Mr.N.B.Jilani, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Low power RISC-V Embedded processor design using HDL

[1]Mrs.N.Swarupa Rani,[2]Mr.N.B.Jilani, [3] Ravuri Venkata Koteswararao,[4]Shaik Irfan,[5]Kancharla Chandu,[6]Kuricheti Pavan Sathwik

[1]Assistant Professor,Department of ECE,KITS-Markapur .
[2]Assosiate professor,Department of ECE,KITS-Markapur
[3,4,5,6]Students,Department of ECE,KITS-Markapur

Abstract— We are entering a new age, and with it comes a surge in interest in generative AI and machine learning throughout the world. As computing talents start to reveal their true colors, there is a pressing demand for efficient and fast microprocessors. The RISC design approach, which stresses a smaller and simpler set of instructions, each executed in a constant amount of time, is widely used by current processors to achieve improved efficiency. A five-stage pipelined general-purpose microprocessor architecture with a clock period of 0.59 nanoseconds has been the target of this study, which translates to optimization approaches being implemented in Verilog code. In addition, by switching to True Single- Phase Clocking (TSPC) flip-flops, we may cut power usage by 20% and cut the number of transistors used by half. The use of Verilog HDL on ModelSim and Xilinx to achieve an n-stage pipelined architecture for a general-purpose microprocessor is shown in this work as an example of RISC-V embedded processor design.

Keywords: RISC, CPU, Verilog HDL I.

I. INTRODUCTION

A design philosophy based on simpler instructions is implemented by central processing units (CPUs) using the Reduced Instruction Set Computer (RISC) architecture. These instructions improve computing efficiency by performing fewer operations while executing them more swiftly. Approximately 30 billion smart gadgets will be linked to 62 billion RISC-V processors in the next years, according to recent predictions. Keeping and improving these processors to match increasing performance and energy efficiency needs is of the utmost significance. Prof. David Patterson first proposed the open Instruction Set Architecture (ISA) RISC-V in 1980 [1]. Cognizant of both sequential and combinational logic circuits, this design incorporates both. Besides the RISC-V International defined instruction set extensions, a lot of work has gone into tailoring instruction sets to particular uses. For example, in order to make wireless signal processing more energy efficient for IoT applications, an addition to the Digital Signal Processing (DSP) instruction set has been suggested. In addition, deep learning algorithms used in AI applications have specific instruction set extensions that speed them up. The modification of RISC-V instruction sets for use in HPC, graphics processing, and communication applications is another area of active investigation. systems, and operating systems that support multiple threads and tasks, parallel languages, and compilers that can parallelize and vectorize are all necessities for high-performance parallel computing. Because of its excellent performance per watt, the RISC architecture is ideal for these needs; this is especially true for battery-operated devices. The RISC instruction set's fixed-length nature and the Datapath and Control Unit's smooth implementation are the main reasons for its broad use in pipelining procedures. These elements comprise the backbone of digital systems and are essential to the design of any computer system. An effective architecture plan is the result of a series of iterations The functioning of the processor is thoroughly examined using waveform simulations, assessments of register files, and

optimizations of Datapath, as seen in the tables and figures that are provided. The processor's efficacy in performing core operations is validated by experimental results acquired via Xilinx ModelSim and FPGA synthesis. This highlights its suitability in real world embedded and high-performance computing systems. Insights into the design of scalable, power-efficient RISC-V microprocessors are provided by this work via empirical verification and careful architectural refinement. building blocks that comprise digital systems. improvement, making sure the system reaches the targeted efficiency and performance targets. Efficient architectural design is the result of a process of iteration.

II. ARCHITECTURE AND DESIGN

The central processing unit (CPU) uses a particular instruction set architecture (ISA) to implement the computer's machine language. The following table displays the ISA's built-in data types. The most complicated jobs may be readily accomplished by digital logic, which is known as binary logic, as it only employs binary data, i.e., 0 or 1, as true or false statements. As shown in the flow diagram below, iterative procedures are carried out by a computer program performing the fetch-decode-execute (FDX) cycle on a loop. [2] in The operation to run, parameters to employ, and place to store the result are all determined by decode. The action is carried out by the execute command, which also decides the next instruction to retrieve from the queue.

