Implementation of the RISC-V Architecture with the Extended Zbb Instruction Set

2022 International Ural Conference on Electrical Power Engineering (UralCon) | 978-1-6654-6652-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/UralCon54942.2022.9906776

D. Markov
HSE University
Moscow, Russia
markovdima43@gmail.com

A. Romanov
HSE University
Moscow, Russia
a.romanov@hse.ru

***Abstract*— RISC-V is a new and rapidly developing instruction set architecture, the main feature of which is its openness for the free use. For this reason, RISC-V is becoming more and more popular among companies with a wide range of specializations, both with hardware and software developers. The RISC-V architecture specification defines a set of standard extensions in addition to the basic instruction set architecture that can be useful in a variety of applications among which is the use in automation and process control systems in Electrical power engineering. One of the useful extensions is an extension for bit manipulation, whose instructions were originally invented by Intel and ARM in order to improve power consumption and performance. This work presents an implementation of the Zbb extension of basic bit manipulation in the Verilog language. The developed module is integrated into the schoolRISCV soft processor core. Its modeling and verification of correct operation were carried out using the ModelSim software. The processor was synthesized and prototyped on the Terasic DE1-SoC FPGA board. In addition, support for new instructions from the implemented extension was added to the assembler. Runtime simulator RARS was used also. The analysis of the performance of the extended core showed an average acceleration of the program execution by 29.9%; the amount of memory occupied was reduced by 37.5%. The increase in performance was achieved due to the consumption of a larger number of logic elements and registers.**

Keywords—RISC-V, ISA, Zbb, RTL, FPGA, RARS, microarchitecture.

I. Introduction

In such fields as cryptography, machine learning, digital signal processing, and the Internet of Things, one of the most used types of operations on data is bit manipulation [1-4], operating on individual bits of machine words. For example, they are the operations of bit shift, grouping of bits, counting leading/trailing zero bits, as well as the simplest logical gates AND, OR, NOT, and their combinations. One can also use bit manipulation instructions to perform arithmetic operations, such as finding the average of two integers or the maximum of them. The use of these operations can significantly optimize the power consumption and speed of the processor; instructions for bit manipulation were for the first time included by Intel and ARM as extensions in their x86 and ARM architectures, respectively [5].

The research leading to these results has received funding from the Basic Research Program at the National Research University Higher School of Economics (HSE University).

RISC-V [6] is a new and rapidly developing instruction set architecture (ISA) created in 2010 with the direct participation of David Patterson, the pioneer in the field of microprocessor architecture [7]. The main feature of this ISA is its openness. The architecture is available for free use both in education and in the industry, including commercial industrial implementations directly in the form of a custom chip or FPGA configuration, that could be used in automation and process control systems in Electrical power engineering. This is an undoubted advantage over the x86 and ARM architectures. At the same time, in terms of its capabilities, the RISC-V architecture is comparable to the same x86 and ARM [8]. A commercial architecture license can be very expensive (around 10–20 million dollars), while a RISC-V license does not need to be purchased. One should not confuse an architecture license with a microarchitecture license. The first is the right to create a core with its own microarchitecture that is compatible with the licensed architecture. Core licenses, starting with several tens of thousands of dollars, are much cheaper. The ability to create one's own core is especially relevant in the context of import substitution of the semiconductor companies Intel and AMD, the main players in the microprocessor market, now leaving Russia.

The purpose of the work is to develop a software module in the Verilog hardware description language (HDL) implementing bit manipulation extension of the RISC-V architecture. (The module can be integrated into a single-cycle 32-bit soft processor schoolRISCV core.)

II. Related Works

A large number of attempts have been made by various authors to develop a soft processor based on the RISC-V architecture without and with various standard extensions. There are few implementations of extension B for bit manipulation, and many of them do not contain the complete set of extension B instructions. Therefore, it is necessary to develop a module that implements the entire set of instructions from one of the sub-extensions of extension B.

Most of the scientific papers reviewed [9-12] are similar in that the developed processor is prototyped on FPGA. This can be explained by the fact that the use of FPGAs, in comparison with ASICs, is the optimal method for prototyping IP-cores. In this work, FPGA is also used for these purposes. Also, almost every work uses the Verilog or SystemVerilog language for the development.

978-1-6654-6652-3/22/$31.00 ©2022 IEEE

The advantage of some works, such as [13-15], is they provide a thorough analysis of processor performance using specialized benchmarking software.

