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

2009-SLDM-138 (27)
2009 / 1 / 30

DSP 自動合成を指向したCDFGからの高速な複合演算抽出手法

加藤 俊之† 三宅 貴章† 大亦 真一† 西門 秀人† 山内 寛紀† 小林 士朗††

†立命館大学理工学部 〒525-0123 滋賀県草津市野路東 1-1-1
††旭化成株式会社研究開発本部 〒243-0021 神奈川県厚木市岡田 3050
E-mail: †{ri008042}@se.ritsumei.ac.jp

あらまし 本稿では最右拡張を用いたCDFGからの複合演算抽出手法を提案する．DSPの積和演算器に代表されるように，特定用途向けのLSIには高性能化を狙った専用回路が搭載される事が一般的，専用回路化すべき演算パターンを効率的に求める事ができる．我々はCDFGを用いたC言語からのLSI自動生成システムの研究を進めており，本手法を用いる事で自動的に高性能・高効率なLSIの生成が可能となる．

キーワード CDFG, 最右拡張 , DSP, High-Level syntheses

Combine operation pattern extraction from CDFG for DSP generation

Toshiyuki Kato† Takaaki Miyake† Shinichi Oomata† Hieto Nishikado†
Hironori Yamauchi† Shiro Kobayashi††

† College of science and engineering, Ritsumeikan University 1-1-1 Nojihigassi, Kusatsu-shi, Shiga, 525-0123 Japan
†† Central R&D Administration, Asahikasei 3050 Okada, Atsugi-shi,Kanagawa 243-0021 Japan
E-mail: †{ri008042}@se.ritsumei.ac.jp

Abstract In this paper, we propose a method to extract frequent operation patterns from the CDFG used to Most Right Expansion. Represented for accumulator in DSP, almost specific LSI has exclusive calculator for high performance. To use the method, you can find combined operation pattern that you have to make exclusive calculator easily. Our group is working on development of DSP automatic generation systems. Therefore, we developed a system to generate CDFG based on C language scripts, and to extract frequent operation patterns from CDFG.

Keyword CDFG, Most Right Expansion, DSP, High-Level syntheses

1. はじめに

現在，携帯電話をはじめとする組み込みシステムの発展が著しい．各製品の多機能化が進み開発サイクルも短くなっている．そのためLSIの開発負担の増加が問題となり,抽象度の高い LSI 開発手法の研究が盛んとなってきている．現在では，C言語の動作記述から回路を自動生成する動作合成が開発現場で用いられる様になってきている．しかし，既存の手法[1]ではHW処理部とSW処理部とを人が指定しなければならないためSoCやDSP(Digital Signal Processor)等のLSIの開発に用いる事は出来ない．現在，本研究室では特定用途向けであるプロセッサ,DSP を自動合成のターゲットとしてHW自動合成システムの研究を進めている．

本研究室の提案する DSP 自動合成システムをはじめ，HW 自動生成手法にはCDFGを用いる手法が多く存在する[2][3]．しかし，特殊演算器等のHW処理は人が指定しなければならない．そこで今回CDFGから頻出パターンを発見し頻出演算パターンを求める手法を提案する．この手法を用いる事で自動的に複合演算を求める事ができ HW/SW 分割手法に用いる事ができる．2.で本研究室の提案するDSP自動合成システムの概要を述べる．3.でCDFGについての説明を行い，本手法で用いる最右拡張について 4.で紹介する．そして5.にて本手法の説明を行い，6.でハードウェア生成部を説明し、7.で実装結果を紹介する．

2. DSP 自動合成システム

提案する DSP 自動合成システムは DSP ハードウェ

アの設計期間の短縮を目的としたものである．設計者は所望する機能が記述されたC言語記述を当システムに与える．そうすれば，本システムが記述されたアルゴリズムに最適なDSPのハードウェアの構成と実行コードを出力する．

