

Software and Hardware of Computer, Network, Telecommunication, Control, and Measurement Systems

Компьютерные сети, вычислительные, телекоммуникационные, управляющие и измерительные систем

Research article

DOI: <https://doi.org/10.18721/JCSTCS.19310>

UDC 004.2



APPLICATION OF ASYNCHRONOUS DATA BYPASS TO INCREASE PERFORMANCE OF IN-ORDER SUPERSCALAR RISC-V PROCESSOR CORE

M.D. Usoltsev¹ , F.K. Borisovskii², D.O. Budanov¹ 

¹ Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation;

² CloudBEAR, LTD, St. Petersburg, Russian Federation

 usol.91@mail.ru

Abstract. RISC-V is an open-source architecture that does not require license fees from processor core vendors. This allowed RISC-V to become widely distributed first in the microcontroller-class processor core market and subsequently in the market of higher-performance devices. Currently RISC-V processor cores with in-order execution, which combine mid-range performance, compactness, and energy efficiency are attracting the most interest from customers. This makes the task of continuing to improve them a highly relevant. This paper explores the application of a microarchitectural modification, asynchronous data bypass, to increase the performance of in-order superscalar RISC-V processor core. Asynchronous data bypass has been made for two pairs of functional blocks: between arithmetic-logic unit (ALU) and ALU, between ALU and branch unit. The main obstacle of implementing asynchronous data bypass is the creation of an extremely long logical path, which leads to violations of timing constraints. To overcome this obstacle, an approach was proposed that involves completely excluding a subset of instructions from asynchronous bypass and separating the logic of the remaining instructions into groups based on the length of the logical path. This approach made it possible to keep the logical path of asynchronous bypass no longer than the longest path of a single execution unit. The results showed that the use of asynchronous bypass can significantly increase the performance of in-order superscalar processor core. At the same time, the area overhead is very low.

Keywords: processor core, superscalarity, in-order, asynchronous data bypass, logical path

Citation: Usoltsev M.D., Borisovskii F.K., Budanov D.O. Application of asynchronous data bypass to increase performance of in-order superscalar RISC-V processor core. Computing, Telecommunications and Control, 2026, Vol. 19, No. 3, Pp. 121–129. DOI: 10.18721/JCSTCS.19310

Научная статья

DOI: <https://doi.org/10.18721/JCSTCS.19310>

УДК 004.2



ИСПОЛЬЗОВАНИЕ АСИНХРОННОЙ ПЕРЕСЫЛКИ ДАННЫХ ДЛЯ УВЕЛИЧЕНИЯ ПРОИЗВОДИТЕЛЬНОСТИ СУПЕРСКАЛЯРНОГО RISC-V ПРОЦЕССОРНОГО ЯДРА С ИСПОЛНЕНИЕМ КОМАНД В ПРЯМОЙ ОЧЕРЕДНОСТИ

М.Д. Усольцев¹ , Ф.К. Борисовский², Д.О. Буданов¹ 

¹ Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация;

² ООО «Клаудбейр» (ООО «КБ»), Санкт-Петербург, Российская Федерация

✉ usol.91@mail.ru

Аннотация. RISC-V является открытой процессорной архитектурой и не требует лицензионных отчислений от разработчиков процессорных ядер. Такой подход позволил RISC-V широко распространиться сначала на рынке процессорных ядер микроконтроллерного класса, и впоследствии выйти на рынок более производительных устройств. К настоящему моменту наибольший интерес у потребителей вызывают RISC-V процессорные ядра с исполнением команд в прямой очередности, сочетающие в себе производительность среднего уровня, компактность и энергоэффективность. Данное обстоятельство делает актуальной задачу их дальнейшего усовершенствования. В данной статье исследовано применение микроархитектурной модификации, асинхронной пересылки данных, для увеличения производительности суперскалярного RISC-V ядра с исполнением команд в прямой очередности. Рассмотрено внедрение асинхронной пересылки данных для двух пар функциональных блоков: между арифметико-логическим устройством (АЛУ) и АЛУ, между АЛУ и блоком обработки переходов. Главным препятствием при введении асинхронной пересылки является возникновение чрезмерно длинного логического пути, приводящего к нарушению временных ограничений. Для преодоления данного обстоятельства предложен подход, заключающийся в полном исключении части инструкций из асинхронной пересылки и разделении логики оставшихся инструкций на группы по длине логического пути. Данный подход позволил сделать логический путь пересылки не длиннее, чем самый длинный путь одиночного исполнительного блока. Полученные результаты показывают, что применение асинхронной пересылки данных может ощутимо увеличить производительность суперскалярного процессорного ядра с исполнением команд в прямой очередности. При этом затраты по площади являются незначительными.

