Performance-Optimised Design of the RISC-V Five-Stage Pipelined Processor NRP

Hongkui Li[1], Chaoxia Jing[2], Jie Liu[3]*
School of Information Engineering, Huzhou University, Huzhou, China[1, 2, 3]

***Abstract*—The five-stage pipeline processor is a mature and stable processor architecture suitable for many applications in the field of computer hardware. Based on the RISC-V instruction set architecture, the five-stage pipeline processor has advantages in performance, functionality, and power consumption. This paper presents an optimized RV32I five-stage pipeline processor, NRP, and proposes two optimization methods to improve the performance of NRP. These methods include instruction decoding unit optimization and branch prediction optimization. We implemented NRP using Verilog HDL and verified its performance using Vivado and the Xilinx Artya7-35T FPGA board. Experimental data shows that after adopting these methods, the CoreMark score of the five-stage pipeline processor reached 3.11 CoreMark/MHz, representing an 11.07% performance improvement.**

***Keywords*—Architecture; FPGA; RISC-V; RV32I; Verilog HDL; five-stage**

I. Introduction

The Instruction Set Architecture (ISA) is the foundation of computer architecture. Existing ISAs (such as X86, ARM, etc.) have hindered the advancement and proliferation of technology through patent protection [1]. In 2010, the University of California, Berkeley, first released the RISC-V Instruction Set Architecture [2]. RISC-V is an open and free ISA.

In recent years, research on the RISC-V ISA has become a major focus. For example, Alibaba's Xuantie-910 [4, 5], Western Digital's SweRV [3], UC Berkeley's Rocket [6,7], IIT Madras' SHAKTI project [8], ETH Zurich's Pulpino [9-11], the open-source processor mriscv [12], and VexRiscv [13].

This paper presents an optimized five-stage pipeline RV32I scalar processor, NRP (New RISC-V Processor). The main contributions of this paper are as follows:

- To improve performance, we modified and optimized the ID and EX stages of the processor, reducing the negative impact of dependency conflicts on the processor.
- We implemented these optimizations using Verilog HDL and evaluated hardware resource utilization and processor performance. From the evaluation results, we found that this processor outperforms the classic five-stage pipeline processor.

II. Related Works

Dependency conflicts are an important factor affecting the performance of a five-stage pipeline processor. Dependency conflicts refer to the data dependency, control dependency, and structural dependency between instructions, which can lead to instruction hazards in the pipeline, thereby affecting the processor's performance.

In study [14], the authors designed and implemented a Tournament Branch Predictor, which improved the accuracy of branch prediction and enhanced the processor's efficiency. In reference [15], the authors combined the instruction fetch stage with the pre-fetch stage into a two-stage pipeline, resulting in a 17.6% improvement in processor performance. In reference [16], the authors proposed reducing hazards through the use of techniques such as data forwarding and branch prediction, leading to a 7.82% increase in processor performance. In reference [17], the authors optimized the instruction fetch unit, ALU, and data memory, increasing the processor's operating frequency.

The optimization strategies in references [14, 15] resulted in significant performance improvements but increased the complexity and hardware resources of the branch predictor. The optimization strategies in references [16, 17] had lower hardware overhead but led to smaller improvements in processor performance. This paper comprehensively compares these optimization strategies and proposes a new optimization strategy that achieves performance improvements with minimal hardware overhead.

III. The Design and Implementation of the NRP

A. Processor Architectures

The NRP processor adopts a five-stage pipeline design. As shown in Fig. 1, instructions undergo the following five stages during execution: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB) [18]. The design of the ID and EX stages in the NRP processor differs from that of the classic five-stage pipeline processor. The ID stage of the NRP processor consists of both the instruction decode unit and the decode execute unit, whereas the classic five-stage pipeline processor only has the instruction decode unit. The EX stage of the NRP processor consists of the execute unit and the branch prediction auxiliary unit, while the classic five-stage pipeline processor only has the execute unit.

B. Instruction Execution Process

This paper categorizes all instructions in RV32I into special instructions and regular instructions. Branch jump instructions and instructions similar to branch jump instructions in terms of computational operations are defined as special instructions. The computational operations of special instructions, originally executed in the EX stage, are now completed in the ID stage.

*Corresponding Author.

