Dr.A.Ranganayakulu, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

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

Design and implementation of RV321 processor core in VLSI

[1] Dr.A.Ranganayakulu,[2] Dr.D.Satyanarayana,[3] Venna Kasieswari,[4] Dudekula Rijvana,[5] Duddela Ramya,[6] Shaik Farida

[1] Professor & HOD, Department of ECE, Krishna Chaitanya Institute of Technology & Sciences, Markapur.
[2] Associate Professor, Department of ECE, Krishna Chaitanya Institute of Technology & Sciences, Markapur.
[3,4,5,6] Student, Department of ECE, Krishna Chaitanya Institute of Technology & Sciences, Markapur.

Abstract— A single-cycle processor with support for 30 instructions based on the RV32I instruction set architecture (ISA) is presented in this research. Because it was necessary to fetch, decode, execute, and finish each instruction within one clock cycle, the single-cycle architecture was selected. When simplicity and power economy are more important than execution speed, this strategy works well for small-scale applications. Important parts of the architecture include a data memory for storing information, an arithmetic logic unit (ALU) for performing logical and mathematical operations, a register file for storing instructions, and instruction memory for storing data. In order to control the flow of data and make sure instructions are executed correctly, control signals are created depending on the opcode. The Verilog hardware description language (HDL) was used in the development of the CPU. As a platform for implementation, the Arty-S7-FPGA board was the goal of the design synthesis. Basic processes including data transmission, math computations, and control flow are effectively handled by the architecture. The test bench is built using verilog and all thirty instructions are tested in the Vivado software environment for operation verification. This research explores the RISC-V architecture from the bottom up and paves the way for improved implementations of pipelined processors in the future. Nouns: Vivado, Verilog, Single-Cycle Processor, RV32I Instruction Set Architecture.

Keywords: RISC-V, RV32I Instruction Set Architecture, Single-Cycle Processor, Verilog HDL, Vivado, FPGA Implementation, Arty-S7 FPGA Board, Arithmetic Logic Unit (ALU), Register File, Instruction Memory, Data Memory, Control Signals, Processor Design, Computer Architecture, Hardware Description Language.

I. INTRODUCTION

These days, processors are the brains of the computer, thanks to developments in large-scale integrated circuit technology and improvements in processor architecture. Electronic systems rely on Central Processing Units (CPUs), integrated circuits, and memory components to carry out the essential tasks of information processing. Designing and implementing an efficient central processing unit (CPU) has become a critical topic of emphasis because of the profound effect it has on system performance. When it comes to computer systems, efficiency and scalability are heavily dependent on the processor architecture decision.A more accessible and affordable alternative to proprietary architectures like ARM and MIPS, open-source instruction sets like RISC-V have changed the game when it comes to processor design. RISC-V is appealing to both academics and businesses because of its open-source nature, which gets rid of licensing fees and lowers development expenses. If space and power are paramount in an embedded system, the RRISC-V architecture is a great choice because to its small size, scalability, low power consumption, and cost effectiveness. RISC-V is an open-source ISA that was created at UC Berkeley in 2010. Its design goals were to be simple, modular, and extendable. A versatile framework for a broad variety of applications, from tiny embedded devices to high-performance supercomputers, it offers a mini-malistic base instruction set and the opportunity to build bespoke extensions. Rapid acceptance across sectors has been driven by RISC-V's versatility and open nature, which in turn has encouraged innovation and cooperation in processor design.

The functionality and interaction of processors with their respective assembly languages are defined by ISAs. Included in these essential components are instructions, registers, memory access techniques, mathematical operations, and data transfer. The basis for trustworthy hardware design is this standardization, which guarantees that processors consistently and properly execute machine code. The efficiency, simplicity, and low resource needs of the RV32I ISA—a 32-bit subset of the RISC-V architecture—make it an ideal choice for embedded systems, and this article details the design and development of an efficient processor based on this architecture. We have successfully synthesized the processor with all instructions verified by a Verilog test-bench. It was developed and implemented using Verilog. The literature overview is covered in Section II of this study, with an emphasis on the most recent advances in RISC-V architectures. The third section delves into the system design, with an emphasis on the processor's capabilities. In Section IV, we learn everything about the system's architecture, including detailed descriptions of each component. The Vivado-simulated system's tests and results are presented in Section V. The study is concluded with a summary of the main results in Section VI.

