Project Report On

Design, Implementation, and Verification of 32-bit RISC V processor

Submitted in Partial Fulfillment for the award of

Post Graduate Diploma in VLSI Design

(PG-DVLSI)

from

C-DAC, ACTS (Pune)

Guided by:
Mr. Suyash Jain

Presented by:

Mr.Pinaki Mohanty	**PRN:230940133021**
Mr.Saksham Pathak	**PRN:230940133025**
Mr.Saravanan L	**PRN:230940133027**
Mr.Shivam R Gupta	**PRN:230940133030**
Mr.Tushar Ganotra	**PRN:230940133031**

Centre for Development of Advanced Computing (C-DAC), Pune

ACKNOWLEDGEMENT

This project “Design, Implementation and Verification of 32-bit RISC V processor” was a great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We are very glad to mention the name of *Mr.Suyash Jain* for their valuable guidance to work on this project. Their guidance and support helped me to overcome various obstacles and intricacies during the course of project work.

Our heartfelt thanks goes to *Ms. Namratha and Ms.Sajida shaikh* (Course Coordinator, *PG-DVLSI*) who gave all the required support and kind coordination to provide all the necessities like required hardware, internet facility, and extra Lab hours to complete the project and throughout the course up to the last day here in C-DAC ACTS, Pune.

TABLE OF CONTENTS

ABSTRACT

RISC-V is an open-source reduced instruction set-based instruction set architecture, and the processor based on this architecture can be modified accordingly.

The base integer instruction extension supports the operating system environment and is also suitable for embedded systems. It is a 32-bit instruction extension and is defined as RV32I.

In this project, we propose a 32-bit integer instruction-based RISC-V processor core.

The proposed core has a five-stage , including the optimized arithmetic and logic unit. In , the execution of instructions is broken down into multiple stages, and each stage is executed concurrently with the stages of other instructions. This allows for improved throughput and better utilization of hardware resources.

It enables the systematic storage and execution of instructions.

A load–store architecture (or a register–register architecture) is an instruction set architecture that divides instructions into two categories: memory access (load and store between memory and registers) and ALU operations (which only occur between registers).

Notable features of the RISC-V ISA include: instruction bit field locations chosen to simplify the use of multiplexers in a CPU, a design that is architecturally neutral and a fixed location for the sign bit of immediate values to speed up sign extension.

The instruction set is designed for a wide range of uses. The base instruction set has a fixed length of 32-bit naturally aligned instructions, and the ISA supports variable length extensions where each instruction can be any number of 16-bit parcels in length. Extensions support small embedded systems, personal computers, supercomputers with vector processors, and warehouse-scale parallel computers.

The project is implemented in Vivado, a hardware design software from Xilinx. The project is verified using a testbench.

The RISC V processor is implemented using Verilog, and the resource utilization is verified for FPGA.

INTRODUCTION

The origins of RISC-V can be traced back to the University of California, Berkeley, where it was initially developed as a research project in 2010. The project aimed to create a new, open-source ISA that would address the limitations of existing proprietary ISAs and provide a foundation for future processor designs.

The first version of the RISC-V ISA, known as the "RV32I" base integer instruction set, was released in 2011. This initial release focused on simplicity and efficiency, adhering to the principles of reduced instruction set computing (RISC). Over the years, the RISC-V ISA has evolved through several iterations, with the addition of new extensions and features to enhance its capabilities and address a broader range of applications.

The evolution of RISC-V has been driven by several factors, including the need for greater customization and flexibility in processor design, the desire to reduce the reliance on proprietary ISAs, and the growing demand for energy-efficient and cost-effective computing solutions. By providing an open, modular, and extensible ISA, RISC-V has enabled a new era of innovation in processor design and has the potential to reshape the lanscape of the semiconductor industry.

RELEVANCE

The evolution of RISC-V has been driven by several factors, including the need for greater customization and flexibility in processor design, the desire to reduce the reliance on proprietary ISAs, and the growing demand for energy-efficient and cost-effective computing solutions. By providing an open, modular, and extensible ISA, RISC-V has enabled a new era of innovation in processor design and has the potential to reshape the landscape of the semiconductor industry.

The rise of RISC-V coincides with a couple of other events in the industry. The first is the slowing of Moore's Law, meaning that increases in total processing power no longer comes along with each new fabrication node. The second is the meteoric rise in machine learning, demanding massive increases in processing power.

Electronic products are defined by their functionality, a lot of which is defined by software, which is running on processors. Everything needs some form of machine learning today. It doesn't matter if we are talking about your phone, taking better pictures, whatever it is, there are huge amounts of computing needed. And what people realized is that they needed lots of processors. They needed their own fabrics of processors. You need to configure them the way you want. Off-the-shelf technologies don't help you. So there's a change in the electronic product marketplace saying, We need freedom to architect chips, and freedom to architect the processors and the fabrics of processors that live in these chips.

RISC-V is creating breakthroughs is multiple areas, and the reasons for success in each are different. To understand this, it's necessary to separate out various aspects of RISC-V's success. First is the architecture itself. Second are the plethora of open-source implementations of the architecture that are being made available. A third area is the support cores that are becoming available to surround the processor core. And finally, there are the tools necessary to help with the implementation and verification of a RISC-V processor.

COMPARISON RISC VS CISC:

Reduced Instruction Set Computing (RISC)

At the core of the RISC-V architecture is the concept of reduced instruction set computing (RISC). RISC is a processor design philosophy that emphasizes simplicity and efficiency by using a small set of simple and general-purpose instructions. This contrasts with complex instruction set computing (CISC), which employs a larger set of more complex instructions that can perform multiple operations in a single instruction.

RISC architectures prioritize simplicity and execute one instruction per clock cycle, resulting in streamlined designs and efficient decoding. CISC architectures, on the other hand, employ complex instructions capable of performing multiple actions but may require several clock cycles for execution. Both the CPUs aim to enhance CPU performance.

Aspect	RISC	CISC
Instructions Per Cycle	Small and fixed length	Large and variable length
Instruction Complexity	Simple and standardised	Complex and versatile
Instruction Execution	Single clock cycle	Several clock cycles
RAM Usage	Heavy use of RAM	More efficient use of RAM
Memory	Increased memory usage to store instructions	Memory efficient coding
Cost	Higher Cost	Cheaper than RISC

The RISC approach has several advantages over CISC:

1. **Simplifies Hardware Implementation:** It simplifies the hardware implementation of the processor, as fewer instructions need to be decoded and executed. This can lead to faster execution times and lower power consumption.

2. **Higher Instruction Level Parallelism:** RISC processors typically have a higher instruction-level parallelism, allowing them to execute multiple instructions simultaneously, which can further improve performance.
3. **Simplicity:** The simplicity of the RISC instruction set makes it easier to develop compilers and other software tools that can generate efficient code for the processor.

RISC-V adheres to the RISC philosophy by providing a minimal set of instructions that can be combined to perform complex operations. This simplicity allows for a more streamlined processor design, resulting in improved performance and energy efficiency. Additionally, the RISC-V instruction set is designed to be easily extensible, enabling the addition of new instructions and features as needed to address specific application requirements. This combination of simplicity and extensibility makes RISC-V a versatile and powerful foundation for processor design.

Modularity and Extensibility

Another key design principle of the RISC-V architecture is its modularity and extensibility.

Modularity refers to the organization of the ISA into separate, independent components that can be combined in various ways to create a customized processor. Extensibility, on the other hand, refers to the ability to add new instructions, features, or extensions to the ISA without disrupting existing functionality.

The RISC-V ISA is organized into a base integer instruction set and a set of optional extensions. The base integer instruction set provides the core functionality required for general-purpose computing, while the extensions add specialized capabilities for specific applications or domains. This modular approach allows designers to select the features they need for their particular use case, resulting in a processor that is optimized for performance, power consumption, or other design goals.

The extensibility of RISC-V is achieved through a well-defined extension mechanism that enables the addition of new instructions and features without affecting the compatibility of existing software. This allows the ISA to evolve over time and adapt to new technologies and application requirements. The RISC-V community has developed a range of standard extensions, such as floating-point arithmetic, vector processing, and cryptographic operations, which can be incorporated into processor designs as needed.

Let's say a company is designing a processor for embedded systems used in digital signal processing (DSP) applications. DSP tasks often involve complex mathematical operations like vector multiplication. Instead of relying solely on the base RISC-V instructions, the company can create a custom instruction extension for vector multiplication. These custom extensions are like individual modules designed to enhance the processor's capabilities for DSP workloads. Extensibility is the broader ability of the RISC-V architecture to evolve and adapt to new requirements, such as supporting DSP operations (here), while maintaining its core design principles.

The combination of modularity and extensibility in RISC-V provides a high degree of flexibility and customization, enabling the development of processors that are tailored to specific applications and use cases. This, in turn, can lead to improved performance, energy efficiency, and cost-effectiveness, making RISC-V a compelling choice for a wide range of computing platforms and devices.

LITERATURE SURVEY

Proprietary ISAs were tightly controlled by specific companies, limiting access to their architecture and imposing licensing fees. This lack of openness hindered innovation, discouraged competition, and made it challenging for smaller companies or academic institutions to experiment and develop custom processors. This led to the rise of RISC-V.

The origins of RISC-V can be traced back to the University of California, Berkeley, where it was initially developed as a research project in 2010. The project aimed to create a new, open-source ISA that would address the limitations of existing proprietary ISAs and provide a foundation for future processor designs. The RISC-V project was led by computer scientists Krste Asanović, Yunsup Lee, and Andrew Waterman, who were inspired by the success of open-source software and sought to bring similar benefits to the hardware domain.

The first version of the RISC-V ISA, known as the "RV32I" base integer instruction set, was released in 2011. This initial release focused on simplicity and efficiency, adhering to the principles of reduced instruction set computing (RISC). Over the years, the RISC-V ISA has evolved through several iterations, with the addition of new extensions and features to enhance its capabilities and address a broader range of applications.

In 2015, the RISC-V Foundation was established to promote the adoption and standardisation of the RISC-V ISA. The foundation brought together industry leaders, academic institutions, and individual contributors to collaborate on the development and dissemination of RISC-V technology. Since its inception, the RISC-V Foundation has grown to include over 200 member organizations, and the RISC-V ISA has been adopted by numerous companies for various applications, from microcontrollers and embedded systems to high-performance computing and data centre processors.

Before RISC-V, there were several RISC (Reduced Instruction Set Computer) processors in the market. Notable examples include MIPS, SPARC, and PowerPC. These architectures were considered efficient and had their applications, but they often came with licensing costs and restricted access to their inner workings.

The evolution of RISC-V has been driven by several factors, including the need for greater customization and flexibility in processor design, the desire to reduce the reliance on proprietary ISAs, and the growing demand for energy-efficient and cost-effective

computing solutions. By providing an open, modular, and extensible ISA, RISC-V has enabled a new era of innovation in processor design and has the potential to reshape the landscape of the semiconductor industry.

RISC-V (pronounced "risk-five") is an open-source instruction set architecture (ISA) based on Reduced Instruction Set Computing (RISC) principles. It offers several advantages over other ISAs, contributing to its increasing popularity and adoption in various domains. Here are some of the key advantages of RISC-V:

1. Openness and Licensing: One of the most significant advantages of RISC-V is its open nature. The ISA specifications are freely available, allowing anyone to implement RISC-V processors without needing to pay royalties or obtain licenses. This openness promotes innovation, collaboration, and the development of diverse implementations across different industries and applications.

2. Modular Design: RISC-V features a modular design that allows for customization and specialization to meet specific requirements. The ISA is divided into multiple standard extensions, such as integer, floating-point, vector, and cryptographic extensions. This modularity enables efficient implementations tailored for various use cases, from embedded systems to high-performance computing.

3. Scalability: RISC-V supports scalability across a wide range of devices, from microcontrollers and IoT devices to high-performance servers and supercomputers. Its modular design and customizable features allow for implementations with varying complexity, performance levels, and power requirements. This scalability makes RISC-V suitable for diverse applications and markets.

4. Performance and Efficiency: RISC-V's RISC-based architecture prioritizes simplicity, efficiency, and performance. By reducing the complexity of instructions and emphasizing a streamlined instruction set, RISC-V processors can achieve high performance while consuming less power and area compared to more complex architectures. This makes RISC-V particularly well-suited for embedded and energy-efficient computing applications.

5. Community Support and Ecosystem: RISC-V has garnered significant momentum and support from a vibrant and growing community of developers, researchers, and industry stakeholders. This ecosystem includes open-source software tools, development boards,

simulators, compilers, and libraries tailored for RISC-V architecture. The active community contributes to the evolution, improvement, and standardization of the RISC-V ecosystem, fostering innovation and collaboration.

6. Portability and Longevity: RISC-V's open nature and standardized ISA specifications contribute to its portability and longevity. Developers can write software targeting the RISC-V architecture with confidence that it will remain compatible across different implementations and vendors. This portability reduces vendor lock-in and enables seamless migration between RISC-V-based platforms, enhancing flexibility and long-term sustainability.

7. Security: RISC-V provides features and extensions to support modern security requirements, such as hardware-based memory protection, privilege levels, and cryptographic instructions. These security features help mitigate common security threats, enhance system robustness, and facilitate the development of secure and trustworthy computing platforms.

Overall, RISC-V offers a compelling combination of openness, modularity, scalability, performance, community support, and security, making it an attractive choice for a wide range of applications across various industries. Its advantages continue to drive its adoption and influence the future of computing architecture.

RISC-V is an open-source reduced instruction set-based instruction set architecture, and the processor based on this architecture can be modified accordingly. The base integer instruction extension supports the operating system environment and is also suitable for embedded systems. It is a 32-bit instruction extension and is defined as RV32I. In this project, we propose a 32-bit integer instruction-based RISC-V processor core. The proposed core has a five-stage , including the optimized arithmetic and logic unit

The processor is implemented using Verilog HDL, and the resource utilization is verified for FPGA.

AIM OF PROJECT

The project aims to design, implement and verify RISC V processor which has 32 bit instruction and that instruction passes through five stages respectively fetch stage (IF), instruction decode stage (ID), execute stage (EX), memory access stage (MEM), and write back stage (WB).

The top module has instance of program counter,instruction memory,ALU,control unit,memory unit,write back written in verilog and to implement it on an FPGA (Field Programmable Gate Array) device. By doing so, it will demonstrate how 32 bit instructions are implemented in hardware.

The primary aim of implementing a RISC-V processor is to improve performance and efficiency by executing instructions concurrently in multiple stages of the .processor

SCOPE AND OBJECTIVE

The project Design, Verification and Implementation of RISC V PROCESSOR is aimed to improve performance and efficiency by executing instructions concurrently in multiple stages of the RISC-V offers a unique set of features that allow users to customize and optimize both software and hardware for specific use cases, resulting in faster development cycles and better design tradeoffs for performance, power and area. RISC-V serves as an instruction set architecture (ISA) that provides the foundation for designing processors. Its open and modular design allows for the creation of a wide range of processors, from simple microcontrollers to high-performance server chips. The implementation of a 5-stage d 32-bit High Performance RISC V is presented in this project.

This processor was created with the goal of enhancing the processor's overall speed by performing a small set of instructions. Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB) are the five steps of Instruction Memory, Data Memory, ALU, Registers, and other modules are employed.

Verilog is used to create the design.The primary aim of implementing a RISC-V processor is to improve performance and efficiency by executing instructions concurrently in multiple stages of the.

1. **Increased Throughput:** By breaking down the instruction execution process into multiple stages and allowing multiple instructions to be processed simultaneously, the aims to increase the overall throughput of the processor. This enables faster program execution by overlapping the execution of different instructions.

2. **Reduced Latency:** The aims to reduce the latency of individual instructions by allowing them to progress through the stages concurrently. This leads to faster execution times for programs, as instructions can be processed more quickly.

3. **Improved Resource Utilization:** The architecture aims to improve the utilization of hardware resources by keeping different parts of the processor busy with different instructions at the same time. This leads to more efficient use of functional units, registers, and other processor components, maximizing overall performance.

4. **Optimized Instruction Cycle Time:** By executing instructions in parallel stages, the aims to reduce the instruction cycle time, enabling faster clock speeds and

higher instruction execution rates. This results in improved overall performance of the processor.

5. **Enhanced Branch Prediction:** processors often incorporate branch prediction techniques to mitigate the performance impact of branch instructions. The aims to predict the outcome of branches early in the , allowing the processor to continue fetching and executing instructions speculatively, reducing stalls and improving efficiency.

6. **Flexibility and Adaptability:** The architecture aims to provide flexibility and adaptability to various applications and workloads. Designers can adjust the depth and structure of the to optimize performance for specific use cases, balancing factors such as latency, throughput, and power consumption.

7. **Scalability:** RISC-V processors aim to support scalability, enabling the efficient implementation of processors with varying levels of complexity and performance. This scalability makes suitable for a wide range of applications, from low-power embedded systems to high-performance computing environments.

