

PERFORMANCE METRICS OF 16-BIT VLSI RISC ARCHITECTURE

A Project Report Phase-II

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering

by

G. Surya Teja (39130156)

J. Sai Teja (39130177)



**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
SCHOOL OF ELECTRICAL AND ELECTRONICS**

SATHYABAMA

**INSTITUTE OF SCIENCE AND
TECHNOLOGY(DEEMED TO BE
UNIVERSITY)**

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI -600

119APRIL - 2023



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600 119

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of G . Surya Teja (39130156), J.Saiteja (39130177) who carried out the project entitled "Performance of Metrics of 16-Bit VLSIRISC Processor" under our supervision from November 2022 to April 2023.

Internal Guide

Dr. M. SUBRAMONIAM, M.E., Ph.D.,

Head of the Department

Dr. T. RAVI, M.E., Ph.D.,

Submitted for Viva voce Examination held on 24.04.2023

Internal Examiner

External Examiner

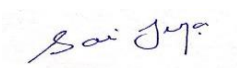
DECLARATION

We, **SURYA TEJA GUNDLAPUDI (39130156)** and **SAI TEJA JAMPULA(39130177)** here by declare that the Project Report entitled "**PERFORMANCE METRICS OF 16-BIT RISC ARCHITECTURE**" done by us under the guidance of **Dr. M . SUBRAMONIAM, M.E., Ph.D.**, Associate Professor, Dept. of Electronics and Communication Engineering, Sathyabama institute of science and technology , Chennai, submitted partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Electronics and Communication Engineering.

DATE :24..4.23

PLACE :Chennai

SIGNATURE OF CANDIDATES

1. 
2. 

ACKNOWLEDGEMENT

We are pleased to acknowledge my/our sincere thanks to the Board **of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I/We are/are grateful to them.

I/We convey my/our thanks to **Dr. N. M. NANDHITHA, M.E., Ph.D., Professor & Dean, School of Electrical and Electronics** and **Dr. T. RAVI, Ph.D., Professor & Head, Department of Electronics and Communication Engineering** for providing me/us necessary support and details at the right time during the progressive reviews.

I/We would like to express my sincere and deep sense of gratitude to my/our **Project Guide Dr.M.SUBRAMONIAM, M.E., Ph.D.**, Associate professor, Dept. of Electronics and Communication Engineering, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of our project work.

We wish to express our thanks to all Teaching and Non-teaching staff members of the Department of Electronics and Communication Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

RISC stands for Reduced Instruction Set Computer. In Reduced Instruction Set Computer (RISC) architecture, the instruction set of the computer is simplified to reduce the execution time. RISC has a small set of instructions, which generally include register-to-register operations. Thus, data is stored in processor registers for computations, and results of the computations are transferred to the memory using store instructions. All operations are performed within the registers of the CPU. In RISC, all instructions have simple register addressing and hence use less number of addressing modes. RISC uses a relatively simple instruction format and is easy to decode. Here, the instruction length can be fixed and aligned on word boundaries. The RISC processors can execute one instruction per clock cycle. This is done using pipelining, which involves overlapping the fetch, decode, and execute phases of two or three instructions. As RISC takes relatively a large number of registers in the processor unit, it takes less time to execute its program when compared to CISC. This project includes the designing of a 16-Bit RISC processor and modeling of its components using Verilog HDL. The processor is based on Harvard architecture. The instruction set adopted here is extremely simple and gives an insight into the kind of hardware which should be able to execute the set of instructions properly. Along with sequential and combinational building blocks of a processor such as adders and registers more complex blocks such as ALU and memories have been designed and simulated.

KEY WORDS : RISC Processor,CISC Processor,Registers,Fetch ,Decode ,CPU , Instruction Set , ALU(Arithmetic logical unit) , Von Neumann Architecture ,Harvard Architecture, Operations, Pipelining.

TABLE OF CONTENTS

Chapter No	TITLE	Page No.
	ABSTRACT	V
	LIST OF FIGURES	viii
1	INTRODUCTION	
	1.2 Types of processors	1
	1.3 Processor Architectures	3
	1.4 Classification of Microprocessors	4
	1.5 Advantages of Processors	6
	1.6 Applications of Processor	9
2	LITERATURE SURVEY	12
	2.1 Inferences from Literature Survey	14
	2.2 Open Problems in existing systems	
	2.3 Proposed System	
3	ARCHITECTURE DESIGN OF PROCESSOR BY RISC	16
	3.1 Arithmetic and logical unit (ALU)	16
	3.2 Control Unit	21
	3.3 Program Counter	23
	3.4 Instruction Memory And Data Memory	25
	3.5 Pipelining	26

4	TOOLS USED	28
4.1	Software Tool : Verilog	28
4.2	XILINX ISE	33
4.3	XILINX VIVADO	34
5	DESIGN OF 16-BIT PROCESSOR	35
5.1	Design Methodology	35
5.2	Architecture of Design	36
5.3	Data Path	37
5.4	Control Unit	38
6	SIMULATION RESULTS	40
7	CONCLUSION	44
7.1	Conclusion	44
7.2	Summary	44
8	REFERENCES	45

APPENDIX

1.	SCREENSHOTS	51
2.	RESEARCH PAPER	57
3.	PLAGARISM REPORT	62

LIST OF FIGURES

FIGURE NO	TITTLE	PAGE NO
Figure 1.1	Parts Of Processor Fit Together	2
Figure 1.2	Von Neumann Architecture	5
Figure 1.3	Harvard Architecture	5
Figure 3.1	Arithmetic And Logic Unit	16
Figure 3.2	Arithmetic And Logic Unit Operations	18
Figure 3.3	Flow Diagram Of ALU	20
Figure 3.4	Control Unit Flow Diagram	21
Figure 3.5	Flow Diagram Of Program Counter	24
Figure 3.6	Pipelining Of 5 Instructions	27
Figure 5.1	Design Methodology Flow Diagram	35
Figure 5.2	16 Bit Processor Architecture Proposed work	36
Figure 6.1	Simulation Output Screenshot Of RISC Processor	40
Figure 6.2	16-Bit Registers Timing Diagram	41
Figure 6.3	Total ALU Operations Result	42

	Screenshot	
Figure 6.4	Simulation output of multiplexers	43

CHAPTER- 1

INTRODUCTION

1.1 INTRODUCTION

Processors are the heart of all "smart" devices, whether they be electronic devices or otherwise. Their smartness comes as a direct result of the decisions and controls that the processor makes. For example, one can usually not consider a car to be an electronic device. However, it certainly has many complex, smart electronic systems, such as the anti-lock brakes and the fuel-injection systems. Each of these systems is controlled by a processor. There are generally two types of processors: general-purpose processors and dedicated processors. General purpose processors such as the Pentium CPU can perform different tasks under the control of software instructions. General-purpose processors are used in all personal computers. Dedicated processors also known as application-specific integrated circuits (ASICs), on the other hand, are designed to perform just one specific task. For example, inside the cell phone, there is a dedicated processor that controls its entire operation. The embedded processor inside the cell phone does nothing, but controls the operation of the phone. Dedicated processors are, therefore, usually much smaller and not as complex as general-purpose processors. The different parts and components fit together to form the processor shown in fig.1.1[1]. From transistors, the basic logic gates are built. Logic gates are combined together to form either combinational circuits or sequential circuits. The difference between these two types of circuits is only in the way the logic gates are connected together. Latches and flip-flops are the simplest forms of sequential circuits, and they provide the basic building blocks for more complex sequential circuits. Certain combinational circuits and sequential circuits are used as standard building blocks for larger circuits, such as the processor. These standard combinational and sequential components usually are found in standard libraries and serve as larger building blocks for the processor. Different combination and sequential components are connected together to form either the data path or the control unit of a processor.

Finally, combining the data path and the control unit together. It will produce the circuit for either a dedicated or a general processor. Parts of the processor which are fitted together are shown in the figure 1.1.

However, they are used in every smart electronic device such as musical greeting cards, electronic toys, TVs, cell phones, microwave ovens, and anti-lock brake systems in car. Although the small dedicated processors are not as powerful as the general-purpose processors they are being sold and used in a lot more places than the powerful general-purpose processors that are used in personal computers.

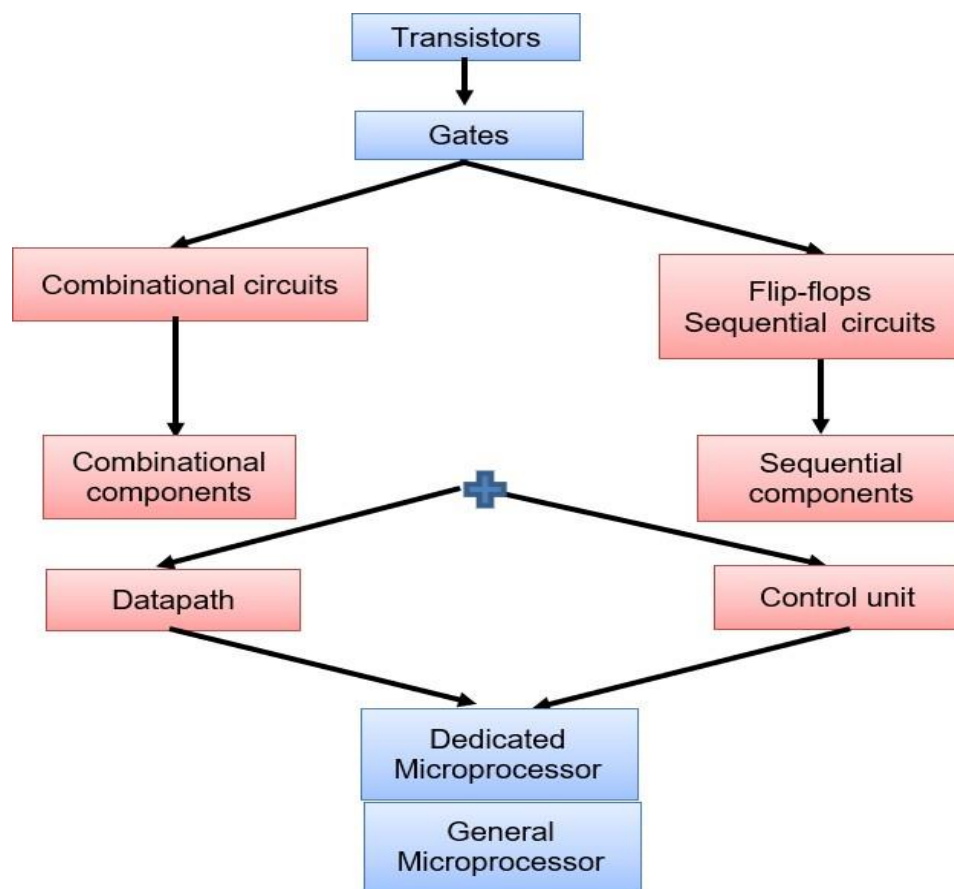


Fig:1.1: Parts of processor fit together

1.2 TYPES OF PROCESSORS

Processors are classified by the semiconductor technology of their design (**TTL** (Transistor-Transistor Logic), CMOS (Complementary Metal Oxide Semiconductor), or ECL (Emitter Coupled Logic)); by the width of data format (4-bit, 8-bit, 16-bit, 32-bit, or 64-bit) they process; and by their instruction set (CISC (Complex Instruction Set Computer), or RISC (Reduced Instruction Set Computer)). **TTL** technology is most commonly used while CMOS is favored for portable computers and other Battery-powered devices because of its low power consumption. ECL is used where the need for its greater speed offsets the fact that it consumes the most power. Four-bit devices, while inexpensive and are good only for simple control applications. In general, the wider the data format, the faster and more expensive the devices. Processors such as 8-bit, 16-bit and 32-bit processors are classified depending upon the demand of performance, cost, power and programmable. 8-bit processors have extremely low cost and consume less power for simple control systems. 16-bit processors have high performance and power than 8-bit processors and lower power consumption than 32-bit processors. They are often used in 16-bit applications such as disk driver controllers, cellular communication and airbags. In contrast 32-bit processors have high programmable, high performance and are widely used in cellular phones and PDA (Personal Digital Assistant) that need high computation but have high power consumption.