図1にDSP自動合成システムの処理フローを示す．当システムは，大きく分けてCDFG生成部とCDFG最適化部、ハードウェア生成部の3つからなる．CDFG生成部では，入力されたC言語記述を解析し、構文解析木を作る。次に,この構文解析木の依存解析を行う．その結果からCDFGを生成する。CDFG最適化部では、CDFGから性能向上が多く見込める演算パターンを発見し複合演算を抽出する。この複合演算の情報はプロセッサ内の演算器を生成するときに必要となる。そして、ハードウェア生成部ではCDFGと複合演算の情報を元にハードウェア構成を決定する.出力はHDLとする。また、そのプロセッサ上で実行するための実行コードも同時に生成する。

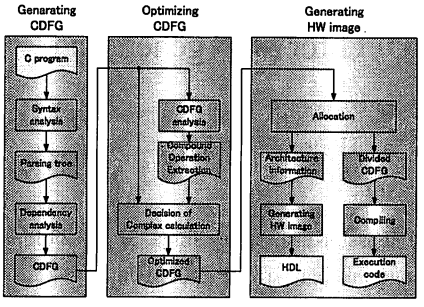

図 1 DSP自動合成システムの処理フロー

3. CDFG

3.1. CDFG

DFG(Data Flow Graph)とはデータの流れを表したグラフの事をいう．DFGの例を図2に示す．円で囲まれた演算子は演算処理を表し，各円を繋ぐ線はデータの流れを表している．このDFGはA=(B+C)+Dを表している．CDFGとは，このDFGに,分岐や繰り返し等の制御情報を加えたグラフの事をいう．一般的なCDFGを図3に示す．ここでの三角形は分岐処理を表しており，制御変数'x'の値によって異なる演算処理が行われる．このCDFGは'x'が0ならa=(b-d)+c; 'x'が1ならa=b+d; の処理が行われる事を示している．CDFGには特定の書式は決まっておらず多くの種類のCDFGが提案されている．本研究室でもDSP自動合成に適した独自のCDFGを提案しており，次項に本研究室の提案するCDFGの説明を行う.今回の複合演算抽出手法では，このCDFGを用いているが，本稿の提案手法はCDFG全てに用いることが可能である．

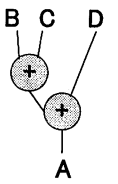

図 2 DFG

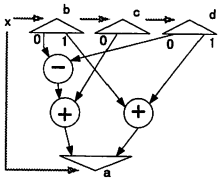

図 3 CDFG

3.2. 提案するCDFG

本研究室で提案するCDFGの例を図4に示す．図4における四角の枠は階層を表している．C言語ではIf文やfor文などの制御処理は入れ子状に幾つでも重ねる事が可能である．(いくらでもネストが可能)そこで,このCDFGでは制御を階層構造を用いて表している．これによりCDFGとC言語記述との対応関係が理解しやすいだけでなくマクロからミクロの可変の大きさで並列性を抽出することができる．各階層の右上に四角い枠に囲まれた表記がある．各階層には処理の種類によってラベルを付けている．関数全体はFB(function block).分岐処理はSLB(selection block).ループ処理部はLB（loop block）及びループ毎の処理をITB(iteration block).演算部をSB(statement block)としている.LBの右上にはラベル名と同時にループ条件を表記する．演算処理を表すSBでは，演算子を表すノードに円と五角形が存在している．これはアドレス演算部とデータ演算部のCDFGを区別するためである．五角形のノードで表現された部分はアドレス演算部となっている．丸いノードで表現された演算はデータ演算である．アドレス演算部とデータ演算部を分離しているため処理の流れを理解しやすい．またDSPの様にアドレス計算器を持つアーキテクチャに対しても最適な情報を得ることが可能となっている．

