2008－EMB－10 (4)
2008／11／27

MIPSシステムシミュレータSimMipsを活用した組込みシステム開発の検討

渡邉伸平[†1] 藤枝直輝[†1] 若杉祐太[†1]
高前田伸也[†2] 森洋介[†2] 吉瀬謙二[†1]

FPGAデバイスの大容量化に伴い，柔軟かつ効率的な組込みシステム開発に適したソフトプロセッサ (ソフトマクロのマイクロプロセッサ) の利用が広がっている．我々は，本研究室で開発しているMIPSシステムシミュレータSimMipsのMIPSコア部分をVerilog HDLに移植することにより，シンプルでカスタマイズ可能なMIPS32命令セットの一部を実装するソフトプロセッサであるMipsCore，及びMipsCoreを利用したシンプルな組込みシステムSimplemを開発している．本稿では，MipsCoreの開発背景とそのコンセプトについて述べ，既存のソフトプロセッサとの比較を行う．さらにSimplem及びその上で動くアプリケーションについて述べる．

Development of Simple Embedded System with MIPS System Simulator SimMips

Shimpei Watanabe,[†1] Naoki Fujieda,[†1] Yuhta Wakasugi,[†1]
Shinya Takamaeda,[†2] Yosuke Mori[†2] and Kenji Kise[†1]

The growth of FPGA device capacity enables us to use soft-processor which makes development of embedded system flexible and efficient. We are developing a simple and full-customisable MIPS32 ISA soft-processor MipsCore and a simple embedded system Simplem including MipsCore. To develop MipsCore, we use SimMips —our designed MIPS system simulator—. In this paper, we first describe the background and concept of MipsCore and compare with other soft-processors. We also describe about Simplem and applications run on it.

1. はじめに

近年では，FPGA(Field Programmable Gate Array) デバイスの大容量化に伴い，ソフトプロセッサ (ソフトマクロのマイクロプロセッサ) の利用が広がっている[1)–4)]．ソフトプロセッサでは，プロセッサの機能の追加や変更が可能であり，これを用いることにより柔軟でかつ効率的な組込みシステム開発を比較的容易に実現することが期待できる．

我々は，本研究室で開発しているC++で記述されたMIPSシステムシミュレータSimMips[5)]を利用し，その一部をVerilog HDLに移植することにより，MIPS32命令セットの一部を実装するシンプルでカスタム可能なソフトプロセッサ**MipsCore**を開発した．さらに，MipsCoreを組み込んだシンプルな組込みシステム**Simplem**を開発した．

本稿の構成を述べる．2章では，我々が開発しているシステムシミュレータSimMipsについて述べる．さらにソフトプロセッサMipsCoreを開発した背景，及びその設計コンセプトについて述べる．3章では，ソフトプロセッサMipsCoreの詳細と，その開発，及びMipsCore上で動くアプリケーションについて述べ，既存のソフトプロセッサとの比較を行う．4章では，MipsCoreを利用したシンプルな組込みシステムSimplemについて述べる．5章で本稿をまとめる．

2. 背景とコンセプト

2.1 システムシミュレータSimMips

プロセッサアーキテクチャの教育・研究のツールとして様々なプロセッサシミュレータが用いられている[6),7)]．さらに，このようなプロセッサシミュレータに加え，シミュレータ上でOSを動作させて，入出力などを含めた計算機システムとしてシミュレートを行

†1 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†2 東京工業大学 工学部情報工学科
Department of Computer Science, Tokyo Institute of Technology

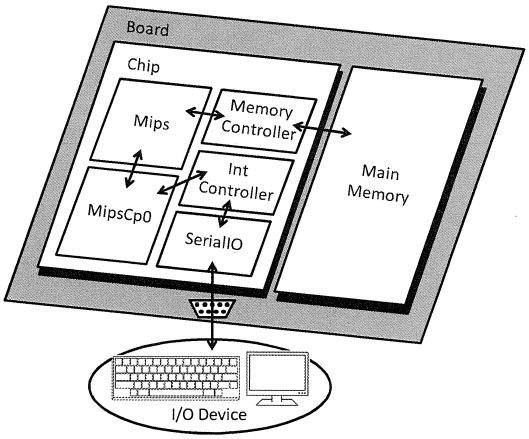

図 1 SimMips がシミュレートするシステムのハードウェア構成. IntController は割り込みコントローラ (Interrupt Controller) を意味する. 特に関係の強いユニット同士を矢印で結んでいる.

