FPGAのDSPブロック向け汎用的テクノロジマッパの網羅的探索による実現可能性検討

柴田 晃陽[1,a)]　今川 隆司[2,b)]　越智 裕之[1,2,c)]

概要：算術演算を多く含むデジタル信号処理回路を FPGA に効率良く実装するためには，DSP ブロックと呼ばれるハードウェアブロックの有効活用が重要であり，ターゲットデバイスの DSP ブロックを用いた最適なテクノロジマッピングを探索できるツールが必要である．また，FPGA アーキテクチャの研究開発のためには，様々な DSP 構造に柔軟に対応できるマッピングツールが強く望まれる．既存のマッピング手法は特定の FPGA デバイスのみを対象としており，また発見的アルゴリズムであるため，効率の良いマッピングができない場合がある．本稿では，与えられた DSP ブロック構成から自動抽出された有効なコンフィギュレーションのデータベースと，アプリケーション回路の記述から生成された data flow grpah (DFG) を入力として，最適なマッピングを網羅的に探索するアルゴリズムを提案する．このとき，必要に応じて演算子を複製することにより，従来のアルゴリズムで得られた最適解よりも少ない数の DSP ブロックでのマッピングを可能とし，同時にネット数やファンアウト数も削減する．また最適性を失わない範囲での枝刈りを行うことで探索時間を短縮する．ノード数が 33 個，58 個，100 個の DFG を用いて従来手法と提案手法のマッピング結果を比較したところ，提案手法は，探索時間は増加するものの，DSP ブロック数を 7.94%から 10.81%削減した．

キーワード：データパス合成，設計空間探索，データフローグラフ，最適カバリング，適用可能構造列挙

A Feasibility Study on Realizing General-purpose Technology Mapper for DSPs of FPGAs Using Exhaustive Search

Koyo Shibata[1,a)]　Takashi Imagawa[2,b)]　Hiroyuki Ochi[1,2,c)]

Abstract: For highly efficient implementation of arithmetic-intensive digital signal processing applications in modern FPGAs, efficient utilization of DSP blocks is important, and algorithms to find optimal technology mapping for a DSP block of the target device are required. Additionally, flexible mapping algorithms applicable to various kind of DSP-block structures is desired for research and development of FPGA architectures. However, the existing mapping methods are dedicated to a specific DSP-block structure, and they may fail to find optimal mapping due to their heuristic algorithms. In this paper, we propose an optimal mapping algorithm based on an exhaustive search using a database of valid configurations from a structure of a target DSP block and data flow graph (DFG) generated from the application circuits. Replication of the operators allows us to find solutions with a smaller number of DSP blocks than those by the conventional algorithm, while at the same time reduces the number of global nets and fan-outs. To reduce runtime, we also introduce pruning techniques that do not affect the optimality. Comparisons of the mapping results between the proposed and conventional methods for DFGs with 33, 58, and 100 nodes show that the proposed method reduces the number of DSP blocks by 7.94-10.81%, at the cost of increased runtime.

Keywords: data-path synthesis, design space exploration, data flow graph, optimal covering, appliable structure enumeration

1. はじめに

2000 年頃から，FPGA の主なターゲットは，積和演算のような算術演算が頻出するデジタル信号処理となった．LUT ベースでの算術演算の実現には，多数の論理ブロックの相互接続が必要となり，遅延や面積，電力の面で ASIC に大きく劣る．これに対し，内部にハードマクロ乗算器等を含む DPS ブロックを備えた FPGA が多く見られるようになり，高速化が図られてきた [1].

DSP ブロックの有効活用のためには，ターゲットとする FPGA がもつ DSP ブロックを考慮したうえで，アプリケーション回路の中から DSP ブロックへのマッピングに適した構造を見つけ，性能を最適化するマッピングを探索するテクノロジマッピングアルゴリズムが不可欠である．文献 [2] は，Xilinx 社の DSP48E1 をターゲットとし，スループットを最大化するマッピングを探索するグリーディーなアルゴリズムを提案している．