Ключевые слова: процессорное ядро, суперскалярность, прямая очередность, асинхронная пересылка данных, логический путь

Для цитирования: Usoltsev M.D., Borisovskii F.K., Budanov D.O. Application of asynchronous data bypass to increase performance of in-order superscalar RISC-V processor core // Computing, Telecommunications and Control. 2026. Т. 19, № 3. С. 121–129. DOI: 10.18721/JCSTCS.19310

Introduction

The RISC-V architecture had been developed in 2010, and in 2011 the first version of its base instruction set was published¹. Open source and the absence of licensing fees for using the architecture were declared to be the main principles of this architecture [1]. These conditions, which provide greater independence for processor core developers, allowed RISC-V to gain attraction in the market for energy-efficient microcontrollers and subsequently expand into the market for higher-performance cores [2–4].

Most of the implemented RISC-V processor cores has in-order execution [5]. In-order processor core executes program instructions strictly in the order in which they are located in the execution flow. If the current instruction cannot be executed, the entire core pipeline stops and waits until that instruction is completed. On the other hand, out-of-order processor cores analyze the dependencies between instructions and reorder them to achieve maximum performance. In-order processor cores have less performance than out-of-order processor cores. However, they are simpler to design, take up less area and consume less energy [6, 7].

RISC-V in-order microcontroller-class processor cores are currently widely available on the market^{2,3,4}. They are low-performance cores designed for maximum compactness and autonomous operation. They are used in the Internet of Things (IoT), various sensors, meters and simple Bluetooth devices [8, 9]. The other fast-growing class is higher-performance in-order cores. They are intended for use as coprocessors in data storage devices, energy-efficient components in systems on a chip, in automotive security systems, in industrial machine tools, in communication systems, and for devices that require an operating system: network routers and gateways, database servers, industrial terminals and access panels with an advanced graphical user interface [10–15].

Compared to microcontrollers, these processor cores typically feature an additional instruction execution pipeline (superscalar architecture), a greater number of pipeline stages, improved branch predictors and cache [16–18]. At the same time, improving the cores in this class remains a priority. Of particular interest are modifications that increase performance without significantly increasing the core area.

In this paper, we investigate a microarchitectural modification: asynchronous data bypass. It will be implemented for some execute units of the RISC-V in-order superscalar core. It is assumed that the introduction of this data pass-through will help eliminate some of the stalls in the core caused by dependencies between instructions entering the core's pipeline in a single clock cycle. This modification is expected to increase the core's performance while only slightly increasing its area.

Materials and methods

The microprocessor core used in this work belongs to a class of cores that offer higher performance than microcontrollers and are capable of running an operating system. It is an in-order superscalar processor core. The instruction execution pipeline of this core is shown in Fig. 1.

The instruction execution pipeline includes the following execution units:

- The arithmetic-logic unit (ALU) performs arithmetic and logical operations in a single clock cycle. The instruction execution subsystem contains two ALUs each in the E0 and CM stages. ALU pairing allows two ALU instructions to be executed in a single clock cycle. Additional ALUs at the CM stage enable ALU instructions to be sent to the CM stage and help avoid some of the stall time caused by dependencies.

