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

2007－ＳＬＤＭ－132 (23)
2007／11／22

ASIP 短期開発のための高い拡張性を有するベースプロセッサの提案

岩戸 宏文[†,††] 稗田 拓路[†,††] 田中 浩明[†,††] 佐藤 淳[†††,††] 坂主 圭史[†,††]
武内 良典[†,††] 今井 正治[†,††]

† 大阪大学 大学院情報科学研究科 〒565–0871 大阪府吹田市山田丘 1-5
†† エイシップ・ソリューションズ株式会社 〒541–0053 大阪市中央区本町 2–3–8
††† 鶴岡工業高等専門学校 電気電子工学科 〒997–8511 山形県鶴岡市井岡字沢田 104
E-mail: †{h-iwato,t-hieda,h-tanaka,sakanusi,takeuchi,imai}@ist.osaka-u.ac.jp, ††jun@tsuruoka-nct.ac.jp

あらまし ASIP (Application Specific Instruction-set Processor) はアプリケーションごとに特化した命令セットアーキテクチャを持つプロセッサであり，ASIC 並の性能と汎用プロセッサ並のプログラマビリティを備えた組み込みプロセッサとして注目されている．ASIP はアプリケーションごとにプロセッサを設計しなければならないことから，開発が長期化することが問題とされている．ASIP の開発を効率化するためのツールとして ASIP Meister が知られている．ASIP Meister は ASIP をプロセッサ仕様記述から自動生成するツールであり，大幅に ASIP の開発期間を短縮できる．しかし，近年の短 TAT 要求の高まりを受け，ASIP 開発のより一層の効率化が望まれている．そこで本研究では，ベースプロセッサ・ファミリ Brownie を提案することで ASIP の開発期間を短縮する．Brownie は ASIP Meister と高い親和性を持つベースプロセッサ群であり，多数の候補からベースプロセッサとして最適なプロセッサを選択できることなどから，ASIP を効率的に設計できる．本稿では Brownie の設計思想，アーキテクチャ概要，Brownie をベースとした ASIP Meister による ASIP 設計例を示し，Brownie の有用性を示す．

キーワード ASIP, ベースプロセッサ, Brownie, ASIP Meister

A Highly Extensible Base Processor for Short-term ASIP Design

Hirofumi IWATO[†,††], Takuji HIEDA[†,††], Hiroaki TANAKA[†,††], Jun SATO[†††,††], Keishi SAKANUSHI[†,††], Yoshinori TAKEUCHI[†,††], and Masaharu IMAI[†,††]

† Graduate School of Information Science and Technology, Osaka University 1–5, Yamadaoka, Suita-shi, Osaka, 565–0871 Japan
†† ASIP Solutions, Inc. 2–3–8, Honmachi, Chuo-ku, Osaka-shi, Osaka, 541–0053 Japan
††† Department of Electronical and Electronics Engineering, Tsuruoka National College of Technology 104 Sawada, Ino-oka, Tsuruoka-shi, Yamagata, 997-8511 Japan
E-mail: †{h-iwato,t-hieda,h-tanaka,sakanusi,takeuchi,imai}@ist.osaka-u.ac.jp, ††jun@tsuruoka-nct.ac.jp

Abstract ASIPs (Application Specific Instruction-set Processors) are embbeded processors whose architectures are customized for specific applications. A problem of employing ASIPs is a long period of development due to the need of designing a processor for each application respectively. ASIP Meister is a tool that is able to automatically generate HDL descriptions of ASIPs from their specifications. However, although designers use ASIP Meister, it is hard to develop ASIPs in a required period because the demand of shorter TAT has been increasing today. Therefore, base processors for ASIP Meister are required to accelerate developing speed of ASIPs. In this paper, we propose a base processor called Brownie, which is suitable for ASIP Meister, and show the effectiveness of using Brownie through designing several samples.

