Vol.2015-SLDM-170 No.49
Vol.2015-EMB-36 No.49
2015/3/7

再構成可能配線構造検討のための性能評価環境の構築

森岡 俊樹[1,2]　山本 浩平[1,2]　密山 幸男[1,2,a)]

概要：再構成可能デバイスはその構造的特徴から配線部が面積効率や動作速度に与える影響が大きくなるため、再構成可能アーキテクチャ開発において、配線構造を慎重に検討することが必要である。粗粒度再構成可能アレイは、FPGA 等の細粒度型と比較して高い面積効率と電力効率が期待できることが知られている。しかし、面積効率を追及するあまりに配線リソースを削減し過ぎると、マッピング効率が低下して必要なアレイサイズが大きくなり、結果的に面積効率の低下につながる。さらに、ルーティングが複雑になることで、経由スイッチ数に依る配線遅延が大幅に増大する。そこで本稿では、再構成可能配線構造を検討するうえで、高い精度で配線遅延や面積を評価することができる環境を構築した。本評価環境を用いることにより、配線構造のタイプと配線リソースをパラメータ化して定義することで、自動配置配線処理によるアプリケーションマッピング結果から、配線構造の有効性を定量的に評価することが可能である。

キーワード：再構成可能デバイス，配線構造，評価環境，自動配置配線，アーキテクチャ探索

1. はじめに

FPGA に代表される細粒度再構成可能アレイが LUT (Look-Up Table) ベースの基本要素 (Logic Element: LE) で構成されることに対して、粗粒度再構成可能アレイ (Coarse Grained Reconfigurable Array: CGRA) は演算器ベースの基本要素 (Processing Element: PE) によって構成される。CGRA は、ターゲットアプリケーションがアーキテクチャに適合する場合は高い面積効率や動作速度を実現することが可能であるが、再構成可能デバイスの構成的特性上、配線部が面積効率や動作速度に与える影響は大きい [1][2]。

配線リソースが少ない場合、配線部のスイッチ数が減少する。そのため、FIR フィルタのような配線が簡易なアプリケーションを実装する場合には、高い面積効率と動作速度を実現することができる。しかし、FFT のように演算器間の配線接続が複雑なアプリケーションを実装する場合には、配線リソースが乏しいと、必要なアレイサイズが大きくなり、結果的に面積効率が大幅に低下する。また、大きく迂回するパスが発生することにより、動作速度が著しく低下することが予測される。

一方、配線リソースが多い場合、配線部のスイッチ数は増加する。しかし、配線の複雑なアプリケーションをマッピングするときには配線リソースが少ない場合に比べて小さいアレイサイズでのマッピングが可能であるため、結果的に面積効率が向上する。また、経由スイッチ数が減ることによって動作速度の向上が見込める。一方、配線の簡易なアプリケーションをマッピングする場合には、配線スイッチの面積・遅延オーバヘッドの大きさが影響し、配線リソースが少ない場合に比べて面積効率と動作速度ともに低下することになる。

また、配線構造が異なると同程度の配線リソースであっても、面積効率や動作速度が大きく異なる。また、再構成可能アレイの特徴として、アプリケーションマッピング（配置配線）の結果によっても面積や遅延が大きく変わる。そのため、配置配線ツールを用いた実際のアプリケーションマッピング結果から配線構造を定量的に評価することが重要である。

本研究では、配線構造と配線リソースをパラメータ化して定義することで、配線遅延や面積の高精度な評価を行うことができる環境を構築した。本環境により、アプリケーションマッピング結果から、配線構造の性能を高精度に評価することが可能となる。

2. 配線構造の定義

2.1 直接接続型配線構造

直接接続型配線構造は、MorphoSys[3] や MATRIX[4] 等で採用されており、長さの異なる複数種類の直接配線により PE 同士が接続される配線構造である。図 1 に直接接続

[1] 高知工科大学
Kochi University of Technology
[2] 独立行政法人科学技術振興機構, CREST
CREST, JST
[a)] mitsuyama.yukio@kochi-tech.ac.jp