CISC processors which have 70 to several hundred instructions are easier to program than RISC processors but are slower and more expensive. RISC has simple instructions few in number, fixed length instructions, complexity in compiler, Only LOAD/STORE instructions access memory and few addressing modes. CISC has many complex instructions, variable length instructions, complexity in microcode, many instructions can access memory and many addressing modes. The difference between RISC and CISC becomes evident through the basic computer performance

equation. RISC systems shorten execution time by reducing the clock cycles per instruction. CISC systems shorten execution time by reducing the number of instructions per program.

1.3 PROCESSOR ARCHITECTURES

A computer architecture is a detailed specification of the computational, communication, and data storage elements (hardware) of a computer system, how those components interact (machine organization), and how they are controlled (instruction set). A processor chip's architecture is a description of its fundamental parts and functions. Each family of processors has a unique architecture. A programming view of a particular processor's architecture is provided through assembly language. Every processor type has a unique assembly language. A machine's architecture determines which computations can be performed most efficiently, and which forms of data organization and program design will perform optimally. We studied various processor architectures. They are used according to different applications.

A. Von Neumann Architecture

The Von Neumann architecture is a design model for a stored-program Digital Processor that uses a processing unit and a single storage to hold both instructions and data. The Von Neumann architecture consists of an arithmetic unit, a programme control unit, a single bus for memory access, and a single shared memory for programmes and data. The substantial fetching and execution cycles of the Von Neumann processor A stored-program processor is one that keeps its programmed instructions, as well as its data, in read-write, random access memory (RAM). As we know that many Intel's processors are based on Neumann architecture, like Intel 8085, 8088 etc. This particular sort of digital computer architecture is built around the idea of computers with stored programmes. The Neumann architecture (3)

is shown in figure 1.3.

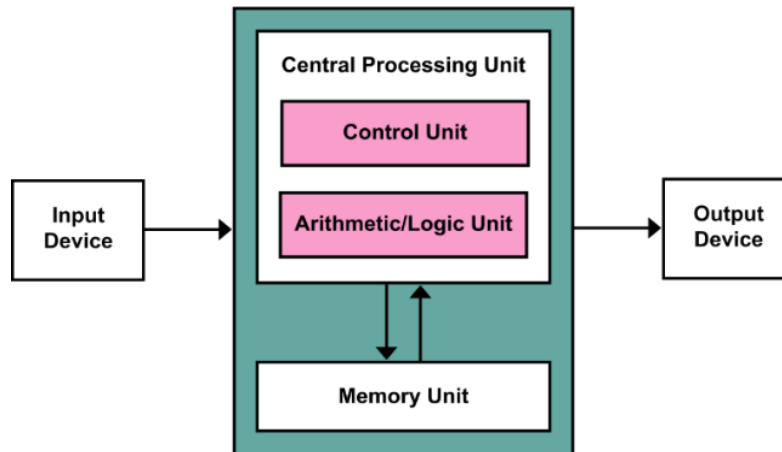


FIG:1.2: VON NEUMANN ARCHITECTURE

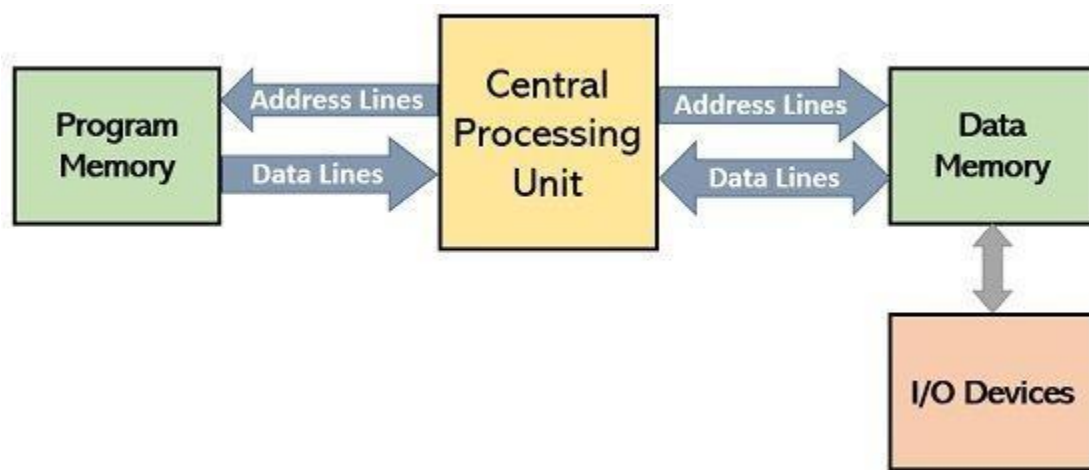


FIG:1.3: HARVARD ARCHITECTURE

The Harvard architecture is Processor architecture with physically separate storage and signal pathways for instructions and data. In Harvard architecture, instructions can be stored in read-only memory while data memory generally requires read-write memory. **Fig:1.3** shows that in some systems, there is much more instruction memory than data memory so instruction addresses are wider than data addresses. Basically

Digital Signal Processor (DSP) based on Harvard architecture.

B. Super Harvard Architecture

It is an advanced version of Harvard architecture, same as Harvard having two separate storage for Data and instruction, but the only difference is additional memory called Cache memory. Cache memory (also called buffer memory) is local memory that reduces waiting times for information stored in the RAM (Random Access Memory). In effect, the computer's main memory is slower than that of the processor. There are, however, types of memory that are much faster, but which have a greatly increased cost. The solution is therefore to include this type of local memory close to the processor and to temporarily store the primary data to be processed in it. Recent model computers have many different levels of cache memory: Dual memory and Cache memory. Figure 1.5 shows the operation of Super Harvard architecture.

1.4 CLASSIFICATION OF MICROPROCESSORS

Besides the classification based on the word length, the classification is also based on the architecture i.e. Instruction Set of the microprocessor. These are categorized into RISC and CISC.

A. RISC:

RISC processors were first introduced in the 1980s as a response to the complexity of CISC processors. RISC processors use a simplified instruction set that makes the design of the processor simpler and more efficient. In RISC architecture, each instruction performs a single task, such as a load or a store operation. RISC processors typically have fewer instructions than CISC processors, but they execute them faster.

The Reduced Instruction Set Computing (RISC) architecture is a type of processor design that emphasizes simplicity and efficiency. A smaller set of instructions very quickly, rather than supporting a large set of instructions that may be rarely used. The main features of RISC processors include: A small set of simple

instructions:

RISC processors typically have a small set of instructions that can be executed quickly. This simplifies the design of the processor and makes it easier to optimize for speed.

1.Load-store architecture: RISC processors typically use a load-store architecture, which means that instructions only operate on data that is in registers. Memory access is done using specific load and store instructions.

2.Pipelined execution: RISC processors are often designed to execute instructions in a pipelined fashion, where each stage of the instruction execution is done in a separate pipeline stage. This can increase the throughput of the processor and improve overall performance.

3.Fixed-length instructions: RISC processors typically use fixed-length instructions, which makes it easier to decode and execute instructions quickly.

4.Register-oriented: RISC processors often have a large number of registers that are directly accessible by instructions. This can improve performance by reducing the need to access memory for temporary storage.

Example: SPARC,MIPS, ARM,PowerPC

B. CISC:

CISC processors were first introduced in the 1970s as a way to make programming easier for computer programmers. CISC processors use a complex instruction set that allows each instruction to perform multiple tasks. This approach reduces the number of instructions needed to perform a task, which simplifies programming.

CISC processors use a memory-to-memory architecture, which means that data can be processed directly from memory, without being loaded into registers first. This

approach reduces the number of memory accesses needed. CISC processors also use microcode to implement the instruction set, which simplifies the design of the processor. One of the key advantages of CISC processors is their ability to handle complex calculations efficiently. CISC processors are also highly versatile and can be used in a wide range of applications, from desktop computers to servers to embedded systems.

Examples: Intel architecture, AMO

Some of the advantages of CISC processors are:

1. Versatility: CISC processors are highly versatile and can handle a wide range of tasks efficiently. They are commonly used in desktop computers, servers, and embedded systems.

2. Efficiency for complex calculations: CISC processors can perform complex calculations efficiently due to their large and complex instruction set. They can perform multiple operations in a single instruction, which reduces the number of instructions needed and improves performance.

3. Code density: CISC processors have a higher code density than RISC processors, which means that less memory is required to store the same program. This is because CISC instructions are more complex and can perform more tasks in a single instruction.

4. Easy programming: CISC processors use a complex instruction set, which makes programming easier for developers. This is because each instruction can perform multiple tasks, reducing the number of instructions needed and simplifying programming.

5. Memory access: CISC processors use a memory-to-memory architecture, which allows data to be processed directly from memory without the need to load it into registers first.

This can improve performance by reducing the number of memory accesses needed.

CISC	RISC
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design – the ISA does as much as possible using hardware circuitry	Software-centric design – High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (can cause bottlenecks if RAM is limited)
Complex and variable length instructions	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

TABLE:1.1: Difference Between RISC And CISC

1.5 ADVANTAGES OF PROCESSORS

1. Less Power Consumption: These processors have low power consumption and dissipation. This is a very attractive quality for embedded systems.

Low power consumption should be a low percentage of the power consumption of the node in standby without WuR. It again gives two advantages: Much longer use of battery. No need for cooling devices.

2. Simpler hardware: All the components of the processor are fabricated on the same chip. Smaller chips allow a semiconductor manufacturer to place more parts on a single silicon wafer, which can lower the per-chip cost dramatically.

3. Less area: The chip area, dedicated to realization of control unit, the so called control unit is considerably reduced. Therefore, there is more area available for more features. There is a higher chance of fitting a whole CPU and some additional features on a chip.

1.6 APPLICATIONS OF PROCESSOR

1. Single-board Microcomputers:

These are the simplest and cheapest general-purpose processor-based systems with the minimum possible hardware and software configuration. Primarily, these are used for two purposes. In universities and educational institutions they are used for imparting training to the student. In industries these are either used for evaluation of the processor or for building prototype systems.

2. Terminals:

Terminals are used for communication between a computer system and its user. Commands and data to be communicated to the computer are entered with the help of a typewriter-like keyboard and the response of the computer is displayed on a television-like screen known as cathode rays tube (CRT)

3. Personal computers:

Personal computer range from very inexpensive home computers to moderately costly business and intelligent workstations. 8-bit processors are primarily used for playing video games and learning simple programming and 16-bit processors are used for a variety of purposes such as word processing, payroll, business accounts,

medical record keeping and inventory control.

