設計自動化 64－11
(1992.10.23)

ASIP用ハードウェア/ソフトウェア・コデザインシステム PEASの実現とその評価

佐藤 淳[1)], Alauddin Alomary[2)], 本間啓道[2)], 中田武治[2)], 博多哲也[3)], 今井正治[2)], 引地信之[4)]

[1)] 鶴岡工業高等専門学校, [2)] 豊橋技術科学大学,
[3)] 熊本電波工業高等専門学校, [4)] SRA

あらまし

CPUコアと周辺回路を集積化した高機能ASICであるASIP(Application Specific Integrated Processor)の応用システムの開発を効率良く行なうためのアプローチとしては，ハードウェアとソフトウェアの同時並行設計が有効であると考えられる．本稿ではASIPのハードウェアとソフトウェアの同時並行設計を行なうための設計手法を提案し，そのプロトタイプシステムであるPEAS(Practical Environment for ASIP Development)の実現とシステムの評価結果について述べる．

和文キーワード　特定用途向きマイクロプロセッサ, ハードウェア/ソフトウェア・コデザイン, 高位論理合成

Implementation and Evaluation of PEAS: Practical Environment for ASIP Development

Jun Sato[1)], Alauddin Alomary[2)], Yoshimichi Honma[2)], Takeharu Nakata[2)], Tetsuya Hakata[3)], Masaharu Imai[2)], Nobuyuki Hikichi[4)]

[1)] Tsuruoka National College of Technology, [2)] Toyohashi University of Technology, [3)] Kumamoto National College of Technology,
[4)] Software Research Associates

Abstract

Due to the advance of VLSI technologies, it is now possible to design and realize a very large scale ASIP(Application Specific Integrated Processor) which contains CPU core, memory and peripheral circuits. In order to develop design environment for application systems including ASIPs, Hardware/Software CoDesign is effective method. First, a Hardware/Software CoDesign system for ASIPs named PEAS(Practical Environment for ASIP) is proposed. Then, evaluation results of the PEAS system is described.

英文 key words　Application Specific Integrated Processor, Hardware/Software CoDesign, High-level synthesis

1. はじめに

半導体集積回路の集積度の向上およびASIC設計技術の発達に伴い，専用のCPUコアと周辺回路を集積化した高機能ASICであるASIP(Application Specific Integrated Processor)の開発が可能になった．ASIPはシステムの高機能化および小型化に大きく寄与するため，組み込み用途の応用システムへの利用が期待される．

ASIP応用システムの開発を行なうためにはハードウェアの開発に加えて応用プログラム開発環境もあらたに準備する必要がある．ハードウェアのみの開発には既存のASIC設計技術の適用が可能である．しかし，応用プログラムの開発にはコンパイラ，アセンブラ，シミュレータなどのソフトウェア開発ツールの準備が不可欠である．また，ハードウェアの性能を十分に引きだすソフトウェアを開発するためには最適化コンパイラの使用が必須である．

ASIP応用システムの開発を効率良く行なうためには，次のようなアプローチが有効であると考えられる．

(1)ハードウェアとソフトウェアの同時並行設計（コンカレント・デザイン）

(2)ハードウェアとソフトウェアのバランスのとれたシステム設計

筆者らは上記の問題点を解決する新しい設計手法の提案を行なった[4]．本稿では，この設計手法に基づいたプロトタイプ・システムPEAS(Practical Environment for ASIP Development)の実現と評価結果について述べる．

2. システムの特徴

PEASシステムは，特定の分野に適したシステムの開発を目標としている．本システムの設計手法の基本的な考えは，特定分野の応用プログラムの集合の挙動を解析し，得られた結果をもとに応用プログラムの実行に適したハードウェアの生成と生成されたハードウェアのソフトウェア開発ツールの生成を並行して行なうことにより，システムの開発期間を短縮することである．加えて，適用分野の特徴をもとにシステム設計を行なう場合，ハードウェア生成とソフトウェア開発ツールの生成を協調して行なうことにより，ハードウェアとソフトウェアのバランスのとれたシステムを開発することである．