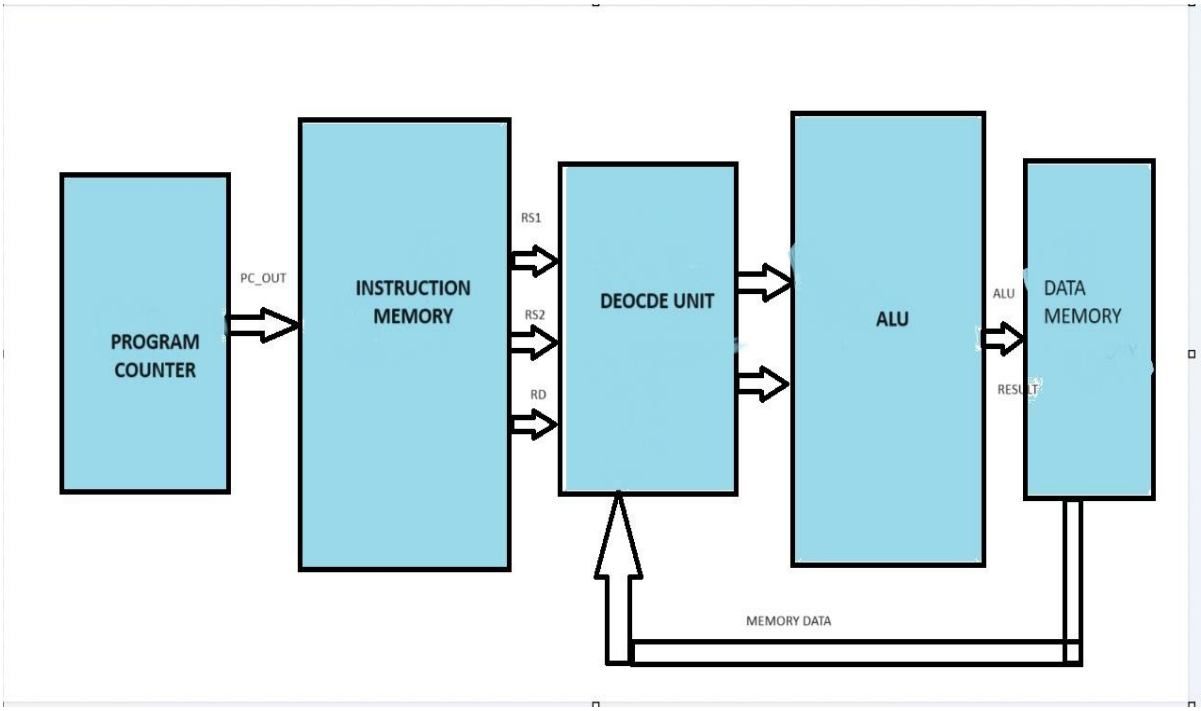
Overall, the aim of a RISC-V processor is to achieve improved performance, efficiency, and flexibility compared to non-d architectures, meeting the demands of modern computing applications.

THEORETICAL DESCRIPTION OF PROJECT

The five stage RISC

1. CONSTRUCTION OF THE PROCESSOR

Basic five-stage in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).



RISC V Architecture

Instruction Fetch Unit: The first stage in the is the Instruction Fetch. Instructions are fetched from the memory and the Instruction Pointer (IP) is updated. The function of the instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction as shown in Figure. This stage is where a program counter will pull the next instruction from the correct location in program memory. In addition the program counter will updated with either the next instruction location sequentially, or the instruction location as determined by a branch.

The instruction fetch stage is also responsible for reading the instruction memory and sending the current instruction to the next stage in the , or a stall if a branch has been detected in order to avoid incorrect execution. The instruction fetch unit contains the following logic elements that are implemented in Verilog: 32-bit program counter (PC) register, an adder to increment the PC by four, the instruction memory, a multiplexor, and an AND gate used to select the value of the next PC. Program counter and instruction memory are the two important blocks of Instructions Fetch Unit. Program counters (PC): It is an 32 bit device that is connected to the data bus and the address bus. It will hold its value unless told to do something. If the I/P is kept high the device will count, i.e. it will increment by 4.

Instruction memory (IM): During the Instruction Fetch stage, a 32-bit instruction is fetched from the memory. The PC predictor sends the Program Counter (PC) to the Instruction memory to read the current Instruction. At the same time, the PC predictor predicts the address of the next instruction by incrementing the PC by 4.

INSTRUCTION DECODE STAGE:

Instruction registers (IR): An instruction register (IR) is the part of control unit that stores the instruction currently being executed or decoded. In simple processors each instruction to be executed is loaded into the instruction register which holds it while it is decoded, prepared and ultimately executed, which can take several steps. RISC processors use a of instruction registers where each stage of the does part Of the decoding, preparation or execution and then passes it to the next stage for its step. Modern processors can even do some of the steps of out of order as decoding on several instructions is done in parallel. Decoding the opcode in the instruction register includes determining the instruction, where

its operands are in memory, retrieving the operands from memory, allocating processor resources to execute the command. The output of IR is available to control circuits which generate the timing signals that controls the various processing elements involved in executing the instruction.

Instruction Decode Unit The Instruction Decode stage is the second stage in the Branch targets will be calculated here and the Register File, the dual-port memory containing the register values, resides in this stage.

The, reside here. Their function is to detect if the register to be fetched in this stage is written to in a later stage. In that case the data is forward to This stage and the data hazard is solved. This stage is where the control unit determines what values the control lines must be set to depending on the instruction. In addition, hazard detection is implemented in this stage, and all necessary values are fetched from the register banks. The Decode Stage is the stage of the CPU's where the fetched instruction is decoded, and values are fetched from the register bank. It is responsible for mapping the different sections of the instruction into their proper representations (based on R or I type instructions). The Decode stage consists of the Control unit, the Hazard Detection Unit, the Sign Extender, and the Register bank, and is responsible for connecting all of these components together. It splits the instruction into its various parts and feeds them to the corresponding components. Registers are fed to the register bank, the immediate section is fed to the sign extender, and the ALU opcode and function codes are sent to the control unit. The outputs of these corresponding components are then clocked and stored for the next stage The Control unit takes the given Opcode, as well as the function code from the instruction, and translates it to the individual instruction control lines needed by the three remaining stages. This is accomplished via a large case statement.

Register files (RF): During the decode stage, the two register Rs & Rd are identified within the instruction, and the two registers are read from the register file. In this design, the register file had 32 entries. At the same time the register file was read, instruction issue logic in this stage determined if the was ready to execute the instruction in this stage. If not, the issue logic would cause both the Instruction Fetch stage and the Decode stage to stall. If the instruction decoded was a branch, the target address of the branch was computed in parallel with reading the register file. The branch condition is computed after the register file is read, and if the branch is taken ; the PC predictor in the first stage is assigned the branch target, rather than the incremented PC that has been computed.

Control unit: The control unit of processor examines the instruction opcode bits [6 – 0] and decodes the instruction to generate nine control signals to be used in the additional modules. The Reg Dst control signal determines which register is written to the register file. The Branch control signal is used to select the branch address to be sent to the PC. The MemRead control signal is asserted during a load instruction when the data memory is read to load a register with its memory contents. The MemtoReg control signal determines if the ALU result or the data memory output is written to the register file. The ALUOp control signals determine the function the ALU performs. (E.g. and, or, add, sbu, slt) The MemWrite control signal is asserted when during a store instruction when a registers value is stored in the data memory.

The ALUSrc control signal determines if the ALU second operand comes from the register file or the sign extend. The RegWrite control signal is asserted when the register file needs to be written.

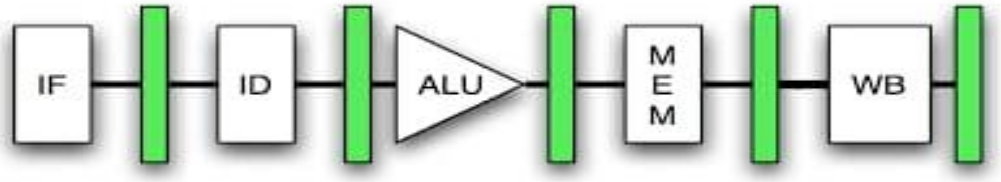
Execution Unit : The third stage in the is where the arithmetic- and logic-instructions will be executed. All instructions are executed with 32- bit operands and the result is a 32-bit word. The execution unit of theprocessor contains the arithmetic logic unit (ALU) which performs the operation determined by the ALUop signal. The branch address is calculated by adding the PC+4 to the sign extended immediate field shifted left 2 bits by a separate adder. The logic elements to be implemented in VERILOG.

ALU unit: The arithmetic/logic unit (ALU) executes all arithmetic and logical operations. The arithmetic/logic unit can perform four kinds of arithmetic operations, or mathematical calculations: addition, subtraction, multiplication, and division. As its name implies, the arithmetic/logic unit also performs logical operations. A logical operation is usually a comparison. The unit can compare numbers, letters, or special characters. The computer can then take action based on the result of the comparison.

Memory Access unit: The memory access stage is the fourth stage of . This is where load and store instructions Will access data memory. During this stage, single cycle latency instructions simply have their results forwarded to the next stage. This forwarding ensures that both single and two cycle instructions always write their results in the same stage of the , so that just one write port to the register file can be used, and it is always Available. If the instruction is a load, the data is read from the data memory

Data Memory Unit (DM): The data memory unit is only accessed by the load and store instructions. The load instruction asserts the MemRead signal and uses the ALU Result value as an address to index the data memory. The read output data is then subsequently written into the register file. A store instruction asserts the MemWrite signal and writes the data value previously read from a register into the computed memory address. The VERILOG implementation of the data memory was described earlier.

Write back unit: During this stage, single cycle write their results into the register file. In the write back stage, the result of the executed instruction is written back to the destination register. This involves updating the value stored in the register file with the computed result. In a non-pipeline processor, each instruction is executed sequentially, and the write back stage occurs immediately after the execution of each instruction. Unlike a pipelined processor where multiple instructions are in various stages of execution simultaneously, in a non-pipeline processor, only one instruction is processed at a time, from fetch to write back.Overall, the write back stage ensures that the results of executed instructions are properly stored in the register file, maintaining the correct state of the processor for subsequent instructions.



RISC V INSTRUCTION CYCLE

IMPLEMENTATION OF RISC V PROCESSOR

The RISC single-cycle processor performs the tasks of instruction fetch, instruction decode, execute, memory access and write-back all in one clock cycle. First the PC value is used as an address to index the instruction memory which supplies a 32-bit value of the next instruction to be executed. This instruction is then divided into the different fields.The instructions opcode field bits [6-0] are sent to a control unit to determine the type of instruction to execute. The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, thus decoding the instruction. The instruction register address fields rs1 bits [24 - 20], rs2 bits [19 - 15], and rd bits [11-7] are used to address the register file. The register file supports two independent register reads and one register write in one clock cycle. The register file reads in the requested addresses and outputs the data values contained in these registers. These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address(e.g. load or store), compute an arithmetic result (e.g. add, sub), or perform a compare (e.g. branch). If the instruction decoded is arithmetic, the ALU result must be written to a register. If the instruction decoded is a load or a store, the ALU result is then used to address the data memory. The final step writes the ALU result or memory value back to the register file.

Once the RISC single-cycle VERILOG implementation is completed, our next task is to the RISC processor. , a standard feature in RISC processors, is a technique used to improve both clock speed and overall performance. allows a processor to work on different steps of the instruction at the same time, thus more instruction can be executed in a shorter period of time. For example in the VERILOG RISC single-cycle implementation, the data path is divided into different modules, where each module must wait for the previous one to finish before it can execute, thereby completing one instruction in one long clock cycle. When the RISC processor is d, during a single clock cycle each one of those modules or stages is in use at exactly the same time executing on different instructions in parallel.

A Reduced Instruction Set computer is a microprocessor that had been designed to perform a small set of instructions, with the aim of increasing the overall speed of the processor .The RISC concept proved that 20% of instruction did 80% of the work .The RISC architecture follows the philosophy that one instruction executes in one clock cycle

RISC-V Instruction Set

The RISC-V instruction set is a collection of instructions that define the operations a RISC-V processor can perform. These instructions are designed to be simple, efficient, and easily extensible, allowing for a high degree of customization and optimization. The instruction set is organized into a base integer instruction set and a set of optional extensions, which provide specialized functionality for specific applications or domains.

32-bit RISC-V instruction formats

Format	Bit																													
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Register/register	funct7							rs2					rs1					funct3			rd					opcode						
Immediate	imm[11:0]												rs1					funct3			rd					opcode						
Branch	[12]	imm[10:5]						rs2					rs1					funct3			imm[4:1]				[11]	opcode						

RV32I INSTRUCTION SET

The base integer instruction set, also known as the "RV32I" instruction set, depending on the address space size, provides the core functionality required for general-purpose computing. It includes instructions for arithmetic, logical, and control operations, as well as memory access and manipulation. The base integer instruction set is designed to be minimal and efficient, adhering to the principles of reduced instruction set computing (RISC).

RISC-V instructions are encoded using a fixed-length 32-bit format, which simplifies decoding and execution. The instruction formats are R-type , I-type,B-type. Each format serves a specific purpose and has a unique encoding structure:

- **R-type instructions:** Used for register-to-register operations, such as arithmetic and logical operations. They include three register operands: two source registers and one destination register. Eg:- add *(Add 2 registers and store results in another)*
- **I-type instructions:** Used for immediate operations, such as arithmetic and logical operations with an immediate value. They include two register operands and a 12-bit immediate value. Eg:- li *(Load immediate value)*

- **B-type instructions:** Used for conditional branch operations, which transfer control to a different instruction based on a condition. They include two register operands and a 12-bit immediate value for the branch target address. Eg:- beq *(compare and label)*

RISC-V Register File

The RISC-V register file is a key component of the RISC-V architecture, providing a set of storage locations for holding data during the execution of instructions. The register file is organized into a set of integer registers and floating-point registers, depending on the extensions implemented in the processor. Registers play a crucial role in the RISC-V architecture, as they enable fast access to data and help improve the performance and efficiency of the processor.

R-type RV32I Instruction Format

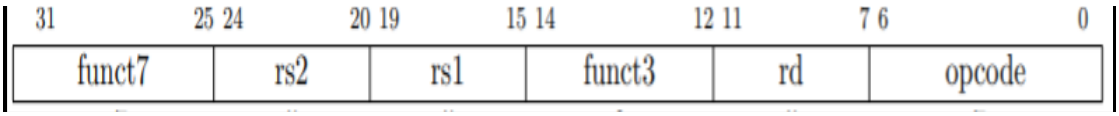


Figure 3.1 depicts the Register-type RV32I ISA V 2.0. It has a total of six fields. Opcode width is 7 bits, which is used to indicate the type of instruction. Source registers (rs1, rs2) and destination register (rd) are indicated by five-bit fields. The function field is of a total of 10 bits, which is used for identifying the type of operation to be performed. The instructionswhich are supported by this format are:

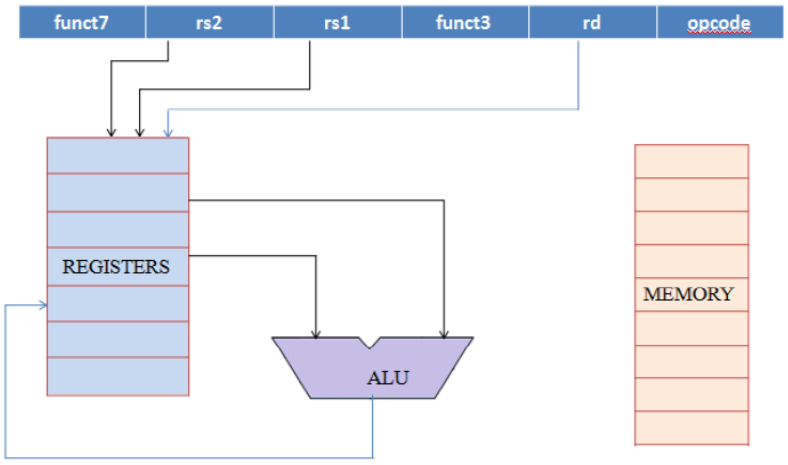
- ADD -Addition
- SUB -Subtraction
- XOR –Xor operation
- OR –or operation
- AND –and operation

R-TYPE INSTRUCTION

NAME	TYPE	OPCODE	FUNCT3	FUNCT7	DESCRIPTION
ADD	R	0110011	0X0	0X00	rd=rs1 + rs2
SUB	R	0110011	0X0	0X20	rd=rs1 - rs2
XOR	R	0110011	0X4	0x00	rd=rs1 ^ rs2
OR	R	0110011	0X6	0x00	rd=rs1 | rs2
AND	R	0110011	0X7	0x00	rd=rs1 & rs2