うシステムシミュレータへの要求が高まっている.

このような背景の下, 我々は MIPS32 命令セット[8] のプロセッサを含むシステムシミュレータ SimMips を開発している.

SimMips はシンプルさと可読性を重視して設計されており, 記述には自由度や柔軟性が高く, またデバッグが行いやすい C++が用いられている. SimMips は 5000 行以下とコンパクトなコード量であるにもかかわらず, コードを修正していない最新版の Linux がこの上で動作する. 理解しやすく学びやすい構造とするため, SimMips はハードウェア構成を意識した設計となっている.

SimMips は ELF 形式の実行ファイルを読み込み, 記述されている命令の処理をシミュレートする. この際 OS モード, App モードと呼ばれる 2 つの実行モードを提供する. OS モードでは, OS カーネルを ELF 形式の実行ファイルとして読み込み, Linux 等の OS を動作させることができる. App モードでは, 静的にリンクされたユーザプログラムを動作させることができる. App モードを用いることにより, SimMips をシステムシミュレータとしてだけではなく, シンプルなプロセッサシミュレータとして利用することもできる.

SimMips を用いたシミュレーションにより, 通常のアプリケーションや OS の出力に加えて, メモリやレジスタの値, 実行された命令のトレース, 統計データなどを得ることができる.

図 1 に, SimMips がシミュレートするシステムのハードウェア構成を示す. シミュレートするシステム全体のことを Board と呼ぶ. Board はチップ (Chip) と主記憶 (MainMemory) から構成される. Chip には MIPS プロセッサ (Mips) やシステム制御コプロセッサ CP0(MipsCp0) に加え, メモリインタフェース (MemoryInterface), 割り込みコントローラ (IntController), シリアル I/O コントローラ (SerialIO) が集積されている. 入出力にはシリアルコンソールを用いる.

2.2 MipsCore 開発の背景とコンセプト

FPGA デバイスの大容量化に伴い, ソフトプロセッサの利用が広がっている. それに伴い, FPGA メーカーなどから多くのソフトプロセッサコアが配布されている.

ソフトプロセッサの利点として, プロセッサコアが実装している命令及び機能を追加・変更することが可能であり, 柔軟で効率的なシステム開発が可能となるというものがある. しかし FPGA メーカーなどにより配布されているソフトプロセッサコアでは, カスタマイズのしやすさを欠くことがあり, ソフトプロセッサのメリットが十分に活かせないことがある.

そこで我々は, カスタムが容易でオープンソースのソフトプロセッサコアを提供することを目指し, シンプルな MIPS ソフトプロセッサコア MipsCore の開発を行っている. 前述したシステムシミュレータ SimMips は, シミュレーション対象システムのハードウェア構成を考慮した設計であることから, ハードウェア記述言語と非常に相性が良い. この特徴を活かし, この計算コア部分を Verilog HDL に移植することにより, SimMips のもつシンプルさと可読性という長所を活かしつつ, 比較的低コストかつ短期間での開発を可能としている. また, メインメモリを除く大部分がプラットフォームに依存しない論理合成可能な完全な HDL で記述されている. さらに, IP を搭載する場合にも比較的少ない変更で対応できる.

MipsCore は SimMips と併用することにより, その柔軟性, カスタマイズのしやすさを増すことができる. 例えば MipsCore をカスタマイズしたい場合には, まず SimMips に変更を施すことにより, 自由度が高くデバッグの幅が広い C++で, 比較的容易に柔軟性の高い検討を行うことができる. さらに, MipsCore に変更を施した後のデバッグ作業にも, SimMips を用いることは有用となるであろう.

