール内の演算ノードを *list* から取り除く．
11: **end if**
12: $r(v_i) = l$ を満たす演算ノード $v_i \in \mathbb{N}_o$ のうち，入力エッジが 1 本のみのものを *list* に挿入する．
13: **end for**
14: *list* 内の演算ノードをそれぞれ個別モジュールとして確定させる．
出力: モジュールグラフ $G_m = (\mathbb{N}_m, \mathbb{E}_m)$

3.2 アルゴリズム

本節では，前節の方針を踏まえ，バッファコスト最小化問題の優良解を効率的に探索するためのアルゴリズムとして「部分結合法」を提案する．部分結合法では，与えられた演算グラフに対して位階を設定することで各演算ノードの実行タイミングを簡易的に見積もり，その後，入力エッジを複数持つ演算ノードのみを起点としてモジュール結合を検討することで解探索時間の短縮を図る．

3.2.1 位階の設定

アルゴリズムの第一段階として，問題の入力として与えられた演算グラフ $G_o = (\mathbb{N}_o, \mathbb{E}_o)$ 中の各演算ノード $v_i \in \mathbb{N}_o$ に位階 $r(v_i)$ を設定する．位階 $r(v_i)$ は演算ノード v_i のおおよその実行タイミングを表す指標となっている．演算グラフ G_o に対し位階を式 (8) で設定する．

$$r(v_i) = \max_{v_j \in \mathbb{N}_o}(L(v_j)) - L(v_i) + 1 \tag{8}$$

ここで，$L(v_i)$ は演算ノード v_i の出力ノードからの距離を表し，ここでの距離は v_i から出力ノードに到達するまでに通過するエッジの最大本数として定義される．

3.2.2 モジュール結合

アルゴリズムの第二段階では，各演算ノードに設定した位階にもとづいて適切なモジュール結合の組み合わせを探索する．本稿では，モジュール結合の探索アルゴリズムとして**アルゴリズム 1** を提案する．

部分結合法では，位階 1 の演算ノードから順に演算グラフを調査する．位階 l において入力エッジを複数持つような演算ノードが発見された場合（アルゴリズム 1 の 7 行目で条件式が真となった場合），その演算ノードに位階 l 未満の演算ノードを加えてモジュール結合の最適な組み合わせを探索する．モジュール結合の組み合わせは全通りが検討され，モジュール結合がなされていない演算ノードは各々

表 1: シミュレーション結果.

演算グラフ（演算ノード数）	M	p	単純法	貪欲法			部分結合法		
			C	$C\ (B_I, B_E)$	N_m	$T\ [msec.]$	$C\ (B_I, B_E)$	N_m	$T\ [msec.]$
演算グラフ 1（9）	3	0.3	384	262.4 (448,128)	5	4	195.2 (544,32)	4	16
		0.5		304 (352,128)	5	12	272 (224,160)	5	16
		0.7		304 (320,80)	5	36	316.8 (224,160)	5	32
演算グラフ 2（10）		0.3	328	156.8 (256,80)	6	32	198.4 (368,88)	6	36
		0.5		208 (256,80)	6	40	272 (368,88)	6	36
		0.7		259.2 (256,80)	6	36	328 (0,328)	8	288
演算グラフ 3（14）		0.3	832	462.4 (528,304)	9	32	401.6 (752,176)	7	452
		0.5		568 (528,304)	8	52	480 (704,128)	7	1932
		0.7		673.6 (528,304)	9	64	620.8 (704,128)	7	1928
演算グラフ 4（25）		0.3	3160	2708.8 (1496,2260)	16	96	2004 (1960,1416)	12	7764
		0.5		2572 (1184,1980)	17	184	2236 (1944,1264)	13	9832
		0.7		3042.4 (392,2768)	17	200	2602.4 (1912,1264)	12	9256
演算グラフ 5（51）		0.3	7344	6492 (1680,5988)	39	1080	5016 (3720,3900)	24	168704
		0.5		7004 (1672,6168)	40	1264	5444 (4216,3336)	24	448080
		0.7		7024.4 (1292,6120)	39	1416	6364 (3640,3816)	23	451808

