社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

プラグイン方式に基づくBinutilsおよびGDBの自動リターゲッティング

多賀惣一朗† 久村 孝寛††,††† 石浦菜岐佐† 武内 良典††† 今井 正治†††

† 関西学院大学 〒669-1337 兵庫県三田市学園 2-1
†† 日本電気株式会社 〒216-8666 川崎市中原区下沼部 1753
††† 大阪大学 〒 565-0871 大阪府吹田市山田丘 1-5

あらまし 本稿では「プラグイン方式」に基づくソフトウェア開発ツールの自動リターゲッティング手法を提案する. プラグイン方式は, 命令セットの拡張部分に対する処理機能を持ったプラグインをツールチェーンに組み込むことにより, コンフィギュラブルプロセッサ用ソフトウェア開発ツールの命令セット拡張を実現する手法である. 本手法では, 命令やレジスタを一つも持たないプロセッサのツールチェーン (フレームワーク) を用意し, ここにターゲットプロセッサの全命令と全レジスタをプラグイン方式により追加することにより, コンフィギュラブルプロセッサに限定されないソフトウェア開発ツールの自動リターゲッティングを行う. 本手法は XML による簡潔な命令セットアーキテクチャの記述から, GNU Binutils および GDB の生成が可能である. 本手法を実装し, 16 ビットプロセッサ MeDIX-I を対象に適用実験を行った結果, 1475 行のアーキテクチャ記述から GNU Binutils および GDB を生成し, アセンブラ, 逆アセンブラ, リンカ, シミュレータ, デバッガの機能を確認することができた.

キーワード ソフトウェア開発ツール, リターゲッティング, Binutils, GDB, プラグイン方式

Automatic Retargeting of Binutils and GDB Based Plug-in Method

Soichiro TAGA†, Takahiro KUMURA††,†††, Nagisa ISHIURA†, Yoshinori TAKEUCHI†††, and Masaharu IMAI†††

† Kwansei Gakuin University 2-1 Gakuen, Sanda City, Hyogo, 669-1337, Japan
†† NEC Corporation Shimonumabe 1753, Nakahara-ku, Kawasaki City, 216-8666 Japan
††† Osaka University Yamadaoka 1-5, Suita City, Osaka, 565-0871 Japan

Abstract This paper proposes a method of automatic retargeting of software development tools based on a *plug-in method*. The plug-in method is originally designed to augment a base processor's toolchain by adding plug-in's, which are software components that handle extra instructions and registers. Our method attempts to retarget a toolchain to an arbitrary processor based on the plug-in method, by embedding functions to deal with all its instructions and registers into a framework, which is a toolchain for a processor with no instructions nor registers. GNU Binutils and GDB are autogenerated from a simple XML specification of the target instruction set architecture. An experimental tool based on this method successfully generated, from 1475 lines of architecture description of a 16bit processor MeDIX-I, Binutils and GDB which are capable of assembling, disassembling, linking, simulating, and debugging.

Key words software development tool,retargeting, Binutils, GDB, plug-in method

1. はじめに

特定用途向けプロセッサ (ASIP; application specific instruction set processor) はターゲットアプリケーション向けに最適化されたアーキテクチャによる高い性能電力比と, システムの仕様変更やバグ修正に対応できる柔軟性を兼ね備えており, マルチメディア処理や無線通信処理などの組込みシステム用プロセッサとして利用されるようになってきている [3]. 近年, ASIP の設計を効率化するために, 専用のツールも種々開発されている [4], [14].