この設計手法では，適用分野の特徴を反映するプログラムとデータが与えられ，開発マシンでデータの型やオペレーションの種類を解析することにより，適用分野の特性を抽出できることを前提としている．また，本システムが生成するCPUコアのフルセットの命令をコンパイラが生成可能な命令セットとすることにより，ソフトウェアとハードウェアのセマンティック・ギャップを小さくすると共にCPUコアの生成を容易にしている．

本システムでは，応用プログラムの命令の実行頻度を解析し，各々の命令の実現方法くを決定し，この情報を使用してCPUコアのデザインとソフトウェア開発ツールの生成を行なう部分の実現を行なった．

本システムでは，以下のような実現を行なうことにより，CPUコアの設計情報の決定およびCPUコアとソフトウェアの同時並行設計を可能にしている．

(1)C言語で記述された応用プログラムの動的解析を行ない，CPUコアのアーキテクチャ情報を自動生成する．

(2)CPUコアのフルセットの命令をGNU Cコンパイラの中間言語(RTL)に対応させることにより，ソフトウェアとハードウェアのセマンティック・ギャップを小さくする．

(3)ターゲットマシンへの移植が容易なGNU Cコンパイラを利用して最適化コンパイラの自動生成を行なう．

(4)CPUコアをパラメタ化して命令の実現方法の変更を容易にする．

3. システムの構成

本システムは以下に示す4つの系から構成される．

(1) 応用プログラム解析系

(2) アーキテクチャ生成系

(3) ハードウェア生成系

(4) ソフトウェア開発ツール生成系

アーキテクチャ生成系は応用プログラム解析系から得られる応用プログラムの解析結果をもとにASIPのアーキテクチャを決定する．このアーキテクチャ情報をもとに，ハードウェア生成系とソフトウェア開発ツール生成系の処理を並行して行なう．ハードウェア生成系はCPUコアや周辺回路の生成を行ない，ソフトウェア開発ツール生成系はコンパイラ，アセンブラ，シミュレータなどを生成する．

本システムはCPUコアの生成だけを対象にしているので，応用プログラム解析系は命令レベルの動的解析を行ない，その結果をもとにアーキテクチャ生成系はアーキテクチャ情報(arc)として各命令の実現方法を決定す

る．ハードウェア生成系とソフトウェア開発ツール生成系はアーキテクチャ情報をもとにCPUコアのデザインの生成，およびコンパイラ，アセンブラ，シミュレータの生成を行なう．また，アーキテクチャ生 成系とソフトウェア開発ツール生成系はモジュールデータベース