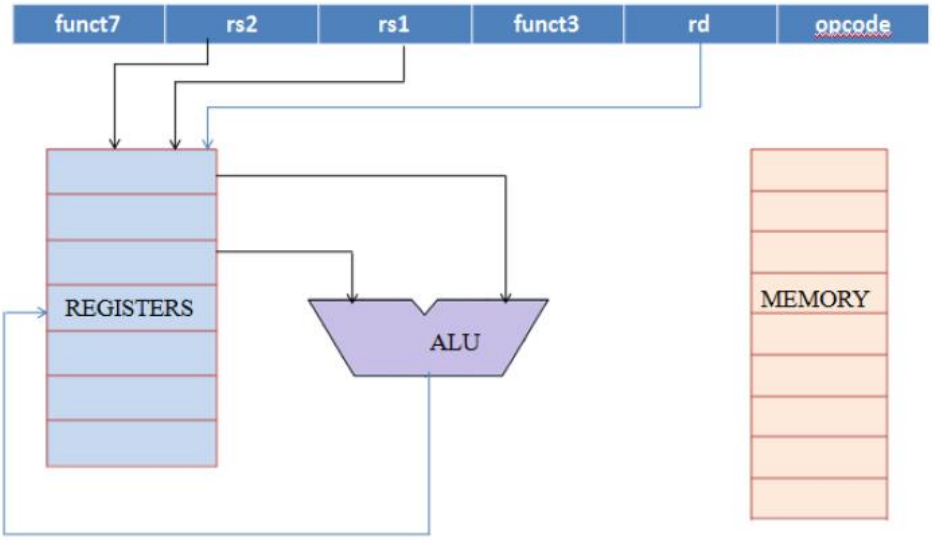
EXECUTION OF OPERATIONS IN R-FORMAT

ADD (Addition)



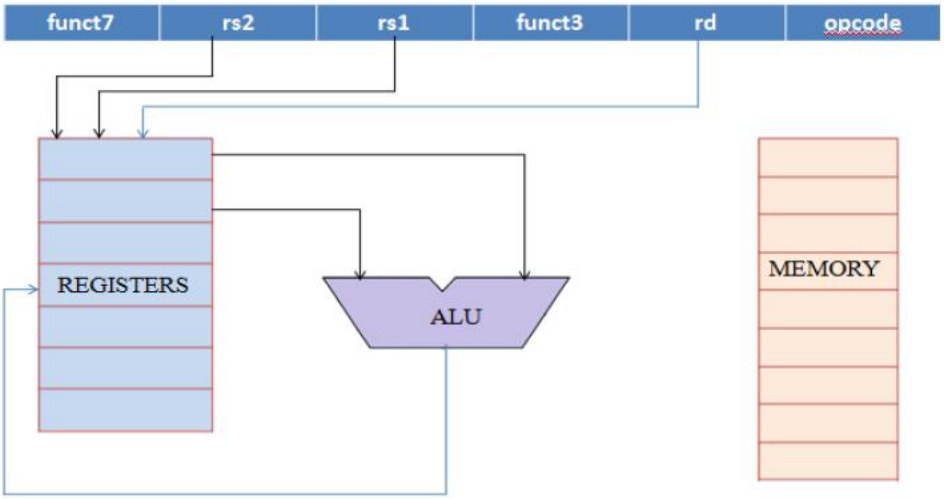
funct7=0000000;funct3=000;opcode =0110011

SUB (Subtraction) R-FORMAT



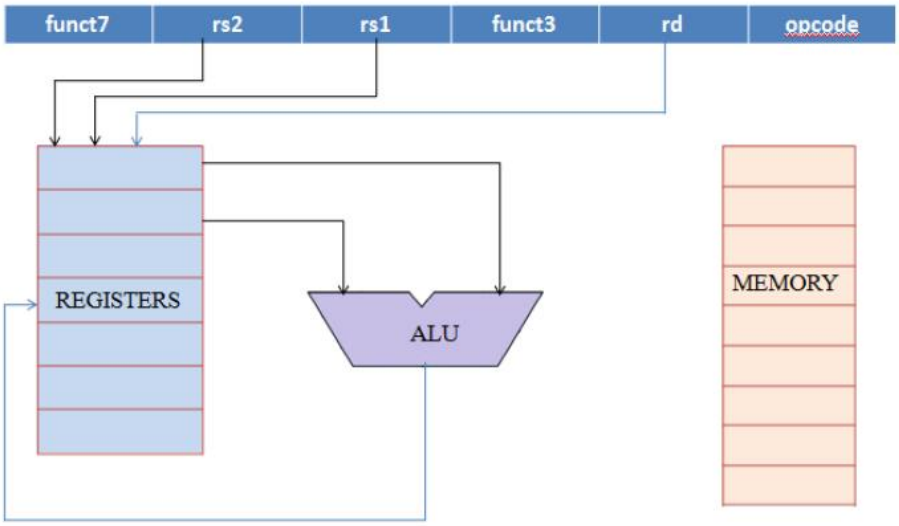
funct7=0100000;funct3=000;opcode =0110011

OR(OR Operation) R-FORMAT



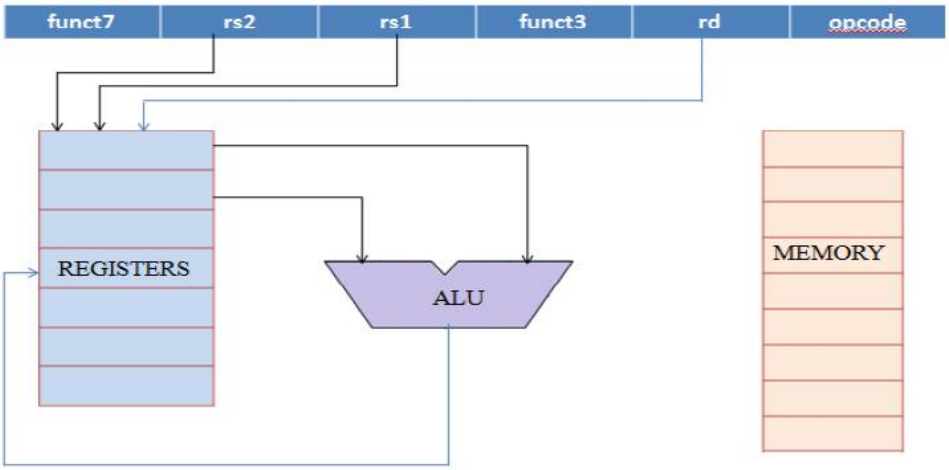
funct7=0000000;funct3=110;opcode =0110011

AND (AND Operation) R-FORMAT



funct7=0000000;funct3=111;opcode =0110011

XOR(XOR Operation) R-FORMAT



funct7=0000000;funct3=100;opcode =0110011

I-type RV32I Instruction Format

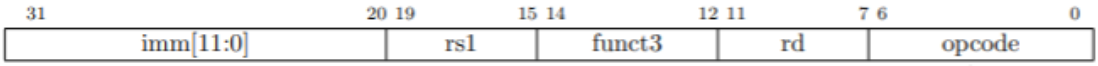
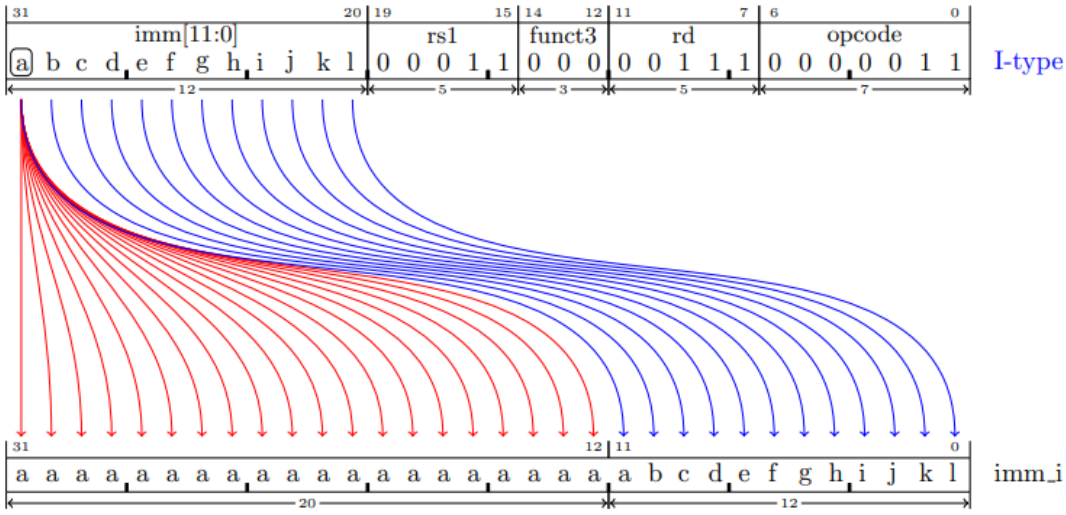


Fig: I-Type RV32I V 2.0 Instruction Format



Decoding an I-type Instruction

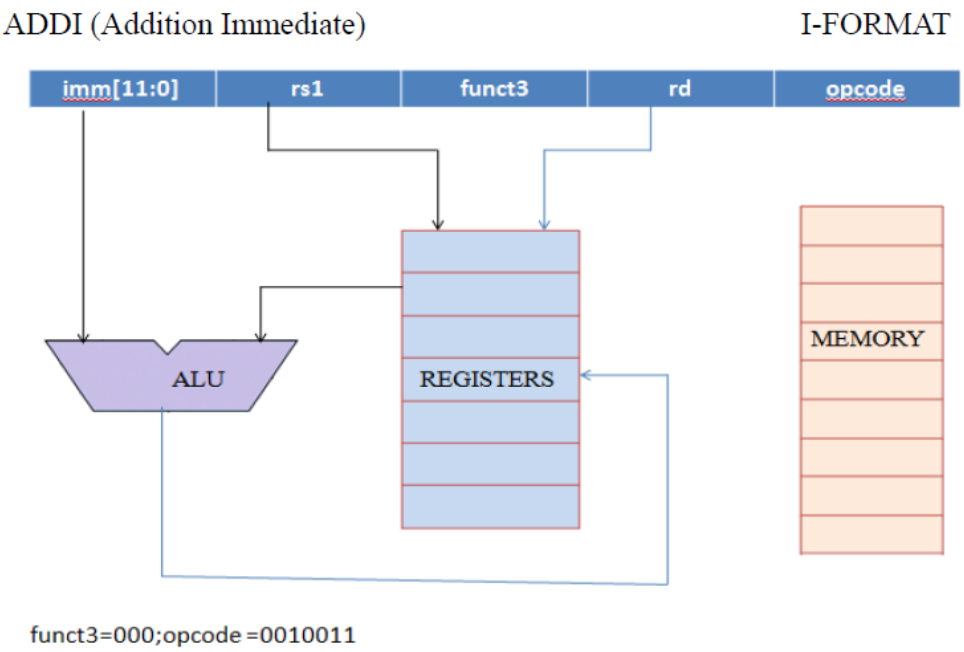
Figure: depicts the Register-type RV32I ISA V 2.0. Similar to R-type, Opcode width is of 7 bits. Source registers (rs1) and destination register (rd) are indicated by five bit fields.

The function field of 3 bits, is used for identifying the type of operation to be performed. Ithas a separate 12 bit field for holding the immediate operand, used for immediate data operations.

I-TYPE INSTRUCTION

NAME	TYPE	OPCODE	FUNCT3	DESCRIPTION
ADDI	I	0010011	0X0	rd=rs1+imm

I -FORMAT



B-type RV32I Instruction Format

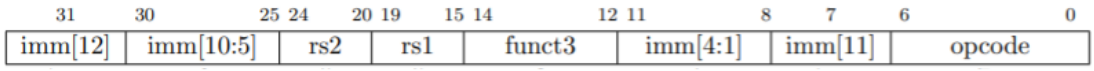


Fig: B-Type RV32I V 2.0 Instruction Format

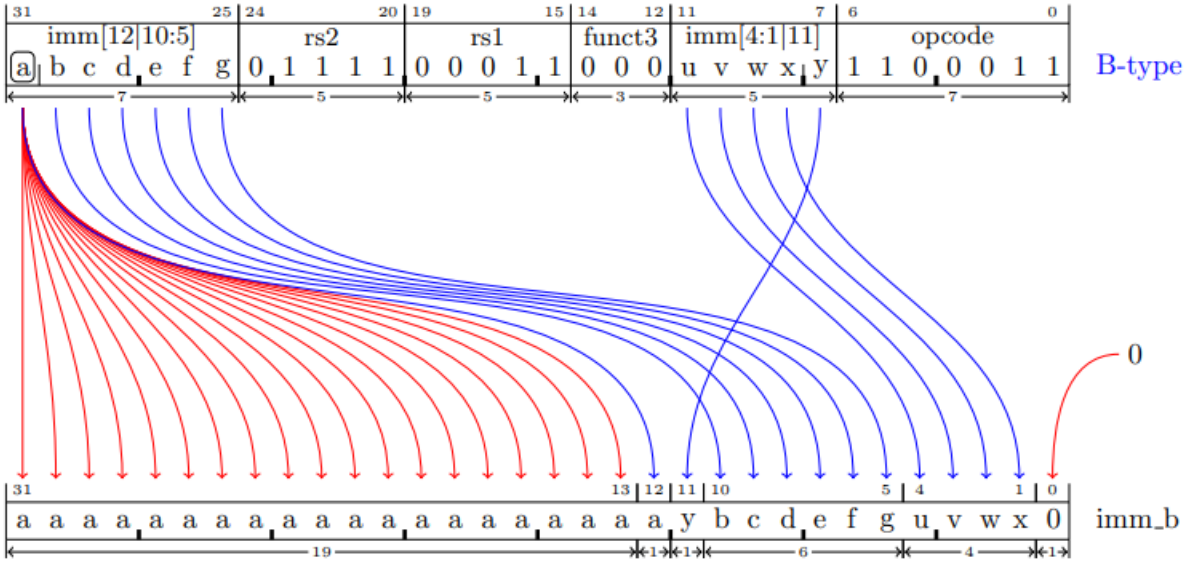


Fig : Decoding a B-type Instruction.

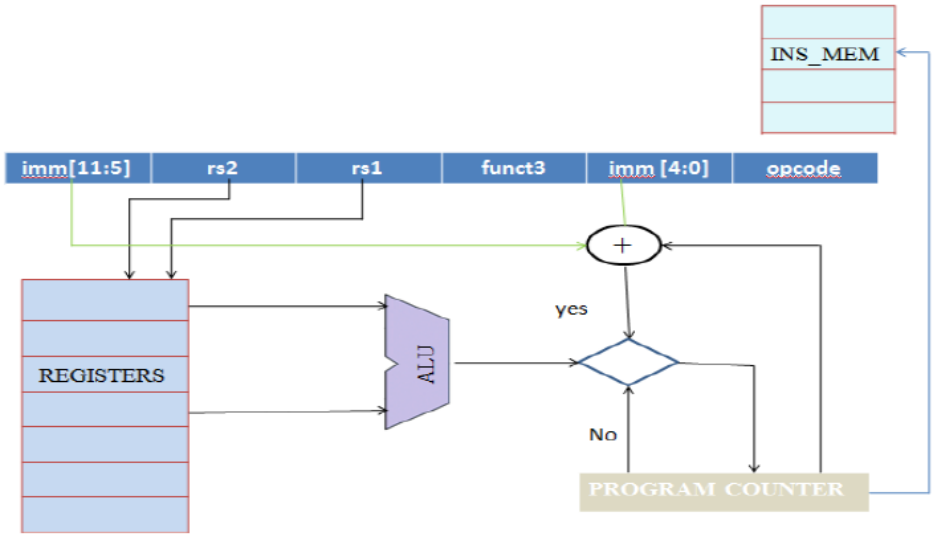
Figure 3.7 shows the Branch-type RV32I ISA V 2.0. Similar to other instructions, the Opcode width is of 7 bits. Source registers (rs1 and rs2) are indicated by five bit fields which are used for comparison for branching. The function field is of 3 bits, which is used to indicate the type of condition that need to be checked for branching. It has a separate 7+5=12 bit field space for holding the immediate operand, which is added to the program counter if a branch is taken.