II. LITERATURE REVIEW

A. Barriga has detailed a design technique that makes it easier to implement processors based on the RISC-V instruction set architecture (ISA). It has three RRISC-V core design examples. The RV32I versions are used in all three implementations. A high-level state machine based on RISC-V is the first design. It goes into detail on how the processor's instruction set works. It seems like this design isn't made for hardware.

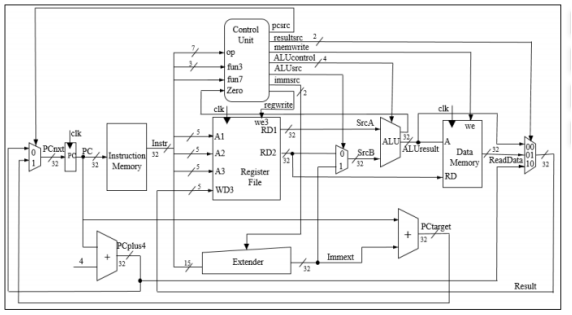

Fig. 1. System Architecture of Single Cycle RISC-V Processor

execution due to the synthesized circuit's low performance and excessive cost. Application, System, and Technique for the Treatment of Information is the second design. This CPU follows the principles of a Von Neumann design. The instructions are carried out in a sequential manner. The third design is called RISS-V, and it's associated with the University of Seville. It is designed by Harvard University. Through the use of pipeline, this design enables the parallelization of instruction execution. An Artix Xilinx field-programmable gate array (FPGA) device hosts the processors. By using an FPGA, Bit Manipulation Instruction on RISC-V Architecture was developed by V. Jain et al. [2].

With the help of two more RISC-V BMIs and the open-source RISC-V (RV32I) ISA, they were able to simplify the design of a fully synthesizable 32-bit processor bitRISC and put it into action on the intended device. Before being implemented on FPGA, the CPU was developed using Verilog HDL. The implemented single-cycle processor consisted of three parts: memory, the control unit, and the datapath unit. The Data Path Unit is the structural backbone of the processor and contains all of its architectural components. Decoding is done by the Control Unit. Recalling information required two parts.Instruction memory contains a set of mandatory instructions, whereas data memory acts as a storage component. The creation of a low-power, open-source implementation of a RISC-V processor and its transfer to a Field Programmable Gate Array system have been described in detail by L. Poli et al. [3]. Each component's implementation and test benches were created using conventional and

concis System Verilog. They used an abbreviated version of the RV64I add-on. A 32-bit RISC V processor with cryptographic accelerators implemented on an FPGA was detailed by D. T. Nguyen-Hoang et al. [5] using a mix of Verilog HDL and SpinalHDL, two hardware building languages.One benefit of this work is that it integrates cryptography cores with the 32-bit VexRiscV CPU core as a VexRiscV-based system on a chip. Therefore, this method worked well for Internet of Things (IoT) devices that were protected at the edge.

The MEIMAT (MEiji Microprocessor Architecture Design Tools) were created by L. Jiahui et al. [7]. The MEIMAT meta-instruction may express any instruction from any processor in two ways: semantically and functionally. They made sure that MEIMAT supported RISC-V and then enhanced it such that it supports RISC-V even more. They delineated the syntax of RV32I instructions and demonstrated how the matching MEIMAT instructions were expressed semantically and functionally. The power efficiency of ALU for RISC-V ISA has been reported by K. Puli and V. Pudi [8]. They brought out the fact that RISC architectures have become more well-known for their simplified instruction sets among energy-efficient designs. These sets helped improve energy efficiency by cutting down on power use and superfluous procedures. Concerning the RV32I version, they elaborated.