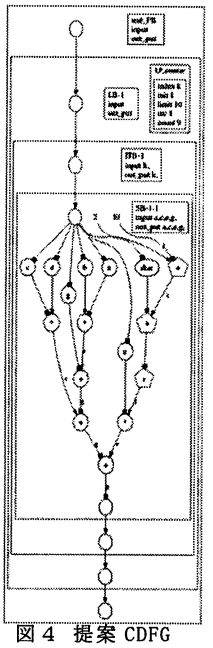

図 4 提案CDFG

本研究室ではこのCDFGの自動描画システムの開発を行っている．そのためC言語を入力として図4の様なCDFGを自動的に描画することができる．

4. 最右拡張

最右拡張とは効率のよい順序木枚挙法である。デー

タベースに蓄積された大量のデータから，自明でない規則性やパターンを半自動的にとりだす方法についての科学的研究であるデータマイニング手法の一つである。以下、最右拡張について説明する。ここで，順序木Sの最右拡張とは、Sの最右枝(根節点から最右葉に至る経路)に節点kを追加して得られる順序木Tである.ただし、kはTの兄弟関係において末弟であるように追加する。特に，Sの最右葉(最も右に位置する葉節点)rml(S)=k-1のp代前の親π(k-1)に、ラベルlをもつ節点kを追加して得られたSの最右拡張を、(p,l)拡張と呼び（図5），具体例を図6に示す。また、サイズ0のパターン空木を仮定し、任意の1-パターンは空木の最右拡張とする。

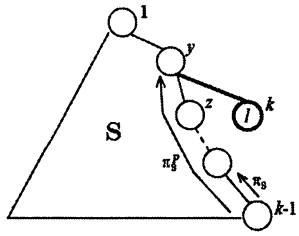

図 5 順序木 S の(p,l)拡張

最右拡張を繰り返すことにより，すべてのパターンを重複なく生成することができる．この順序木枚挙手法は，Bayard による集合枚挙木に基づくアイテム集合枚挙法を，順序木に拡張したものである．

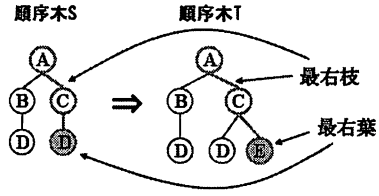

図 6 最右拡張の具体例

5. 頻出パターン抽出手法

頻出演算用に専用演算器を持つためには，頻出演算パターンを知る必要がある．本提案手法では，最右拡張を用いて，頻出パターンを抽出する．前述の通り最右拡張は効率のよい順序木枚挙法であるため,本手法では最右拡張を CDFG 向けに拡張して用いている．CDFG を最右拡張することで CDFG 上に存在する演算のパターン全てが羅列されるのだ．

5.1.1. 演算部(Statement Block)

CDFG は演算部と制御部に分かれるが,CDFG 上の制御部には演算は無い．そのため，まずは演算部（SB）における CDFG を最右拡張し、パターンを生成する．

最右拡張は最初に CDFG のデータ木を巡回しながら、最右拡張の対象となる CDFG のラベルの出現数と出現位置を調べる。CDFG にはデータを読み書きする r,w などの様々なラベルがあるので、今回は＋，－，＊，／，％の 5 つの演算のラベルに限定する。つまり、この 5 つの演算を組み合わせた複合演算を抽出する。次に、抽出したそれぞれの CDFG に対して最右拡張を行い、拡張した CDFG のみを新たな演算パターンとして保存する。これは、パターンとデータ木のマッチング情報を効率よく保存している。あるパターンがもつ全ノードの位置情報を保存するのではなく、パターンの最右葉 k=rml(T)の CDFG 情報だけを保存し、計算している。図 7 に演算部分のラベル＋をもつ CDFG を最右拡張した順序木枚挙グラフを示す。同じように－，＊，／，％についても同じように拡張し図示できる。ただし、拡張し存在したパターンだけをグラフとして枚挙する．

5.1.2. 制御部

制御部にはループや分岐がある．ループ処理ではループ回数を CDFG が保持しているため，ループ回数を出現回数に乗算する．分岐処理では各分岐の分岐数に応じて出現回数を除算する．

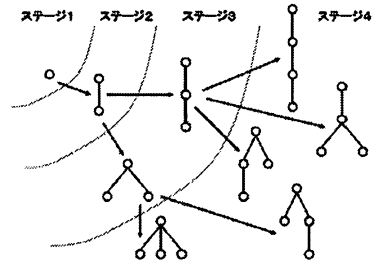

図 7 CDFG を最右拡張した順序木枚挙グラフ

5.2. 複合演算決定

最右拡張を用いて頻出パターンを知る事ができた．この結果から複合演算を決定する．順序木枚挙グラフには演算パターンと演算パターンの存在回数が出ているので，ここから「百回以上存在するステージ 3 の演算のパターン」や「ステージ 3 演算の最も多いパターン」などの条件を与える事で簡単に求める事ができる．