ASIP の活用において, 設計ツールとともに重要になるのが, ソフトウェア開発ツールである. 特に, システムの性能評価やアーキテクチャ探索のために, アーキテクチャ設計段階の早期にソフトウェア開発ツールを用意することは, 非常に重要とな

る．このため, ターゲットプロセッサの命令セットなどの情報を入力し, ターゲットプロセッサに対応したソフトウェア開発ツールを自動生成 (自動リターゲッティング) する研究が数多く行われている．

自動リターゲッティングの研究の多くは, 独自のコンパイラ, アセンブラ, シミュレータなどに基づいてソフトウェア開発ツールを生成する [9], [12]．一方, GNU ツールチェーンに基づいてソフトウェア開発ツールを生成する手法もいくつか提案されている [10], [11], [13]．GNU ツールチェーンは広く使われているだけでなく, 利用者がカスタマイズやチューニングを行って再配布を行うことができる．しかし, GNU ツールチェーンに基づく従来のツール生成手法では, 主にアセンブラなどの GNU Binutils やシミュレータのリターゲッティングを行っており, GNU GDB のデバッガ機能のリターゲッティングは行われていなかった．

そこで本稿では, 「プラグイン方式」に基きソフトウェア開発ツールを自動生成する手法を提案する．プラグイン方式は, コンフィギュラブルプロセッサ用ソフトウェア開発ツールの命令セット拡張 [1], [2] で使われている手法であり, 追加命令の処理機能を持ったプラグインをツールチェーンに組み込む手法である．本手法ではプラグイン方式を利用し, ターゲットプロセッサのすべての命令を扱うプラグインをツールチェーンに組み込むという方法によりソフトウェア開発ツールの自動リターゲッティングを行う．

本手法を実装し,16 ビットプロセッサ MeDIX-I [5], [6] を対象に GNU Binutils および GDB の自動リターゲッティングを行った．その結果, 生成したアセンブラ, 逆アセンブラ, リンカ, シミュレータ, デバッガの機能を確認することができた．

2. 関 連 研 究

ソフトウェア開発ツールの自動生成を行う手法には, 既存のツールチェーンの命令セットを拡張する手法と, ツールチェーンの命令セットをすべて変更する (リターゲッティングを行う) 手法がある．

既存のツールチェーンの命令セットを拡張する手法では, 基となるプロセッサ (ベースプロセッサ) の命令に対応したツールチェーンの資源を利用することができ, コンパイラなどの複雑なツールを生成することができる．Tensilica 社の Xtensa [14], [15] 向けのソフトウェア開発ツールを生成するシステムでは, レジスタ数や演算器や命令セットをカスタマイズ可能であり, 拡張した Xtensa のための GNU Binutils, GDB, シミュレータ, コンパイラを生成できる．しかし, ベースプロセッサは Xtensa に限られ, 新たなリロケーションタイプの追加については記載されていない [15]．プラグイン方式に基づく命令セット拡張 [1], [2] では, ベースプロセッサに命令やレジスタを追加したターゲットプロセッサ用の GNU Binutils, GDB, GCC の生成に加えて, 新たなリロケーションタイプの追加なども行うことができる．しかし, これら既存のツールチェーンを拡張する研究では, 新しい命令セットアーキテクチャの ASIP のソフトウェア開発ツールを生成することはできない．

ツールチェーンのリターゲッティングを行う手法では, 新しい命令セットに対応したソフトウェア開発ツールを生成することができる．リターゲッティングされるツールチェーンに関しては, 独自のコンパイラ, アセンブラ, シミュレータなどを生成するものと, GNU ツールチェーンを利用するものがある．ACE 社の CoSy [16] や Coware 社の LISATek [17] は, 独自のソフトウェア開発ツールのリターゲッティングを行う．他にも [9], [12] は独自のソフトウェア開発ツールに基づいている．これらはコンパイラなどの複雑なツールの生成を行うことができるが, ライセンスの制約などから, 生成されたツールのカスタマイズや配布が強く限定される．

GNU ツールチェーンのリターゲッティングを行う研究には, CGEN, rbinutils, ArchC などがある．RedHat 社の CGEN [10] は GNU Binutils や GDB の移植を容易にするためのオープンソースソフトウェアである．CGEN は GNU Binutils 全体をリターゲッティングするものではなく, 命令のエンコードやデコードを行うライブラリ opcode や命令実行の関数群だけをプロセッサ仕様記述から生成するので, デバッガを生成することはできない．文献 [11] の rbinutils は仕様記述から GNU Binutils を生成するツールである．rbinutils が生成可能なのは GNU Binutils だけで, シミュレータやデバッガを生成することはできない．ArchC は仕様記述から GNU Binutils やシミュレータを生成するオープンソースソフトウェアである．文献 [13] において, ArchC を使って複数の CPU で GNU Binutils のリターゲッティングとシミュレータ生成が可能であることが報告されているが, デバッガを生成することはできていない．