Special instructions include ADD, ADDI, SUB, SLT, SLTU, SLTI, SLTIU, BEQ, BNE, BLT, BGE, BLTU, BGEU, JAL, JALR, LB, LH, LW, LBU, LHU, LWU, SB, SH, SW. The ID and EX stages of the NRP processor differ from those of the classic five-stage pipeline processor, resulting in differences in the execution process of instructions in the ID and EX stages. Fig. 2 illustrates the main execution process of instructions in the ID and EX stages.

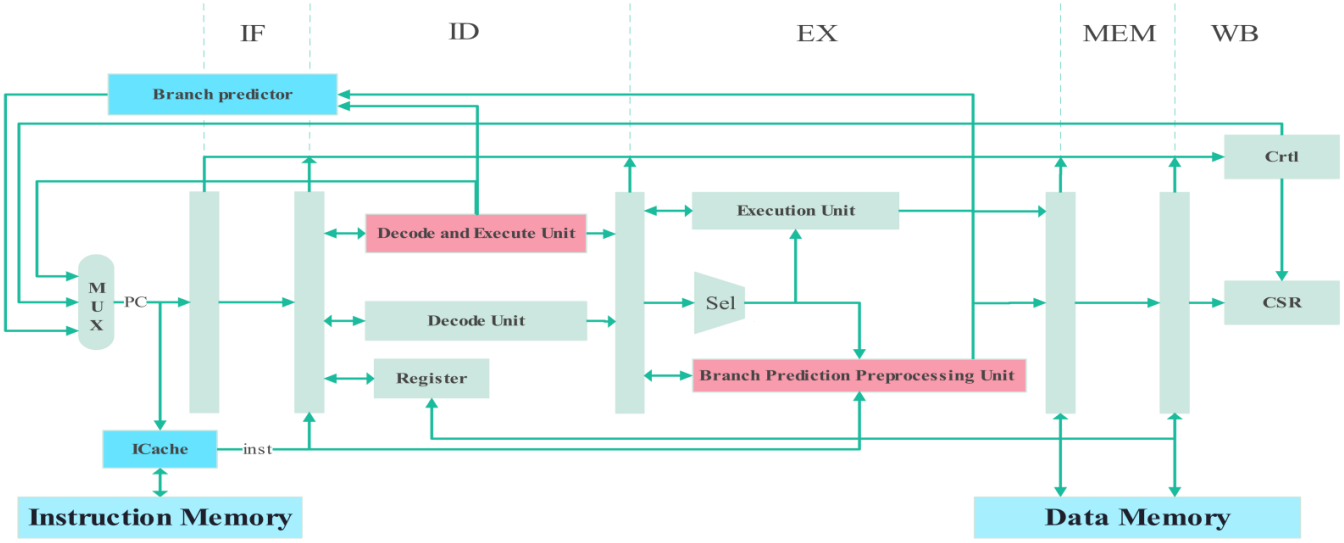

Fig. 1. A block diagram of the five-stage pipelined processor NRP.

The ID stage of the NRP processor consists of the instruction decode unit and the decode execute unit, with each functional unit's decoder responsible for decoding a portion of the instructions. In the ID stage, the execution logic of special instructions involves first decoding the instructions by the decode execute unit and then completing the computational operations required by the instruction opcode within this module. The instruction decode unit is responsible for decoding regular instructions and forwarding the instruction decode information and source operands to the next stage. If an unsolvable data dependency conflict occurs during the execution of a special instruction in the ID stage, the instruction is flagged and then resolved through data forwarding in the EX stage.

The EX stage of the NRP processor consists of the execute unit and the branch prediction auxiliary unit. The execute unit performs operations based on the type of instruction, including regular instructions and flagged special instructions. The branch prediction auxiliary unit is responsible for handling unflagged special instructions and generating branch prediction auxiliary information.

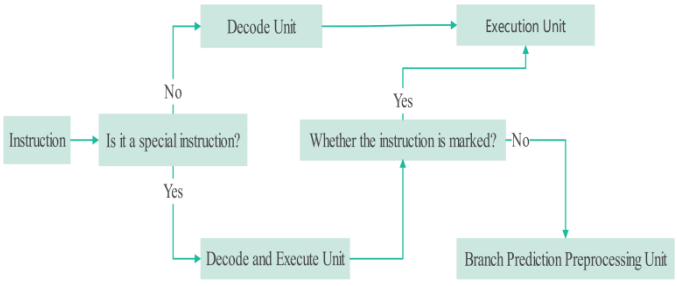

Fig. 2. The execution process of instructions in the ID and EX stages.