Vol.2015-SLDM-170 No.49
Vol.2015-EMB-36 No.49
2015/3/7

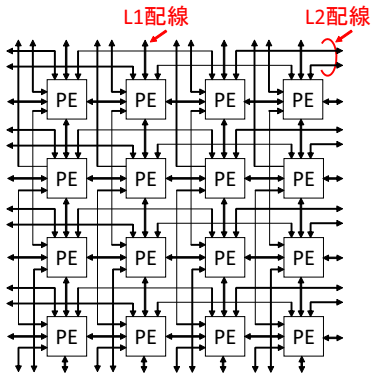


図 1　直接接続型配線構造

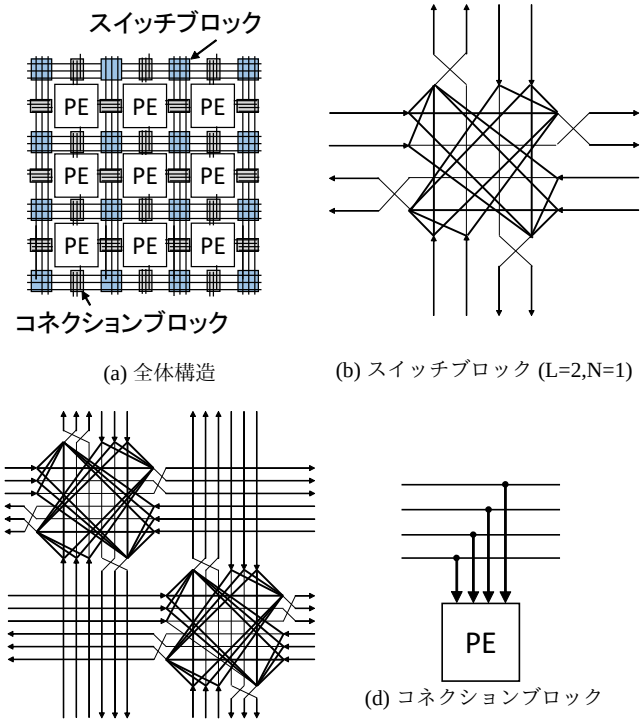
型配線構造を示す。図 1 では直接配線を双方向で表しているが、実際には片方向の直接配線がペアになっている。図 1 に示す配線構造は、1 つ隣の PE 同士を接続する配線長 1 の L1 配線を 1 セットと、2 つ隣の PE 同士を接続する配線長 2 の L2 配線を 1 セットもつ直接接続型配線構造である。直接接続型配線構造は配線パラメータとして、配線長とその本数を定義する。図 1 に示す直接接続型配線構造のパラメータを (L1)x1,(L2)x1 として表す。

直接接続型配線構造では、PE は演算リソースとして使用しつつ、ルーティングパスの経由を 1 本まで許す。直接接続型配線構造は、他の配線構造に比べて配線自由度は低下するがスイッチ数が少なくなる。アプリケーションをマッピングした結果、多数の直接配線を経由するパスが存在しない場合には、高い面積効率と動作速度を実現する。一方、複数の直接配線を通るパスが多くなるようなアプリケーションマッピングにおいて、演算リソースとして使用できない PE が多数発生するため、面積効率と動作速度ともに著しく低下する。

2.2　バス型配線構造

バス型配線構造は、DRP[5] や HSRA[6]、RaPiD[7] 等で採用されており、PE 同士がコネクションブロックとスイッチブロックを介して接続される配線構造である。本研究では、[8] で提案されている片方向のバス型配線構造に適した構造を用いる。片方向バス型配線構造を図 2 に示す。スイッチブロック中の太線は、スイッチを 1 個もつ配線を表す。L=2,N=1 のバス型配線構造は、2 個先の PE までスイッチを介さず接続可能な配線長 2 のバス配線が 1 系統あることを表し、L=3,N=1 のバス型配線構造は、3 個先の PE までスイッチを介さず接続可能な配線長 3 のバス配線が 2 系統あることを表す。バス型配線構造は配線パラメータとして、バス長 L とそのセット数 N を定義する。

バス型配線構造は、直接接続型に比べて配線部の面積が大きくなるが、配線自由度が高いことが特長である。このため、バス型配線構造は、直接接続型配線構造では複数の直接配線を通るパスが多くなるアプリケーションマッピングにおいて、直接接続型配線構造より高い面積効率と動作速度を実現する。他方、複数の直接配線を通るパスが少なくなるようなアプリケーションマッピングにおいて、配線部の面積・遅延オーバヘッドが大きいバス型配線構造は、直接接続型配線構造に比べて面積効率や演算性能が低くなる。