B-TYPE INSTRUCTION

NAME	TYPE	OPCODE	FUNCT3	DESCRIPTION
BEQ	B	1100011	0x0	If(rs1 == rs2) PC+= imm
BNQ	B	1100011	0x1	If(rs1 ! = rs2) PC+= imm
BLT	B	1100011	0x4	If(rs1 < rs2) PC+= imm
BGT	B	1100011	0x5	If(rs1 > rs2) PC+= imm

EXECUTION OF INSTRUCTION IN B-FORMAT

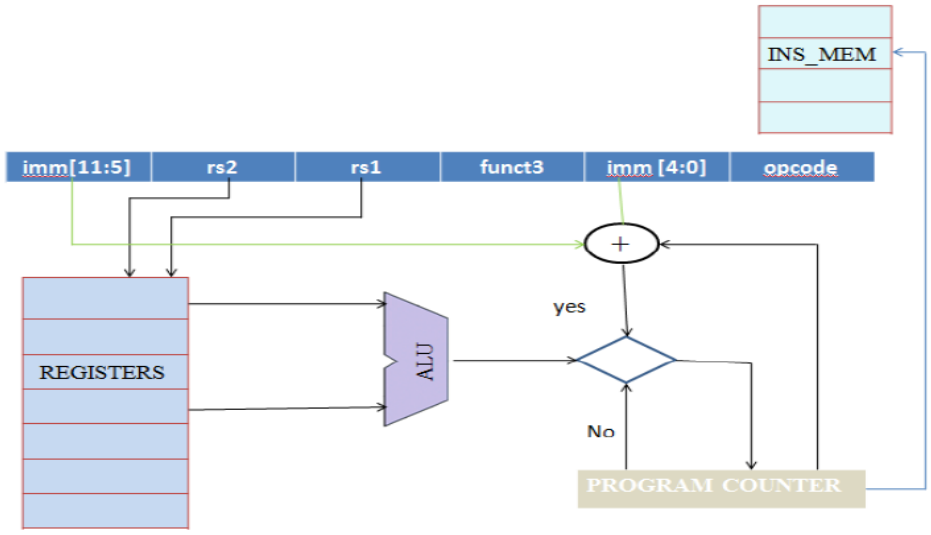
BEQ (Branch Equality)



funct3=000:opcode =1100011

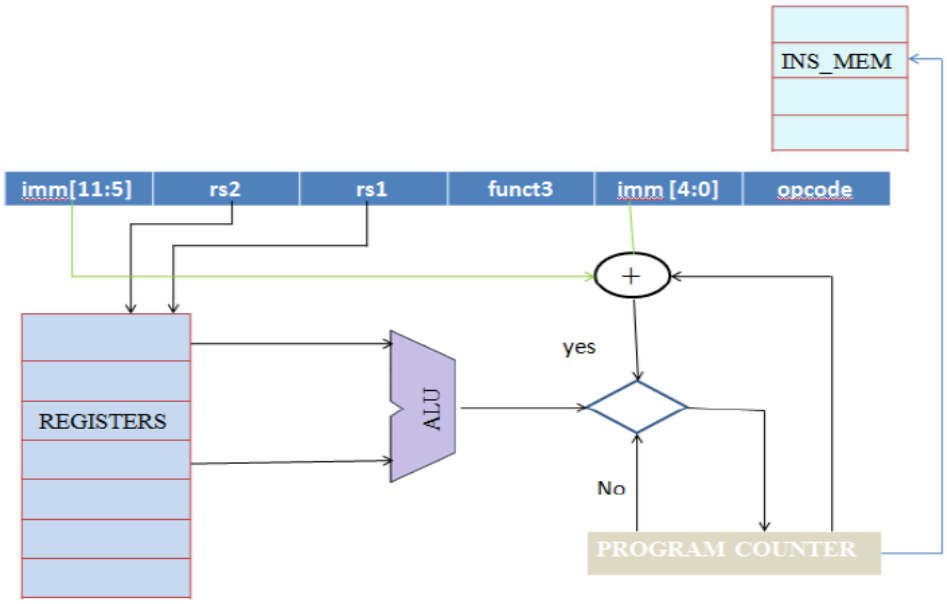
BNE (Branch Not Equal)

B-FORMAT



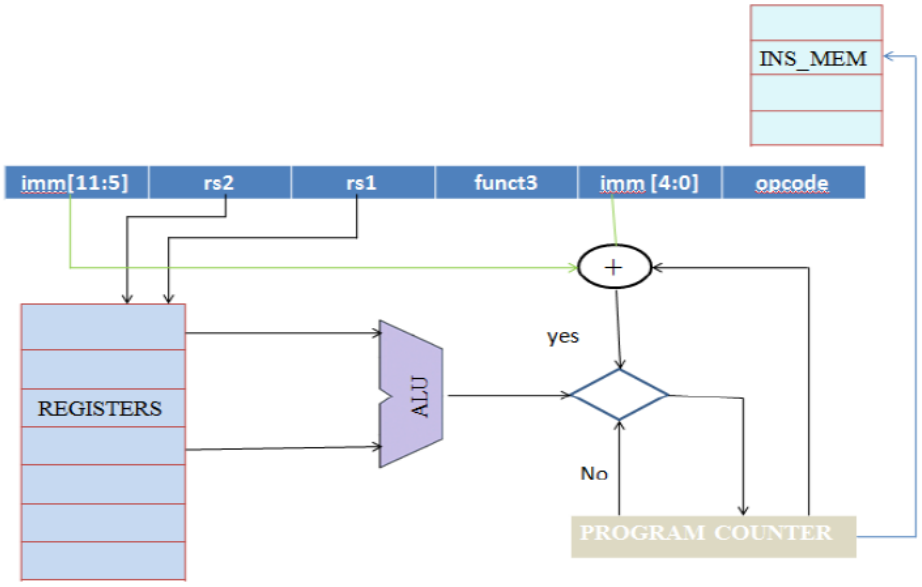
funct3=001:opcode =1100011

BLT(Branch Less Than) B-FORMAT



funct3=100;opcode =1100011

BGE (Branch Greater Than) B-FORMAT



funct3=101;opcode =1100011

TOOL USED AND RESULT

The following tools are used-

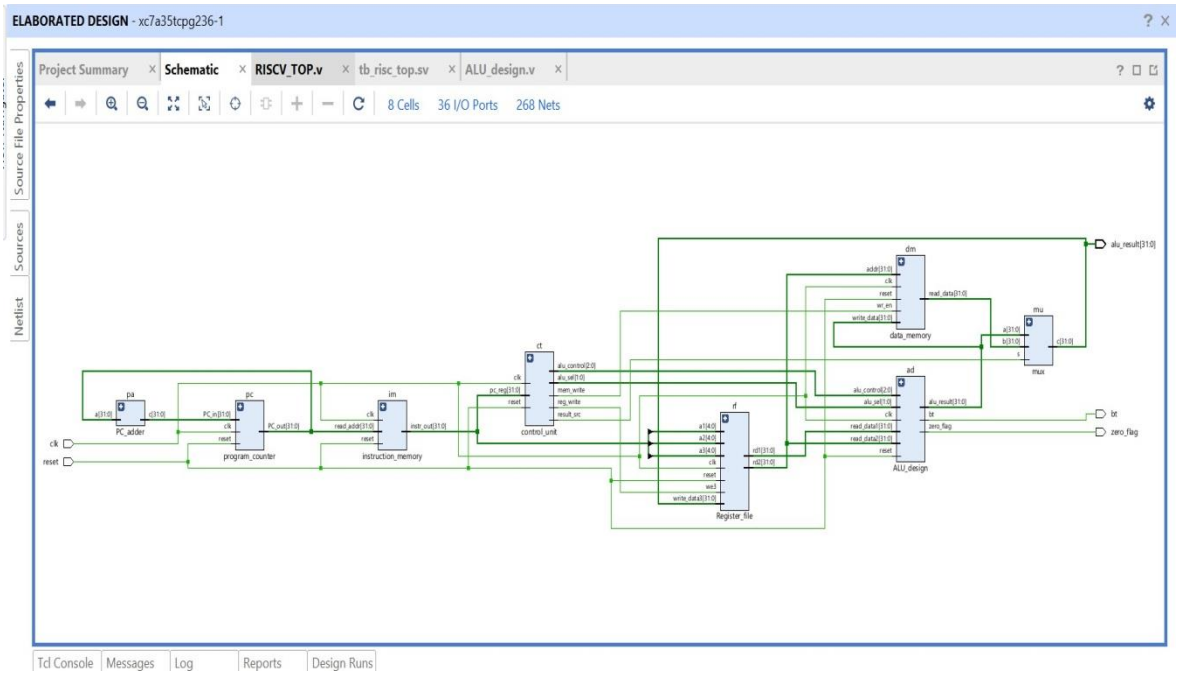
1.XILINX VIVADO 2019

2.QUESTASIM

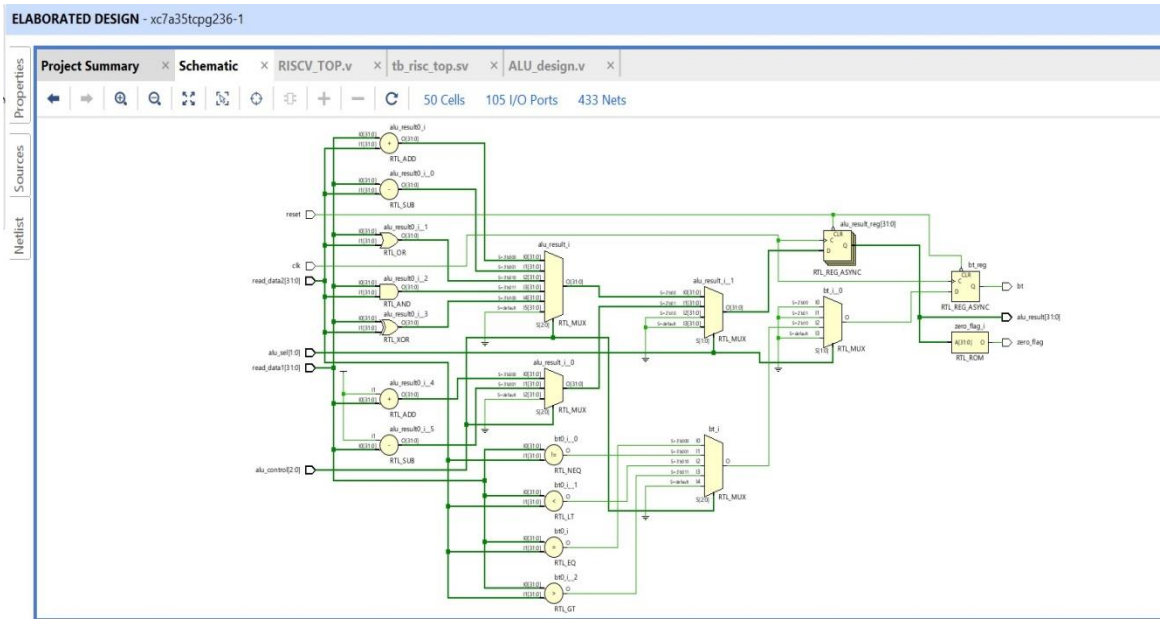
ELABORATED DESIGN

ORGANIZATION OF THE RISC-V PROCESSOR

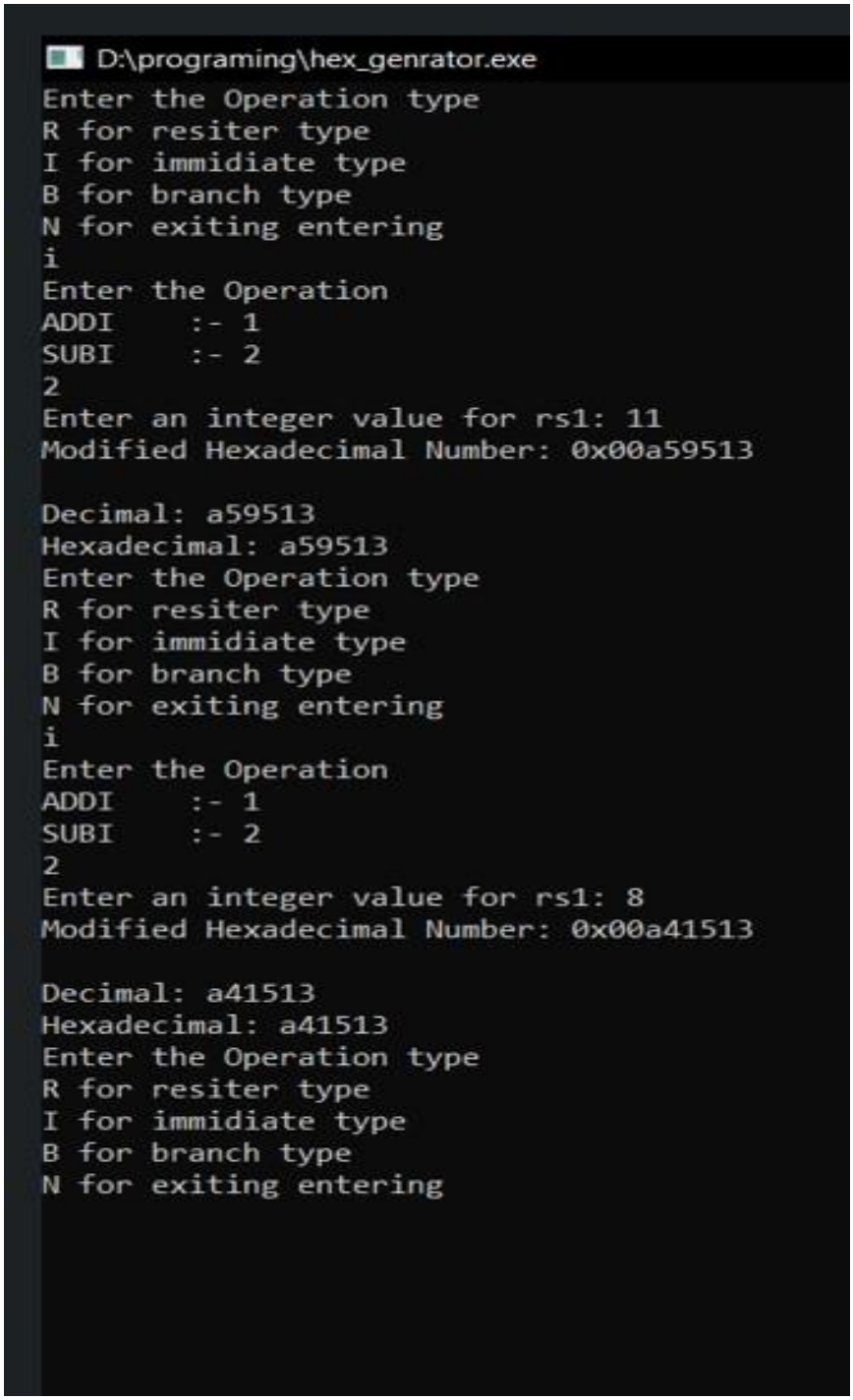
This RISC processor design has been constructed using five stages. The used stages are the Instruction Fetch stage, Instruction Decode stage, Execution stage, Memory Access stage and Write Back stage.



SCHEMATIC OF ARITHMETIC AND LOGIC UNIT



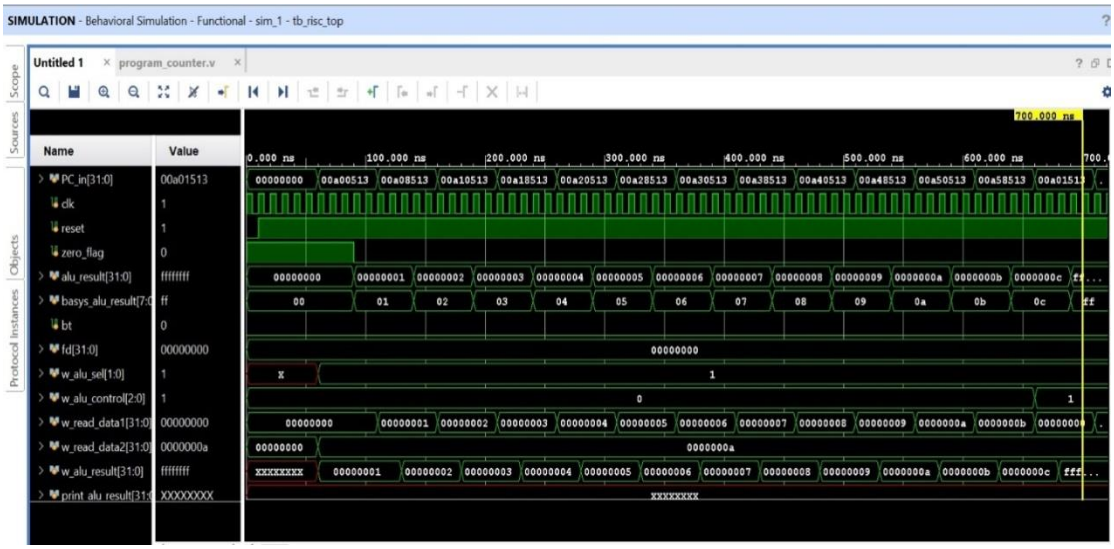
HEX FILE GENERATION THROUGH C++ CODING

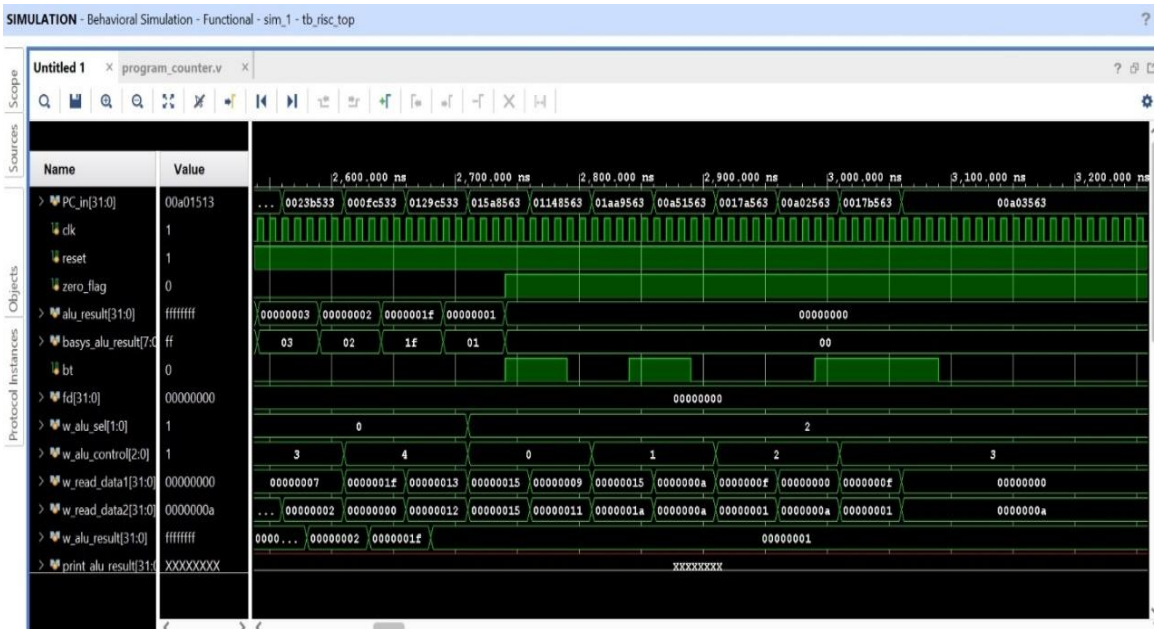



```
00a00513
00a08513
00a10513
00a18513
00a20513
00a01513
00a09513
00a11513
00a19513
00a21513
00328533
00520533
00f50533
01478533
00441533
00a61533
00a79533
01bc9533
0083a533
0107a533
00312533
```