本論文で提案するテクノロジマッピング手法では，算術演算子の入出力関係を表すデータフローグラフ (DFG) を入力として，頂点である演算子の複製を考慮しながら，使用する DSP ブロック数やブロック間の相互接続を最小化するマッピングを網羅的に探索する．これは，LUT 向けのテクノロジマッピングにおいて行われる，複数のファンアウトをもつ論理ゲートの複製による LUT の総数の削減を参考にしている．また提案手法はマッピングの探索に先立って，ターゲットの DSP ブロックの構造が記述されたファイルから DSP ブロックが取りうる有効なコンフィギュレーションを列挙してデータベース化する．探索の際にこのデータベースを参照することで，特定の DSP ブロックに特化しない汎用的なアルゴリズムとなっている．

以下，第 2 章で準備として DSP ブロック，および既存のマッピング手法について述べる．第 3 章で提案するマッピング手法について述べ，第 4 章で既存手法と提案手法を用いてアプリケーション回路をマッピングして評価した結果を示す．最後に第 5 章でまとめと今後の課題について述べる．

2. 準備

2.1 DSP ブロック

DSP ブロックは高速性や電力効率が求められる算術演算器のための専用回路を主要な構成要素とし，複雑で多様な演算にも対応できるよう，一定程度のプログラマビリティも持たせた FPGA 内のハードウェアブロックである [1]. 信号処理の分野では積和演算が頻出するため，乗算器と加減算器を組み合わせ，積和演算を 1 つのブロックで実現できるようにした DSP ブロックが多く見られる．しかし，FPGA ベンダごとの設計思想の違いや，ターゲットとするアプリケーションドメインの違いなどによって，様々な演算に対応できるよう，FPGA 毎に DSP ブロックの構成は異なる．図 1 は Xilinx 社の FPGA に搭載されている DSP ブロックである DSP48E1 のブロックダイアグラムである．前置加減算器や乗算器，アキュムレータの他に複数のレジスタを備えており，DSP 単体での積和演算の実現が可能になっている．なお本稿で扱う探索アルゴリズムでは，DSP 内部のレジスタは考慮しないものとする．

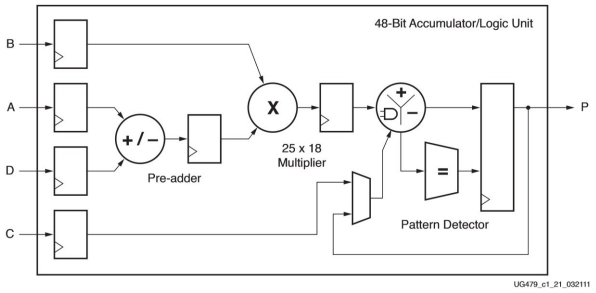


図 1 Xilinx 社の DSP48E1 の構成 [3]

2.2 既存の DSP 向けテクノロジマッピングアルゴリズム

Ronak らが提案した手法 [2] では，アプリケーション回路を計算式のリストで表現し，これを入力として後述するアルゴリズムによりテクノロジマッピングを行い，その結果を Verilog-HDL の形式で生成する．

まずマッピングの準備としてテンプレートデータベースを作成する．文献 [2] でターゲットとしている DSP48E1 は図 2 (A) のように 3 個の演算器が直列接続されたものとしてモデル化される．これに対する有効なコンフィギュレーションは，有効にする演算器の組合せから 7 通りが考えられるので，これらを手作業でテンプレートデータベースに格納する．このデータベースに基づいてマッピングが行われるが，その際のアルゴリズムには Greedy Segmentation と Improved Segmentation の 2 種類が提案されている．

2.2.1 Greedy Segmentation

図 3 に示す Greedy Segmentation のフローは以下の通りである．

(1) DFG 中の 1 つのノードをランダムに選択する．
(2) ノードに単一の子があればそれらを結合しサブグラフ化する．
(3) サブグラフにも単一の子がある場合は，改めてその子も含めてサブグラフ化する．

[1] 立命館大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Ritsumeikan University
[2] 立命館大学情報理工学部
〒 525-8577 滋賀県草津市野路東 1-1-1
College of Information Science and Engineering, Ritsumeikan University
Nojihigashi 1-1-1, Kusatsu, Shiga, 525-8577 Japan
a) is0358ir@ed.ritsumei.ac.jp
b) takac-i@fc.ritsumei.ac.jp
c) ochi@cs.ritsumei.ac.jp