4.Super minis and CAD Machines:

As a result of the advancement of solid state technology, processor can perform powerful functions, and their word size and memory capacity has also increased. Processors have been used to build powerful microcomputers with capabilities and features resembling traditional mini and mainframe computers. These systems are used as an engineering workstation and computer aided design (CAD) machines.

5.Consumers:

The use of microprocessors in toys, entertainment equipment and home appliances is making them more entertaining and easy to use and full of novel features.

6. Instrumentation:

By making an instrument processor based frequency meters, frequency synthesizers, function generator, spectrum analyzer and many other instruments are available where processors are used as controllers. They are also used in medical instrumentation, e.g., patient monitoring in intensive care units (ICU), pathological analysis and the measurements of parameters like blood pressure and temperature.

CHAPTER-2

LITERATURE SURVEY

2.1 INFERENCES FROM LITERATURE SURVEY

Presented in the 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT) by Jikku Jeemon. The paper available on IEEE Xplore, presents a low power and simple implementation of an 8-bit RISC processor design on a Spartan-6 SP605 Evaluation Platform FPGA using Verilog HDL. The processor is designed using Harvard architecture, having separate instruction and data memory. It has an instruction set of only 29 instructions. The proposed processor is physically verified on Xilinx Spartan-6SP605 Evaluation Platform.

This project includes the designing of 16- Bit RISC processor and modeling of its components using Verilog HDL. The processor is based on Harvard architecture. The instruction set adopted here is extremely simple and gives an insight into the kind of hardware which should be able to execute the set of instructions properly. Along with sequential and combinational building blocks of a processor such as adders and registers more complex blocks such as ALU and memories have been designed and simulated.

The modeling of ALU which has been done in this project is fully structural starting from half adders. At the end the semi-custom layout was developed for ALU. Complex blocks such as memories have been modeled using a behavioral approach, whereas simple blocks such as adders have been done through a structural approach. The tools which have been used throughout the project work are ModelSim and Intel Quartus prime 18.1. For synthesis purposes the targeted FPGA device technology is ALTERA.

This paper presents the design and implementation of a low-power 16-bit RISC processor using VHDL and FPGA.

The proposed processor architecture is based on a Harvard architecture with separate instruction and data memory, and includes a three-stage pipeline, a custom instruction set, and power optimization techniques such as clock gating and data gating. The processor was implemented on a Xilinx Spartan-3 FPGA, and the results show that the processor operates at a clock frequency of 50MHz and consumes only 32mW of power. The performance of the processor was evaluated using a set of benchmark programs, and the results demonstrate that the processor can execute the benchmark programs with a reasonable speed and efficiency, while consuming significantly less power compared to other existing processors. The proposed low-power 16-bit RISC processor is suitable for use in low-power embedded systems, such as wireless sensor networks, portable devices, and Internet of Things (IoT) applications.

RISC (Reduced Instruction Set Computer) is a design methodology which supports small and simple sets of instructions that all require the same amount of time to execute. A design of a 16-bit RISC processor is presented in this paper, using the Verilog HDL. The RISC processor comprises the blocks mainly ALU, Controller, Register files, and Data memory unit. The proposed processor was carried out with the help of van Neumann architecture, having only one shared memory for instruction and data. The proposed RISC processor which can be very simple and clean to investigate contains 15 instructions. Modelsim is used for logical verification. The proposed RISC processor is analyzed using the Xilinx ISE tool.

These RISC or Reduced Instruction Set Computers are a design philosophy that has become a mainstream in Scientific and engineering applications. The main objective of this paper is to design and implement a 32 - bit RISC (Reduced Instruction Set Computer) processor using XILINX VIRTEX4 Tool for embedded and portable applications. The design will help to improve the speed of the processor, and to give the higher performance of the processor. The most important feature of the RISC processor is that this processor is very simple and supports load/store

architecture. The important components of this processor include the Arithmetic Logic Unit, Shifter, Rotator and Control unit. The module functionality and performance issues like area, power dissipation and propagation delay are analyzed using the Virtex4 XC4VLX15 **XILINX** tool.

2.2 OPEN PROBLEMS IN EXISTING SYSTEMS

The problems in the existing system according to the research papers that have been presented in the literature survey.

Research Paper - 1:

According to simple implementation of an 8-bit RISC processor design on a Spartan-6 SP605, 8-bit RISC processors have a limited instruction set which can make it difficult to perform complex operations, making them less suitable for applications that require high processing power, and can only address up to 256 bytes of memory, which can limit the amount of data that can be processed at one time. The accuracy of the 8-bit RISC processor is limited due to the small number of bits they use to represent data, which can make them less suitable for applications that require high precision.

Research paper - 2:

This implementation of 16-bit RISC processor is based designed based on the structural modeling and the main drawbacks of structural modeling involves defining a design in terms of interconnected instances of logic gates and other lower-level building blocks, and structural modeling can be time-consuming, as each gate and flip-flop must be manually connected to form the desired functionality. This can lead to long design times, especially for large and complex designs like a 16-bit RISC architecture. Structural modeling can also make testing difficult, as the design is defined in terms of lower-level building blocks rather than functional behavior. This

can make it challenging to write test benches that cover all possible scenarios.

Research paper - 3:

This implementation of 16-bit RISC processor architecture using VHDL has some drawbacks like VHDL is a verbose language, and it requires a lot of syntax and rules to be followed, which can make it more difficult for designers who are not familiar with the language to understand. VHDL has less support for code reuse compared to Verilog, which has better support for building modular designs. This makes it more difficult to create reusable code in VHDL, resulting in longer design times. VHDL has limited support for simulation when compared to other hardware description languages, which can make it difficult to perform functional verification of the design.

Research paper - 4:

In this proposed methodology of 16-bit RISC processor architecture using Von-Neumann has a drawback of slower performance. Since the instruction and data must be accessed sequentially in the unified memory, it can result in slower instruction execution and also the unified memory can result in a bottleneck for processor performance, as the memory bus is shared between the instruction and data accesses. The Von Neumann architecture can result in simpler implementation and lower cost but may have slower performance and a memory bottleneck.

Research paper -5:

This implementation of 32-bit RISC processors are more complex than 8-bit or 16-bit processors, which can make them more difficult to design, debug, and maintain and 32-bit RISC processors require more memory to store instructions and data, which can be a disadvantage in applications with limited memory resources. The larger instruction set of 32-bit RISC processors can result in longer instruction latencies,

which can impact the overall performance of the processor. The cost of implementing a 32-bit RISC processor can be higher than that of implementing an 8-bit or 16-bit processor, which can be a disadvantage for applications with strict cost constraints.

2.3 Proposed System

This project includes the designing of 16- Bit RISC processor and modeling of its components using Verilog HDL. The processor is based on Harvard architecture. The instruction set adopted here is extremely simple that gives an insight into the kind of hardware which should be able to execute the set of instructions properly. Along with sequential and combinational building blocks of a processor such as adders and registers more complex blocks such as ALU and memories have been designed and simulated.

The Reduced Instruction Set Computer or RISC is a microprocessor design principle that favours a smaller and simpler set of instructions that all take same amount of time to execute. RISC architecture is used across a wide range of platforms from cellular phones to super-computers. In this paper the behavioural design and functional characteristics of 16- bit RISC processor is proposed, which utilizes minimum functional units without compromising in performance. The design is based on Harvard architecture having separate data memory and instruction memory. The instruction word length is 24-bit wide. The processor supports 16 instructions with three addressing modes. It has 16 general purpose registers. Each register can store 16-bit data. The processor has 16-bit ALU capable of performing 11 arithmetical and logical operations. The processor also incorporates a flag register which indicates carry, zero and parity status of the result. All the modules in the design are coded in Verilog. The individual modules are designed and tested at each level of implementation and finally integrated in a top level module by appropriate mapping. The design entry and synthesis is done using Xilinx ISE 10.1 tool and simulation results are verified using Modelsim 10.

CHAPTER-3

ARCHITECTURAL DESIGN OF PROCESSOR BY RISC

DESIGN OF MODULES

This contains the design of important processor modules such as ALU, data path and control circuit. After designing, simulation and synthesis of each block is carried.

3.1 ARITHMETIC AND LOGICAL UNIT (ALU):

In the computer system, ALU is a main component of the central processing unit, which stands for arithmetic logic unit and performs arithmetic and logic operations. It is also known as an integer unit (IU) that is an integrated circuit within a CPU or GPU, which is the last component to perform calculations in the processor.

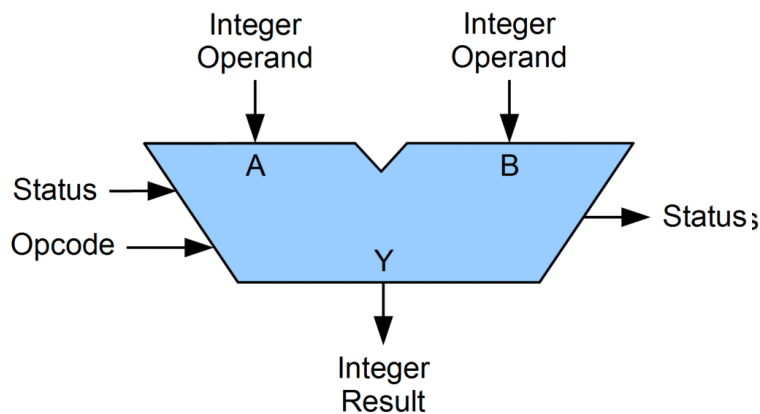


FIG:3.1: ARITHMETIC AND LOGIC UNIT

A.FUNCTIONAL DESCRIPTION:

The arithmetic logic unit (ALU) is the main block of the processor which performs most of the instructions. It has two operands, data1 and data2, and it is designed to perform the arithmetic and logical operations.

The result of the execution of instruction is written to the output . Designing of ALU is done for arithmetic operation such as addition, subtraction, multiplication, increment, decrement, addition with carry, subtraction with borrow etc. and basic logic operation such as NOT, AND, OR, NANO, NOR, XOR, XNOR etc. The inputs are 16-bit wide with type unsigned. A detection of carry and borrow is also supported.

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU)

Typically, the ALU has direct input and output access to the processor controller, main memory (random access memory or RAM in a personal computer) and input/output devices. Inputs and outputs flow along an electronic path that is called a bus. The input consists of an instruction word, sometimes called a machine instruction word, that contains an operation code or "opcode," one or more operands and sometimes a format code. The operation code tells the ALU what operation to perform and the operands are used in the operation.

For example, two operands might be added together or compared logically. The format may be combined with the opcode and tells, for example, whether this is a fixed-point or a floating-point instruction. The output consists of a result that is placed in a storage register and settings that indicate whether the operation was performed successfully. If it isn't, some sort of status will be stored in a permanent place that is sometimes called the machine status word.

ALU is made up of three different sorts of functional components: storage registers, operations logic, and sequencing logic. The inputs and outputs of the ALU are linked to the programme execution control unit and cache memory, among other CPU functional units.

