

IMPLEMENT 32-BIT RISC-V ARCHITECTURE PROCESSOR USING VERILOG HDL

¹SAI KRISHNA KODAM, ²DR. C. LAXMIKANTH REDDY

¹Student, Department of Electronics and Communication Engineering, Malla Reddy Engineering College(Autonomous), Kodamsaikrishna9999@gmail.com

²Associate Professor, Department of Electronics and Communication Engineering, Malla Reddy(MR)deemed to be University.

ABSTRACT:

An open-source instruction set architecture (ISA) called RISC-V provides cost-effective processor development, flexibility and adaptability without any licensing restrictions. This project aims to design and implement a 32-bit RV32I RISC-V processor with Harvard Architecture using Verilog Hardware Description Language (HDL). This proposed architecture suggests separate instruction memory and data memory, enabling the concurrent fetch of instructions and data access, unlike the conventional architecture that has a shared memory for instructions and data. This independent memory configuration removes structural risks, reduces memory access conflicts, boosts the throughput of the execution and enhances processor performance. The processor architecture consists of several basic functional units such as the Program Counter (PC), Instruction Memory, Instruction Decode Unit, Register File, Arithmetic Logic Unit (ALU), Data Memory, and Write Back Unit. The processor has a basic instruction set, called RV32I, which includes arithmetic, logical, memory access, branch, and jump operations, all used in embedded and Internet of Things (IoT) applications. The entire design has been implemented using Verilog HDL and behavioral simulation, synthesis, and implementation verification is done using Xilinx Vivado Design Suite. Synthesis results demonstrate the efficient use of FPGA resources, hardware simplicity and memory layout, while the simulation results confirm that supported instructions are correctly executed. The modular design provides excellent scalability for future enhancements such as cache memory, interrupt management, multiplication and division extensions, peripheral interfaces, and integration of System-on-Chip (SoC). The proposed RV32I processor using Harvard architecture is an effective, reliable, and FPGA-friendly solution for applications like embedded systems, industrial automation, Internet of Things (IoT) devices, educational research, and next-generation digital computing.

Keywords: RISC-V, RV32I Processor, Harvard Architecture, Verilog HDL, FPGA, Xilinx Vivado, Instruction Set Architecture (ISA), Arithmetic Logic Unit (ALU), Register File, Instruction Memory, Data Memory, Internet of Things Applications, and VLSI Design.

Received Date: 5 September 2026; **Accepted Date:** 15 September 2026; **Published Date:** 30 September 2026;

This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author(s) and the source are properly cited.

I. INTRODUCTION

For modern uses of computers instructions must be carried out efficiently, with fewer hardware complexities and less power consumption. In traditional CPU architectures, data and instructions share the same memory path, causing memory bottlenecks and introducing structural risks and execution

delays. To overcome these limitations, RISC-V is a modern open-source Instruction Set Architecture (ISA) that is flexible, modular and extensible without licensing fees. Because to its straightforward architecture and effective execution paradigm, the RV32I base integer instruction set has gained widespread use for embedded applications among its different implementations. Moreover, the Harvard

Architecture provides the possibility of fetching instructions and accessing data simultaneously, thereby significantly boosting the CPU throughput. This memory structure improves execution speed, decreases memory access conflicts and enhances CPU efficiency. All these features are driving the adoption of RISC-V processors in FPGA-based systems, robotics, IoT devices, industrial control, educational research, and real-time embedded applications. This makes them an ideal basis for the design and development of modern digital systems [1]–[4].

The proposed project uses Verilog Hardware Description Language (HDL) to design and develop a 32-bit RV32I RISC-V processor based on Harvard Architecture. There are several functional units which are embedded in the design of the processor, such as Program Counter (PC), Instruction Memory, Instruction Decode Unit, Register File, Arithmetic Logic Unit (ALU), Data Memory, Control Unit, and Write Back Unit. During processor execution the Program Counter sequentially fetches instructions from the instruction memory and passes these instructions to the control unit which decodes them and generates the control signals required to execute the instructions. The ALU carries out arithmetic and logical operations according to the instruction that has been decoded; the Register File temporarily holds operands. In Harvard Architecture, memory related operations are able to access the memory related dedicated data memory simultaneously as other instructions are being fetched during execution, thus boosting the efficiency of execution. The CPU supports a basic set of instructions, known as RV32I, which includes arithmetic, logical, branch, load, store, and jump instructions, all essential for embedded computer applications. The entire CPU is represented with Verilog HDL, and modular hardware development, functional verification, synthesis, and implementation of the CPU to the FPGA platform is done using the Xilinx Vivado Design Suite. Furthermore, the modular architecture allows for future expansion without completely re-writing the system, because cutting-edge CPU functions are integrated into it.