TABLE I. Data Type SIZE OF DIFFERENT DATA-TYPES IN ISA

Data Type	Size (Bytes)	Range
int	4	-231 to 231
unsigned int	4	0 to 232
long	4	-231 to 231
unsigned long	4	0 to 232
signed char	1	-27 to 27
unsigned char	1	0 to 28
float	4	-2127 to 2128
double	8	-21024 to 21024

In-depth knowledge of the following units is required for designing a RISC-V processor utilizing Verilog as the HDL language:

In an Arithmetic Logic Unit (ALU), two 32-bit operands provide the basis for binary arithmetic and logic operations; a control signal serves as an enable or select signal to carry out a particular function; and a 64-bit signal constitutes the condition signal. Its capabilities in these areas are evident from its name: addition, subtraction, multiplication, and division. [2] in For these tasks, businesses commonly turn to complementary metal-oxide-semiconductor (CMOS) technology, which is widely employed in the design of VLSI (Very large-scale integration) systems. Among the many types of logic gates, the most common ones are primary ones like INV, NAND, and NOR, and complex ones like AOI and OAI.

The following are examples of additional operations: comparison, shift, rotate, and result handling, the latter of which usually involves storing the result in the CPU's register or sending it to other components for further processing. If you want to know how the various CPU registers work, we've given you the machine level language.[3] Some basic programs have been implemented on the GNU simulator, including operations like adding and dividing numbers, as well as move and storage operations. These operations are carried out under specific registers. For example, if you look at the numbers h and d, which have hex values of 4f10h and 4f15h, respectively, and use the mvi command to move, the result will be stored at data addresses (2000h, 2001h, and 2002h), as requested in the machine cycle code. Similarly, for the division operation, we have stored two binary numbers, 25h and 07h, with values stored at addresses after 3000h needed by the user.

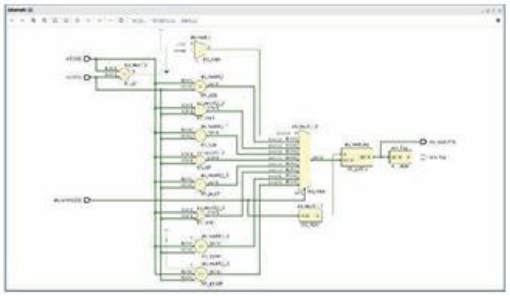

Fig. 1. ALU Schematic Design (Nexys4 DDR technology)

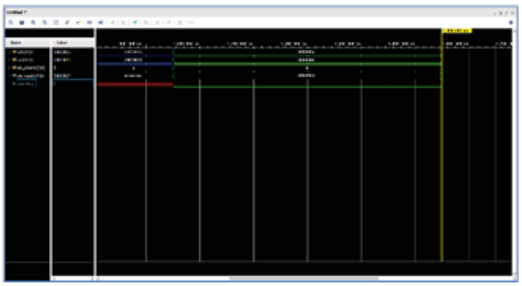

Fig. 2. ALU Waveforms on Xilinx FPGA Suite For the Arithmetic Logic Unit (ALU) design we have made use of the select lines shown below in Table-II.

TABLE II. ALU CONTROL LINES AND FUNCTIONS PERFORMED.

ALU Control Lines	Function (Bitwise)
0000	Add (A, B)
0001	AND
0010	Subtract (A, B)
0100	OR
1000	Set on less than
0011	Multiply
0101	XOR
0110	Shift Left Logical
0111	Shift Right Logical

An important part of the FDX cycle is the Datapath Unit (DU), which follows instructions from the ISA to copy data from one place to another, without changing the original data, using a process known as destination storage. This group includes the MVI, LOAD, and STORE commands. Within the central processing unit (CPU), there are a number of registers that serve as temporary storage for data while processing is on.