BLOCK DIAGRAM:

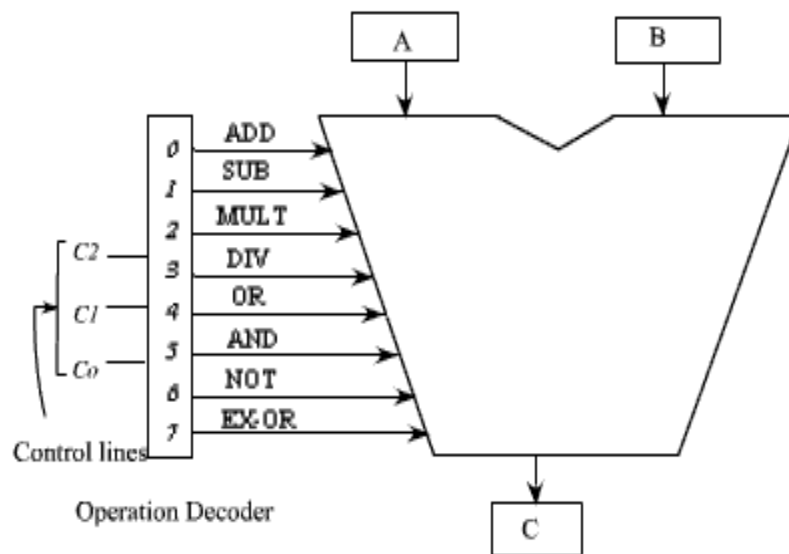


FIG:3.2: ARITHMETIC AND LOGIC UNIT OPERATIONS

B.ALU OPERATIONS:

The Arithmetic Logic Unit (ALU) is a fundamental component of a processor that performs arithmetic and logical operations on binary data. In a 16-bit RISC processor architecture, the ALU typically supports a set of operations including arithmetic, logical, shift, compare, and branch operations. These operations are performed on 16-bit data and enable the processor to execute a variety of instructions to perform complex computations. Efficient implementation of the ALU is critical to the overall performance and power consumption of the processor. In this report, we will discuss in-depth the design methodology of the data path of a 16-bit RISC processor using Verilog, with a specific focus on the ALU design and its operations.

1.Arithmetic Operations:

The arithmetic operations performed by the ALU include addition, subtraction, increment, and decrement.

These operations are typically performed using the two's complement representation of numbers. In addition and subtraction, the carry bit and overflow bit are important in detecting overflow conditions. The increment and decrement operations are simple additions or subtractions of 1 to the input operand.

2. Logical Operations:

The logical operations performed by the ALU include AND, OR, XOR, and NOT. These operations are performed bitwise on the operands. The AND and OR operations are used in logical comparisons, whereas the XOR operation is used in cryptography and error detection. The NOT operation is a simple inversion of the bits of the input operand.

3. Shift Operations:

The shift operations performed by the ALU include logical shift left, logical shift right, arithmetic shift left, and arithmetic shift right. The logical shift operations shift the bits of the operand to the left or right, filling the empty bits with zeros. The arithmetic shift operations shift the bits of the operand to the left or right, filling the empty bits with the sign bit of the operand.

4. Compare Operations:

The compare operations performed by the ALU include equal, less than, and greater than. A comparison operation specifies how the system should compare an incoming value that has been supplied as a context parameter to an expression with a preset value or value range. In addition to being supplied via a context parameter, the test parameter may also be the output of a nested expression. These operations compare two operands and set the result bit to 1 if the condition is true, and 0 otherwise. The compare operations are used in conditional branching instructions.

5. Branch Operations:

The branch operations performed by the ALU include conditional branch if equal, branch if not equal, branch if less than, and branch if greater than. These operations allow the processor to jump to a new instruction based on the condition set by the compare operation.

The ALU is a critical component in the design of a processor, and the design of the ALU can have a significant impact on the performance and power consumption of the processor. Therefore, it is important to carefully select the ALU operations and implement them efficiently in hardware.

Mnemonic	Name and Format	Opcode (binary)	Assembly Format	Action
add	Add RRR-type	000	add rA, rB, rC	Add contents of regB with regC , store result in regA .
addi	Add Immediate RRI-type	001	addi rA, rB, imm	Add contents of regB with imm , store result in regA .
nand	Nand RRR-type	010	nand rA, rB, rC	Nand contents of regB with regC , store results in regA .
lui	Load Upper Immediate RI-type	011	lui rA, imm	Place the 10 ten bits of the 16-bit imm into the 10 ten bits of regA , setting the bottom 6 bits of regA to zero.
sw	Store Word RRI-type	101	sw rA, rB, imm	Store value from regA into memory. Memory address is formed by adding imm with contents of regB .
lw	Load Word RRI-type	100	lw rA, rB, imm	Load value from memory into regA . Memory address is formed by adding imm with contents of regB .
beq	Branch If Equal RRI-type	110	beq rA, rB, imm	If the contents of regA and regB are the same, branch to the address $PC+1+imm$, where PC is the address of the beq instruction.
jlr	Jump And Link Register RRI-type	111	jlr rA, rB	Branch to the address in regB . Store $PC+1$ into regA , where PC is the address of the jlr instruction.

FIG:3.3: ALU Operation Table

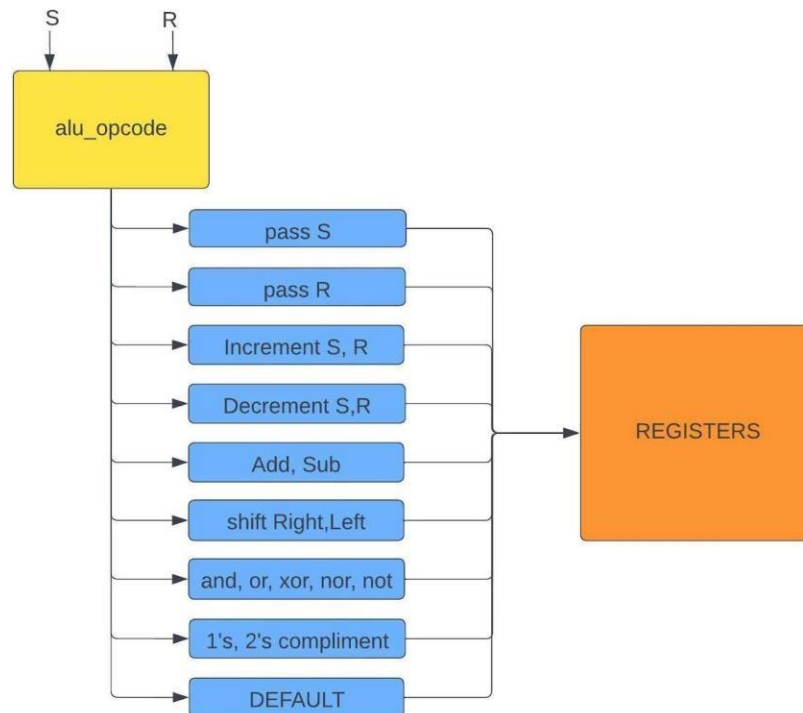


FIG:3.4: Flow Diagram Of ALU

3.2 CONTROL UNIT

The design of the control unit is described using an algorithmic state machine as shown in Figure (2). The ASM is designed to perform the logical, arithmetic and shifting operations. instruction fetch, decode and execute. The first phase i.e Instruction fetch has two states S_f1 and S_f2. When reset signal is high the controller goes to state S_f1, the contents of the program counter (PC) will be transferred to Add Reg in this state through the Bus1 and Bus2 by selecting proper select lines for mux1, mux2 and the controller goes to next state S_f2. In the next state, instruction register (IR) will be loaded and the contents of the memory addressed by the Add Reg through Bus2 and the program counter (PC) is incremented subsequently, and the controller goes to the next state i.e instruction decode (S_dec3) state. In this state instruction decoding takes place, if the instruction decoded is NOP then the controller goes back to instruction fetch, else other instruction would be executed. The circuit is simulated in **XILINX** and its simulation results.

A. FUNCTIONAL DESCRIPTION

The control unit is a sequential circuit in which its outputs are dependent on both its current and past inputs. This history of past inputs is stored in the state memory and is said to represent the state of the circuit. Thus, the circuit changes from one state to the next when the content of the memory changes. Depending on the current state of the circuit and the input signals, the next state logic will determine what the next state ought to be by changing the content of the state memory. Hence, a sequential circuit executes by going through a sequence of states. Since the state memory is finite, therefore the total number of different states that the circuit can go to is also finite. This is not to be confused with the fact that the sequence length can be infinitely long. However, because of having only a finite number of states, a sequential circuit is also referred to as a finite-state machine (FSM).

The control unit inside the microprocessor is thus an example of a finite-state machine. By stepping through a sequence of states, the control unit controls the operations of the data path. For each state that the control unit is in, the output logic that is inside the control unit will generate all the appropriate control signals for the data path to perform one operation. The speed in which the finite-state machine sequences through the states is determined by the clock signal. At each active edge of the clock signal, the state memory register is enabled and the next state value is stored in. The limiting factor for the clock speed is whether all the operational units inside the data path can finish their operations within one clock period. Figure 1.2 shows the block diagram of the control unit. The control unit is responsible for controlling all the operations of the data path by providing appropriate control signals to the data path at the appropriate time. At any one time, the control unit is said to be in a certain state as determined by the content of the state memory. The state memory is simply a register with one or more (D) flip-flops. The control unit operates by transitioning from one state to another – one state per clock cycle. Depending on the current state, the control inputs and the status signals, the next-state logic in the control unit will determine what state to go to next in the next clock cycle.

Thus, the control unit is also referred to as a finite-state machine (FSM). In every state, the output logic that is in the control unit generates all the appropriate control signals for controlling the data path. The data path, in return, provides status signals for the next state logic. Upon completion of the computation, the control output line is asserted to notify external devices that the value on the data output lines is valid.

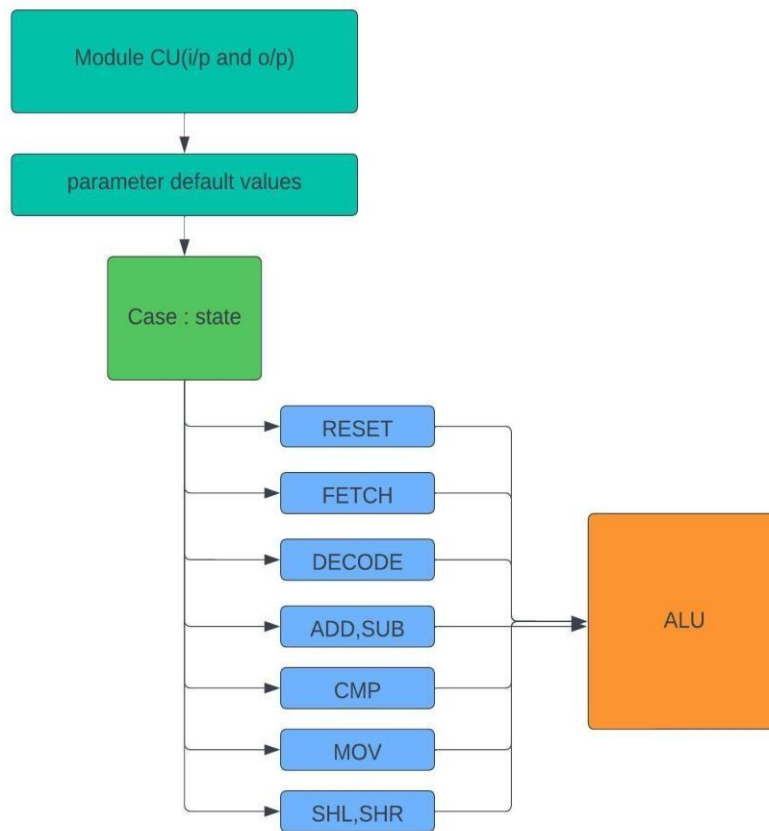


FIG:3.4: Control Unit Flow Diagram

3.3 PROGRAM COUNTER

The Program Counter (PC) is a register in a processor that holds the address of the next instruction to be fetched from memory. In a 16-bit RISC processor architecture.