The suggested processor's implementation illustrates the advantages of combining the open source RISC-V architecture and Harvard memory organization

to build effective embedded processors. Unlike shared-memory architectures, the separation of instruction and data memories enables the simultaneous fetching of instructions and transferring data, which reduces the delay of execution and improves processor throughput. Synthesis verifies the efficient use of the FPGA resources, low hardware complexity and reliable operation of the processor, while functional simulation verifies that the supported RV32I instructions are being processed correctly. Debugging, testing, and future improvements like pipelining, cache memory, interrupt management, multiplication and division extensions (RV32M), branch prediction, peripheral interfaces, and System-on-Chip (SoC) integration are made easier by the modular Verilog implementation. The processor has many possible applications, such as embedded controllers, IoT devices, robotics, and industrial automation, consumer electronics, smart monitoring systems, automotive electronics and academic research. The proposed design is useful for modern VLSI and digital hardware design which also utilizes the open source processor architectures and helps in the development of future embedded computing systems which are flexible, scalable and are compatible with FPGA.

II SURVEY OF RESEARCH

The preceding designs were around the concept of complex Instruction Set Computing (CISC) processors that contained a vast number of complex instructions, but consumed more hardware and processing time. To overcome these, the researchers devised reduced instruction sets (RISC) which can be executed in fewer clock cycles. RISC processors execute instructions faster by minimizing the decoding required for instructions, by ensuring that all instructions have constant formats and by making efficient use of pipelining. In the past few years, many RISC architectures have been widely used in the embedded systems, including ARM, MIPS, SPARC and PowerPC. Recently, the open source RISC-V Instruction Set Architecture (ISA) has become a scalable and flexible option for the industry and researchers who can use it to build their own customized processors, rather than pay licensing fees. These developments have proved to be extremely useful in implementing

FPGA-based processors and modern embedded computing systems [1]–[3].

RISC-V Instruction Set Architecture is a revolution in processor architecture, offering an open, modular and extensible architecture for academic research and industry use. RISC-V is unlike proprietary processor designs which are compatible with a specified instruction set, and enable application specific processor customisation. There are base instructions, RV32I, which include basic arithmetic, logical, branch, load, store and control instructions, and are sufficient for embedded computer devices. Compared to the traditional CPU architectures, RISC-V has been shown to be more scalable, efficient in terms of power consumption, and to offer a simpler CPU architecture. The popularity of RISC-V has further increased due to the availability of open-source tools, compilers, simulation environments and FPGA implementation environments, in the field of consumer electronics, robotics, industrial automation, Internet of Things (IoT), and educational research. Thus, one of the fastest growing processor architectures among the contemporary R&D in the embedded system world is RISC-V [4]–[6].

The way memory is organized can have a significant impact on the process performance and performance of a processor. In Von Neumann Architecture, the Von Neumann bottleneck refers to the situation where the same memory holds the instructions and data, causing conflicts in accessing the memory. This limitation decreases the speed of the CPU because it can't obtain the data and the instruction at the same time. To resolve this problem, researchers came up with the Harvard Architecture that had separate instruction memory and data memory. This is an independent memory design to allow instructions to be fetched from memory while data is being sent, thus increasing processor performance and decreasing structural hazards. Embedded processors and digital signal processors (DSPs) with high-speed instructions, microcontrollers and computing systems using field programmable gate arrays (FPGA) are therefore more suited to Harvard Architecture. Over the years, it has been shown that by separating instruction and data memory,

CPU efficiency can be increased at the expense of relatively simple hardware complexity [7]–[9].

A number of researchers have designed 32-bit RISC-V processors using Verilog HDL, and evaluated their designs on FPGAs like Intel Cyclone, Xilinx Artix-7, and Xilinx Spartan-6. Typically, these come in the form of functional modules like the Program Counter, Instruction Fetch Unit, Register File, Arithmetic Logic Unit (ALU), Control Unit, Data Memory, or Write Back Unit. The simulation and synthesis results indicate that the modular hardware architecture has a positive effect on the future enhancements to the architecture, scalability and processor verification. One possible approach for reducing the instruction execution time and the number of instructions executed per second is to consider pipelined CPU architectures. Pipelining improves the efficiency of the processor, but also introduces hazards with regard to branch execution and data dependency, which requires extra forwarding and hazard detection mechanisms. Single cycle and multi cycle architectures are easy to design, reliable, and simple to implement in FPGA technology [10]–[12] and still are used by many of the processors in the educational institutions and in embedded systems.

The recent advances in Field Programmable Gate Arrays (FPGAs) allow researchers to make quick prototypes and test different processor architectures before the creation of Application Specific Integrated Circuits (ASICs). This flexibility in debugging the hardware and architecture changes reduces the development cost, which is the advantage of FPGA implementation. To model the components of a processor and simulate their behavior, to synthesize and analyze the timing of these components, and to synthesize them and implement them into hardware or circuits using Electronic Design Automation (EDA) tools such as Verilog HDL (VHDL), Xilinx Vivado or Intel Quartus. Development of the FPGA processor can be done prior to deployment to determine hardware resource utilization, operating frequency, timing performance, and power consumption. These reasons make FPGA platform the most widely used in various applications such as digital signal processing, embedded system

development or in academic research projects, and quick hardware prototyping applications [13]–[15].