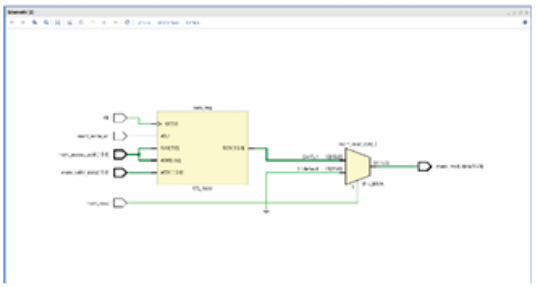

Fig. 3. Datapath Unit Design (Nexys4 DDR technology)

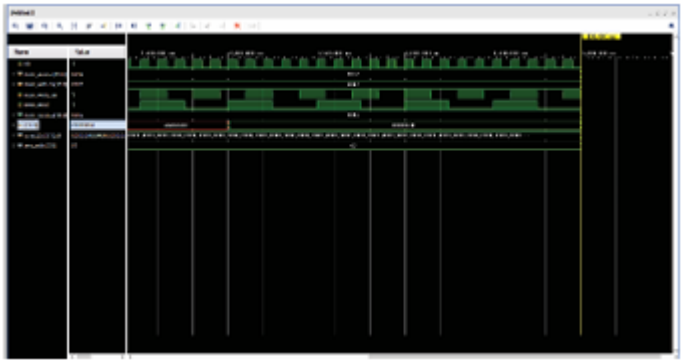

Fig. 4. DPU Waveforms on Xilinx FPGA Suite

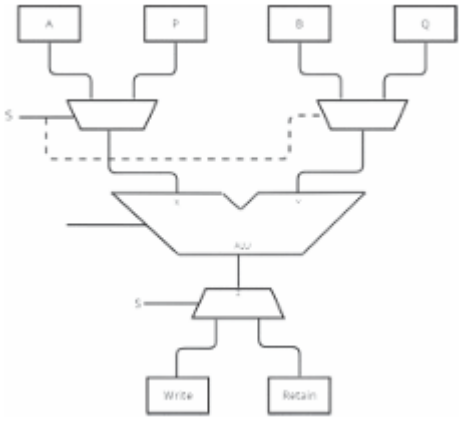

Fig. 5. DPU and ALU Flow Diagram

Coordinating the functioning of the complete datapath unit (DPU) and generating control signals that allow us to control or choose from storage computational routing components is the control unit's (CU) job [3]. If you want to see how DPU and CU work, you may think of them as strings that a puppet master uses to manipulate a particular puppet in a puppet show. With the use of sequential logic architecture and components

like multiplexers, flip-flops, encoders, etc., we may do a number of logical operations with the help of finite state machines (FSMs).

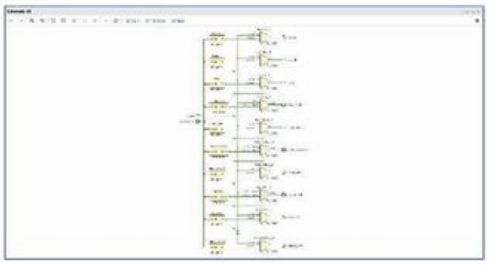

Fig. 6. Control Unit Design (Nexys4 DDR technology)