The disadvantage of many of the works considered [13-17] is they have not studied the problem of automating the decoding of instructions in the assembler. The present work is different in that it solves this problem by adding support for new commands to the RARS assembler.

III. Bit Manipulation Extension

The RISC-V architecture is a RISC type. The basic architecture of the RV32I is 32 bits wide and defines 32 general purpose registers. A feature of RISC-V is its modular design. The specification defines a set of standard extensions, one of which is extension B for bit manipulation [18].

The extension B specification for bit manipulation is comprised of 4 sub-extensions. One of them is the Zbb extension for basic bit manipulation. It contains more instructions and has a wider scope, while Zba, Zbc, Zbs have a narrower focus. The Zbb sub-extension, like extension B itself, has not yet received wide community attention, so there are few attempts to implement it as a separate IP-core. Thus, in this work, the choice was made in favor of the Zbb sub-extension. The list of all instructions, contained in the Zbb extension for the 32-bit version of the architecture, is presented in Table I.

The implementation of all 18 machine instructions contained in the 32-bit version of the extension was developed [19]. The developed IP-core is a combinational module similar to ALU. It receives, as input, the operands and bits of the instruction for its decoding and outputs the result of the calculation of the corresponding bit operation.

IV. Developing of the Implementation of the Zbb Extension

Hereinafter, the designed module that implements the Zbb extensions will be called BBMU (Basic Bit Manipulation Unit).

First of all, it was necessary to develop the external interface of the BBMU. It performs bit operations and can be thought of as a "black box", where the inputs and outputs are known, but the implementation details are hidden [20].

The BBMU is expected to perform calculations over operands from the register file. Since the processor core is 32-bit, all the data is 32 bits wide. So, the BBMU must have 2 32-bit inputs, as well as a 32-bit output.

Since the BBMU is expected to process different types of bit operations, the module must also be able to distinguish between them, decode, and execute what is expected. To unambiguously establish which instruction is being given, it is enough to receive the bits of the instruction, with which they are encoded as input. They are: 7-bit opcode field, 3-bit funct3 field, 7-bit funct7. In addition, a 12-bit immI immediate field is used to encode some Zbb extension instructions; it must be send to the BBMU input. Given all these fields, the BBMU is able to uniquely identify the bit operation.

TABLE I. Zbb Extension Instructions

Functional purpose	Mnemonics	Instruction description
Logical with negate	andn rd, rs1, rs2	AND with inverted operand
	orn rd, rs1, rs2	OR with inverted operand
	xnor rd, rs1, rs2	Exclusive NOR
Count leading / trailing zero bits	clz rd, rs	Count leading zero bits
	ctz rd, rs	Count trailing zero bits
Count population	cpop rd, rs	Count set bits
Integer minimum / maximum	max rd, rs1, rs2	Maximum
	maxu rd, rs1, rs2	Unsigned maximum
	min rd, rs1, rs2	Minimum
	minu rd, rs1, rs2	Unsigned minimum
Sign- and zero-extension	sext.b rd, rs	Sign-extend byte
	sext.h rd, rs	Sign-extend halfword
	zext.h rd, rs	Zero-extend halfword
Bitwise rotation	rol rd, rs1, rs2	Rotate Left (Register)
	ror rd, rs1, rs2	Rotate right (Register)
	rori rd, rs1, shamt	Rotate right (Immediate)
OR Combine	orc.b rd, rs	Bitwise OR-Combine, byte granule
Byte-reverse	rev8 rd, rs	Byte reverse register

It is necessary to consider the case when the signals, applied to the input of the module, do not correspond to any known instruction [21]. Then the result of the BBMU calculation is unpredictable, and its output cannot be trusted. Especially for this case, another output indicator (valid bit) was added to the module, which serves as a signal that the instruction, received by the module, is correct.

A visual representation of the BBMU module, summarizing all of the above, is shown in Fig. 1. The module contains 2 data inputs – operands din_rs1 and din_rs2 – and 4 inputs for the fields opCode, funct3, funct7, and immI. Also, the module has 2 outputs: data output dout_rd and a valid bit isZbbInstr.

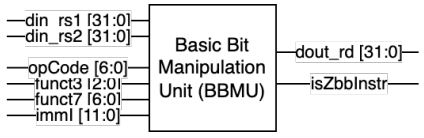

Fig. 1. Representation of the BBMU module as a "black box".

A separate task is to combine the developed BBMU module with the schoolRISCV soft processor core [22], the data path of which is shown in Fig. 2. This should not violate the integrity of the core: it must still be able to process the instructions of the basic architecture.