3. プラグイン方式に基づいたソフトウェア開発ツールの命令セット拡張 [1], [2]

ソフトウェア開発ツールの命令セット拡張のための一手法であるプラグイン方式の概念を図 1 に示す．プラグイン方式は, 追加命令の処理機能を持ったプラグインをツールチェーンに組み込むことにより, ソフトウェア開発ツールの命令セット拡張 (命令やレジスタの追加) を実現する．ツールジェネレータは, ターゲットプロセッサの拡張部分の仕様記述からプラグインを自動で生成し, ベースプロセッサ用のツールチェーン (ベースツール) へ組み込むことにより, ターゲットプロセッサ用のツールチェーンを生成する．

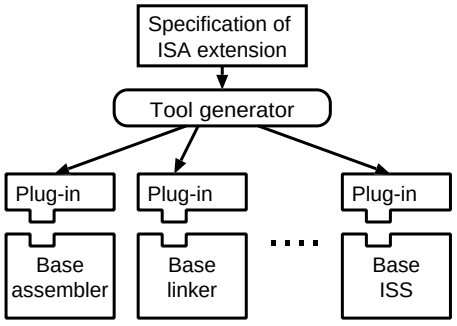

図 1: プラグイン方式の概念

ターゲットプロセッサ用ツールチェーンにおける処理の流れを図 2 に示す．命令は, ベースツールの処理パスか, プラグインによる処理パス, のいずれかで処理される．まず, ターゲットプロセッサ用ツールは, 与えられた命令が追加命令であるか否かを判定する．続いて, その命令が追加命令であればプラグインによる処理パスで処理し, そうでなければベースツールの処理パスで処理する．ベースツールは, 予めプラグインを接続できるよう修正しておく必要がある．

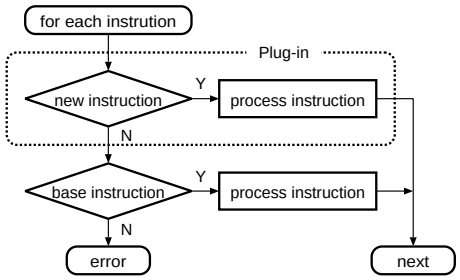

図 2: 命令セット拡張におけるツールチェーンの処理の流れ

ツールジェネレータの入力は, ターゲットプロセッサの仕様を記述した XML 文書と, 追加命令セットの命令動作記述から成る．XML 文書の記述例を図 3 に, 命令動作記述の例を図 4 に示す．入力記述は, 拡張命令セットや拡張レジスタの情報と, ベースプロセッサのアーキテクチャ情報で構成される．図 3 の <insn> タグ (10〜31 行目) が MYADD という追加命令の仕様を表し, ニモニック, 構文, フィールド情報, 入出力情報などを記述する．図 4 は追加命令 MYADD の動作記述を表し, C 言語で命令の動作を記述する．REG_read32 (3, 4 行目), REG_write32 (8 行目) はレジスタアクセスのために用意された関数である．<insn> と命令動作記述は, 追加命令の数だけ記述することになる．図 3 の 6, 7 行目の <register_bank> タグが MYGPR という追加レジスタの仕様を表す．base 属性を false にすることにより追加レジスタを表し, レジスタタイプや個数を記述する．図 3 の 2〜5, 8 行目がベースプロセッサのアーキテクチャ情報である．アーキテクチャ情報としては, ベースプロセッサの名前, レジスタに関する情報, 命令長などを記述する．

```
 1:  <Processor>
 2:  <nickname>cpu</nickname>
 3:  <register_type length="32">GPR_type</register_type>
 4:  <register_bank type="GPR_type" size="32"
 5:       prefix="R" base="true">GPR</register_bank>
 6:  <register_bank type="GPR_type" size="8"
 7:       prefix="MR" base="false">MYGPR</register_bank>
 8:  <insn_length>32</insn_length>
 9:
10:  <insn>
11:    <mnemonic>MYADD</mnemonic>
12:  <syntax>MYADD %reg1, %reg2, %reg3</syntax>
13:  <field type="GPR" length="5">reg1</field>
14:  <field type="opcode" value="0b11_1111"
15:       length="4">opc0</field>
16:  <field type="GPR" length="5">reg2</field>
17:  <field type="opcode" value="0b1111001000"
18:       length="11">opc1</field>
19:  <field type="GPR" length="5">reg3</field>
20:  <input>
21:    <operand type="GPR" width="32">reg1</operand>
22:    <operand type="GPR" width="32">reg2</operand>
23:  </input>
24:  <output>
25:    <operand type="GPR" width="32">reg3</operand>
26:  <output>
27:  <description>
28:  This instruction calculates the sum of the contents of
29:  registers reg1 and reg2, and stores the sum to reg3.
30:  </description>
31:  </insn>
32:
33:  <behavior>cpu-isa.c</behavior>
34:  </Processor>
```

