設 計 自 動 化　51－9
（1990．2．20）

複数のアルゴリズムを実行する専用プロセッサのアーキテクチャ設計

池永 剛　　白井 克彦
早稲田大学　理工学部

LSI技術の進歩により、様々な専用LSIの実現が可能になってきた。その一方でLSIのシステム設計にかかる時間とコストが増大しており、それらを容易に設計可能な支援システムの実現が望まれている。本論文では、高級言語によるアルゴリズム記述を入力とすることにより、幅広い分野のユーザに対してプロセッサ設計環境を与えることを目的とした、専用プロセッサ設計支援システムについて述べる。そして、このシステムを用いて、複数のアルゴリズムを実行する専用プロセッサを設計し、その評価を行なう。複数の信号処理アルゴリズムを実行する専用プロセッサを設計し、既存のDSPと比較した結果、ソフトウェア開発環境に優れたプロセッサ設計が可能なことが示された。

Architecture Design of Special Purpose Processor Executing Multiple Algorithms

Takeshi Ikenaga and Katsuhiko Shirai
Department of Electrcal Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169, Japan

Progress of LSI technology enables to realize various special purpose LSIs. On the other hand, as the time and the cost of LSI system design become increasing, an effective desgin system is required. This paper describes a special purpose processor design system, realizing processor design environment useable in wide variety of fields, adopting algorithm description written by higher level language. Design and evaluation of the special purpose processor executing multiple algorithms are also discussed. Comparing the designed processor which executies multiple signal processing algorithm, with a representative DSP, it is shown that this system can design the processor having a superior software development environment.

1 はじめに

LSI技術の進歩に伴い、その規模は年々増加し、応用分野も急激に広がりつつある。このため、信号処理プロセッサ[1]のような様々な専用プロセッサが製作され、実用に供されるようになってきている。

その一方で、これらのLSIは複雑化、多様化し、そのシステム設計に多くの時間とコストが必要となってきている。今後、特定用途向けLSIの需要はますます高まっていくと考えられるので、それらを容易に設計可能な環境を実現することが重要になってくると考えられる。この実現の一つの形態として、ユーザ自身の仕様記述に基づくLSI設計支援システムが考えられる。このためには、ユーザが通常用いているソフトウェアの概念に近い、高位の入力記述を持ったシステムが望まれる。

我々は、このような背景をもとに、幅広い分野のユーザを対象とした、専用プロセッサ設計支援システムを提案し[2]、研究を行なっている。このシステムは、通常の高級言語で書かれたアルゴリズム記述を入力とし、そのアルゴリズムを効率よく実行するマイクロプロセッサの設計支援を行なうことを目的としている。

本論文では、専用プロセッサ設計支援システムの主眼点、処理概要について説明する。そして、このシステムを用いて、複数のアルゴリズムを実行する専用プロセッサを設計し、その評価を行なう。

適用例としては、13個の代表的なディジタル信号処理アルゴリズムを取り上げる。まず、PARCOR格子型フィルタを用いて、プロセッサ設計過程を示す。次に全てのアルゴリズムに対しプロセッサ設計を行ない、それぞれのハードウェア要素を比較する。また、本システムで設計されたプロセッサの評価のために、代表的なDSPであるTMS320C25を取り上げ、ハードウェアとソフトウェアの面から比較検討する。

2 設計支援システムの主眼点

本システムは、LSI設計を支援するシステムであるが、従来のLSI CADのようにハードウェア設計としての枠組みに縛られず、アーキテクチャ、ソフトウェア(コンパイラ)を含めた統合システムを考える。LSI CAD[3]、プロセッサ[4][5]、コンパイラ[6][7]研究の現状を踏まえて、本システムの基本となる3つの主眼点について述べる。

2.1 幅広いユーザに対するプロセッサ設計環境

ハードウェアやプロセッサの専門家に限らず、システムの高速化を望む幅広い分野のユーザに対して、プロセッサ設計環境を与えることを本システムの一番の主眼点に掲げる。このため、本システムの入力としては、高級言語を用いる。また、その高級言語において、プロセッサの詳細な動作仕様を与えるのではなく、プロセッサ上で実行したいアルゴリズムを通常のソフトウェアを記述するのと同じように与える。

(1) 高位の入力記述

LSI CADの上位レベルの入力はRTレベルが一般的であるが、RTレベルからプロセッサ設計を行なうにはハードウェアの構成要素やタイミング等に関する専門知識が必要である。これがハードウェアの専門家以外のユーザをプロセッサ設計の現場から遠ざける要因になっている。よって、対象ユーザを決めるのに、システムの入力を何にするかが重要なポイントになると考えられる。

