Enabling Reconfigurable High-Throughput RISC-V Systems through Barrel-Processing

Riadh Ben Abdelhamid[1] and Dirk Koch[1]

[1]ZITI, Heidelberg University

Abstract

Traditional optimization techniques for higher-speed operation of CPU cores don't work well, when targeting FPGAs. This mostly holds with one exception: barrel processing*, which is a technique that uses aggressive pipelining together with multithreading such that any pipeline related conflicts will be resolved before a thread is restarted again. This article will outline the advantages of barrel processing for both CPU implementations on ASICs and FPGA targets.*

Introduction

There exist many implementations of RISC-V CPUs that have been implemented on an FPGA. In many cases, those implementations use an FPGA more as a prototyping target while having an ASIC implementation in mind. For instance, BOOM (the Berkeley Out-of-Order Machine) was implemented on FPGA and ASIC [1]. However, even using fast modern FPGA fabrics, BOOM only achieves low microcontroller-like speed (e.g., on AMD 7-series FPGA platforms, configurations like 8 64-bit RISC-V BOOM cores, reach clock speeds of up to 100 MHz while lower-end FPGAs can barely reach 50 MHz. [2]).

It is important to note that these frequencies are generally lower than those achieved in ASIC implementations, where BOOM has been demonstrated to run at 1 GHz at 0.9V and 320 MHz at 0.6V [3], implying an FPGA-to-ASIC speed gap of $\approx 10\times$.

Even (RISC-V) CPUs that target FPGAs as the main target struggle to deliver high performance, when targeting modern fast devices. For instance, the work in [4] claims to provide a high-performance out-of-order softcore implementation that operates at sub 100 MHz. Perhaps, one of the fastest RISC-V softcores is GRVI operating at 375 MHz on a VU9P in a single core configuration [5] .

The reason for this relatively poor performance is that deep pipelining (beyond three stages) does not perform well on FPGAs. As examined by Rose et al. in [6], the dependencies introduced with pipelining are expensive to be resolved when using FPGA softlogic. For instance, register forwarding requires adding multiplexers in front of the ALU, which usually results in a longer combinatorial path, which, in turn, backfires the original goal of pipelining.

One notable exception to this rule is barrel processing. This extended abstract explores how the barrel processing concept can be leveraged to improve CPU performance for both ASIC-based and FPGA-based RISC-V implementations. This paper discusses the use of barrel processors to be used in settings with reconfigurable logic.

Barrel Processing

Barrel processing employs an aggressive form of pipelining combined with multi-threading. It is issuing threads in a round-robin fashion; and because each pipeline stage processes a different thread at any given cycle, any hazards (data, control, or structural) are naturally resolved over time. Consequently, this multi threaded pipeline design can achieve higher throughput and more predictable performance without requiring complex hardware-based hazard detection and resolution mechanisms. Core benefits of barrel processing include:

- **high** aggregated **performance**
- **low cost** (in term of resources)
- **simple design** (primarily simplified verification)
- more **deterministic thread execution** for real-time systems or lockstep operation
- **well-suited for ASIC and FPGA targets**

The main drawback of a barrel processor is its lower single thread performance, which, for FPGA targets, is offset by the much higher achievable clock speed. Moreover, it is relatively easy to prioritize one thread as soon as it is conflict free. This usually boosts the performance of a single thread $2-3\times$.

ASIC Implementations The main overhead introduced with barrel processing stems from a larger registerfile required for storing the thread contexts. The following table lists the register file cost for 1, 2, 4, 8, 16 × 32b contexts when using 2r1w macros generated for the Skywater 130 nm process node using OpenRAM [7]:

total words	32	64	128	256	512
area [mm^2]	0.058	0.074	0.106	**0.168**	0.284

As can be seen, the area for storing 16 thread contexts is just 5× the area of a single context. This is due to the disproportionately high cost for the SRAM sense amplifiers that disproportionally kick in harder for smaller RAM macros.

We implemented and taped out a 8 stage/thread barrel processor in the open Skywater 130 process node using OpenLane [8]. The core area is $0.194\,mm^2$ without instruction or data memories (see Figure1a)). The register file for 8 contexts is adding another $0.168\,mm^2$. Implementing the RV32I+lr/sc+csrrs ISA, the core could run at 40 MHz.

When extending a CPU with reconfigurable custom instructions implemented by an eFPGA, the barrel principle can be used to entirely hide the latency of the otherwise slower eFPGA if custom instructions are deeply pipelined too. This is well-supported as the fabric offers a flip-flop after each logic element (LUT).

