社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2005－SLDM－122 (12)
2005／12／1

動作合成におけるチェイニングに関する考察

貞方 毅† 松永 裕介†

† 九州大学
〒 816-8580 春日市春日公園 6-1
E-mail: †{sadakata,matsunaga}@c.csce.kyushu-u.ac.jp

あらまし 動作合成において、演算のチェイニングは、コントロールステップ数を削減するための有用なテクニックの1つである。従来、チェイニングではマルチファンクション演算器が用いられることはなかった。しかし、マルチファンクション演算器を用いることで、従来よりもステップ数を削減できる可能性がある。本稿では、チェイニングとマルチファンクション演算器を統一的に扱う 仮想演算器という概念を導入することで、チェイニング、スケジューリング、アロケーションを整数線形計画問題として定式化し、資源制約下で従来よりもステップ数を削減可能な動作合成手法を提案する。予備実験では、提案手法が有効であることを確認した。
キーワード EDA、動作合成、チェイニング

A Consideration of Operation Chaining in Behavioral Synthesis

Tsuyoshi SADAKATA† and Yusuke MATSUNAGA†

† Graduate School of Information Science and Electrical Engineering
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580 JAPAN
E-mail: †{sadakata,matsunaga}@c.csce.kyushu-u.ac.jp

Abstract In Behavioral Synthesis, operation chaining is one of the efficient techniques to reduce the number of control steps. Almost conventional methods for operation chaining doesn't use multi-functional units. however, in some cases, using multi-functional leads to decreasing number of control steps. In this paper, the new concept "Virtual Functional Unit" is introduced for considering chaining and multi-functional units simultaneously. With this concept, chaining, scheduling, and allocation problems are formulated as an Integer Linear Problem. Preliminary experimental results shows that the proposed approach achieved more reduction of the number of control steps.
Key words EDA, Behavioral Synthesis, chaining

1. はじめに

動作合成において、資源制約下で性能を最大にする問題設定では、スループットを最大、またはレイテンシを最小とすることが目標となる。著者らは特にレイテンシの削減に関して研究を行っている。レイテンシを削減する技術には、並列実行、投機的実行、チェイングなどがある。並列実行は、互いにデータの依存関係が存在しない演算を同時に実行する技術である。投機的実行は、条件分岐の分岐条件が確定する前に分岐先の演算を投機的に実行する技術である。チェイニングは、互いにデータの依存関係にある演算を同時に同一の資源を用いて実行する技術である。資源制約が比較的緩く資源が豊富な場合は、並列実行、投機的実行、チェイニング、いずれもレイテンシの削減効果は比較的得やすい。しかし、資源制約が厳しく資源が少ない場合、並列実行と投機的実行を行うことは困難な場合がある。一方、チェイニングは、チェイニングの対象となる演算の選択方法、及び、演算器やデータパスの構成を工夫することで、資源の共有率を高め、レイテンシを削減できる可能性がある。しかし、チェイニング対象の選択や演算器構成を誤ると、レイテンシが変わらない、または逆にレイテンシが増加する場合もある。

例えば、図1の Data Flow Graph(以降、DFG) に対し、図2の演算器を使用してスケジューリングを行うことを考える。DFG は、ノードが演算のインスタンスに対応し、エッジがインスタンス間のデータの依存関係を表す、Directed Acyclic Graph(以降、DAG) である。構成1の演算器を使用した場合、DFG のデータの依存関係から最小ステップ数は5となる。構成2では、加算と減算をチェイニングすることで最小ステップ数は4となる。構成3は、資源の共有率を高めるために、加算器と減算器の間にマルチプレクサを挿入して構成となっている。

