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

2005－ＳＬＤＭ－119（４）
2005／3／17

チェイニングを考慮した動作合成手法

貞方　毅[†]　松永　裕介[††]

† 九州大学大学院システム情報科学府
〒 816-8580 春日市春日公園 6-1
†† 九州大学大学院システム情報科学研究院
〒 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　動作合成においてコントロールステップ数最小を目指す場合、チェイニングは有効な手段である。しかし、どの演算をチェイニングするのかというチェイニングの方針についてはこれまでほとんど研究されていない。そこで本研究では、効率的に効果的にチェイニングを行う動作合成手法を提案する。実験ではチェイニングの効果を確認し、同時に動作周期または演算器の数が増えると結果が悪化するという問題点を確認した。問題点を改善することが今後の課題である。

キーワード　EDA、動作合成、チェイニング

A Behavioral Synthesis Method Considering Chaining

Tsuyoshi SADAKATA[†] and Yusuke MATSUNAGA[††]

† Graduate School of Information Science and Electrical Engineering
6-1 Kasuga-koen, Kasuga, Fukuoka 816-8580 JAPAN
†† 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 a behavioral synthesis, chaining is an effective technique for minimizing the number of control steps. However, the strategy of chaining has not been studied. In this paper, an effective and efficient behavioral synthesis method considering chaining is proposed. The experimental results show that the method achieved effective chaining, but some results was worse than those under less clock-cycle time or the less number of functional units. To improve the method for solving this problem is the future work.

Key words　EDA, Behavioral Synthesis, chaining

1. はじめに

動作合成の制約条件には、動作周波数、面積、コントロールステップ数、スループットなどがある。その中でも、動作周波数は制約条件として与えられることが多い。なぜならば、実際のLSIの開発において、目標となる動作周波数があらかじめ定められることが多いためである。よって、動作周波数が制約として与えられた場合に、資源制約下で性能を最大、または性能制約下で資源を最小とする問題設定が一般的である。

資源制約下においてコントロールステップ数を最小とする問題設定では、各処理を実行するステップを決定するスケジューリング処理が重要となる。ステップ数が最小となる厳密解を求めるためのスケジューリングアルゴリズムとして、整数線形計画法による方法 [1] [2] が提案されている。また、近似解を求めるためのヒューリスティックなアルゴリズムとして、リストスケジューリング [3]、Force-directed リストスケジューリング [4] などが提案されている。ただし、これらのアルゴリズムでは、データ依存関係にある演算は必ず異なるステップにスケジューリングされる。そのため、例えば内部のデータ構造がData-Flow Graph(DFG) であるとすると、ステップ数の下限はDFGのパスの中でステップ数が最大のものとなり、それ以上ステップ数は削減することができない。

データ依存関係にある演算を同一のステップにスケジュールするテクニックはチェイニングとして知られている。チェイニングを行うことで、よりステップ数を削減することが可能となる。例えば図1のようなDFGに対しスケジューリングを行う

とする。図1において、'+' とラベルづけされたノードは加算を表し、'*' とラベルづけされたノードは乗算を表す。チェイニングを行わなければ、図の左側のようなスケジューリング結果となり、総ステップ数は2となる。チェイニングを行い、データの依存関係が存在する2つの加算を1ステップにスケジュールすると、総ステップ数は1となる。ただし、チェイニングを行うために加算器を1つ追加し、加算器を2段にする必要がある。このように、チェイニングを行う場合には、演算器の構成を同時に考える必要がある。

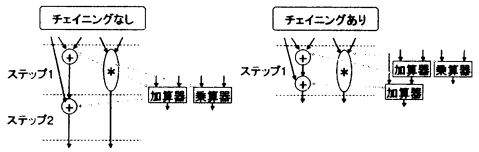

図1 チェイニングの例

チェイニングで重要となるのは、どの演算をチェイニングするかということである。特に、資源制約下においては、単純にステップ数の削減量が大きくなるようにチェイニングを行えば良いとは限らない。例えば、図2のような場合である。図2のDFGに対し、資源制約として加算器の数が3個であると仮定する。チェイニングを行う場合、図2のように2通りにチェイニングが考えられる。チェイニング1のように、ステップ数の削減量が大きくなるようにチェイニングを行ったとしてもステップ数は3にしかならない。ところが、チェイニング2のようにチェイニングを行うとステップ数は2まで削減することができる。このように、どの演算をチェイニングするかでステップ数は大きく変わる。