III. SYSTEM ARCHITECTURE

Figure 1 depicts a single-cycle processor, which is capable of executing a single instruction per clock cycle. To provide clarity and modularity, the design is implemented using Arty S7 Xilinx FPGA. Each component is arranged as an independent module. Each module's functioning is simulated and verified using a Verilog testbench. For integer operations requiring 32 bits of data, the RV32I instruction set architecture (ISA) is used. With the help of the Program Counter (PC), the Instruction Fetch Unit (IFU) retrieves the instruction from Instruction Memory, starting the process. Every time an instruction is executed, the PC increases by 4, barring any branching. After retrieving the instruction, the Control Unit decodes it and then produces the appropriate control signals. To perform arithmetic or branch operations, the 32-register Register File is queried for operands, and the Im mExt unit extends immediate values. By taking values from the registers or the immediate values as inputs, the Arithmetic Logic Unit (ALU) may execute calculations like adding, subtracting, or logical comparing.

The R-type, S-type, I-type, Branch, and Jump instruction types are all supported by the CPU according to the RV32I Instruction Set Architecture. Data Memory is accessed using the address determined by the ALU for load and store operations. Returns to the Register File are written whenever an ALU operation or memory access returns a result. The PC updates to the destination address computed using the immediate value in the event of branch and jump instructions, ensuring that execution flows smoothly. For fundamental RISC-V operations, this architecture strikes a good compromise between being simple and being functional. This solution is perfect for small-scale applications since it eliminates pipelining, which may lead to risks and complexity in more sophisticated systems. In addition, this project showcases the use of modular design concepts to process sor implementation, which lays a solid foundation for future improvements like pipelining or more instructions support. The adaptability and scalability made possible by the Verilog code's modular structure make it ideal for a wide range of computer applications.

IV. SYSTEM DESIGN

The RV32I ISA processor's hardware implements the RISC-V instruction set and comprises a number of critical components. The processor retrieves the next instruction from a certain position in memory, which is determined by the

Program Counter (PC). In most cases, the PC will increase by 4 after an instruction is executed to indicate the next consecutive instruction. In order to accurately redirect control flow, the PC is updated with the destination address indicated by the branch or jump instruction.

The processor is able to preserve the execution sequence and smoothly transition between instructions thanks to this technique. 2) Retention of Instructions: A read-only memory (ROM) is used to store the instructions that the processor needs to execute its tasks. It retrieves the matching instruction from memory and executes it based on the location given by the Program Counter (PC). It is possible to adjust the memory capacity according to the needs of various applications since it is parameterizable in the Verilog code. In keeping with the design of the RV32I instruction set, the breadth of each memory location is 32 bits.

The effective storing and retrieval of instructions is made possible by this structure, which allows the processor to run smoothly on diverse workloads. Thirdly, there's the Control Unit, which is like the brains of the processor; it's responsible for producing the necessary control signals for carrying out the instructions. These signals are determined using the opcode, funct3, and the fifth bit of funct7. Opcode (Instr[6:0]), funct3 (Instr[14:12]), and the fifth bit of funct7 (Instr[30]) are the most important fields in the RV32I instruction set.

These signals govern changes to the register file, regulate memory access, direct data flow inside the CPU, and choose suitable ALU operations. Figure 2 shows the syntax of the instruction for correct signal creation, which includes the opcode, funct3, and funct7. There are mainly two parts to the Control Unit: both the ALU Decoder and the Main Decoder. The ALU Decoder determines the specific ALU operation to be executed based on the kind of instruction, while the Main Decoder creates high-level control signals for instruction decoding and execution. 4) The Register File: Data, addresses, and calculation results are temporarily stored in 32 registers, each 32 bits wide, that make up the Register File. Instructions that need two source operands can't be executed without its two independent read ports, which allow for the simultaneous access of two registers. Furthermore, a RegWrite enable signal controls the one write port in the Register File, which allows you to update the contents of a register.