The functional description of the PC in a 16-bit RISC processor can be described as follows:

1.Initialization: When the processor is powered on, the PC is initialized to a known value, typically the starting address of the program.

2.Instruction fetch: The PC is used to address the memory location of the next instruction to be fetched. The contents of this memory location are loaded into the instruction register, and the PC is incremented by 2 to point to the next instruction.

3.Branching: The PC is also used for branching instructions, which allow the processor to jump to a different instruction based on a condition. In a conditional branch instruction, if the condition is true, otherwise the PC is incremented by 2 to fetch the next sequential instruction.

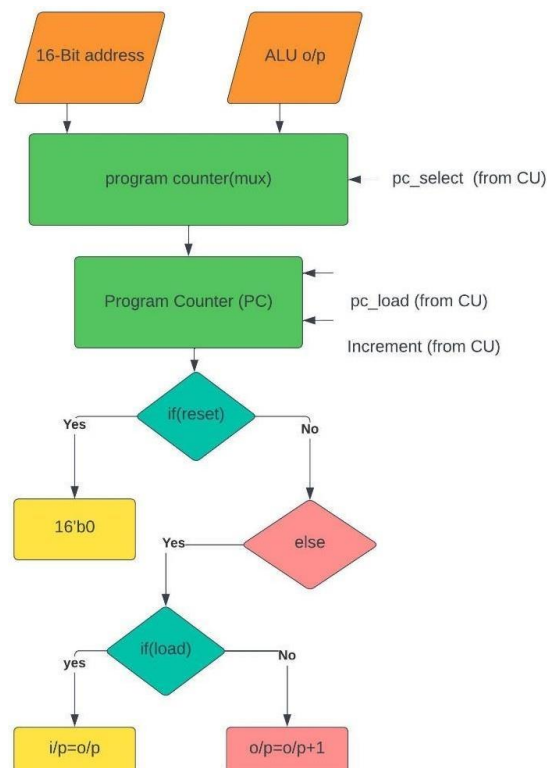


FIG:3.5: Flow Diagram Of Program Counter

4. Interrupt handling: When an interrupt occurs, the PC is used to save the address of the interrupted instruction. After the interrupt is handled, the PC is loaded with this address to resume execution of the program.

5. Return from subroutine: When a subroutine is called, the PC is used to save the return address. When the subroutine is complete, the PC is loaded with this address to return to the main program.

3.4 INSTRUCTION MEMORY AND DATA MEMORY

In a 16-bit RISC processor architecture, the Instruction Memory and Data Memory are two separate memories that serve different purposes.

A. Instruction Memory: The Instruction Memory is a read-only memory that stores the machine code instructions of a program. The functional description of the Instruction Memory can be described as follows:

Initialization: The Instruction Memory is initialized with the program instructions at the start of program execution.

Addressing: The processor uses the Program Counter (PC) to address the next instruction in the Instruction Memory. The contents of this memory location are loaded into the Instruction Register, which holds the current instruction being executed.

Instruction fetch: After the instruction is loaded into the Instruction Register, the PC is incremented to address the next sequential instruction in memory.

B. Data Memory:

The Data Memory is a read/write memory that stores data that the program manipulates during execution. The functional description of the Data Memory can be described as follows:

Initialization: The Data Memory is initialized with the initial values of the program data at the start of program execution.

Addressing: The processor uses memory addresses to access specific data in the Data Memory.

Data read: When data is read from memory, the memory address is used to locate the data, which is then loaded into a register or memory buffer for further processing.

Data write: When data is written to memory, the memory address is used to locate the location in memory where the data is to be stored. The data is then written to that memory location.

3.5 PIPELINING

Pipelining organizes the execution of the multiple instructions simultaneously. Pipelining improves the throughput of the system. In pipelining the instruction is divided into the subtasks. Each subtask performs the dedicated task.

The instruction is divided into 5 subtasks: instruction fetch, instruction decode, operand fetch, instruction execution and operand store. The instruction fetch subtask will only perform the instruction fetching operation, instruction decode subtask will only be decoding the fetched instruction and so on the other subtasks will do. A technique used in advanced microprocessors where the microprocessor begins executing a second instruction before the first has been completed. That is, several instructions are in the *pipeline* simultaneously, each at a different processing stage. The pipeline is divided into segments and each segment can execute its operation concurrently with the other segments. When a segment completes an operation, it passes the result to the next segment in the pipeline and fetches the next operation from the preceding segment. The final results of each instruction emerge at the end of the pipeline in rapid succession. Although formerly a feature only of high-performance and RISC -based microprocessors.

pipelining is now common in microprocessors used in personal computers. Intel's Pentium chip, for example, uses pipelining to execute as many as six instructions simultaneously.

Pipelining is also called pipeline processing. A similar technique used in DRAM, in which the memory loads the requested memory contents into a small cache composed of SRAM and then immediately begins fetching the next memory contents. This creates a two-stage pipeline, where data is read from or written to SRAM in one stage, and data is read from or written to memory in the other stage.

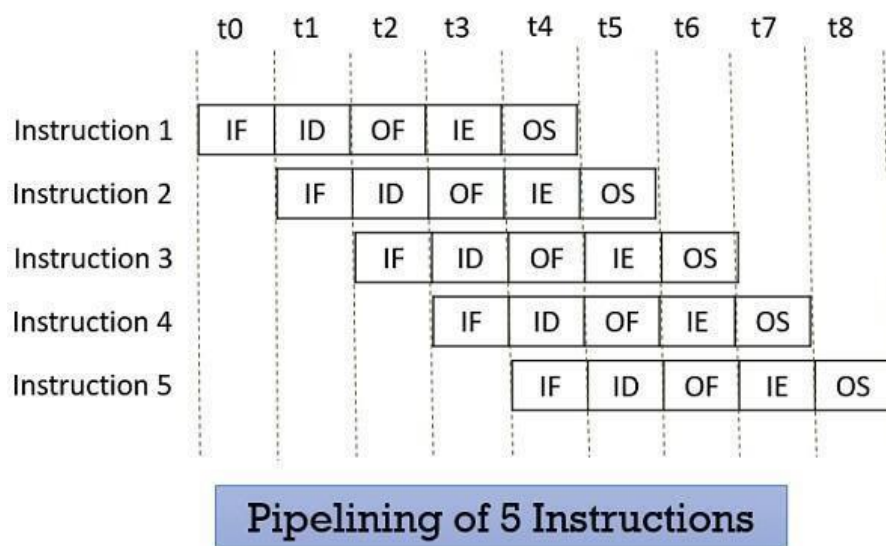


FIG:3.6: Pipelining Of 5 Instructions

CHAPTER- 4

TOOLS USED

2.1 SOFTWARE TOOL: VERILOG

INTRODUCTION TO VERILOG HDL

Verilog HDL (Verilog Hardware Description Language) is commonly used as design-entry language for field programmable gate arrays (FPGA) and application specific integrated circuits (ASIC) in electronic design automation of digital circuits.

Verilog HDL is a fairly general-purpose language, although it requires a simulator on which to run the code. It can read and write files on the host computer, so a Verilog HDL program can be written that generates another Verilog HDL program to be incorporated in the design being developed. Because of this general-purpose nature, it is possible to use Verilog to write a test bench that verifies the functionality of the design using files on the host computer to define stimuli, interact with the user, and compare results with those expected. This is similar to the capabilities of the VHDL. Verilog HDL is a strongly typed language, and as a result is considered by some to be superior to VHDL.

A. FEATURES OF VERILOG

Following are some of the major features and capabilities that the language provides which differentiate it from other descriptive languages.

- The language can be used as an exchange medium between chip vendors and CAD Tool users.
- . The language is publicly available, human readable, machine readable.
- The language supports flexible design methodologies: top-down, bottom down, or mixed. The language supports hierarchy.
- The language is not technology specific, but is capable of supporting the

technology specific features. It can also support various hardware technologies.

- It supports both synchronous and a synchronous timing model. It is an IEEE and ANSI standard; therefore, models described using this language is portable.
- The language supports three basic different description styles: structural, behavioral, and data flow. A design may also be expressed in any combination of these three descriptive styles.
- Test benches can be written using the same language to test other Verilog models. It supports a wide range of abstraction levels ranging from abstract behavioral description to very precise gate level descriptions. It does not, however, support modeling at or below the transistor level.
- Nominal propagation delays, min-max delays, setup and hold timing, timing constraints, and spike detection can all be described very naturally in this language.
- A model can not only described the functionality of a design but can also contain information about the design itself in terms of user defined attributes, such as total area and speed.
- Models written in this language can be verified by simulation since precise simulation semantics are defined for each language construct.
- The capability of defining new data types provides the power to describe and simulate new design techniques at a very high level of abstraction without any concern about the implementation details.

Digital circuits can be designed at any one of several abstraction levels. When designing a circuit at the transistor level, which is the lowest level, one can deal with discrete transistors and connect them together to form the circuit. The next level up in the abstraction is the gate level. At this level, one can work with logic gates to build the circuit. At the gate level, one also can specify the circuit using either a truth table

or a Boolean equation. In using logic gates, a designer usually creates standard combinational and sequential components for building larger circuits.

In this way, a very large circuit, such as a microprocessor, can be built in a hierarchical fashion. Design methodologies have shown that solving a problem hierarchically is always easier than trying to solve the entire problem as a whole from the ground up. These combinational and sequential components are used at the register-transfer level in building the data path and the control unit in the microprocessor. At the register-transfer level, one can concern with how the data is transferred between the various registers and functional units to realize or solve the problem at hand. Finally, at the highest level, which is the behavioral level, one can construct the circuit by describing the behavior or operation of the circuit using a hardware description language. This is very similar to writing a computer program using a programming language.

B. Verilog structural models

A module is declared by writing text describes its functionality, its ports to/from the environment and its timing and other attributes of a design

```
module <module name> ( <list of ports>
); input <width> <input port name>;
output <width> <output port name>;
<variable declarations>
.;; <statements> endmodule
```

A Verilog statement ends with ' ; 'except for the endmodule.

Verilog is a hardware descriptive language that can be used to model the digital system. The digital system can be as simple as a logic gate or as complex as a complete electronic system. A hardware abstraction of this digital system is called an entity in this text. An entity X, when in another entity Y, becomes a component for the entity Y. Therefore, a component is also an entity, depending on the level at which

the model is being tried.

- As a set of interconnected components (to represent structure).
- As a set of concurrent assignment statements (to represent dataflow).
- As a set of sequential assignment statements (to represent behavioral).
- As any combination of the above three.

C.TYPES OF MODELING

1. Structural style of modeling:

When Verilog was first developed (1981) most logic simulators operated on netlists, netlist: list of gates and how they're connected a natural representation of a digital logic circuit not the most convenient way to express test benches. In structural modeling, we first think of our circuit as a box or module which is encapsulated from its outer environment, in such a way that its only communication with the outer environment is through input and output ports. We then set out to describe the structure within the module by explicitly describing its gates and sub modules, and how they connect with one another as well as to the module ports. In other words, structural modeling is akin to drawing a schematic diagram for the circuit.