Key words ASIP, Base Processor, Brownie, ASIP Meister

1. はじめに

ASIP（Application Specific Instruction-set Processor）は各種用途向けに特化された命令セットアーキテクチャを持つプロセッサであり，小面積，低消費電力，高性能なプロセッサを実現できるため，組み込みシステム向けのプロセッサとして注目を集めている．ASIP を用いることで高性能な組み込みシステムを設計できるが，一方で ASIP の設計自体に多大な設計期間がかかるという問題もはらんでいる．

ASIP 開発方式には既存プロセッサ拡張方式，テンプレート方式，ADL（Architecture Description Language）方式，HDL 設計方式の 4 種類の方式が知られている．これらの関係を図 1 に示す．開発効率および設計自由度のトレードオフを考慮すると，既設計のプロセッサを人手でカスタマイズする既存プロセッサ拡張方式は柔軟性が少なく，またプロセッサをスクラッチから人手で設計する HDL 設計方式は開発効率が悪いため，実際にはテンプレート方式か ADL 方式を用いて ASIP 開発が行われることが多い．

テンプレート方式は，あるベースプロセッサを基に専用のツールを用いて命令拡張を行う方式で，MeP [4] や Xtensa [6] などがよく知られている．ASIP の多大な設計期間のうち，大半が基本的なプロセッサの仕様策定や，その実装や検証であったりと，設計者が力を注ぐべき拡張命令の設計とは関係のない部分で労力を浪費している．ASIP 設計においては，設計者が目的に特化した拡張命令の設計に注力できることが理想であり，そのためにはベースとなるプロセッサに拡張命令を追加する形で開発することが望ましい．

このようなベースプロセッサ拡張による ASIP 開発は効率的な設計方法として一般的によく用いられているが，いくつかの問題点も指摘されている．例えば，ベースプロセッサの選択肢がそれほど多くないことや，拡張命令の設計がベースプロセッサのアーキテクチャにしばられてしまうことなどである．目標に出来るだけ近い仕様が備わっているベースプロセッサを利用することが，設計者の余分な負担の削減や，全体の開発期間短縮，ASIP の性能向上のために重要である．しかし，一般的なベースプロセッサを用いた設計では，ベースプロセッサの仕様が目標仕様と必ずしも一致していないこともあり，機能過剰や機能不足などから結局余分な労力を浪費してしまいがちである．ASIP 設計においては，目的・要求に応じた多数のアーキテクチャ候補が存在し，設計者がそこから自由にベースプロセッサを選択できることが望ましい．もうひとつのテンプレート方式による ASIP 開発の問題点は，ベースとするプロセッサのアーキテクチャが固定されており，その制約を強くうけてしまうため，拡張命令の柔軟な設計が望めないことである．設計の柔軟性は効率的な ASIP を設計する際に必要不可欠なものである．

ADL 方式は独自のアーキテクチャ記述言語を用いてプロセッサを設計する方式であり，LISA [1] や ASIP Meister [2] が知られている．ASIP Meister はプロセッサの仕様記述からプロセッサの HDL 記述，およびソフトウェア開発環境を自動的に生成するツールであり，リソースの追加やアーキテクチャの変更，命令の拡張などを柔軟に行えるため，効率的に ASIP を開発できる．しかし，ASIP Meister を利用した ASIP 開発の場合でも，基本的なプロセッサ仕様の策定といった労力は避けられない．ASIP の柔軟な開発を短期間で行うためには，ASIP Meister で利用できる，高い拡張性を有するベースプロセッサが必要である．