IV. The Optimization Design in NRP

Correlation conflicts are a significant factor affecting the performance of a five-stage pipelined processor. These conflicts can lead to pipeline stalls, reducing the processor's performance. The optimization idea proposed in this paper aims to minimize the negative impact of correlation conflicts on processor performance. In this section, we describe the design and implementation of optimization strategies for the NRP processor.

A. Optimization Design of the Decoding Stage

The control dependency conflict in a five-stage pipelined processor refers to the situation where the conditional result of a branch instruction is not yet determined, potentially allowing subsequent instructions to enter the pipeline. If the branch prediction fails, the pipeline needs to be flushed and restarted, causing a stall and impacting processor performance.

The optimization design in the ID stage of the NRP processor aims to reduce the pipeline stall time caused by control dependency conflicts. In a classic RISC-V five-stage pipelined processor, when a branch prediction fails, a stall of two clock cycles is required for pipeline flushing. This paper introduces an additional decode and execute unit in the ID stage of the NRP processor, reducing the stall to just one clock cycle in the event of a branch prediction failure.

The optimization design in the ID stage allows branch instructions to know the branch prediction result and determine if there will be a control dependency conflict. The execution process of special instructions in decode and execute unit is illustrated in Fig. 3. Firstly, the decoder decodes the instruction to obtain instruction information. Then, based on the instruction opcode, it generates a 2-bit enable signal to activate the corresponding arithmetic unit. The arithmetic unit performs

operations on the source operands and communicates using shared data. Finally, the instruction operation result and related information are passed to the EX stage, and the branch prediction result is transmitted to the branch predictor. In the event of a branch prediction failure, the correct PC is passed to the IF stage, and the pipeline pause signal is transmitted to the Ctrl module.

Decode and execute unit consists of a special instruction decoder, an adder, and a comparator. In the implementation process, we virtually divide the full instruction decoder into a special instruction decoder and a regular instruction decoder. When the instruction decoder decodes a special instruction, decode and execute unit is activated. When the instruction decoder decodes a regular instruction, the decode and execute unit does not activate. The primary hardware costs in our optimization design in the ID stage are the adder and the comparator.

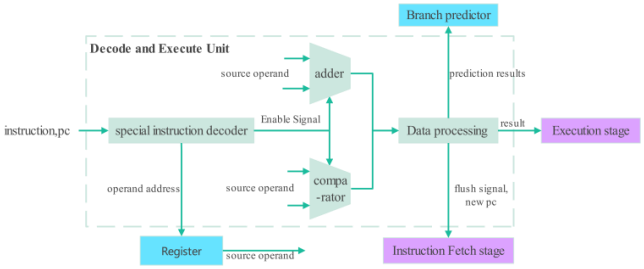

Fig. 3. Decode and execute unit.

B. Branch Predictor Optimization

The branch predictor used in this paper is based on SonicBoom's NLP (Next-Line Predictor), consisting of BHT (Branch History Table), BTB (Branch Target Buffer), and RAS (Return Address Stack). We have optimized the BHT.

Traditional BHT records the state of each branch instruction based on its historical execution results. When a branch instruction is executed for the first time, it defaults to not taken due to the lack of historical execution results. The design proposed in this paper allows obtaining the opcode and the target address of the instruction before its first execution, causing the branch instruction to default to taken upon its first execution. JAL and JALR, as direct jump instructions, always cause a jump upon each execution, which cannot be accommodated by the traditional BHT design.

The workflow of the BHT designed in this paper is as follows: When a branch instruction is first recorded in the BHT, the value of the corresponding two-bit saturating counter table (2BC) is set to 2. If the branch instruction indeed jumps during execution and the jump target address is correct, the value of the two-bit saturating counter table is incremented by 1. If the branch instruction does not jump during execution, then the value of the two-bit saturating counter table is decremented by 1. If the value of the two-bit saturating counter table for the branch instruction is greater than or equal to 2, it is predicted that the instruction will jump.

The branch prediction auxiliary module is crucial for implementing the BHT optimization design, as it allows obtaining the opcode and the target address of the instruction before its execution. The NRP processor classifies instructions into regular and special instructions. Special instructions are decoded and executed in the ID stage, so when a special instruction reaches the EX stage, an idle clock cycle is generated. The branch prediction auxiliary module utilizes this idle clock cycle to perform simple decoding of the instruction and generate data for updating the branch predictor.

