POLITECNICO DI TORINO
Repository ISTITUZIONALE

Seeing Beyond the Order: a LEN5 to Sharpen Edge Microprocessors with Dynamic Scheduling

Original
Seeing Beyond the Order: a LEN5 to Sharpen Edge Microprocessors with Dynamic Scheduling / Caon, M., Petrolo, V., Mirigaldi, M., Guella, F., Masera, G., Martina, M.. - ELETTRONICO. - (2024), pp. 47-50. (21st ACM International Conference on Computing Frontiers Ischia (Italy) May 7 - 9, 2024) [10.1145/3637543.3652880].

Availability:
This version is available at: 11583/2989944 since: 2024-10-07T15:55:08Z

Publisher:
ACM

Published
DOI:10.1145/3637543.3652880

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
ACM postprint/Author's Accepted Manuscript

© ACM 2024. This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in CF '24 Companion: Proceedings of the 21st ACM International Conference on Computing Frontiers: Workshops and Special Sessions, http://dx.doi.org/10.1145/3637543.3652880.

(Article begins on next page)

12 October 2026

Seeing Beyond the Order: a LEN5 to Sharpen Edge Microprocessors with Dynamic Scheduling

Michele Caon[†]
michele.caon@polito.it

Vincenzo Petrolo[†]
vincenzo.petrolo@polito.it

Mattia Mirigaldi[†]
mattia.mirigaldi@polito.it

Flavia Guella[†]
flavia.guella@polito.it

Guido Masera[†]
guido.masera@polito.it

Maurizio Martina[†]
maurizio.martina@polito.it

[†]Politecnico di Torino
Turin, Italy

ABSTRACT

In recent years, the shift towards data-driven workloads has underscored the limitations of traditional Von Neumann embedded computers and centralized processing infrastructures. Heterogeneous embedded Systems on Chip have emerged as a promising alternative offering the performance and energy efficiency benefits of specialized accelerators alongside the versatility of CPU-based systems. However, optimizing operation scheduling and resource utilization at compile time remains a challenging task. In this context, modular Instruction Set Architectures like RISC-V enable the development of tightly-coupled coprocessors that share the code with the host CPU. Techniques exploiting Instruction-Level Parallelism can mitigate the high latency of specialized hardware by dynamically reordering and speculatively executing instructions. This paper presents the first iteration of LEN5, a 64-bit RISC-V microprocessor featuring a modular, dynamically scheduled execution pipeline with Out-of-Order execution and commit. Preliminary implementation figures and benchmarking results over the Embench suite show significant improvements in Instructions Per Cycle of more than 20 % compared to simpler in-order microarchitectures. Additionally, LEN5 achieves a 20 % higher operating frequency when integrated into a small, edge-oriented microcontroller. The 64-bit architecture also enables up to a 2.4 × reduction in the number of instructions required to execute precision-sensitive workloads.

CCS CONCEPTS

• **Computer systems organization → Reduced instruction set computing**; • **Hardware**;

KEYWORDS

CPU Microarchitecture, RISC-V, Out-of-Order, Dynamic Scheduling

ACM Reference Format:
Michele Caon, Vincenzo Petrolo, Mattia Mirigaldi, Flavia Guella, Guido Masera, and Maurizio Martina. 2024. Seeing Beyond the Order: a LEN5 to Sharpen Edge Microprocessors with Dynamic Scheduling. In *Proceedings of XXXX, XXXX, 2024, XXXX*. ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX/XXXXXXX.XXXXXXX

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
XXXX, XXXX, 2024, XXXX
© 2024 Copyright held by the owner/author(s).
ACM ISBN XXXXXXXXXXXXXXXXXXXXXXX
https://doi.org/XXXXXXX/XXXXXXX.XXXXXXX

1 INTRODUCTION

While the advancements in the semiconductor manufacturing process have yielded undeniable benefits since the introduction of embedded computers in the 1970s, they have also represented a paradoxical limitation to innovation in the field of computer architecture. The status quo was disrupted in the last decades by the combined action of a shift in the software programming paradigm towards data-driven algorithms and the ever-increasing difficulties and costs involved when trying to keep pace with Moore's Law. The inherent limitations of traditional Von Neumann architectures have amplified the imperative to relocate computation closer to the data source, driven by the saturation of network infrastructure within the existing centralized computing paradigm.