3. MipsCore の開発

3.1 MipsCore ver.0.9.6 の構成

MipsCore ver.0.9.6 における Verilog HDL モジュールの構成を述べる. MipsCore ver.0.9.6 は, 乗算器 MULUNIT モジュール, 除算器 DIVUNIT モジュール, 汎用レジスタ GPR モジュールと, MIPS 計算コアにおいてその他の計算を行う部分を実装した MipsCore モジュールからなる. MipsCore ver.0.9.6 はメインメモリを含まない. そのため, これを FPGA ボー

ド上で動作させる場合には，数 10KB 程度の 1 サイクルで参照可能なシングルポートメモリをブロック RAM として生成し，これを内部メモリとして用いる必要がある．なお，メモリは命令とデータで共通のものを用いる．

MULUNIT と DIVUNIT は，乗算，除算の処理にそれぞれ 32 サイクルを要するシンプルな構成である．乗算器，除算器，汎用レジスタを別のモジュールとして切り出したことにより，この部分に別の IP を採用することが容易になっている．

MipsCore モジュールはパイプライン化されていないシンプルなマルチサイクルプロセッサとなっている．命令フェッチ (IF)，命令デコード (ID)，レジスタフェッチ (RF)，命令実行 (EX)，メモリリード (MR)，メモリライト (MW)，ライトバック (WB) の 7 ステージから構成されており，基本的に各ステージを 1 サイクルで通過する．そのため，メモリのリードライトを行うロード/ストア命令は 7 サイクル，実行に時間のかかる乗除算には 36 サイクル，その他の命令には 5 サイクルを要する．MIPS32 命令セットの計算コアの命令のうち，積和演算を除いた主要なものである 70 種の命令が実装されている．

MipsCore0.9.6 では，各ステージをモジュールに切り分けていない．これは全体の構成が各ステージをモジュールとして切り分けるほど複雑ではなく，1 つのモジュールにまとまっていた方が見通しが良いと判断したことによる．

3.2 MipsCore の実装

前述したように，MipsCore は SimMips の MIPS コア部分を記述したクラスを Verilog HDL に移植することにより実装されている．

図 2 に，MipsCore のデコードステージのうち，符号なし加算命令 addu をデコードしている部分を，**図 3** に，SimMips のデコードステージのうち，同じく addu をデコードしている部分をそれぞれ示す．各行の記述内容はそれぞれ対応している．

まず 3 行目から 13 行目で，命令を各フィールドに分解している．MipsCore では単純に命令を該当するビット幅に切り分けている．SimMips では，C++ではそのような記述ができないことから，入力された命令をシフトしてマスクするという操作を行うことにより分解を行っている．

15 行目から 17 行目では，初期化作業を行っている．SimMips では，latency を変更することにより 1 命令の実行にかかる時間を変更することができるが，MipsCore ではこの機能はサポートしていない．

19 行目から後はデコード操作である．opcode や funct に合った命令を選び，適切な操作を行っている．

同様に，**図 4** に，MipsCore の実行ステージのうち，addu を実行している部分を，**図 5** に，SimMips の実行ステージのうち，addu を実行している部分をそれ

```
 1 /* MipsInst::decode() */
 2 always@ ( DATA_IN ) begin
 3   IDOPCODE = DATA_IN[31:26];
 4   IDRS     = DATA_IN[25:21];
 5   IDRT     = DATA_IN[20:16];
 6   IDRD     = DATA_IN[15:11];
 7   IDSHAMT  = DATA_IN[10:6];
 8   IDFUNCT  = DATA_IN[5:0];
 9   IDIMM    = DATA_IN[15:0];
10   IDADDR   = DATA_IN[25:0];
11   /* IDCODE_L   not used in MipsCore now
12      IDCODE_S   (used in OS mode)        */
13   IDSEL    = DATA_IN[2:0];
14
15   IDOP     = `UNDEFINED;
16   IDATTR   = `READ_NONE | `WRITE_NONE;
17   /* changing instruction latency is
18                 not supported in MipsCore */
19   case ( IDOPCODE )
20     6'd0: begin
21       case ( IDFUNCT )
22         ...
23         6'd33: begin
24           IDOP   = `ADDU_____;
25           IDATTR = `READ_RS | `READ_RT | `WRITE_RD;
26         end
27         ...
```