The branch prediction auxiliary module consists of a branch instruction decoder and an adder, and its specific workflow is illustrated in Fig. 4. Firstly, the branch instruction decoder in the branch prediction auxiliary module decodes the instruction currently in the cache. If the instruction is a branch instruction, its instruction type and immediate value are obtained after decoding. Then, the PC value and the immediate value of the instruction are sent to the ALU for addition to obtain the jump target address. Finally, the PC value, instruction type, and jump target address of the instruction are sent to the branch predictor.

The primary hardware costs in the branch prediction optimization design are a branch instruction decoder and an adder, with the branch instruction decoder supporting only the decoding of branch instructions.

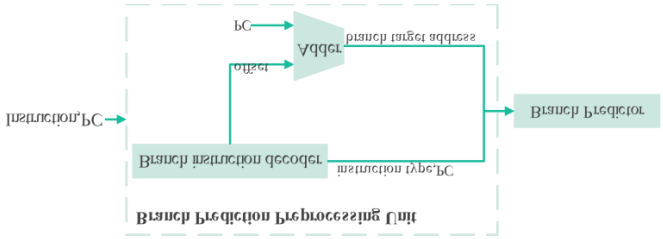

Fig. 4. The branch prediction auxiliary module.

C. Optimization of Dependency Conflict

The Fig. 5 illustrates how a classic five-stage pipelined processor uses data forwarding, pipeline stalling, and branch prediction to resolve various dependency conflicts and their resulting impacts.

In Fig. 5, we can observe the following scenarios. Firstly, the classic five-stage pipelined processor utilizes data forwarding to forward data from the EX stage and MEM stage to the ID stage to resolve non-load instruction-induced data dependency conflicts, and a combination of pipeline stalling and data forwarding is used to resolve load instruction-induced data dependency conflicts. Secondly, the classic five-stage pipelined processor executes branch instructions in the EX stage, and in the event of a branch prediction failure, it requires flushing the pipeline for two clock cycles. Lastly, the unoptimized branch predictor defaults to not taking a branch on the first prediction of a branch instruction, so when the processor executes an immediate jump instruction for the first time, a branch prediction failure and pipeline flush are inevitable.

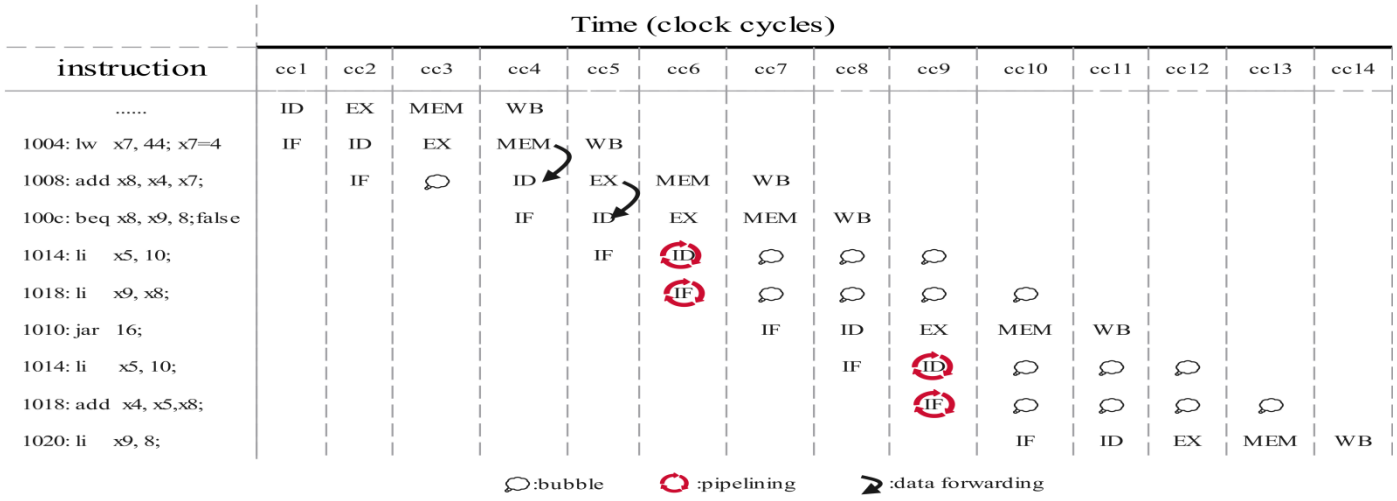

Fig. 5. Methods for handling dependency conflicts before optimization.