Data is written only when necessary, preserving the integrity of stored values, thanks to this signal. The RV32I architecture is able to execute instructions smoothly and efficiently because to this design's data handling capabilities. 5) ImmExt, or the Immediate Extension Unit: It is the job of the 32-bit extender to create extended values from the immediate field of the instruction. It generates sign-extended or zero-extended results according to the instruction type. In instructions that need to keep the immediate value's sign, such arithmetic or branch operations, sign extension is utilized; in cases where unsigned operations are needed, zero extension is used. This extender makes sure that the processor can handle instantaneous values from different sorts of instructions, and it also works with 32-bit operations.

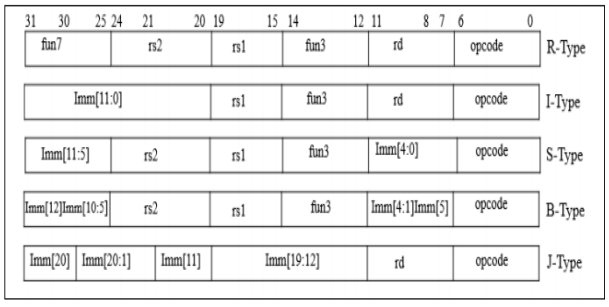

Fig. 2. Instruction Format of RV32I Instruction Set Architecture

6) ALU (Arithmetic Logic Unit):

The inputs given to it and the procedure dictated by the ALUControl signal determine the outputs

produced by the Arithmetic and Logic Unit (ALU). Basic operations like adding, subtracting, bitwise AND, OR, and comparisons are all defined by this signal. During the processor's execution phase, the ALU is crucial because it calculates values needed for arithmetic processing and for making decisions in control flow instructions.

The seventh component is the data memory, which handles read and write operations according to the instruction type and the MemWrite signal. Its size is 2048 x 32 bits. In order to execute a load instruction, the ALU calculates a memory address and then loads the data into the designated register. The MemWrite signal enables the write operation to the memory at the location provided by the ALU, which is necessary for store instructions. Instructions that require data transfer between memory and registers cannot be executed without this capability, which enables the processor to access and alter data stored in memory.

V. TESTS AND RESULTS

The hexadecimal instructions that correspond to a program's assembly-level code are stored in the Instruction Memory, which is an essential part of a processor. The processor is able to perform arithmetic (adding and subtracting), logical (AND and OR), and data transfer (load and store) operations on its registers in response to individual instructions stored in memory. For logical and mathematical instructions, the result is written back to the target registers so they may be used again. A number of different kinds of instructions may be executed by the processor. These include arithmetic, logical, load, store, and branch instructions. The correctness of the design is checked by developing a testbench in Verilog, which creates a simulated environment to test the behavior of the CPU. By running the code via this testbed, designers may find and fix bugs before releasing their hardware by making sure the CPU follows all commands and produces the intended outcomes.

TABLE I
DATA MEMORY LOCATIONS AND ITS VALUE

Data Memory	Values
0	0x30303030
4	0x20202020
8	0x40404040
12	0x00000000

In Table I, you can see the values of the first four spots where the data memory was populated with specified 32-bit values. In order to execute operations or load data into registers, these pre-stored 32-bit values are used during instruction execution. Each of the five sorts of instructions, together with their operation, is shown in Table II, and all thirty of them have been run and tested. By loading the word from memory location 0 into register t1, t1 became 0x30303030, the first instruction (32'h00002303, equivalent to lw t1, 0(x0)) accomplished this. A word from memory location 4 was loaded into register t2 by the second instruction (32'h00402383, equivalent to lw t2, 4(x0)), resulting in t2 = 0x20202020. The third instruction compared the values in registers t2 and t1, which corresponds to beq t2, t1, and 16. The program counter (PC) did not branch and the following instruction was executed sequentially since they were not equal. The value in were bitwise ORed by the fourth instruction at the new location (32'h00406e13, which corresponds to ori t3, x0, 4).