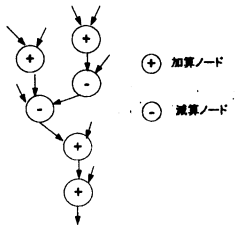

図 1　Data Flow Graph の例

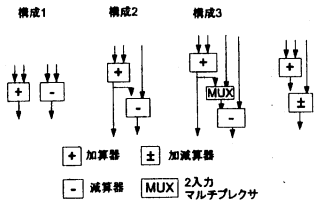

図 2　演算器の構成例

このような構成は、従来のチェイングでよく行われてきた。しかし、図 1 の DFG に対しては、構成 3 は構成 2 と変わらず、最小ステップ数は 4 となる。そこで、構成 4 のように構成 2 の減算器を加減算器に置き換えた構成を考える。この構成を用いることで、最小ステップ数は 3 とすることが可能となる。

従来のチェイング方法では、資源の共有率を高めるためにマルチプレクサを用いてマルチファンクション構成にすることが多かった。しかし、前述の例のように、基本となる演算器自体をマルチファンクション演算器にすることで、さらにステップ数の削減できる可能性がある。

チェイングに関して、これまで様々な研究が行われている。参考文献 [1] [2] [3] では、動作合成を整数線形計画問題として定式化する手法が提案されている。参考文献 [1] では、総面積の最小化を目的関数として、アロケーションとバインディングを合わせて定式化している。チェイングを行う演算はあらかじめ与えられると仮定し、データパス構成は、基本となる演算器を組み合わせることで実現している。参考文献 [2] では、コンポーネント間の接続に関するコストの最小化を目的関数として、スケジューリング、アロケーション、バインディングを合わせて定式化している。チェイングを行う演算は、ステップ数の削減量が多いものを選択している。また、積和演算などに対して専用の演算器を使用することも考慮されている。参考文献 [3] は、積和演算器のような専用の演算器をあらかじめレイアウトレベルまで設計しておくことで、チェイングの効果を高める手法が提案されている。問題の定式化は、参考文献 [2] を基にしている。チェイングする演算はあらかじめ与えられるものと仮定している。参考文献 [4] では、製造時のばらつきによる遅延時間のばらつきを性能確率として定義し、性能確率に基づいてデータパスを逐次的に構成する手法が提案されている。チェイニングする演算はあらかじめ与えられるものと仮定している。参考文献 [5] [6] では、演算をチェイニングした際の遅延時間をビット単位で正確に計算し、資源制約下でクロックサイクルごとの演算器の利用率が高まるようなクロックサイクル時間の選択とスケジューリングアルゴリズムが提案されている。[7] [8] では、演算をビット単位に分解し、ビット単位のチェイニング、スケジューリング、バインディングのアルゴリズムが提案されている。ただし、対象となる演算は加算をベースとしたものに限られる。参考文献 [9] [10] では、チェイニングするパータンをテンプレートとして用意し、テンプレートとマッチする演算を探し、マッチした演算の内から実際にチェイニングを行う演算を選択する手法が提案されている。また、演算器の利用率を高めるために適切なクロックサイクル時間の選択も行っている。参考文献 [11] では、スケジューリングを行う前に、ビット単位の演算を貪欲に探索しチェイニングする方法が提案されている。以上の従来研究では、いずれもマルチファンクション演算器が考慮されていない。

本稿では、演算器とマルチプレクサの組み合わせで実現されるマルチファンクションと、加減算器のように演算器自身が有するマルチファンクションを統合して取り扱う 仮想演算器という概念を導入する。仮想演算器を用いることで、マルチファンクション演算器を含んだ、チェイング、演算器の種類と数の決定、スケジューリング、アロケーション問題を整数線形計画問題として定式化し、動作周波数及び資源制約下で従来よりもコントロールステップ数を削減することが可能な手法を提案する。

本稿の構成は以下の通りである。2 章で準備として本稿で用いる仮定及び定義を説明する。3 章で問題の定式化を行う。4 章で予備実験について述べる。5 章で本稿をまとめる。

2. 準　　備

2.1 仮　　定

チェイニングを行うためには、チェイニング候補の列挙、チェイニング対象の選択、チェイニングを実現するな演算器の構成、演算器の種類と数の決定、スケジューリング、アロケーションを行う必要がある。