The Fig. 6 illustrates how the NRP processor uses data forwarding, pipeline stalling, and branch prediction to resolve various dependency conflicts and their resulting impacts.

In Fig. 6, we can observe the following scenarios. Firstly, in the NRP processor, the use of data forwarding is more extensive, including between ID and EX, ID and MEM, and EX and MEM. Secondly, the NRP processor executes branch instructions in the ID stage to obtain the branch prediction result, so in the event of a branch prediction failure, it requires flushing the pipeline for one clock cycle. Lastly, the NRP processor employs an optimized branch predictor, so when executing an immediate jump instruction for the first time, it correctly takes the jump, avoiding pipeline flushing.

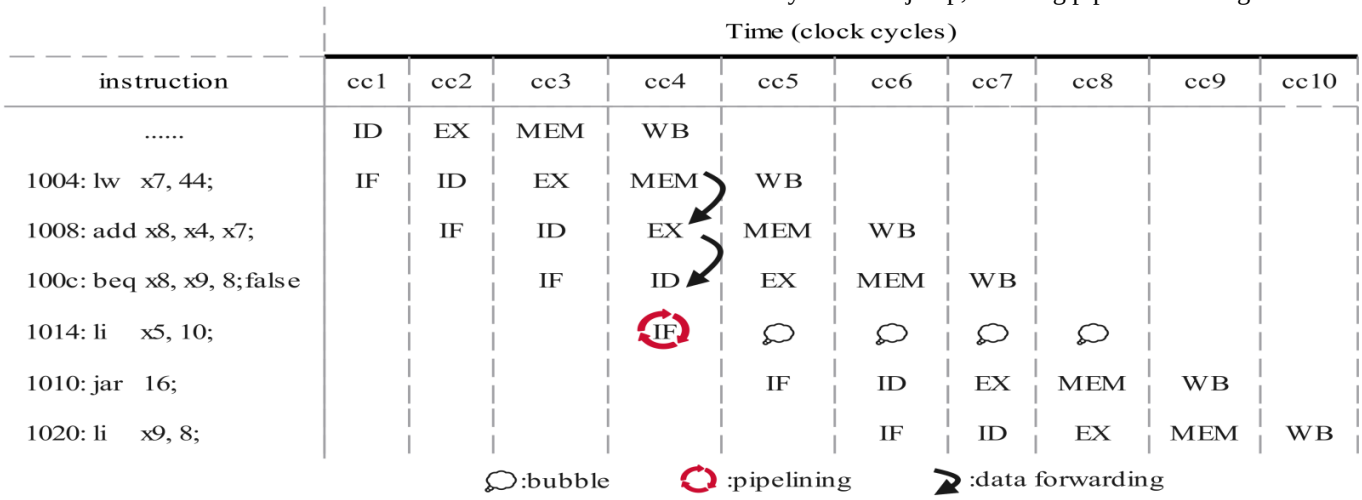

Fig. 6. Methods for handling dependency conflicts after optimization.

V. Experiment and Analysis

A. Functional Test

The COMPLIANCE TEST officially released by RISC-V can test whether the design of a RISC-V core complies with the RISC-V standard [19]. In this paper, joint simulation tests were conducted using Vivado and modsim, and the test results indicate that the NRP complies with the standard of RISC-V core design. Fig. 7 and Fig. 8 show the simulation test results for the ADD instruction and the JAL instruction, respectively.

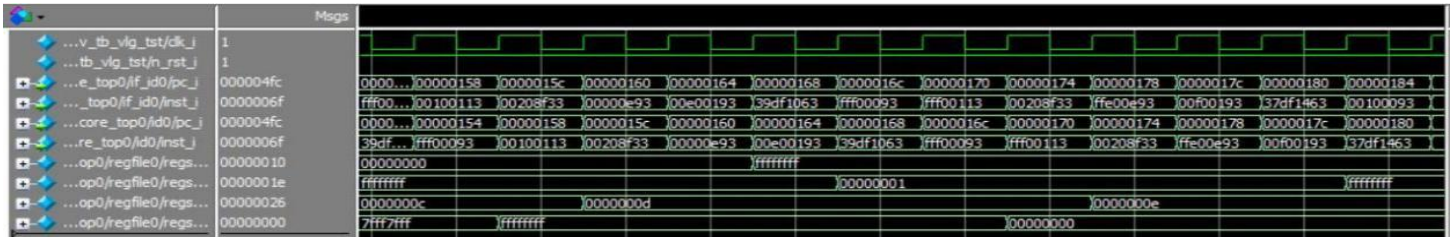

Fig. 7. Validation of add instructions.