図 2 MipsCore における命令デコード部分の記述より，符号なし加算命令 addu をデコードしている部分を抜粋．

```
 1 void MipsInst::decode()
 2 {
 3    opcode = (ir >> 26) &      0x3f;
 4    rs     = (ir >> 21) &      0x1f;
 5    rt     = (ir >> 16) &      0x1f;
 6    rd     = (ir >> 11) &      0x1f;
 7    shamt  = (ir >>  6) &      0x1f;
 8    funct  =  ir        &      0x3f;
 9    imm    =  ir        &    0xffff;
10    addr   =  ir        & 0x3ffffff;
11    code_l = (ir >>  6) &   0xfffff;
12    code_s = (ir >> 16) &     0x3ff;
13    sel    =  ir        &       0x7;
14
15    op = UNDEFINED;
16    attr = READ_NONE | WRITE_NONE;
17    latency = 1;
18
19    switch (opcode) {
20    case 0:
21        switch (funct) {
22        ...
23        case 33:
24            op = ADDU_____;
25            attr = READ_RS | READ_RT | WRITE_RD;
26            break;
27        ...
```

図 3 SimMips における命令デコード部分の記述より，符号なし加算命令 addu をデコードしている部分を抜粋．

```
1  /* Mips::execute() */
2 always @( inst_op or inst_rrs or inst_rrt or ...) begin
3   ...
4   case ( inst_op )
5     ...
6     `ADDU_____ : begin
7       EXRSLT    = inst_rrs + inst_rrt;
8      end
9     ...
```

図 4 MipsCore における命令実行部分の記述より，符号なし加算命令 addu を実行している部分を抜粋．

```
1 inline void Mips::execute()
2 {
3     ...
4     switch (inst->op) {
5     ...
6     case ADDU_____:
7         rrd = rrs + rrt;
8         break;
9     ...
```

図 5 SimMips における命令実行部分の記述より，符号なし加算命令 addu を実行している部分を抜粋．

ぞれ示す．

先ほどのデコードステージでセットした値から行う操作を選び，実行している．

これ以外の部分についても，SimMips の設計がハードウェアを意識したものであったことから，Verilog HDL への移植は比較的容易なものであった．そのため，実装は修士課程の学生 1 名及び学部学生 2 名により，約 1 週間と非常に短期間で行うことができた．

MipsCore のデバッグにおいても SimMips を用いる．MipsCore を Verilog シミュレーションする際に，同じアプリケーションを SimMips で動作させ，各命令実行終了時の汎用レジスタの値をログとして出力する．双方のログ同士を比較することによって，MipsCore が SimMips と同じ動作をしているかどうかを詳細に検証することができる．この検証作業は簡単なシェルスクリプトを記述することで容易に行うことができる．

3.3 他のソフトプロセッサとの比較

表 1 他のソフトプロセッサとの比較

	MipsCore	MicroBlaze	Plasma
(1) 開発	TOKYOTECH	Xilinx	Steve Rhoads
(2) ISA	MIPS32	MicroBlaze	MIPS1
(3) HDL	Verilog	VHDL	VHDL
(4) OS	No	uCLinux	RTOS(独自)
(5) FPU	No	Yes	No
(6) キャッシュ	No	Yes	No
(7) 乗除算器	HDL/IP	IP	HDL/IP
(8) パイプライン	No	Yes	Yes
(9) 実装命令数	70	100 以上	56
(10) スライス数	1110	642	990
(11) 動作周波数	127MHz	98MHz	40MHz
(12) 行数	1081	不明	1656

