設 計 自 動 化 60－7
（1991．12．12）

レジスタ転送レベルからのデータパス合成の一手法

高橋 瑞樹　　野田 浩明　　横山 昌生　　神戸 尚志

シャープ株式会社

レジスタ転送レベルの記述からの自動合成システムの概要と本システムにおけるデータパス合成手法について述べる。データパス合成は、演算器および転送路の資源割付け問題に分けることができる。本手法では、演算器数だけではなく転送路のコストをも考慮した演算器割付けと、マルチプレクサで実現される転送路の割付けにおいて、転送元及び転送先それぞれのマルチプレクサ割付けを行なうことにより、よりコストの小さい回路の合成を可能にしている。

A Datapath Allocation Method in a RTL Synthesis System

Mizuki TAKAHASHI, Hiroaki NODA, Masao YOKOYAMA, Takashi KAMBE

SHARP Corporation

2613-1 Ichinomoto-cho, Tenri, Nara, 632, Japan

In this paper, we describe a datapath allocation method in our RTL synthesis system. The datapath allocation can be defined as a resource allocation problem of functional blocks and data transfer paths. In our method, we solve a functional block allocation problem considering not only the number of functional blocks but also a data transfer paths cost. The data transfer paths are implemented with multiplexers and connections, and their allocation problems are solved by the source multiplexer allocation and the destination multiplexer allocation. These allocation problems are solved by coloring of undirected graphs.

1 はじめに

集積回路技術の進歩に伴うVLSI化可能な論理回路の大規模化は、設計期間の増大や設計者不足を招いており、大規模回路の設計に対応できる支援技術の重要性が高まっている。この問題に対処すべく、レジスタ転送レベルからの自動合成システム [1][2][3] を開発した。本稿では、合成システムの概要と、本システムのデータパス合成について述べる。従来から、演算器のコストを最小化するデータパス合成の手法は多数提案されているが [4]、実際には転送路のコスト（配線エリアのコスト、マルチプレクサのコスト）も回路全体のコストに大きな影響を与えるものであり最適化を行なわなければならない問題の一つとなっている。本稿で提案するデータパス合成では、単に演算器のコストを最小化するだけではなく、転送路のコストをも考慮して演算器割付けを行なうと共に、転送路の割付けにおいてRTレベル記述で指定された転送路を、転送の制御情報を用いてさらに共有化することにより、転送路も含めてよりコストの小さいデータパスの合成を可能にした。

以下、2章でRTL合成システムの概要、3章でデータパス合成の手法、4章で実験結果について述べる。

2 合成システムの概要

本システムは、レジスタ転送レベルの記述（UDL/I記述）から、その動作を実現するためのデータパス、レジスタおよび有限状態機械として実現される制御回路を合成する。合成される回路のブロック図を図1に示す。

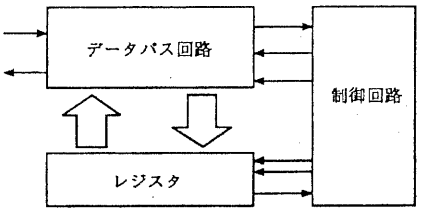

図 1: 合成対象回路

RTL合成システムにおける処理は以下のように分けることができる。

1. レジスタ転送レベル記述からのデータ処理系 / 制御系の分離
   合成対象のデータ処理系の仕様は、ファシリティ（レジスタ、ターミナル）間の転送・演算動作の集合として定義される。合成対象の制御系は、レジスタ転送のクロックに同期して状態が遷移する有限状態機械としてモデル化される。
2. データ処理の実行条件の解析
   制御系の仕様からデータパス系の転送・演算動作間の実行の排他性に関する情報の抽出を行なう。すべての転送動作・演算動作の実行条件のペアに関して論理積の恒偽判定を行ない、実行条件の衝突を解析する。
3. データパス回路の合成
   2で得られた転送・演算の集合と、それらの実行の排他性に関する情報をもとに、演算器等のハードウェア資源の共有を図りながら機能モジュールの割付けを行ない回路を合成する。
4. 制御論理の生成
   状態コードの割付けを行なう。また各演算・転送の機能ブロックへの割付け情報を用いて、データパスの制御論理を生成する。
5. 制御回路の合成
   4で生成された制御論理から、多段論理合成 [5] によって制御回路を合成する。