チェイニング候補の列挙方法には、あらかじめ用意したパターンとのマッチングによる方法 [9] [10] などがある。特徴的な規則性を抽出する方法も考えられるが、規則性の抽出をチェイニングに利用した研究はほとんどない。パターンマッチングや規則性の抽出は比較的計算コストが高いため、候補の列挙はあらかじめ何らかの方法で行われているという仮定を行っている場合が多い。本稿でも、候補は何らかの方法で列挙されているものと仮定する。

チェイニング対象の選択方法には、貪欲に選択する方法 [11]、線形計画問題として定式化して解く方法 [2] などがある。本稿では、チェイングの対象の選択は、整数線形計画問題に組み込むものとする。

演算器の構成方法は、大きく分けて 2 つある。1 つは、基本となる演算器とマルチプレクサなどの接続素子を組み合わせて実現する方法である。もう 1 つは、あらかじめ演算器のライブラリに専用の演算器として用意しておく方法である。前者の方法では、データパス構成の自由度が高いため、資源制約と遅延時間の制約が許す限り演算をチェイニングすることが可能となる。一方、後者の方法では、あらかじめ演算器を設計しておく

必要があるため、チェイニングできる演算の数は限定されてしまうが、あらかじめ設計することで遅延時間や面積を小さくすることが可能となる。ただし、前者はバインディングを含むため、後者よりも問題は複雑になる。本稿では、問題の簡単化のために、後者の方法を用いることにする。

演算器の種類と数の決定、スケジューリング、アロケーションは、チェイニング対象の選択とともに、整数線形計画問題として定式化し同時に解くことにする。

合成対象は、DFG $G(V.E)$ とする。各ノードの演算で取り扱うデータのビット幅は一律とする。子ノードを持たないノードを sink ノードとし、sink ノードは必ず 1 つだけ存在するものとする。また、DFG の構造は変換せず、各ノードの演算は必ず 1 回だけ実行されるものとする。

制約として動作周波数と演算器の総面積が与えられ、目的はコントロールステップ数を最小化することとする。なお、記憶素子、接続素子のコストは考慮しない。

演算器のライブラリには、制約として与えられる動作周波数で動作可能で、かつ、チェイニングを実現可能な演算器が存在するものとする。演算器の面積は、動作周波数ごとに一意に定まるものとする。演算器は、マルチサイクル、マルチファンクション、非パイプラインのものを扱うものとする。

2.2 定　　義

DFG 上でチェイニングの候補となるノードの集合をチェイニング候補と定義する。

[定義 1] DFG $G(V.E)$ のノードの集合 $j \subseteqq V$ の内、j によって誘導さ れる部分グラフ $G'(V', E')$ が、以下の条件を満たすとき、j をチェイニング候補とする。

（1） $|V'| \geqq 1$

（2） $|V'| > 1 \Rightarrow \forall v_1' \in V', \exists v_2' \in V'.(v_1', v_2') \in E \vee (v_2'. v_1') \in E$

（3） $\forall v' \in V', \forall v'' \in PARENTS(v') - (PARENTS(v') \cap V'), |PRED(v'') \cap V'| = 0$

ただし、*PARENTS(v)* はノード v の G 上での親ノードの集合を表し、*PRED(v)* はノード v の G 上の先行ノードでの集合を表す。

1 つ目の条件により、ノード数が 1 の場合も特別なチェイニング候補として含まれる。2 つ目の条件は、部分グラフが G 上で連結であるための条件である。3 つ目の条件は、データの依存関係を満たすための条件である。

次に、演算器のマルチファンクション性を扱うために、仮想演算器という概念を導入する。仮想演算器は、演算器の機能を固定した場合の仮想的な構成を表現するものである。演算器と演算とは、仮想演算器を介して関連付けられる。仮想演算器と演算の対応は、仮想演算器が同時に独立に実行可能な演算の数で関係付けられる。