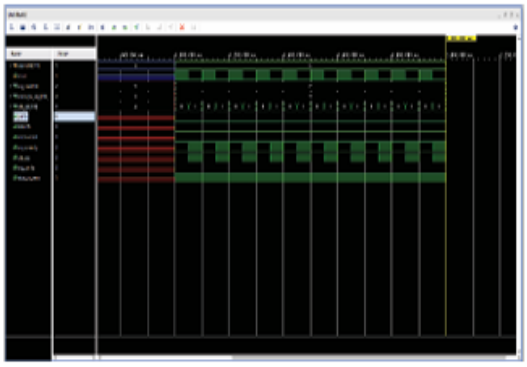

Fig. 7. CU Waveforms on Xilinx FPGA Suite

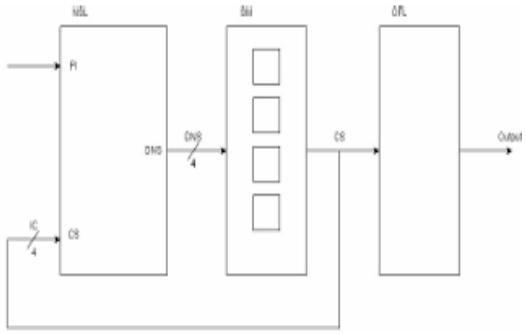

Fig. 8. Fig 8: Control Unit Block Diagram

An essential part of the CPU, the Instruction Fetch Unit (IFU) retrieves relevant data from the memory subsystem according to the program counter (PC) value; it also works with the Memory Management Unit (MMU) to resolve errors that occur when retrieving instructions and may employ buffer insertion techniques to boost performance and decrease delays. A hierarchical memory block is related with the IFU schematic design, which means that the Instruction Fetch Unit is integrated with the Instruction Memory block to perform activities such as the IDX cycle and process register data for the ALU block.

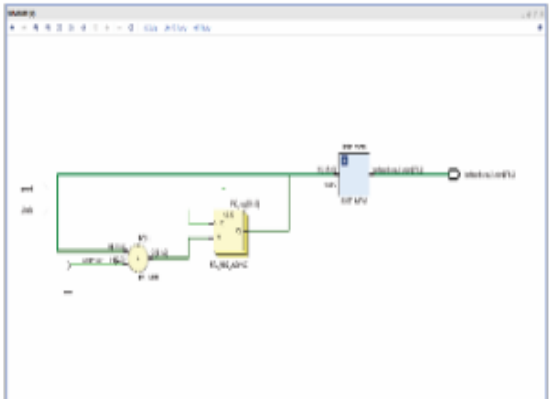

Fig. 9. Instruction Fetch Unit Design (Nexys4 DDR technology)

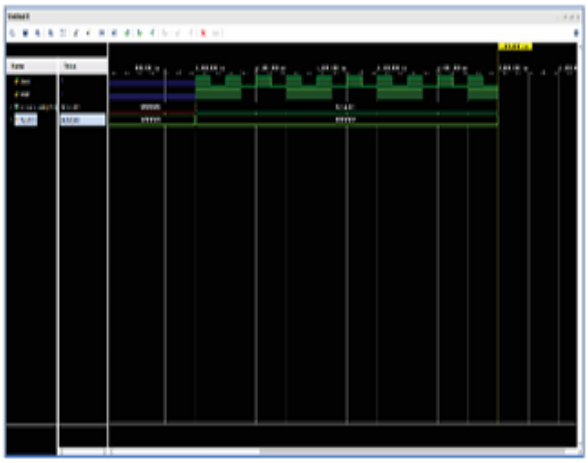

Fig. 10. IFU Waveforms on Xilinx FPGA Suite

Memories: Typically, memory circuits are categorized based on the data storage and access types. There are several kinds of memories, such as read-only memory (ROM), static random access memory (SRAM), dynamic random access memory (DRAM), and so on. Using decoders for row and column address bits is a common way to organize memory arrays. The structure is made up of 2M columns representing bits lines and 2N rows representing word lines. Therefore, there

would be a total of 2N * 2M memory cells in the array.

Fig. 11. Instruction Memory Design (Nexys4 DDR technology)

Fig. 12. IM Waveforms on Xilinx FPGA Suite

Among the many steps in a processor's design are the fetch and decode stages, where the former retrieves instructions from memory and the latter prepares operands for execution [5]. There is a memory stage that retrieves data from memory when the instructions call for it, and an execution stage where the instructions themselves are executed. Improving speed is achieved by the use of techniques like speculative execution, branch prediction, out-of-order execution, and instruction-level parallelism (ILP). The goal of these enhancements is to reduce execution delay and increase instruction throughput.

Fig. 13. 5-Stage Pipelined RISC-V CPU Architecture and Register_File Value Decode.