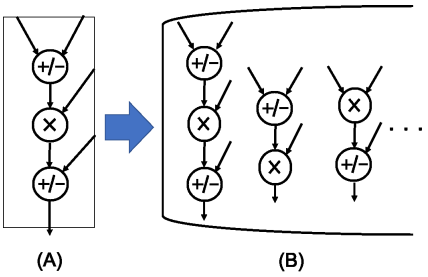


図 2 DSP ブロックの有効なコンフィギュレーションのデータベース作成

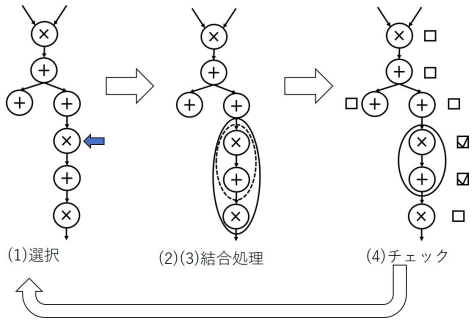


図 3 Greedy Segmentation

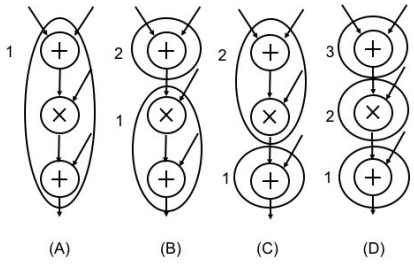


図 4 ランダム選択によるマッピングの違い

(4) サブグラフが抽出される度にテンプレートデータベースと照合し，一致した場合には，サブグラフ内のノードをチェック済みとする．

(5) 全ノードがチェック済みになるまで (1) から (4) を繰り返す．

単一の子のみと結合を行う理由は，例えば DSP48E1 において，3 つの演算器の全てを有効にする場合，2 段目の乗算器の出力を DSP ブロックの外部に出力できず，他の DSP ブロックの入力にできないためである．

このアルゴリズムの計算量は DFG のノード数に比例する一方で，マッピング結果は (1) におけるノード選択の順序に強く依存する．図 4 に，ノードの選択順によって DSP ブロック使用数が変化する例を示す．それぞれのノードに添えられた数字は，ノードを選択した順番である．(A) は最初に 1 番上のノードが選択された場合であり，このときは 3 つのノードが 1 つの DSP ブロックにマッピングされる．一方 (B)〜(D) の順序で選択された場合には，必要な DSP の数が 2 個や 3 個に増加する．

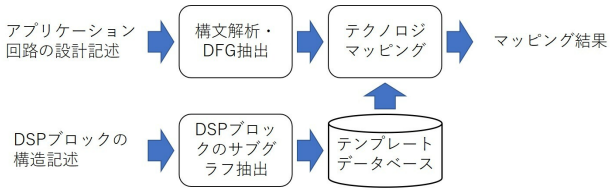


図 5 提案手法のマッピング全体のフロー

2.2.2 Improved Segmentation

Improved Segmentation では，以下のようにテンプレートデータベースの内容を変化させながら Greedy Segmentation を繰り返す．

(1) DSP 内の 3 つの演算器を全て有効にする場合のみをテンプレートデータベースに格納して Greedy Segmentation を行う．

(2) DSP 内の 2 つの演算器を有効にする場合のみをテンプレートデータベースに格納して Greedy Segmentation を行う．

(3) 残ったノードを個別にマッピングする．

このように処理を繰り返すため，Improved Segmentation は Greedy Segmentation と比べて処理量が多くなる一方で，1 つの DSP でより多くのノードをカバーすることを優先するため，DSP の数を削減できる．

2.3 既存手法に対する考察

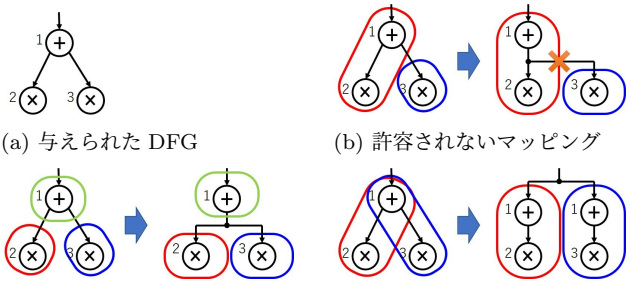
LUT 向けのテクノロジマッピングにおいては，複数のファンアウトをもつ論理ゲートの複製により LUT の総数が削減できることが示されており，広く一般的に用いられている．このことから，DSP ブロックに対するテクノロジマッピングにおいても，同様に演算ノードを複製することで，DSP の総数の削減が可能であると考えられる．