例えば、図 3 の場合を考える。演算器として、加算器 (h_1)、加減算器 (h_2)、加算器と減算器とマルチプレクサを接続したもの (h_3) が用意されている。一方、仮想演算器として、加算器 (m_1)、減算器 (m_2)、加算器と減算器 (m_3)、加算器と減算器を接続したもの (m_4) が用意されている。h_1 は、2 入力の加算器

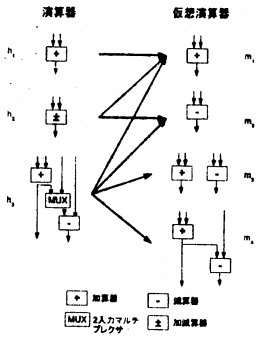

図 3　演算器の種類と仮想演算器の種類の対応関係の例

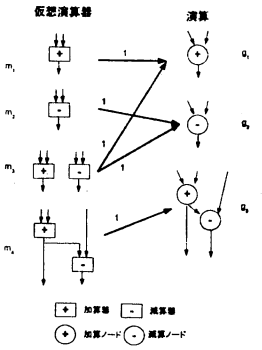

図 4　演算器の種類と仮想演算器の種類の対応関係の例

のため、対応する仮想演算器は同様の構成である m_1 となる。h_2 は加減算器であるため、加算器としても減算器としても用いることが可能である。ただし、加算と減算を同時に実行することはできない。よって、対応する仮想演算器は m_1 または m_2 となる。h_3 は、マルチプレクサによってマルチファンクション構成となっているので、m_3 の構成とも、m_4 の構成ともみなすこともできる。また、m_1 や m_2 とみなすことも可能である。よって、h_3 は、m_1〜m_4 の仮想演算器と対応するものとする。

図 4 は、図 3 で示された仮想演算器と演算との対応関係を表している。演算として、加算 (g_1)、減算 (g_2)、加算と減算をチェイニングしたもの (g_3) がある。m_1 と m_2 は、単純な加算器または減算器であるので、対応する演算は加算である g_1 または減算である g_2 となる。対応関係を表す矢印につけられたラベルは、その仮想演算器によって独立に同時に実行可能な演算の数を表している。例えば、m_3 は、加算と減算を独立に同時に 1 個ずつ実行することが可能である。よって、g_1 と g_2 に対応付けられ、ラベルがそれぞれ 1 となっている。m_4 は、g_3 に対応付けられる。m_4 は、g_1 を実行することも可能であるが、その対応関係は、図 3 の h_3 と m_1 によってすでに対応付けられているので、m_4 と g_1 は対応付けを行わない。

以上のことを踏まえて、表 1 に本稿で用いる集合、写像、関係を定める。なお、本稿では、演算の種類と演算のインスタンス、演算器の種類と演算器のインスタンスを明確に区別している。DFG 上のチェイニング候補 $j \in J$ を演算のインスタンスとみなし、j と演算の種類 $g \in G$ が写像 *optype* で対応付けら

れる。演算の種類はDFGによって表現される演算のパターンを表す。演算器のインスタンス $k \in K$ と演算器の種類 $h \in H$ は写像 $type$ で対応付けられる。演算器の種類と仮想演算器の種類は、写像 q によって関連付けられる。仮想演算器の種類と演算の種類は、写像 p によって関連付けられる。写像 l と a は、演算器の種類 $h \in H$ ごとの処理に要するステップ数及び面積を表す。また、演算器のインスタンス $k \in K$ と仮想演算器 $m \in M$ の実現可能な組み合わせが $(k,m) \in B$ で表される。H、M、q、p、l、a は、演算器のライブラリの情報として与えられるものとする。K 及び B は、制約として与えられる面積と演算器のライブラリの情報に基づいて構成される。ADJ は、チェイニング候補間のデータの依存関係を表しており、$ADJ = \{(j_1,j_2) | j_1 \cap j_2 = \phi \wedge \exists e = (v_1,v_2) \in E, v_1 \in j_1, v_2 \in j_2\}$ として定義される。