¹ About RISC-V International. [online] Available: <https://riscv.org/about/> (Accessed 28.09.2026)

² PicoRV32 – A Size-Optimized RISC-V CPU. [online] Available: <https://github.com/YosysHQ/picorv32> (Accessed 28.09.2026)

³ A FPGA friendly 32 bit RISC-V CPU implementation. [online] Available: <https://github.com/SpinalHDL/VexRiscv> (Accessed 28.09.2026)

⁴ FEMTORV32 / FEMTOSOC: a minimalistic RISC-V CPU. [online] Available: <https://github.com/BrunoLevy/learn-fpga/tree/master/FemtoRV> (Accessed 28.09.2026)

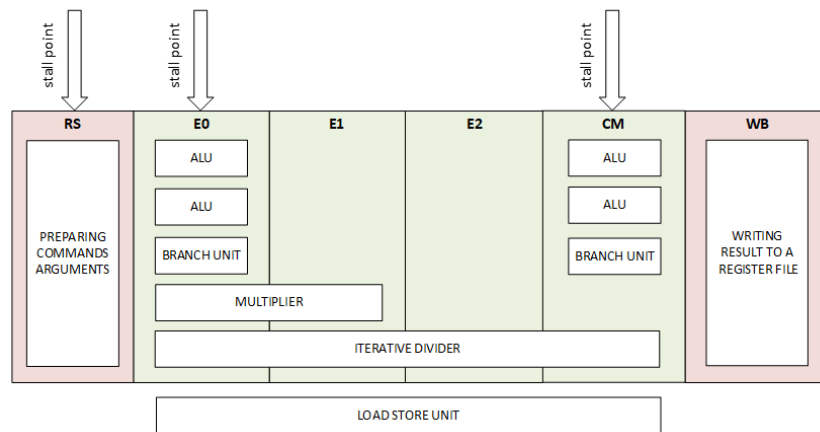


Fig. 1. Schematic of instruction execution pipeline

- The branch unit executes conditional branch instructions, in a single clock cycle. The instruction execution subsystem contains two BPUs each at the E0 and CM stages.
- The multiplier performs multiplication operations in two clock cycles. There is no duplication of this unit in the pipeline.
- The iterative divider performs division operations. The number of cycles required to complete the division depends on the number of bits in the dividend. This unit is not duplicated in the pipeline.
- The load store unit executes instructions that require interaction with memory.

The presence of two execution paths and two ALUs at the stages makes it possible to execute two ALU instructions simultaneously. However, for in-order cores, a stall occurs when two ALU instructions with Read after Write (RAW) dependency enter the pipeline as shown in Fig. 2, *a*. Two instructions have entered the pipeline, and there is a dependency between them. The result of the first instruction in the execution order (slot 0) is an argument for the second (slot 1). Because of this dependency, the instruction in slot 1 stalls for one clock cycle until the result of the instruction in slot 0 becomes ready.

We propose implementing asynchronous data bypass between ALUs. This allows two dependent instructions to move on to the E0 stage together, as shown in Fig. 2, *b*. In this case, the instruction in slot 1 receives data from the instruction in slot 0 directly during execution through the bypass.

An asynchronous bypass can also be added for ALUs on the CM stage. The case of sending ALU instructions to the CM stage is shown in Fig. 3. The instruction in slot 0 depends on the load from memory instruction (lw) and gets sent to the CM stage. At the same time, the asynchronous bypass allows the instruction in slot 1 to also be sent to the CM stage and receive data from instruction in slot 0.

Implementing direct connection between ALUs adds a new logical path to the system. This path could potentially become the longest in the system in terms of signal propagation time through the logic gates (critical path) and lead to a violation of timing constraints (Fig. 4, *a*). To avoid violating timing constraints, the following condition was set when organizing bypass: the logical path between two ALUs must not be longer than the longest path of a single ALU.

It was determined that the following instructions have the longest logical path in the single ALU of the original core:

- cpop – counts the number of logical ones in the register;
- clz – counts the logical zeros in the register up to the first logical one, starting from the most significant bit;
- ctz – counts the number of logical zeros in the register up to the first logical one, starting from the least significant bit.