表 1 に，MipsCore と，既存のソフトプロセッサである Plasma[3] 及び MicroBlaze[4] との比較を示す．

表の上部には，各ソフトプロセッサの主な機能を挙げる．(4)(5)(6)(8) より，他のソフトプロセッサと比べ MipsCore は，OS が動作しない，FPU，キャッシュを搭載しないといった，機能の少なさが目立つ．機能の追加が今後の課題である．

表の下部には，各ソフトプロセッサをカスタマイズ/使用する上での特徴を挙げる．(10) のスライス数は，Xilinx のツールにて論理合成する際の大きさに関する指標である．これらは構成の違いを考慮してソフトプロセッサのコア部分から乗除算ユニットや浮動小数点演算ユニットなどの特別な演算ユニットを除いた部分についての値とした．使用ツールは Xilinx ISE 10.1.02，ターゲットデバイスは Spartan3E XC3S1200E，スピードグレードは −4 である．

使用するスライス数についてみると，MipsCore は他の 2 つより大きくなっている．しかしこの数字は一般的にはそれほど大きな値ではないため，実用に耐えうると言える．

動作周波数に関しては MipsCore が高くなっているが，MipsCore はマルチサイクルで動作するということを考慮に入れる必要がある．パイプライン化されている他のプロセッサとの単純比較はできず，性能は最も低い．しかし，これは今後 MipsCore のパイプライン化を行うことにより改善できる．

行数に関しては，MipsCore のコンパクトさが目立つ．MipsCore はコンパクトなことにより可読性が高く，カスタマイズが容易であると言える．

4. シンプルな組込みシステム Simplem

4.1 Simplem の開発

我々は，MipsCore を搭載したシンプルな組込みシステム Simplem(**Simple** Mips **Em**bedded System) を開発している．

Simplem では，MipsCore を SUZAKU-S[9] に載せ，MipsCore 上で動いているアプリケーションにより，コマンドインタープリター液晶 ITC-2432-035H にコマンドを送り，画面表示をする．現在のバージョンでは，アプリケーションはコンフィグファイル内に格納されており，ブロック RAM の初期化の際に書き込む形となっている．

Simplem の構成を**図 6** に示す．MipsCore に加え，メインメモリ及びメモリマップド I/O を司る memcon，MipsCore のメインメモリ mainmem，キーボードからのスキャンコードを受け取り，アスキーコードにして memcon に渡す kbcon，memcon からの入力をシリアルポートの通信規格に沿った形とし，コマンドインタープリター液晶に対する出力とする lcdcon，それらをまとめた subboard の各モジュールを実装した．これらは Verilog にて 250 行程度で記述できた．

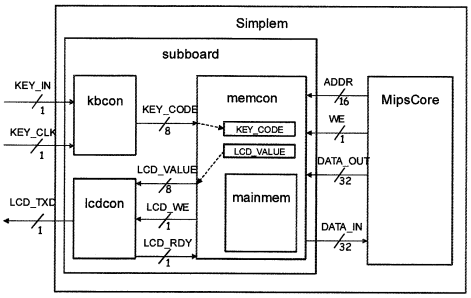

図 6 Simplem のブロック図.