ユーザの要求仕様を比較的容易に、生産性高く記述できるものとして、C言語やPascalといった手続き型の高級言語がある。これらの言語は、現在既に、システム設計を行なう際の仕様記述言語として、多くのユーザに受け入れられており、幅広い分野を対象とした入力言語としては、最も優れていると考えられる。これらの言語は、現状では、コンパイラを用いて特定の汎用のプロセッサのマシン語に変換され、その上で実行されると仮定されており、ソフトウェア記述言語という限定された用途に用いられている。しかし、そこに記述されているアルゴリズムから、プロセッサの仕様情報を抽出することによって、本システムのようなハードウェア設計支援システムの入力としても有効であると考える。そこで、本システムでは、Pascalを入力記述とする。

(2) プロセッサ記述

プロセッサ設計は、命令セット、内部構成、動作仕様等多くの点を考えなければならないが、それにはプロセッサに関する高度な専門知識が要求される。本システムは、Pascalを入力記述としているが、これを用いて、上に示したプロセッサ仕様を全て与えなければならないとすると、やはり対象ユーザを限定してしまう。

専用プロセッサ設計を考えた場合、その可変要素は、命令、記憶要素(レジスタ、RAM等)、データ型に限定することができ、その他の部分は固定要素として、どのプロセッサでも、ほぼ共通していると考えることができる。そこで、本システムではプロセッサの詳細設計をゼロから行なうのではなく、プロセッサの枠組み(固定要素)はテンプレートとして用意し、それに対して入力アルゴリズムを解析し、可変要素を決め、チューニングしていくという形をとる。

入力記述は、設計するプロセッサ上で実行したいアルゴリズムを与えればよく、そのためにどういう仕様を持ったプロセッサが必要かを決めるのは、あくまでシステムの役割とする。これによって、ユーザはハードウェア(プロセッサ)というものを意識することなく、ソフトウェア開発を行なっているのと全く同じ感覚での専用プロセッサ設計が可能となる。

2.2 速度性能を重視したマイクロプロセッサ

専用プロセッサを考える上で、速度性能は最重要項目であり、本システムでも、プロセッサの速度性能を向上

させることを主眼点に掲げる。一般に速度性能をあげるための技術として、セマンティクスギャップの縮小、並列処理、ウェアの強化、高速スイッチング素子の採用がある。本システムで、これらの項目をどの様に考慮していくかの指針を述べる。

(1) セマンティックギャップの縮小

ソフトウェアと命令セットのセマンティックギャップを縮小することによって、速度性能向上が可能である。このため、ソフトウェア概念に近い高機能命令が望まれる。しかし、汎用プロセッサの場合それらの高機能命令は実際に使われる頻度は小さいため、高機能命令導入によってハードウェアが複雑になる割には効果が得られていない。よって、簡単な命令のみを用意し、命令形式の単純化やパイプライン処理などにより、処理性能を上げたもの(RISC)のほうが、かえって高い性能を得るという結果を得ている。

本システムでは専用プロセッサ設計としての特徴を生かし、高機能命令は積極的に取り入れ、セマンティックギャップの縮小をはかる。よって命令セット設計方針は、命令は、最初入力アルゴリズムを実行するのに必要最低限のものとし簡略化を図り、その上でアルゴリズムのセマンティクスを解析し、有用で高頻度な高機能命令を取り入れていくものとする。

(2) 並列処理

並列処理を行なうプロセッサとして、マルチプロセッサやベクトルプロセッサがある。しかし、その特徴を生かせるのは、行列計算におけるベクトル演算などもともと並列性の高い分野のアルゴリズムに限られる。一般的なアルゴリズムはそれほど並列性は高くなく、必要以上に並列実行できるシステムを用いても無駄である。

本システムでは、一般的なアルゴリズムが持つ並列性を必要なだけ生成することを考える。具体的には、本システムではプロセッサの制御方法として、プログラムカウンター制御を行なっているが、並列性を引き出すために高頻度に実行される部分をデータフロー制御にすることなどを考える。

(3) ウェアの強化

一般に、ソフトウェアよりもファームウェア、ファームウェアよりもハードウェアが高速である。このため、機能をできるだけハードウェア寄りに実現することを考える。CISC などにおける高機能命令は、ハードウェアに直結した単純な命令によって、機械命令をインタプリート実行して実現(マイクロプログラム制御)しているが、インタプリートは本来低速な操作なので高速実行には向かない。このため、本システムでは命令はすべて布線論理を用いて実現し、それらをマイクロ命令を用いて直接実行(RISC制御)させ、高速化をはかる。

(4) 高速スイッチング素子

高速スイッチング素子として、ECL、GaAs、ジョセフソン素子がある。これらの素子を利用するためには、ゲート数をできるだけ抑える必要がある。本システムは、専用プロセッサとして特定アルゴリズムを実行するのに必要な最低限のハードウェア構成(命令数、レジスタ数、RAM の大きさ等)とするのでゲート数は汎用プロセッサと比べて少なくてすみ、高速素子を利用できる可能性がある。また、この要因を考慮しながら、高機能命令導入等のトレードオフを行なっていくことも有効である。