3. 定式化

チェイニング対象の選択、使用する演算器の種類と数の決定、スケジューリング、アロケーションを同時に解く問題を、整数線形計画問題として定式化する。

3.1 変数

0または1の値をとる2値変数として、以下の $x_{i,j,k,m}$ 及び y_k を用いる。

$$x_{i,j,k,m} = \begin{cases} 1 & \text{ステップ } i \text{ において} \\ & \text{演算器のインスタンス } k \text{ を} \\ & \text{仮想演算器 } m \text{ として使用して} \\ & \text{チェイニング候補 } j \text{ を } k \text{ で実行開始する} \\ 0 & \text{上記以外} \end{cases}$$

$$y_k = \begin{cases} 1 & \text{演算器のインスタンス } k \text{ を使用する} \\ 0 & \text{上記以外} \end{cases}$$

$x_{i,j,k,m}$ によって、チェイニング候補の選択、スケジューリング、アロケーション、演算器の機能の固定を表す。y_k によって、使用する演算器の種類と数を表す。

3.2 制約1: 実現可能性の制約

B に含まれない k と m の対に対応する $x_{i,j,k,m}$ は実現可能ではないので、あらかじめ0とする。また、B に含まれるが、j を実行可能でない m に対応する $x_{i,j,k,m}$ も実現可能ではないので、0とする。

$$\forall i \in I, \forall j \in J, \quad \sum_{(k,m) \notin B} x_{i,j,k,m} + \sum_{(k,m) \in B : p(m,optype(j)) < 1} x_{i,j,k,m} = 0$$

$x_{i,j,k,m}$ は0または1の値しかとりえないので、上式に含まれる変数はすべて0となる。

3.3 制約2: 演算のインスタンスの実行制約

DFG $G(V,E)$ の各ノード $v \in V$ に対応する演算のインスタンスは、必ず1回だけ実行するという制約を次の式で表す。

$$\forall j \in J, |j| = 1, \quad \sum_{j' \in J : j \subseteq j'} \sum_{i \in R(j')} \sum_{(k,m) \in B : p(m,optype(j')) > 0} x_{i,j',k,m} = 1$$

各チェイニング候補の内、要素数が1、つまりDFGのノードに対応する演算を包含するすべてのチェイニング候補を考える。ノードに対応する演算は、1度だけしか実行できないと仮定したので、これらのチェイニング候補のうち、1つだけしかスケジューリングできないことを上式は表している。

3.4 制約3: 演算器数の制約

使用されない演算器のインスタンスは、最終的に使用する演算器として残しておいても無駄である。以下の式によって、未使用の演算器が選ばれることを防ぐ。

$$\forall k \in K, \quad y_k \leqq \sum_{i \in I} \sum_{j \in J} \sum_{m \in M} x_{i,j,k,m}$$

演算器のインスタンス k がまったく使用されていなければ、右辺が0となるので、y_k も0となる。

3.5 制約4: 演算器の機能固定制約

各演算器のインスタンスは、ステップごとに、1つの機能しか使用できない。以下の式は、演算器を仮想演算器として使用する場合の制約を表す。

$$\forall i \in I, \forall k \in K, \forall j_1, j_2 \in J, \forall m_1, m_2 \in M, m_1 \neq m_2, \quad x_{i,j_1,k,m_1} + x_{i,j_2,k,m_2} \leqq y_k$$

上式により、各演算器のインスタンス k は、ステップごとにある1種類の仮想演算器としてしか使用できないことになる。

3.6 制約5: 仮想演算器の同時実行可能な演算の数の制約

演算器を仮想演算器に基づいて機能を固定した場合の実行可能な演算の数の制約を、以下の式で表す。

$$\forall i \in I, \forall (k,m) \in B, \forall g \in G, \quad \sum_{j \in J : optype(j) = g} x_{i,j,k,m} \leqq p(m,g)$$