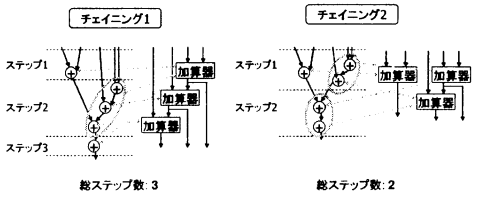

図2 チェイニング対象の違いでステップ数の差が出る例

チェイニングを行うことでステップ数を削減できることは以前から知られていた。しかし、どの演算をチェイニングすれば良いのかというチェイニングの方針に関する研究は全く行われていない。多くの場合、全探索的にチェイニング対象が決定されていた[7]。

そこで本研究では、チェイニングを考慮した動作合成手法を提案する。本手法は、動作周期(動作周波数の逆数)及び演算器の数が制約である場合に、チェイニング対象となる演算と対応する演算器構成の内、特徴的なもののみを列挙することで、効率的に効果的にチェイニングを行う手法である。

本稿の構成は次のとおりである。2章で本研究で用いる概念の定義及び仮定の説明を行う。3章で提案手法の説明を行う。4章で提案手法を用いた実験について説明する。5章で本稿をまとめる。

2. 準　　備

本研究では、動作周期及び基本となる演算器の数が制約条件として与えられた場合に、ステップ数の最小化を目指す動作合成に取り組む。合成の対象は、Data-Flow Graph(DFG) とする。DFG は次のように定義する。

[定義 1] DFG $G_d = (V, E)$ は、基本演算間のデータ依存関係を表す有向非循環グラフである。V は、ノードの集合で、基本演算と対応する。E は、エッジの集合で、基本演算間のデータ依存関係を表す。

基本演算とは、加算や乗算などのあらかじめ定義された基本的な演算を意味する。各基本演算には、演算を実行するための演算器が存在するとし、それらを基本演算器とする。つまり、基本演算器の集合を BFU とすると、$\alpha : V \mapsto BFU$ という関数が存在するものとする。基本演算器が基本演算を実行するのに要するステップ数は、自然数の集合を N、正の実数の集合を R^+、動作周期を $t_{cycle} \in R^+$ とした場合に、$\beta : BFU \mapsto N$ となる関数で定義されるものとする。ただし、$\forall bfu \in BFU$ の最大遅延時間は、動作周期 t_{cycle} を越えないものとする。

制約である基本演算器の数が、関数 $\lambda : BFU \mapsto N$ で与えられるとすると、チェイニングを行わないスケジューリングは次のように定式化することができる。

[定義 2] スケジューリングとは、次の条件を満たすノードへのラベリングを行う関数 $\varphi : V \mapsto N$ を見つけ、$\max_{\forall v \in V}(\varphi(v) + \beta(\alpha(v)) - 1)$ を最小とすることである。

(1) $\forall (v_x, v_y) \in E, \varphi(v_x) + \beta(\alpha(v_x)) \leqq \varphi(v_y)$

(2) $\forall n \in N, \forall bfu \in BFU, |\{v | v \in V \wedge \varphi(v) \leqq n \leqq \varphi(v) + \beta(\alpha(v)) - 1 \wedge \alpha(v) = bfu\}| \leqq \lambda(bfu)$

第1の条件は、データ依存関係が存在する基本演算は異なるステップにスケジュールされることを表している。第2の条件は、すべてのステップにおいて制約である基本演算器の数を満足することを表している。