2.3 ソフトウェアを含めた総合システム

本システムは、専用プロセッサ単独における性能向上を考えるのではなく、ソフトウェアを含めた総合としての性能向上を考える。よって、専用プロセッサを設計支援するのと同時に、設計されたプロセッサに対する最適化コンパイラを自動合成し、ソフトウェア開発のため環境も提供する。

3 処理概要

本システムは、中間情報生成系、解析系、合成系、コード生成系の4つ系から構成される。(図1)

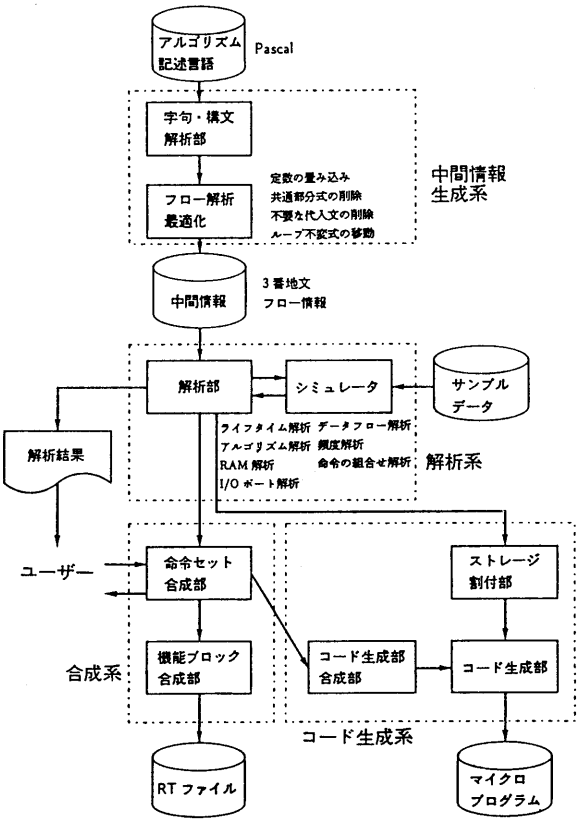

図1 処理概要

中間情報生成系は、Pascalで記述された入力アルゴリズムに対し字句・構文解析を行ない、種々の最適化を行なった後、中間情報(3番地文、フロー情報)を生成する。

解析系は、中間情報に対しライフタイム解析、アルゴリズム解析、RAM解析、I/Oポート解析、頻度解析、命令の組合せ解析、データフロー解析等の解析を行う。前者4つは入力アルゴリズム実現に最低限どれだけの命令、ハードウェア要素が必要かの情報を調べ、後者3つは命令高機能化のための情報を調べる。

合成系は、解析系で得られた諸情報を基に命令セットを合成する。この際、ユーザは解析結果をもとに命令セット、記憶要素(レジスタ、RAM等)を変更することができる。そして、合成された命令セットを基に機能ブロックを合成し、プロセッサ情報(ハードウェア要素、マイクロ命令のフォーマット等)を出力する。

コード生成系は、合成系で得られた命令セットを基にコード生成部を合成し、マイクロプログラムを出力する。

4 複数のアルゴリズムを実行するプロセッサ設計

専用プロセッサ設計を考えた場合、対象をできるだけ狭い範囲に限定する方が、より高性能なプロセッサ設計が可能である。しかしながら、個々のアルゴリズムごとに、数多くのLSIを設計するよりも、速度性能を多少犠牲にしても、ある程度の汎用性を持ったプロセッサ設計への要求は高い。この場合、幅広いアルゴリズムを対象とすると、無駄な要素が多く生成され、そのための性能低下は避けられないので、専用プロセッサを設計する意味が薄れてきてしまう。しかし、アルゴリズムの特徴が似ているものに限定すれば、本システムのようなプロセッサ設計では、布線論理を用いたLSI設計と違って多くのハードウェア要素を共有できるため、それほど無駄な要素が生成されることはなく、性能低下も少ないと考えられる。

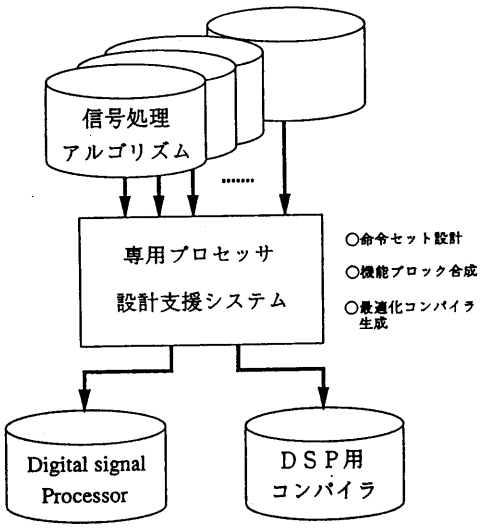

図2 複数アルゴリズム

特徴が似たアルゴリズム群の代表例としては、音声、画像処理等で用いられるディジタル信号処理アルゴリズムが考えられる。本システムを用いて、複数のアルゴリズムを実行する信号処理プロセッサを設計する際の処理概要を図2に示す。システムは入力された信号処理アルゴリズムそれぞれに対して、解析を行ない、命令セット、記憶要素の最適設計を行う。そしてそれらのOR要素を取って全体を統合し、合成したプロセッサをRT情報として出力する。また、同時にそのプロセッサ向けの最適化コンパイラを生成する。