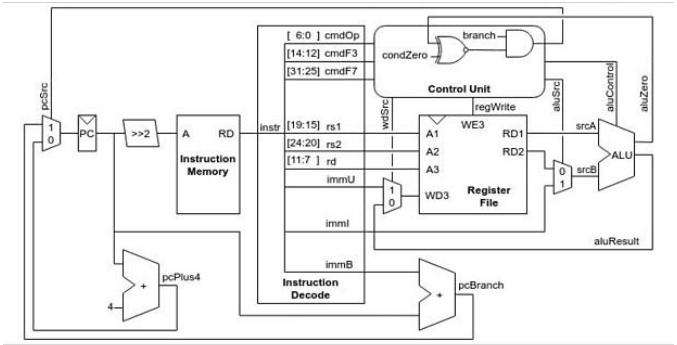

Fig. 2. Diagram of the schoolRISCV core datapath.

The extension module for bit manipulation is very similar to an ALU and even works similarly. It differs only in that it decodes the instruction itself. From this we can conclude that the BBMU should work in parallel with the arithmetic logic unit. Depending on the instruction, the final result of the calculation is selected either from the output of the ALU or from the output of the BBMU. The module is associated with other elements of the data path. The inputs din_rs1 and din_rs2 receive operands read from the register file – this is the data on which calculations are performed. The 4 input signals (opcode, funct3, funct7, immI) are derived from the instruction bits. A 2-input multiplexer was added to the end of the datapath. One of its inputs is the result of the ALU execution and the other is the result of the BBMU. The other output of the bit operation unit, isZbbInstr, is used to control the multiplexer. If the instruction belongs to the Zbb extension, the result of BBMU is taken. Otherwise, the result of the ALU. The output of the multiplexer goes to a register file so that the result of the calculation is stored (or not stored, depending on the instruction). A visual representation of the data path of the schoolRISCV core with the BBMU module is shown in Fig. 3.

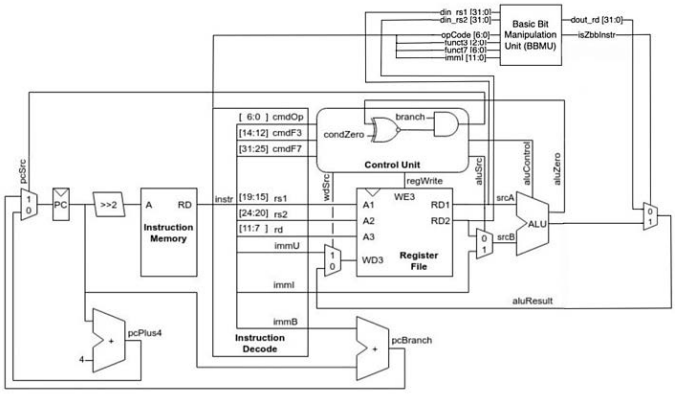

Fig. 3. Datapath diagram with BBMU module.

V. Adding New Instructions Support to the RARS Assembler and Simulator

RARS [23] is a RISC-V assembler and runtime simulator. It is used to translate RISC-V assembler language programs into binary machine code. RARS is open source software, which means that anyone can get, read, and modify its code.

Since extension B for bit operations was recently ratified, instructions from it have not yet been added to RARS. Therefore, we did it ourselves. The RARS software package is developed in the Java high-level programming language, so the modification of the source code is made in it. RARS adds instructions from the Zbb extension for basic bit operations. The extended functionality of RARS allows translating assembler language programs with instructions from the Zbb extension into machine code. The machine code is further required for modeling in ModelSim and downloading the program to the FPGA board. The modified RARS simulator Java code is hosted on GitHub: https://github.com/markoved/rars/tree/zbb.

VI. Verification in ModelSim

Using the ModelSim software, modeling and verification of the module developed in Verilog HDL, integrated into the schoolRISCV soft processor core, were carried out. A test program in assembler language, containing one test case for each implemented instruction, was developed. The program is translated into machine code using the RARS assembler and loaded into the processor's memory. Waveforms in ModelSim allow checking the values in specific registers of a register file at arbitrary clock cycles to check their correctness. A testbench, which automatically checks the operation of the processor in test cases, was developed. It is shown that the developed BBMU module correctly calculates the results of bit manipulation operations.