図 3: 命令セット拡張の仕様記述の例

```
1:  behavior (MYADD)
2:  {
3:    int32_t val_reg1 = REG_read32 (GPR,reg1);
4:    int32_t val_reg2 = REG_read32 (GPR,reg2);
5:    int32_t val_reg3;
6:
7:    val_reg3 = val_reg1 + val_reg2;
8:    REG_write32 (GPR,reg3,val_reg3);
9:  }
```

図 4: 命令動作記述の例

4. プラグイン方式に基づくソフトウェア開発ツールの自動リターゲッティング

4.1 概　　要

本稿では, 前節のプラグイン方式に基づいて新しい命令セットアーキテクチャのためのソフトウェア開発ツールの自動リターゲッティングを行う手法を提案する．本手法では命令もレジスタも持たないプロセッサ用のツールチェーンの「フレームワーク」を用意し, ターゲットプロセッサの全命令とレジスタをプラグイン方式により追加することによりこれを実現する．

本手法におけるツール生成の概念を図 5 に示す．ツールジェネレータはターゲットプロセッサの仕様記述からプラグインを自動で生成し, フレームワークにこれを組み込むことにより, ターゲットプロセッサ用のツールチェーンを生成する．フレームワークは, プラグイン方式に基づくソフトウェア開発ツール拡張手法におけるベースツールに対応するものであるが, 命令やレジスタの情報を持っていない．

フレームワークにおける処理の流れを図 6 に示す．フレームワークにおける各命令の処理では, 命令はすべてプラグインによって処理する．フレームワークは本手法で扱える全ての命令セットアーキテクチャに共通であるため, ユーザがフレームワークを用意する必要はない．

本手法におけるプラグインは, ターゲットプロセッサのすべ

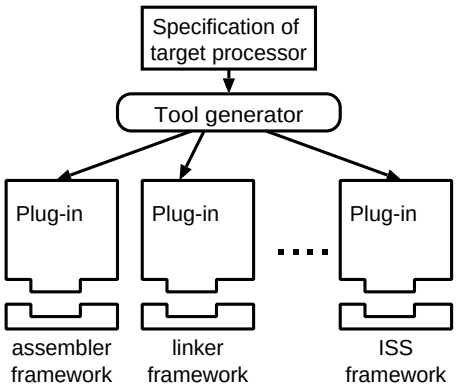

図 5: プラグイン方式に基づくツールの自動リターゲッティング

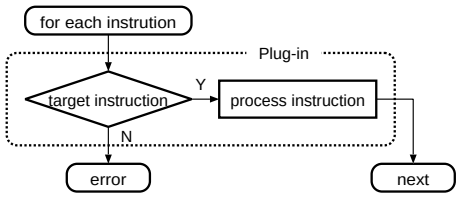

図 6: 本手法における各ツールのフレームワークの処理の流れ

ての命令に対して, アセンブル/逆アセンブル/命令エンコード/命令デコード/命令実行等を行う. それぞれの処理において, プラグインはターゲットプロセッサのアーキテクチャ情報を利用する.