5 適用例

本システムを用いた専用プロセッサ設計と、その評価に用いる入力アルゴリズムとして、次に示す13個の代表的な信号処理アルゴリズムを取り上げる。

○ PARCOR格子型フィルタ	… PAR
○ 自己相関関数	… COR
○ デルタ変調方式の符号化	… CDM
○ デルタ変調方式の復合化	… DDM
○ ADPCM符号化	… CAD
○ ADPCM復合化	… DAD
○ CEPSTRUM算出	… CEP
○ Durbin-Levinson-板倉法	… DUR
○ FFT(高速フーリェ変換)	… FFT
○ Pitch抽出	… PIT
○ Biquadフィルタ	… BIQ
○ FIRフィルタ(1次)	… FIR
○ IIRフィルタ(3次)	… IIR

これらのアルゴリズムを用いて、種々のプロセッサの設計、評価を行なう。まず、PARCOR格子型フィルタをシステムに入力し、設計の流れを示す。そして、全てのアルゴリズムを対象として、複数のアルゴリズムを実行するプロセッサの設計を行なう。

5.1 PARCOR格子型フィルタ

PARCOR格子型フィルタのアルゴリズム記述の例を図3に、頻度解析結果を図4に、命令の組合せ解析結果を図5に、合成された命令セットフォーマットを図6に、コード生成結果を図7に示す。

アルゴリズム記述は、ビット型による変数宣言、入出力変数宣言(input,output)以外は、ほぼPascalの言語仕様に基づいて記述できる。ビット型は、ユーザがデータの精度をより厳密に記述できる手段を与えるものである。入出力変数は、外部とのデータのやり取りを明確にするためのもので、ハードウェア化された場合、入出力端子

```
program parcorfilter (s_in, rk_in, e_out);
const    m1 = 13; n = 128;
type     integer = signed bit 16;
input    s_in, rk_in  : signed bit 12;
output   e_out  : signed bit 12;
var      ft, gto,rk  : array [m1] of integer;
         s,e  : array [n] of signed bit 12;
         i, j  : integer;
begin
  { data input }
  for i := 0 to n-1 do
    s[i] := s_in;
  for i := 0 to m1-1 do
    rk[i] := rk_in;
  { Filter main program }
  for i := 0 to m1-1 do begin
    gto[i] := 0; ft[i]  := 0 end;
  e[0] := s[0];
  for j := 1 to n-1 do
    begin
      ft[0] := s[j]; gto[0] := s[j-1];
      for i := 1 to m1-1 do
        begin
          ft[i] := ft[i-1] - rk[i-1] * gto[i-1];
          gto[i] := gto[i-1] - rk[i-1] * ft[i-1]
        end;
      e[j] := ft[m1-1]
    end;
  { data output }
  for i := 0 to n-1 do
    e_out := e[i]
end.
```

図3　アルゴリズム記述

として実現される。

頻度解析は、シミュレーターにサンプルデータを与え動的解析を行なった結果、各基本ブロックが何回参照されたかをそのフローグラフと共に示したものである。全19ブロック中、1回しか実行されないブロックもあれば、ループが多重になり1500回以上も実行されているブロック(B0013、B0014)もあることを示している。

命令の組合せ解析は、頻度解析の結果、高頻度に実行されているB0014に対して、命令の複合化を行なったものである。ステート32と33、36と38が同じ積差演算の形をしており、それらの高機能化を行なった命令としてOP1を合成している。また、ステート39、40はインクリメントと無条件分岐と互いに無関係な命令の組合せであり、その並列化を行なった命令としてOP2を合成している。

命令セット合成は、解析系で得られた情報をもとに、演算命令、分岐命令、メモリアクセス命令、複合命令ごとに規則性を保った上で、命令長ができるだけ短くなるように決定する。図の左側は、命令の型ごとの命令フォーマット、図の右側は、生成された命令とその命令に割り付けられたビット列を示している。

コード生成は、自動合成されたコード生成部を用いて行なわれる。図の左側は中間表現(基本ブロックによって構成された3番地文の列)、中央は生成されたコードに対するニーモニック、右側はビット列を示している。この例