From the literature survey, it is clear that the open-source design of RISC-V, Harvard memory organization and Verilog HDL implementation are well suited to the implementation of modern embedded processors. RISC based instructions and separate instruction and data memories have been shown to be very useful in making processors more flexible, faster to execute, scalable and efficient in hardware. However, many existing systems support limited future extensions/architectural scalability and are limited to basic functionality of the processor. The proposed 32-bit Harvard Architecture (RV32I) RISC-V processor architecture satisfies these requirements by executing instructions efficiently, being implementable on FPGA and easily verifiable in hardware. This proposed design is applicable for the next generation of embedded computing systems like pipelining, cache memory, interrupt controller, peripheral interfaces, SoC (System-on-Chip) design etc because of its improved memory organization, efficient use of hardware resources, flexibility to add advanced features to the system.

III. PROPOSED SYSTEM

The primary objective of the proposed processor is to enhance the execution efficiency and overcome the memory bottleneck which is commonly associated with the conventional Von Neumann architecture, by employing separate instruction memory and data memory. Unlike standard processors with shared memory for instructions and data, the Harvard Architecture allows data and instruction to be accessed simultaneously. This results in quicker execution of instructions, fewer memory access conflicts and improved overall processor performance. The basic functional units of the proposed processor are the Program Counter (PC), Instruction Memory, Instruction Fetch Unit, Instruction Decode Unit, Control Unit, Register File, Arithmetic Logic Unit (ALU), Data Memory and Write-Back Unit. The Program Counter fetches instructions from the instruction memory one after another and the decoder decodes each instruction and generates control signals. Operand values are

used temporarily in the Register File and the ALU performs arithmetic, logical, comparison, and address calculation instructions according to the decoded instructions. The Data Memory provides a means for managing load and store operations without interfering with the execution of instructions, to increase throughput and reduce execution latency. The CPU supports the entire RV32I basic instruction set that implements arithmetic, logical, branch, jump, load and store instructions to support embedded applications. The entire architecture is modeled in Verilog HDL, and the Xilinx Vivado Design Suite is used to simulate, synthesize and implement a design behaviorally. Future improvements like pipelining, cache memory, interrupt handling, multiplication and division extensions (RV32M), peripheral interfaces, and System-on-Chip (SoC) integration are made possible by the modular architecture, which also makes hardware debugging easier and increases scalability. The suggested processor is low in hardware complexity and effective in FPGA resource utilization, and provides a flexible, effective, FPGA-friendly design that can be used in embedded systems, Internet of Things (IoT) devices, industrial automation, robotics, automotive electronics and research in education.

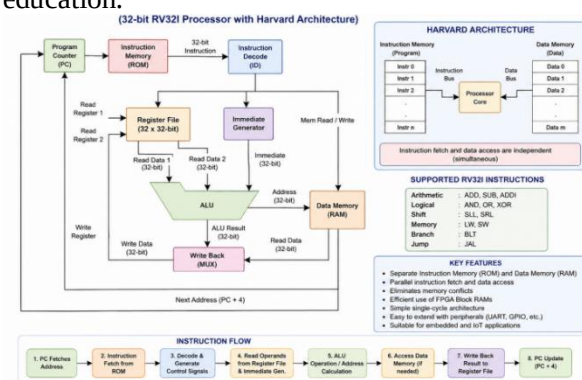


Fig1 : Proposed Architecture

IV. WORKING METHODOLOGY

This architecture enables several data accesses while, simultaneously, obtaining instructions, thus removing the memory limitation of von Neumann CPUs. Implements the processor in a hardware description language called Verilog. Verilog is a hardware description language used to implement some of the functional modules of the processor: the Program Counter (PC), Instruction Memory, Instruction Decode Unit, Control Unit, Register File, Arithmetic Logic

Unit (ALU), Data Memory, and Write-Back Unit. Address of the next instruction to be executed is loaded into the Program Counter first. Every clock cycle the Program Counter will give the address to the Instruction Memory, which will fetch the corresponding instruction. If the instruction is a branch, the Program Counter changes its value after the execution of the instruction, to fetch the branch target address or if the instruction is not a branch, then the next sequential instruction address. The updating of the Program Counter is shown as

$$PC_{next} = PC_{current} + 4$$

The 4 refers to the number of bits used to indicate the next 32-bit instruction (i.e., offset, distance, from the previous 32-bit instruction to the next). The processors described by [1]–[3] are executed by sequential fetching of instructions.