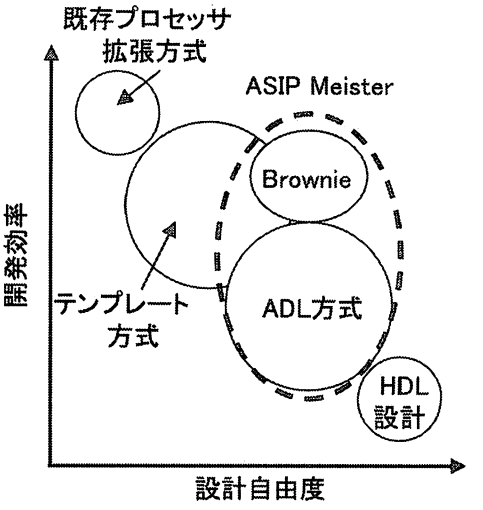

図 1　ASIP 開発方式と Brownie の位置づけ

そこで本研究では，ASIP Meister を利用した ASIP 開発を効率化するため，ベースプロセッサ・ファミリ Brownie を提案する．ASIP の開発方式と Brownie の位置づけを図 1 に示す．Brownie は ASIP Meister の開発効率を向上させるベースプロセッサである．ASIP Meister と Brownie を併用することで，図 1 中の破線で示したように，ADL 方式の設計自由度を保ったまま，テンプレート方式並みの開発効率を実現できる．

本稿の構成は以下の通りである．2 節で ASIP Meister の概要を説明し，3 節で Brownie の設計思想およびアーキテクチャの概要に関して説明する．4 節では Brownie を用いた命令の拡張例を示し，最後に 5 節で本稿をまとめる．

2. ASIP Meister

本節では ASIP 開発ツール ASIP Meister [2] を概説する．

ASIP Meister は ASIP の HDL 記述およびソフトウェア開発環境をプロセッサ仕様記述から自動生成するツールである．プロセッサ仕様記述は，主にプロセッサのパイプライン段数，遅延分岐ステージ数，リソース，命令タイプ，オペコード定義，各命令のマイクロ動作記述 [5] からなり，設計者はこのプロセッサ仕様記述を用いてプロセッサのアーキテクチャを定義する．マイクロ動作記述とは，命令のデータフローを記述する ADL である．設計者は ASIP Meister 上でマイクロ動作記述を用い，各命令のデータフローを個々に記述しながらプロセッサを設計できることから，他の ASIP 開発ツールに比べて柔軟に ASIP を開発できる．加えて，ASIP Meister を用いることでプロセッサの仕様を容易に変更できることから，ASIP の設計空間探索

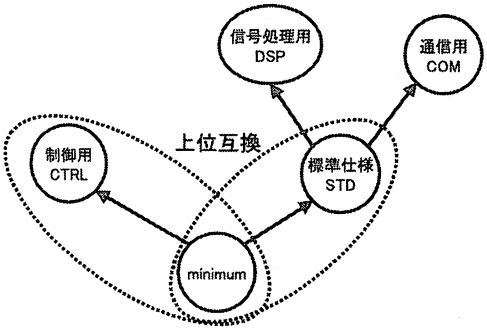

図 2 Brownie ファミリ

にも利用可能であり，最適なアーキテクチャを短期間で見つけられる．

ASIP Meister を利用することで ASIP 開発期間を大きく短縮できるが，依然として基本プロセッサの仕様策定，開発および検証は設計者自身が行わなければならない．ASIP の設計者は，本来，拡張命令の設計に注力するべきであり，このような基本プロセッサの開発期間を短縮することで，より一層の効率的な ASIP 開発が期待できる．

3. ベースプロセッサ・ファミリ Brownie の提案

ベースプロセッサを用いた ASIP 開発には，開発効率は高いが設計の柔軟性が低いという問題がある．そこで本研究ではベースプロセッサ・ファミリ Brownie を提案する．Brownie と ASIP Meister を併用することにより，開発効率と設計自由度を兼ね備えた ASIP 設計を実現できる．

Brownie は多数の命令セット・シリーズで構成され，さらにそれぞれの命令セット・シリーズの中に，ビット幅の異なるベースプロセッサ群が存在する．このように，多数のベースプロセッサ候補を用意することで，設計者は拡張命令の実装時にベースプロセッサとして最適なプロセッサを選択することできる．また，このように ASIP Meister を併用することで，命令の追加・変更だけではなく，ビット幅，パイプライン・アーキテクチャ，割り込み処理，内部リソースなども容易に変更することができる．