3 データパス合成

データパス合成は、データパスで実行される動作（転送・演算）の集合とそれらの実行の排他性に関する情報を入力として、ライブラリに登録された機能モジュールで構成されたデータパス回路を出力する。機能モジュールとして用意されているものは、ALU（加算、減算、論理演算）, 比較器, インクリメンタ, デクリメンタ, マルチプレクサおよび複数ビット幅の論理演算器であり、転送路はマルチプレクサによって実現される。

データ転送、同種の演算はそれらが同時に実行されなければ、それぞれ転送路、演算器を共有することができ、データパス合成においては、よりコストの小さい回路を得るために、これらの共有を最適に決定する必要がある。

本データパス合成では、まず演算器の割付けを行ない、その後、演算器へのデータ転送や、ファシリティ間のデータ転送を行なうための転送路の割付けを行なう。

3.1 演算器割付け

演算器割付けは、

- 演算器数最小
- 演算器入力側のマルチプレクサのコスト削減

を目的として、演算を実行する演算器を割付ける。

演算動作をその機能とビット幅によって分割し、分割されたそれぞれの演算動作について資源共有を決

```
a=a+r1;   — ①
if c then
  a=a+r2; — ②
  a=a+r3; — ③
else
  a=a+r4; — ④
  a=a+r5; — ⑤
end_if;
```

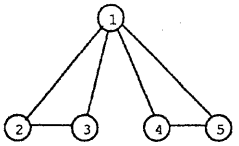

図 2: コンフリクトグラフ

定する。論理演算については、加減算との資源共有（ALU による）は行なうが、論理演算同士のみでの資源の共有は行なわない。これは、共有による論理演算器のコスト削減に比べ入力を選択するための転送路のコスト増加の方が大きくなってしまうためである。このため、論理演算は他の演算（算術演算、比較演算）とは異なる手法で割付けを行なう。

演算器を共有する算術演算および比較演算の決定は、演算をノードとし、同時に実行される可能性がある演算に対応するノード間にエッジを設けたグラフ（コンフリクトグラフ 図2）を作り、このグラフの着色問題を解くことにより行なう。用いているグラフの着色アルゴリズム [6] を付録1に示す。

コンフリクトグラフの着色により同色に塗られたノードに対応する演算は、全て排他的に実行されるので一つの演算器に割付けることができ、最小演算器数の構成を求めることができる。しかし、着色問題を単純に解いただけでは演算器数を最小にする解が得られるだけであり、必ずしも演算器入力側の転送路のコストが小さい解が得られるわけではない。そこで、着色問題を解く際に、演算をオペランドの共有度で順序付けしておき、同色にするノードの候補を選ぶ際の優先順位として用いることにより、演算器入力側の転送路のコスト削減を可能にしている。また、演算器割付け決定後に可換な2項演算に関してはオペランドの入れ換えを行ない、さらに転送路のコスト削減を行なう。