6. ハードウェア生成部

入力した C アプリケーションプログラムに対して生成される RISC プロセッサのイメージを図 2-5 に示す．ハードウェア生成部では解析部で生成された CDFG および,最適化部で抽出された複合演算情報を元に RISC プロセッサに専用演算器を搭載して生成する.そして，ハードウェア部で生成された HDL 記述を最終的に 1 つの LSI にする．

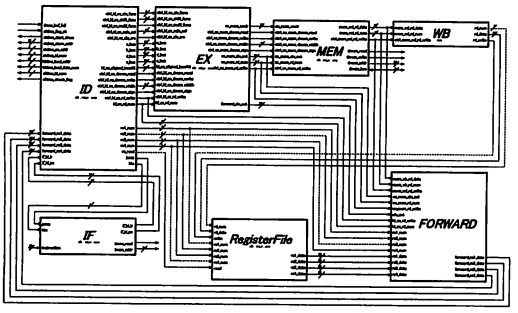

図 8 RISC プロセッサイメージ

6.1.1. HW 生成部との連携

本章では，頻出演算抽出手法を用いて求めた複合演算パターンを RISC プロセッサ自動合成における HW 生成部へ渡すインターフェースについて説明する．

HW 生成部では複合演算情報を受け取り専用演算器が生成されると共に，HW 全体が生成される．複合演算抽出部から HW 生成部には「複合演算情報」のデータが渡される．

図 9 に「複合演算情報」の例を示す．この図の左に示した CDFG パターンが複合演算として定めた際には，右下図の様な出力ファイルが複合演算情報として HW 生成部に渡される．出力ファイルの先頭には 600 という数字がある．この数字は複合演算の番号を表している．複合演算は複数設定する事が可能であるため複合演算情報にも固有の番号を振る必要があるのである．本システムでは 600～650 番を複合演算用の演算子固有番号として定めているため，ここでは 600 番が使われている．そして，その横に"AB+CDE＊－－"と文字列が描かれている．これが 600 番の複合演算を表している．これは逆ポーランド記法を用いて左上の CDFG を表したものとなっている．次の行に 601 番の複合演算として右上の CDFG が逆ポーランド記法を用いて表されている．

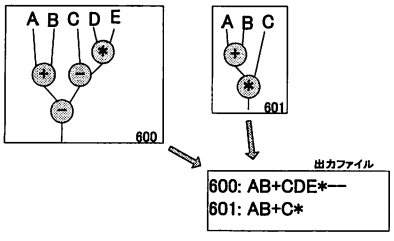

図 9 HW 生成部へ渡す「複合演算情報」の例

6.1.2. プロセッサ IP の概要

DLXMC は 32bit-RISC プロセッサをベースとし，命令フェッチ(IF)，命令デコード(ID)，実行(EX)，メモリアクセス(MEM)，ライトバック(WB)からなる 5 段パイプライン構成となっている．また IMEM と DMEM の分離(ハーバードアーキテクチャの採用)，およびフォワーディング機能の搭載によってパイプライン・ハザードの解消を行っている．

アーキテクチャの特徴として典型的なロード／ストアがある．ロード／ストア・アーキテクチャでは，メモリアクセスは単体のロード／ストア命令のみで行う．全ての演算命令では，演算対象のデータはプロセッサ内部の汎用レジスタ（あるいは即値）からのみ参照して演算器(ALU)の入力とし，また，演算器からの演算結果も同様に，プロセッサ内部の汎用レジスタに格納され，メモリに直接アクセスすることはない．これによって全ての演算はレジスタ-レジスタ演算となるため，1 クロック内での演算が可能である．