チェイニングを行う場合、チェイニングの対象となるのはDFG 上の連結な部分グラフが表す演算になる。そこで、チェイニングの対象となる演算を複合演算と定義する。

[定義 3] DFG $G_d = (V, E)$ の連結な部分グラフ $G' = (V', E')$ が次の条件を満たすとき、G' を複合演算とする。

(1) $\forall (v_x, v_y) \in E'$ に対し、G_d 上の v_x から v_y へのすべてのパスが $V - V'$ に含まれるノードを通らない

上記の条件は演算のデータ依存関係に矛盾を生じさせないための条件である。なお、上記の定義では、ノード数が1、つまり基本演算のみからなる部分グラフも複合演算となる。実際には、チェイニングの対象となるのはノード数が2以上の複合演算である。ノード数が1の複合演算は、区別するために単一演算と呼ぶことにする。また、複合演算を実現するために、基本演算器を組み合わせて構成される演算器を複合演算器とする。

DFG G_d に対し、複合演算の集合を $COPE$、複合演算器

の集合を CFU、複合演算と複合演算器の対応を表す関数を $\rho : COPE \mapsto 2^{CFU}$、各複合演算器が複合演算を実行するのに要するステップ数を表す関数を $\sigma : CFU \mapsto N$、各複合演算器が使用する基本演算器の数を表す関数を $\tau : CFU \times BFU \mapsto N \cup \{0\}$ とし、チェイニングを用いたスケジューリングを次のように定式化する。

[定義 4] スケジューリングとは、次の条件を満たす $COPE' \subseteqq COPE$、$\varphi : COPE' \mapsto N$、$\psi : COPE' \mapsto CFU$ を見つけ、$\max_{\forall G' \in COPE'} (\varphi(G') + \sigma(\psi(G')) - 1)$ を最小にすることである。

(1) $\bigcap_{\forall G'=(V',E') \in COPE'} (V') = \phi$

(2) $\bigcup_{\forall G'=(V',E') \in COPE'} (V') = V$

(3) $\forall G' \in COPE', \rho(G') \neq \phi \wedge \psi(G') \in \rho(G')$

(4) $\forall (v_x, v_y) \in E, \exists G'_x, G'_y \in COPE' \ (G'_x \neq G'_y \wedge v_x \in V'_x \wedge v_y \in V'_y) \Rightarrow \varphi(G'_x) + \sigma(\psi(G'_x)) \leqq \varphi(G'_y)$

(5) $\forall n \in N, \exists COPE'_n = \{G' | G' \in COPE' \wedge \varphi(G') \leqq n \leqq \varphi(G') + \sigma(\psi(G')) - 1\}, \forall bfu \in BFU, \sum_{\forall G' \in COPE'_n} (\tau(\psi(G'), bfu)) \leqq \lambda(bfu)$

$COPE'$ は選択された複合演算を表す。このうち、ノード数が 2 以上の複合演算はチェイニングが行われたことを意味する。φ は、各複合演算を実行するステップを表す。ψ は、複合演算と複合演算を実行する複合演算器との対応を表す。また、第 1 の条件は、選択された複合演算間でノードの重複がないことを表す。第 2 の条件は、選択されたすべての複合演算のノードで DFG のノードを被覆することを表す。第 3 の条件は、選択した複合演算と複合演算器との対応を満足するための条件である。第 4 の条件は、複合演算間のデータ依存関係を保つための条件である。第 5 の条件は、資源制約を満たすための条件である。

なお、本研究では問題の簡単化のために、$\forall cfu \in CFU$ に対し、$\sigma(cfu) = 1$ とする。

3. 提案手法

3.1 概　　要

本手法は、複合演算と対応する複合演算器の内、特徴的なものを列挙しチェイニングすることで、効率的に効果的にスケジューリングを行う手法である。

本手法の特徴は 2 つある。1 つ目は、複合演算器の共有を積極的に行うことである。複合演算間で複合演算器を共有することで、チェイニングを行うことができる複合演算の数を増やし、全体的なステップ数の削減を図る。