以下の節で、個々の割付け手法について述べる。

3.1.1 算術・比較演算の割付け

算術演算（加算、減算、インクリメント、デクリメント）、比較演算の演算器への割付けは、以下の手続きにより行なう。

1. 算術・比較演算動作をビット幅で分割する
2. 1 の結果をさらに
   - 加算・減算
   - インクリメント・デクリメント
   - 比較演算

で機能別に分割し、そのそれぞれについて (a) ～ (d) を行なう。

(a) 演算をノードとみなし、演算器を共有できない（実行条件が衝突する）演算に対応するノード間にエッジを設けたグラフ G_a を作る。

(b) 演算をオペランドの共有度で順序付けする。これは、i ～ iv の手順で行なう。

$T = \{t_1, t_2, \ldots, t_{Nt}\}$: 順序づけする演算の集合

$S = \{s_1, s_2, \ldots, s_{Ns}\}$: T 内の演算のソース（データ入力元）の集合

$TS(s_i)$: s_i をソースとしている演算の集合

$ST(t_i)$: t_i のソースの集合

とする時

i. すべての $s \in S$ について、$|TS(s)|$ を求める。

ii. $s \in S$ を $|TS(s)|$ で昇順にソートし、その順番を $Priority(s)$ で表す。

iii. $t \in T$ について、

$$X_s(t) = \sum_{s \in ST(t)} Priority(s)$$

を求める。

iv. $X_s(t)$ の昇順に演算を順序づけする。

(c) グラフ G_a を、付録1のアルゴリズムで着色する。

隣接していない2つのノード a,b の選択は、上記 (b) の iv によって、グラフ $G_a = (v_a, e_a)$ の対応するノードが $f : V_a \rightarrow \{1, \ldots, N\}$ ($| V_a | = N$) によって順序づけされているとする。G_a を着色している途中で得られるグラフ $G = (V, E)$ のノード $v \in V$ に対して、

$$G(v) = \min_{v_i \in Nl(v)} (f(v_i))$$

($Nl(v)$ は付録1中で定義されたノードの属性）としたとき、隣接していないノード a, b で $g(a), g(b)$ ともに最小のものを選ぶ。

(d) 同色に着色された演算のグループに対して、一つの演算器をライブラリより選択し割付ける。ただし、加算・減算演算のグループに対しては論理演算の割付けを行なった後で ALU を割付ける。

3.1.2 論理演算の割付け

論理演算の割付けは、算術演算・比較演算の割付け後に、資源共有可能な加算・減算演算の集合を見つけ出す形で以下1～3の手続きにより行なう。

前節の算術演算の割付けで求められたビット幅 w の加算、減算演算を含むグループの集合を AG^w とする。

1. w の昇順に以下の (a)(b) を繰り返す。
   (a) ビット幅 w 以下でかつまだグルーピングされていない論理演算の集合を L^w とする。
   (b) L^w の中の演算について、(AG^w 中で同時実行可能性のあるグループ数) + (L^w 中で同時実行可能性のある演算数) の大きいものから順に i,ii を行なう。
      i. AG^w の中で、演算器共有可能なグループをすべて探す。
      ii. 上記の i で得られたグループ（もしあれば）のうち、コスト増加がもっとも小さくなるグループに目的の論理演算を入れる。
2. どのグループにも入れられなかった論理演算については、個々の演算を1つのグループとする。
3. 各 AG^w および一つの論理演算からなるグループに、それぞれ ALU および論理演算器をライブラリより選択し割付ける。

3.2 転送路割付け

転送路割付けは、与えられた転送動作の集合とファシリティの参照・書き込み条件の排他性に関する情報をもとに、転送元から転送先へ至る転送路を決定するものである。本データパス合成では、複数の転送元から一つの転送先に対して転送がある場合には、マルチプレクサによって競合解消を行なう。このマルチプレクサを転送先マルチプレクサと呼ぶ。一方、転送元から転送先マルチプレクサに至る途中の転送路の共有が行なわれた場合にも、転送路にのせるデータを選択するためのマルチプレクサが割付けられる。これを転送元マルチプレクサと呼ぶ。

転送路割付けの処理は、転送元マルチプレクサ割付け、転送先マルチプレクサ割付け、後処理による最適化の3つのステップからなる。

3.2.1 転送元マルチプレクサ割付け

同時に参照されることのない転送元同士を一つの転送路にのせる。転送路への割付けは、演算器割付けと同様に各転送元の参照条件に関するコンフリクトグラフを作り着色問題を解いて行なう。その際、転送元の参照条件に現れる制御信号の共有度により同色に着色するノード選択の優先順位を定める。これにより、複数の転送先から参照されることの多い、ある制御信号のデコード結果で制御される転送元のグループを同じ転送路にまとめることができ、転送先のマルチプレクサのチャネル数の削減が可能となっている（図3）。