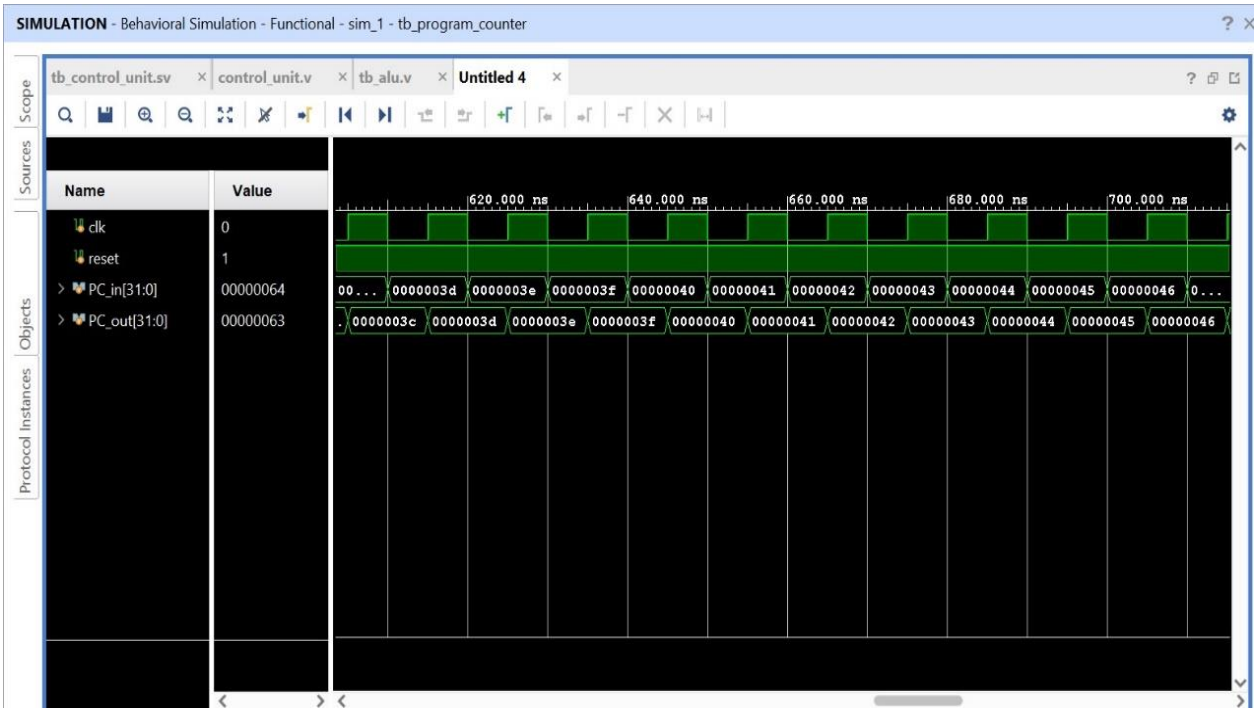
5.2. SIMULATION RESULTS

RISC TOP

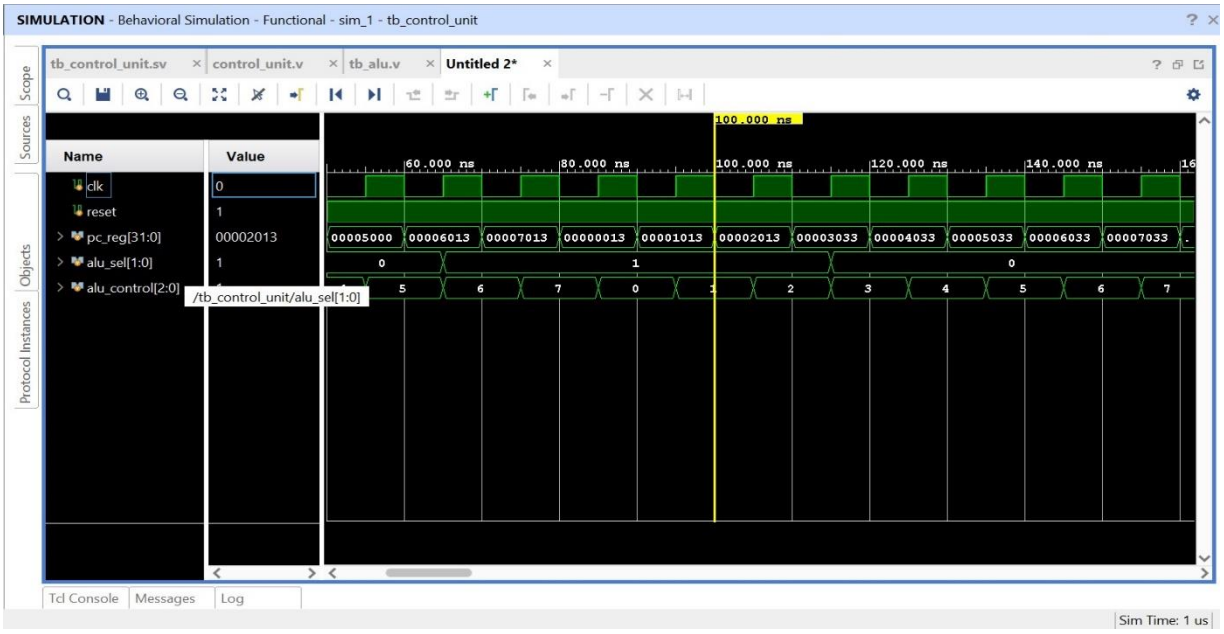




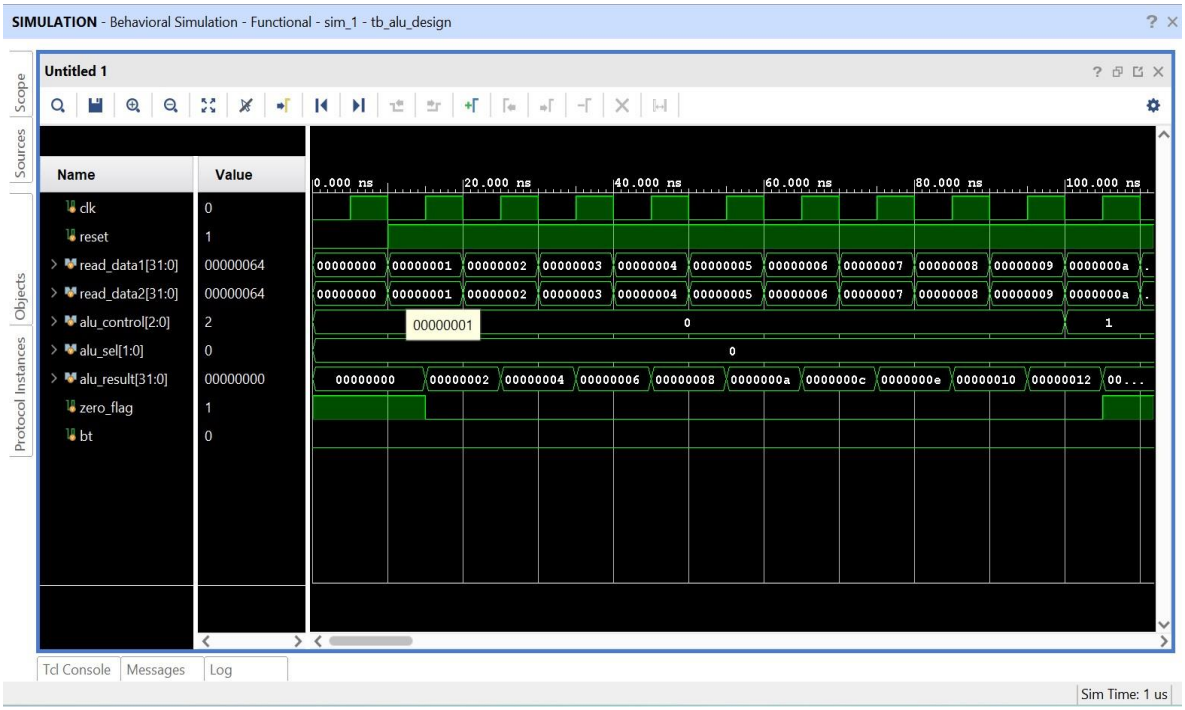
PROGRAM COUNTER



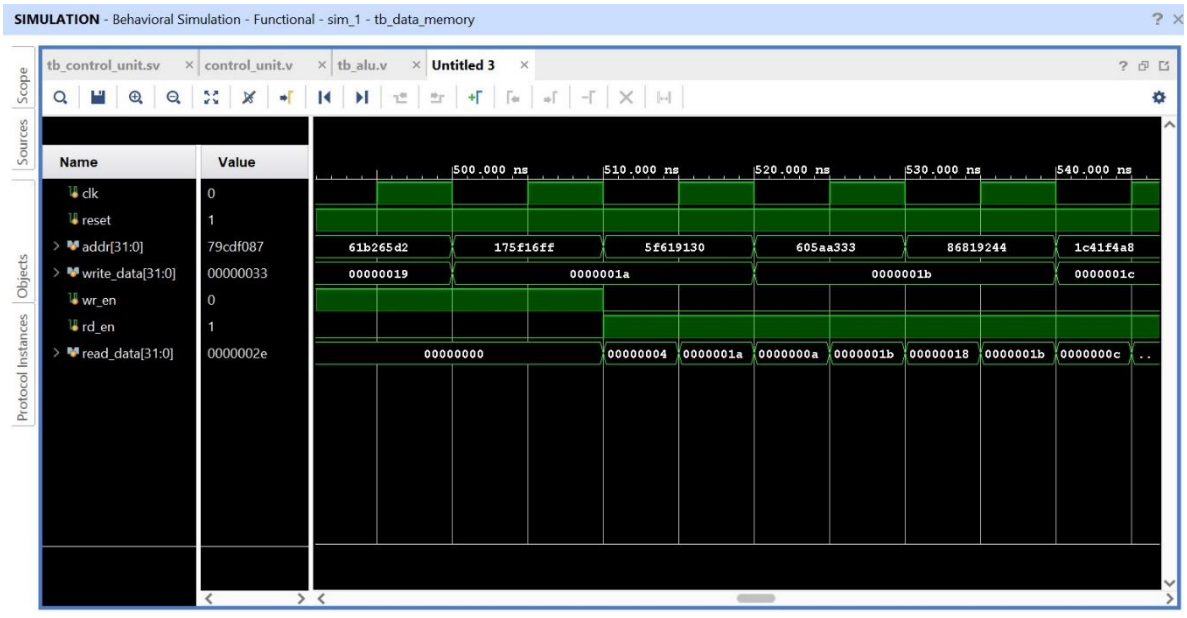
CONTROL UNIT



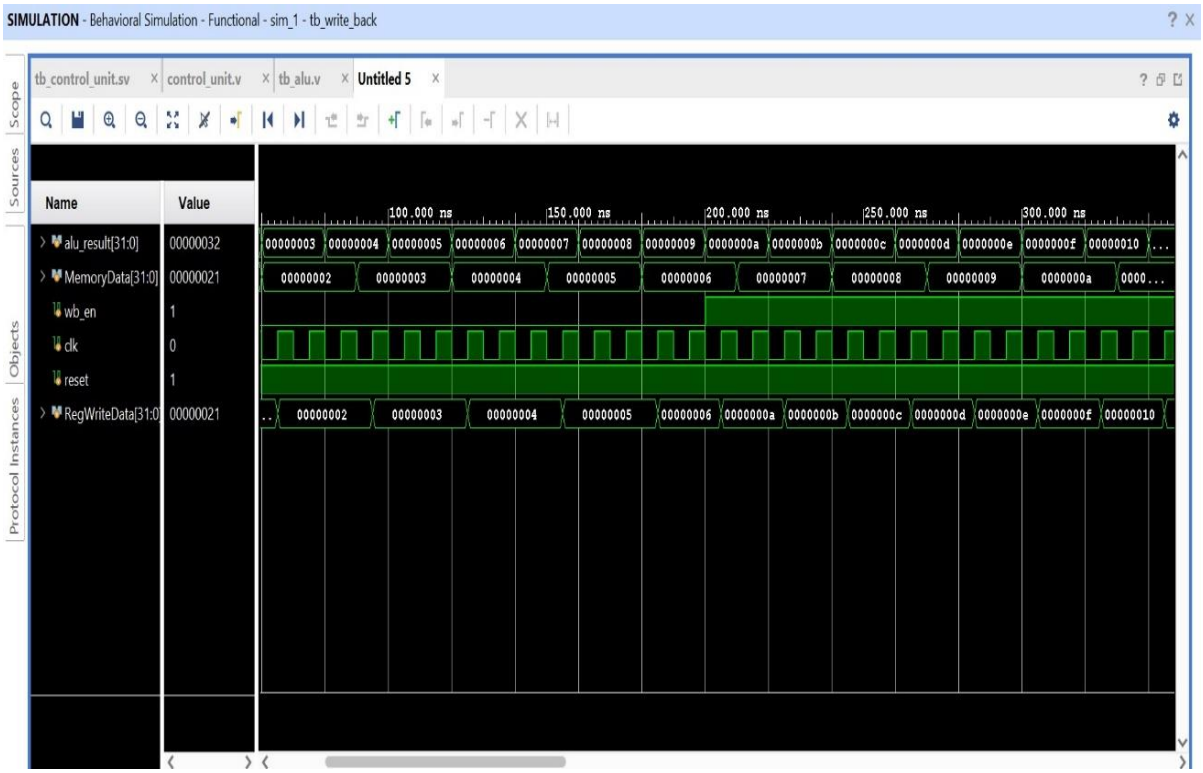
ALU SIMULATION



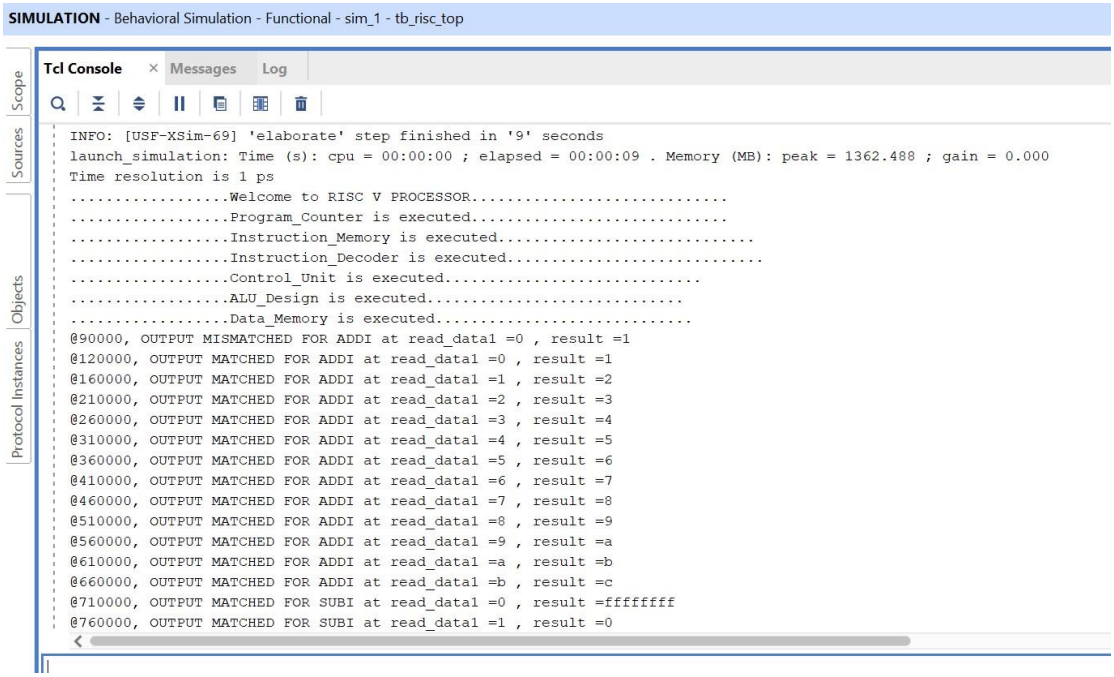
DATA MEMORY

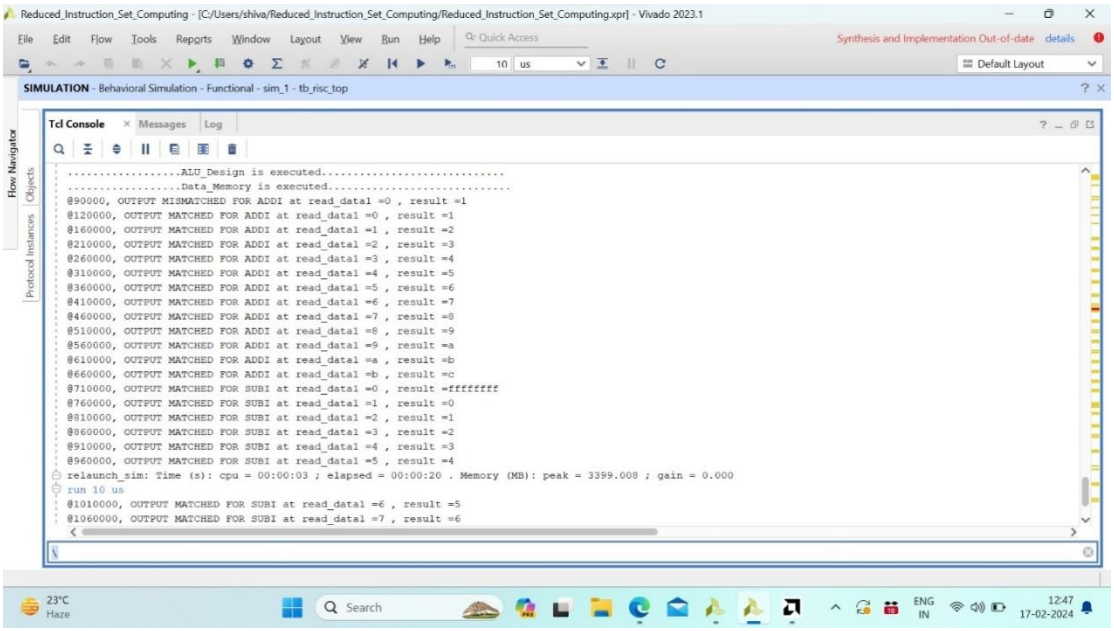


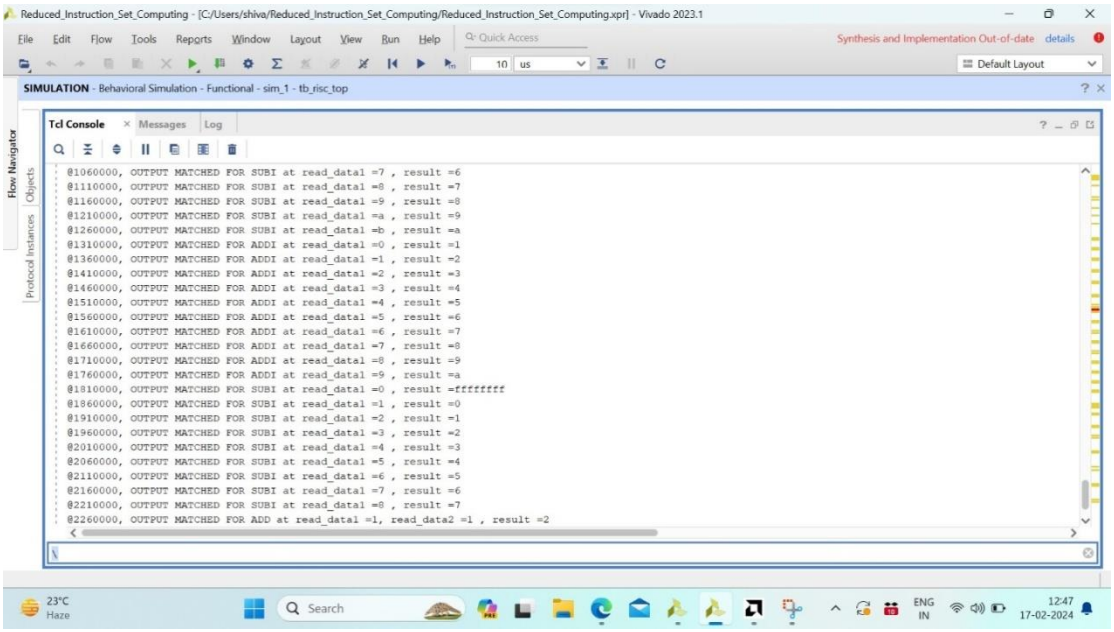
WRITE BACK

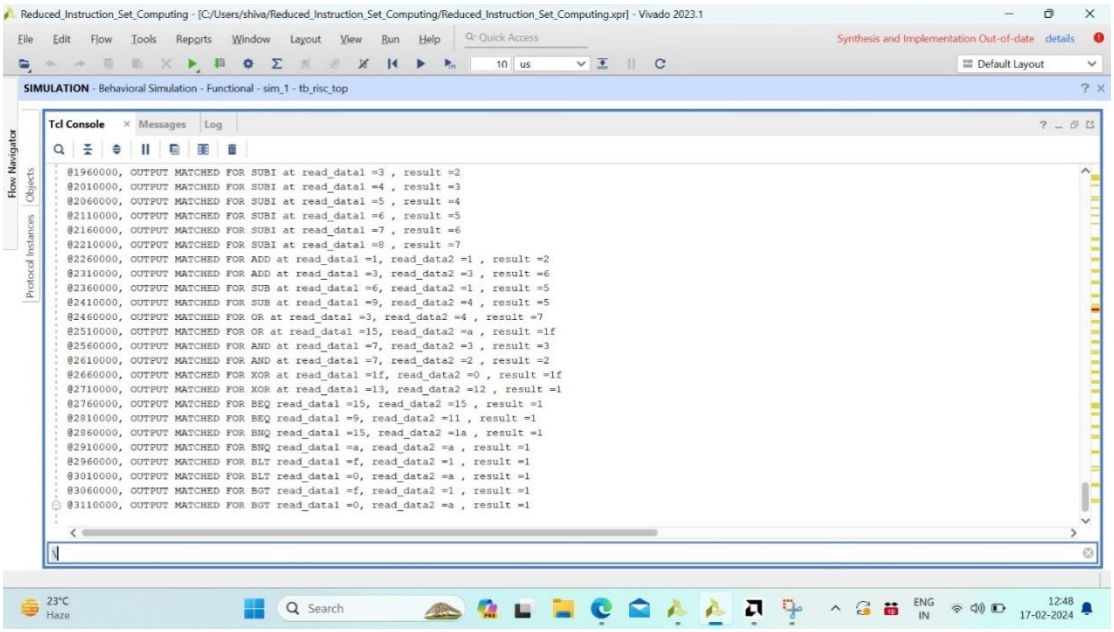

SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_write_back
tb_control_unit.sv
control_unit.v
tb_alu.v
Untitled 5
Scope
Sources
Objects
Protocol Instances
Name
Value
100.000 ns
150.000 ns
200.000 ns
250.000 ns
300.000 ns
alu_result[31:0]
00000032
MemoryData[31:0]
00000021
wb_en
1
clk
0
reset
1
RegWriteData[31:0]
00000021

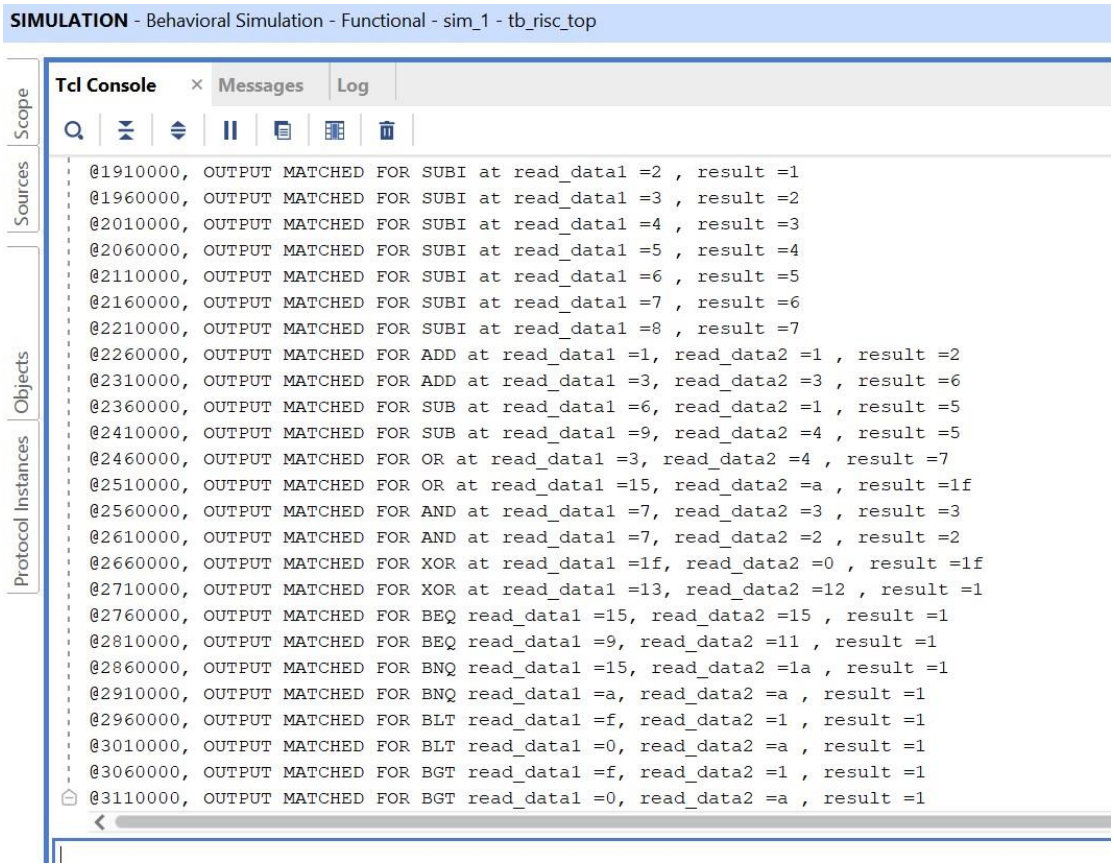
TCL CONSOLE RESULTS





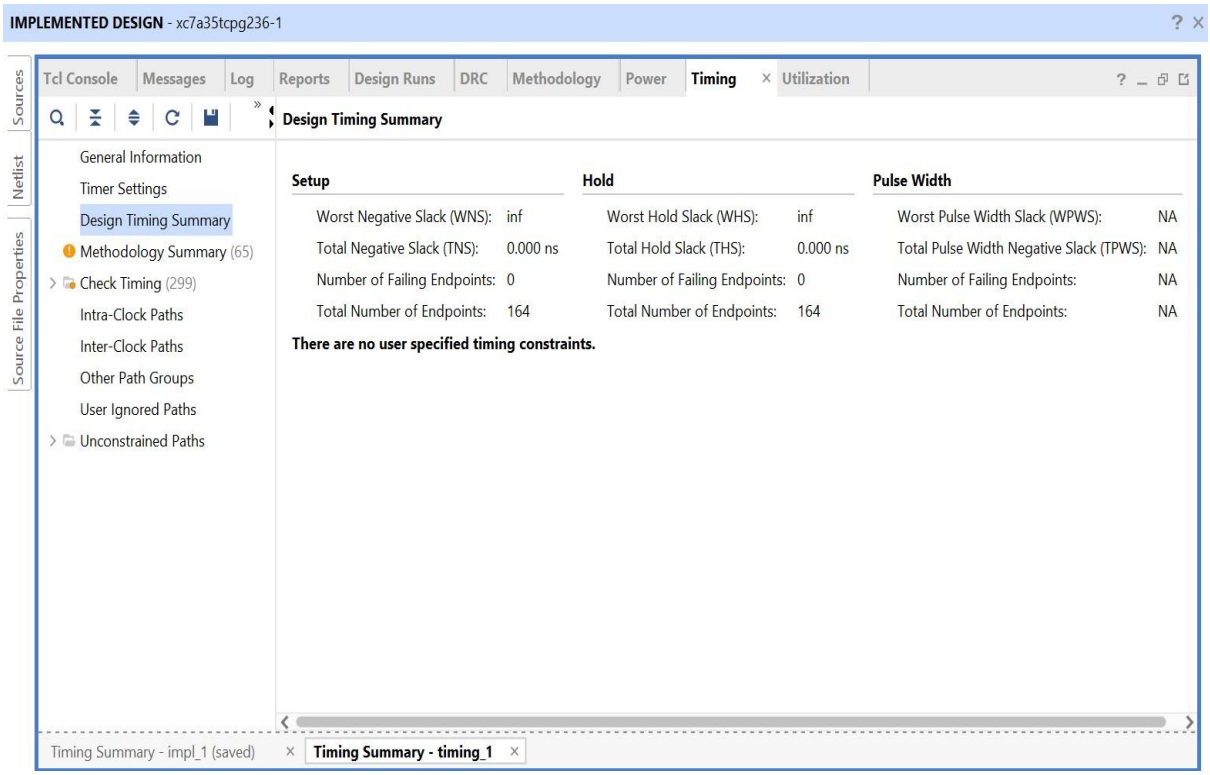

Reduced_Instruction_Set_Computing - [C:/Users/shiva/Reduced_Instruction_Set_Computing/Reduced_Instruction_Set_Computing.xpr] - Vivado 2023.1
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_risc_top
Tcl Console
Messages
Log
@1060000, OUTPUT MATCHED FOR SUBI at read_data1 =7 , result =6
@1110000, OUTPUT MATCHED FOR SUBI at read_data1 =8 , result =7
@1160000, OUTPUT MATCHED FOR SUBI at read_data1 =9 , result =8
@1210000, OUTPUT MATCHED FOR SUBI at read_data1 =a , result =9