4.2 プロセッサ仕様記述

ツールジェネレータの入力には, プラグイン方式に基づいたソフトウェア開発ツールの命令セット拡張の場合と同様に, ターゲットプロセッサの全ての命令とレジスタ, およびその他のアーキテクチャ情報を記述する.

命令とレジスタの情報は, 命令セット拡張の場合 (図 3, 図 4) と同様, それぞれ <insn>,<register_bank> などを用いて記述する.

アーキテクチャ情報に関しては, ソフトウェア開発ツールの命令セット拡張におけるベースプロセッサのアーキテクチャ情報に加えて, 以下の項目が必要である.

（1） マジックナンバー

オブジェクトの生成, 読み込みを行う際に利用する.

（2） エンディアン

命令のエンコード, デコードを行う際に利用する.

（3） アドレス語長

リロケーションを行う際に利用する.

（4） シミュレータ (run) を終了させるトラップ命令

シミュレータの終了判定に利用する.

（5） デバッガ (GDB) のブレイクを引き起こすトラップ命令

デバッガでブレイクポイント機能を実現するために利用する.

4.3 デバッガ機能の実現

GNU GDB では次の処理によってブレイクポイント機能を実現している.

（1） ブレイクポイントに指定された命令アドレスに存在する命令を退避し, その命令アドレスにトラップ命令を書き込む.

（2） トラップ命令を実行するとシミュレータを一時停止し, 退避しておいた命令をその命令アドレスに書き戻す.

（3） 実行再開の際には 1 命令だけステップ実行してからその命令アドレスにある命令を再び退避しトラップ命令を書込む.

この処理の大半は機械独立になっており, GDB の機械依存部分のみを書き換えることにより, ブレイクポイント機能をリターゲッティングすることができる. 本手法では以下の処理を行うことでブレイクポイント機能を実現する.

（1） GDB がトラップ命令のコードをわかるようにする

プラグインで GDB のデータ構造に命令コードを格納する.

（2） トラップ命令実行時にシミュレータを一時停止する

プラグインが命令を実行する際に, トラップ命令かどうかの判定を行い, GDB 内で利用されているトラップ用の関数を起動する.

トラップ命令は, コンパイラやユーザが使用する命令であってはならず, かつ全命令の中で語長がもっとも短くなければならない.

シミュレータの終了のためのトラップ命令に関しても, (2) と同様にプラグインが判定を行い, GDB 内で利用されている終了用の関数を起動する.

4.4 プロセッサ仕様記述の記述例

16 ビットプロセッサ MeDIX-I [5], [6] の仕様を記述した XML 文書の一部を図 7 に示す. ターゲットプロセッサの名前は <nickname> タグ (2 行目) に記述し, 生成するツールチェーンの名前に反映する. エンディアンの情報は <endian> タグ (4 行目) に big または little で記述する. アドレス語長は <bits_per_address>(5 行目) タグに, マジックナンバーは <magic_number> タグ (6 行目) に記述する. レジスタの情報はプラグイン方式に基づくソフトウェア開発ツールの命令セット拡張と同じように <register_type> タグ (8〜9 行目), <register_bank> タグ (10〜13 行目), <register_alias> タグ (14〜18 行目) に記述する. 命令語長は <insn_length>(19 行目) に記述する. 各命令の仕様はプラグイン方式に基づくソフトウェア開発ツールの命令セット拡張と同じように <insn> タグ (21〜34 行目) に記述する. シミュレータを終了させるトラップ命令, デバッガのブレイクを引き起こすトラップ命令は, それぞれ <insn> タグの iss_halt_insn 属性 (21 行目), iss_break_insn 属性 (22 行目) を true にすることにより指定する.