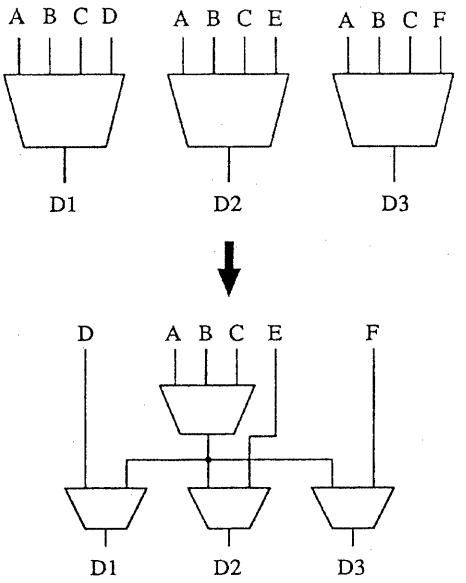

図 3: 転送元マルチプレクサの割付け

転送元マルチプレクサ割付けの手続きを以下に示す。

1. 全ての転送を、転送のビット幅でグループ分けし、それぞれのグループに対して以下の (a) ～ (d) を行なう。
   (a) 転送をその転送元でグループ分けする。
   (b) 各々のグループをノードとみなしたグラフ $G_t = (V_t, E_t)$ を作る。2つのノード間のエッジは、対応するグループ間に転送条件が衝突する転送が存在する場合に設けられる。
   (c) グラフ G_t を付録1のアルゴリズムで着色する。隣接しない2つのノードを選択する際の優先順位としては、転送条件の制御信号の共有度（最も多くの転送の転送条件に現れる制御信号線に対して、その制御信号線を転送条件に持つ転送の転送元が高い順位になる）を用いる。

(d) 同色に着色されたノードに対応する転送元からの転送に対して、マルチプレクサをライブラリより選択して割付け、一つの転送路にまとめる。

3.2.2 転送先のマルチプレクサ割付け

複数の転送元から転送がある転送先に対してはマルチプレクサが必要となる。同時に書き込みが起こらない転送先に対するマルチプレクサは、一つにまとめることで転送路のコストを小さくできる（図4）。これも、演算器割付けと同様に各転送先の書き込み条件に関するコンフリクトグラフを作り着色問題を解いて行なっている。この時、転送元の共有度により同色に着色するノード選択の優先順位を定める。これにより、共通の転送元を持つ転送先のグループが優先的に一つのマルチプレクサに割付けられ、転送先マルチプレクサの総入力チャネル数の削減が可能となっている。

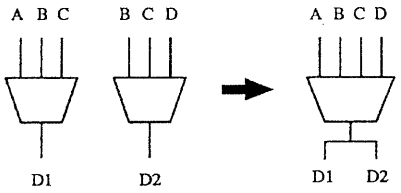

図 4: 転送先マルチプレクサのマージ

転送先マルチプレクサ割付けは以下の手続きにより行なう。

1. 全ての転送を、転送のビット幅でグループ分けし、それぞれのグループに対して (a) ～ (d) を行なう。
   (a) 転送をその転送先でグループ分けする。
   (b) 各々のグループをノードとみなしたグラフ $G_t = (V_t, E_t)$ を作る。2つのノード間のエッジは、対応するグループ間に転送元が異なりかつ転送条件が衝突する転送が存在する場合に設けられる。
   (c) グラフ G_t を付録1のアルゴリズムで着色する。隣接しない2つのノードを選択する際の優先順位としては、転送元の共有度（最も多くの転送先で参照されている転送元に対し、その転送元からの転送の転送先が高い順位になる）を用いる。
   (d) 同色に着色されたノードに対応する転送先への転送に対して、マルチプレクサをライブラリより選択して割付ける。これらの転送は、一つのマルチプレクサで競合解消された後、各転送先へ送られる。