Heterogeneous embedded Systems on Chip (SoCs) like [3] have emerged as a promising solution to the performance and energy efficiency challenges faced by edge devices. In such systems, computationally intensive tasks are delegated to specialized domain- or application-specific accelerators, offering superior execution speed and energy efficiency compared to the main Central Processing Unit (CPU). These accelerators are typically integrated as memory-mapped peripheral units, communicating with the host CPU via the system bus. Consequently, dedicated software libraries and drivers are essential for data exchange and control of their operation. However, optimizing scheduling and memory bandwidth utilization presents significant challenges, often leading to underutilization of the accelerator's potential performance and energy benefits.

In this context, modular and extendable Instruction Set Architectures (ISAs) offer an alternative solution by enabling the integration of custom domain-specific instruction set extensions alongside tightly-coupled Execution Units (EUs) or coprocessors which share both code and memory space with the host CPU. This approach enhances efficiency compared to memory-mapped devices and benefits from compiler assistance in automatically mapping and scheduling accelerated tasks within the program. However, offloaded instructions usually implement computationally intensive operations that result in a latency, possibly variable, of tens to thousands of cycles. Optimizing coprocessor utilization while mitigating stalls on the host CPU presents therefore challenges that cannot be fully resolved at compile time. This issue is particularly pronounced in scenarios where the same ISA extension

may be implemented differently across various microarchitectures. Instruction-Level Parallelism (ILP) and dynamic execution scheduling are valuable tools that embedded systems can borrow from application-class and high-performance computing systems to improve the overall system performance and hide the latency of the offloaded instructions. The same benefits also apply to covering the access time to memory-mapped peripherals. On the other hand, dynamic scheduling implies more complex control and consequently a significant increase in area and energy consumption [7]. Therefore, it is important to balance the trade-off between the performance improvement and the efficiency degradation. Modularity and configurability are key features to mitigate this issue, allowing the designer to tailor the processor characteristics to the specific deployment requirements and the expected workload.

This paper presents LEN5, a highly configurable, modular, speculative, 64-bit RISC-V microprocessor featuring in-order issue, Out-of-Order (OoO) execution and OoO commit. The main features of LEN5 architecture presented in Section 2 are:

- A *scalable and modular core infrastructure* that allows for easy configuration to meet various computing requirements.
- Efficient *latency masking* and *dependency handling* to ensure high utilization of the EUs with diverse workloads.
- Wide *configurability* to tailor the base microarchitecture to the area and timing constraints of the target application.

The preliminary results when running the Embench benchmark suite, discussed in Section 3, highlight the potential benefits of the proposed architecture compared to simpler in-order architectures. On the other hand, they expose some limitations of the current implementation when dealing with workloads with a complex and data-dependent control flow. The area and timing characteristics obtained show the potential for a significant improvement in terms of clock frequency at the cost of a moderate system-level area increase when targetting small, edge-oriented SoCs.

LEN5 RTL description and benchmarking software is available under an open-source library at: https://github.com/vlsi-lab/len5.

2 CPU MICROARCHITECTURE

The primary goal of LEN5 microarchitecture is to provide a modular core infrastructure that can a) be easily configured and extended to meet diverse computing needs, and b) mask the latency of and dependencies of any executed instruction to guarantee high performance even when hosting serial or heavily pipelined EUs. To achieve this, LEN5 leverages a combination of branch prediction in its fetch stage (based on [1]), speculative OoO execution and OoO commit in its backend. The top-level block diagram of the microprocessor is shown in Fig. 1. In its current state, LEN5 minimal configuration supports the `RV64I` base instruction set and the `Zicsr` ISA extension. The `M` extension can be optionally enabled, with or without a hardware integer divider. Both a 32-bit serial and a 64-bit pipelined divider are currently available. The choice to implement the 64-bit instruction set was made to cover those applications that benefit from larger operands and to pave the way for broader Operating Systems (OSs) support [4]. The possibility to switch to a 32-bit version of the core is still under development.