VII. FPGA Prototyping

The Quartus Prime package includes a programmer for uploading a synthesized project to an FPGA board. Quartus Prime supports many models and families of FPGAs, including the Terasic DE1-SoC development board (with a Cyclone V family chip) [24].

The DE1-SoC board has 6 seven-segment indicators that display the value of the selected register. The LEDR[0] is clocked, and the LEDR[9:1] indicators show bits 8-0 of the selected register. The register number is set by switches SW[4:0]. The processor core is clocked by 50 MHz clock signal generated from the development board. Since this frequency is too high to be observed by the human eye, a mechanism for dividing it is provided. The divider is set by switches SW[8:5]. Clock generation can be enabled or disabled by the switch SW[9]. It is also possible to send single clock signals by pressing the KEY[1] button. The KEY[0] button is used to reset the processor state: to reset the program counter and the register file.

By running on the board the assembler program developed, one can verify that it works correctly by checking the values in the registers for compliance with the expected ones.

VIII. Comparison of Basic and Extended Core Versions

The Quartus Prime package allows estimating the amount of resources used (logic gates, registers, and memory cells). Adding a BBMU module for bit operations to the schoolRISCV core requires a significant amount of additional resources. The analysis shows that the number of used FPGA logic cells increased from 99 to 986 (almost 10 times). At the same time, this is still a small part of the total FPGA resources, which means that such a processor core can be used in lightweight projects and even as a module for multiprocessor systems-on-chip, as, for example, in [25-27]. The number of registers increased by more than 8 times: from 103 to 821. The amount of pins and memory cells did not change (Table II).

To evaluate the benefits of the extended core and bit operations module on the real examples, a set of 10 small benchmark programs in C programming language was developed. Each of them was copied in two versions: for the RV32I base architecture and the RV32I_ZBB extended architecture. Table III shows the properties of the test programs

for the two architectures, such as the size of the program and the number of clock signals required to execute.

The source code of the project is located on GitHub: https://github.com/markoved/schoolRISCV/tree/Zbb.

TABLE II. COMPARISON OF PROCESSOR CORES WITH BASIC AND EXTENDED ARCHITECTURE

Characteristic	Core with basic architecture	Core with extended architecture
Maximum clock frequency, Fmax	104.35 MHz	54.82 MHz
Amount of FPGA logic cells	99	986
Amount of registers	103	821
Amount of memory cells	768	768
Amount of pins	141	141

TABLE III. COMPARISON OF THE PROPERTIES OF TEST PROGRAMS FOR THE TWO ARCHITECTURES

Benchmark program	Program size, lines		Number of clock signals required to execute					
			RV32I			*RV32I_ZBB*		
	RV32I	*RV32I_ZBB*	*min*	*max*	*avg*	*min*	*max*	*avg*
isPowerOfTwo	7	4	3	5	5	4	4	4
resetNthBit	5	4	5	5	5	4	4	4
isEven	3	3	3	3	3	3	3	3
max (conditional)	3	2	2	3	2.5	2	2	2
max (bit magic)	4	2	3	4	3.5	2	2	2
abs	4	3	4	4	4	3	3	3
sign0	4	4	4	4	4	4	4	4
popCount	7	2	7	131	69	2	2	2
minOf3	9	3	5	6	5.5	3	3	3
reverseBits	34	23	34	34	34	23	23	23
Average	8	5			13.55			5

In order to estimate the execution time of each of the programs, it is necessary to divide the average number of cycles required to execute the program by the maximum processor frequency:

$$t = \frac{n}{F_{max}}, \quad (1)$$

where n is the average number of cycles required to execute the program; F_{max} is the maximum clock frequency of the processor.

The analysis showed that the size of the program decreased by an average of 37.5%; there is also a 63.1% decrease in the average number of cycles required to complete the program. The average program execution time on the basic core was 129.9 ns; the average program execution time on the extended core was 91.1 ns; the acceleration was 29.9%.

At the same time, it would seem that the negative result that Fmax decreased by almost 2 times, in fact, means that this will lead to a decrease in dynamic power consumption. On the other hand, due to the fact that hardware costs have increased significantly, this will affect static power consumption due to leakage current. Power consumption could be reduced using various tricks, for example, Clock gating [28–29]. More precise estimates can be made using specialized tools such as Synopsys PrimeTime PX [30], which is planned to be done in the future.