また先行研究の問題点として，Xilinx 社の DSP48E1 に特化した手法で汎用性がないため，異なる構成の DSP ブロックをもつ FPGA には適用できないことが挙げられる．

3. 提案手法

3.1 全体の流れ

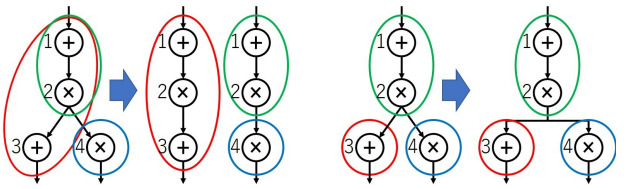
提案するマッピング手法全体のフローを図 5 に示す．入力として，ターゲットアーキテクチャの DSP ブロックの構造記述と，マッピングしたい設計の記述を与える．前者を用いて DSP ブロックのサブグラフが自動的に抽出され，テンプレートデータベースに格納される．ここで DSP ブロックのサブグラフは，1 個の DSP ブロックを構成する全部または一部の演算器からなるグラフであり，コンフィギュレーションによって 1 個の DSP ブロックで実現可能な演算機能に対応する．一方，後者が与えられると構文解析が行われ，DFG が生成される．この DFG とテンプレートデータベースを参照しながら，網羅的に最適なマッピングを探索するテクノロジマッピングを行い，解を出力する．



(a) 与えられた DFG　(b) 許容されないマッピング

(c) ノードを複製しないマッピング (DSP 3 個，総演算器 3 個)　(d) ノード複製を許容するマッピング (DSP 2 個, 総演算器 4 個)

図 6　ノード複製



(a) 冗長なノード複製を行った場合 (DSP 3 個, 総演算器 6 個)　(b) ノード複製を行わなかった場合 (DSP 3 個, 総演算器 4 個)

図 7　冗長なノード複製

以下，提案手法の重要な構成要素であるテクノロジマッピングのアルゴリズムについて述べる．

3.2　ノード複製

提案するマッピングアルゴリズムは，中間出力が取り出せない 1 種類の DSP ブロックが搭載された FPGA アーキテクチャをターゲットとし，DSP ブロックだけからなるネットリストにマッピングすることを想定している．例えば図 6(a) のような DFG が与えられた場合，図 6(b) のようなマッピングを行うことは許容されない．従来手法では図 6(c) のようにファンアウトが 2 以上のノードが必ず DSP ブロックの出力となるマッピングを行っており，3 個の DSP が必要であった．提案手法ではノード複製を行って DSP ブロック使用数を減らすことで，効率的なマッピングを行うことを考える．提案手法では図 6(d) のようにノード 1 を複製することで，2 個の DSP でのマッピングが可能となる．ただし，ノード複製を行うので総演算器数は増加する．

ノード複製を行わない方がよい場合もあるので留意しなければならない．図 7 の (a)，(b) の DSP 数はともに 3 個であるが，図 7(a) は総演算器数が (b) より 2 個多く必要であり，冗長な複製が行われていると言える．

3.3　マッピングアルゴリズム

提案するテクノロジマッピングアルゴリズムは，DFG を始端ノードから深さ優先で辿り，各ノードをカバーする DSP に対応するサブグラフを順次選択する．あるノードをカバーするサブグラフの候補は多くの場合，複数あるので，深さ優先で網羅的に解空間を探索し，最適解を出力する．提案するアルゴリズムの概要を図 8 に示す．

```
01 // TDB : テンプレートデータベース
02 // DFG : データフローグラフ
03 mapper_main() {
04     T = DFG の全ての始端ノード;
05     dfs(T, φ) を呼び，返り値の中で最適なものを出力;
06 }
07 dfs(T, M) {
08     if (T == φ) return φ;
09     t = pop(T);
10     S = t を含むサブグラフの集合 ∩TDB;
11     for (i=0; i < |S|; i++) {
12         M_new = S_i がマッピングするノード;
13         T_new = (M_new ∪ M) に含まれない M_new の子ノード;
14         R_i = dfs(T_new ∪ T, M_new ∪ M)×{S_i};
15     }
16     return ∪_{i=0}^{|S|} Ri;
17 }
```