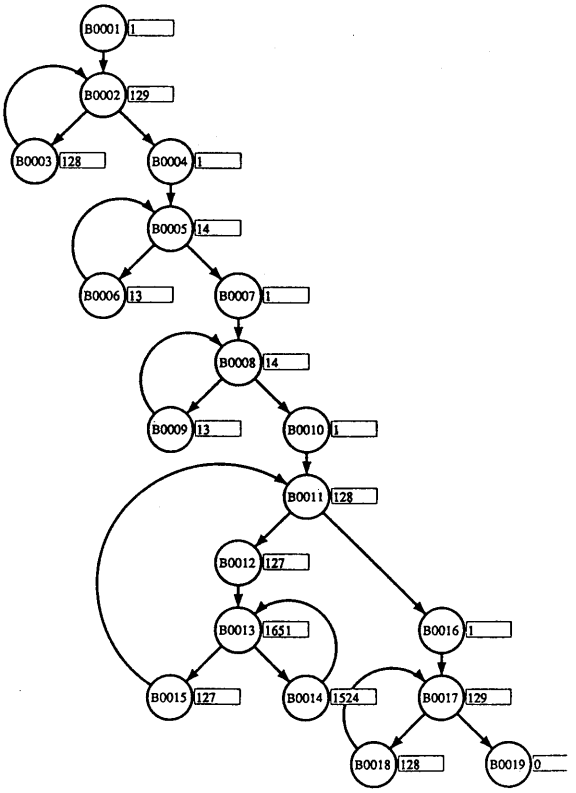

図4　頻度解析

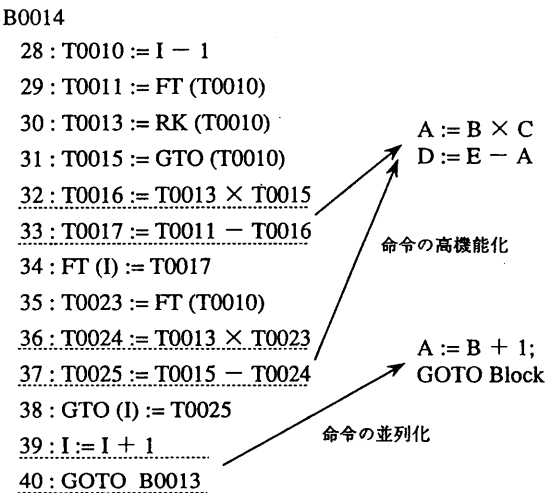

図5　命令の組み合せ解析

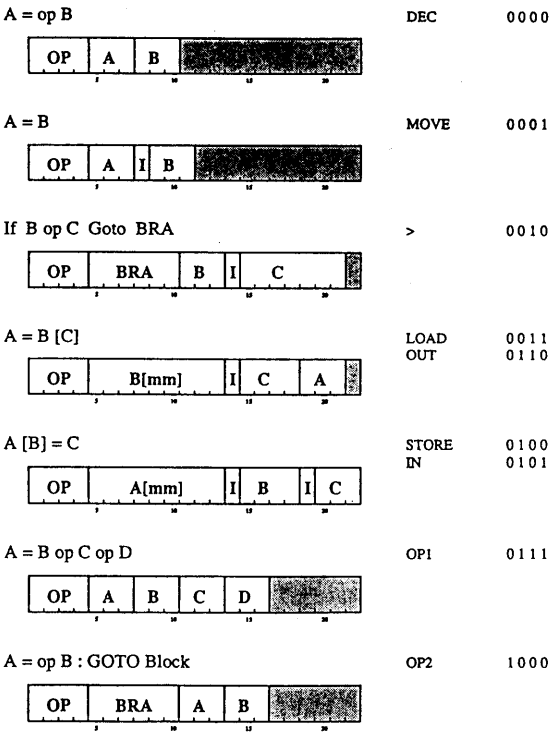

図6　命令セットフォーマット

では、複合命令の導入により、全41個の3番地文に対して、生成されたコード数は41である。

5．2　複数アルゴリズムを実行するプロセッサ

5．1に示した、PARCOR格子形フィルタ同様に、残りの12個の信号処理アルゴリズムに対しても、プロセッサ設計を行なう。また、同時に13個のアルゴリズムを入力し、複数のアルゴリズムを実行するプロセッサ設計を行ない、それぞれを比較する。

各アルゴリズムに対するレジスタ数の関係を図8に、データRAMの関係を図9に、プログラムROMの関係を図10に、命令数の関係を図11に示す。

レジスタ数は、入力アルゴリズムが、その変数格納にどれだけのレジスタが必要かの度合を示す。アルゴリズムによって、必要なレジスタは2から32までグループ分けされる。全体の数は、最も多い32(FFT)となる。

RAM数は、入力アルゴリズムが、配列データ格納にどれだけの内部RAMが必要かの度合を示す。やはりアルゴリズムによってグループ分けされる。全体は、最も多くの配列データを扱うPIT(4096W)によって決められる。

ROM数は、入力アルゴリズムが、どれくらいの長さを持つかの度合を示す。やはりアルゴリズムによるグループ分けが可能である。全体は、アルゴリズムの長いCAD、FFT、PIT(128W)によって決められる。

