Performance Evaluation on RISC-V Architecture

Electrical and Computer Engineering Department - California State Polytechnic University - Pomona

Carson Quesada, Derek Jacoby, Demajay Francis, Cole Pederson, Professor Mohammed Aly
California State Polytechnic University, Pomona
Correspondence: email@example.com

***Abstract*—This paper presents the benchmarking and evaluation of several RISC-V softcore implementations deployed on the Nexys A7 FPGA board. The study examines performance across multiple instruction categories and processor architectures using cycle counts, instruction counts, CPI, latency, and power measurements.**

***Index Terms*—RISC-V, FPGA, benchmarking, softcore processor, RV32IMC**

I. Introduction

RISC-V is an open instruction set architecture gaining popularity due to its modularity, scalability, and open-source ecosystem. With many available softcore implementations, comparing their performance is essential for selecting an appropriate core in embedded applications. This work benchmarks four RISC-V softcores on a Nexys A7 FPGA board using a custom-designed testing methodology.

II. Methodology

To benchmark the RISC-V softcores on the Nexys A7 FPGA board, we used several development tools along with custom-built test programs. This section describes the development workflow, hardware setup, and benchmarking methods.

A. Nexys A7 Development Environment

All FPGA development was performed in Xilinx Vivado. The Verilog design programmed onto the Nexys A7 included three components:

1) The RISC-V softcore under test,
2) I/O interfaces for user control and UART output,
3) On-chip BRAM used as program memory.

1) Softcore Implementations: Four open-source RISC-V softcores supporting RV32IMC were selected for evaluation:

- PicoRV32 — 1-stage pipeline,
- Ibex — 2-stage pipeline,
- VexRiscv — 5-stage pipeline,
- cv32e40p — 4-stage pipeline.

Each softcore was placed in its own Vivado project and instantiated inside the `top.v` module.

2) I/O Integration: To control the benchmarks and observe results, the following interfaces were implemented:

- Two pushbuttons (start test, skip test),
- UART output for transmitting benchmark results to a serial terminal.

3) Program Memory: Approximately 64 kB of BRAM was allocated to store the benchmark program and necessary data. BRAM was initialized using a HEX memory file generated by the RISC-V toolchain.

B. Test Program Development

The benchmark program executed common RISC-V operations and accessed hardware counters to measure performance.

1) Compilation Workflow: We used the xPack `riscv-none-elf-gcc` toolchain:

1) The benchmark program was written in C,
2) Compiled into an ELF file,
3) Converted from ELF to HEX format,
4) Loaded into BRAM during synthesis.

2) Benchmark Operations: The following instruction categories were benchmarked:

- Addition and subtraction,
- Multiplication and division,
- Load and store instructions,
- Branch instructions,
- Bit shifting operations.

For each category, we collected:

- Cycle count,
- Instruction count,
- CPI (cycles per instruction),
- Execution latency,
- Power usage.

3) Performance Counter Access: The benchmark program accessed RISC-V machine counters:

- `mcycle` / `mcycleh` — cycle counter,
- `minstret` / `minstreth` — instruction-retired counter.

Before and after each operation, the program read the counters and used the difference to compute performance metrics.

4) Power Measurement: Power consumption was measured using a USB inline power meter. The method was:

1) Measure idle power of the board,
2) Run each operation in a tight loop for 5–10 seconds,
3) Record active power,
4) Compute net power usage by subtracting idle power from active power.

C. Running the Tests

Benchmark results were viewed through a UART serial terminal using PuTTY. Once the bitstream was programmed to the FPGA, the benchmark application automatically began execution and printed results to the terminal.

Acknowledgment

The authors thank the Electrical and Computer Engineering Department at Cal Poly Pomona for laboratory resources and academic support throughout this project.

References

[1] Clifford Wolf, "PicoRV32 — A Size-Optimized RISC-V CPU," GitHub Repository.
[2] lowRISC Contributors, "Ibex RISC-V Core," GitHub Repository.
[3] SpinalHDL Contributors, "VexRiscv CPU Core," GitHub Repository.
[4] OpenHW Group, "cv32e40p RISC-V Core," GitHub Repository.