(a) 全体構造　(b) スイッチブロック (L=2,N=1)

(c) スイッチブロック (L=3,N=2)　(d) コネクションブロック

図 2　バス型配線構造

2.3　複合型配線構造

複合型配線構造とは、前述のバス型配線構造において隣接する PE 同士を接続する隣接配線を追加した提案配線構造である。図 3 に複合型配線構造を示す。隣接配線はバスを経由することなく、1 つ隣の PE との接続のみが可能である。複合型配線構造は配線パラメータは、バス型配線構造と同様のバス長 L とそのセット数 N を定義する。複合型配線構造は、直接接続型配線とバス接続型配線の特長を併せ持つ。

3.　評価環境の構築

3.1　評価用ツールフロー

構築した評価環境を図 4 に示す。アプリケーションネットリストは、動作合成により生成されたものを使用する。配線構造の種類と配線パラメータ、アレイサイズから、スクリプトによって配置ツール用にアーキテクチャ定義と遅延定義が自動生成される。配置ツールにより定義したアーキテクチャに対してアプリケーションネットリストを自動

Vol.2015-SLDM-170 No.49
Vol.2015-EMB-36 No.49
2015/3/7

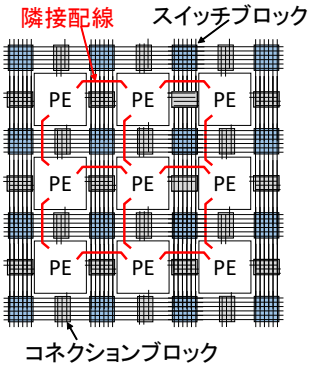


図 3　複合型配線構造

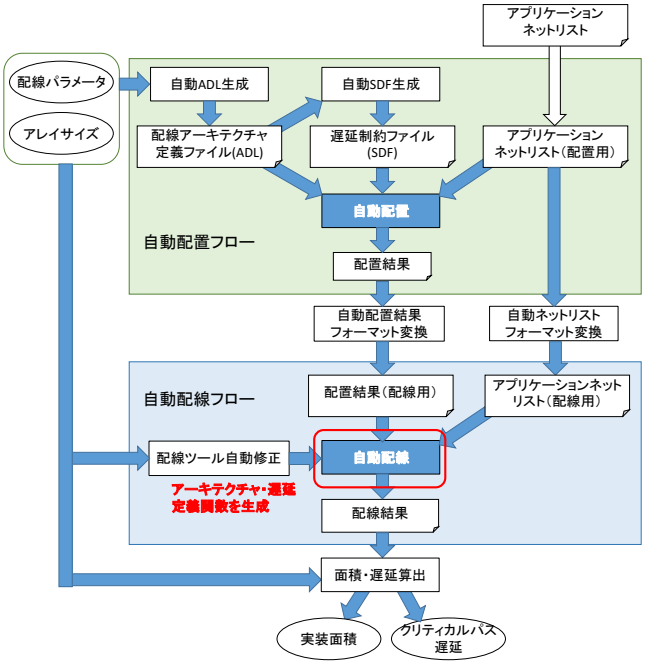


図 4　自動配置配線と遅延評価の詳細フロー

配置する。アーキテクチャのパラメータから配線ツールのアーキテクチャ・遅延定義関数を自動生成することで、配線ツール用のアーキテクチャと遅延を定義する。また、スクリプトにより配置ツール用から配線ツール用にアプリケーションネットリストを自動変換する。アーキテクチャ・遅延定義とアプリケーションネットリスト、配置結果から配線ツールによる自動配線を行う。自動配置配線により得られた結果から実装面積やクリティカルパス遅延を算出することで、配線構造を定量的に評価することのできる環境を構築した。

本研究では、自動配置配線は立命館大学開発の自動配置配線ツールを用いる。

3.2　自動配置ツール

自動配置アルゴリズムは、Simulated Annealing (SA) 法のアルゴリズムを採用しており、アプリケーションマッピング時の総遅延をコストとしている。

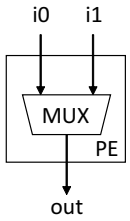


図 5　PE の構成例

```
module TOP_ARRAY( /* 入出力信号 */ );
 PE PE_R00C00( /* PEの入出力信号*/  );   // 0行0列
 PE PE_R00C01( /* PEの入出力信号*/  );   // 0行1列
    :
endmodule

module PE( i0, i1, out );
  input i0, i1;
  output out;
  MUX MUX( .a(i0), .b(i1), .x(out) );
endmodule
```