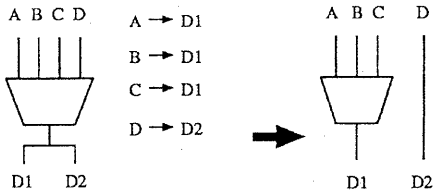

図 5: マルチプレクサ割付けの後処理

3.2.3 後処理による最適化

転送元および転送先マルチプレクサ割付けの後、各マルチプレクサに対して、マルチプレクサで競合解消を行なう必要のない転送を取り除くことにより、さらに入力チャネル数を削減する。（図5）

後処理による最適化は、全てのマルチプレクサに対して以下1～3の処理で行なう。

1. マルチプレクサに割付けられた転送 $T = \{t_1, \ldots, t_{Nt}\}$ の転送元を $S = \{s_1, \ldots, s_{Ns}\}$、転送先を $D = \{d_1, \ldots, d_{Nd}\}$ とする。
2. s_i から d_j への転送で、以下の条件を満たすものを探索する。
   (a) s_i から d_j 以外の転送先への転送が T 中に存在しない
   かつ
   (b) s_i 以外の転送元から d_j への転送が T 中に存在しない
3. 2で見つかった転送をマルチプレクサからはずし、s_i から d_j への直接の転送に変える。

4 実験結果

最後に本システムの適用例として、8ビットCPUのRTレベル記述からの合成例を示す。合成に用いた8ビットCPUは、アドレス16ビット、データ16ビット、アドレスバス16ビット、データバス8ビット、インストラクション数14、汎用レジスタ数16bit×4本の評価用CPUである。インストラクションで用意している演算は、加減算、論理演算（AND, OR, NOT）、比較演算（lower than）である。

合成結果を、表1に示す。ここでは、本データパス合成手法のコスト削減における有効性を示すために、以下の4通りの合成を行なっている。

合成 1 本手法による合成結果

表 1: 評価用 CPU の合成結果

	ゲート数						マルチプレクサ	
	レジスタ	データパス		制御回路		計	総チャネル数	個数
合成 1	1228	806	(91.7)	460	(92.9)	2494 (95.8)	31	11
合成 2	1228	831	(94.5)	481	(97.2)	2540 (97.6)	33	5
合成 3	1228	854	(97.2)	464	(93.7)	2546 (97.8)	35	13
合成 4	1228	879	(100.0)	495	(100.0)	2602 (100.0)	37	7

合成 2 転送元マルチプレクサ割付けを行なわなかった場合の合成結果

合成 3 演算器割付けにおける転送路コストの考慮を行なわなかった場合の合成結果

合成 4 転送元マルチプレクサ割付けと演算器割付けにおける転送路コストの考慮の両方を行なわなかった場合の合成結果

表 1の 4 つのデータパスの合成結果において、割付けられた演算器モジュール (8bit ALU, 8bit incrementer, 8bit comparator それぞれ 1 個づつ) は全く同じであるので、合成結果のゲート数の差は転送路の差である。表 1のマルチプレクサの項目に着目すると、本手法のマルチプレクサ割付け及び演算器割付けにより、転送路のコスト (マルチプレクサの総チャネル数) が削減できている。転送路をトライステートバスで実現する場合には、マルチプレクサの個数がバスの本数に対応することになり、合成 2、すなわち転送元マルチプレクサ割付けに対応する処理を行なわない方が良い結果が得られると考えられる。

また、ここでは転送路の構成最小単位はマルチプレクサとしているが、マルチプレクサよりも下位のレベルで考えた場合、マルチプレクサ間での資源の共有 (図 6) も可能となり、さらに転送路のコストを削減することができると考えられる。

5 おわりに

本合成システムのデータパス合成手法において、転送路のコストを考慮した演算器割付け、及び転送元マルチプレクサ割付けによる転送路コスト削減が、よりコストの小さい回路の合成に有効であることを示した。今後は、本設計手法の適用範囲を拡大すべく、多相クロック、複数の有限状態機械からの合成、ラッチへの対応等、より複雑なモデルからの合成を可能としていく。

参考文献

[1] 野田、高橋、横山、神戸: “レジスタ転送レベル合成システム (1)”, 1991 年電子情報通信学会春季全国大会.