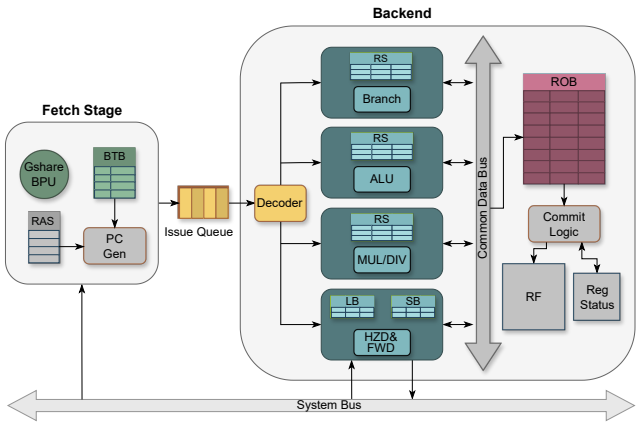

Figure 1: Block diagram of the LEN5 microprocessor.

2.1 Instruction Fetch

The *gshare* branch predictor and the Branch Target Buffer (BTB) in the frontend Branch Prediction Unit (BPU) speculatively update the Program Counter (PC) when known branch and jump instructions are fetched to cover the latency of branch and jump resolution in the backend. An early jump-and-link decode unit featuring a Return Address Stack (RAS) also recognizes unknown static jumps (`jal`) and known subroutine return instructions (`ret`) and speculatively updates the PC in advanced to mitigate the jump misprediction penalty in these cases as well. New instructions are finally pushed to an Issue Queue (IQ), waiting to be decoded and dispatched for execution by the main instruction decoder in the backend. To recover faster from branch or jump mispredictions, the fetch unit is notified of the misprediction as soon as the Branch Unit (BU) detects it, so the PC can be updated to the correct target address and the IQ can be populated with new instructions while the backend waits for all previous instructions to complete before flushing the in-flight ones.

2.2 Instruction Execution

The microarchitecture of the OoO instruction execution pipeline is based on an enhanced version [2] of Tomasulo's approach to dynamic scheduling [6]. Instructions are decoded in program order and dispatched to a buffer, referred to as Reservation Station, inside the target EU. At the same time, they are pushed into the ReOrder Buffer (ROB), where they wait for execution completion and commit. Each instruction source operand can be forwarded from the Common Data Bus (CDB) or the ROB if produced by some previous instruction that has already been executed, or fetched from the register file if no in-flight instruction is writing the corresponding register. If the instruction operands are not all available at dispatch time (i.e., they are produced by an in-flight instruction that has not been executed yet), the instruction waits for them in the target Reservation Station (RS) until they are broadcast on the CDB once the producing instruction completes its execution. During this time, instructions in the same RS whose operands become available can be executed, possibly out of program order. All the in-flight instructions write their result in the ROB using the CDB, which also broadcasts it to all the waiting RSs. This enables dynamic OoO

execution scheduling while resolving all the operand dependencies among instructions without the need for a large centralized instruction status table. As a consequence, the only modifications that are required to add hardware support for custom extensions are to extend the main instruction decoder and add dedicated RS and compute engine to the backend. LEN5 microarchitecture provides all that is necessary to handle hazards, operand forwarding, and result commit regardless of the latency of the new EU. Also, thanks to the distributed control flow and execution isolation, adding new EUs does not significantly impact the timing performance of the system, resulting in a highly scalable microarchitecture.