2. Dataflow type of modeling (Concurrent type):

At this level, the module is designed by specifying the data flow. The designer is aware of how data flows between hardware registers and how the data is processed in the design. Describe the continuous assignment (assign) statement, restrictions on the assign statement, and the implicit continuous assignment statement. Explain assignment delay, implicit assignment delay, and net declaration delay for continuous assignment statements. Define expressions, operators, and operands. List operator types for all possible operations? Arithmetic, logical, relational, equality, bitwise, reduction, shift, concatenation, and conditional. Use dataflow constructs to model

practical digital circuits in Verilog.

3. Behavioral style of modeling:

A circuit is defined in terms of its behavior: e.g. in showing the relationships between the outputs and inputs. This may be done using truth-tables, Boolean equations, or arithmetic equations. A much easier way to write test-benches is also good for more abstract models of circuits easier to write, simulates faster, more flexible provides sequencing verilog succeeded in part because it allowed both the model and the testbench to be described together. Two main components of Verilog, Concurrent, event-triggered processes (behavioral). It has initial and always blocks, imperative code that can perform standard data manipulation tasks (assignment, if-then, case) Processes run until they delay for a period of time or wait for a triggering event; the basic essence of a behavioral model is the process. The process can be thought of as an independent thread of control. The process can be implemented as a sequential state machine, as a microcoded controller, as an asynchronous clearing of a register, or as a combinational logic circuit. The key idea is that we conceive the behavior of digital systems as a set of these independent, but communicating processes. The actual implementation is left to the context of the description (what level of abstraction we are dealing with) and the time and area constraints of the implementation.

```
module
behavioralNand (out, in1, in2, in3);
output out;
input in1,in2,in3;
reg out;
parameter delay =5;
always@(in1 or in2 or in3) #delay out= ~(in1 & in2 & in3);
endmodule
```

Mixed style of modeling: Verilog allows the designer to mix and match all four levels of abstractions in a design. In the digital design community, the term register transfer level (RTL) is frequently used for a Verilog description that uses a combination of behavioral and dataflow constructs and is acceptable to logic synthesis tools.

D.SIMULATION

Test bench generates stimulus and checks response coupled to model of the system pair is run simultaneously.

E.ADVANTAGES OF VERILOG

- VERILOG when used for systems design allows the behavior of required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).
- VERILOG allows the description of a concurrent system (many parts, each with its own sub-behavior, working together at the same time). This is unlike many of the other computing languages such as BASIC, Pascal, C, or low-level assembly language which runs at machine code level, which all run sequentially, one instruction at a time on von Neumann architectures.

VERILOG model is translated into the gates and wires" that are mapped onto a programmable logic device such as a CPLD or FPGA, and then it is the actual hardware being configured, rather than the VERILOG code being "executed" as if on some form of a processor chip.

2.2 XILINX ISE

A. INTRODUCTION TO XILINX ISE SOFTWARE

Xilinx ISE is a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize ("compile") their designs.

They perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

After file synthesis with the Xilinx XST tool and the simulation analysis, some numerical results were obtained. These results reflect the hardware resources consumption and timing performance of the processor. The Virtex series is the high performance member of the Xilinx family. The Virtex series has in addition to the normal FPGA logic fabric embedded fixed function hardware for commonly used functions such as multipliers, memories, serial transceivers, and microprocessor cores. The number of multipliers range from a few tens to several hundreds, depending on device. Likewise, the amount of block rams ranges from a few hundred kilobits to ten megabits. While all models include some block rain and multipliers, embedded PowerPC cores and a few simple hardware are only available on select models.

2.3 XILINX VIVADO:

Xilinx Vivado is a software suite used for designing and programming Field Programmable Gate Arrays (FPGAs) and other digital circuits. It offers a complete development environment with an integrated design flow that allows designers to create, simulate, and synthesize their designs. Vivado supports a range of FPGA families, including Xilinx's newest UltraScale and UltraScale+ devices. One of the key features of Vivado is its High-Level Synthesis (HLS) capability, which allows designers to write their designs in C, C++, or SystemC, and then automatically generate RTL code for implementation on an FPGA. Additionally, Vivado offers advanced debugging and verification tools, including a simulator that allows designers to test their designs. Another advantage of Vivado is its compatibility with third-party tools and languages. Vivado supports a range of industry-standard design languages such as Verilog, VHDL, and SystemVerilog, and can interface with other EDA tools such as MATLAB, Simulink, and ModelSim. Overall, Vivado is a powerful tool for FPGA design, offering a comprehensive suite of features that make it a popular choice among FPGA designers.

CHAPTER- 5

DESIGN OF 16-BIT PROCESSOR

5.1 DESIGN METHODOLOGY

Usually, all classical design methods involve one or more PCBs (printed circuit board) that contain many chips together with other components. After that the required integrated circuits chips are selected followed by the PCBs that house and connect the chips together are designed. Since the complexity of circuits implemented on individual chips and on the circuit boards is usually very high, it is very much essential to make use of Xilinx software. The design method of the proposed 16 bit processor is shown in Fig below FPGA devices are software configured and field programmable. Hence, there is a significant cost saving in design and productions. Design entry of the processor is carried out by using VHDL code. Initial synthesis generates an initial circuit, based on data entered during the design entry stage. Physical design determines how to implement the optimized circuit in a FPGA chip. Timing simulation determines the propagation delays that are expected in the implemented circuit.

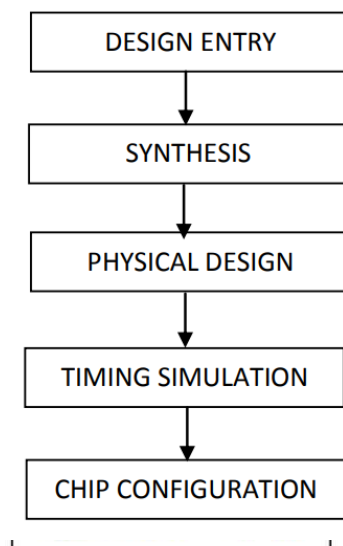


FIG:5.1: Design Methodology Flow Diagram

5.2 ARCHITECTURE OF DESIGN

The architecture of the proposed 16-bit Processor is shown in Fig.2 which consists of a processor block and memory block communicating through a data bus, an address bus and a few control lines [8-10]. The architecture of the 16-bit processor is designed based on three 16-bit instruction formats R-format, I-format and J-format. The design of this processor consists of 16-bit instructions and 16-bit data path. The implementation of the processor performs fetch, decode, and execute operations. The fetch stage obtains the requested instruction from memory. The operation of the fetch stage starts when the program counter (PC) a 16-bit register is sent out to fetch the instruction from memory into the instruction register (IR) and the PC is incremented to address the next sequential instruction. The IR is used to hold the instruction needed on subsequent clock cycles.

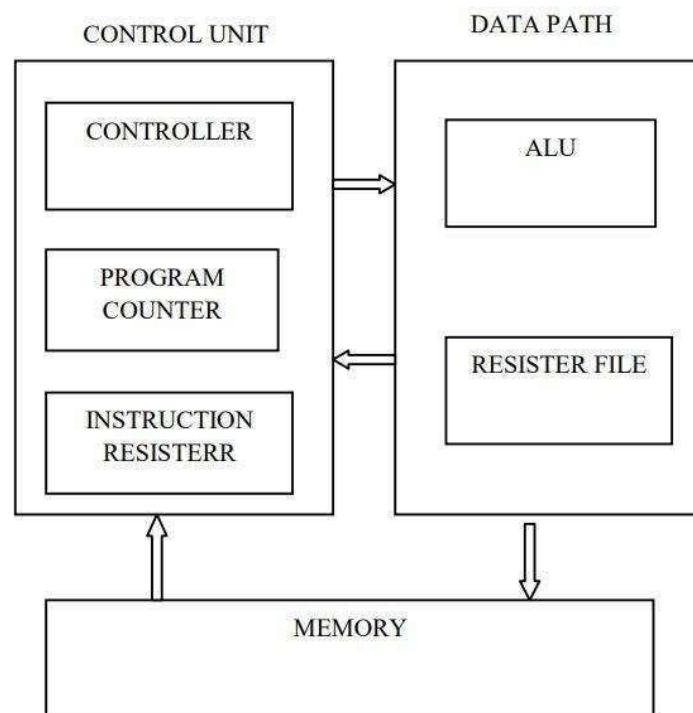


FIG:5.2: Proposed 16-bit Processor Architecture

The processor's 16-bit registers are stored in a 'register file' that contains the register state of the machine. For R-format instructions, there are three register operands. Two data words are read from the register file and one data word is written into the register file for each instruction. The register number inputs are 4 bits wide to specify one of the 16 registers, whereas the data input and two data output buses are each 16 bits wide. When the instruction is fetched from the fetch stage, the instruction's operation code is sent to the control unit. The instruction's register address fields are used to address the two port register files. The two-port register performs two independent reads and one write in one clock cycle. In the decode stage, the instructions are decoded and the register file is accessed to read the registers. The outputs of the general purpose registers are read into two registers, register 1 and register 2. After the instructions decode the stage, the execution stage performs calculations.

5.3 DATA PATH

There are 16 numbers of 16 bit registers in the register file. So the Register file of the data path consists of a 4-bit address bus and 16-bit data bus. This is a two port register file which can perform two simultaneous read and one write operations. It contains sixteen 16-bit general purpose registers. The registers are named R0 through R15. When the Register Write signal is high, a write operation is performed to the register indicated by the write address, otherwise it outputs the value contained in the register indicated by the read address. The ALU is responsible for all arithmetic and logic operations that take place within the processor. These operations can have one operand or two, with these values coming from either the register file or from the immediate value from the instruction directly. The output of the ALU goes either to the memory (in the case where the output is an address) or through a multiplexer

back to the register file. The ALU has two 16-bit data inputs for two operands and 16-bit output for result. It can perform Booth multiplication, restoring division, addition and subtraction functions. The small multiplexer of the data path has three 16-bit inputs and one 16-bit output.

It selects one of the 16-bit inputs and sends to the output depending on the condition of the two-bit multiplexer select signal. The multiplier unit has two inputs and one output. The internal structure consists of four units namely Booth Encoder, Partial Product generator, Carry Save Adder, and Carry Look Ahead Adder. The top design of Booth Encoder has one input and three outputs. The Booth Encoder generates the booth codes to encode the multiplicand into the partial products. The outputs of the Booth Encoder are fed to the inputs of the partial product generator (PPG) module. The PPG module reads the booth code signals generated from the Booth Encoder to encode the multiplicand into the partial products. The outputs of the partial product generators are taken as the inputs of the Wallace tree. The Wallace tree uses the 3 to 2 carry save adder to implement the Wallace tree. The outputs of the Wallace tree are fed to the inputs of the Carry Look Ahead (CLA) adder. The CLA is used to perform the addition of the final sum and carry vector. After implementing all the four components of the fast multiplier unit, the desired multiplication result is obtained. In restoring division, the divisor is positioned appropriately with respect to the dividend. Then the divisor is subtracted from the dividend. If the remainder is zero or positive, a quotient bit of '1' is determined. The remainder is extended by another bit of the dividend and the divisor is repositioned and another subtraction is performed. If the remainder is negative, a quotient bit of '0' is determined and the dividend is restored by adding back the divisor. Then the divisor is repositioned for another subtraction.