6.1.3. プロセッサ IP のデータパス

DLXMC のデータパス構成を図 4-2 に示す．DLXMC は 3 本のプロセッサバスである Source バス 2 本(S1,S2)，Destination バス 1 本(D)と，それらに ALU(Arithmetic and Logic Unit)，汎用レジスタ，特殊レジスタが接続されている．汎用レジスタは 32 個(r0～r31)からなる．そのうち，r0 は常に 0 を保持し，r31 はジャンプアンドリンク命令のリターンアドレスの格納用に予約されている．特殊レジスタには，プログラムカウンタ(PC)，命令レジスタ(IR)，割り込みアドレスレジスタ(IAR)，メモリアドレスレジスタ(MAR)，メモリデータレジスタ(MDR)，テンポラリレジスタ(Temp)がある．これらはすべて 32bit で統一されている．

また，DLXMC に接続するメモリは，命令メモリとデータメモリの 2 つが存在する．名前の通り，命令メモリとは命令（プログラム）を格納する専用のメモリであり，データメモリとは被演算データとその結果データ，およびプロセッサ間通信でやりとりするデータなどを格納するためのデータ専用のメモリである．この 2 つのメモリでの運用は，パイプライン動作時における資源競合（構造ハザード）を簡単に防ぐ効果もある．命令メモリ，データメモリともにワード・アドレッシングによる 32bit アドレス空間をもつ．

7. 結果

今回，C 言語記述を入力として，CDFG からシステムの試作を行った．本節ではその結果を紹介する．今回，評価として入力した C 言語記述は DCT（離散コサイン変換）である．DCT は最も有名な信号処理の一つであるために適していると考えた．入力した DCT の記述を以下の図 10 に示す．

```
void main(void)
{

double cos_table[32]={1.0, 0.9807853, 0.92387953, 0.831469612, 0.70710678,
                      0.555570233, 0.3826834, 0.1950903, 0.0,
                      -0.1950903, -0.3826834, -0.555570233, -0.70710678,
                      -0.831469612, -0.92387953, -0.9807853, -1.0,
                      -0.9807853, -0.92387953, -0.831469612, -0.70710678,
                      -0.555570233, -0.3826834, -0.1950903, 0.0,
                      0.1950903, 0.3826834, 0.555570233, 0.70710678,
                      0.831469612, 0.92387953, 0.9807853};

double cv[7]={0.70710678,1.0,1.0,1.0,1.0,1.0,1.0};
double cu[7]={0.70710678,1.0,1.0,1.0,1.0,1.0,1.0};
double dct_after[10][10];
double dct_before[10][10];

 int x,y,u,v;
 double sum;
 for(v=0; v<8; v=v+1){
        for(u=0; u<8; u=u+1){
   sum=0;
   for(y=0; y<8; y=y+1){
                for(x=0; x<8; x=x+1){
                        sum =sum + dct_before[y][x] * cos_table[((2*x+1)*u)%32] *
cos_table[((2*y+1)*v)%32];
                }
   }
        dct_after[v][u] = sum*cu[u]*cv[v]/4;

  }
 }
}
```

図 10 DCT 記述

7.1. CDFG

図 11 に CDFG の生成結果を示す．今回は一部しか掲載していないが,全体像での生成を行う事ができた．

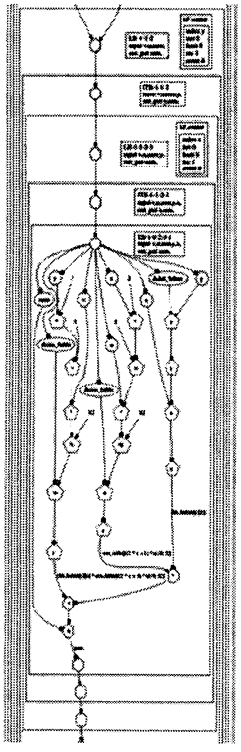

図 11 DCT の CDFG

階層も表現されている．データ演算とアドレス演算との表現を変えてはいるが，理解が難しい事が解った．理解しやすいグラフに改良が必要と考えられる．しかしながら，このグラフは graphviz[6]というツールを用いて描いているため,この graphviz の性能に依存しており，多少入り組んだグラフになることは仕様となる．