2 つ目は、チェイニング対象となる複合演算を始めに限定することにある。複合演算は連結な部分グラフであるので、複合演算の総数は最悪 $2^{|V|}$ 個となる。よって、ノード数が 20 程度でもすべての複合演算を列挙することが困難である。そこで、チェイニングの効果が高いと予想される複合演算に的を絞り、部分的に複合演算を列挙することで、効率的にチェイニングを行う。

3.2 手　　順

提案手法の手順は図 3 のようになる。

```
(COPE_all, CFU_all, ρ, σ, τ) = enum_cope_cfu(G_d, t_cycle, λ, BFU)
η_best = init_select_cfu(λ, BFU, COPE_all, CFU_all, ρ, σ, τ)
(COPE'_best, φ_best, ψ_best) = schedule(G_d, λ, BFU, COPE_all, CFU_all, ρ, σ, τ, η_best)
CFU = CFU_all
cfu_best = undef
while CFU ≠ φ begin
 RESULT = {(COPE'_best, φ_best, ψ_best, η_best, cfu_best)}
 cfu_select = select_cfu (λ, BFU, COPE_all, CFU, ρ, σ, τ)
 η = add_cfu(λ, BFU, COPE_all, CFU, ρ, σ, τ , η_best, cfu_select)
 CFU = CFU − {cfu_select}
 (COPE', φ, ψ) = schedule(G_d, λ, BFU, COPE_all, CFU_all, ρ, σ, τ, η)
 RESULT = RESULT ∪ {(COPE', φ, ψ, η, cfu_select)}
 if cfu_best ≠ undef then begin
  η = add_cfu(λ, BFU, COPE_all, CFU, ρ, σ, τ , η_best, cfu_best)
  (COPE', φ, ψ) = schedule(G_d, λ, BFU, COPE_all, CFU_all, ρ, σ, τ, η)
  RESULT = RESULT ∪ {(COPE', φ, ψ, η, cfu_best)}
 end
 (COPE'_best, φ_best, ψ_best, η_best, cfu_best) = select_result(RESULT)
 CFU = update(BFU, CFU, ρ, σ, τ, η_best)
end
```

図 3　提案手法の手順

始めの enum_cope_cfu では、DFG から複合演算及び複合演算器を列挙する。単一演算はすべて列挙する。ノード数が 2 以上の複合演算は、対応する複合演算器が制約を満たすものだけ、次の 3 通りの方法で一定の上限値を定めて列挙する。

1 つ目の方法は、ノード数が小さい複合演算から全列挙する方法である。これは、ノード数が少ないほど複合演算器を共有しやすいという予想に基づく方法である。

2 つ目の方法は、あるノードを基点として、インエッジ方向に深さ優先探索的に列挙する方法である。例えば、図 4 の左側のように列挙する。インエッジ側に隣接するノードのうち、基点となるノードから最も遠いノードを 1 つ選択し複合演算に加えることを繰り返す。最も遠いノードが複数存在する場合は、ノードのスラック [1] [2] が小さいものを選択する。基点となるノードもスラックが小さい順に選ぶ。この方法は、ステップ数の削減量が多いものを列挙する方法である。

3 つ目の方法は、あるノードを基点として、インエッジ方向に幅優先探索的に列挙する方法である。例えば、図 4 の右側のように列挙する。インエッジ側に隣接するノードのうち、基点となるノードから最も近いノードを 1 つ選択し複合演算に加えることを繰り返す。最も近いノードが複数存在する場合は、2 つ目の方法と同様にスラックが小さいものを選択する。基点となるノードもスラックが小さいものを選択する。この方法は、方法 1 と方法 2 の中間的な方法である。これらの 3 つの方法による違いは後ほど実験で確かめる。