TABLE II
RISC-V INSTRUCTIONS WITH MACHINE CODE, OPERATION, AND

Instruction	Machine Code	Operation
lw t1,(x0)	32'h00002303	t1 = M[x0+32'h0]
lw t2,4(x0)	32'h00402383	t2 = M[x0+32'h4]
beq t2,t1,16	32'h00639463	if(t2 == t1) PC += 32'h16
ori t3,x0,4	32'h00406e13	t3 = 32'h4
and t4,t2,x0	32'h0003feb3	t4 = t2 & x0
sw t3,8(x9)	32'h01c4a423	M[x9+32'h8] = t3
jal t1,8	32'h0080036f	t1 = PC+4; PC += 32'h8

the instantaneous value 4 and the zero-based variable x0, we get t3 = 0x00000004. The value in t2 was ANDed with the value in x0 in the fifth instruction (32'h0003feb3, which corresponds to and t4, t2, x0), and the result was stored in t4. The sum of the values in x9 and the immediate offset 8 is stored in register t3 by the sixth instruction (32'h01c4a423, which corresponds to sw t3, 8(x9)). The seventh instruction (32'h0080036f, which corresponds to jal t1, 8) stores the return address in t1 and changes the program counter to PC + 8.

TABLE III
RESOURCE UTILIZATION

Resources	Utilization	Utilization %
Look-up Tables	1814	5.56
Flip-Flops	160	0.25
IO	34	16.15

In Table III, we can see that out of all the available Look-Up Tables (LUTs), 1814 (or 5.56 percent) were utilized by the single-cycle RV32I CPU that was developed in Verilog. Additionally, 160 Flip-Flops, or 0.25 percent of the total, were used up by the design. On top of that, out of all the IO resources, 34 (or 16.15 percent) were used. What this means is that the processor implementation made good use of the FPGA resources.

Figures 3–5 show the output of the single-cycle processor running lw, beq, and ori. Program Counter (PC), Instruction Memory (IM), Extender (E), and ALU (ALU) output is shown in Fig. 3, which shows their functions during execution. Data read and write operations are highlighted in Fig. 4, which is the result of the Data Memory and Register File. Figure 5 shows the Control Unit's output, which is utilized to govern the processor's operations and shows how control signals are created.

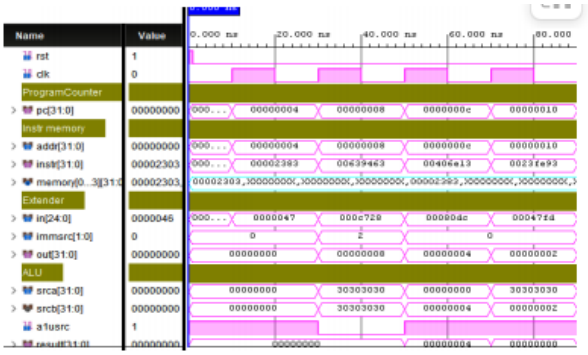

Fig. 3. Output of single cycle processor(PC, Instruction memory, Extender and ALU)

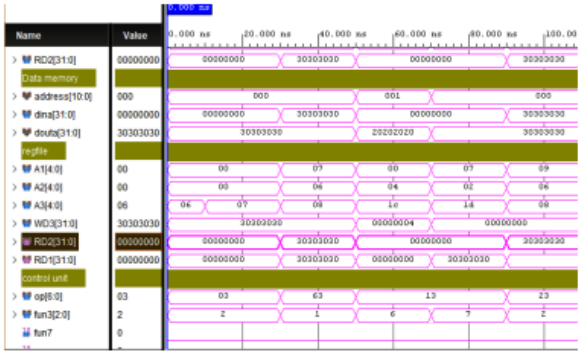

Fig. 4. Output of single cycle processor(Data memory and Register file)