以降の節では，Brownie の設計思想および Brownie プロセッサのアーキテクチャを説明する．

3.1 Brownie の設計思想

Brownie は ASIP Meister 上で高い拡張性を持つように設計されたベースプロセッサ・ファミリである．図 2 は Brownie ファミリの概念を示している．

Brownie ファミリには目的に応じた命令セット・シリーズがあり，制御用途向け命令セットを持った BrownieCTRL シリーズ，一般用途向け命令セットを持った BrownieSTD シリーズ，信号処理用途向けを持った BrownieDSP シリーズなどがある．Brownie の各命令セット・シリーズ間には命令セット互換性がある．たとえば，BrownieDSP シリーズは BrownieSTD シリーズの命令セットの上位互換にあたる．このような命令セット互換性は後に述べるソフトウェア開発環境の構築に重要である．これらの命令セット・シリーズの中で，さらにデータバス・ビット幅が異なるプロセッサが存在しており（STD シリーズならば 16，32，64bit），設計者はこれらのベースプロセッサ候補の中から開発のベースに最も適したプロセッサを選択する．

表 1 BrownieSTD32 のアーキテクチャ概要

項目	内容
パイプライン	4 段パイプライン
メモリ・アーキテクチャ	ハーバード・アーキテクチャ
命令語長	32bit
データ語長	32bit
レジスタ数	32bit x 32 本
命令数	39 命令
遅延分岐	なし
外部割込み	あり
内部割込み	あり

Brownie のもう一つの特徴として，ASIP Meister との高い親和性があげられる．すなわち，設計者は Brownie と ASIP Meister を併用することで直ちに拡張命令の設計を始められ，アーキテクチャやリソースの変更を伴うような複雑な拡張命令の設計も容易に行える．また，Brownie はコンパイラ作成に必要最低限の命令セットしか実装しておらず，ASIP の命令セット拡張を妨げない．

ベースプロセッサへの要求はこのようなアーキテクチャのバリエーションやベースプロセッサのカスタマイズの柔軟性・容易性だけではない．ASIP 設計のもうひとつの大きな課題として，ソフトウェア開発環境の作成がある．拡張命令を追加した ASIP を設計してもコンパイラ等のソフトウェア開発環境がなければ実際のシステムで利用するのは困難である．Brownie は単純な命令セット，シンプルなアーキテクチャをもつことや，Brownie ファミリ内での命令セット互換性を保つことで，コンパイラの拡張のしやすさも重視している．命令セットの単純化はプロセッサの性能向上や，コンパイラの作成上重要である．BrownieSTD よりも上位の Brownie には GNU 開発環境が開発されており，コンパイラの拡張等が容易に行える．筆者らは ASIP のための GNU ソフトウェア開発環境を自動的に構築するため，Brownie 用の GNU 環境を拡張する形で自動的にソフトウェア開発環境を構築できるシステムを開発中である．

このように Brownie は ASIP のためのベースプロセッサとして十分な要求を満たしたベースプロセッサであり，ASIP Meister と組み合わせて利用することで高い開発効率と設計自由度を実現できる．

3.2 BrownieSTD32

本節では，Brownie ファミリの中で最も基本的なアーキテクチャを持つ 32bit 版 BrownieSTD，BrownieSTD32 について説明する．

表 1 に BrownieSTD32 アーキテクチャの概略を示す．BrownieSTD32 は 32bit の命令語長，データ語長を持つ RISC プロセッサ [3] であり，4 段パイプライン・アーキテクチャで構成される．BrownieSTD32 は 3.1 節で述べたように，命令セット拡

```
variable{
  wire [31:0] result;
}
stage 1{
  Fetch()
}
stage 2{
  GPRDoubleRead(rs1, rs2)
}
stage 3{
  wire [31:0] tmp_result1;
  wire [31:0] tmp_result2;
  wire [15:0] result_upper_half;
  wire [15:0] result_lower_half;
  <tmp_result1, tmp_result2>
                = MADD1.madd(source1, source2);
  result_upper_half = tmp_result1[15:0];
  result_lower_half = tmp_result2[15:0];
  result = <result_upper_half,result_lower_half>;
  ForwardDataFromEXE(rd, result)
}
stage 4{
  WriteBack(rd, result)
  ForwardDataFromWB(rd, result)
}
```