図 8　テクノロジマッピングフロー

まずメインルーチン *mapper_main*() でマッピングの起点となるターゲットノードのリスト (以下，ターゲットリスト) T を作成し，探索を行う関数 *dfs*() を呼ぶ．*dfs*() の第 1 引数 T の初期値は DFG の全ての始端ノードであり，第 2 引数 (マッピング済みのノードのリスト) の初期値は空集合 ϕ である．再帰的関数 *dfs*() では，ターゲットリスト T から先頭ノード t を取り出し，そのターゲットノード t を含むサブグラフを抽出する．このとき，DSP ブロックのノード数をサブグラフの最大ノード数とし，有効なサブグラフすべてを抽出し，その中でテンプレートデータベース (TDB) に構造が合致するものが無いものを除いて，t をカバーするサブグラフの候補 S とする．このサブグラフの数 $|S|$ が探索の分岐の数となる．

なお，実際には重複した探索を避けるため，t を含むサブグラフの抽出に制限を設けている．ターゲットノード t の下流ノードに複数のファンアウトがある場合，ファンアウト先ノード間に優先順位 (左側優先等) を設け，いずれか 1 つのファンアウト先以外をカバーするサブグラフは除外する (ここで除外されたファンアウト先は後述するようにターゲットリストに加えられる)．例えば，ターゲットノードがノード A でノード 3 つのサブグラフを図 9(1) の DFG から抽出しようとしたとき，サブグラフは ABC と ABD の 2 つが存在するが，このうち一方だけが S に加えられる．またターゲットノード t の上流ノードに関して以下の 3 つに場合分けし，異なった方法で S を抽出する．

(1) t の親ノードがマッピング済みで中間出力が得られない場合
(2) t の親ノードが複数の場合
(3) t の親ノードが単一の場合

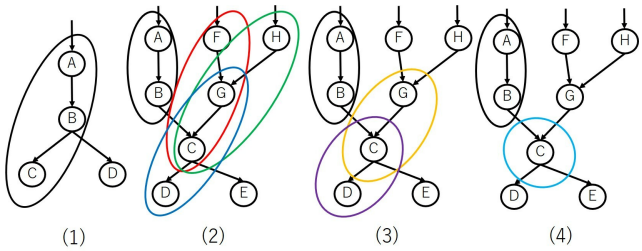


図 9 サブグラフ抽出の例

(1) のケースは，例えば図 9(1) でターゲットノードがノード D である場合が該当する．このように D の親ノード B がマッピング済みで出力が得られない場合，ノード複製を行い B の中間出力を得る必要がある．そこで，グローバル配線から全入力が得られる図 9(1) のノード A を先頭ノードとし，ノード A からのサブグラフを抽出する．A の下流に複数出力先がある場合，ターゲットノードとしていたノード D を含むようなサブグラフ抽出を行う．サブグラフのノード数によっては，ターゲットノードがマッピングされない場合がある．(2) のケースは，図 9(2)，(3)，(4) のノード C がターゲットノードである場合のように，ターゲットノードの親ノードが複数ある場合を指す．このような場合，ターゲットノードを含む全パタンのサブグラフを抽出する．ただし，親ノードがマッピング済みで入力が得られないサブグラフは破棄する．この例では，サブグラフのノード数を 3 つとした場合，抽出されるサブラフは図 9(2) のような 3 つ，ノード数を 2 つとした場合，図 9(3) のような 2 つ，ノード数を 1 つとした場合，図 9(4) のような 1 つサブグラフが抽出される．(3) のケースでは，マッピング済みのノードを含まないサブグラフを抽出する．

こうして得られた S のサブグラフ数が探索木の分岐の数となるが，後述する枝刈りの効果を高めるため，実際にはノード数が大きいサブグラフから優先的にマッピングする．S からサブグラフ S_i が選択された分岐では，再帰的に *dfs*() を呼ぶにあたり，元の T から t を取り除き，代わりに，S_i でマッピングされるノードの子ノードで，まだマッピングされていないノードを加えたものを新たなターゲットノードとする．第 2 引数には，元の M に加え，S_i によってカバーされたノードを加えたものをマッピング済みノードリストとして与える．

深さ方向の探索の完了判定では，すべてのノードがマッピングされているかの判定をターゲットリストにノードが存在するかどうかで行い，バックトラックを行う．メインルーチンに戻ったとき，網羅的探索探索を終えたことになる．