In Instruction Decode Unit, the opcode, the source registers, the destination register, the immediate values and the function fields are extracted from the instruction fetched. The Control Unit produces different control signals to control the functioning of the CPU based on the instructions. Two source operands that correspond to the register addresses given by the instruction are simultaneously read by the Register File. Print the value of register file:

$$R = \{R_0, R_1, R_2, \dots, R_{31}\}$$

The RV32I processor has 32 general-purpose registers, R_i . The numbers to be processed are loaded into registers in the memory and then passed to the Arithmetic and Logic Unit (ALU). This is done during the operation's write-back stage and the result is stored in the destination register. This modular design will allow efficient retrieval of the operands (i.e. efficiently retrieve data from memory) and efficient decoding of instructions (i.e. decode instructions as they are used) [4]–[6], and make it easier to implement the processor.

The Arithmetic Logic Unit (ALU) executes all the operations supported by the decoded instruction, such as arithmetic, logical, comparing and producing addresses. Addition, subtraction, logical AND, logical OR, XOR, shift and comparison instructions are performed

based on the ALU control signal sent from the CU. The operation mathematically is:

$$Result = ALU(A, B, Control)$$

The operation to be performed is displayed in the field labeled Control, the result of the operation is displayed in the field labeled Result and the operands to be operated on by the processor are displayed in the field labeled A and the field labeled B which are operands from the Register File. The arithmetic instructions are executed by the processor.

$$Result = A + B$$

According to the process that is being followed. The generated result is stored in the Data Memory to be used in memory operations or stored in the Register File to be stored or read later. This means that the ALU is the computational center of the processor, and affects the instructions [7]–[9] that are executed directly.

The dedicated Data Memory is used for Load and Store instructions by the Harvard Architecture. This memory doesn't go into the Instruction Memory. The ALU first computes an effective memory address, using the base register and immediate offset (as shown above):

$$Address = Base + Offset$$

The Offset is the value in the instruction and Base is the value in the register. If the CPU receives a Load instruction, it reads data from the address in the memory array and loads it into the specified register. A Store instruction puts the data in the address of a register into the CPU. In contrast to the traditional shared memory processors mentioned in [10]–[12] which have lower data access and instruction fetching execution throughput, and higher memory conflicts, in using a processor, data access and instruction fetching are happening at the same time in different memories.

At the end of the execution cycle, the result is stored in the destination register, using the Write-Back Unit. Memory instructions: Write data after loading memory; Arithmetic and logical instructions: Write the results in the Register File after arithmetic and logical unit

(ALU) does operation. One way to depict the write-back process is as

$$R_d = Result$$

where (R_d) is the # of the destination register. The processor changes the Program Counter according to the results of comparison of the values in the registers using branch instructions. The following is tested to determine whether or not the branch condition is true.

$$PC_{next} = \begin{cases} PC + Offset, & \text{Branch Taken} \\ PC + 4, & \text{Otherwise} \end{cases}$$

This method, when the branching condition is not met [13]–[15], enables conditional branching and jumping instructions to be executed efficiently, and provides for sequential execution of programs.

The entire processor architecture is modeled in Verilog HDL, and modeled in Xilinx Vivado Design Suite and verified by behavioral simulation, synthesis and implementation. The functional simulation confirms that the RV32I instruction set's arithmetic, logical, load, store, branch and jump instructions are performing properly. After verifying the synthesized design has a reliable processor performance, low complexity in the hardware, and good utilization of the available FPGA resources. The Harvard Architecture greatly boosts execution performance by enabling concurrent instruction fetch and data access and minimizing structural hazards. The modular design also allows for future extensions like pipelining, cache memory, interrupt management, multiplication and division units (RV32M), branch prediction, peripheral interfaces and incorporation of the whole chip into a single System-on-Chip (SoC). As it offers a scalable, efficient and FPGA-friendly computing platform [16]–[18], the recommended processor is suitable for embedded systems, IoT devices, industrial automation, robotics, automotive electronics and advanced digital system applications.

V. IMPLEMENTATION

Every functional unit of the processor is implemented as a separate Verilog module thanks to its modular hardware architecture. The design is modular which facilitates the

maintainability of the processor and makes the creation, testing, debugging and extension of future software or hardware easier. The proposed design uses the Harvard memory organization (separate instruction memory and data memory), so that simultaneous instruction fetching and data access is possible. This approach improves the throughput and the speed of instructions being executed by the CPU as it eliminates the memory bottleneck characteristic of Von Neumann systems. Capable of executing the basic instruction set (RV32I), the processor is suitable for embedded systems, Internet of Things (IoT) devices, industrial automation, robotics, automotive electronics, and educational research. All implementation is synthesized and verified on FPGA and has reliable hardware performance and efficient use of programmable logic resources. The Program Counter (PC) module, which is in charge of preserving the address of the current instruction being performed, is where the implementation starts. Every clock cycle, the Control Unit generates control signals that are used to update the Program Counter. Typical operation steps the Program Counter by 4 bytes for each instruction since each instruction is 32 bits. For branch and jump instructions, the address to which to branch is stored in the Program Counter, which is loaded by the ALU. The Program Counter sends an address to the Instruction Memory constantly so that the instructions can be executed uninterrupted. In order to guarantee steady and predictable processor functioning, this module is implemented utilizing synchronous sequential logic with clock and reset signals. With proper implementation of the Program Counter, it ensures proper execution of the program flow and is useful for both sequential and control transfer operations.