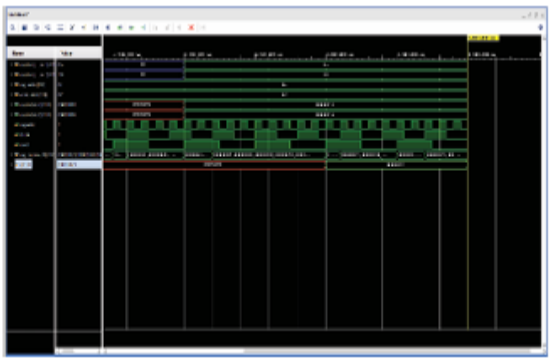

Fig. 14. Register File Waveform values from RISC-V Embedded Processor

III. ANALYSIS AND PERFORMANCE CALCULATIONS

Based on what we learned about RISC architecture, we need a different principle to handle all the different tasks in a single cycle. One possible solution is the Complex Instruction Set Architecture (CISC), which is like a calculator in that it can store previous answers, perform calculations, and display the results. The complexity of the idea lies in the fact that a single instruction can handle loading, evaluating, and storing operations.[6-7] A single cycle may not be sufficient to carry out an action using CISC architecture since instructions are bigger than one word. Both methods, however, aim to boost CPU efficiency. Reduced instruction cycle count (RISC) technology is a trade-off for increased program complexity.

$$CPU\ Time = \frac{Seconds}{Program} = \frac{\text{Instructions} \times \text{Cycles} \times \text{Seconds}}{\text{Program} \times \text{Instructions} \times \text{Cycles}} \quad (1)$$

The CISC method increases the amount of cycles needed to execute each instruction, but also aims to reduce the total number of instructions in a program. Throughput, or the total quantity of work accomplished in a given period, and reaction time, or the time between the start and completion of a job, are two common metrics

used to measure the efficiency of central processing units. It is possible to compare the performance of two processors that have varied execution times and how it impacts a specific machine.

The power and area of the manufactured chip, as well as the CPU time determined using the provided formula, determine the CPU performance that we want to supply to the user. [8] In terms of performance, let's pretend that X's performance is inversely proportional to its execution or reaction time and Y's performance is inversely proportional to its execution or response time. With this assumption, we may say :

$$Performance(X) = \frac{1}{Execution\ time\ (X)}$$

$$Performance(Y) = \frac{1}{Execution\ time\ (Y)}$$

But if the performance of X is greater than Y. Thus,

$$Performance\ (X)\ >\ Performance\ (Y)$$

$$\frac{1}{Execution\ time\ (X)} > \frac{1}{Execution\ time\ (Y)}$$

$$(Execution\ time\ (X))\ <\ (Execution\ time\ (Y))$$

By adding a constant, "n," to the performance ratio, we may express X as being "n" times faster. The formula may be used to determine the time it takes for an operation to run:

$$Execution\ time\ =\ Clocks\ \times\ Clock\ period \quad ($$

Depending on the instruction set and compiler, we may also determine the number of clock cycles consumed by a program's central processing unit (CPU). This information is often used for hardware/software interfaces. In equation (6), CPI stands for Common Instruction Clock Cycles. An old-fashioned formula for CPU performance may be constructed using the above formulae.

$$Execution\ time\ (ET)\ =\ Instruction\ Count\ (IC)$$
$$Clocks\ per\ instruction\ (CP)\ \times\ Clock\ period$$

$$CPU\ Clocks = \sum_{i=1}^{n}(CPI^i \times C^i)$$

Million instructions per second (MIPS) is a popular statistic to use instead of time when doing computations. For an application,

$$MIPS = \frac{Instruction\ count}{Execution\ time \times 10^6} = \frac{Clock\ rate}{CPI \times 10^6} \quad (9)$$

$$Execution\ time = \frac{Instruction\ count}{MIPS \times 10^6} \quad (10)$$

Since MIPS does not account for the instruction count—that is, how many machine instructions does a high-level language program translate into—it cannot be used to compare CPUs with various instruction sets. Even though a RISC CPU's MIPS rate is four times that of a CISC processor, the performance gap may be as little as three times.

However, MIPS does not take instruction count into account, therefore CPUs with quick but simple instructions might have a low CPI and a high clock rate and a high MIPS rating without necessarily having proportionately better performance.