3.4 枝刈り

網羅的探索によるマッピング結果の数，所要時間は膨大となるため，全分岐の網羅的探索は現実的ではない．そこで，最適性を損なわない枝刈りを行うことが必要となる．提案するアルゴリズムでは次の枝刈りを導入する．

(1) 残りのノードをマッピングをするのに必要な DSP 最小数での枝刈り
(2) 冗長なノード複製の枝刈り
(3) 非効率なマッピングの枝刈り
(4) 単一入力かつ，単一出力または終端ノードの評価

以下，それぞれの枝刈りの説明を行う．ここでの枝刈りは，*dfs*() の再帰呼び出し (図 8 の 14 行目) を行わないことを意味する．

3.4.1 必要な DSP 最小数での枝刈り

この枝刈りでは，マッピングされていない残りノードをマッピングするのに必要な DSP ブロック最小数を算出し，その数とすでにマッピングされた DSP ブロック数の和が，そのときの最適解である DSP ブロック使用数より上回ったとき，枝刈りを行う．

提案手法では，単純に DFG の残りの総ノード数を DSP ブロックの総ノード数で割るのではなく，ノードの演算子を考慮して算出する．演算子の種類ごとに，DFG の残りのノード数を DSP ブロックのノード数で割る．そのときの最大値が DSP ブロックの必要最小数となる．この値とすでにマッピングされた DSP ブロック数の和をとることで，マッピングの完了に必要な DSP ブロック最小数を求め，これに基づいて枝刈りを行う．

3.4.2 冗長なノード複製の枝刈り

この枝刈りでは本章で挙げた冗長なノードの複製がされた探索木の枝刈りを行う．本章で述べたサブグラフ抽出方法 (1) を行う際に，ターゲットノードを含むサブグラフが抽出出来なかったとする．その場合，そのサブグラフは必ず全てのノードがマッピング済みのノードになる．具体例として，図 10(1) を考える．複数出力先を持つノード A は出力先ノード B とともにマッピング可能だが，出力先ノード C とマッピング出来ない場合，中間出力を得るためにノード A を複製する必要がある．そこで，複数出力先を持つノード A の出力が得られないサブグラフの枝刈りを行う．

別の具体例として，図 10(2) を考える．ノード C のように出力先のノード群にマッピング済みのノードが含まれる場合，ノード C の出力は必須となる．ノード C をマッピングされていないノード D とともにマッピングを行う場合，ノード C を複製して個別でマッピングすることで中間出力を得る必要がある．そこで，ノード C の出力が得られないサブグラフの枝刈りを行う．これらの枝刈りにより，探索木の浅い階層での枝刈りが可能なため，効率的な探索が期待できる．

3.4.3 非効率なマッピングの枝刈り

この枝刈りでは，効率的なマッピングを探索し，枝刈りする．例として，図 11(1) のような DFG をマッピングす

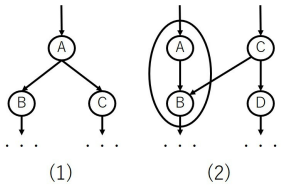


図 10 冗長なノード複製の枝刈り

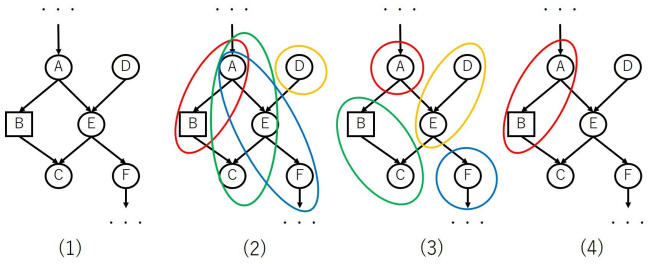


図 11 非効率なマッピングの枝刈り

ることを考える．B は単一出力の 1 つ以上のノード群で，ノード C は終端ノード，ノード D は始端ノードである．この DFG は図 11(2) のようにノード複製を行い，DSP ブロック 4 個でマッピングできる．しかし，図 11(3) のようにノード複製を行わなくても，同じく DSP ブロック 4 個で効率的にマッピングできる．そこで，(3) のマッピングを見つけ，(2) のマッピングを枝刈りする．