5.4 CONTROL UNIT

Control unit composed of controller, program counter, instruction register, and

multiplexer. The controller provides the necessary signal interaction to make the data flow through the processor and perform expected function. The 16-bit program counter indicates the address of the next location where the next instruction is to be fetched. It has 16-bit program counter input, 16-bit program counter output, and some control signals like read, write, and clear. The Program Counter (PC) contains the address of the instruction that will be fetched from the Instruction Memory during the next clock cycle.

Normally the PC is incremented by one during each clock cycle unless a branch instruction is executed. When a branch instruction is encountered, the PC is incremented or decremented by the amount indicated by the branch offset. The instruction register (IR) has one 16-bit input and two 16-bit outputs. The instruction set describes the bit-configurations allowed in the IR. The instruction consists of operation codes and operands. The processor supports direct, register, registers indirect and immediate addressing modes. In immediate addressing; the operand field contains the data itself. In register addressing, the operand field contains the address of a data path register in which the data resides. In registers indirect addressing, the operand field contains the address of a register, which in turn contains the address of a memory location in which the data resides. In direct addressing, the operand field contains the address of a memory location in which the data resides. In indirect addressing, the operand field contains the address of a memory location, which in turn contains the address of a memory location in which the data resides.

Memories are used for storage of both instructions and data. The process of storing data into memory is called writing and retrieving data or operation codes from the memory is called reading. For reading and writing data, the particular memory location is identified and then reading or writing is done. There are 256 numbers of 16-bit memory locations. It has an 8-bit address and 16-bit word.

CHAPTER- 6

SIMULATION RESULTS

We have synthesized and simulated the VHDL code of the proposed processor using Xilinx Integrated Software Environment tool (Version 10.1). The synthesis results and simulation results of the processor are presented for justification. The proposed 16 bit RISC processor is coded with VHDL (very high speed integrated circuit hardware description language). Using Xilinx ISE 10.1 software the code is tested and checked for errors. When there is no error, the code is synthesized and simulated using Xilinx ISE 10.1 software. The synthesis and simulation results are compared with the theoretical results. Before the start of simulation, the memory is loaded by writing the instructions and data into the memory. The completed processor with memory is tested for addition, subtraction, multiplication, and division. When the VHDL code is 100% synthesized, and then the code is downloaded to the Spartan FPGA device. After downloading the code, the functionality of the 16 bit RISC processor is compared with the theoretical result.

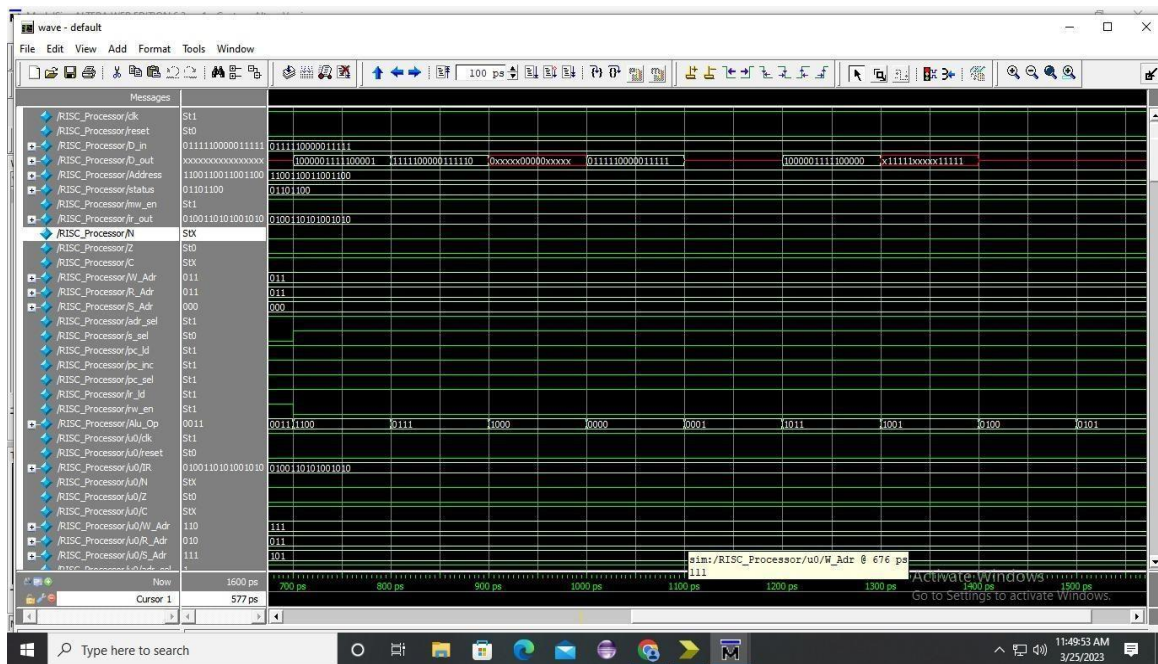


FIG:6.1: Simulation Output Screenshot Of RISC Processor

Simulation analysis of ALU and Timing Diagram of General purpose processor. The simulation result of the proposed 16 bit RISC processor is shown in Fig.4. The two input 16 bit operands to the processor are 7 and 5. They can be either in register location, memory location, or given as immediate values. The two 16 bit outputs from the processor are `cpu_out(15:0)` and `memory_add (15:0)`. The `cpu_out(15:0)` and `memory_add (15:0)` represent arithmetic logic unit result and memory address respectively. Initially '7' decimal equivalent of 16 bit operand is copied into register address R1. Similarly '5' is copied into register address R2. The contents of R1 that is 7 is copied in memory address '19'. Similarly contents of R2 is stored in memory address '20'. Now '7' contents of R1 is added with '5' contents of R2 to produce result '12' and is stored in R1. The additional result in R1 is copied in memory address '21'. Similarly '5' contents of R2 is subtracted from '12' modified contents of R1 to produce result '7' and is stored in R1. The subtraction result in R1 is copied in memory address '22'. Likewise '7' modified contents of R1 is multiplied with '5' contents of R2 to produce result '35' and is stored in R1. The multiplication result in R1 is copied in memory address '23'. Then '35' modified contents of R1 is divided with '5' contents of R2 to produce result '12' and is stored in R1. The division result in R1 is copied in memory address '24'.

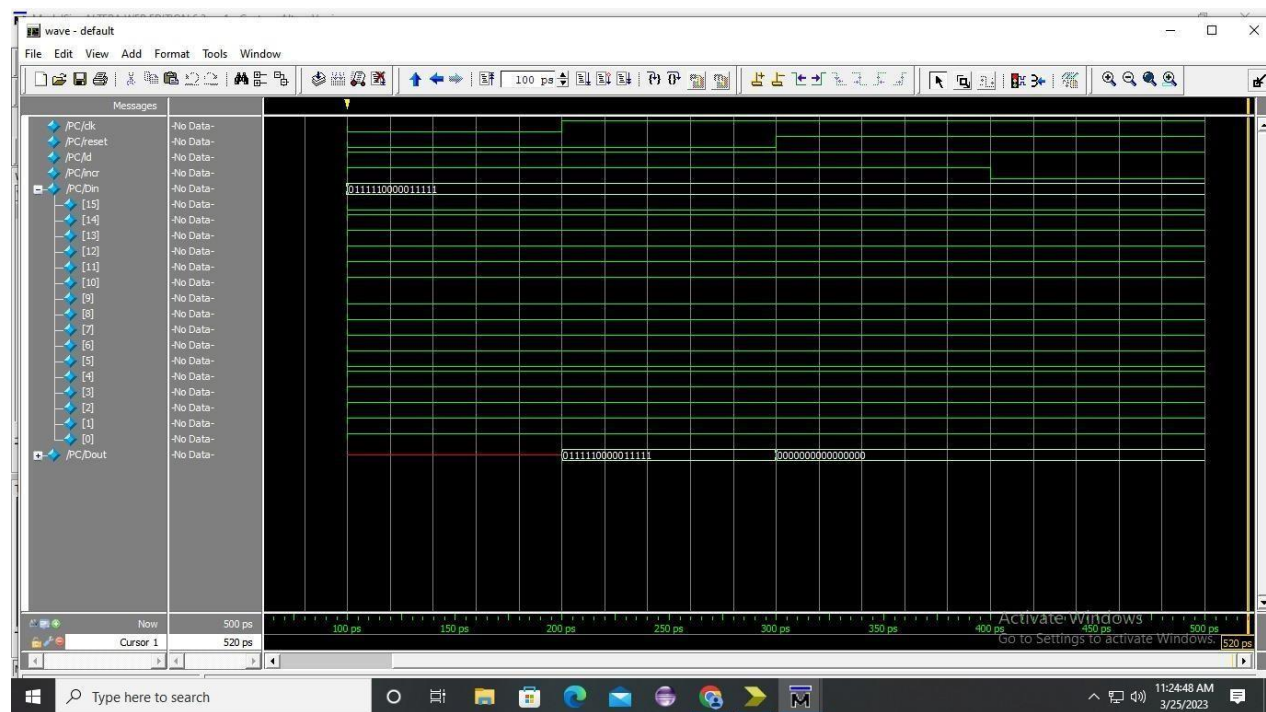
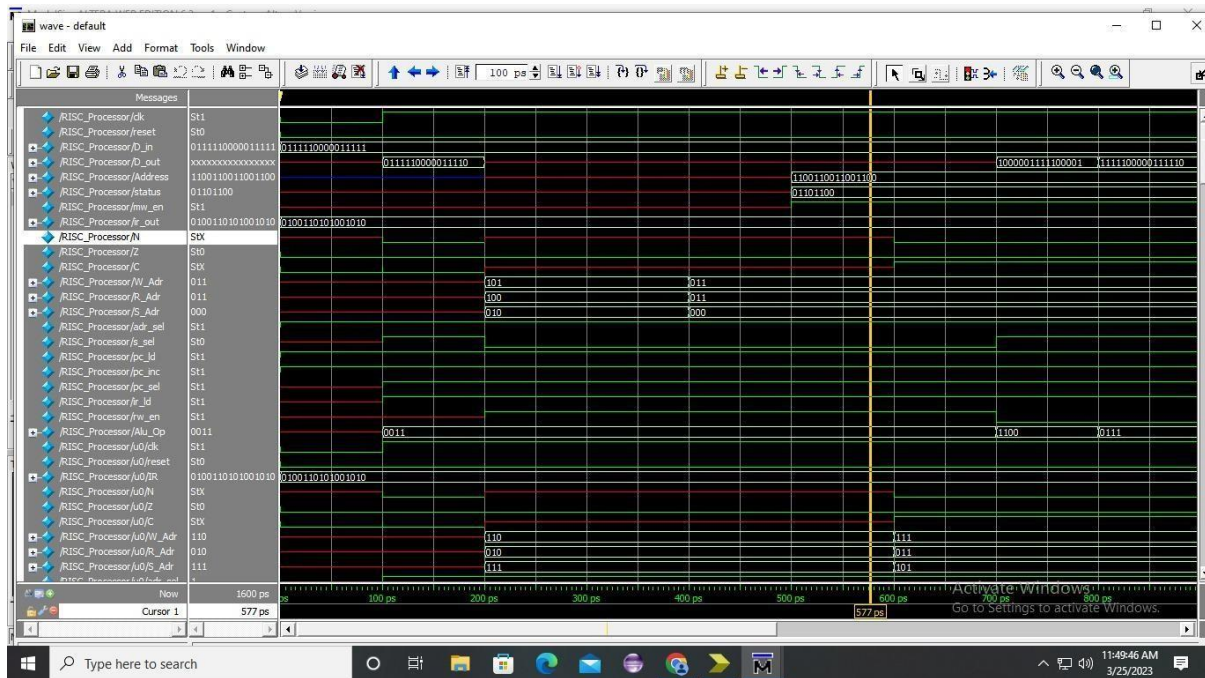