命令数は、入力アルゴリズムがどれくらいの複雑さを持つかの度合を示す。全体は、全ての命令のOR要素を取って決められる。

信号処理アルゴリズムの中でも、どれくらいのレジスタ、RAM、ROM、命令が必要か等のアルゴリズム的特徴は様々である。複数のアルゴリズムを実行するプロセッ

```
B0001 :
  1 : I := 0                    : MOVE WR6 0              : 00011101000000000000
B0002 :
  2 : IF I > 127 GOTO B0004     : > WR6 127 B0004         : 00100001011101111111110
B0003 :
  3 : S (I) := S_IN             : IN S WR6 IPORT          : 01010000110100011100111
  4 : I := I + 1                : OP2 B0002 WR6 WR6       : 10000000101101100000000
  5 : GOTO B0002                :                         :
B0004 :
  6 : I := 0                    : MOVE WR6 0              : 00011101000000000000
B0005 :
  7 : IF I > 12 GOTO B0007      : > WR6 12 B0007          : 00100010011101000011000
B0006 :
  8 : RK (I) := RK_IN           : IN RK WR6 IPORT         : 01010100110100011100111
  9 : I := I + 1                : OP2 B0005 WR6 WR6       : 10000001101101100000000
  10 : GOTO B0005               :                         :
B0007 :
  11 : I := 0                   : MOVE WR6 0              : 00011101000000000000
B0008 :
  12 : IF I > 12 GOTO B0010     : > WR6 12 B0010          : 00100011101101000011000
B0009 :
  13 : GTO (I) := 0             : STORE GTO WR6 0         : 01000000011010011010000
  14 : FT (I) := 0              : STORE FT WR6 0          : 01000000000000011010000
  15 : I := I + 1               : OP2 B0008 WR6 WR6       : 10000010101101100000000
  16 : GOTO B0008               :                         :
B0010 :
  17 : T0004 := S (0)           : LOAD WR0 S 0            : 00110000110101000000000
  18 : E (0) := T0004           : STORE E 0 WR0           : 01000101001111000000000
  19 : J := 1                   : MOVE WR5 1              : 00011011001000000000000
B0011 :
  20 : IF J > 127 GOTO B0016    : > WR5 127 B0016         : 00101001101011111111110
B0012 :
  21 : T0006 := S (J)           : LOAD WR0 S WR5          : 00110000110100010100000
  22 : FT (0) := T0006          : STORE FT 0 WR0          : 01000000000001000000000
  23 : I := 1                   : MOVE WR6 1              : 00011101001000000000000
  24 : T0007 := J - 1           : DEC WR0 WR5             : 00000001010000000000000
  25 : T0008 := S (T0007)       : LOAD WR0 S WR0          : 00110000110100000000000
  26 : GTO (0) := T0008         : STORE GTO 0 WR0         : 01000000011011000000000
B0013 :
  27 : IF I > 12 GOTO B0015     : > WR6 12 B0015          : 00101000101101000011000
B0014 :
  28 : T0010 := I - 1           : DEC WR0 WR6             : 00000001100000000000000
  29 : T0011 := FT (T0010)      : LOAD WR1 FT WR0         : 00110000000000000000010
  30 : T0013 := RK (T0010)      : LOAD WR2 RK WR0         : 00110100110100000000100
  31 : T0015 := GTO (T0010)     : LOAD WR3 GTO WR0        : 00110000011010000000110
  32 : T0016 := T0013 * T0015   : OP1 WR2 WR3 WR1 WR1     : 01110100110010010000000
  33 : T0017 := T0011 - T0016   :                         :
  34 : FT (I) := T0017          : STORE FT WR6 WR1        : 01000000000000011000001
  35 : T0023 := FT (T0010)      : LOAD WR0 FT WR0         : 00110000000000000000000
  36 : T0024 := T0013 * T0023   : OP1 WR2 WR0 WR0 WR3     : 01110100000000110000000
  37 : T0025 := T0015 - T0024   :                         :
  38 : GTO (I) := T0025         : STORE GTO WR6 WR0       : 01000000011010011000000
  39 : I := I + 1               : OP2 B0013 WR6 WR6       : 10000110001101100000000
  40 : GOTO B0013               :                         :
B0015 :
  41 : T0027 := FT (12)         : LOAD WR0 FT 12          : 00110000000001110000000
  42 : E (J) := T0027           : STORE E WR5 WR0         : 01000101001110010100000
  43 : J := J + 1               : OP2 B0011 WR5 WR5       : 10000100011011010000000
  44 : GOTO B0011               :                         :
B0016 :
  45 : I := 0                   : MOVE WR6 0              : 00011101000000000000
B0017 :
  46 : IF I > 127 GOTO B0019    : > WR6 127 B0019         : 00101010001101111111110
B0018 :
  47 : E_OUT := E (I)           : OUT OPORT E WR6         : 01100101001110011011110
  48 : I := I + 1               : OP2 B0017 WR6 WR6       : 10001001111101100000000
  49 : GOTO B0017               :                         :
B0019 :
```