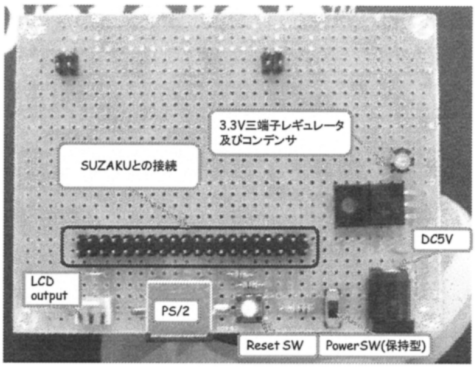

図 7 Simplem 用マザーボード

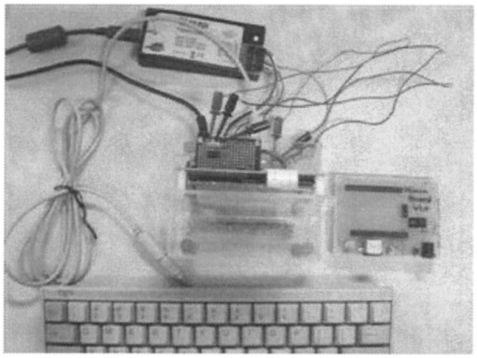

図 8 Simplem 全体図

さらに，キーボードやコマンドインタープリター液晶との接続及び電源の確保のため，**図 7** に示すマザーボードを製作した．

加えてコマンドインタープリター液晶及びSUZAKU-S を収納するボックス，コマンドインタープリター液晶への入力線をまとめるボードを製作し，これらすべてを適切に繋いだものが**図 8** である．

図 9 は Simplem を後ろからみた図である．配線が非常に少なく，またシンプルになっていることがわかる．

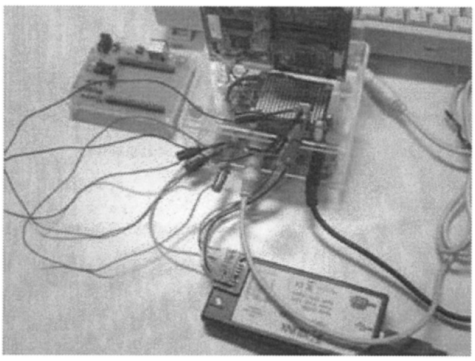

図 9 Simplem 後方の配線

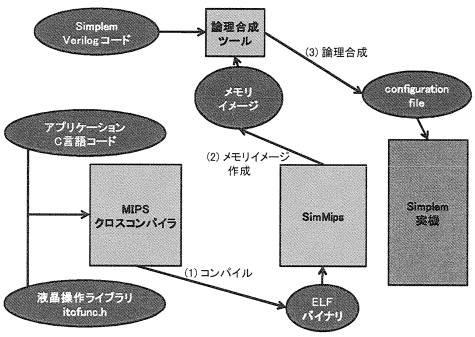

図 10 Simplem 上で動作するアプリケーションの開発フロー

4.2 Simplem 上で動作するアプリケーション

Simplem 上で動作するアプリケーションは，**図 10** に示すフローで開発することができる．なお，このフローでは C 言語を用いることを想定している．

まずはアプリケーションの記述を行う．我々はコマンドインタープリター液晶 ITC-2432-035H を操作するためのライブラリ itcfunc.h を開発しており，Simplem 上で動作するアプリケーションを簡単に書くことができる．記述したソースを通常の MIPS クロスコンパイラにてコンパイルする．この際ライブラリをスタティックリンクする必要がある．次に，生成された ELF バイナリを SimMips 上で実行し，メモリイメージを作成する．さらに，メモリイメージを Simplem の Verilog コードとともにツールにて論理合成し，出力されたコンフィグレーションファイルを Simplem 実機に書き込めば，アプリケーションを動作させることができる．

近年では，Buildroot[10] などのツールを用いることにより，手軽にクロスコンパイル環境の構築ができるようになってきている．また，SimMips もオープンソースのシミュレータとして公開することを予定している．このようにアプリケーションを作るために特殊なコンパイラやリンカなどが必要なく，容易に開発環