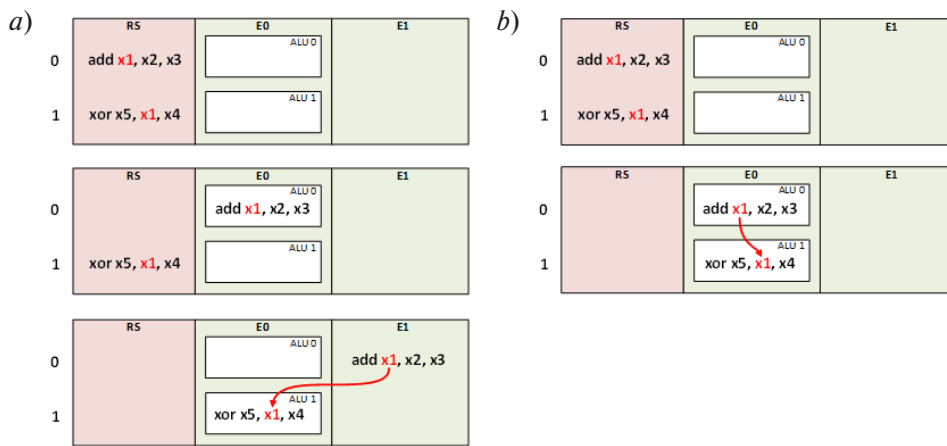


Fig. 2. Execution of two ALU instruction with dependency: without asynchronous bypass (a); with asynchronous bypass (b)

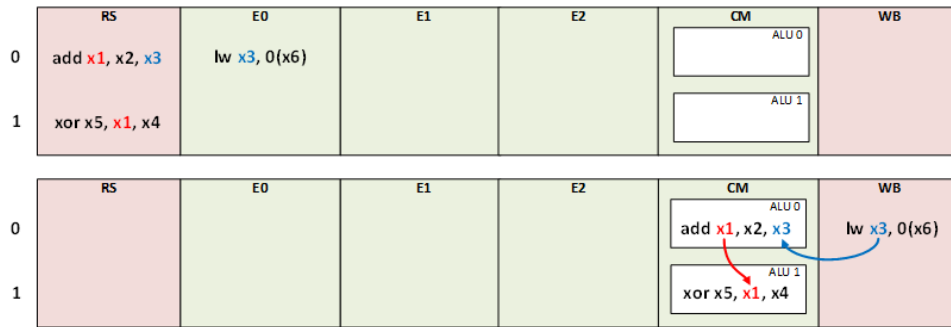


Fig. 3. Execution of two ALU instruction with dependency with asynchronous bypass on CM stage

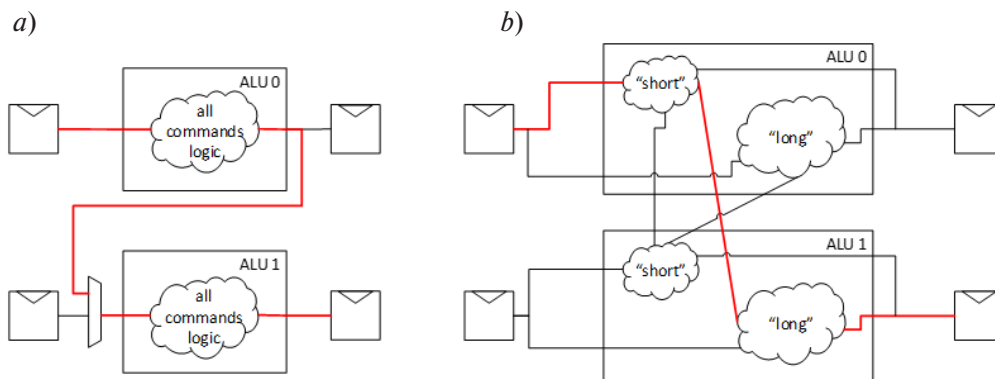


Fig. 4. Critical path of asynchronous bypass between ALUs: direct connection of two ALUs (a); critical path optimization by separating commands logic (b)

Adding these instructions would inevitably result in failure to meet the specified condition and violate timing constraints. Therefore, these instructions were completely excluded from the asynchronous data bypass. All other instructions executed on the ALU were separated into two groups depending on their logical path length: "short" and "long," and the asynchronous data bypass was organized as shown in Fig. 4, b.