複合演算を列挙した後、複合演算間で複合演算器を共有でき

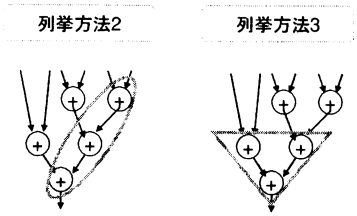

図 4　複合演算の列挙方法 2 及び 3 の例

るか調査する。複合演算間で複合演算器を共有するための必要十分条件は、複合演算の入出力に対応関係が存在し、各出力の論理関数が等しいことである。この複合演算間の関係を代替関係と呼ぶことにする。論理関数が等しいかどうかは Satisfiability (SAT) 問題を解く必要がある。SAT 問題は NP 完全問題であるので、一般的に解くことは難しい。そこで、SAT 問題を解く代りに、ラベル付きグラフの部分グラフ同形問題を解く。例えば、図 5 の場合を考える。図 5 の複合演算 A は複合演算 B と同形な部分グラフを含み、入出力の対応関係が存在する。よって、複合演算 A は複合演算 B を代替可能である。ただし、この条件は十分条件である。部分グラフ同形問題も NP 完全問題であるが、複合演算のノード数を小さくすることで現実的な時間で解くことが可能である。

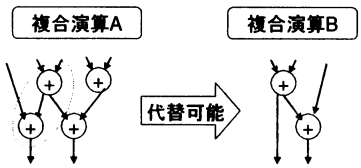

図 5　複合演算間の代替関係

複合演算器を構成する際、複合演算に加算、減算、乗算が含まれる場合は、Carry-Save Adder(CSA) を用いて高速な複合演算器を構成する。これにより、各基本演算器を単純に接続する場合に比べて最大遅延時間を削減し、よりステップ数削減量が大きい複合演算を選択できるようにする。

init_select_cfu では、使用する複合演算器の数を表す関数 $\eta : CFU \mapsto N \cup \{0\}$ を決定し、η_{best} として保持する。ただし、ここでは基本演算器のみを制約を満たす範囲で最大限使用する。基本演算器が 2 つ以上で構成される複合演算器は後ほど 1 つずつ追加する戦略を取る。

最初の schedule では、init_select_cfu の結果である η_{best} に基づき、チェイニングを行わないスケジューリングを行う。結果は、$COPE'_{best}$、φ_{best} ψ_{best} として保持する。

select_cfu では、複合演算器に優先順位をつけ、優先度の高い複合演算器を選択する。選択した複合演算器は後ほど CFU から削除する。優先度は次の 2 つの方法で決定する。

1 つ目の方法は、複合演算器で実行可能 (代替実行含む) な複合演算をチェイニングした場合のステップ数削減量の総和を優先度とする方法である。これは、実行可能な複合演算が多いもの、または、対応する複合演算のステップ数削減量が多いものを選択する方法である。

2 つ目の方法は、対応する複合演算をチェイニングした場合のステップ数削減量を優先度とする方法である。これは、とにかくステップ数削減量が多いものを選択する方法である。

優先度による違いは後ほど実験で確かめる。

add_cfu では、select_cfu で選択した複合演算器を、前回の選択結果である η_{best} に追加し、新たに η を定める。η_{best} で定められていた基本演算器の数は、追加した複合演算器に含まれる基本演算器の数だけ削減する。

繰り返し内の schedule では、η に基づきスケジューリング行う。スケジューリングアルゴリズムはリストスケジューリングをベースとしたものを用いる。スケジュールの単位は複合演算とし、リストの優先度は複合演算をチェイニングした際のステップ数削減量とする。ステップ数削減量が同じ場合は、複合演算内の基本演算のスラックの最小値が低いものを優先する。

select_result では、最もステップ数が小さくなった結果を選択する。前回の繰り返しで選択された結果に含まれる複合演算器 cfu_{best} が未定義 (undef) でなければ、select_result を行う前に、η_{best} に cfu_{best} を追加してスケジューリング行う。select_result では、前回の結果、cfu_{select} を追加した結果、cfu_{best} を追加した結果を比較し、最も良い結果を選択する。

update では、残りの基本演算器では実現できない複合演算器を CFU から削除する。

以上の処理を、CFU が空になるまで繰り返す。

4. 実　　験

4.1 目　　的

実験の目的は 2 つある。1 つ目は、チェイニングを用いることで実際にどれほどの効果が得られるかを確かめることである。2 つ目は、提案手法において複合演算の列挙方法と複合演算器の優先度を複数提案しているため、これらの違いによる結果の差を確かめることである。