タベースは各々の命令を実現する際に必要なゲートサイズおよび消費電力などを含んでいる．図1に今回実現したシステムの構成を示す．

次に，今回実現したシステムの構成要素について述べる．

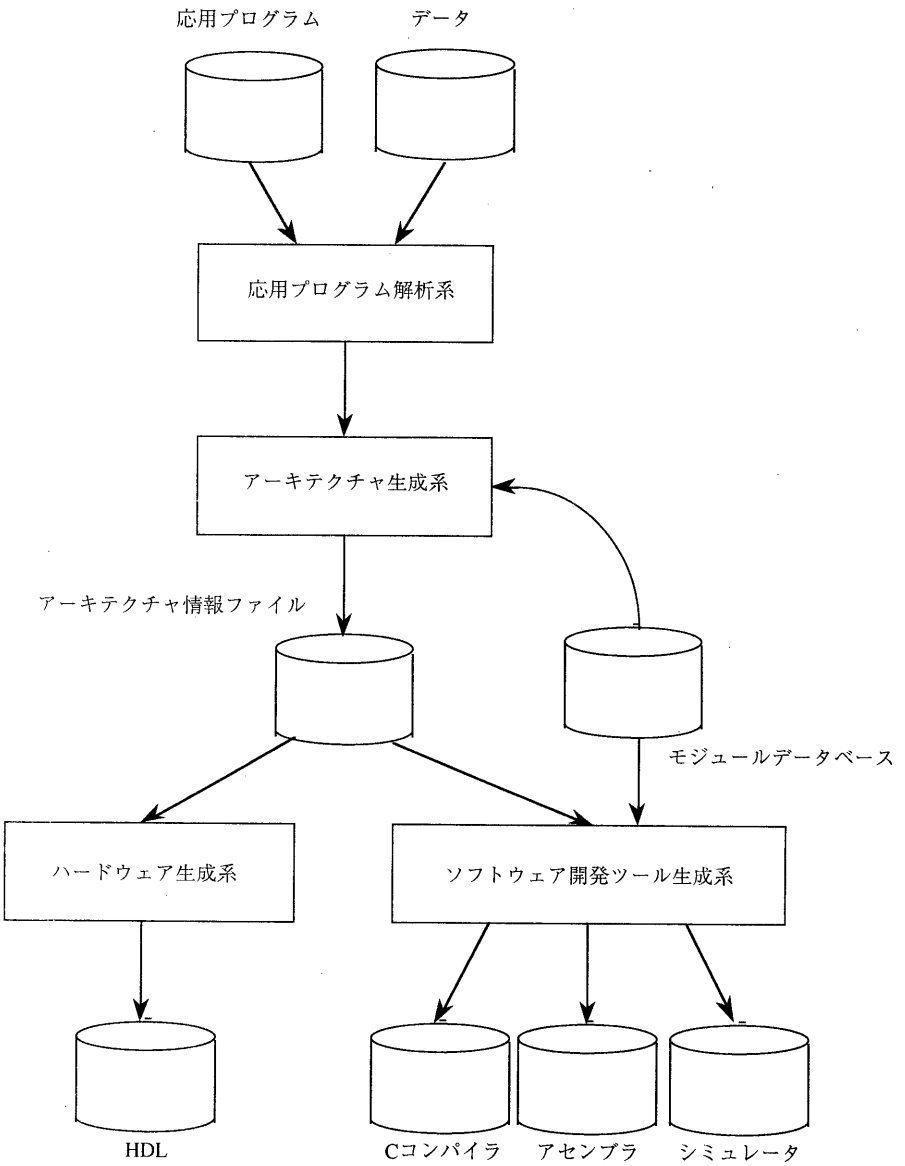

図1　PEASシステムの構成

3.1 応用プログラム解析系

応用プログラム解析系は応用プログラムの動的解析結果を命令レベルで行なう．今回は，フルセットの命令セットを持たせたCコンパイラで応用プログラムをコンパイルして得られたコードをシミュレータで実行することにより命令レベルの動的解析を行なった．

3.2 アーキテクチャ生成系[2]

命令セットは，与えられた制約条件（チップ面積，消費電力）のもとで，性能を最大とする命令の実現方法を選択することにより決定される．この問題を組み合わせ問題として定式化し，分岐限定法にもとづくアルゴリズムを使用して最適な命令セットおよび命令の実現方法の決定を行なう．

アーキテクチャ生成系はモジュールデータベースの情報を利用して，CPUコアの生成を行なう前に実行速度とゲート数の予測を行なうことが可能である．アーキテクチャ生成系はMFPS(Million Functionality Per Second)と呼ぶ単位で性能の見積もりを出力する．以下にCPUコアのクロック数を10MHzとした場合のMFPSの定義式を示す．

$$MFPS=(\sum_{i=1}^{n} f_i/(\sum_{i=1}^{n} f_i \times t_i)) \times 10$$

ここで，nは命令の個数，f_iはi番目の命令の実行回数，t_iはi番目の命令の実行時間である．

3.3 ハードウェア生成系

ハードウェア生成系はCPUコアのデザインをハードウェア記述言語(HDL)の形式で出力する．HDLとしてはSFL(Structured Function description Language)[3]を用いる．この出力を高位合成ツールPARTHENONに入力することにより最終的なハードウェア・デザインを生成することが可能である．