2.3 Instruction Commit

LEN5 commit stage picks instructions that have completed their execution from the ROB and schedules them for commit. While the ROB keeps track of the oldest instruction and prioritize it for commit, an alternative commit slot can select a newer instruction for commit, provided that it does not cause Write-After-Write (WAW) hazards on the register file and all previous instructions can no longer disrupt the execution flow (e.g., by triggering an exception or a branch misprediction). This allows LEN5 to commit instructions out of program order whenever the oldest instruction in the ROB has not yet completed its execution. The commit of memory access instructions (loads and stores) is handled separately by the LEN5 Load-Store Unit (LSU). The LSU contains an OoO Load Buffer (LB) and an in-order Store Buffer (SB). Both buffers support outstanding memory requests and OoO memory responses, regardless of the memory access latency. Besides working as RS for store instructions, the SB is also used as a *level-zero* cache by holding committed store instructions until new space is needed and forwarding their store value to any future same-width load that accesses the same memory address, saving a load access to the memory hierarchy.

3 EXPERIMENTAL RESULTS

The first part of this section analyses LEN5 area and timing characteristics obtained from the logic synthesis. It compares them with the 32-bit in-order `cv32e40p` core [5] to evaluate the implementation cost of supporting the 64-bit RISC-V ISA. The impact of integrating LEN5 in a small, edge-oriented Microcontroller Unit (MCU) system is also evaluated by taking the X-HEEP MCU [3] as a reference. Later, the performance of LEN5 in terms of Instructions Per Cycle (IPC) is evaluated using cycle-accurate Register Transfer Level (RTL) simulation when running the Embench benchmark.

3.1 Logic Synthesis

LEN5 synthesis is performed using Synopsys Design Compiler (DC) with the TSMC 65 nm LP CMOS technology library, suitable for power-constrained edge devices, using worst-case operating conditions. The modularity of LEN5 is showcased by synthesizing three core configurations targetting deployment scenarios with very different area and performance constraints. The first variant, shown in Table 1 as *Max Perf* features supports the `rv64im` ISA with a 2-stage pipelined 64-bit multiplier and a serial divider, and achieves the highest performance on disparate application benchmarks while relaxing the area constraints. The size of the data structures is tailored to minimize structural hazards in the EU RSs, support up to 32 in-flight instructions and 16 cached stores to reduce stalls due to long latency division operations and memory dependencies. *Min Area* is an alternative variant suitable for area-constrained systems and workloads that tolerate longer execution times due to the absence of the hardware multiplier and divider. The *Avg Perf* variant is an intermediate variant featuring no divider and a 32-bit multiplier executing 64-bit operations in 3 cycles to 4 cycles, still achieving comparable IPC to *Max Perf* on most applications while limiting area. Table 1 shows that the *Min Area* and *Avg Perf* variants have respectively $0.5\times$ and $0.6\times$ the area of *Max Perf*. The different area contributions of *Max Perf* configuration are analyzed in Figure 3. The parallel multiplier accounts for the highest portion of the EUs area, followed by the Arithmetic Logic Unit (ALU) due to its 8-instruction RS. The serial divider takes about 7 % of the area of the Execution Stage. This analysis motivates our choices to configure the *Avg Perf* variant with a serial multiplier, that has a $1.8\times$ smaller area than the parallel one at the expense of throughput, a 4-instruction in place of an 8-instruction RS for the ALU, and not include a hardware divider. Two timing scenarios are considered for the logic synthesis of LEN5 and `cv32e40p`, as shown in Table 1: (1) a bus input delay of 2.2 ns and an output delay of 0.4 ns to reproduce a common MCU memory configuration using single-port, 32kiB, 65 nm SRAM banks under worst-case operating conditions; (2) an input and output delay of 0 ns to obtain best-case timing performance. In scenario (1), the memory input delay determines the critical path delay for both the CPUs, except for LEN5 *Avg Perf* variant where the serial multiplier dominates. When the memory input and output delays are neglected, the critical path of the *Max Perf* variant is in the pipelined multiplier. The maximum frequency the core can achieve at 65 nm is 578 MHz.