(a) アーキテクチャ定義

```
(CELLTYPE “MUX”)
  (PORT a 0.1)
  (PORT b 0.1)
  (IOPATH a x 0.1)
  (IOPATH b x 0.1)
  (DEVICE x 1)
```

(b) 遅延定義

図 6　アーキテクチャ・遅延定義のフォーマット（配置）

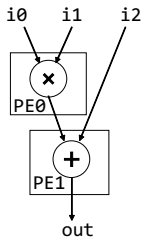


図 7　アプリケーションの例

自動配置ツール用のアーキテクチャ定義ファイルと遅延定義ファイルの PE の記述フォーマットを説明するために、図 5 に PE の構成例を示す。実際は PE には演算器等が含まれるが、図 5 では説明のために、PE の内部はマルチプレクサ (MUX) のみに簡略化している。

図 5 の PE を、配置ツール用のアーキテクチャ定義ファイルおよび遅延定義ファイルに記述した例を図 6 に示す。配置ツール用のアーキテクチャ定義ファイルは、Verilog HDL に準拠したフォーマットで記述され、アーキテクチャを階層的に記述できる。しかし、assign 文や always 文は使用できず、モジュール呼び出しのみによってアーキテクチャを定義する。配置ツールでは座標の概念がないため、座標情報を含んだインスタンス名をスクリプトにより生成する。配置ツールの遅延定義ファイルは、Standard Delay Fromat (SDF) に準拠し、各要素の入出力ポートの遅延と要素内部での遅延を設定可能である。

図 7 に示すアプリケーションのデータフローグラフ (Data Flow Graph: DFG) を定義する配置ツール用のネットリスト記述を 図 8 に示す。配置ツール用アプリケーションネットリストも、アーキテクチャ定義と同様のフォーマットで記述される。

```
module application ( i0, i1, i2, out );
  input i0, i1, i2;
  output out;
  wire net0;
  PE MLT ( .IN0(i0), .IN1(i1), OUT0(net0) );
  PE ADD ( .IN0(i2), .IN1(net0), OUT0(out) );
endmodule
```

図 8 アプリケーションネットリストのフォーマット（配置）

```
void rrginit(){

  ws[0].type = input;              // i0の定義
  ws[0].x= 0;  ws[0].y = 0;
  ws[0].i= 0;  ws[0].w = 1;

  ws[1].type = input;              // i1の定義
  ws[1].x= 0;  ws[1].y = 0;
  ws[1].i= 1;  ws[1].w = 1;

  ws[2].type = output;             // outの定義
  ws[2].x= 0;  ws[2].y = 0;
  ws[2].i= 0;  ws[2].w = 10;

  putsw(input,0,0,0,mux2alu,0,0,0); // i0->out
  putsw(input,0,0,1,mux2alu,0,0,0); // i1->out

}
```

図 9 アーキテクチャ・遅延定義関数（配線）

表 1 配線定義のための構造体変数のメンバ

メンバ	概要
type	配線の種類（入力や出力）
x	PE の X 座標
y	PE の Y 座標
i	PE 内の 配線識別番号
w	遅延コスト

3.3 自動配線ツール

自動配線ツールは negotiation ベースのアルゴリズムで、配置ツールと同様に総遅延をコストとする。自動配線ツールには、配置ツールのようなアーキテクチャ定義ファイルや遅延定義ファイルは存在せず、rrginit 関数によってアーキテクチャ定義と遅延定義を行う。図 5 の PE のアーキテクチャ定義と遅延定義を行う rrginit 関数の記述を図 9 に示す。rrginit 関数は、構造体 ws により配線を定義し、putsw 関数により各配線をスイッチで接続することでアーキテクチャを定義する。配線定義の構造体 ws のメンバを表 1 に示す。putsw 関数には、配線の種類、PE の x 座標、PE の y 座標、PE 内の識別番号を接続元の配線、接続先の配線の順に引数として渡す。

図 7 に示したアプリケーションの配線ツール用のネットリストの記述を図 10 に示す。配線識別番号 i は、PE 内に同じ種類の配線が複数存在する場合にそれらを識別するために定義される。