ハードウェア生成系では，命令セットの実現方法，レジスタファイルの大きさ，アドレスバスの幅が選択可能である．これらの項目はパラメタ化されているのでアーキテクチャの変更への対応を容易にすると共にモジュールデータベースのデータ生成も容易にしている．

3.4 ソフトウェア開発ツール生成系[1]

ソフトウェア開発ツール生成系は，コンパイラ生成部，アセンブラ生成部，シミュレータ生成部から構成される．これらは，アーキテクチャ情報をもとにCコンパイラ，アセンブラ，シミュレータを同時に生成する．

コンパイラ生成部ではGNU Cコンパイラをベースにして性能の良いコンパイラの生成を行なう．GNU Cコンパイラは移植性が高く最適化の能力が高いので採用した．

今回は，レジスタの割り付けの最適化をGNU Cコンパイラ，選択された命令の実現方法に関する最適化をアセンブラが行なっている．

4. システムの評価

今回行なったシステムの評価実験では，アーキテクチャ生成系とソフトウェア開発ツール生成系の評価を行なった．

4.1 アーキテクチャ生成系の評価

アーキテクチャ生成系の評価のために使用したサンプルプログラムをプログラム1に示す．このサンプルプログラムは自乗平均の近似値(y=sqrt(a^2+b^2)を求めるプログラムである．このサンプルプログラムを用いて生成されるCPUコアのゲート数および性能の見積もりを行ない，合成されたCPUコアのゲート数および性能のシミュレーション結果との比較を行なった．

今回使用したサンプルプログラムの命令の実行頻度を表1に示す．ここで，本システムが生成するCPUコアはALUと1ビットシフタなどの最小限の演算器をデフォルトで持つ．kernelはデフォルトのハードウェアで実現される命令を示す．

表1　命令の実行頻度

命令の種類	実行頻度
kernel	944
mulsi	14
ashlsi	14
ashrsi	28

この結果から，デフォルトのハードウェア以外に選択される可能性がある演算器はバレルシフタと乗算器である．表2に演算器の仕様を示す．

表2　演算器の仕様

モジュール名	ゲート数	実行時間（クロック数）
kernel	15672	1
b_sht	855	1
mul32	2938	32
mul17	3924	17

したがって，組み合わせは以下に示す6通りが考えられる．

(a) kernel

(b) kernel+b_sht

(c) kernel+mul32

(d) kernel+mul17

(e) kernel+b_sht+mul32

(f) kernel+b_sht+mul17

ここで，(a)～(d)のように実現すべき命令に直接対応する演算器が選択されない場合はkernelで実現される命令で構成されているランタイムルーチンを使用する．

次にゲート数の制約を16000～21000まで変化させてアーキテクチャの選択を行なった結果を表3，4に示す．ここで，結果が2種類ある理由は，ランタイムルーチンの実行時間が入力されるデータに依存するためである．ランタイムルーチンの実行時間として最大値を選択した場合がワーストケース，最小値を選択した場合がベストケースの結果である．

ゲート数が20000ゲートの場合，(d)の組み合わせは19596ゲートであるが，MFPSの値が3.13であるために選択されない．したがって，最適な組み合わせを選択していることがわかる．

次に，選択された演算器の組み合わせについて，論理合成およびソフトウェア開発ツールを生成し，ゲート数および性能の評価を行なった結果を表5に示す．

表3　性能の見積もり（ワーストケース）

制約条件	MFPS	ゲート数	選択された組み合わせ
16000	1.71	15672	(a)
16500	1.71	15672	(a)
17000	2.59	16527	(b)
17500	2.59	16527	(b)
18000	2.59	16527	(b)
18500	2.59	16527	(b)
19000	2.90	18610	(c)
19500	6.97	19465	(e)
20000	6.97	19465	(e)
20500	8.16	20451	(f)
21000	8.16	20451	(f)