このプログラムから生成した CDFG データ木から最右拡張により抽出したパターンを図 9 に示す。テキスト文書でパターンの型は PATTERN NUMBER、頻出回数出力は PATTERN COUNT で示される。また、その隣に演算パターンのラベルが示され、右から左へ拡張した順に羅列されている。得られた結果から頻出の高い演算パターンを選び、図 11 中の複数の演算ノードを 1 つの複合ノードに置き換える。つまり、複合演算処理をするアセンブラコードを生成し、1 つの複合命令として処理できる。

```
PATTERN NUMBER =101 : PATTERN COUNT =28992 : EXPAND IDENTIFIER =+
PATTERN NUMBER =102 : PATTERN COUNT =0 : EXPAND IDENTIFIER =
PATTERN NUMBER =103 : PATTERN COUNT =24704 : EXPAND IDENTIFIER =*
PATTERN NUMBER =104 : PATTERN COUNT =64 : EXPAND IDENTIFIER =/
PATTERN NUMBER =105 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =%
PATTERN NUMBER =206 : PATTERN COUNT =12288 : EXPAND IDENTIFIER =*+
PATTERN NUMBER =207 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =%+
PATTERN NUMBER =208 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =**
PATTERN NUMBER =209 : PATTERN COUNT =4160 : EXPAND IDENTIFIER =**
PATTERN NUMBER =210 : PATTERN COUNT =64 : EXPAND IDENTIFIER =*/
PATTERN NUMBER =211 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =*%
PATTERN NUMBER =312 : PATTERN COUNT =4096 : EXPAND IDENTIFIER =**+
PATTERN NUMBER =313 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =*%+
PATTERN NUMBER =314 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =**
PATTERN NUMBER =315 : PATTERN COUNT =64 : EXPAND IDENTIFIER =**/
PATTERN NUMBER =316 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =**%
PATTERN NUMBER =417 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =**%+
PATTERN NUMBER =418 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =**+%
PATTERN NUMBER =519 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =**+%+

Max count pattern When 3 stage
MAX PATTERN NUMBER =313 : PATTERN COUNT =8192 : EXPAND IDENTIFIER =*%+
```

図 12 演算パターン抽出結果

7.2. 複合演算器搭載プロセッサ IP

出力されるプロセッサの RTL 記述は，6.で述べたプロセッサ IP のコアに複合演算器を追加したものである(図 8)．追加した複合演算は図で示された PATTERN NUMBER=313 である。ノードが 3 つの演算の内（実行時にノード数を指定し、最大頻出のパターンを演算気対象パターンとする)、最大の頻出をとるため複合演算器はこのパターンを搭載する。

ALU と同列に存在し，排他的に動作してターゲットとするプログラムに頻出する 4 入力 1 出力の演算を 1 クロックで行なう．4 入力 1 出力の演算は，順に計算していく直列型と，2 つの独立した演算の結果を用いて演算を行なう並列型の 2 種類のタイプがあり(図 6.3)，複合演算器の構成は行なう演算の種類と演算の順序の情報の組み合わせにより決定される．