@1260000, OUTPUT MATCHED FOR SUBI at read_data1 =b , result =a
@1310000, OUTPUT MATCHED FOR ADDI at read_data1 =0 , result =1
@1360000, OUTPUT MATCHED FOR ADDI at read_data1 =1 , result =2
@1410000, OUTPUT MATCHED FOR ADDI at read_data1 =2 , result =3
@1460000, OUTPUT MATCHED FOR ADDI at read_data1 =3 , result =4
@1510000, OUTPUT MATCHED FOR ADDI at read_data1 =4 , result =5
@1560000, OUTPUT MATCHED FOR ADDI at read_data1 =5 , result =6
@1610000, OUTPUT MATCHED FOR ADDI at read_data1 =6 , result =7
@1660000, OUTPUT MATCHED FOR ADDI at read_data1 =7 , result =8
@1710000, OUTPUT MATCHED FOR ADDI at read_data1 =8 , result =9
@1760000, OUTPUT MATCHED FOR ADDI at read_data1 =9 , result =a
@1810000, OUTPUT MATCHED FOR SUBI at read_data1 =0 , result =ffffffff
@1860000, OUTPUT MATCHED FOR SUBI at read_data1 =1 , result =0
@1910000, OUTPUT MATCHED FOR SUBI at read_data1 =2 , result =1
@1960000, OUTPUT MATCHED FOR SUBI at read_data1 =3 , result =2
@2010000, OUTPUT MATCHED FOR SUBI at read_data1 =4 , result =3
@2060000, OUTPUT MATCHED FOR SUBI at read_data1 =5 , result =4
@2110000, OUTPUT MATCHED FOR SUBI at read_data1 =6 , result =5
@2160000, OUTPUT MATCHED FOR SUBI at read_data1 =7 , result =6
@2210000, OUTPUT MATCHED FOR SUBI at read_data1 =8 , result =7
@2260000, OUTPUT MATCHED FOR ADD at read_data1 =1, read_data2 =1 , result =2
23°C
Haze
Search
ENG
IN
12:47
17-02-2024