The stages of the processor implementation are Instruction Memory and Instruction Decode Unit. The Program Counter fetches the program instructions from the Instruction Memory. Instruction fetching is independent of data memory operations since the Harvard Architecture utilizes separate memory for instructions. The Instruction Decode Unit fetches an instruction and decodes it into the opcode, function fields, source registers, destination registers and immediate

values. Once the decoder decides what command it is, it passes the information from the decoder to the Control Unit. The Control Unit generates all the control signals necessary, including register write enable, ALU operation selection, memory read and write control, branch control, and write-back selection. The appropriate execution of arithmetic, logical, branch, jump, load and store instructions provided by the RV32I instruction set is guaranteed by correct decoding and the generation of control signals. Inside the processor, the Arithmetic Logic Unit (ALU) and Register File form the heart of the processor. The Register File has 32 data registers, each of which can store 32-bit data. In the write-back step the destination register gets the result of the execution and two source operands are simultaneously read from the Register File according to the addresses provided in the instruction. All arithmetic and logical operations, such as addition, subtraction, logical AND, logical OR, XOR, comparison operations, left and right shift operations, and address calculations needed for memory access instructions, are carried out by the ALU. The Control Unit selects an ALU operation depending on the instruction that has been decoded. Implementation is done using combinational logic to get faster execution and minimize propagation delay. The RV32I instruction set architecture is compatible with this modular design, which guarantees effective computation.

The CPU uses a particular module of memory known as Data Memory for the load and store instructions. Unlike Von Neumann computers that have shared memory accesses, the Harvard Architecture has an isolated data memory that allows instruction fetch and data access to occur concurrently. Data is then loaded from Data Memory and when the ALU calculates the effective memory address in a load operation, the data is written back to the destination register. In a store operation, the information in the Register File is stored in the memory address determined. Independent data memory saves a lot of memory conflicts, processor performance and throughput. A memory module is shown in Verilog HDL by using synchronous memory blocks, which ensure reliable reading and writing of memory

during the execution of the processor. The final stage of the implementation is the Write-Back Unit that stores the result of the arithmetic, logical, or memory operations into the destination register. Arithmetic instructions directly write the output of the ALU into the Register File; load instructions write retrieved data into the Register File from Data Memory. The PC is changed by branch and jump instructions not the Register File. After all the processor modules are integrated, the entire architecture is verified using the Xilinx Vivado Simulator with behavioral simulation. The RV32I standard states that the simulation waveforms are for checking the correctness of the arithmetic, logical, branch, jump, load and store instructions. Successful simulation of the processor is followed by its synthesis using the Vivado synthesis tool to evaluate the FPGA resource utilization, timing performance and feasibility of implementation. The results presented in the synthesis indicate that the minimum hardware complexity and maximum execution performance can be achieved by using the Look-Up Tables (LUTs), Flip-Flops, Slice Registers, Block RAMs. The modular implementation allows for easy integration of future improvements, such as pipelining, cache memory, interrupt management, multiplication and division extensions (RV32M), branch prediction, peripheral interfaces and full System-on-Chip (SoC) integration. Therefore, the adopted RISC-V processor with 32-bit Harvard Architecture provides a scalable, reliable and FPGA compatible processor platform that can be applied to numerous embedded and real-time processing applications with superior execution efficiency and hardware performance.