図 3 バタフライ演算命令の例

張のしやすさ，およびコンパイラの作りやすさを考慮し，39 個という少ない命令セットを実装している．この 39 個の命令セットはコンパイラを作成する際に最低限必要であり，かつコンパイラが効率的なコードを生成できるように選定された命令セットである．なお，BrownieSTD32 は浮動小数点命令をサポートしていない．

4. 拡張命令追加例

本節では Brownie を用いた ASIP 設計例を示す．

この例では ASIP Meister を利用した ASIP 設計経験のある設計者が，以下の 4 種の命令を追加した．

- バタフライ演算命令
- DCT 演算命令
- 複合演算命令
- SIMD 命令

それぞれの場合について，Brownie に対して ASIP Meister を用いて命令拡張を行い，その作業時間を調べた．

なお今回の実験では，全ての命令追加に関して外部仕様およびリソースは全て提供されている状態からの設計を行った．したがって，具体的な作業時間は，主に追加命令のオペコード割付，リソースの追加・変更，およびマイクロ動作記述の作成に要した時間である．

4.1 バタフライ演算命令追加

JPEG アプリケーションでは頻繁にバタフライ演算が実行される．したがって，バタフライ演算を専用ハードウェアを用いて高速実行できれば，全体の処理時間短縮が期待できる．

本拡張例では，BrownieSTD32 を基に，3 種類のバタフライ

```
variable{
}
stage 1{
  Fetch()
}
stage 2{
  GPRDoubleRead(rs1, rs2)
}
stage 3{
  null = DMAU.dct(source2, source1);
}
stage 4{
}
```

図 4 DCT 演算命令の例

演算を拡張命令で実現した．バタフライ演算を実現するためには，定数乗算後に加減算を行う専用ハードウェアが必要であるが，本拡張例では，既設計のバタフライ演算ハードウェアを用いて設計を行った．

$$out1 = in1 * W_1 - in2 * W_2$$

$$out2 = in1 * W_1 + in2 * W_2$$

例として，上記のようなバタフライ演算を行う命令を追加するためのマイクロ動作記述を図 3 に示す．上式において，$out1$，$out2$ は出力，$in1$，$in2$ は入力を示している．マイクロ動作記述はパイプライン・ステージごとに記述する．`stage n` はパイプライン・ステージを示している．`Fetch()`，`GPRDoubleRead()`，`WriteBack()`，`ForwardDataFromEXE()`，および `ForwardDataFromWB()` はマクロであり，それぞれ命令フェッチ，レジスタフェッチ，レジスタ書き戻し，フォワーディングの機能を実現する．これらマクロは Brownie の設計記述に既に含まれている．Brownie の設計済みマクロを利用することで，拡張命令の本質とは関係ない部分の記述負荷を軽減している．バタフライ演算は `stage 3` でバタフライ演算リソース `MADD1` を利用して実現されている．なお，`MADD1` は入力データの下位 16bit を計算する 16bit 演算器であるので，出力の 16bit データを結合し，32bit データとしてレジスタファイルに書き戻している．

4.2 DCT 演算命令追加

JPEG アプリケーションでは頻繁にバタフライ演算が実行されることは 4.1 節で述べたが，本拡張例では，演算粒度をさらに荒くし，DCT 演算を 1 命令で行うような命令拡張を行った．この場合，DCT 演算ハードウェアが必要となる．

本拡張例では，BrownieSTD32 を基に，DCT 演算および逆 DCT 演算を拡張命令で実現した．本拡張例では，既設計の DCT 演算ハードウェアを用いて設計した．