上式により、各ステップで、演算器のインスタンス k を仮想演算器 m としてチェイニング候補を実行する際、写像 p によって定められる独立に同時に実行可能な演算の数を超えて、同じ演算の種類のチェイニング候補がスケジューリングされることを防ぐ。

3.7 制約6: 演算器の排他的使用に関する制約

演算器のインスタンスで演算の実行を行っている間は、別の演算を実行できない。以下の式で、演算器の排他的使用の制約を表す。

$$\forall i \in I, \forall (k,m) \in B, \forall j_1, j_2 \in J, \quad x_{i,j_1,k,m} + \sum_{i'=i+1}^{i+l(type(k),m)-1} \sum_{m' \in M : (k,m') \in B} x_{i',j_2,k,m'} \leqq 1$$

上式は、あるステップ i で演算器のインスタンス k を仮想演算器 m として使用しチェイニング候補 j_1 の実行を開始する場合、次のステップから実行が完了するまでのステップの間は、k は他のチェイニング候補の実行を開始できないことを表している。

表 1　本稿で用いる集合、写像、関係

N	自然数の集合
R^+	0 以上の実数の集合
$G(V, E)$	合成対象の DFG
$I \subseteqq N$	コントロールステップ i の集合
$J \subseteqq 2^V$	チェイニング候補 j の集合
K	演算器のインスタンス k の集合
G	演算の種類 g の集合
H	演算器の種類 h の集合
M	仮想演算器の種類 m の集合
$R(j) : J \mapsto 2^I$	j の開始ステップの集合を返す写像
$type(k) : K \mapsto H$	k から h を得る写像
$optype(j) : J \mapsto G$	j から g を得る写像
$p(m, g) : M \times G \mapsto N$	m が同時に独立に実行可能な g の数を返す写像
$q(h) : H \mapsto 2^M$	h が実現可能な m の集合を返す写像
$l(h, m) : H \times M \mapsto N$	h を m として使用する際に処理の完了に要するステップ数を返す写像
$a(h) : H \mapsto R^+$	h の面積を返す写像
$B = \{(k, m) \mid k \in K, m \in q(type(k))\}$	実現可能な k と m の対を表す関係
$ADJ \subseteqq J \times J$	チェイニング候補の DFG 上の隣接関係

表 2　各演算器の面積

演算器	面積 [μm^2]
h_1	4270
h_2	6666
h_3	68859
h_4	3840
h_5	75118
h_6	9923
h_7	72622
h_8	77452
h_9	8340
h_{10}	78420
h_{11}	79235
h_{12}	78198
h_{13}	11205
h_{14}	8279
h_{15}	79672
h_{16}	15528
h_{17}	82068
h_{18}	84971

3.8　制約 7: チェイニング候補の依存関係の制約

チェイニング候補のデータの依存関係の制約を次式で表す。

$$\begin{aligned}
&\forall (j_1, j_2) \in ADJ, |j_2| = 1, \\
&\sum_{i_1 \in R(j_1)} (i_1 \cdot \sum_{(k,m) \in B} x_{i_1, j_1, k, m} \\
&+ \sum_{(k,m) \in B} x_{i_1, j_1, k, m} \cdot l(type(k), m)) \\
&\leqq \sum_{\substack{j_2' \in J : |j_1 \cap j_2'| = 0 \\ \wedge |j_2 \cap j_2'| = 1}} \sum_{i_2 \in R(j_2')} (i_2 \cdot \sum_{(k,m) \in B} x_{i_2, j_2', k, m})
\end{aligned}$$

チェイニング候補 j_1 にデータ依存しているノード数が 1 の候補 j_2 を包含する候補は、すべて j_1 よりも後のステップにスケジューリングされなければならないことを上式は表している。左辺の第一項は j_1 の開始ステップを表し、第二項は実行が完了するのに要するステップ数を表す。右辺は、j_2' の開始ステップを表す。