IV. SIMULATION RESULTS AND PLOTS

This research paper details the building of a 32-bit RISC-V ISA based CPU utilizing Xilinx FPGA Suite software, and it covers the architectural and performance computation models of designing the Central Processing Unit. To run, decode, and retrieve data, the unit's structure includes sub-units that use the modules and UVM testbenches. You may test and debug these modules over a certain time range (in picoseconds) using the provided schematics and waveforms [9]. In the final implementation, all the components that were constructed independently are combined to form a working processor. The waveforms we generated using Modelsim and the Xilinx (.xcd) constraints file show the values of the read/write data for the registers, as well as information on the uniform clock and reset cycles. For different amounts of clock cycles and unsigned decimal and hexadecimal values, we have reproduced the waveforms. We have also tried to show how the logic operation is executed, which includes

changing values by storing and loading bits of data into registers. Built within this 32-bit RISC-V processor are arithmetic logic units, memory and write-back function registers, an array of registers for instructions fetch and decode cycles, a 4-bit adder, and a program counter [10]. In order to implement a synchronous sequential operation cycle with other logical units, we used a 32-bit SRAM that was populated with instructions from the instruction set. This cycle was created using repetitions of "for" loops and conditionals (if-else).

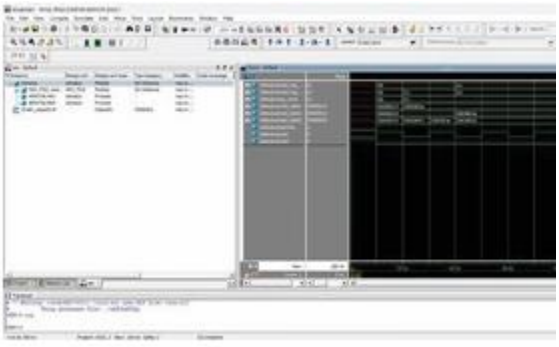

Fig. 15. Waveforms showing RISC-V processor performance (10)

Algorithm for Finite State Machine Implementation of a 5-stage pipelined RISC-V Processor on Intel Mentor ModelSim within 100 to 200 clock cycles (Hexadecimal Number System)

Alg.1. Improved PAD Product Pseudo Code

```
# Front-End: Instruction Fetch and Decode
1. file-In = open ("ref_file_data.vec", "w")
2. file-Out_CPU = open ("Cpu.vec", "r")
3. file-Out_Mem = open ("Mem.vec", "r")
4. for (i = 0; i < n; i++)                    # Initial Stage
   the pipelined processor
                mem_rd = 0 #
                1'b0 mem_wr =
                0 # 1'b0 reg_wr
                = 0    # 1'b0
5. if (ac = = 0):
     if (pc_en = = 0)
   elif (pc_en = = 1)
     elif (jmp = = 1)
          elif (pc_en = = 1)
            if (jmp = = 0)
              elif (reg_wr = = 0)
                elif (mem_rd = = 0)
                                elif (mem_wr = = 0)
                              elif (A_L_instm = = 4'b1001)
                                Execute -> HLT
6. elif (reg_wr = = 1):
          Execute -> MVI
7. elif (mem_rd = = 1):
          Execute -> LOAD
8. elif (mem_rd = = 1):
                          if (reg_wr = = 1)
                        elif (mem_wr = = 1):
                  Execute -> STORE
                          elif (reg_wr = = 1):
                        Execute -> ALU_Instructions
                        elif (mem_wr = = 1):
                          Execute -> STORE
                        elif (mem_wr = = 1):
                          Execute -> LOAD
                        elif (pc_en = = 0):
                          Execute -> HLT
              elif (mem_rd = = 1):
                if (pc_en = = 1)
                    elif (pc_en = = 0)
                      elif (jmp = = 1)
                          elif (pc_en = = 1):
                            if (pc_en = = 0)
                        Execute -> HLT
                                    else:
                                  if (jmp = = 1)
                                  Execute -> MVI
                                    else
                              return 0    # Return to the 1st stage of
                                  processor
9. fileOut.close ()
```

The five-step procedure that goes into making a standard hardware processor is the basis of our design. The five steps comprise: Instruction Fetch (IF) Cycle is the first, second, third, and fifth process. Cycle for Instruction Fetch and Decode (ID). Lifespan of Execution (EX) Write Back (WB) Phase of Memory Access (MEM) In our Verilog code for the system design, we defined the positive clock edge triggers, and logic units like multiplexers, adders, and program counters (PC) help with the instruction fetch and decode cycles [11, 12].