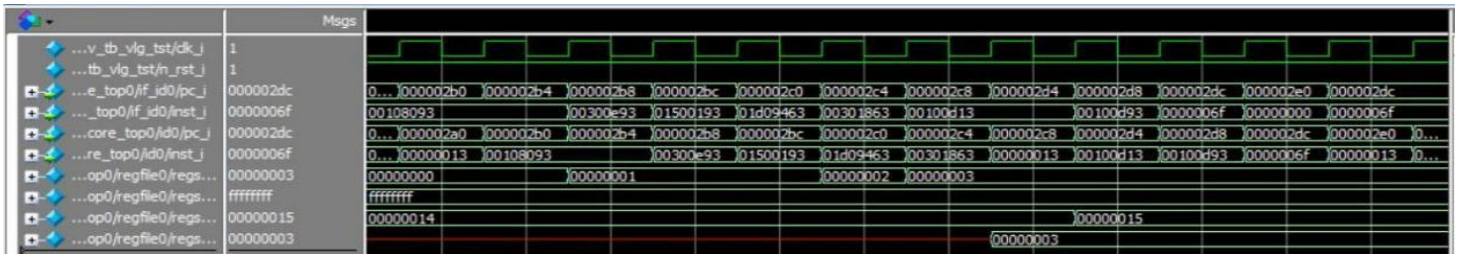

Fig. 8. Validation of JAL instructions.

B. Performance Test

CoreMark is a straightforward yet sophisticated benchmark designed specifically to evaluate the performance of a processor core. In this paper, the CoreMark program and the NRP processor core were ported to Xilinx's ARTYA7-35T development board using Vivado, and the clock function and serial print function were rewritten. The main frequency of the NRP processor core was set to 50MHz for testing, and the results were transmitted to a PC for display via a serial tool. Fig. 9 presents the serial print results, showing that the NRP processor achieved a final CoreMark score of 3.11 CoreMark/MHz.

```
2K performance run parameters for coremark.
CoreMark Size : 666
Total ticks : 321543408
Total time (secs): 67
Iterations/Sec : 155.6
Iterations : 1000
Compiler version : GCC12.2.0
Compiler flags : -O2 -fno-common -funroll-loops -finline-functions --param
max-inline-insns-auto=20 -falign-functions=4 -falign-jumps=4 -falign-loops=4
Memory location : STATIC
seedcrc : 0xe9f5
[0]crclist : 0xe714
[0]crcmatrix : 0x1fd7
[0]crcstate : 0x8e3a
[0]crcfinal : 0xd340
Correct operation validated. See readme.txt for run and reporting rules.

CoreMark : (Iterations/Sec) / Mhz=3.11
```

Fig. 9. CoreMark scores [20].

C. Experimental Analysis

We implemented various versions of the NRP processor in Verilog HDL and evaluated their performance on Xilinx's ARTYA7-35T development board. Based on the optimization level of the NRP processor, we categorized it into three versions. The version without any optimization design is defined as NRP-Original, the version with optimization design only in the decode stage is defined as NRP-OptID, and the version with simultaneous optimization design in the decode stage and branch predictor is defined as NRP-Final.

Fig. 10 displays the CoreMark scores for each version of the NRP processor. After optimizing the design of the ID stage and the branch predictor, the performance of the NRP processor improved by 11.07%.

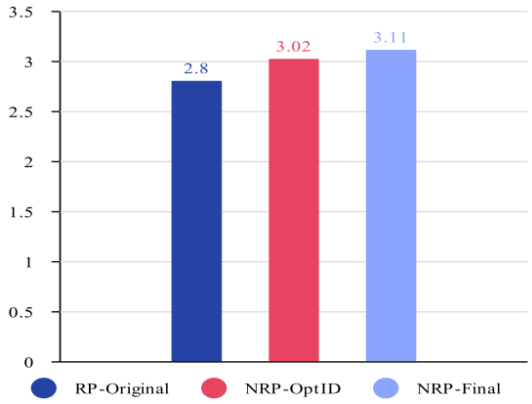

Fig. 10. Performance test results of different versions of NRP.