IX. CONCLUSION

As a result, an implementation of the RISC-V architecture with an extended set of instructions for bit manipulation was developed. The development was carried out in the Verilog language, the extension module was integrated into the basic 32-bit single-cycle schoolRISCV soft processor core. The internal structure of the developed module was designed in such a way that all bit operations are performed simultaneously, and the result is taken in accordance with the fields of the command, with which it is encoded. This allows the performance increase by reducing the critical path in the combinational circuit of the module.

To verify the developed module, a RISC-V assembler language program that checks each command of bit manipulation extension was created, and modeling in the ModelSim software was carried out. The correct operation of all machine instructions was shown.

The extended processor core in the form of RTL was synthesized, and its prototyping was carried out on the DE1-SoC FPGA board. The operation of the processor with an arbitrary program can be demonstrated on the development board.

The peculiarity of the work is that support for extension B instructions for bit manipulation was added to the assembler and the RARS runtime simulator using the Java language. This is necessary in order to translate commands from the extension for bit manipulation into machine code.

To estimate the maximum clock frequency of the modified core, Quartus Prime Timing Analyzer was used. The integration of the bit manipulation module into the schoolRISCV soft processor core reduced the maximum processor frequency by 47%: from 104.35 MHz to 54.82 MHz. At the same time, with approximately 10 times more logical elements and 8 times more registers, there was an increase in the resource use. However, the processor still requires a small fraction of FPGA resources – only 3% of the total number of logic elements.

To evaluate the real benefits from the extended processor, we compared the speed of execution of programs from the test set of benchmarks on two versions of the processor. The analysis showed that, despite the lower frequency of the extended processor, the acceleration of program execution on it was 29.9% compared to the core of the basic RISC-V architecture. Another useful result – a reduction in the size of programs by 37.5% and hence, the amount of memory occupied were achieved. Possible changes in circuit power consumption are also discussed.

REFERENCES

[1] K. N. Kumar, K.T.P Kumar, G. V. S. R. Kumar and P. C. Sekhar, “Bitwise operations based encryption and decryption,” International

Journal on Computer Science and Engineering (IJCSE), vol. 3, no. 1, pp. 50–546 2011.

[2] M. Kim and P. Smaragdis, "Bitwise neural networks for efficient single-channel source separation," 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 701–705, 2018. DOI:10.1109/ICASSP.2018.8461824.

[3] S. H. Jeong, M. H. Sunwoo and S. K. Oh, "Bit manipulation accelerator for communication systems digital signal processor," EURASIP Journal on Advances in Signal Processing, pp. 2655–2663, 2005. DOI:10.1155/ASP.2005.2655.

[4] P. S. Babu, S. Sivaraman, D. N. Sarma and T. S. Warrier, "Evaluation of bit manipulation instructions in optimization of size and speed in RISC-V," 2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID), pp. 54-59, 2021. DOI:10.1109/VLSID51830.2021.00014.

[5] B. Koppelmann, P. Adelt, W. Müller and C. Scheytt, "RISC-V extensions for bit manipulation instructions," 2019 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 41-48, 2019. DOI:10.1109/PATMOS.2019.8862170.

[6] V. A. Frolov, V. A. Galaktionov and V. V. Sangarov, "Investigation of the RISC-V," Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS), vol. 32(2), pp. 81–98, 2020. DOI:10.15514/ISPRAS-2020-32(2)-7.

[7] A. Kayid, Y. Khaled and M. Elmahdy, "Performance of CPUs/GPUs for Deep Learning workloads," German University in Cairo, p. 17, 2018. DOI:10.13140/RG.2.2.22603.54563.

[8] S. L. Harris and D. Harris, Digital design and computer architecture. RISC-V Edition. Morgan Kaufmann, 2021, p. 300.

[9] D. K. Dennis et al., "Single cycle RISC-V micro architecture processor and its FPGA prototype," 2017 7th International Symposium on Embedded Computing and System Design (ISED), pp. 1–5, 2017. DOI:10.1109/ISED.2017.8303926.

[10] V. Jain, A. Sharma and E. A. Bezerra, "Implementation and extension of bit manipulation instruction on RISC-V Architecture using FPGA," 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), pp. 167–172, 2020. DOI:10.1109/CSNT48778.2020.9115759.

[11] A. Menon, M. Subadra, C. Rebeiro, N. Gala and K. Veezhinathan, "Shakti-T: A RISC-V processor with light weight security extensions," the Hardware and Architectural Support for Security and Privacy, pp. 1-8, 2017. DOI:10.1145/3092627.3092629.