Table 1: Area and Clock Frequency Comparison

	Memory Input Delay [ns]	**Max Perf**	**Avg Perf**	**Min Area**	**`cv32e40p`**
Conf [1]		[4,4,8,4,8,4,4, 4,1,0,1,16,8,32]	[4,4,8,2,4,4,-, 4,1,1,0,8,4,8]	[4,4,8,2,4,-,-, 4,0,-,0,8,4,8]	-
Clk Freq [MHz]	2.2	438	406	448	360
Area [μm^2]		422879	251329	204928	56064
Area [kGE] [2]		294	174	142	39
Clk Freq [MHz]	0	490	394	578	465
Area [μm^2]		395352	223211	200117	55908
Area [kGE] [2]		274	155	139	39

[1] The *Conf* string encodes the size of the data structures in this order: BTB bits, BPU bits, RAS size, IQ size, ALU RS size, MULT RS size, DIV RS size, BU RS size, MULT (1 if instantiated), MULT arch. (0: pipelined, 1: serial), DIV (1 if present), SB size, LB size, ROB size.
[2] GE is the 2-input drive strength-one NAND gate equivalent area.

The obtained results are compared with those for the `cv32e40p` core synthesised with the same technology library and timing scenarios. Every variant of our core reaches a higher clock frequency in scenario (1) since LEN5 samples the incoming data or instruction from the memory before forwarding or decoding them. Overall, LEN5 shows a 5 % to 20 % frequency improvement. An estimation of the overhead of inserting our core into a MCU system is performed to make fair considerations on the area overhead compared to `cv32e40p`. The open-source, edge-oriented MCU system X-HEEP with 8×32 KiB SRAM banks and the `cv32e40p` CPU, synthesised at a 4 ns clock period, is taken as a reference. Replacing `cv32e40p` with LEN5 *Max Perf* as the system CPU would cause an overall area increase of 12.7 %. This overhead is acceptable considering that the performances of our CPU are estimated, by looking at Figure 3, to increase with a super-linear trend compared to the area when

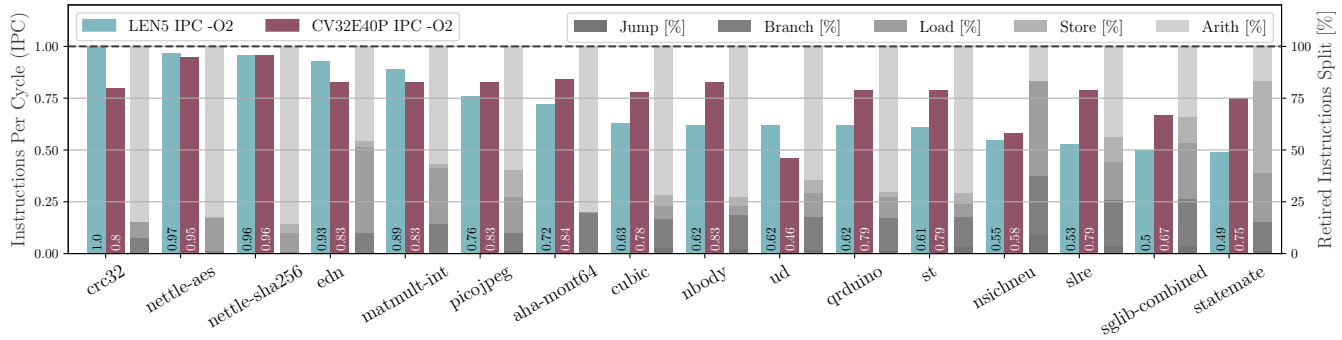

Figure 2: Instructions Per Cycle comparison with `cv32e40p` over the Embench suite* (colour) and executed instruction composition (greyscale).
*`huffbench` and `minver` did not finish on LEN5 and `cv32e40p` respectively.

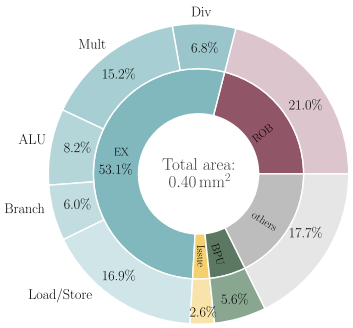

Figure 3: Area Partitions for the *Max Perf* LEN5 variant in Table 1.

adding multiple-issue support. This would mainly impact the issue and commit control logic, which represent a small portion of the total area. Moreover, LEN5 area is comparable with that of the known 64-bit `cva6` core [7]: 210 kGE.