4.2 合 成 対 象

合成対象には、HLSynth'92 [5] の 5 次の楕円フィルタと微分器、Mediabench [6] の ADPCM のデコーダの記述から変換した DFG を用いた。5 次の楕円フィルタには制御構造が含まれないため、そのまま DFG に変換した。微分器、ADPCM のデコーダにはメインのループが存在するため、ループの内部を DFG に変換した。5 次の楕円フィルタでは、26 回の加算を行う。データ型は 16 ビットの整数である。微分器では、6 回の乗算、2 回の加算、2 回の減算、1 回の大小比較を行う。データ型は 32 ビットの整数である。ADPCM のデコーダでは、5 回の大小比較、7 回の加算、2 回の減算、6 回の等価比較、4 回の右シフト、10 回の選択、7 回のビット積を行う。データ型は 32 ビットの整数である。

4.3 手　　順

提案手法を用い、制約条件である動作周期及び基本演算器の数を変化させながらスケジューリングを行った。基本演算器の数については、DFG において支配的な基本演算に対応する基

本演算器の数を変化させた。具体的には、5次の楕円フィルタでは加算器を、微分器では乗算器を、ADPCMのデコーダでは加算器の数を変化させた。それ以外の基本演算器の数は、一定の資源制約がかかるとみなし、DFG内の基本演算のノード数の半分とした。具体的には、微分器の場合、加算器1個、減算器1個、大小比較器1個とした。ADPCMでは、減算器1個、大小比較器2個、右バレルシフタ2個、等価比較器3個、ビットAND 3個、マルチプレクサ5個とした。

複合演算の列挙方法が3通り、複合演算器の優先度が2通り、計6通りの組合せで実験を行った。なお、列挙する複合演算の上限値は、5次の楕円フィルタと微分器で100、ADPCMのデコーダで50とした。

4.4 環　　境

提案手法はC++言語で実装した。コンパイラはGNU Compiler Collection Version 3.3.3を用いた。作成したプログラムをIntel社のXeon 3.2GHzを搭載したパソコン上で実行させ実験を行った。

使用する基本演算器の遅延情報は、VDECで提供されている、Synopsys社のDesign Compiler(Version 2003.06-SP1)、日立0.18μmプロセステクノロジ用の京都大学作成のセルライブラリを用い、あらかじめ速度優先で論理合成を行って得た。

4.5 結　　果

図6及び図7は5次の楕円フィルタの結果を表している。図6は、動作周期を2.6[ns]に固定して加算器の数を変化させた場合の結果である。なお、16ビットの加算器の最大遅延時間は0.86[ns]である。楕円フィルタは加算しか行わないため、チェイニングの効果は高いと予想される。実際、チェイニングを行わない場合と比べると、6通りの方法はどれもチェイニングの効果が出ていることがわかる。特に効果が高かったのは、複合演算の列挙方法が2または3と優先度が1の組合せである。ところが、優先度が2の場合、列挙方法によらず、加算器の数が増えると逆に総ステップ数が増加するという逆転現象が見られる。図7は、加算器の数を7個に固定して動作周期を変化させた場合の結果である。動作周期が3.0[ns]以下は列挙方法3と優先度1の組合せが、3.0[ns]以上では列挙方法1と優先度1の組合せが最も良い結果となった。しかし、列挙方法1と優先度1組合せ以外はすべて、動作周期が増えるにつれ総ステップ数が増加する逆転現象が見られる。これらの逆転現象については後ほど考察する。