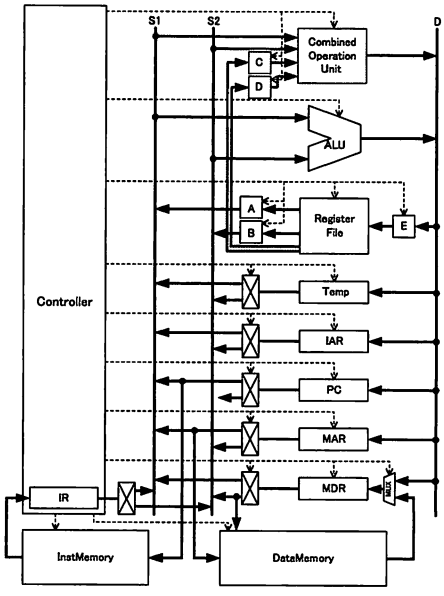

図 13 複合演算器が搭載されたプロセッサ

IP コアのブロック図

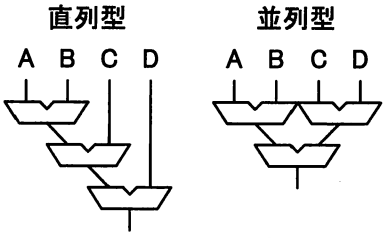

図 14 入力 1 出力の複合演算器

複合演算器を搭載したプロセッサーで2次元DCT（図8）を実行したところ、実行サイクル数は45568サイクルとなった。複合演算器を搭載しないプロセッサーで実行すると、61952サイクルかかるので16384サイクルを削減することができた。よって、複合演算器を自動合成し追加するとクロック削減に有効であることが明らかになった。

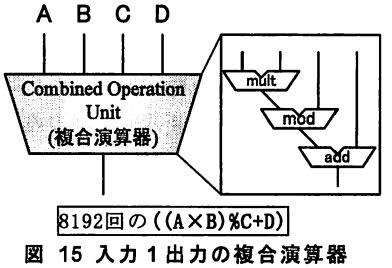

図 15 入力 1 出力の複合演算器

文　献

8. まとめ

本稿では，CDFG からの最右拡張を用いた複合演算抽出手法の提案を行った．C 言語などの高位言語からの LSI 自動設計システムでは，一度高位言語を CDFG に変換して，そこから LSI を設計していく手法が多く提案されている．そのため CDFG の中から頻出演算パターンを自動的に求める手法があれば，専用演算器の指定や，HW/SW コデザインにおける HW/SW 処理の分離に用いる事が可能であると考えている．本手法では XML 文書などにしか使われていなかった最右拡張を CDFG という半構造データの領域に持ち込み，非常に効率的に頻出パターンの抽出を行っている．

今後は，最右拡張からの複合演算抽出を行っていく．出現回数は知る事ができたが各演算の重み付け等を行っていないので，正確にどの演算パターンを HW 化すれば最も効率が良いか知る事ができない．また，どれくらいの性能向上が見込めるかの指標も求めて行きたいと考えている．

複合演算器生成に関しては 4 入力 1 出力の演算パターンのみに限定し、その有効性を検証した。今後は抽出したどのパターンにも対応できる複合演算器生成をめざし、そのデータ入力がロード/ストアによってクリティカルパスにならないようなアーキテクチャ構成を自動的に合成できるようにしていきたい。

文　献

[1] D. Gajski, A. Wu, N. Dutt, and S. Lin, “High-level Synthesis: Introduction to Chip and System Design.”, Kluwer Academic Publishers, 1992.

[2] 川田容子,戸川望,佐藤政生,大附辰夫,“動作記述からのデータフローグラフ生成手法,”電子情報通信学会技術研究報告，VLD，vol95(307)，pp55-62，Nove.1995.

[3] 浅井達哉，有村博紀，“半構造データマイニングにおけるパターン発見技法”，電子情報通信学会論文誌，vol.J87-D1，no.2，pp.79-96,2004

[4] 中西恒夫，中野猛，福田晃，“辞書式コード圧縮支援機構の遺伝的アルゴリズムによる最適化，”電子情報通信学会技術研究報告，ARC，Vol.2001(39)，pp.19-24,(2005).

[5] H. Arimura, S. Arikawa, S. Shimozono, Effcient discovery of optimal word-association patterns in large Text databases, New Generation Computing, 18, pp.-49-60, 2000

[6] 西口健一,石浦菜岐佐,西村啓成，神原弘之，冨山宏之，高務祐哲，小谷学，“ソフトウェア互換ハードウェアを合成する高位合成システム CCAP における変数と関数の扱い,”電子情報通信学会技術研究報告，ICD，Vol.105(446)，pp.19-24(2005)

[7] 房延慎二，浅井達哉，有村博紀，宇野毅明，中野眞一，“半構造データマイニングのための高速な無順序木パターン発見手法”，電子情報通信学会 DE 研究会第 15 回データ工学ワークショップ(DEWS2004)，6-A-03，2004

[8] http://graphviz.org