Fig. 11 presents the CoreMark test results for other open-source processors, showing that the performance of the NRP processor is significantly better than that of other processors [21]-[27].

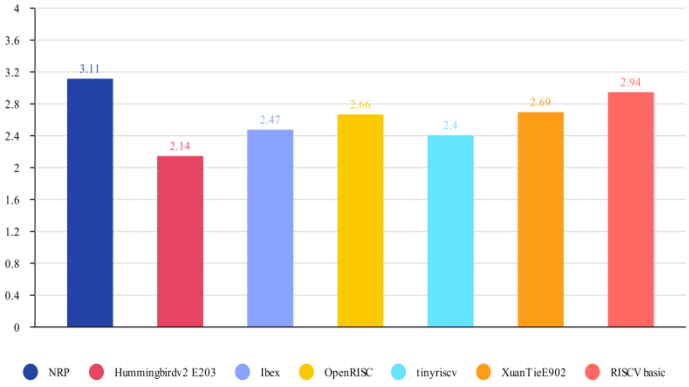

Fig. 11. Performance comparison of different open-source processors.

VI. Conclusion

We have proposed a five-stage pipelined processor based on RISC-V architecture. In this processor, we have employed instruction decoding unit optimization and branch prediction optimization as effective methods to improve operating frequency. We implemented the proposed processor in Verilog using Vivado and conducted tests and evaluations on the processor's performance and hardware resource consumption. The CoreMark test results for the NRP processor after adopting optimization strategies show a score of 3.11 CoreMark/MHz, representing an 11.07% improvement over the non-optimized design.

This research improves the performance of the five-stage pipeline processor based on RISC-V, which can improve the application range of the five-stage pipeline processor and promote the development of the community ecology of RISC-V instruction set architecture. In the future work, we will extend the design of this paper to the five-stage pipeline design of out-of-order execution, and reduce the impact of correlation conflicts on processor performance in out-of-order execution.

REFERENCES

[1] Liu, C, et al. "A Review of Research on RISC-V Instruction Set Architecture." Journal of Software, vol. 32,no.12,pp.3992-4024,2021,10.13328/j.cnki.jos.006490.

[2] A. Waterman, Y. Lee, R. Avizienis, D. A. Patterson, and K. Asanović, "The RISC-V instruction set manual volume II: Privileged architecture version 1.9.1," EECS Department, University of California, Berkeley, UCB/EECS-2016-161,2016.[Online].Available:http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-161.html Accessed on: Mar. 20, 2023

[3] T. Marena, "RISC-V: high performance embedded SweRVTM core microarchitecture, performance and CHIPS Alliance," 2019. [Online].Available: https://riscv.org/wp-content/uploads/2019/04/RISC-V_SweRV_Roadshow-.pdf

[4] C. Chen et al., "Xuantie-910: A Commercial Multi-Core 12-Stage Pipeline Out-of-Order 64-bit High Performance RISC-V Processor with Vector Extension : Industrial Product," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 2020, pp. 52-64, doi : 10.1109/ISCA45697.2020.00016.

[5] Z. Zhou et al., "Cache Design Effect on Microarchitecture Security: A Contrast between Xuantie-910 and BOOM," 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Wuhan, China, 2022, pp. 1199-1204, doi: 10.1109/TrustCom56396.2022.00166.

[6] B. Zimmer et al., "A RISC-V Vector Processor With Simultaneous-Switching Switched-Capacitor DC-DC Converters in 28 nm FDSOI," in IEEE Journal of Solid-State Circuits, vol. 51, no. 4, pp. 930-942, April 2016, doi: 10.1109/JSSC.2016.2519386.

[7] Y. Lee et al., "A 45nm 1.3GHz 16.7 double-precision GFLOPS/WRISC-V processor with vector accelerators," ESSCIRC 2014 - 40th European Solid State Circuits Conference (ESSCIRC), Venice Lido, Italy, 2014, pp. 199-202, doi: 10.1109/ ESSCIRC.2014.6942056.

[8] N. Gala, A. Menon, R. Bodduna, G. S. Madhusudan and V. Kamakoti, "SHAKTI Processors: An Open-Source Hardware Initiative," 2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID), Kolkata, India, 2016, pp. 7-8, doi: 10.1109/VLSID.2016.130.