5. 実装と実験

プラグイン方式に基づくソフトウェア開発ツールの命令セット拡張ツール [1], [2] を拡張し, 提案手法を実装した. 自動リターゲッティングツールは C++で実装し, Linux 系 OS や Cygwin 上で動作する. ターゲットプロセッサの仕様を記述した XML 文書と命令動作記述を入力として, ターゲットプロセッサに対応したツールチェーンのソースコードを生成する. 実装の対

```
 1: <Processor>
 2: <nickname>medix1</nickname>
 3:
 4: <endian>big</endian>
 5: <bits_per_address>16</bits_per_address>
 6: <magic_number>0x1010</magic_number>
 7:
 8: <register_type length="16">GENERAL_type
 9:    </register_type>
10: <register_bank type="GENERAL_type" size="16"
11:    prefix="r">GPR</register_bank>
12: <register_bank type="GENERAL_type" size="1"
13:       prefix="pc">PC</register_bank>
14: <register_alias>
15:   LR=r3
16:   SP=r5
17:   FP=r6
18: </register_alias>
19: <insn_length>16</insn_length>
20:
21: <insn iss_halt_insn="false"
22:       iss_break_insn="false">
23:   <mnemonic>ADD</mnemonic>
24:   <syntax>ADD %rd,%rs</syntax>
25:   <field type="opcode" length="4"
26:         value="0b1000">opcode0</field>
27:   <field type="opcode" length="4"
28:         value="0b0000">opcode1</field>
29:   <field type="GPR" length="4">rd</field>
30:   <field type="GPR" length="4">rs</field>
31:   <description>
32:     GPR(rd) = GPR(rd) + GPR(rs);
33:   </description>
34: </insn>
35: .....
36: <behavior>medix1-isa.c</behavior>
37: </Processor>
```

図 7: ターゲットプロセッサの仕様記述の例

象としたツールチェーンは, GNU Binutils-2.16.1, GDB-6.4 である.

ツールチェーンの元となるフレームワークは, RISC 型 32bit プロセッサ BrownieSTD32 [7] 用の Binutils-2.16.1, GDB-6.4 からプロセッサ固有の情報を削除し, プラグインを組み込めるように修正することにより作成した.

生成したツールチェーンは, リトル/ビッグエンディアン, 即値オペランドを持つ命令のリロケーション, 絶対/相対分岐命令に対応している. 生成したツールチェーンを使って, アセンブル, 逆アセンブル, リンク, シミュレータによる実行, デバッガによる実行を行うことができる. リンクを行う際には, GNU LD で利用できる形式のリンカスクリプトを指定する. デバッガでの実行では, ブレイクポイント機能, 命令トレース機能を使うことができる. 命令語長は 1 バイトから 8 バイトまで可変語長に対応している.

適用実験として, 4.3 節で述べた 16 ビットプロセッサ MeDIX-I [5], [6] の命令セット (48 命令) の仕様を記述した XML 文書と命令動作記述を用意し, Binutils および GDB を生成した. 実験は Ubuntu8.04, インテル Core2Duo T7200 2GHz, メモリ 1GB の環境で行った. 実験の結果を表 1 に示す. XML 文書は 815 行, 命令動作記述は 660 行となった. これらを入力として, MeDIX-I 用ツールチェーンのソースコードを出力するのに要した時間は約 0.8 秒であり, このソースコードからツールチェーンをビルドするのには約 339.5 秒を要した. C コンパイラ用テストスイート testgen テストスイート [8] を, 別途開発された MeDIX-I 用 GCC を使ってコンパイルすることにより, MeDIX-I のアセンブリプログラム 2343 個をテストプログラムとして用意した. 本手法で生成したツールチェーンによって, これらのアセンブリプログラムをアセンブル, リンク, 実行し, すべてのプログラムについて正しい出力が得られることを確認した. また, いくつかのプログラムにおいて, 逆アセンブル, デバッガでの実行 (ブレイクポイント機能の利用, 命令トレース機能の利用, レジスタやメモリの内容の表示) が正しく行えることを確認した.

表 1: MeDIX-I プロセッサへの適用結果

XML 文書	815 行
命令動作記述	660 行
ソースコード出力	約 0.8 秒
ツールチェーンのビルド	約 339.5 秒

6. むすび

本稿ではプラグイン方式に基づくソフトウェア開発ツールの自動リターゲッティング手法を提案した. 簡潔なターゲットプロセッサの仕様記述と命令動作記述から, GNU Binutils および GDB のリターゲッティングを自動で行うツールを作成し, 本手法で生成したツールチェーンがアセンブル, リンク, 実行, デバッグを行えることを確認した.