図8と図9は微分器の結果である。微分器は列挙方法及び優先度によらず、すべて同じ結果となった。図8は、動作周期を8.0[ns]に固定して乗算器の数を変化させた場合の結果である。なお、32ビットの演算器の最大遅延時間は、乗算器が3.47[ns]、加算器が0.99[ns]、減算器が0.94[ns]、大小比較器が0.60[ns]である。図8では乗算器3個以上でチェイニングの効果が表れている。微分器は演算の数が少ないので、手作業で最適解に近いであろう近似解を求めた。その結果、提案手法による結果と一致した。図9は、乗算器の数を4個に固定し動作周期を変化させた場合の結果である。こちらも、チェイニングの効果が表れている。楕円フィルタのような逆転現象は起きていない。総じて良い結果が得られている。

図10と図11はADPCMのデコーダの結果である。図10は、動作周期を4.0[ns]に固定して加算器の数を変化させた場合の結果である。なお、32ビットの演算器の最大遅延時間は、加算器が0.99[ns]、減算器が0.94[ns]、大小比較器が0.60[ns]、等価比較器が0.63[ns]、2入力マルチプレクサが0.21[ns]、右バレルシフタが0.85[ns]、AND論理ゲートが0.08[ns]である。ADPCMのデコーダは多くの種類の演算が複雑な依存関係にあるため、チェイニングの効果はそれほど大きくないと予想される。しかし、図10を見ると、楕円フィルタほどではないが、チェイニングの効果が出ており、効果も小さくはない。列挙方法2と優先度2の組合せが最も良くなっているが、列挙方法及び優先度の違いによる差は小さい。図11は、加算器の数を4個に固定して動作周期を変化させた場合の結果である。列挙方法1または列挙方法2と優先度1の組合せでは、動作周期によらずステップ数は一定である。しかし、それ以外の組合せでは、楕円フィルタと同様に逆転現象が見られる。

楕円フィルタ及びADPCMのデコーダで見られた逆転現象を解析したところ、原因は共有した複合演算器の使用効率が悪いことであることが判明した。例えば、加算器を4個使用する複合演算器があるとする。対応する複合演算または同一パターンの複合演算を実行する場合は、4個の加算器をすべて使用する。しかし、代替関係にある複合演算の内、ノード数が少ないもの、例えば加算が1つの単一演算を実行する場合は、3個の加算器が使用されないことになる。よって、極端な場合、単一演算を大量に実行し、対応する複合演算を1回しか実行しなければ、3個の加算器はほとんど使用されないことになる。列挙方法が1の場合は、列挙される複合演算のノード数が比較的少ないため、逆転現象はほとんど見られない。しかし、列挙方法2または3では、制約を満たす限り、ノード数が多い複合演算を列挙する。そのため、列挙方法1に比べて逆転現象が起こりやすい。また、優先度1は共有を積極的に行う。そのため、列挙された複合演算のノード数が多い場合は、逆転現象が起こってしまう。また、優先度2は、対応する複合演算のステップ数削減量が大きい複合演算器を選択する。ステップ数削減量が大きい複合演算は、ノード数も比較的多いため、合成対象によっては方法1よりも逆転現象がひどくなる場合がある。逆転現象を防ぐためには、優先度を変更する、またはスケジューリングにおいて複合演算器の共有に制限を加えるといった改良が必要である。

5. おわりに

本稿では、チェイニングを考慮した動作合成手法を提案した。実験では、チェイニングの効果が高いことを確認した。しかしながら、提案手法では複合演算器の使用効率を考慮していないため、動作周期または演算器の数が増えると総ステップ数が増えるという逆転現象が見られた。逆転現象を防ぐために、複合演算器の優先度、もしくはスケジューリングアルゴリズムを改良することが今後の課題である。また、スケジューリングアルゴリズムはヒューリスティックなアルゴリズムであるリストス

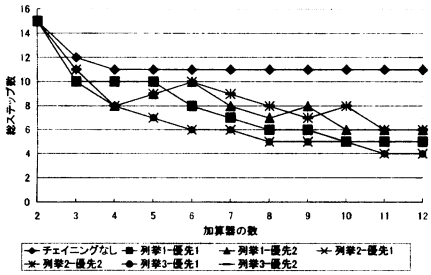

図 6　5 次の楕円フィルタの結果 (動作周期 2.6[ns])