図 11(4) のようにマッピングされている状態で，ターゲットノードがノード C でサブグラフを抽出することを考える．サブグラフの抽出方法 (2) でサブグラフ AEC が抽出される．非効率なマッピングを避けるためにこのサブグラフを破棄したい．条件は次の 3 つである．

- サブグラフに図 11(4) のノード E のような，一方はマッピング済み，もう一方はマッピングされていない複数入力を持つノードが含まれているとき
- マッピングがされていない始端ノード D がノード E とともにマッピング可能なとき
- ノード群 B と終端ノード C がともにマッピング可能なとき

これらの条件をすべて満たすとき，図 11(3) のような効率的なマッピングが見つかることが保証されるため，上記の条件を満たすサブグラフのマッピングの枝刈りをする．

3.4.4 単一入力かつ，単一出力または終端ノードの評価

この枝刈りではサブグラフの抽出を行った際，サブグラフに単一入力かつ，単一出力または終端ノードが含まれていた場合，そのときのグラフの評価をし枝刈りを行う．

単一入力かつ，単一出力または終端ノードにはそれぞれグラフの評価値として，そのノードをマッピングするのに必要な最小の DSP ブロック使用数を保持する．例として，図 12 の (1) を考える．図 12 の (2) のようなマッピングがされたとき，ノード B，C ともに，マッピングするために

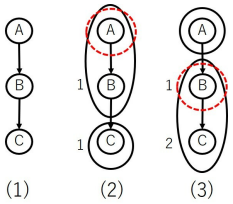


図 12 グラフの評価値

必要な DSP ブロック使用数は 1 個なため，評価値は 1 となる．図 12 の (3) のようなマッピングがされたとき，ノード B をマッピングするために必要な DSP ブロック使用数は 1 個なため，評価値は 1 となる．ノード C をマッピングするために必要な DSP ブロック使用数は，ノード B の出力が必要なため 2 個となり，評価値 2 となる．

この評価値を用いて枝刈りを行う．ターゲットノードが単一入力かつ，単一出力または終端ノードのとき，そのときの DSP ブロック使用数とそのノードの評価値を比較し，評価値より上回った場合，枝刈りを行う．評価値がそのときの DSP ブロック使用数を下回ったとき，評価値を更新する．

4. 提案マッピング手法の性能評価

本章では，2 章で述べた 2 つの既存手法と提案手法によるマッピング結果を，DSP ブロック使用数，ネット数，ファンアウト数，総演算器数，所要時間の観点から比較する．ここでのネット数は DSP 間をつなぐデータパスの本数であり，FPGA 上で使用されるグローバル配線の数と関連する．既存手法の Greedy Segmentation と Improved Segmentation は，ノードの選択順によって結果が変わる発見的探索であるため，選択順をランダムに変えながら 10 万回の試行を繰り返し，DSP ブロック数が最小となる場合の結果と，1 回の試行に要する平均所要時間を取得した．評価には，演算ノード数がそれぞれ 33，58，100 個の 3 つの DFG を用いた．各ノードにおける演算は加算，減算，乗算の中からランダムに設定される．また各ノードは 1 つまたは 2 つの入力をもち，他のノードまたは外部入出力と接続される．ターゲットとする DSP ブロックは，先行研究でターゲットとしていた Xilinx 社の DSP48E1 と同じ構造とした．既存手法と提案手法のアルゴリズムはいずれも Python (ver.3.6.9) と PyPy (ver.7.3.1) を用いて実装および動作させ，構文解析には PLY (ver.3.11) を，グラフデータの管理と処理には Networkx (ver.2.2) をそれぞれ用いた．

既存手法と提案手法でマッピングを行った場合の DSP ブロックの最小数と，そのときのネット数，ファンアウト数，総演算器数を表 1，表 2，表 3 に示す．これらの評価から，3 つの DFG の全てに対して提案手法の使用 DSP ブロック数が最小であり，Improved Segmentation と比較

表 1 ノード 33 個での既存手法と提案手法の結果

マッピング手法	最小 DSP ブロック使用数	ネット	ファンアウト	総演算器
Greedy	22	23	31	33
Improved	22	23	31	33
提案手法	20	21	30	36

表 2 ノード 58 個での既存手法と提案手法の結果

マッピング手法	最小 DSP ブロック使用数	ネット	ファンアウト	総演算器
Greedy	37	39	52	58
Improved	37	39	52	58
提案手法	33	35	49	63