3.4 面積・遅延算出

本稿では、これまでは配線間の接続をスイッチとして説明してきたが、実際には MUX により実現する。MUX は

```
// 配線名 PE(source) fan-out数 PE(target1) PE(target2) ...
   i0 input 1 PE0.i0
   i1 input 1 PE0.i1
   i2 input 1 PE1.i1
   net0 PE0.out 1 PE1.in0
   out PE1.out 1 output
```

図 10 アプリケーションネットリストのフォーマット（配線）

表 2 評価用アプリケーションの演算器数

評価用アプリケーション	演算器数
7 タップ FIR フィルタ	15
8 点 FFT	36
DCT[11]	46

表 3 配線パラメータとチップ面積

配線構造	パラメータ	チップ面積 [mm2]
直接接続型	(L1)x2	0.37
	(L1)x2,(L2)x1,(L4)x1	0.43
バス型	L=2,N=1	0.39
	L=4,N=2	0.72
複合型	L=2,N=1	0.42
	L=4,N=2	0.75

入力数に応じて面積と遅延が決まる。アレイサイズと配線パラメータから、PE 内の MUX 数と各 MUX の面積と遅延が分かる。PE 内の MUX 数と各 MUX の面積からアレイ全体の面積を算出する。さらに、各 MUX の遅延と配置配線結果から得られるクリティカルパスの経由 MUX 数から、クリティカルパス遅延を算出する。

4. 評価実験

4.1 実験条件

本実験で対象とする CGRA のアレイサイズは、DCT のアプリケーションの規模から 8x8 とする。対象アプリケーションとその使用演算リソース数を表 2 に示す。DCT は Chen のアルゴリズム [11] を用いる。

アプリケーションマッピングでは、アレイ上部からデータを入力することを考える。SA の初期温度を 40 度、減少刻み温度を 0.1 度とし、各配線パラメータとアプリケーションに対して配置配線を 5 回行い、その中で最も密に配置できたときの面積効率やクリティカルパス遅延を評価する。

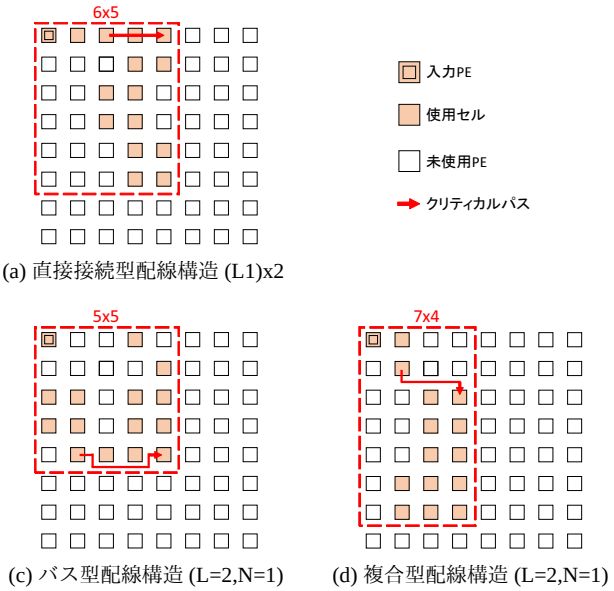
今回評価する配線パラメータと、65nm プロセスで設計したときのチップ面積を表 3 に示す。以上のことから、本研究で構築した評価環境により、各配線構造の面積を定量的に評価することが可能となったと言える。

配線リソースが少ない場合は 3 種類の配線構造でチップ面積に大差はないが、配線リソースが多い場合には直接接続型配線構造のチップ面積は、他の配線構造よりも大幅に小さくなっている。また、複合型配線構造のチップ面積は、隣接配線により MUX の面積が増加した分、同じ配線パラメータのバス型配線構造に比べて大きくなっている。

Vol.2015-SLDM-170 No.49
Vol.2015-EMB-36 No.49
2015/3/7

表 4　実験結果（7 タップ FIR）

配線構造	パラメータ	アレイサイズ	最大経由 MUX 数	最大遅延 [ns]
直接接続型	(L1)x2	6x5(100%)	1	1.76(100%)
	(L1)x2, (L2)x1, (L4)x1	6x7(140%)	0	0.95(54%)
バス型	L=2,N=1	5x5(83%)	3	2.45(139%)
	L=4,N=2	6x5(100%)	1	1.48(84%)
複合型	L=2,N=1	7x4(93%)	1	1.27(72%)
	L=4,N=2	3x8(80%)	2	2.38(135%)