例として，DCT 演算を行う命令を追加するためのマイクロ動作記述を図 4 に示す．DCT 演算ハードウェアはメモリ・アクセスユニットの拡張型として実現されており，入力メモリアドレスから 8 つのデータを連続で読み出し．DCT を行った後に出力メモリアドレスに 8 つ連続で書き戻す設計となっている．

```
variable{
}
stage 1{
  Fetch()
}
stage 2{
  GPRTripleRead(rs1, rs2, rs3)
}
stage 3{
  MULExec(mul, source1, source2)
  ALUExec(add, mul_result, source3)
  ForwardDataFromEXE(rd, alu_result)
}
stage 4{
  WriteBack(rd, alu_result)
  ForwardDataFromWB(rd, alu_result)
}
```

図 5 MAC 演算命令の例

したがって，データのフォワーディングさえも必要なく，マイクロ動作記述は非常に簡単に記述されている．

なお，Brownie に含まれているフォワーディング・マクロを用いることで，命令のフォワーディング機能の有無についても柔軟かつ容易に実現できる．

4.3 複合演算命令追加

FIR フィルタ・アプリケーションでは，乗算後に加算を行うような MAC 命令や，乗算後にシフトする命令などが多用されることから，そのような拡張命令を追加することで高速化が期待できる．この場合，新たな演算ハードウェアを必要とせず，既存リソースの組み合わせとして実現可能である．

本拡張例では，MAC 演算および MARS（Multiply Arithmetic Right Shift）演算命令を追加する．この際，新しいリソースは追加せず，Brownie に含まれるリソースを組み合わせることで拡張命令の設計を行った．

例として，MAC 演算を行う命令を追加するためのマイクロ動作記述を図 5 に示す．MAC 演算は Brownie に含まれる乗算器と ALU を接続することで実現可能である．ただし，本拡張命令では 3 つのデータを同時に読み出す必要があるため，レジスタファイルの同時読み出しポート数を 2 から 3 に拡張し，3 オペランド命令タイプを追加した．レジスタファイルの仕様は ASIP Meister 上で容易に変更可能である．`MULExec()` および `ALUExec()` は，それぞれ乗算器と ALU を使った演算を容易に記述するマクロであり，既に Brownie に含まれている．

4.4 SIMD 命令追加

画像処理などでは，複数のデータに同様の演算を行う場合が多く発生する．そのような処理を高速化する命令として SIMD 命令がよく知られている．

SIMD 命令を追加する場合，最低限 16bit データを 4 つ同時に演算できることが求められるため，SIMD 命令拡張のためには，16bit × 4 = 64bit のデータバス幅が要求される．そこで本拡張例では，上記 3 例とは異なり，64bit 版 BrownieSTD，BrownieSTD64 を基に，4 並列 SIMD 演算 4 種類（加算，減算，

```
variable{
  wire [63:0] result;
}
stage 1{
  Fetch()
}
stage 2{
  GPRDoubleRead(rs1, rs2)
}
stage 3{
  wire [15:0] src1_1;
  wire [15:0] src1_2;
  wire [15:0] src1_3;
  wire [15:0] src1_4;
  wire [15:0] src2_1;
  wire [15:0] src2_2;
  wire [15:0] src2_3;
  wire [15:0] src2_4;
  wire [15:0] res1;
  wire [15:0] res2;
  wire [15:0] res3;
  wire [15:0] res4;
  wire [3:0] flag1;
  wire [3:0] flag2;
  wire [3:0] flag3;
  wire [3:0] flag4;
  src1_1 = source1[63:48];
  src1_2 = source1[47:32];
  src1_3 = source1[31:16];
  src1_4 = source1[15:0];
  src2_1 = source2[63:48];
  src2_2 = source2[47:32];
  src2_3 = source2[31:16];
  src2_4 = source2[15:0];
  <res1, flag1> = SIMDALU0.add(src1_1, src2_1);
  <res2, flag2> = SIMDALU1.add(src1_2, src2_2);
  <res3, flag3> = SIMDALU2.add(src1_3, src2_3);
  <res4, flag4> = SIMDALU3.add(src1_4, src2_4);
  result = <res1, res2, res3, res4>;
  ForwardDataFromEXE(rd, result)
}
stage 4{
  WriteBack(rd, result)
  ForwardDataFromWB(rd, result)
}
```