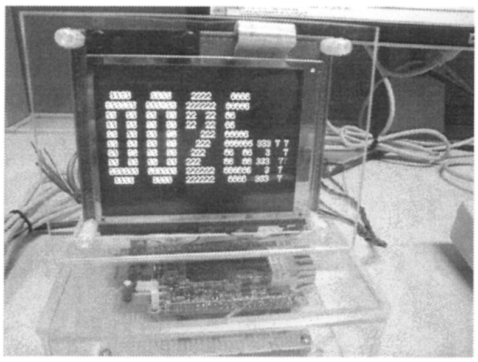

図 11 Simplem 上でタイマーアプリケーション Timer が動作しているところ

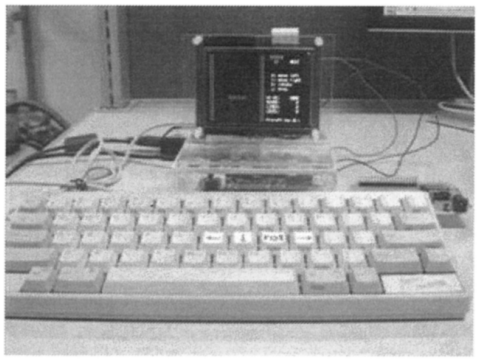

図 12 Simplem 上でブロック消去ゲームが動作しているところ

境が構築できるという点も，Simplem の大きな強みである．

実際に動いているアプリケーションの例を挙げる．ここでは論理合成ツールに ISE を用いた．

図 11 は，Simplem 上にてタイマーアプリケーション Timer が動いているところである．これは電源を入れてからの時間を表示するアプリケーションである．今後，一定時間経過後にブザーを鳴らすなどといった改良を加える予定である．

図 12 は，落ちてくる様々な形状のブロックを横一列にそろえて消去するパズルゲームが Simplem 上で動作しているところである．キーボードからの入力を受け取り，ブロックを操作することが可能である．

5. おわりに

FPGA デバイスの大容量化に伴い，ソフトプロセッサの利用が広がっている．我々は，MIPS32 命令セットの一部を実装するソフトプロセッサコア MipsCore 及び MipsCore を載せたシンプルな組込みシステム Simplem を開発している．本稿では，MipsCore の元となった MIPS システムシミュレータ SimMips について，及び MipsCore のコンセプト，開発について述べ，その有用性を明らかにした後，シンプルな組込みシステム Simplem について述べた．

今後の課題を述べる．現在のバージョンの MipsCore は，SimMips の MIPS 計算コア部分だけを移植したものである．OS を動かし，実用的なソフトプロセッサとするためには，コプロセッサや割込みコントローラ，DRAM コントローラ，キャッシュコントローラなどを加えて実装する必要がある．また，パイプライン化することにより大幅な性能向上が見込める．これはコードの可読性を失う恐れがあるため，慎重に行う必要がある．

参考文献

1) Dimond, R., Mencer, O. and Luk, W.: CUSTARD-A Customisable Threaded FPGA Soft Processor and Tools, *International Conference on Field Programmable Logic (FPL)* (2005).
2) Gonzalez, R., Inc, T. and Santa Clara, C.: Xtensa: a configurable and extensible processor, *Micro, IEEE*, Vol. 20, No. 2, pp. 60–70 (2000).
3) Rhoads, S.: Plasma CPU, http://plasmacpu.no-ip.org:8080/.
4) Xilinx, I.: MicroBlaze Processor Reference Guide, *reference manual* (2006).
5) 藤枝直輝，渡邉伸平，吉瀬謙二：SimMips: 教育・研究に有用な Linux が動く 5000 行の MIPS システムシミュレータ，第 20 回コンピュータシステム・シンポジウム ComSys 2008 (2008).
6) Burger, D. and Austin, T. M.: The Simplescalar Tool Set, Version 2.0, Technical Report CS-TR-1997-1342, University of Wisconsin-Madison (1997).
7) 吉瀬謙二，片桐孝洋，本多弘樹，弓場敏嗣：SimCore/Alpha Functional Simulator の設計と実装，電子情報通信学会論文誌，Vol.J88-D-I, No.2, pp.143–154 (2005).
8) Sweetman, D.: *See MIPS Run Linux Second Edition*, Morgan Kaufmann (2006).
9) 株式会社アットマークテクノ：SUZAKU-S, http://www.atmark-techno.com/products/suzaku/suzaku-s.
10) Andersen, E.: Buildroot. http://buildroot.uclibc.org/.