(a) 直接接続型配線構造 (L1)x2
(c) バス型配線構造 (L=2,N=1)　(d) 複合型配線構造 (L=2,N=1)

図 11　FIR マッピング結果

4.2　7 タップ FIR フィルタマッピング結果

7 タップ FIR フィルタマッピング時の実験結果を表 4 に、マッピング結果を図 11 に示す。

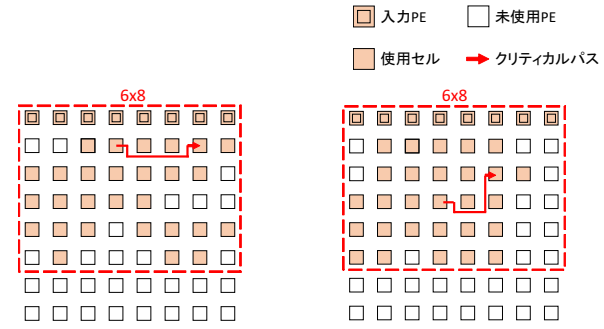
図 11 から、アプリケーションのマッピング領域には未使用 PE が多く存在することが確認できる。これは、配置アルゴリズムは総配線遅延が SA のコストとして定義されており、PE 同士を密に配置することは考えられてないためである。

遅延に関しては、直接接続型配線構造では配線リソースが多い場合のほうが個々の MUX の遅延は大きいが、経由 PE 数が少ないことで結果的にクリティカルパス遅延が小さくなっている。バス型配線構造も直接接続型配線構造と同様に、配線リソースが多いほうがクリティカルパス遅延が小さくなっている。しかし、複合型配線構造では配線リソースが少ない場合のほうがクリティカルパス遅延が小さくなっている。これは、前述の通り配置ツールのコストである総配線遅延を小さくした結果、クリティカルパス遅延が大きくなる配置になったためだと考えられる。配線構造同士で比較すると、直接接続型配線構造が他の配線構造に対して遅延が小さくなる傾向がある。

以上のことから、本環境を用いることによって経由 MUX 数やクリティカルパス遅延を定量的に評価することが可能となったと言える。

表 5　実験結果（8 点 FFT）

配線構造	パラメータ	アレイサイズ	最大経由 MUX 数	最大遅延 [ns]
直接接続型	(L1)x2	8x8(100%)	マッピング不可	
	(L1)x2, (L2)x1, (L4)x1	8x8(100%)		
バス型	L=2,N=1	6x8(75%)	3	2.45(100%)
	L=4,N=2	6x8(75%)	2	2.22(91%)
複合型	L=2,N=1	6x8(75%)	3	2.52(103%)
	L=4,N=2	6x8(75%)	2	2.38(97%)



(a) バス型配線構造 (L=2,N=1)　(b) 複合型配線構造 (L=2,N=1)

図 12　FFT マッピング結果

4.3　8 点 FFT マッピング結果

8 点 FFT マッピングの実験結果を表 5 に、マッピング結果を図 12 に示す。

配線リソースが不足したことで、直接接続型配線構造では 8x8 のアレイに 8 点 FFT をマッピングできなかった。また、20x20 のアレイサイズの直接接続型配線構造の CGRA に対しても、マッピングできなかった。従って、FFT のように演算器間の配線接続が複雑なアプリケーションに対して、直接接続型配線構造の実装効率は著しく低下することが分かる。バス型配線構造と複合型配線構造ともに配線リソースが多い場合のほうが経由 MUX 数が減少したことによって、結果的にクリティカルパス遅延が小さくなっている。また、バス型配線構造と複合型配線構造の経由 MUX 数は同じだが、複合型はバス型に対して個々の MUX 遅延が増加したことで、クリティカルパス遅延が僅かに大きくなっている。