Reduced_Instruction_Set_Computing - [C:/Users/shiva/Reduced_Instruction_Set_Computing/Reduced_Instruction_Set_Computing.xpr] - Vivado 2023.1
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_risc_top
Tcl Console
Messages
Log
@1960000, OUTPUT MATCHED FOR SUBI at read_data1 =3 , result =2
@2010000, OUTPUT MATCHED FOR SUBI at read_data1 =4 , result =3
@2060000, OUTPUT MATCHED FOR SUBI at read_data1 =5 , result =4
@2110000, OUTPUT MATCHED FOR SUBI at read_data1 =6 , result =5
@2160000, OUTPUT MATCHED FOR SUBI at read_data1 =7 , result =6
@2210000, OUTPUT MATCHED FOR SUBI at read_data1 =8 , result =7
@2260000, OUTPUT MATCHED FOR ADD at read_data1 =1, read_data2 =1 , result =2
@2310000, OUTPUT MATCHED FOR ADD at read_data1 =3, read_data2 =3 , result =6
@2360000, OUTPUT MATCHED FOR SUB at read_data1 =6, read_data2 =1 , result =5
@2410000, OUTPUT MATCHED FOR SUB at read_data1 =9, read_data2 =4 , result =5
@2460000, OUTPUT MATCHED FOR OR at read_data1 =3, read_data2 =4 , result =7
@2510000, OUTPUT MATCHED FOR OR at read_data1 =15, read_data2 =a , result =1f
@2560000, OUTPUT MATCHED FOR AND at read_data1 =7, read_data2 =3 , result =3
@2610000, OUTPUT MATCHED FOR AND at read_data1 =7, read_data2 =2 , result =2
@2660000, OUTPUT MATCHED FOR XOR at read_data1 =1f, read_data2 =0 , result =1f
@2710000, OUTPUT MATCHED FOR XOR at read_data1 =13, read_data2 =12 , result =1
@2760000, OUTPUT MATCHED FOR BEQ read_data1 =15, read_data2 =15 , result =1
@2810000, OUTPUT MATCHED FOR BEQ read_data1 =9, read_data2 =11 , result =1
@2860000, OUTPUT MATCHED FOR BNQ read_data1 =15, read_data2 =1a , result =1
@2910000, OUTPUT MATCHED FOR BNQ read_data1 =a, read_data2 =a , result =1
@2960000, OUTPUT MATCHED FOR BLT read_data1 =f, read_data2 =1 , result =1
@3010000, OUTPUT MATCHED FOR BLT read_data1 =0, read_data2 =a , result =1
@3060000, OUTPUT MATCHED FOR BGT read_data1 =f, read_data2 =1 , result =1
@3110000, OUTPUT MATCHED FOR BGT read_data1 =0, read_data2 =a , result =1
23°C
Haze
Search
ENG
IN
12:48
17-02-2024

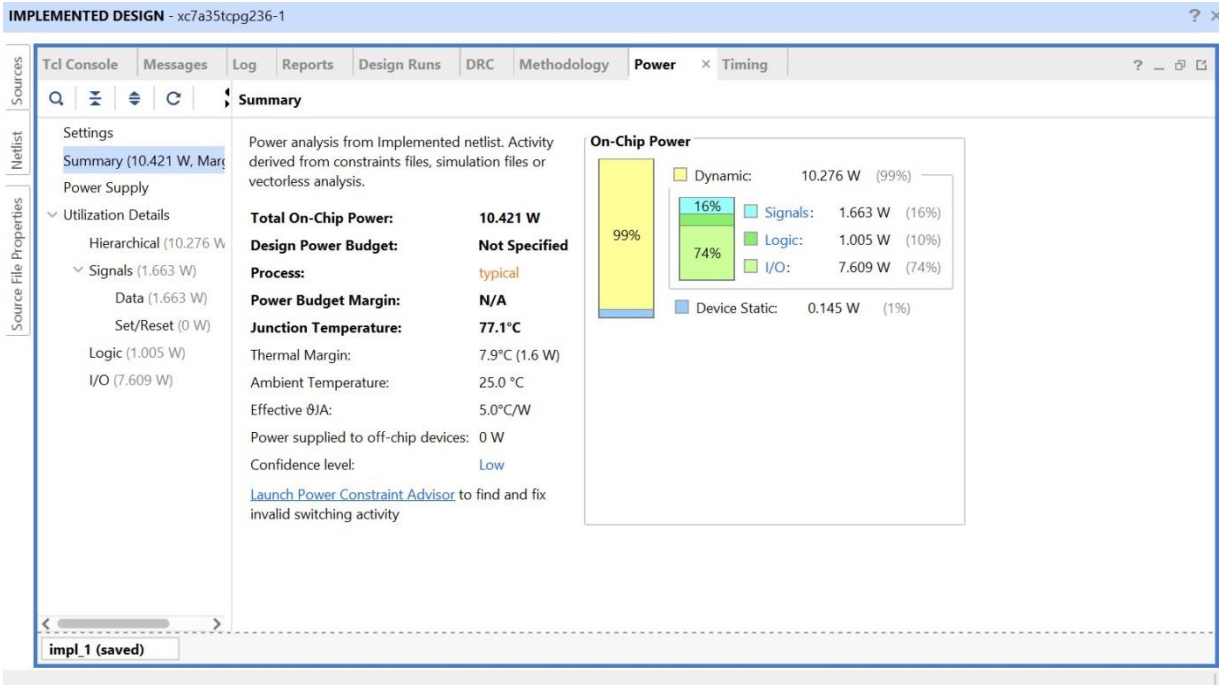

SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_risc_top
Scope
Sources
Objects
Protocol Instances
Tcl Console
Messages
Log
@1910000, OUTPUT MATCHED FOR SUBI at read_data1 =2 , result =1
@1960000, OUTPUT MATCHED FOR SUBI at read_data1 =3 , result =2
@2010000, OUTPUT MATCHED FOR SUBI at read_data1 =4 , result =3
@2060000, OUTPUT MATCHED FOR SUBI at read_data1 =5 , result =4
@2110000, OUTPUT MATCHED FOR SUBI at read_data1 =6 , result =5
@2160000, OUTPUT MATCHED FOR SUBI at read_data1 =7 , result =6
@2210000, OUTPUT MATCHED FOR SUBI at read_data1 =8 , result =7
@2260000, OUTPUT MATCHED FOR ADD at read_data1 =1, read_data2 =1 , result =2
@2310000, OUTPUT MATCHED FOR ADD at read_data1 =3, read_data2 =3 , result =6
@2360000, OUTPUT MATCHED FOR SUB at read_data1 =6, read_data2 =1 , result =5
@2410000, OUTPUT MATCHED FOR SUB at read_data1 =9, read_data2 =4 , result =5
@2460000, OUTPUT MATCHED FOR OR at read_data1 =3, read_data2 =4 , result =7
@2510000, OUTPUT MATCHED FOR OR at read_data1 =15, read_data2 =a , result =1f
@2560000, OUTPUT MATCHED FOR AND at read_data1 =7, read_data2 =3 , result =3
@2610000, OUTPUT MATCHED FOR AND at read_data1 =7, read_data2 =2 , result =2
@2660000, OUTPUT MATCHED FOR XOR at read_data1 =1f, read_data2 =0 , result =1f
@2710000, OUTPUT MATCHED FOR XOR at read_data1 =13, read_data2 =12 , result =1
@2760000, OUTPUT MATCHED FOR BEQ read_data1 =15, read_data2 =15 , result =1
@2810000, OUTPUT MATCHED FOR BEQ read_data1 =9, read_data2 =11 , result =1
@2860000, OUTPUT MATCHED FOR BNQ read_data1 =15, read_data2 =1a , result =1
@2910000, OUTPUT MATCHED FOR BNQ read_data1 =a, read_data2 =a , result =1
@2960000, OUTPUT MATCHED FOR BLT read_data1 =f, read_data2 =1 , result =1
@3010000, OUTPUT MATCHED FOR BLT read_data1 =0, read_data2 =a , result =1
@3060000, OUTPUT MATCHED FOR BGT read_data1 =f, read_data2 =1 , result =1
@3110000, OUTPUT MATCHED FOR BGT read_data1 =0, read_data2 =a , result =1

SYNTHESIS RESULT

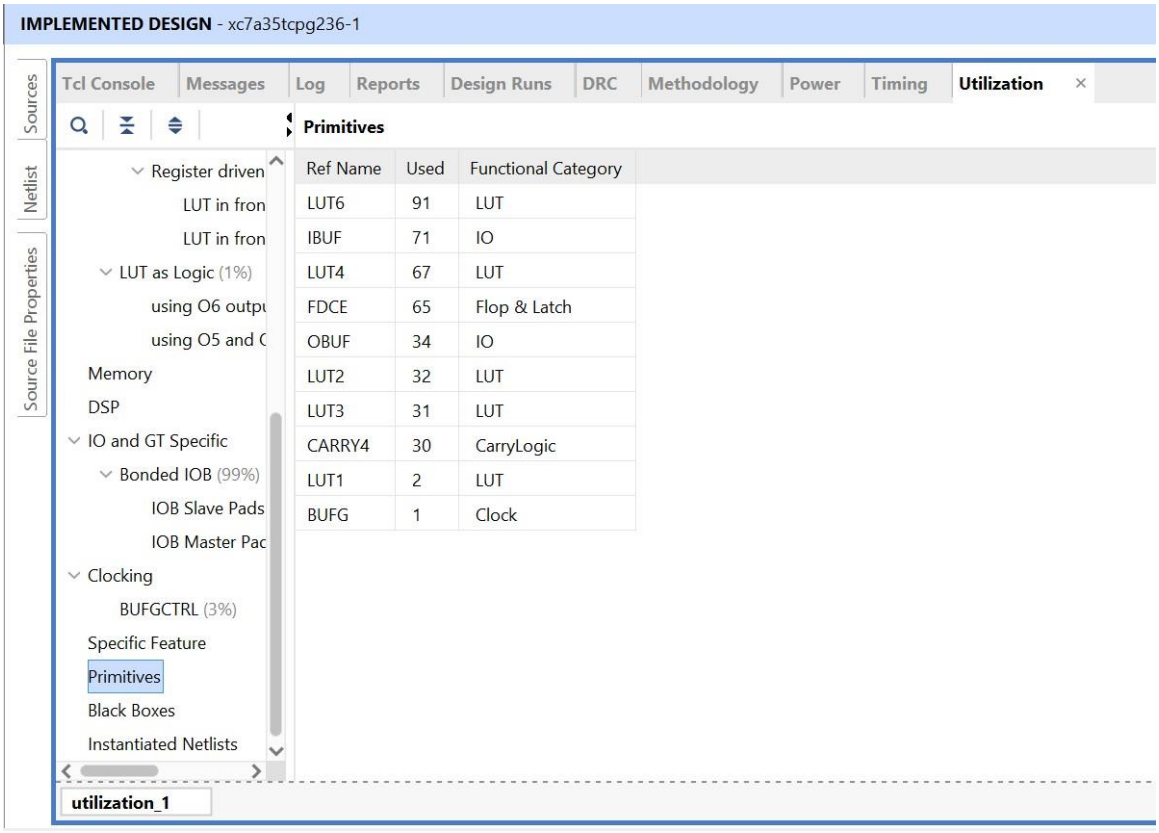
TIMING SUMMARY

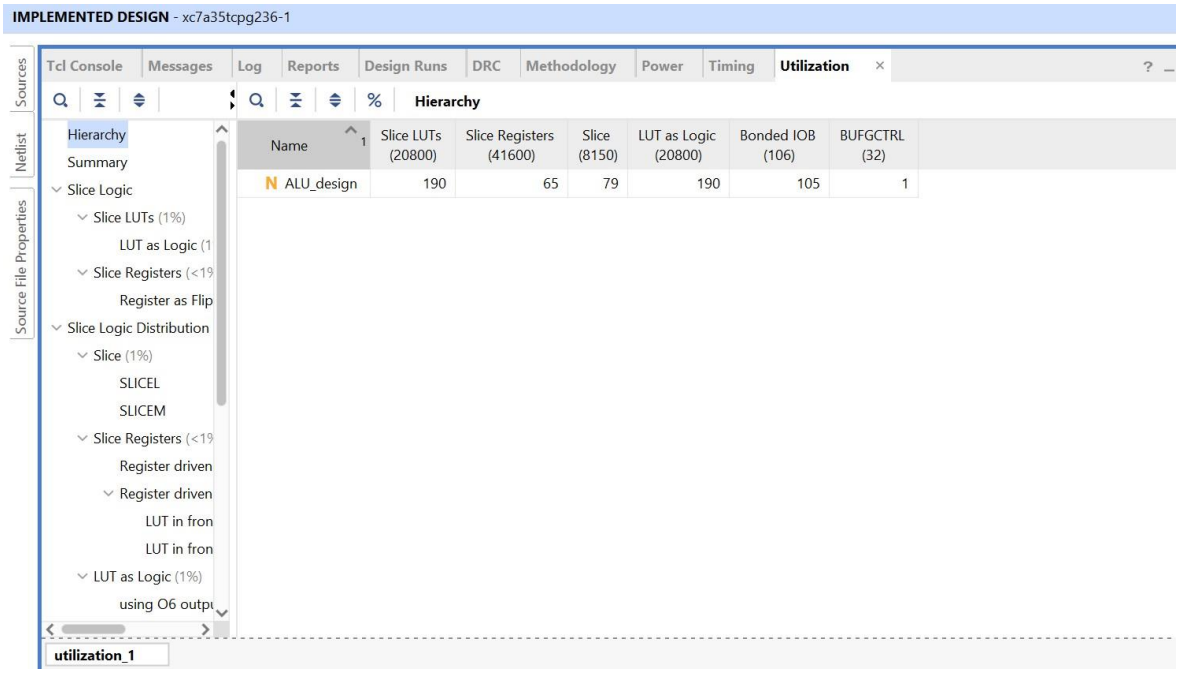


POWER



UTILIZATION REPORT





HARDWARE IMPLEMENTATION

ZC702 Overview

The ZC702 evaluation board for the XC7Z020 SoC provides a hardware environment for developing at and evaluating designs targeting the Zynq® XC7Z020-1CLG484C device. The ZC702 board provides features common to many embedded processing systems, including DDR3 component memory, a tri-mode Ethernet PHY, general purpose I/O, and two UART interfaces. Other features can be supported using VITA-57 FPGA mezzanine cards (FMC) attached to either of two low pin count (LPC) FMC connectors.

ZC702 Board Features

The ZC702 board features are listed in here. Detailed information for each feature is provided in Feature Descriptions.