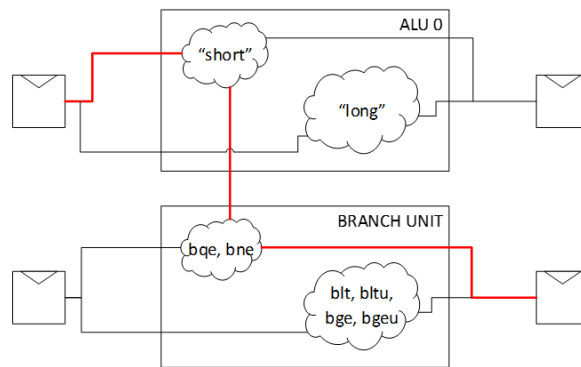


Fig. 5. Critical path of asynchronous bypass between ALU 0 and branch unit

The following instructions were classified as “short”:

- Logical instructions: and, or, xor, their versions with a constant argument – andi, ori, xori, their versions with an inverted argument – andn, orn, xorn.
- Single-bit instructions: set bit – bset, clear bit – bclear, invert bit – binv, get bit value – bext.
- Data extension instructions: sext.b – signed extension of the lower 8 bits, sext.h – signed extension of the lower 16 bits, zext.h – unsigned extension of the lower 16 bits.
- Byte instructions: orc.b – logical OR for all bits in a single byte, rev8 – reversing the byte order, brev8 – reversing the bit order within a single byte.
- Conditional assignment instructions: czero.eqz, czero.nez.

The remaining operations performed on the ALU were classified as “long”: addition, subtraction, comparisons, shifts. Their logic path is much longer than that of the “short” operations, but significantly shorter than that of the cpop, clz, and ctz operations.

We also tested the use of asynchronous data bypass between ALU 0 and branch unit. This time, it will allow for the simultaneous execution of an ALU instruction in slot 0 and a dependent conditional branch instruction in slot 1. The result of the instruction from ALU 0 is passed to one of the arguments of the conditional branch operation in the branch unit.

In this case, to meet timing constraints, the next condition was set: logical path between ALU 0 and the branch unit must not be longer than the longest path in the single branch unit. To satisfy this condition, only the following were added to the asynchronous data bypass:

- ALU 0 new “short” instructions: logical, data extension, byte instructions.
- Branch unit conditional branch instructions with an absolute equality condition: beq, bne, beqz, bnez.

The remaining instructions executed by the ALU and the branch instructions with longer logical paths (blt, bltu, bge, bgeu) were excluded. Thus, the asynchronous bypass between ALU 0 and branch unit was organized as shown in Fig. 5.

Results and discussion

After implementing the asynchronous bypass, performance measurements were taken relative to the original processor core. The following tests were used to measure performance: Coremark, Dhrystone, integer tests from the SPEC2006 and SPEC2017 collections. The results are shown in Fig. 6.

The test results show a noticeable performance gain when an asynchronous bypass is implemented. It can be seen that the results for Dhrystone increased more than the others. This is probably associated with the limited variety of operations in Dhrystone and its relatively small size. Due to

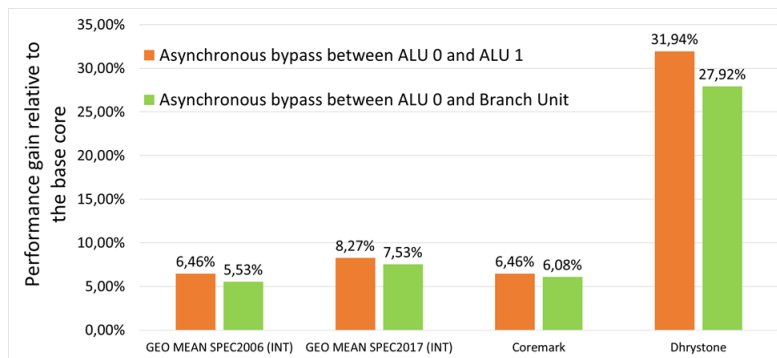


Fig. 6. Processor core performance gain after implementing of an asynchronous bypass

these factors, the acceleration of basic operations may be more pronounced in this test. Moreover, the introduction of an asynchronous bypass between ALU 0 and ALU 1 increased performance more significantly than the bypass between ALU 0 and the branch unit.