表4　性能の見積もり（ベストケース）

制約条件	MFPS	ゲート数	選択された組み合わせ
16000	5.12	15672	(a)
16500	5.12	15672	(a)
17000	7.81	16527	(b)
17500	7.81	16527	(b)
18000	7.81	16527	(b)
18500	7.81	16527	(b)
19000	7.81	16527	(b)
19500	7.81	16527	(b)
20000	7.81	16527	(b)
20500	8.12	20451	(f)
21000	8.12	20451	(f)

表5　生成されたCPUコアの評価結果

選択された組み合わせ	MFPS	ゲート数	オブジェクトのサイズ	実行時間(サイクル数)
(a)	4.57	15672	1208	1564
(b)	6.71	16618	656	1064
(c)	4.69	18136	1096	1524
(d)	5.20	19029	1096	1374
(e)	6.97	19010	544	1024
(f)	8.17	19905	544	874

表3，4と表5を比較した結果を図2に示す．ここで，実線は見積もりの結果を示し，点は論理合成を行なった結果を示している．実際に論理合成を行なった結果はワースト ケースとベストケースの間にほぼ収まっている．したがって，アーキテクチャ生成部は現実のCPUコアのゲート数および性能の評価を行なっていることが知られる．

4.2 ソフトウェア開発ツール生成系の評価

あらかじめ，レジスタの個数を8，16，32としたCコンパイラを生成しておき，各々のコンパイラでプログラム2(reg-32.c)およびプログラム3(reg-12.c)をコンパイルして得られたオブジェクトコードの実行速度をシミュレータで測定することにより，レジスタの割付が適切に行なわれているかを調べた．reg-32.cとreg-12.cの違いは使用しているレジスタ変数の個数であり，reg-32.cは32個，reg-12.cは12個である．実験で使用しているCコンパイラはGNU Cコンパイラバージョン1.40をベースにしており，コンパイルは最適化なしで行なった．

次に，表6に実験の結果を示す．レジスタ数が大きいほど実行時間が短縮されていることが確認できる．

表6　レジスタ割付の実験結果

レジスタ数	reg-32.cの実行時間（サイクル数）	reg-12.cの実行時間（サイクル数）
8	955	323
16	871	303
32	799	299
64	787	299

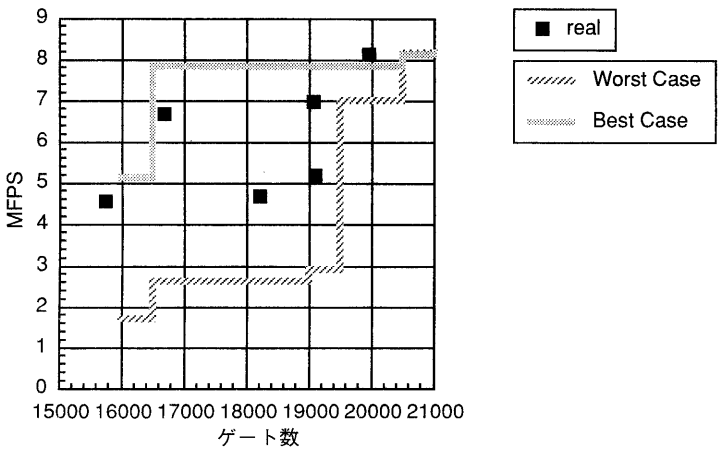

図2　性能の見積もりと合成してえられた結果の比較

5 おわりに

ハードウェアとソフトウェアの同時変更設計を行なうASIP設計システムPEASの概要と評価について述べた．

現段階では，ASIPのCPUコアに関連するシステムの実現を行なった．今後は，CPUコア・アーキテクチャの改良，浮動小数点命令の実現，拡張命令の実現，周辺回路の取り扱い方法などの検討を行なう予定である．

謝辞

本研究を御支援いただく(株)SRA 先端技術開発部，(株)NTT コミュニケーション科学研 究所，VLSIテクノロジー(株)，富士通VLSI(株)，および御討論いただいた豊橋技術科学 大学 VLSI設計研究室の諸兄に深謝いたします．

参考文献

[1] 博多哲也, 佐藤淳, 今井正治, 引地信之:"ASIC CPU向きソフトウェア開発環境生 成系の実現," 電子情報通信学会技術報告, VLD92-25, pp. 43-48, 1992.