[9] M. Gautschi, M. Schaffner, F. K. Gürkaynak and L. Benini, "An Extended Shared Logarithmic Unit for Nonlinear Function Kernel Acceleration in a 65-nm CMOS Multicore Cluster," in IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 98-112, Jan. 2017.[Online].Available : http://ieeexplore.ieee.org/document/7756672/

[10] F. Conti et al., "An IoT endpoint system-on-chip for secure andenergy-efficient near-sensor analytics," IEEE Trans. Circuits Syst. I, Reg.Papers, vol. 64, no. 9, pp. 2481 - 2494, Sep. 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7927716/

[11] M. Gautschi et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst., vol. 25, no. 10, pp. 2700-2713, Oct. 2017. [Online].Available: http://ieeexplore.ieee.org/document/7864441/

[12] Available, "MRISCV," GitHub, Mar. 23, 2023. [Online].Available :https://github.com/onchipuis/mriscv Accessed on: Mar. 20, 2023

[13] VEXRISCV. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[14] A. Choudhury, S. V. Siddamal and J. Mallidue, "An optimized RISC-V processor with five stage pipelining using Tournament Branch Predictor for efficient performance," 2022 International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Shivamogga, India, 2022, pp. 57-60, doi: 10.1109/DISCOVER55800.2022.9974891.

[15] A. Tiwari, P. Guha, G. Trivedi, N. Gupta, N. Jayaraj and J. Pidanic, "IndiRA: Design and Implementation of a Pipelined RISC-V Processor," 2023 33rd International Conference Radioelektronika (RADIOELEKTRONIKA), Pardubice, Czech Republic, 2023, pp. 1-6, doi: 10.1109/RADIOELEKTRONIKA57919.2023.10109058.

[16] I. Thanga Dharsni, K. S. Pande and M. K. Panda, "Optimized Hazard Free Pipelined Architecture Block for RV32I RISC-V Processor," 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC), Trichy, India, 2022, pp. 739-746, doi: 10.1109/ICOSEC54921.2022.9952122.

[17] Hiromu Miyazaki, Takuto Kanamori, Ashraful Islam and Kenji. Kise, "RVCoreP: An optimized RISC-V soft processor of five-stage pipelining", Special Section on Parallel Distributed and Reconfigurable Computing and Networking, 2020.

[18] S. S. Khairullah, "Realization of a 16-bit MIPS RISC pipeline processor," 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 2022, pp. 1-6, doi: 10.1109/HORA55278.2022.9799944.

[19] LowRISC, " RISC-V Compliance Task Group," GitHub. https://github.com/lowRISC/riscv-compliance, Accessed on: May 30, 2023.

[20] Coremark Scores,[Online].Available:https://www.eem bc.org/coremark/scores.php,Accessed on: Mar. 25, 2023

[21] Nuclei System Technology, "Hummingbirdv2 E203 Core and SoC," GitHub, [Online]. Available:https://github. com/riscv-mcu/e203_hbirdv2,Accessed on: Mar. 25, 2023

[22] lowRISC, " Ibex RISC-V Core," GitHub, [Online]. Available: https://github.com/lowRISC/ibex, Accessed on: Mar. 25, 2023.

[23] N. Dao, A. Attwood, B. Healy and D. Koch, "FlexBex: A RISC-V with a Reconfigurable Instruction Extension," 2020 International Conference on Field-Programmable Technology (ICFPT), Maui, HI, USA, 2020, pp. 190-195, doi: 10.1109/ICFPT51103.2020.00034.

[24] liangkangnan, "tinyriscv," GitHub, [Online]. Available:https://github.com/liangkangnan/tinyriscv, Accessed on: Mar. 25, 2023.

[25] M. Gautschi et al., "Tailoring instruction-set extensions for an ultra-low power tightly-coupled cluster of OpenRISCcores," in Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC), Oct. 2015, pp. 25-30.

[26] T-Head_Communications, " XuanTieE902," GitHub, [Online]. Available:https://github.com/T-head-semi /opene 902, Accessed on: Mar. 25, 2023

[27] M. Gautschi et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," IEEE Trans. Very Large ScaleIntegr. (VLSI) Syst., vol. 25, no. 10, pp.2700 - 2713,Oct. 2017.[Online]. Available:http://ieeexplore.ieee.org/document/7864441/.