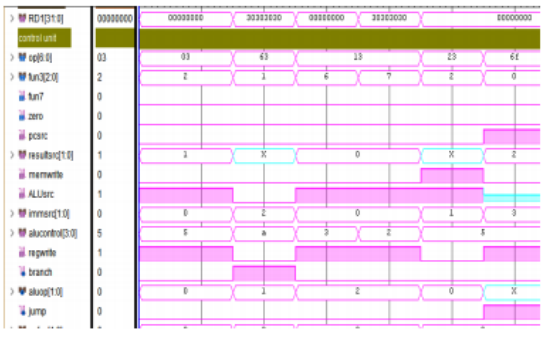

Fig. 5. Output of single cycle processor(Control Unit)

TABLE IV
COMPARISON OF LUTS AND FFS WITH EXISTING WORKS

Year	Paper	LUTs	FFs
2020	[2]	2312	2035
2021	[3]	322	229
2022	[5]	8697	7577
2024	[10]	4210	2270
This work		**1814**	**160**

Table IV compares the LUTs and FFs used in different RISC-V processor architectures that are based on FPGAs. With only 160 FFs and 1814 LUTs, this work is far more efficient than its predecessors.

VI. CONCLUSIONS

Through the use of Test bench and the Vivado software environment, it was successfully proved that a RISC-V processor capable of executing 30 instructions may be implemented. The execution of all 30 instructions, including those for load and store (lw, sw), branch and jump (beq, jal), and arithmetic and logical (andi, ori), was flawless, proving that the RV32I instruction set architecture was functional. The efficacy and aptitude to handle different activities were validated by the successful execution of these 30 orders. This study highlights the capabilities of Vivado for synthesis, simulation, and thorough testing of CPU designs, as well as the adaptability of Verilog for hardware description. The research also shows that the processor can effectively handle fundamental RISC-V operations.

REFERENCES

[1] A. Barriga, **"RISC-V processors design: a methodology for cores development,"** 2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS), Segovia, Spain, 2020

[2] V. Jain, A. Sharma and E. A. Bezerra, **"Implementation and Extension of Bit Manipulation Instruction on RISC-V Architecture using FPGA,"** 2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), Gwalior, India, 2020

[3] L. Poli, S. Saha, X. Zhai and K. D. Mcdonald-Maier, **"Design and Implementation of a RISC V Processor on FPGA,"** 2021 17th International Conference on Mobility, Sensing and Networking (MSN), Exeter, United Kingdom, 2021

[4] J. -Y. Lai, C. -A. Chen, S. -L. Chen and C. -Y. Su, "**Implement 32-bit RISC-V Architecture Processor using Verilog HDL**," 2021 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), Hualien City, Taiwan, 2021

[5] D. T. Nguyen-Hoang, K. -M. Ma, D. -L. Le, H. -H. Thai, T. -B. -T. Cao and D. -H. Le, **"Implementation of a 32-Bit RISC-V Processor with Cryptography Accelerators on FPGA and ASIC,"** 2022 IEEE Ninth International Conference on Communications and Electronics (ICCE), Nha Trang, Vietnam, 2022

[6] J. Saussereau, C. Jego, C. Leroux and J. -B. Begueret, **"Design and Implementation of a RISC-V core with a Flexible Pipeline for Design Space Exploration,"** 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Istanbul, Turkiye, 2023

[7] L. Jiahui, T. Morimoto, T. Ogita, R. Kawamata, W. Ziming and T. Tsutsumi, **"Microprocessor Instruction Design Tool for RISC-V Architecture,"** 2023 22nd International Symposium on Communications and Information Technologies (ISCIT), Sydney, Australia, 2023

[8] K. Puli and V. Pudi, **"Design of Low Power ALU for RISC-VISA,"** 2023 IEEE International Symposium on Smart Electronic Systems (iSES), Ahmedabad, India, 2023

[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

[10] M. Wygrzvwalski, P. Skrzypiec and R. Szczygiel, **"Hardware Acceleration Method Using RISC-V Core with No ISA Extensions,"** 2024 31st International Conference on Mixed

Design of Integrated Circuits and System (MIXDES), Gdansk, Poland, 2024