[12] E. Gür, Z. E. Sataner, Y. H. Durkaya and S. Bayar, "FPGA implementation of 32-bit RISC-V processor with web-based assembler-disassembler," 2018 International Symposium on Fundamentals of Electrical Engineering (ISFEE), pp. 1-4, 2018. DOI:10.1109/ISFEE.2018.8742406.

[13] P. S. Babu, S. Sivaraman, D. N. Sarma and T. S. Warrier, "Evaluation of bit manipulation instructions in optimization of size and speed in RISC-V," 2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID), pp. 54-59, 2021. DOI:10.1109/VLSID51830.2021.00014.

[14] T. Kanamori and K. Kise, "RVCoreP-32IC: An optimized RISC- V soft processor supporting the compressed instructions," 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pp. 38-45, 2021. DOI:10.1109/MCSoC51149.2021.00014.

[15] Y. Hilewitz and R. B. Lee, "Performing advanced bit manipulations efficiently in general-purpose processors," 18th IEEE Symposium on Computer Arithmetic (ARITH '07), pp. 251-260, 2007. DOI:10.1109/ARITH.2007.27.

[16] T. Shimizu et al, "A 32-bit microprocessor with high performance bit-map manipulation instructions," Proceedings 1989 IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 406-409, 1989. DOI:10.1109/ICCD.1989.63397.

[17] S. Payvar, E. Pekkarinen, R. Stahl, D. Mueller-Gritschneder and T. D. Hämäläinen, "Instruction extension of a RISC-V processor modeled with IP-XACT," 2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), pp. 1-5, 2019. DOI:10.1109/NORCHIP.2019.8906975.

[18] The RISC-V Instruction Set Manual, Volume I: UserLevel ISA, Document Version 20191213. RISC-V Foundation, 2019, p. 220.

[19] RISC-V Bit manipulation extension, Working Draft of proposed standard, Rev. 1.0.0. RISC-V Foundations Bitmanip Extension working group, 2021, p. 58.

[20] S. D. Kim, S. H. Jeong, M. H. Sunwoo and K. H. Kim, "Novel bit manipulation unit for communication digital signal processors," 2004 IEEE International Symposium on Circuits and Systems (ISCAS), pp. II-385, 2004. DOI:10.1109/ISCAS.2004.1329289.

[21] A. Raveendran, V. B. Patil, D. Selvakumar and V. Desalphine, "A RISC-V instruction set processor-micro-architecture design and analysis," 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), pp. 1–7, 2016. DOI:10.1109/VLSI-SATA.2016.759304.

[22] schoolRISCV GitHub repository. [Online]. Available: https://github.com/zhelnio/schoolRISCV.

[23] RARS GitHub repository. [Online]. Available: https://github.com/TheThirdOne/rars.

[24] Terasic DE1-SoC User Manual, Rev. 1.2.2d, Terasic Technologies, p. 113, 2014.

[25] A. E. Ryazanova, A. A. Amerikanov and E. V. Lezhnev, "Development of multiprocessor system-on-chip based on soft processor cores schoolMIPS," Journal of Physics: Conference Series, vol. 1163, no. 1, pp. 1–7, 2019. DOI:10.1088/1742-6596/1163/1/012026.

[26] Chethan Kumar H B, Prashant Ravi, Gourav Modi and Nachiket Kapre, "120-core microAptiv MIPS Overlay for the Terasic DE5-NET FPGA board," Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17), Association for Computing Machinery, pp. 141–146, 2017, DOI:10.1145/3020078.3021751.

[27] A. Y. Romanov, A. D. Ivannikov and I. I. Romanova, "Simulation and synthesis of networks-on-chip by using NoCSimp HDL library," IEEE 36th International Conference on Electronics and Nanotechnology, pp. 300–303, 2016. DOI:10.1109/ELNANO.2016.7493072.

[28] N. Koduri and K. Vittal, "Power analysis of clock gating at RTL," Accessed: Jan. 6, 2019. [Online]. Available: https://www.design-reuse.com/articles/23701/power-analysis-clock-gating-rtl.html.

[29] J. Kathuria, M. Ayoubkhan and A. Noor, "A review of clock gating techniques," MIT International Journal of Electronics and Communication Engineering, vol. 1, no. 2, pp. 106–114, 2011.

[30] S. M. Bhagat and S. U. Bhandari, "Design and analysis of 16-bit RISC processor," 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), pp. 1-4, 2018. DOI:10.1109/ICCUBEA.2018.8697859.