Figure 1b) shows a CPU with a FABulous eFPGA [9] that can implement reconfigurable custom R and I type instructions where the operands $rs1, rs2/imm$ and the result rd are directly connected to the fabric. Opposed to [10], where reconfigurable instructions will stall the CPU for a programmable time, the barrel approach can accept a new custom instruction each clock cycle but mandates a fixed number of pipeline stages for the execution inside the eFPGA (usually 4-8 stages).

Table 1: *Comparison with related works.*

	OoO [4], 2019	in-order [5], 2016	**Barrel** [11], 2025
FPGA	XC7Z020	VU9P	VU9P
LUT	∼15k	**320**	646
BRAM	6	0.5-1**	1
ISA	RV32IM	RV32I+lr/sc-bshift	RV32I+lr/sc+csrrs
FMAX	95.3 MHz	375 MHz	**737 MHz**
MIPS/LUT	0.012	0.73 [5]	**1.14**

* The work [5] reports a BRAM utilization of 4 to 8 in a cluster of 8 cores which leads to 0.5 to 1 BRAM for a single core.

FPGA Softlogic Implementations When implementing the register file of a RV32I FPGA implementation in a block RAM (BRAM), the smallest BRAM on a Xilinx/AMD UltraScale+ FPGA can fit 16 threads. We used this for an improved 16 stage BRISKI barrel processor [12] implementation. The results in Table 1 show that the barrel processor delivers fastest instruction throughput and best MIPS/LUT density as compared with related approaches. The achieved 737 MHz is essentially the top speed of the AMD Virtex UltraScale+ BRAM primitives. With this, the barrel approach delivers 1.14 MIPS per LUT. The approach was scaled up to 1024 cores /16K threads operating at 500 MHz on a VU9P FPGA in [13] (Figure 1c)).

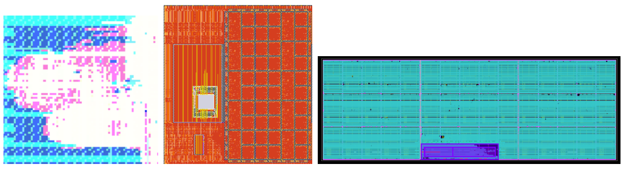

Figure 1: *l2r.: a) Barrel Processor core with 8 pipeline stages; b) RISC-V CPU with attached eFPGA; c) 1,024 cores (16,384 threads) Barrel system with PCIe backplane on a VU9P FPGA running at 500 MHz.*

Discussion and Conclusion

Barrel processing is a technique allowing superior aggregated raw performance in multi-threaded processor systems. In particular systems using FPGA softlogic can benefit from the barrel approach. This holds for both pure softlogic implementations and CPU implementations using reconfigurable custom instructions.

Future work will focus on refining thread scheduling policies, developing dynamic thread-management schemes, and integrating specialized functional units tailored to specific application domains (e.g., signal processing, cryptography).

References

[1] C. Celio et al. *BOOM v2: an open-source out-of-order RISC-V core.* Tech. rep. UCB/EECS-2017-157. 2017.

[2] E. Tarassov. *AMD/Xilinx Vivado block designs for FPGA RISC-V SoC running Debian Linux distro.* 2024.

[3] E. T. *BROOM An open-source out-of-order processor with resilient low-voltage operation in 28nm CMOS.* 2018.

[4] S. Mashimo et al. "An Open Source FPGA-Optimized Out-of-Order RISC-V Soft Processor". In: *ICFPT*. 2019.

[5] J. Gray. "GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Framework". In: *Hot Chips*. 2017.

[6] P. Yiannacouras et al. "The Microarchitecture of FPGA-Based Soft Processors". In: CASES '05. ACM, 2005.

[7] M. R. Guthaus et al. "OpenRAM: An Open-Source Memory Compiler". In: *ICCAD*. 2016.

[8] eFabless. `https://openlane.readthedocs.io/en/`.

[9] D. Koch. "FABulous: An Embedded FPGA Framework". In: FPGA '21. https://fabulous.readthedocs.io/. 2021.

[10] N. Dao et al. "FlexBex: A RISC-V with a Reconfigurable Instruction Extension". In: *FPT*. IEEE. 2020.

[11] Riadh Ben Abdelhamid. *BRISKI RISC-V Barrel Processor.* 2024. URL: `https://github.com/riadhbenabdelhamid/BRISKI`.

[12] R. B. Abdelhamid and D. Koch. "BRISKI: A RISC-V barrel processor approach for higher throughput with less resource tax". In: *MCSoC*. 2024.

[13] R. B. Abdelhamid, et al. "SPARKLE: A 1,024-Core/16,384-Thread Single FPGA Many-Core RISC-V Barrel Processor Overlay". In: *ASAP*. 2024.