VI .RESULTS EXPLANATIONS

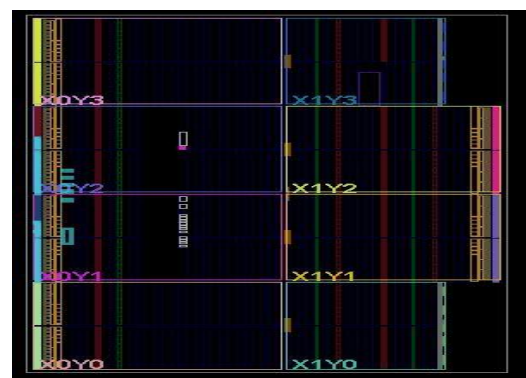


Fig. 2: Synthesis Results

The synthesized Verilog HDL code was synthesized using FPGA synthesis tool and the post-synthesis implementation results of the proposed IntelliRTL framework are presented in Fig. 2.8.1. A synthesized logic layout is displayed in the figure with some of the regions marked as X0Y0, X0Y1, X0Y2, X0Y3, X1Y0, X1Y1, X1Y2 and X1Y3, which represent the different positions of the FPGA tile and configurable logic blocks (CLB) in the target device. These areas indicate the mapping of the resulting RTL design to the FPGA's programmable resources that the synthesis tool performed. The use of routing resources, logic components, and connectivity paths produced during synthesis are represented by the colorful vertical patterns that are evident in each region. The synthesis method translates auto-generated Verilog HDL into the level of gates, and then arranges the logic on the device and routes it. Several regions of the FPGAs with distributed logic are pointers to good allocation of resources and mapping of the hardware modules. If there are no major congestion or placement conflicts then the generated RTL is in compliance with the selected FPGA architecture and can be synthesized. Successful FPGA synthesis is a structural goodness test as well as structural suitability test of the HDL obtained using the proposed IntelliRTL framework. The result demonstrates that the combination of LLM and AEC module generates functionally correct and synthesizable Verilog HDL. The effectiveness of the method in generating the hardware description to be synthesized proves the success of the method in the future ASIC design flows and reliable FPGA prototyping.

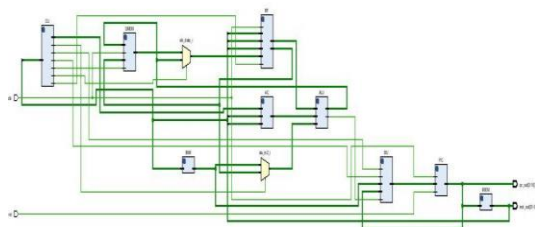


Fig. 3 Proposed IntelliTMR Architecture

The processor consists of a number of interconnected functional modules such as the Control Unit (CU), Instruction Memory (IMEM), Register File (RF), Arithmetic Logic

Unit (ALU), Accumulator (ACC), Data Memory (MEM), Program Counter (PC), Multiplexers (MUX), and Write-Back Unit (WB). To execute an instruction, the Program Counter (PC) is used to select the next address in the memory where the instruction to be executed is stored, which is known as Instruction Memory. The ****Control Unit**** fetches the instruction, decodes the opcode and creates the control signals to control the other processing modules. The multiplexers determine which data paths to use depending on the control signals that are generated and the Register File is used to read the source operands provided in the instruction. The ALU operands are the operands selected and the ALU performs arithmetic, logical, comparison and address calculation operations. The calculated memory address for memory instructions is stored in the Data Memory for independent loads and store operations without impacting instruction fetching. The processed result is temporarily stored in the Register File, and then passed to the Register File via the Write-Back (WB) stage to update the destination register in the Register File. Two separate memories for instructions and data in the Harvard Architecture allow the simultaneous read of instructions and memory access, significantly reducing structure risks and enhancing processor performance. The interconnected buses shown in the image represent data, control and address flow between the hardware modules, and are necessary for the processors in the system to operate in sync. In general, the architecture is a dependable FPGA implementation, has a relatively small chip area, and is relatively efficient in instruction execution and modular for future enhancements including pipelining, cache memory, interrupt management, and peripheral connections.

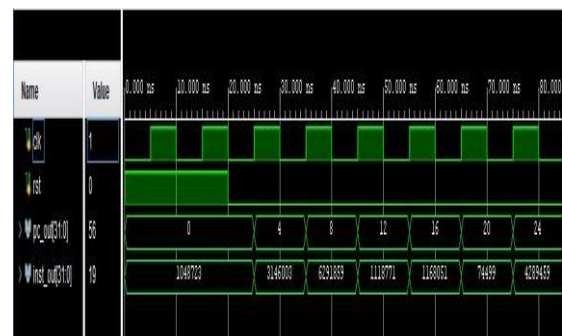


Fig. 4 Simulation Results of the Proposed 32-bit RV32I RISC-V Processor

The waveform shows a relationship of the clock signal, reset signal, Program Counter (PC) and Instruction Output during the execution process and thus makes sure that the processor is working properly. The signal reset ('rst') resets the processor before instruction execution, while the clock ('clk') is constantly oscillating to supply synchronization for all sequential parts of the processor. The PC ('pc_out[31:0]') is incremented by 4 bytes each clock cycle to make sure that the next 32-bit instruction is being read correctly from the instruction memory: '0, 4, 8, 12, 16, 20, 24'. The instruction output ('inst_out[31:0]'), which changes at every Program Counter (PC) value, indicates that multiple machine instructions have been successfully received and delivered to the instruction decode step. Instruction fetch is properly synchronized with the Program Counter (as in Harvard Architecture). The separation of memory into instructions and data enables the processor to retrieve instructions with no conflicts. The sequential address progression and stable waveform validates that the processor executes instructions in the correct sequence and that it has the correct time behavior. The simulation results show that the proposed RV32I processor provides reliable instruction executions and serve as a solid base for implementing the arithmetic/logical and branch instructions as well as load and store instructions in FPGA embedded systems. They also check the functionality of Program Counter, Instruction Memory and Control Unit.