図7　コード生成

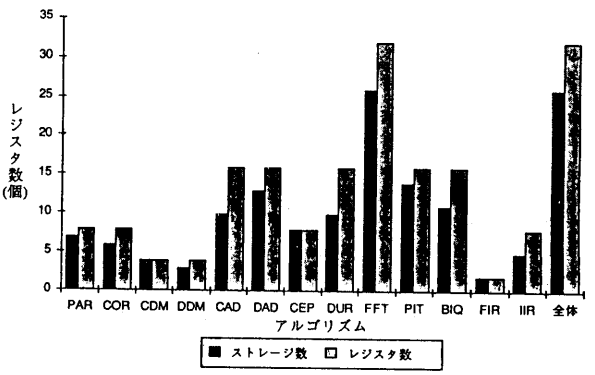

図8　各アルゴリズムに対するレジスタ数

サを設計する場合、より、高性能化を望むなら、アルゴリズムの特徴によって、さらにグループ分けを行なった設計が有効である。

6 評価

本システムを用いて設計された、複数の信号処理アルゴリズムを実行するプロセッサと、信号処理プロセッサTMS320C25(以下C25)の比較検討を行なう。表1に両者をハードウェアとソフトウェアの面から比較を行なった結果を示す。

ハードウェアの比較において、命令数はC25と比較して少ないが、これは本システムの場合、汎用性を持たせたといっても、完全な汎用DSPではないためである。本システムでは、設計途中で命令の追加等のユーザの介入を許しており、これによりさらに汎用性を追求したプロセッサ設計も可能である。

データ型は、両者とも符号付きの固定小数点を用いている。最近のDSPは、浮動小数点を取り扱えるものが増えてきているが、これについては検討中である。

特殊命令は、両者とも信号処理アルゴリズム特有な積和、積差といった演算を行なうものを備えている。

内部データRAMはC25と比較して多いが、これはピッチ抽出など多くのデータ領域を必要とするアルゴリズムが含まれるためである。このアルゴリズムを実行する場合、C25ではデータ格納のために内部RAMと外部RAMを使い分けなければならず、効率が悪い。

プログラムROMはC25と比較して少ないが、これは入力アルゴリズムの規模がそれほど大きくないのと、コンパイラが効率のよいコード生成を行なうためである。C25では、ROM領域の大部分は使用されず、無駄なハードウェア要素となる。

レジスタは多いが、これは本システムがRISCのようにレジスタ演算を基本としているためである。C25はレジスタが少ないため、メモリ演算が多くなり効率が悪い。

ソフトウェアの比較において、ソフトウェア開発は、本システムではPascal、C25ではアセンブリ言語を用いて行う。C25はC言語によるソフトウェア開発も可能であるが、次に示すように、コンパイラは、現状ではプロセッサの性能を十分に引き出していない。

コード生成は、同じアルゴリズムに対し、コンパイラを用いてコード生成をおこない、そのコード数を比較したものである。図3のPARCOR格子型フィルタを等価なC言語に書き直し、C25用のC言語コンパイラを用いてコード生成を行なう。最も高頻度に参照されるB0014のコード数を比較した結果、10ステップと111ステップになった。このブロックは1500回以上も参照されるので、プログラム実行時間に大きな差が出ると思われる。C25の場合、アセンブリ言語を使用すれば、かなりこのステップの短縮が期待できることから、C言語コンパイラによるソフトウェア開発は実用に至っていないと考えられる。

一般に、信号処理プロセッサのソフトウェア開発は低次元のアセンブリ言語を用いて行なわねばならず、効率が悪い[8]。本システムは、高級言語によるソフトウェア開発を前提としており、命令セット等もコンパイラが効率のよいコードを生成できるように決められる。このように、ソフトウェア開発環境は本システムの方が優れている。