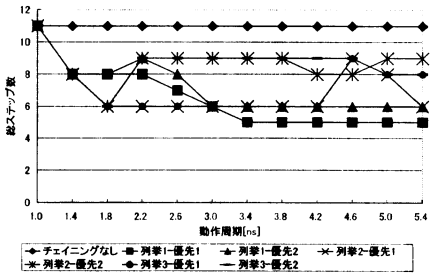

図 7　5 次の楕円フィルタの結果 (加算器 7 個)

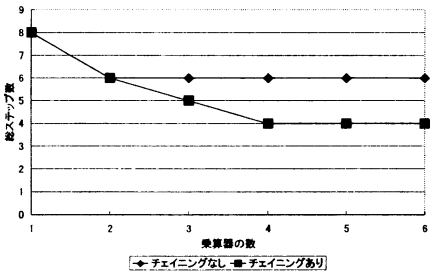

図 8　微分器の結果 (動作周期 8.0[ns])

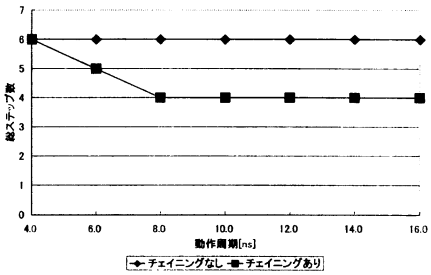

図 9　微分器の結果 (乗算器 4 個)

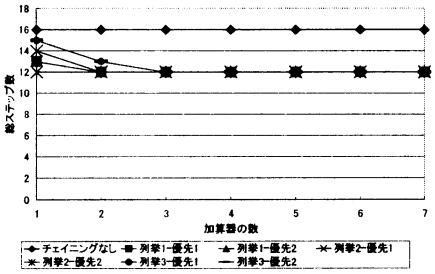

図 10　ADPCM のデコーダの結果 (動作周期 4.0[ns])

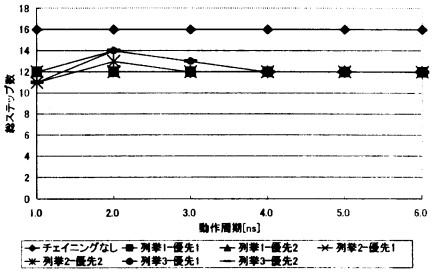

図 11　ADPCM のデコーダの結果 (加算器 4 個)

ケジューリングをベースにしている。そのため、選択される複合演算器の組合せによっては、チェイニングを行う複合演算の選択を誤り、解が最適解から大きくはずれた極小解になってしまっている可能性がある。よって、スケジューリングアルゴリズムを評価し、他のアルゴリズムを検討することも今後の課題である。

謝辞

本研究の一部は、福岡地域の文部科学省知的クラスター創成事業の支援による。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、シノプシス株式会社の協力で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] Daniel Gajski, Allen Wu, Nikil Dutt, Steve Lin, "HIGH-LEVEL SYNTHESIS", Kluwer Academic Publishers, 1992.

[2] Giovanni De Micheli, "SYMTHESIS AND OPTIMIZATION OF DEIGITAL CIRCUITS", McGraw-Hill, 1994.

[3] J. Nestor and D. Thomas, "Behavioral Synthesis with Interfaces", Proceedings of the International Conference on Computer Aided Design, pp. 112-115, 1986.

[4] Pierre G. Paulin and John P. Knight, "Scheduling and Binding Algorithms for High-Level Synthesis", Proceedings of the 26th ACM/IEEE Design Automation Conference, pp. 1-6, 1989.

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

[6] C. Lee, M. Potkonjak, and W. H. Mangione-Smith "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communicatons Systems", International Symposium on Microarchitecture, pp. 330-335, Dec 1997.

[7] Anatoly Prihozhy, "Net Scheduling in High-Level Synthesis", IEEE Design & Test, vol. 13 no. 1, pp. 26-35, March 1996.