- Zynq XC7Z020-1CLG484C device
- 1 GB DDR3 component memory (four 256 Mb x 8 devices)
- 128 Mb Quad SPI flash memory
- USB 2.0 ULPI (UTMI+ low pin interface) transceiver
- Secure Digital (SD) connector
- USB JTAG interface using a Digilent module
- Clock sources:
  - Fixed 200 MHz LVDS oscillator (differential)
  - I2C programmable LVDS oscillator (differential)
  - Fixed 33.33 MHz LVCMOS oscillator (single-ended)
- Ethernet PHY RGMII interface with RJ-45 connector
- USB-to-UART bridge
- HDMI codec
- I2C bus

I2C bus multiplexed to:

- Si570 user clock
- ADV7511 HDMI codec
- M24C08 EEPROM (1 kB)
- 1-To-16 TCA6416APWR port expander
- RTC-8564JE real time clock
- FMC1 LPC connector
- FMC2 LPC connector
- PMBUS data/clock
- Status LEDs:
  - Ethernet status
  - Power good
  - FPGA INIT

FPGA DONE

• User I/O:

° Two programmable logic (PL) user pushbuttons
° PL user DIP switch (2-pole)
° Eight PL user LEDs
° Two processing system (PS) pushbuttons shared with PS 2-pole DIP switch
° Two PS user LEDs
° Dual row Pmod GPIO header
° Single row Pmod GPIO header

• SoC PS Reset Pushbuttons:
° SRST_B PS reset button
° POR_B PS reset button

• Two VITA 57.1 FMC LPC connectors
• Power on/off slide switch
• Power management with PMBus voltage and current monitoring via TI power controllers
• Dual 12-bit 1 MSPS XADC analog-to-digital front end

• Configuration options:

Quad SPI flash memory

° USB JTAG configuration port (Digilent module)
° Platform cable header JTAG configuration port
° 20-pin PL PJTAG header
° 20-pin PS JTAG header

ZYNQ -7000 SOC BLOCK DIAGRAM

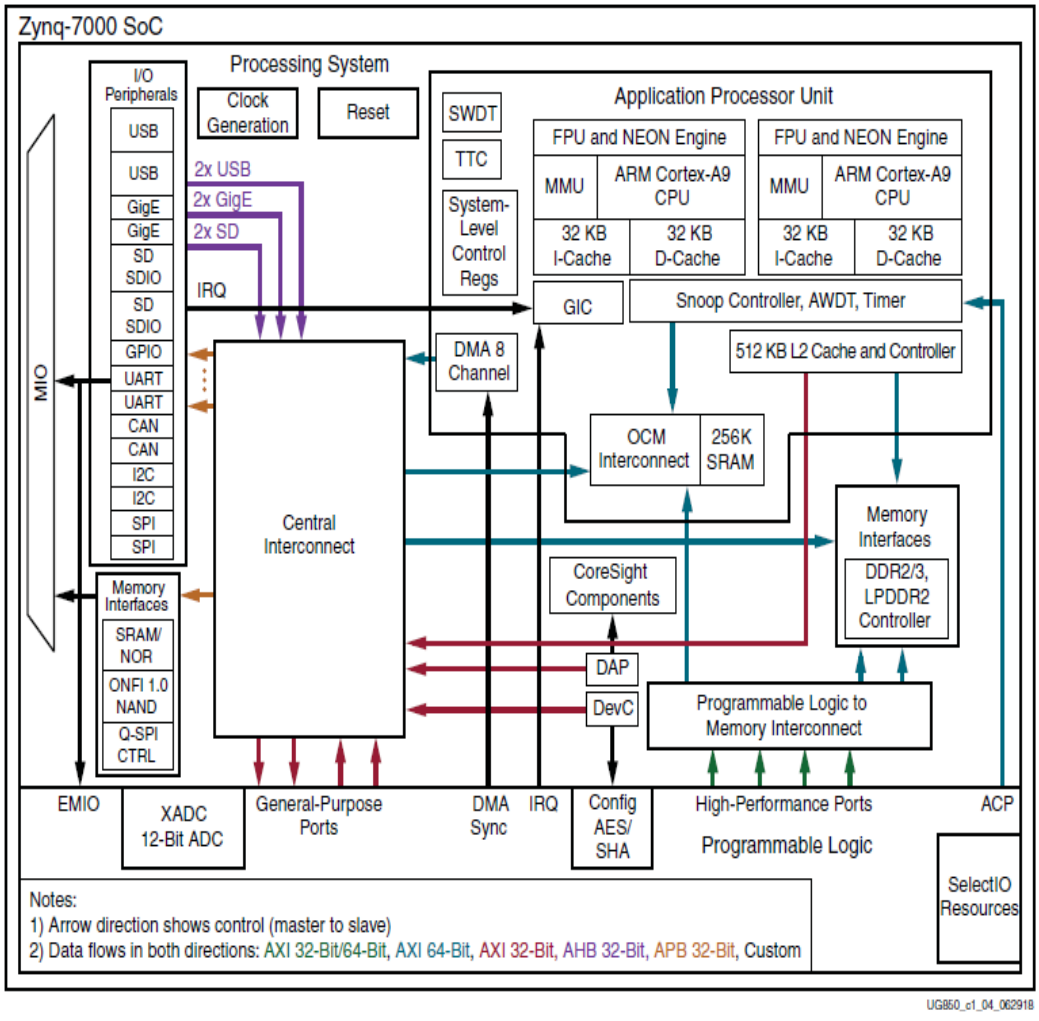


Figure 1-4: **Zynq-7000 SoC Block Diagram**

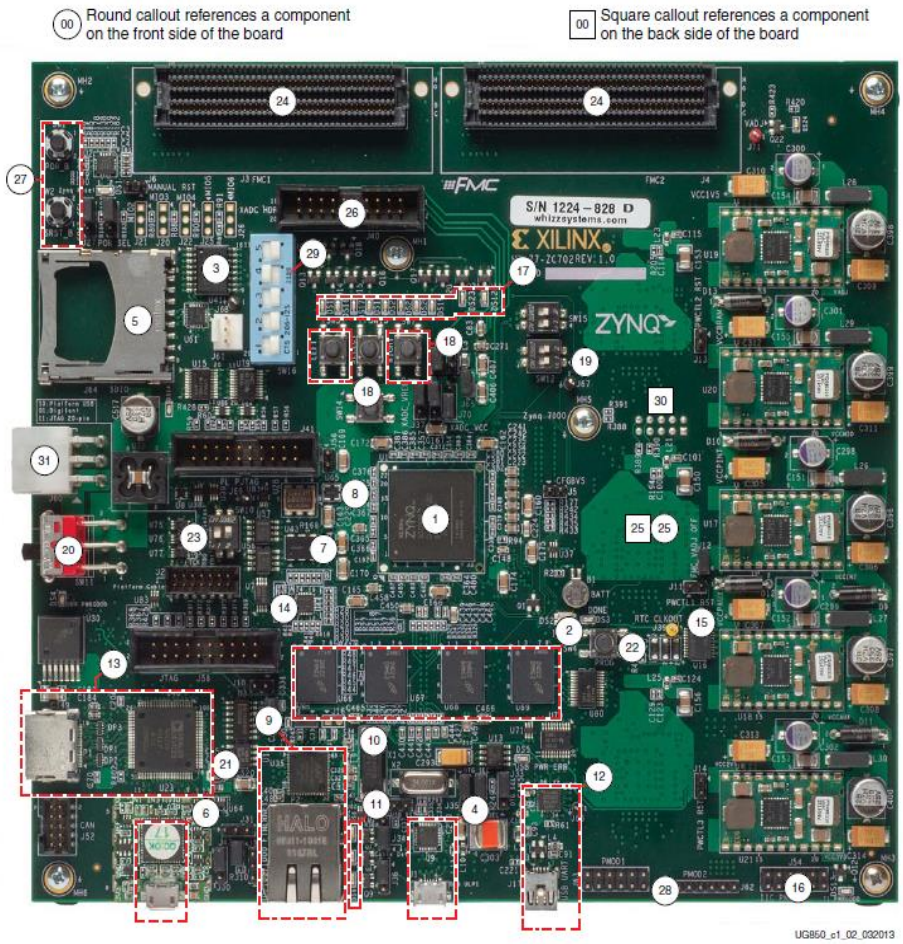


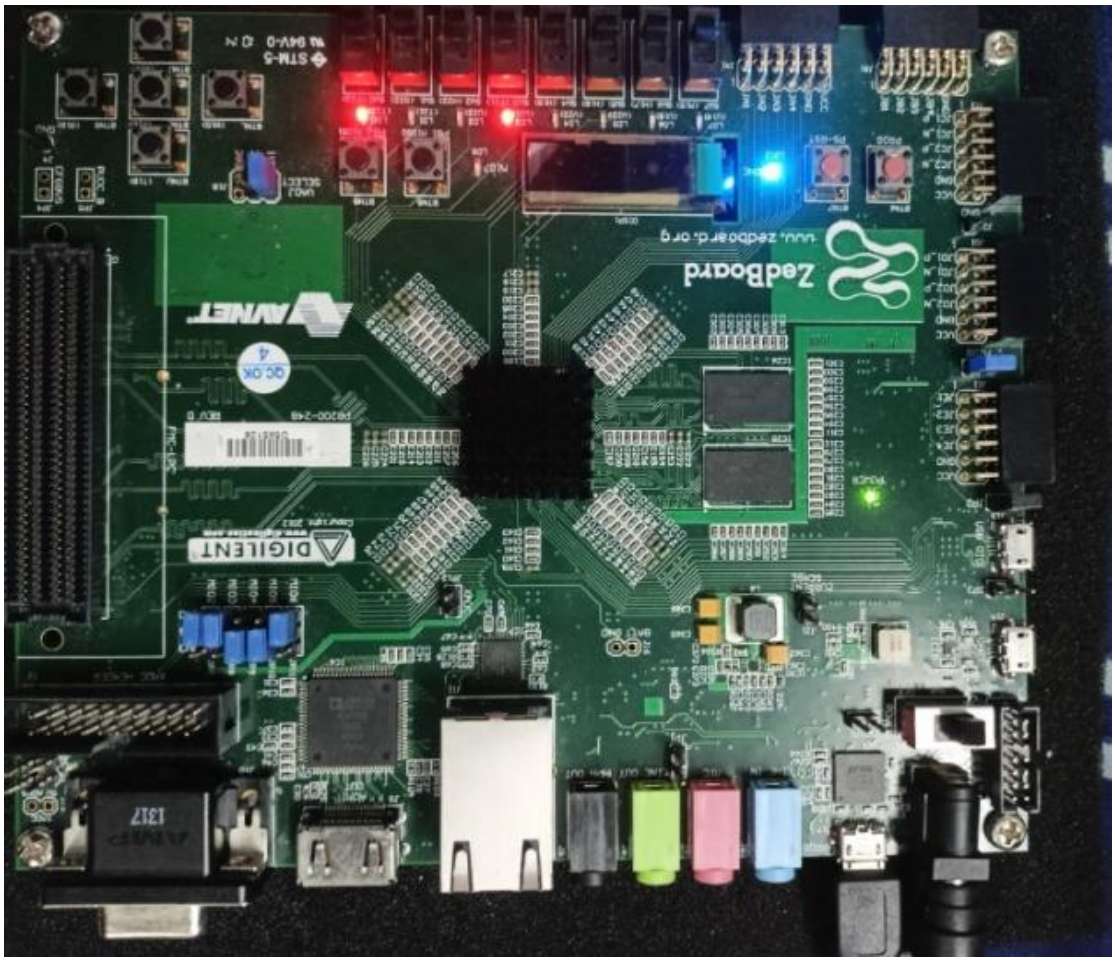
Figure 1-2: **ZC702 Board Component Locations**

Table 1-1: **ZC702 Board Component Descriptions**

Callout	Reference Designator	Component Description	Notes	Schematic(1) 0381449 Page Number
1	U1	Zynq-7000 XC7Z020 SoC	Xilinx part number: XC7Z020-1CLG484C	
2	U66–U69	DDR3 Component Memory, 1 GB	4 each 256Mb X 8 SDRAM Micron Technology Inc, MT41J256M8DA-107	16–19
3	U41	Quad-SPI Flash Memory, 128 Mb	Micron N25Q128A11ESF40G	20
4	U9, J1	USB 2.0 ULPI Transceiver, USB Mini-B connector	SMSC USB3320-EZK High-Speed USB transceiver	27

5	J64	SD Card Interface connector	Molex 67840-8001 SDIO Memory card connector	22
6	U23	Programmable Logic JTAG Programming Options with integrated Micro-B connector	Digilent USB JTAG Module	15
7	U43	System Clock, 200 MHz, 2.5V LVDS oscillator	SiTime SIT9102-243N25E200.0000	30
8	U28, U65	Programmable User Clock and Processing System Clock Source	Silicon Labs SI570BAB0000544DG, default 156.250MHz, PS fixed 33 MHz clock	30
9	U35, P2	10/100/1000 MHz Tri-Speed Ethernet PHY, RJ45 w/magnetics	Marvell 88E1116RA0-NNC1C000, Halo HFJ11-1G01ERL	25–26
10	X1	Ethernet PHY Clock Source, 25.000 MHz	Epson MA-506-25.000m-CO:ROHS	25
11	DS6–DS8	Ethernet PHY User LEDs	Ethernet PHY User LEDs, GREEN	25
12	U36, J17	USB-to-UART Bridge, USB Mini-B connector	Silicon Labs CP2103GM, Molex 54819-0589	36
13	U40, P1	HDMI Video Output	Analog Devices ADV7511KSTZ-P HDMI transmitter, Molex 500254-1927 HDMI receptacle	28–29
14	U44	I2C Bus	TI PCA9548ARGER	32
15	U16	Real-Time Clock	Epson RTC-8564JE:3:ROHS	33
16	J54	I/O Expansion Header driven from I2C Expander U80	2-row pin header	33
17	DS15–DS22	User LEDs	GPIO LEDs, GREEN 0603	34
18	SW5, SW7	User Pushbuttons SW5 = Left, SW7 = Right	E-Switch TL3301EP100QG	34
19	SW12	GPIO DIP Switch	2-pole C&K SDA02H1SBD	34
20	SW11	Power On/Off Slide Switch	C and K 1201M2S3AQE2	47
21	U14	High Speed CAN Transceiver	NXP TJA1040T/VM	21
22	SW4	Program_B Pushbutton	E-Switch TL3301EP100QG	34
23	SW10, J2	Programmable Logic JTAG Select Switch, JTAG Cable Connector	2-pole C and K SDA02H1SBD MOLEX 87832-1420	15
24	J3, J4	FPGA Mezzanine (FMC) Card Interface	Samtec ASP_134486_01	23, 24
25	U32, U33, U34	Power Management (bottom and top of board)	TI UCD9248PFC in conjunction with various regulators	39–47
26	J40	XADC Analog-to-Digital Converter	2X10 0.-inch male header	31
27	SW1, SW2	PS Power-On and System Reset Pushbuttons	Panasonic EVQ-11L07K 14	35, 36
28	J62, J63	User PMOD GPIO Headers	J63 2 x 6 0.1 inch J63 1 x 6 0.1 inch male headers	34, 35
29	SW16	5-pole SPDT MIO DIP switch	CTS 206-125. See Table 1-2 for switch settings.	14
30	J59	2x5 shrouded PMBus connector (bottom of board)	ASSMAN HW10G-0202	47
31	J60	12V power input 2x6 connector	MOLEX 39-30-1060	47

RISC V IMPLEMENTATION IN FPGA BOARD



VERIFICATION USING UVM

5.1.1 Verification Technology – UVM

UVM(Universal Verification Methodology) is a Standard Verification Methodology which uses System Verilog constructs based on which a fully Functional Testbench can be built to verify functional correctness of Design Under Test(DUT). It is an IEEE standard/methodology based on System Verilog language.

Some of the Salient Features are

- A library of base classes for building testbench components (Agent, Sequencer, Driver, Monitor, Scoreboards, Environment class etc)
- Provides Verification phases for synchronizing concurrent process.
- Provides a reporting mechanism for a consistent way of printing and logging results.
- Provides a Factory, for constructing objects and substituting objects

5.1.2 Basic Test bench Functionality

The purpose of a test bench is to determine the correctness of the DUT. This is accomplished by the following steps.

- Generate stimulus
- Apply stimulus to the DUT
- Capture the response
- Check for correctness
- Measure progress against the overall verification goals

Some steps are accomplished automatically by the testbench, while others are manually determined by you. The methodology you choose determines how the preceding steps are carried out.

5.1.3 Methodology Basics

- Constrained-random stimulus
- Functional coverage
- Layered test bench using transactors
- Common test bench for all tests
- Test case-specific code kept separate from the testbench

CONCLUSION

In the proposed project, a 32-bit RISC V d processor is successfully designed, modelled, and simulated. With a design and Verilog HDL, it is effectively executed. It is determined that the functionality is correct by comparing the simulation's results to those that were predicted.

The 32 bit RISC V processor operation is implemented on ZYNQ ZEDBOARD 702 FPGA. Vivado Design Suite had used for synthesis and implementation.All transaction were verified as a collective operation of design using UVM environment in the QUESTA SIM simulation tool.

FUTURE SCOPE

RISC-V is already enabling unprecedented innovation in embedded design. From IoT devices and wearables to supercomputers and data centers, new hardware, software, and processors built for this open-source ISA are arriving on the market regularly. This only paves the way for the future of innovation across the full spectrum of computing.

Manufacturers and developers are continuing to embrace RISC-V and recognize its numerous benefits. As this community of like-minded people and organizations grows, we can expect a tidal wave of applications built using this flexible and scalable architecture in the near future. The excitement around this architecture is well-founded.

While being free is a huge draw for these developers and manufacturers, the real advantage of RISC-V is the freedom it brings with it. As RISC-V ushers in a new wave of collaboration in the silicon sector, industry-wide adoption might be just around the corner.

The RISC-V architecture represents a significant shift in the world of processor design, offering a flexible, modular, and extensible open-source ISA that can be tailored to specific applications and use cases. By adhering to key design principles such as reduced instruction set computing, modularity, and extensibility, RISC-V enables the development of processors that can deliver optimized performance, power consumption, and cost-effectiveness. With a growing ecosystem and community supporting its development and adoption, RISC-V has the potential to reshape the landscape of the semiconductor industry and drive innovation in a wide range of applications, from embedded systems and IoT devices to high-performance computing and artificial intelligence.

REFERENCES

1. https://riscv.org/technical/specifications/Unprivileged Specification version 20191213

2. International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com (ISSN 2250-2459, Volume 2, Issue 4, April 2012) 340 Design and Implementation of 5 Stages d Architecture in 32 Bit RISC Processor

3. https://www.wevolver.com/article/risc-v-architecture-a-comprehensive-guide-to-the-open-source-isa

4.https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISCVGreenCarv8-20151013.pdf

5. 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), 2016, pp. 1-7, doi: 10.1109/VLSI-SATA.2016.7593047.

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

7. S. Samiappa Sakthikumaran, V. S. Salivahanan and Kanchana Bhaaskaran, "16-Bit RISC processor design for convolution application", *International Conference on Recent Trends in Information Technology (ICRTIT)*.