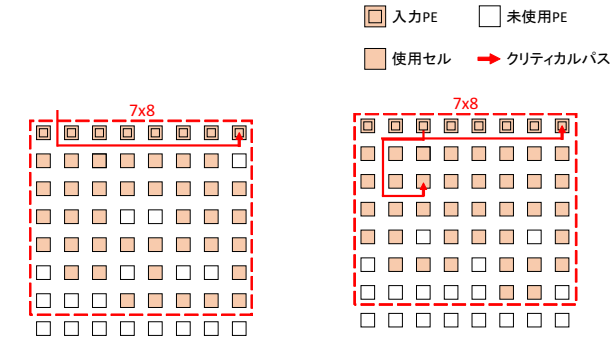
4.4　DCT マッピング結果

8 点 FFT マッピングの実験結果を表 6 に、マッピング結果を図 13 に示す。

遅延については 8 点 FFT と同様の傾向が見られた。バス型と複合型ともに配線リソースの多いほうが経由 MUX 数が減少し、クリティカルパス遅延が小さくなっている。また、バス型に対して複合型が遅延が大きくなっている。

表 6　実装結果（DCT）

配線構造	パラメータ	アレイサイズ	最大経由 MUX 数	最大遅延 [ns]
直接接続型	(L1)x2	8x8(100%)	マッピング不可	
	(L1)x2, (L2)x1, (L4)x1	8x8(100%)		
バス型	L=2,N=1	7x8(87%)	4	3.08(100%)
	L=4,N=2	7x8(87%)	2	2.22(72%)
複合型	L=2,N=1	7x8(87%)	4	3.15(102%)
	L=4,N=2	7x8(87%)	2	2.38(77%)



(a) バス型配線構造 (L=2,N=1)　(b) 複合型配線構造 (L=2,N=1)

図 13　DCT マッピング結果

5. まとめ

本研究では、再構成可能アレイの配線構造を検討するために、自動配置配線によるアプリケーションマッピング結果から、面積効率や動作速度の評価を高精度で行うことのできる環境を構築した。本環境により、提案した 3 種類の配線構造のタイプと配線リソースをパラメータとして定義することで、アプリケーションマッピングによってチップ面積やクリティカルパス遅延を算出し、配線構造の有効性を定量的に評価できるようになったことを確認した。

謝辞　本研究は一部、越智裕之教授（立命館大学/JST）の開発による配置配線ツールを使用して行った。

参考文献

[1] W. Mark Freg, S. Kaptanoglu, “Designing efficient input inter-connect blocks for LUT clusters using counting and entropy,” in Proc. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA)*, pp.23-32, Feb., 2007.

[2] M. Sheng, et al., “Mixing Buffers and Pass Transistors in FPGA Routing Architectures,” in Proc. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA)* pp. 75-84, Feb., 2001.

[3] H. Singh, et al., “MorphoSys: an integrated reconfigurablesystem for data-parallel and computation-intensive applications,” IEEE Transactions on Computers, vol. 49, no. 5, pp. 465-481, May., 2000.

[4] E. Mirsky, et al., “MATRIX: a reconfigurable computing architecture with configurable instruction distribution and deployable resources,” in Proc. *IEEE International Symposium on Field-Programmable Custom Computing Machines(FCCM)*, pp. 157-166, Apr, 1996.

[5] M. Motomura, “A Dynamically Reconfigurable Processor Architecture,” Microprocessor Forum, Oct., 2002.

[6] W. Tsu, et al., “HSRA: high-speed, hierarchical synchronous reconfigurable array,” in Proc. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA)*, pp. 125-134, Feb., 1999.

[7] C. Ebeling, et al., “RaPiD - Reconfigurable Pipelined Datapath,” in Proc. *International Conference on Field Programmable Logic and Applications (FPL)*, pp 126-135, Sep., 1996.

[8] G. Lemieux, et al, “Directional and single-driver wires in FPGA interconnect,” in Proc. *International Conference on Field Programmable Technology(ICFPT)*, pp. 41-48, .Dec., 2004.

[9] Betz. V, et al., “VPR: A New Packing, Placement and Routing Tool for FPGA Research,” in Proc. *International Conference on Field Programmable Logic and Applications (FPL)*, pp. 213-222, Sep., 1997.

[10] S. Friedman, et al., “SPR: an architecture-adaptive CGRA mapping tool,” in Proc. *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays(FPGA)*, pp. 191-200, Feb., 2009.

[11] W. H. Chen, et al., “A Fast Computational Algorithm for the Discrete Cosine Transform,” IEEE Transactions on Communication, vol. 25, pp. 1024-1009, Sept., 1977.