We also evaluated the increase in processor core area resulting from the introduction of an asynchronous bypass. A simulation-based synthesis was performed to estimate the processor core area with an approximation to its actual fabrication on a chip. The increase of the core area compared to the base one is shown in Table 1.

Table 1

Comparison of area increase relative to the base processor core

Modification	Core area increase
Asynchronous bypass between ALU 0 and ALU 1	2.35%
Asynchronous bypass between ALU 0 and branch unit	0.56%

The introduction of an asynchronous bypass between ALU 0 and ALU 1 resulted in a greater increase in area. This is because a significantly larger number of instructions were included in this asynchronous bypass. First, this put more pressure on the logic path of the bypass, increasing the area cost during simulation synthesis. Second, it led to increased area costs for data multiplexing and the formation of the bypass context.

Conclusion

In this work, we investigated the use of asynchronous data bypass. This microarchitectural modification was implemented to increase the performance of a superscalar in-order RISC-V processor core.

As part of this work, an asynchronous data bypass was implemented between the following execution units: ALU 0 and ALU 1, ALU 0 and branch unit. To meet the specified timing constraints, the following approach was proposed: instructions with the longest logical paths were excluded from the asynchronous bypass, and the logic of the remaining instructions was separated into groups based on logical path length. A greater performance gain was achieved by introducing a bypass between ALU 0 and ALU 1 than by introducing a bypass between ALU 0 and the branch unit. However, this also required a greater increase in core area: 2.35% against 0.56%, respectively.

Overall, the asynchronous data bypass succeeded in increasing the performance of higher-performance class core that already incorporated previously known modifications. At the same time, the area costs were relatively minor. This modification can be used to increase the performance of other similar cores as well.