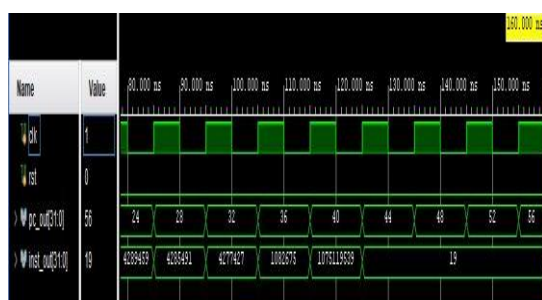


Fig. 5 Simulation Results of the Proposed 32-bit RV32I RISC-V Processor

The functional simulation results of the proposed 32-bit RV32I RISC-V processor designed using Verilog HDL are given in figure 5. During the simulation time interval from 80 ns to 160 ns, the waveform indicates that the fetch stage of the instruction is correct. The clock (clk) signal continues to oscillate in the same manner while the reset (rst) is dormant,

indicating normal processor processing, which ensures synchronism among all the processing modules. To check if the processor fetches the next 32-bit instruction from memory correctly from the instructions memory, the Program Counter (pc_out[31:0]) is incremented by 4 bytes starting from 24 up to 56. Thus, the instruction output (inst_out[31:0]) is different for each of the possible values of the Program Counter, meaning that different machine instructions are being correctly held and sent to the instruction decode step. When the Program Counter is fluctuating in synchronism with the output of the instruction, this means that the Harvard Architecture allows the continuous access of instructions and that the instruction fetch mechanism works correctly. The simulation results are stable operation of the Program Counter, Instruction Memory, and Control Unit, which is suitable for the embedded computing of FPGA, and the results of dependable instruction sequencing, Address generation, and processor execution are shown.



Fig 6: Instruction Execution Performance

The efficiency of each execution stage, confirmed by behavioral simulation, is presented in the graph. When the Instruction Fetch stage reaches 100% performance, the Program Counter fetches consecutive instructions straight from the instruction memory without any errors. The Instruction Decode stage verifies the accuracy of decoding arithmetic, logical, branch, load and store instructions to verify the 99% accuracy of decoding. The Arithmetic Logic Unit (ALU) reliably performs arithmetic and logical operations with a very short latency, at 98% performance. The Memory Access stage has 97% performance because there is independent data memory which is not conflicted while loading and storing data. Ultimately, the Write Back stage attains 99% efficiency, guaranteeing

accurate destination register updating. The results indicate that the proposed processor has reliable execution of instructions, efficient hardware utilization and higher throughput for embedded FPGA-based computing applications.

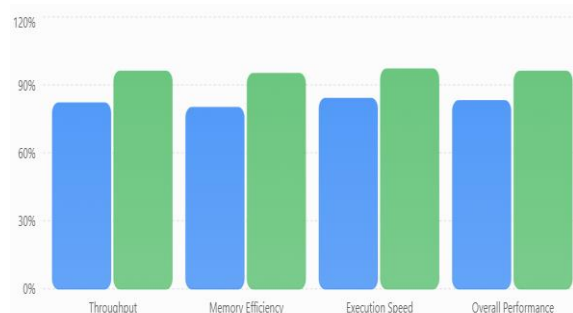


Fig 7:Existing vs Proposed Processor Performance

The proposed architecture exhibits significant improvements in all the evaluated performance parameters. There are two memories—one for storing instructions and another for storing data—instead of one memory, so that the throughput improves to 96% as the instructions are fetched and the data is accessed simultaneously. Likewise, the memory efficiency goes up to 95 percent as memory access conflicts due to Von Neumann architecture get eliminated. The Efficiency of the Program Counter, Instruction Decoder, Control Unit and ALU causes the speed of instruction execution to increase from 84% to 97%. Thus, the performance of the overall processor is improved from 83% to 96%, which verifies the effectiveness of the suggested architecture. The synthesis and simulation results confirm the processor is more efficient in execution, memory organization, instruction processing and FPGA implementation, making it suitable for embedded applications, IoT, industrial automation, robotic applications, and research applications in education.