今後の課題としては, 関数呼び出し規則を定義する機能, スタックフレームの解析機能, リンク時に適切な命令語長を選択する relax 機能, VLIW 型プロセッサへの対応などが挙げられる.

謝辞 本研究を進めるにあたり御助言・御検討を頂きました, 日本電気株式会社 システム IP コア研究所の新淳氏, 大阪大学大学院情報科学研究科の稗田拓路氏をはじめ今井研究室の諸氏, および関西学院大学の石浦研究室の諸氏に感謝します.

文　献

[1] T. Kumura, S. Taga, N. Ishiura, Y. Takeuchi, and M. Imai: “Software Development Tool Generation Method Suitable for Instruction Set Extension of Embedded Processors,” Trans. System LSI Design Methodology, Vol. 3, pp. 207–221 (Aug. 2010).

[2] 久村, 多賀, 石浦, 武内, 今井: “組込みプロセッサの命令セット拡張に適したソフトウェア開発ツール生成手法,” 信学技報, VLD2009-120, pp. 127–132 (Mar. 2010).

[3] M. Imai, Y. Takeuchi, K. Sakanushi, and N. Ishiura: “Advantage and Possibility of Application-domain Specific Instruction-set Processor (ASIP),” Trans. System LSI Design Methodology, Vol. 3, pp. 161–178 (Aug. 2010).

[4] Y. Takeuchi, K. Sakanushi, and M. Imai: “Generation of

Application-domain Specific Instruction-set Processors," in Proc. 2010 International SoC Design Conference, pp. 75–78 (Nov. 2010).

[5] 大沢, 近藤, 岩戸, 坂主, 武内, 今井: "カプセル型体内圧力測定システムのための小型低消費電力プロセッサへの通信誤り訂正方式の実装," 信学技報, VLD2010-6, pp. 49–54 (May 2010).

[6] Y. Takeuchi, H. Ohsawa, K. Kondo, H. Iwato, K. Sakanushi, and M. Imai: "Low Energy MDPC Implementation using Special Instructions on Application Domain Specific Instruction-set Processor," in Proc. APSIPA Annual Summit and Conference 2010 (Dec. 2010).

[7] 岩戸, 稗田, 田中, 佐藤, 坂主, 武内, 今井: " 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, "信学技報, VLD2007-92, pp. 19–24 (Nov. 2007).

[8] 内山, 引地, 石浦, 永松: "C コンパイラ用テストスイートおよびその生成ツール testgen," 信学技報, VLD2006-95, pp. 7–11 (Jan. 2007).

[9] S. Kobayashi, Y. Takeuchi, A. Kitajima, and M. Imai: "Compiler Generation in PEAS-III: an ASIP Development System," in Proc. International Workshop on Software and Compilers for Embedded Processors (Feb. 2001).

[10] CGEN, the Cpu tools GENerator,
see http://sources.redhat.com/cgen/.

[11] M. Abbaspour and J. Zhu: "Retargetable Binary Utilities," in Proc. 39th DAC, pp. 331–336 (June 2002).

[12] M. Hohenauer, H. Scharwaechter, K. Karuri, O. Wahlen, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, and H. van Someren: "A Methodology and Tool Suite for C Compiler Generation from ADL Processor Models," in Proc. DATE, pp. 1276–1281 (Feb. 2004).

[13] A. Baldassin, P. Centoducatte, and S. Rigo: "An Open-Source Binary Utility Generator," ACM Trans. Design Automation of Electronic Systems, Vol. 13, No. 2, Article 27 (Apr. 2008).

[14] R. E. Gonzalez: "Xtensa: A Configurable and Extensible Processor," in Proc. IEEE Micro, Vol. 20, Issue 2, pp. 60–70 (Mar.–Apr. 2000).

[15] Tensilica, Inc.: "Tensilica Instruction Extension (TIE) Language Reference Manual," (Nov. 2006).

[16] CoSy, see http://www.ace.nl/.

[17] LISAtek, see http://www.coware.com/.