[2] Imai, M., Alomary, A.Y., Sato, J., Hikichi N.: "An Integrated Approach to Instruction Implementation Method Selection Problem," Euro-DAC'92, pp. 106-111, Sep. 1992.

[3] NTTデータ通信: PARTHENON User's Manual, 1989.

[4] Sato, J., Imai, M., Hakata, T., Alomary, A.Y., and Hikichi, N.: "An Integrated Design Environment for Application Specific Integrated Processor," Proceedings of ICCD'91, pp. 414-417, Oct. 1991.

プログラム 1　mul.c

```
#define ABS(X) ((X < 0) ? (-X) : (X))
main()
{
 int i, x, y;
 int z;
 x = 3; y = 12;
 for (i = 0; i < 10; i++) {
  x++; y++;
  z = _psqrt(x, y);
 }
}
int _psqrt(x, y)
int x, y;
{
 int a, b, result, dummy;
 x = ABS(x); y = ABS(y);
 if (x >= y) {
  a = x; b = y;
 }
 else {
  a = y; b = x;
 }
 result = ((a * 7) + (b * 4)) >> 2;
 dummy = result & 0x00000001;
 if (a > result) result = a;
 return(result);
}
```

プログラム 2　reg-32.c

```
main()
{
 int x;
 int r;
 x = var(0);
 r = var(x);
 x = var(r);
 r = var(x);
}
var(x)
int x;
{
 register x0, x1, x2, x3, x4, x5, x6, x7;
 register x8, x9, x10, x11, x12, x13, x14, x15;
 register x16, x17, x18, x19, x20, x21, x22, x23;
 register x24, x25, x26, x27, x28, x29, x30, x31;
 int r1, r2, r3, r4;

 x0 = x; x1 = 1; x2 = 2; x3 = 3;
 x4 = 4; x5 = 5; x6 = 6; x7 = 7;
 x8 = 8; x9 = 9; x10 = 10; x11 = 11;
 x12 = 12; x13 = 13; x14 = 14; x15 = 15;
 x16 = 16; x17 = 17; x18 = 18; x19 = 19;
 x20 = 20; x21 = 21; x22 = 22; x23 = 23;
 x24 = 24; x25 = 25; x26 = 26; x27 = 27;
 x28 = 28; x29 = 29; x30 = 30; x31 = 31;

 r1 = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7;
 r2 = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 +
   x8 + x9 + x10 + x11 + x12 + x13 + x14 + x15;
 r3 = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 +
   x8 + x9 + x10 + x11 + x12 + x13 + x14 + x15 +
   x16 + x17 + x18 + x19 + x20 + x21 + x22 + x23;
 r4 = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 +
   x8 + x9 + x10 + x11 + x12 + x13 + x14 + x15
   x16 + x17 + x18 + x19 + x20 + x21 + x22 + x23 +
   x24 + x25 + x26 + x27 + x28 + x29 + x30 + x31;
 return(r1 + r2 + r3 + r4);
}
```

プログラム 3　reg-12.c

```
main()
{
 int x;
 int r;

 x = var(0);
 r = var(x);
 x = var(r);
 r = var(x);
}

var(x)
int x;
{
 register x0, x1, x2, x3, x4, x5, x6, x7;
 register x8, x9, x10, x11;
 int r1, r2, r3, r4;

 x0 = x; x1 = 1; x2 = 2; x3 = 3;
 x4 = 4; x5 = 5; x6 = 6; x7 = 7;
 x8 = 8; x9 = 9; x10 = 10;

 r1 = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7;
 r2 = x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7 +
   x8 + x9 + x10;

 return(r1 + r2 + r3 + r4);
}
```