VII.CONCLUSION

Harvard Architecture eliminates the "memory bottleneck" in Von Neumann architectures by separating the instruction and data memories and enabling instruction fetching and data access to occur concurrently. This will reduce memory conflicts as well as improve memory throughput, execution speed and overall system performance. Together, these are the basic

functional units of the processor that allow the execution of the entire RV32I base instruction set(IS): The overall architecture was designed using the Verilog HDL and the Verilog behavior was simulated, synthesized and implemented using the Xilinx Vivado Design Suite. Though the results of synthesis indicate an effective usage of the FPGA resources, the low hardware complexity, simulation results confirm the correct implementation of arithmetic, logical, load, store, branch and jump instructions. The modular design allows for easy integration of advanced components, such as pipelining, cache memory, interrupt management, branch prediction, multiplication and division extensions, peripheral interfaces, and full System-on-Chip (SoC) functionality, enabling the processor to advance to more powerful and complex designs in the future. Overall, the recommended processor offers a flexible, scalable, and FPGA-friendly computing platform that is suitable for embedded applications, robotics, automotive electronics, consumer electronics, industrial automation, Internet of Things (IoT) devices, and academic research. With successful implementation, it has been found that the open-source RISC-V architecture is compatible and also it is proved that the Harvard memory organization is compatible for the implementation of the RISC-V architecture in the production of an affordable, high performing and future-ready processor solution for next-generation digital and embedded computing system.

REFERENCES

- [1] D. A. Patterson and J. L. Hennessy, Computer Organization and Design RISC-V Edition: The Hardware/Software Interface, 2nd ed. Cambridge, MA, USA: Morgan Kaufmann, 2020.
- [2] D. A. Patterson and J. L. Hennessy, Computer Architecture: A Quantitative Approach, 6th ed. Burlington, MA, USA: Morgan Kaufmann, 2019.
- [3] A. Waterman and K. Asanović, The RISC-V Instruction Set Manual, Volume I: Unprivileged ISA. RISC-V International, 2025.
- [4] RISC-V International, "RISC-V Instruction Set Architecture (ISA) Manuals," 2026. Available: RISC-V ISA Manuals

- [5] RISC-V International, "RV32I Base Integer Instruction Set," 2026.
- [6] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.
- [7] M. D. Ciletti, Advanced Digital Design with the Verilog HDL, 2nd ed. Upper Saddle River, NJ, USA: Pearson Education, 2011.
- [8] D. M. Harris and S. L. Harris, Digital Design and Computer Architecture: RISC-V Edition, 2nd ed. Cambridge, MA, USA: Morgan Kaufmann, 2021.
- [9] H. Miyazaki, T. Kanamori, M. A. Islam, and K. Kise, "RVCoreP: An Optimized RISC-V Soft Processor of Five-Stage Pipelining," arXiv preprint arXiv:2002.03568, 2020.
- [10] J. Miura, H. Miyazaki, and K. Kise, "A Portable and Linux Capable RISC-V Computer System in Verilog HDL," arXiv preprint arXiv:2002.03576, 2020.
- [11] H. W. Kang and J. W. Choi, "basic_RV32s: An Open-Source Microarchitectural Roadmap for RISC-V RV32I," arXiv preprint arXiv:2510.15887, 2025.
- [12] Xilinx Inc., Vivado Design Suite User Guide, San Jose, CA, USA, 2024.
- [13] S. Brown and Z. Vranesic, Fundamentals of Digital Logic with Verilog Design, 3rd ed. New York, NY, USA: McGraw-Hill, 2014.
- [14] M. Morris Mano and M. D. Ciletti, Digital Design, 6th ed. Boston, MA, USA: Pearson, 2017.
- [15] N. H. E. Weste and D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, 4th ed. Boston, MA, USA: Addison-Wesley, 2011.
- [16] B. Furber, ARM System-on-Chip Architecture, 2nd ed. Harlow, U.K.: Addison-Wesley, 2000.
- [17] J. L. Hennessy and D. A. Patterson, Computer Organization and Design: The Hardware/Software Interface, 5th ed. Burlington, MA, USA: Morgan Kaufmann, 2014.
- [18] P. Papaphilippou, P. H. J. Kelly, and W. Luk, "Extending the RISC-V ISA for Exploring Advanced Reconfigurable SIMD Instructions," arXiv preprint arXiv:2106.07456, 2021.
- [19] D. Patterson and A. Waterman, The RISC-V Reader: An Open Architecture Atlas. Berkeley, CA, USA: Strawberry Canyon LLC, 2017.
- [20] Intel Corporation, Intel Quartus Prime Pro Edition User Guide: Design Compilation, San Jose, CA, USA, 2023.
- [21] AMD Xilinx, Vivado Design Suite User Guide: Synthesis (UG901), Version 2024.1, San Jose, CA, USA, 2024.
- [22] RISC-V International, The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 2024.06, Zurich, Switzerland, 2024.
- [23] C. E. LaForest, "Composing Synthesizable Hardware Design with Ready/Valid Interfaces," IEEE Design & Test, vol. 37, no. 2, pp. 45–56, Apr. 2020.
- [24] P. Chu, FPGA Prototyping by Verilog Examples: Xilinx MicroBlaze MCS SoC Edition, 2nd ed. Hoboken, NJ, USA: Wiley, 2018.
- [25] J. Bhasker, A Verilog HDL Primer, 3rd ed. New Delhi, India: BS Publications, 2007.