REFERENCES

1. Chaver D., Harris S., Pinuel L., Kindgren O., Kakakhel Z., Owen C. The RISC-V FPGA (RVfpga) Teaching Package. *IEEE Access*, 2026, Vol. 14, Pp. 18455–18475. DOI: 10.1109/ACCESS.2026.3658743
2. Sharma M., Bhatnagar E., Puri K., Mitra A., Agarwal J. A survey of RISC-V CPU for IoT applications. *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*, 2022, Pp. 1–4. DOI: 10.2139/ssrn.4033491
3. Hassan Q.F., Sagahyroon A. RISC-V: A comprehensive overview of an emerging ISA for the AI-IoT era. In: *Advances in the Internet of Things* (ed. Q.F. Hassan), 2025, Pp. 244–284. DOI: 10.1201/978100350-6638-15
4. Liu Y., Ye K., Xu C.Z. Performance evaluation of various RISC processor systems: A case study on ARM, MIPS and RISC-V. *Cloud Computing – CLOUD 2021*, 2022, Vol. 12989, Pp. 61–74. DOI: 10.1007/978-3-030-96326-2_5
5. Gao J., Zhang J. Research and Design of RISC-V Four-Stage Out-of-Order Execution Processor. *2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, 2022, Pp. 1–3. DOI: 10.1109/ICSICT55466.2022.9963329
6. Fu Z., Tedeschi R., Ottavi G., Wistoff N., Fuguet C., Rossi D., Benini L. Ramping up open-source RISC-V cores: Assessing the energy efficiency of superscalar, out-of-order execution. *Proceedings of the 22nd ACM International Conference on Computing Frontiers*, 2025, Pp. 12–20. DOI: 10.1145/3719276.3725186
7. Chatzopoulos O., Papadimitriou G., Wong W.S., Gizopoulos D. Energy Efficiency of Out-of-Order CPUs: Comparative Study and Microarchitectural Hotspot Characterization of RISC-V Designs. *2023 IEEE International Symposium on Workload Characterization (IISWC)*, 2023, Pp. 216–220. DOI: 10.1109/IISWC59245.2023.00032
8. Cui E., Li T., Wei Q. RISC-V Instruction Set Architecture Extensions: A Survey. *IEEE Access*, 2023, Vol. 11, Pp. 24696–24711. DOI: 10.1109/ACCESS.2023.3246491
9. Schiavone P.D., Conti F., Rossi D., Gautschi M., Pullini A., Flamand E. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications. *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2017, Pp. 1–8. DOI: 10.1109/PATMOS.2017.8106976
10. Wheeler B. WD Rolls Its Own RISC-V Core: Western Digital Opens Superscalar Swerv CPU Source to Others. *The Linley Group: Microprocessor Report*, 2019, Pp. 1–3. [online] Available: https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/analyst-report/WD_Rolls_Its_Own_RISC-V_Core.pdf
11. Western Digital. RISC-V and Open Source Hardware Address New Compute Requirements. [online] https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/tech-brief/tech-brief-western-digital-risc-v.pdf (Accessed 28.07.2026)
12. Petrisko D., Gilani F., Wyse M., Jung D.C., Davidson S., Gao P. BlackParrot: An Agile Open-Source RISC-V Multicore for Accelerator SoCs. *IEEE Micro*, 2020, Vol. 40, No. 4, Pp. 93–102. DOI: 10.1109/MM.2020.2996145
13. Gonzalez A., Zhao J., Korpan B., Genc H., Schmidt C., Wright J. et al. A 16mm² 106.1 GOPS/W Heterogeneous RISC-V Multi-Core Multi-Accelerator SoC in Low-Power 22nm FinFET. *ESSCIRC 2021 – IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, 2021, Pp. 259–262. DOI: 10.1109/ESSCIRC53450.2021.9567768
14. Numan Ince M., Ledet J., Gunay M. Building An Open Source Linux Computing System On RISC-V. *2019 1st International Informatics and Software Engineering Conference (UBMYK)*, 2019, Pp. 1–4. DOI: 10.1109/UBMYK48245.2019.8965559
15. Gookyi D.A.N., Ryoo K. Selecting a Synthesizable RISC-V Processor Core for Low-cost Hardware Devices. *Journal of Information Processing Systems*, 2019, Vol. 15, No. 6, Pp. 1406–1421. DOI: 10.3745/JIPS.03.0129

16. Dörflinger A., Albers M., Kleinbeck B., Guan Y., Michalik H., Klink R., Blochwitz C., Nechi A., Berekovic M. A comparative survey of open-source application-class RISC-V processor implementations. *Proceedings of the 18th ACM International Conference on Computing Frontiers*, 2021, Pp. 12–20. DOI: 10.1145/3457388.345865
17. Li Y., Yu Z., Ji W. Design and Performance Optimization of a Risc-V Processor with Five-Stage Pipeline. *2025 IEEE 6th International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 2025, Pp. 1–5. DOI: 10.1109/AINIT65432.2025.11035005
18. Kra Y., Shoshan Y., Rudin Y., Teman A. HAMSA-DI: A Low-Power Dual-Issue RISC-V Core Targeting Energy-Efficient Embedded Systems. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024, Vol. 71, No. 1, Pp. 223–236. DOI: 10.1109/TCSI.2023.3323425

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Mikhail D. Usoltsev

Усольцев Михаил Дмитриевич

E-mail: usol.91@mail.ru

Fedor K. Borisovskii

Борисовский Фёдор Константинович

E-mail: feka09@gmail.com

Dmitry O. Budanov

Буданов Дмитрий Олегович

E-mail: budanov_do@spbstu.ru

ORCID: <https://orcid.org/0000-0002-5925-5628>

Submitted: 23.05.2026; Approved: 17.07.2026; Accepted: 07.09.2026.

Поступила: 23.05.2026; Одобрена: 17.07.2026; Принята: 07.09.2026.