V. CONCLUSION

This article details the computational and methodological steps used to develop a central processing unit (CPU) that adheres to the ISA standards for RISC-V processors, independent of the embedded processor's size. A different approach might be to use FPGA arrays (based on Basys3 technology) instead of the nexys4 DDR technology board, which would allow for more efficient and quicker calculations despite its lower speed grade (-2). Nexys4 DDR was selected mainly because of its cost-effectiveness in relation to its speed grade and area covered. There is a lot of wiggle space in the design hardware for future enhancements, such the ability to generate variants with an n-stage pipelining process and support for other instruction sets like RV64IM. For the most part, we use 18-T TSPC Flip Flop designs to implement CPU designs (20-T with Reset Pin) because they are efficient, use little power, and have little delay. However, we ran into some problems with functionality after replacing them, so we stuck with standard DFFs for the register files and adopted TSPC FFs for the stage registers (a total of 94 FFs). The system receives the values to flush and two instances of conflicts between the ID stage and the HDU (Hazard detection unit) from the late and early branch models in the exception handling procedure.

REFERENCES

1. 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), Bengaluru, India, SATA.2016.7593047. 2016, pp. 1-7, doi: 10.1109/VLSI
2. K. Stangherlin and M. Sachdev, "Design and Implementation of a Secure RISC-V Microprocessor," in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 30, no. 11, pp. 1705 1715, Nov. 2022, doi: 10.1109/TVLSI.2022.3203307.
3. A. Oleksiak, S. Cieślak, K. Marcinek and W. A. Pleskacz, "Design and Verification Environment for RISC-V Processor Cores," 2019 MIXDES - 26th International Conference "Mixed Design of Integrated Circuits and Systems", Rzeszow, Poland, 2019, pp. 206 209, doi: 10.23919/MIXDES.2019.8787108.
4. China RISC-V Alliance. accessed on Jan. 1, 2023.
5. SEMICO Research Corporation, "RISC-V market analysis: The new kid on the block." Nov. 2019.
6. Y. Lee et al., "An Agile Approach to Building RISC-V Microprocessors," in IEEE Micro, vol. 36, no. 2, pp. 8-20, Mar.-Apr. 2016, doi: 10.1109/MM.2016.11.
7. B. W. Mezger, D. A. Santos, L. Dilillo, C. A. Zeferino and D. R. Melo, "A Survey of the RISC-V Architecture Software Support," in IEEE Access, vol. 10, pp. 10.1109/ACCESS.2022.3174125. 51394-51411, 2022, doi:
8. Gao TWang YZhu MWu XZhou DBi Z(2025)An RISC-V PPA Fusion Cooperative Optimization Framework Based on Hybrid Strategies Transactions on Very Large Scale Integration (VLSI) Systems10.1109/TVLSI.2024.349685833:1(140-153)Online publication date: Jan-2025
9. E. Cui, T. Li and Q. Wei, "RISC-V Instruction Set Architecture Extensions: A Survey," in IEEE Access, vol. 11, pp. 24696-24711, 2023, doi: 10.1109/ACCESS.2023.3246491.
10. S. Bora and R. Paily, "A High-Performance Core Micro-Architecture Based on RISC-V ISA for Low Power Applications," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 6, pp. 2132-2136, June 2021, doi: 10.1109/TCSII.2020.3043204.
11. K. Asanovic and D. A. Patterson, "Instruction sets should be free: ´ The case for RISC-V," Dept. EECS, Univ. California, Berkeley, Tech. Rep. UCB/EECS2014-146, Aug.2014.
12. people.eecs.berkeley.edu/~krste/papers/EECS-2016-1