3.2 Benchmarking

LEN5 IPC performance is compared to the `cv32e40p` 32-bit in-order RISC-V when integrated inside the X-HEEP microcontroller when running the Embench benchmark suite. All LEN5 results were obtained using the *Max Perf* variant with a single pipeline register in the multiply unit. A fair comparison is ensured by configuring X-HEEP with a parallel bus and a memory layout to support parallel instruction and data memory accesses. For both CPUs, the code was compiled using GCC with the `-O2` optimization level and the appropriate ISA flags: version 13.2 with `rv64im` for LEN5 and version 11.1 `rv32imc` for `cv32e40p`. The IPC is measured through the performance counters `mcycle` and `minstret`. The open-source tool Verilator (version 4.210) is used for the RTL simulation. Figure 2 reports the IPC results of the benchmarking campaign. As expected, LEN5 demonstrates higher IPC, particularly in benchmarks with low control-flow orientation. Notably, the `crc32` benchmark stands out with an IPC of 1.0 and an improvement of more than 20 %, showcasing the significant benefits of LEN5's dynamic instruction re-ordering capabilities that succeeded in hiding data dependencies. Superior IPC performances are observed in tests like `edn` and `matmult-int` for vector and matrix multiplication. Here, LEN5 can compensate for most of the 2-cycle latency of memory accesses, representing 30 % to 40 % of the executed instructions. However, LEN5 exhibits lower IPC as the benchmarks include more unpredictable and data-dependent jumps, where the misprediction penalty is higher than `cv32e40p` and not compensated by the BPU. This is due to a known synchronization issue between LEN5 BPU and BU that causes frequent jump mispredictions. The `aha-mont64` benchmark, on the other hand, represents a use case that greatly benefits from implementing the 64-bit instruction set: when compiled for LEN5, this benchmark retires 2.4 × fewer instructions than the equivalent 32-bit version for `cv32e40p`, resulting in significantly lower execution time despite the slightly lower IPC. The same applies for `statemate` (2 ×), `st` (1.4 ×), and `matmult-int` (1.6 ×).

4 CONCLUSIONS

This paper introduces a versatile RISC-V CPU featuring OoO execution and commit, demonstrating notable gains in IPC compared to simpler architectures, along with increased clock frequencies. Future work will focus on enhancing branch prediction to handle more complex scenarios, supporting multiple issue capabilities, and conducting comparative analyses against leading-edge OoO CPUs.

ACKNOWLEDGMENTS

This work was supported in part by the Key Digital Technologies Joint Undertaking and Its Members through the TRISTAN Project Under Grant 101095947.

REFERENCES

[1] Marco Andorno. 2019. Design of the frontend for LEN5, a RISC-V Out-of-Order processor. https://webthesis.biblio.polito.it/13198/
[2] Michele Caon. 2019. Design of the execution pipeline for LEN5, an out-of-order RISC-V processor. https://webthesis.biblio.polito.it/13205/
[3] Simone Machetti, Pasquale Davide Schiavone, Thomas Christoph Müller, et al. 2024. X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators. *arXiv preprint arXiv:2401.05548* (2024).
[4] Matteo Perotti. 2019. Design of an OS compliant memory system for LEN5, a RISC-V Out of Order processor. http://webthesis.biblio.polito.it/13231/
[5] Pasquale Davide Schiavone, Francesco Conti, Davide Rossi, et al. 2017. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. In *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 1–8.
[6] Robert M Tomasulo. 1967. An efficient algorithm for exploiting multiple arithmetic units. *IBM Journal of Research and Development* 11, 1 (1967), 25–33.
[7] Florian Zaruba and Luca Benini. 2019. The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 11 (2019), 2629–2640.