3.9　制約 8: 資源制約

資源制約として、使用する演算器のインスタンスの総面積 $A \in R$ が与えられるものとする。資源制約は、次式で表される。

$$\sum_{k \in K} y_k \cdot a(type(k)) \leqq A$$

3.10　目的関数: コントロールステップ数の最小化

前述の通り、DFG には sink ノード必ず 1 つだけ存在すると仮定した。sink ノードは実際の演算のインスタンスに対応するものではなく、目的関数の定式化のために便宜的に追加されたノードである。sink ノードだけを含むチェイニング候補を j_{sink} とした場合、目的関数は次式で表される。

$$minimize(\sum_{i \in R(j_{sink})} i \cdot \sum_{(k,m) \in B} x_{i, j_{sink}, k, m})$$

以上の定式化を用い、動作周波数及び演算器の総面積が制約として与えられた場合に、ステップ数が最小となる、チェイング対象の選択、演算器の種類と数の決定、スケジューリング、アロケーションを実現する。

4.　予備実験

動作合成用ベンチマーク集 HLSynth92 [12] に含まれる微分器のループボディの DFG を合成対象として予備実験を行った。図 5 は、その DFG である。基本となる演算は加算、減算、乗算、大小比較である。チェイニング候補はノード数が 2 のものとした。ただし、2 つの乗算からなるチェイニング候補は対応する演算器が巨大になるため候補から除いた。

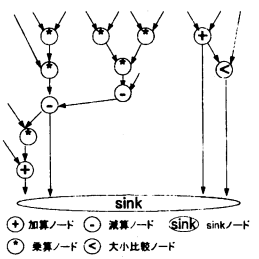

図 5　微分器の DFG

演算器は図 6 のものを用意した。h_1〜h_9 は、マルチファンクションではない演算器である。h_{10}〜h_{13} は、マルチプレクサを用いた従来のマルチファンクション演算器である。h_{14}〜h_{18} は、加減算器を含むマルチファンクション演算器である。これらの演算器を、VDEC で提供されている日立 0.18μm プロセス向けの京都大学作成のセルライブラリと、Synopsys 社の DesignCompiler(W-2004.12-SP4 版) を用いて、動作周波数を 8.0ns として面積最小で論理合成を行い、面積の値を得た。表

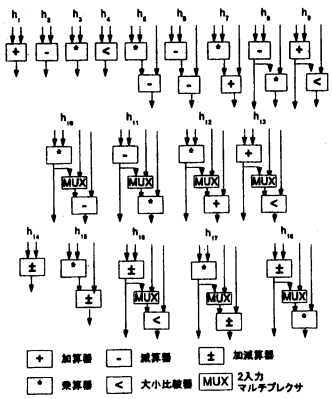

図 6　実験に用いた演算器

表 3　微分器の合成結果

面積 [μm^2]	h_1〜h_4	h_1〜h_{13}	h_1〜h_{18}
85000	8	7	7
95000	8	7	6
105000	8	7	6
115000	8	7	6
125000	8	7	6
135000	8	7	6
145000	8	7	6
155000	6	4	4
165000	6	4	4

2 が、得られた面積の一覧である。

演算器の面積の値を基に、整数線形計画問題として定式化し、ILOG 社の線形計画問題ソルバーである CPLEX を用いて解を得た。表 3 は、得られた結果である。h_1〜h_4 は、h_1 から h_4 までの演算器を使用した、チェイニングを行わない場合の結果である。h_1〜h_{13} は、h_1 から h_{13} までのマルチプレクサによるマルチファンクション構成の演算器を含めた、従来のチェイニング方法の結果である。h_1〜h_{19} は、h_1 から h_{19} までの加減算器を含めた、提案手法による結果である。面積が 85,000μm^2 未満では、どの場合においても面積制約を満たす組み合わせは存在せず、解は得られなかった。面積制約が 90,000〜150,000μm^2 では、提案手法では、従来手法よりも 1 ステップ削減できていることがわかる。面積制約が 155,000μm^2 以上では、提案手法も従来手法も同じ結果となった。このように、基本となる演算器としてマルチファンクション演算器を用いることで、面積制約が厳しい場合でも、従来よりステップ数が削減できることが確認できた。