図 6 SIMDADD 演算命令の例

乗算，シフト）を拡張命令で実現した．BrownieSTD シリーズには 16bit，32bit，64bit 版が存在するので，このように目的に応じて最も適切なベースプロセッサを選択でき，ASIP を効率的に開発できる．本拡張例では，BrownieSTD64 に 16bit ALU を 4 つ，16bit 乗算器を 2 つ，16bit シフタを 4 つ追加した．

例として，SIMD 加算演算を行う SIMDADD 命令を追加するためのマイクロ動作記述を図 6 に示す．SIMDADD 命令は入力データを 4 分割し，4 つの ALU で演算し，結合しなおして書き戻す処理として記述される．データの切り出し，結合等はマイクロ動作記述上で容易に実現できる．

4.5 拡張命令の追加時間

表 2 に拡張命令追加に要した時間をまとめる．いずれの命令追加においても，作業時間は 1 時間未満と短く，Brownie と ASIP Meister を併用した開発が有効であることがわかる．

表 2 拡張命令の追加に要した時間

拡張命令	時間
バタフライ演算命令 (3 種)	30 分
DCT 演算命令 (2 種)	20 分
複合演算命令 (2 種)	45 分
SIMD 命令 (4 種)	30 分

5. ま と め

本研究では ASIP 開発効率を向上させるため，ベースプロセッサ Brownie を提案した．Brownie は ASIP Meister と親和性の高いベースプロセッサであり，Brownie と ASIP Meister を併用することで，ASIP 設計者が直ちに設計を始められ，拡張命令の開発に注力できる．拡張命令設計例ではバタフライ演算命令，DCT 演算命令，複合命令，SIMD 命令の 4 種の命令追加を行った．命令拡張に要した時間はいずれも 1 時間以内で完了したことから，Brownie は ASIP を柔軟に，かつ短期間で開発するために有効であることを確認した．

今後の課題として，命令拡張されたプロセッサの GNU 開発環境の自動生成が挙げられる．

謝辞 本研究は大阪大学大学院情報科学研究科 集積システム設計学講座を通し，エイシップ・ソリューションズ株式会社の協力で行われたものである．本研究を進めるにあたり，貴重な助言を頂いた関西学院大学情報科学科 石浦菜岐佐 教授に感謝する．

文 献

[1] A. Hoffman, H. Meyer, and R. Leupers, "Architecture exploration for embedded processors with LISA, " Kluwer Academic Publishers, Boston, 2002.

[2] M. Imai, Y. Takeuchi, A. Shiomi, J. Sato, and A. Kitajima, "An Application Specific Processor Development Environment: ASIP Meister, " In Proceedings of ICICT'05, 2002.

[3] J.L. Hennesy and D.A. Patterson, "Computer Architecture: A Quantitative Approach, " Morgan Kaufmann Publishers, San Francisco, 2003.

[4] A. Mizuno, K.Kohno, R. Ohyama, T.Tokuyoshi, H.Uetani, H.Eichel, T.Miyamori, N.Matsumoto, M.Matsui, "Design methodology and system for a configurable media embedded processor extensible to VLIW architecture, " In Proceedings of the ICCD'02, 2002.

[5] M. Itoh, Y. Takeuchi, M. Imai, and A. Shiomi, "Synthesizable HDL generation for pipelined processors from a Micro-Operation Description, " IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, Vol. E83-A(3), pp. 394–400, 2000.

[6] R.E. Gonzalez, "Xtensa: a configurable and extensible processor, " IEEE Micro, Vol. 20, Issue 2, pp. 60–70, 2000.