64bit RISC-V SoC Rocket-Chip を用いたカスタム命令の実装と評価

1230063 小林 奏斗（集積システム研究室）
（指導教員 密山 幸男 教授）

1. はじめに

オープンソースプロセッサの普及にともない、ターゲットアプリケーションドメインを特化した高効率プロセッサの研究開発が注目されている。本研究では、高効率なプロセッサの開発することを目指し、オープンな ISA (Instruction Set Architecture) である RISC-V に注目し、64bit RISC-V SoC (System on a Chip) である Rocket-chip[1]を用いてカスタム命令の実装とその評価を行った。

2. Rocket-chip

Rocket-chip はカリフォルニア大学バークレイ校と SiFive 社が開発した RISC-V SoC である。記述言語には Verilog HDL より上位の言語である Chisel が用いられている。Rocket-chip の外部には RoCC (Rocket Custom Coprocessor) と呼ばれるアクセラレータが用意されていることが最大の特長のひとつである。Rocket-chip にカスタム命令を追加する方法として、RoCC を用いるほか複数の方法が報告されている[2]。

3. Rocket-chip を用いたカスタム命令の実装

ビット反転命令とパリティ計算命令を評価実験カスタム命令として追加した。ビット反転命令は 64bit のビット列を反転させる。パリティ計算命令は 64bit のすべてのビットに対して XOR 演算を実行した結果を最下位ビットに出力する。

命令の追加方法として以下の 3 つのハードウェア変更方法を採用した。コンパイラは gcc に変更を加えたものを使用した。

(1) ALU の拡張：ALU 内部にカスタム命令専用の回路を追加する方法（図 1）
(2) 専用 Unit の追加：ALU と並列にカスタム命令専用の Unit を追加する方法（図 2）
(3) RoCC の使用：RoCC の内部にカスタム命令専用の回路を追加する方法（図 3）

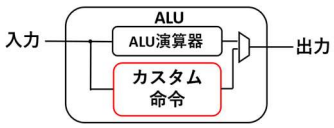

図 1. ALU の拡張

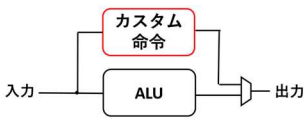

図 2. 専用 Unit の追加

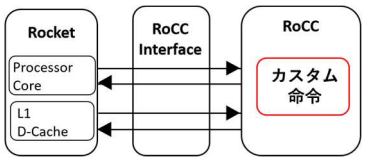

図 3. RoCC の使用

コンパイラでカスタム命令を扱うために、gcc にカスタム命令を新たに定義する。まず、カスタム命令の MASK、MATCH、DECLARE を定義する。MASK は funct7、funct3、opcode を抽出するためのビットパタンである。MATCH はカスタム命令に対して MASK を用いて抽出した値である。DECLARE は MASK によって抽出された値が MATCH と等しいときにカスタム命令と判定するための関数である。次に、命令のフォーマットを記述し、命令の名前やオペランド、命令の判定方法などを指定する。

4. 実験方法

Rocket-chip を用いてシミュレーションと FPGA 実装の 2 種類の方法で実験を行う。ビット反転とパリティ計算を追加したプロセッサと、カスタム命令を実装していないプロセッサ（ベースライン）でサイクル数と命令数を比較評価する。カスタム命令を実装したプロセッサではカスタム命令を使用する C プログラムを用い、ベースラインでは比較用の C プログラムを用いる。

シミュレーションではオープンソース Verilog シミュレータである Verilrator を使用した。FPGA 実装では Xilinx Artix-7 FPGA を搭載する Digilent Nexys Video を用い、2 種類の実行方法でプログラムを動作させた。ひとつはベアメタルプログラムで OS を介さずに実行する方法である。もうひとつの方法は Rocket-chip 上で OS（Linux）を動作させ、プログラムを実行する方法である。

5. 実験結果

シミュレーションで得られたサイクル数と命令数を表 1 と表 2 にそれぞれ示す。ベースラインと比較すると、カスタム命令の追加によりサイクル数と命令数を大きく削減できることを確認した。ALU の拡張と専用 Unit の追加ではサイクル数と命令数に差が生じなかった。

表 1 シミュレーション結果（ビット反転）

ALU 拡張		専用 Unit		RoCC 使用		ベースライン	
Cycle	Inst.	Cycle	Inst.	Cycle	Inst.	Cycle	Inst.
8	5	8	5	14	6	226	80

表 2 シミュレーション結果（パリティ計算）

ALU 拡張		専用 Unit		RoCC 使用		ベースライン	
Cycle	Inst.	Cycle	Inst.	Cycle	Inst.	Cycle	Inst.
8	5	8	5	14	6	109	35

FPGA 実装で得られたサイクル数と命令数を表 3 と表 4 に示す。シミュレーションと同様に、カスタム命令を追加することでサイクル数と命令数を大きく削減できた。ただし、Linux 上での実行はいずれの結果と比較してもサイクル数と命令数が増加していた。

表 3 ベアメタルプログラム実行結果（ビット反転）

ALU 拡張		RoCC 使用		ベースライン	
Cycle	Inst.	Cycle	Inst.	Cycle	Inst.
8	5	14	6	89	80

表 4 Linux 上でのプログラム実行結果（ビット反転）

ALU 拡張		RoCC 使用		ベースライン	
Cycle	Inst.	Cycle	Inst.	Cycle	Inst.
17	9	23	10	410	77

6. まとめ

Rocket-chip にカスタム命令を 3 通りの方法で実装し、シミュレーションと FPGA 実装によってカスタム命令の有効性を比較評価した。実験結果からカスタム命令が正しく追加され、有効に機能していることを確認した。ターゲットアプリケーションにおいて適切なカスタム命令を追加することにより、高効率プロセッサの開発が可能になると考えられる。今後の課題として Chisel の習熟や、RoCC からコアへの割り込みの実装などが挙げられる。

参考文献

[1] Krste Asanovi´c, et al., “The Rocket Chip Generator,” Technical Report No. UCB/EECS-2016-17, 2016.
[2] 中尾怜史, 武内良典, “RISC-V を用いた命令拡張のためのプロセッサ開発環境の検討,” 情報処理学会 DA シンポジウム, 2021 年 9 月.