表 3 ノード 100 個での既存手法と提案手法の結果

マッピング手法	最小 DSP ブロック使用数	ネット	ファンアウト	総演算器
Greedy	63	70	90	100
Improved	61	68	88	100
提案手法	56	63	84	107

した際の削減率は 7.94%から 10.81%であることが分かる．従って提案手法は，既存手法では発見できなかった，DSP ブロック使用数のより少ないマッピングを発見できているといえる．また総演算器数が増加している一方で，ネット数とファンアウト数は減少していることから，FPGA 上の配線資源の利用効率の改善が見込め，提案マッピング手法によるアプリケーション回路の性能向上が期待できる．

所要時間に関しては，表 4 に既存手法における 1 回あたりの試行にかかる所要時間の平均値を，表 5 に提案手法の所要時間をまとめる．表 5 における – は，処理時間が 86400 秒を超えたために処理を途中で打ち切り，マッピングが完了しなかったことを意味する．また 3.4 節で提案したそれぞれの枝刈り手法の，所要時間削減に対する寄与を評価するため，適用する枝刈り手法を変化させた場合の結果も示している．ここでの枝刈り 1 から枝刈り 4 は，3.4.1 節から 3.4.4 節で述べた手法とそれぞれ対応する．表 5 の結果から，3.4.1 節および 3.4.4 節で述べた枝刈り手法が，所要時間の削減に大きく寄与していることが示されている．

表 4 と表 5 を見比べると，提案手法は既存手法と比べて所要時間が大幅に増加しているように見える．しかし，既存手法は発見的探索であるために，高品質なマッピングを得るためには複数回の試行を必要とすることと，提案手法は枝刈りの適用により所要時間が減少したことを考慮すれば，提案手法の所要時間は既存手法に比べ、今回実験した DFG のノード数の範囲では遜色ないと考えられる．

表 4 既存手法での 1 回あたりの平均所要時間 [s]

マッピング手法	ノード 33	ノード 58	ノード 100
Greedy	0.000204	0.000359	0.000680
Improved	0.000205	0.000351	0.000647

表 5 提案手法での所要時間 [s]

マッピング手法	ノード 33	ノード 58	ノード 100
枝切りなし	59356.47	–	–
枝刈り 1	63.26	–	–
枝刈り 1，2	7.96	156.52	–
枝刈り 1，2，3	7.61	81.14	–
枝刈り 1，2，3，4	1.75	6.64	23.89

5. おわりに

本論文では，論理ブロックとは別に，DSP ブロックを搭載した FPGA において，いかなる構成の DSP ブロックにも対応できる汎用的なテクノロジマッピングを行う手法と網羅的なマッピング手法を提案した．その結果，複雑な DSP ブロックにおいて有効なコンフィギュレーションを検出でき，DSP ブロックのサブグラフを自動で抽出することが可能となった．網羅的なマッピング手法では，ノード複製による DSP ブロック使用数の効率化を行った．それにより，複数の出力をもつ演算ノードがある場合でも，効率的なマッピングが可能になった．さらに，探索の枝刈りを行うことで所要時間の短縮を図った．提案手法のマッピング手法を用いることで，既存手法と比べ総演算器数，所要時間は増加するものの，最良の結果を得ることができ，DSP ブロック数の削減率は 7.94%から 10.81%となった．

今後の課題としては，DSP ブロックを構成する際，グラフ構造が与えられた場合のみでなく，機能仕様のみの場合でも有効なコンフィギュレーションを自動で検出するアルゴリズムや，面積, 電力, 遅延を評価指標とし，これらを最適化するアルゴリズムの構築が挙げられる．さらには，アプリケーションが必要とする演算粒度が単一の DSP ブロックを超えたとき，複数の DSP ブロックを組み合わせて演算を実現するアルゴリズムへの改良も考えられる．

参考文献

[1] 天野英晴：FPGA の原理と構成，pp. 86–91，オーム社 (2016).
[2] Ronak, B. and Fahmy, S. A.: Mapping for Maximum Performance on FPGA DSP Blocks, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, Vol. 35, No. 4, pp. 573–585 (2016).
[3] Xilinx Corporation: *7 Series DSP48E1 Slice User Guide*, `https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf` (2011).