[2] 高橋、野田、横山、神戸: “レジスタ転送レベル自動合成システム (2)”, 1991 年電子情報通信学会春季全国大会.

[3] 野田、高橋、横山、神戸 : “レジスタ転送レベル合成システム”, 1991 年 DA シンポジウム.

[4] Alice C. Parker, Jorge “T” Pizarro, Mitch Mlinar: “MAHA: A Program for Datapath Synthesis”, Proceedings of the 23rd DAC, pp. 461-466, 1986.

[5] 三浦、竹田、神戸: “多段論理合成における時間最適化の一手法”, VLD89-85, 1989.

[6] Bernard Carré: Graphs and networks, CLARENDON PRESS · OXFORD, 1979.

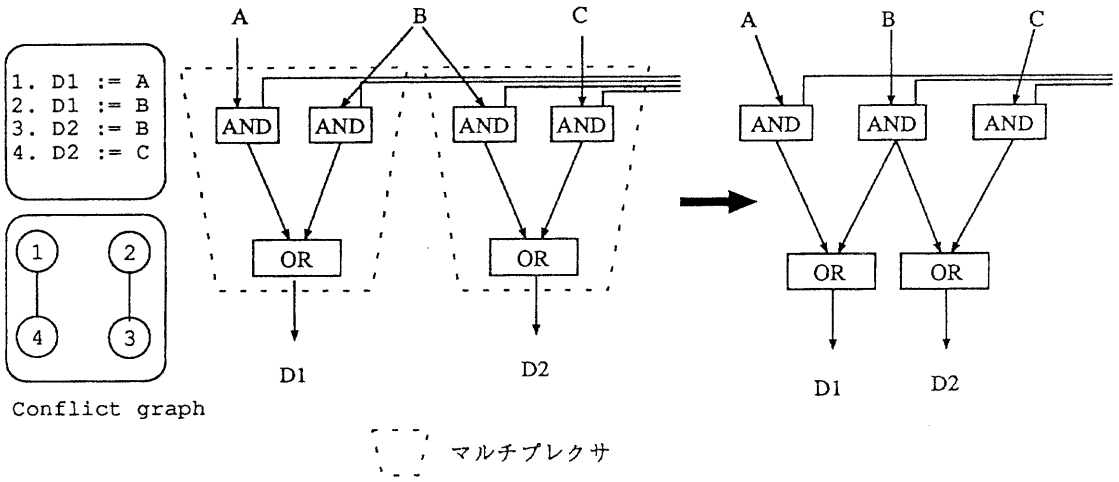

図 6: マルチプレクサ間の資源共有

着色の対象となる初期グラフ $G_{init} = (V_{init}, E_{init})$ のノード $v_{init} \in V_{init}$ は、属性 $Nl(v_{init}) \subseteq V_{init}$ を持ち、$Nl(v_{init}) = \{v_{init}\}$ である。また、再帰処理中にあらわれる $G = (V, E)$ のノード $v \in V$ も属性 $Nl(v) \subseteq V_{init}$ を持つ。

```
着色 (G) {
        if (G が完全グラフ) {
                / * G は一つの解である * /
                if (| G |<| G_min |) { G_min = G;}
                return;
        }
        隣接していないノード a,b を選ぶ;
        G の 2 つのノード a,b をマージして v_ab としたグラフ G' = (V', E') をつくる;
        Nl(v_ab) = nl(a) ∪ Nl(b) とする;
        / * v_ab の G' における隣接集合 Adj_G'(v_ab) は Adj_G'(v_ab) = Adj_G(a) ∪ Adj_G(b) となる * /
        着色 (G');
        if (!(Adj_G(a) ⊆ Adj_G(b) または Adj_G(b) ⊆ Adj_G(a))) {
                G の a,b 間にエッジを設けたグラフ G'' を作る;
                着色 (G'');
        }
}
```

付録 1 ： グラフの着色アルゴリズム