が個別モジュールであるものとしてバッファコストを見積もる．モジュール結合の結果バッファコストを削減可能である場合にはそのモジュール結合を確定させ，確定した演算ノードに対しては以降検討を行わない．このように，部分結合法は狭い範囲での全探索を繰り返すアルゴリズムであり，探索時間を抑えつつ局所的な最適解を積み重ねてバッファコスト最小化問題の解を得る手法である．

4. 計算機シミュレーション

本章では，部分結合法の有効性検証のため，バッファコスト最小化問題の解探索アルゴリズムを計算機上でシミュレーションする．本シミュレーションに用いた計算機環境は Intel Core i7-4650U CPU，実装メモリ 16.0 [GB] であり，C 言語で実装したアルゴリズムを Ubuntu 16.04 による Linux 仮想環境上で実行させた．本シミュレーションでは，以下に示す 3 通りの手法で得た解（モジュールグラフ）についてバッファコスト C，モジュール数 N_m，探索にかかる時間 T を比較する．

1. **単純法**: 与えられた演算グラフから基本モジュールグラフを生成する手法．各モジュールが内部に持つ演算ノードが 1 個のみであるため，モジュール内に遅延バッファが発生しない．故に，$C = B_E$ である．
2. **貪欲法**: 演算グラフから M 個以下の互いにエッジで接続される演算ノードをランダムに抽出し，モジュール結合を検討する手法．モジュール結合を検討する際には結合前後でバッファコストを比較し，コストの削減が見込まれる場合に限り結合を採用する．このとき生成されたモジュール内の演算ノードは以降の探索に含めない．本手法の探索はバッファコストの削減が見込めなくなるまで繰り返す．
3. **部分結合法**: 3 章で提案した手法.

本シミュレーションでは，問題の入力となる演算グラフをグラフ自動生成ツール [4] を用いて生成し，演算グラフ中の演算ノード及びエッジにはランダムでクロックサイクル数を付与した．クロックサイクル数はすべて 2^m（m は自然数）で表される数とし，演算ノードに対しては $4 \leq m \leq 9$，エッジに対しては $1 \leq m \leq 6$ の範囲に収まる数とした．演算グラフは演算ノード数が異なる 5 種類を用意し，それぞれに対して貪欲法と部分結合法を用いて解を導出した．本シミュレーションでは，モジュール内演算ノードの最大数は $M = 3$ とし，パラメータ p が 0.3，0.5，0.7 それぞれの場合で探索を行なった．

シミュレーション結果を表 1 に示す．表 1 から，演算グラフ 1（$p = 0.7$），演算グラフ 2（$p = 0.3$，0.5，0.7）を除いて部分結合法を適用したときのバッファコストが貪欲法を適用したときのバッファコストよりも小さくなる結果を得た．特に，演算ノード数の多い大規模な演算グラフを入力とした場合に部分結合法で優れた結果が得られる傾向にある．モジュール数 N_m に着目すると，演算グラフ 4 や演算グラフ 5 では部分結合法の結果が小さくなっており，効果的なモジュール結合がなされていると考えられる．全体としては，貪欲法を用いた探索に比べ，部分結合法は平均 23.5%のバッファコストを削減することに成功した．

5. おわりに

本稿では，高位合成時のモジュール分割におけるバッファコスト最小化問題を定式化するとともに，バッファコスト最小化問題に対する優良解を効率的に探索する部分結合法を提案した．計算機シミュレーションの結果，貪欲法を用いた探索に比べ，部分結合法は平均 23.5%バッファコストを削減することに成功した．

今後の課題として，モジュール内演算ノードの最大数 M を変更した場合の結果を得ること，部分結合法の探索時間を削減することが挙げられる．また，実際のソフトウェアコードを用いた検証を通して，バッファコストの削減が回路面積や消費電力に与える影響を調査する必要がある．

参考文献

[1] 若林一敏, “ソフトウェアプログラムからハードウェア記述を合成する高位合成技術,” *IEICE Fundamental Review*, vol. 6, no. 1, pp. 37–50, 2013.

[2] “Xilinx vivado hls,” http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[3] B. C. Schafer, “Automatic partitioning of behavioral descriptions for high-level synthesis with multiple internal throughputs,” *Proceedings of Electronic System Level Synthesis Conference*, 2013.

[4] “Task graph for free,” http://ziyang.eecs.umich.edu/~dickrp/tgff/.