5. おわりに

本稿では、仮想演算器という概念を導入し、チェイニングとマルチファンクション演算器を同時に考慮することで、資源制約下において従来よりもステップ数の削減を可能とする手法を提案した。予備実験結果では、その効果を確認した。今後は、チェイニング候補のノード数を大きくした実験、他のベンチマークに対しての実験を行っていく予定である。また、バインディングを考慮した定式化、ヒューリスティックな方法の開発が今後の課題である。

謝辞

本研究は、一部科学研究費補助金 (学術創成研究費 (2))(課題番号：14GS0218) による。本研究は、東京大学大規模集積システム設計教育研究センターを通し、Synopsys ツールを用いて行われたものである。本研究は、東京大学大規模集積システム設計教育研究センターを通し、日立製作所株式会社の協力で行われたものである。

文　献

[1] Minjoong Rim, Rajiv Jain and Renato De Leone, "Optimal Allocation and Binding in High-Level Synthesis", In Proceedings of the 29th ACM/IEEE conference on Design automation, pages 120-123, 1992.

[2] Birger Landwehr, Peter Marwedel, Rainer Dömer, "OSCAR: Optimum Simultaneous Scheduling, Allocation and Resource Binding Based on Integer Programming", In Proceedings of the conference on European design automation, pages 90-95, 1994.

[3] Peter Marwedel, Birger Landwehr, Rainer Dömer, "Built-in Chaining: Introducing Complex Components into Architectural Synthesis", In Proceedings of the Asia South Pacific Design Automation Conference, 1997.

[4] Hiroyuki Tomiyama, Akihiko Inoue, Hiroto Yasuura, "Statistical Performance-Driven Module Binding in High-Level Synthesis", In Proceedings of the 11th International Symposium on System Synthesis, pages 66-71, 1998.

[5] Sanghum Park, Kiyoung Choi, "Performance-Driven Scheduling with Bit-Level Chaining", In Proceedings of the 36th ACM/IEEE conference on Design automation, pages 286-291, 1999.

[6] Sanghun Park and Kiyoung Choi, "Performance-Driven High-Level Synthesis with Bit-Level Chaining and Clock Selection", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 20, no. 2, pages 199-212, Feb. 2001.

[7] M.C. Molina, J.M. Mendias, R. Hermida, "Bit-level Scheduling of Heterogeneous Behavioural Specifications", In Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design, pages 602-608, 2002.

[8] R. Ruiz-Sautua, M. C. Molina, J.M. Mendias, R. Hermida, "Behavioural Transformation to Improve Circuit Performance in High-Level Synthesis", In Proceedings of the conference on Design, Automation and Test in Europe, vol. 2, pages 1252-1257, 2005.

[9] M. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, J. Rabacy, "INSTRUCTION SET MAPPING FOR PERFORMANCE OPTIMIZATION", In Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design, pages 518-521, 1993.

[10] Miguel R. Corazao, Marwan A. Khalaf, Lisa M. Guerra, Miodrag Potkonjak, and Jan M. Rabacy, "Performance Optimization Using Template Mapping for Datapath-Intensive High-Level Synthesis", IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 15, No. 8, Aug. 1996.

[11] Tai Ly, David Knapp, Ron Miller, Don MacMillen, "Scheduling using Behavioral Templates", In Proceedings of the 32nd ACM/IEEE conference on Design automation, pages 101-106, 1995.

[12] N. Dutt, "Current Status of HLSW Benchmarks and Guidelines for Benchmark Submission", HLSynth'92 Benchmark, Sept. 1992.