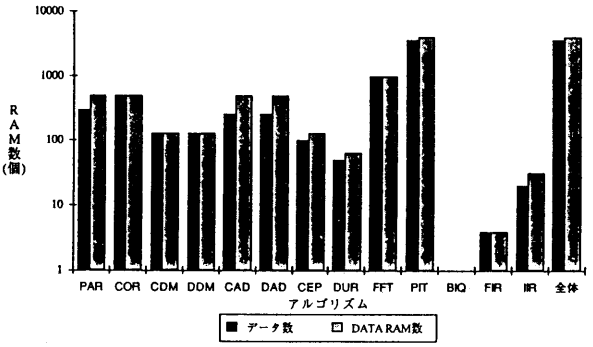

図9 各アルゴリズムに対するRAM数

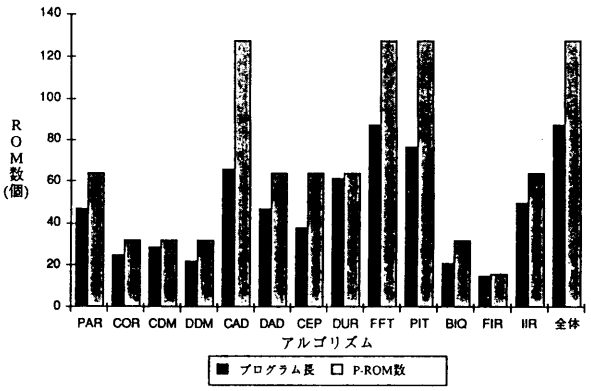

図10 各アルゴリズムに対するROM数

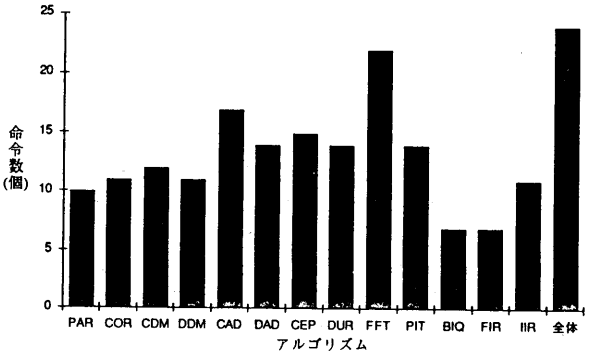

図11 各アルゴリズムに対する命令数

表1　TMS320C25との比較

		本システム	TMS320C25
ハードウェア	命令数	24個	133個
	データ型	32Bit 固定小数点	16Bit 固定小数点
	特殊命令	積和、積差命令	積和、積差命令
	プログラムROM	4096W	544W
	内部データRAM	128W	4096W
	レジスタ	32W	8W
ソフトウェア	開発環境	Pascal	アセンブリ言語(C言語)
	コード生成PARCOR(B14)	10ステップ	111ステップ

7　おわりに

高級言語で書かれたアルゴリズム記述を入力とする専用プロセッサ設計支援システムの、基本となる3つの主眼点、処理概要について述べた。このシステムに複数の代表的なディジタル信号処理アルゴリズムを入力し、種々の専用プロセッサ設計を試みた。この結果、高級言語を用いて、通常のソフトウェア開発と同じ感覚で記述された入力記述から、そのアルゴリズム的特徴を生かした専用プロセッサ開発が可能なことが示された。また、設計された種々のプロセッサのハードウェア要素を比較した結果、信号処理アルゴリズムの中でも、そのアルゴリズム的特徴はさまざまであり、高性能化を望むならば、グループ分けを行なったプロセッサ設計を行なうことが有効であることがわかった。最後に、本システムを用いて設計された複数の信号処理アルゴリズム実行するプロセッサと、代表的な信号処理プロセッサTMS320C25とを比較した結果、特にソフトウェア開発環境に優れた専用プロセッサ設計が可能なことがわかった。

LSI CADは論理設計、機能設計とボトムアップ的に発展してきており、それぞれの設計ツールも実用化されつつある。しかし、その上位のシステム設計、方式設計といったレベルになると、まだ明確な方向付けはなく、現在様々なシステムが試行錯誤されている状態である。こういった中で、本システムのような、ソフトウェア、コンパイラという、従来のLSI CAD発展とは逆の立場に立った設計ツールは、今後のLSI CADの流れの中で、一つの新しいパスになるのではないかと思う。

本論文では適用例として、比較を行なうために、比較的専用プロセッサ化が進んだ信号処理分野を取り上げたが、高速化が期待される分野は他にも数多くあり、本システムはそういった専用化が進んでいない分野の専用プロセッサ開発にも重要な役割を担うと思われる。

最後に今後の課題を述べる。今後は、本システムで得られたプロセッサ情報を、既存のRTレベルのハードウェア記述言語に変換し、それからLSIのマスクパタンまでの設計を行なうハードウェア合成システムと共にトータルなLSI CADの実現を試みたい。また、逆にその合成システムから、ゲート数、命令実行時間等の情報をフィードバックされることにより、本システムのLSI設計の質をさらに向上させていきたい。

参考文献

[1] 持田侑宏：DSPの現状と動向，情報処理，Vol.30, No.11, pp.1291-1299 (Nov. 1989)

[2] 池永，竹沢，白井：高位の仕様記述に基づく専用プロセッサ設計支援システム，情報処理学会第39回全国大会，4X-7 (1989-10)

[3] 高木茂：VLSI設計CADの最近の動向，情報処理，Vol.28, No.5, pp.581-589 (May 1987)

[4] 宇都宮公訓：命令セットアーキテクチャの評価，情報処理，Vol.29, No.12, pp.1474-1481 (Dec. 1988)

[5] Colwel, R.P. et al.: "Computers, Complexity, and Controversy", Computer, Vol.18, No.9, pp.8-19 (Sep. 1985)

[6] Wulf, W.A.: "Compilers and Computer Architecture", Computer, Vol.14, No.7, pp.41-47 (July 1981)

[7] Ganapathi, M. et al.: "Retargetable Compiler Code Generation", Computing Surveys, Vol.14, No.4, pp.573-592 (Dec. 1982)

[8] 小野定康：DSPのプログラム開発環境，情報処理，Vol.30, No.11, pp.1307-1314 (Nov. 1989)