FIG:6.2: 16-Bit Registers Timing Diagram

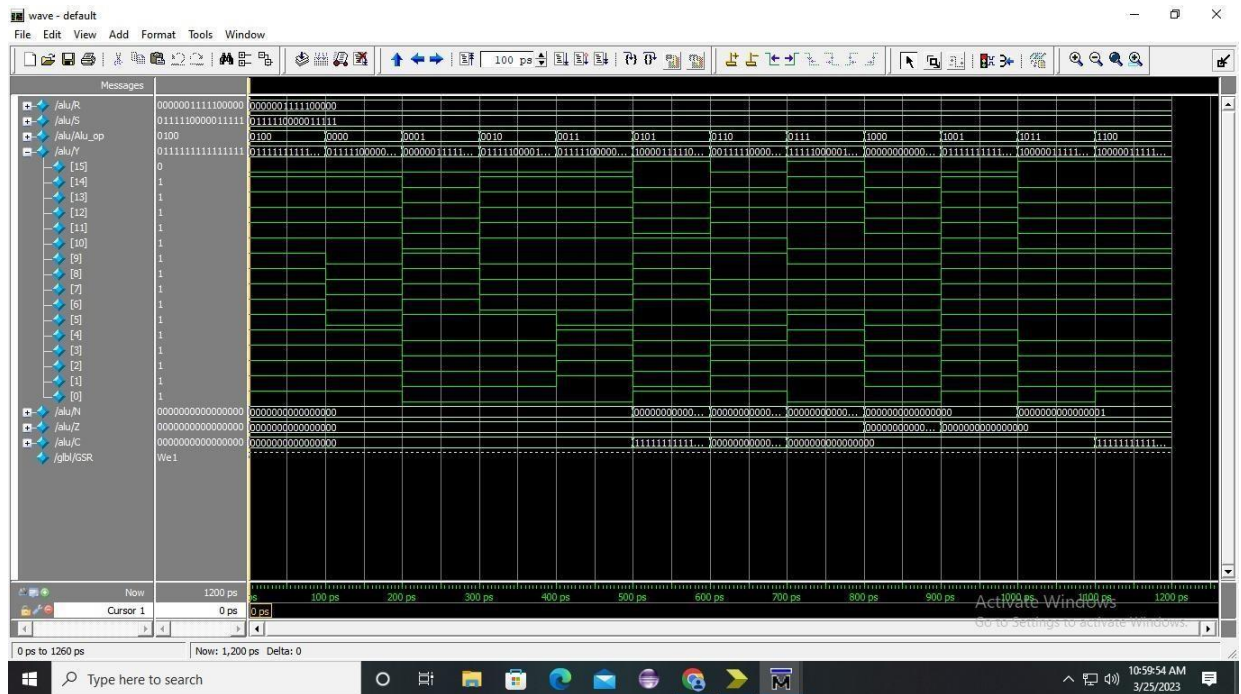


FIG:6.3: Total ALU Operations Results Screenshot

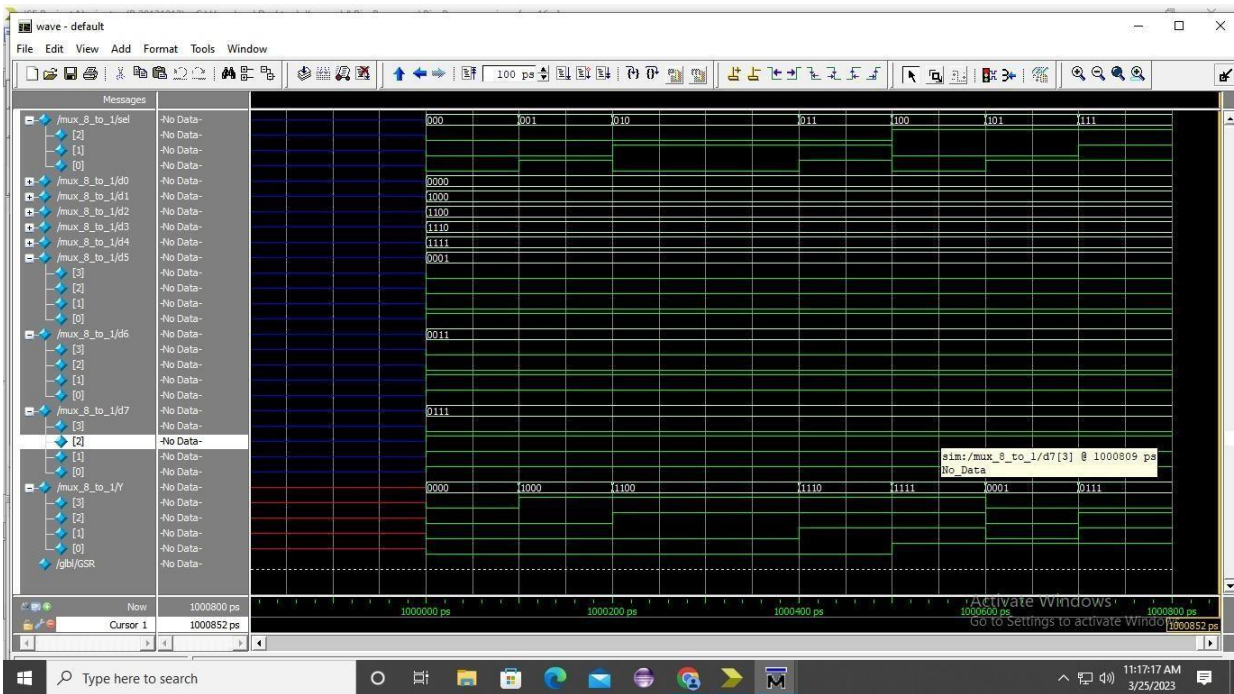


FIG:6.4: Total ALU Operation Result Screenshot

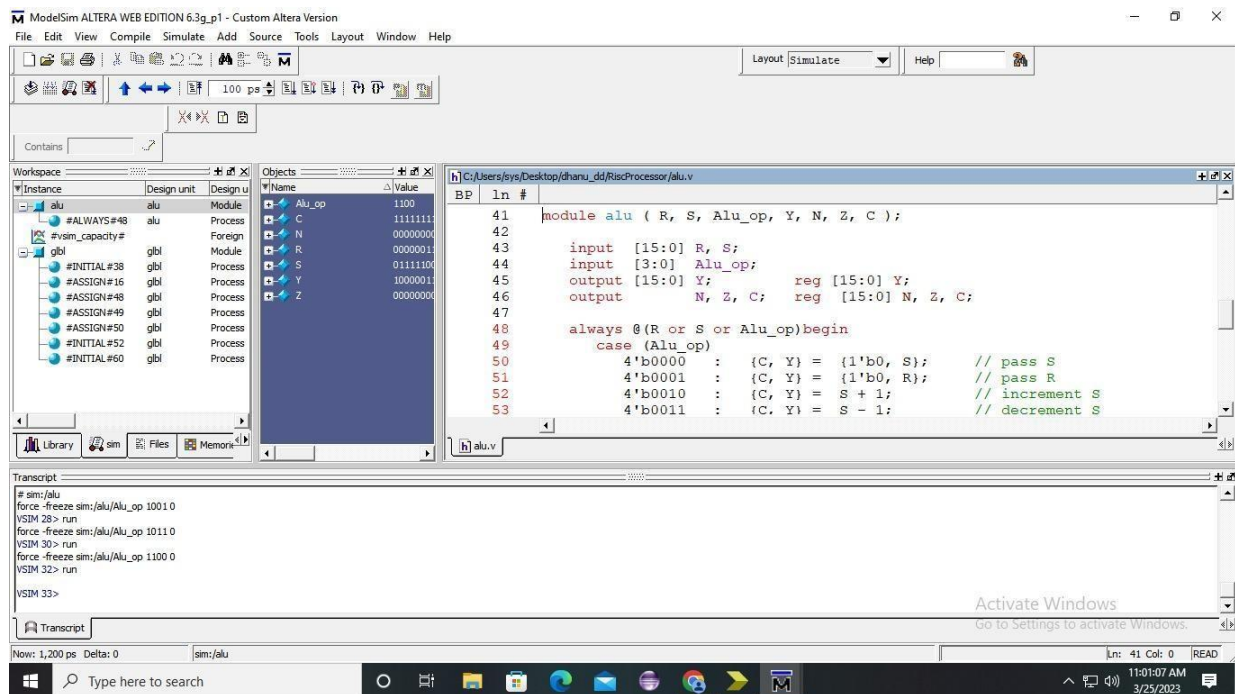


FIG:6.5: Simulation Output Of Multiplexers

CHAPTER- 7

SUMMARY AND CONCLUSION

7.1 CONCLUSION

We tested our processor architecture by running addition, subtraction, booth multiplication, and restoring division programs created using Xilinx Software. We have compared the simulated output results with the expected results. From the synthesis report, the minimum clock period that can be achieved in this proposed architecture is 31.641 ns (31.605 MHz). The booth multiplier unit is tested and the functionality is found correct. Similarly, the functionality of restoring the division circuit is tested and found correct. Finally, the performance of the processor is tested for addition, subtraction, booth multiplication, and restoring division programs and the functionality is found correct.

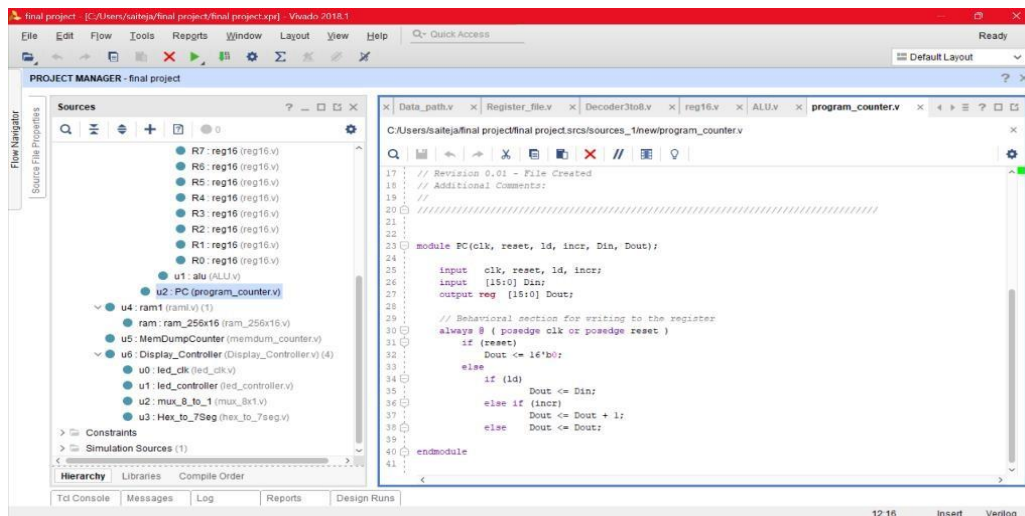
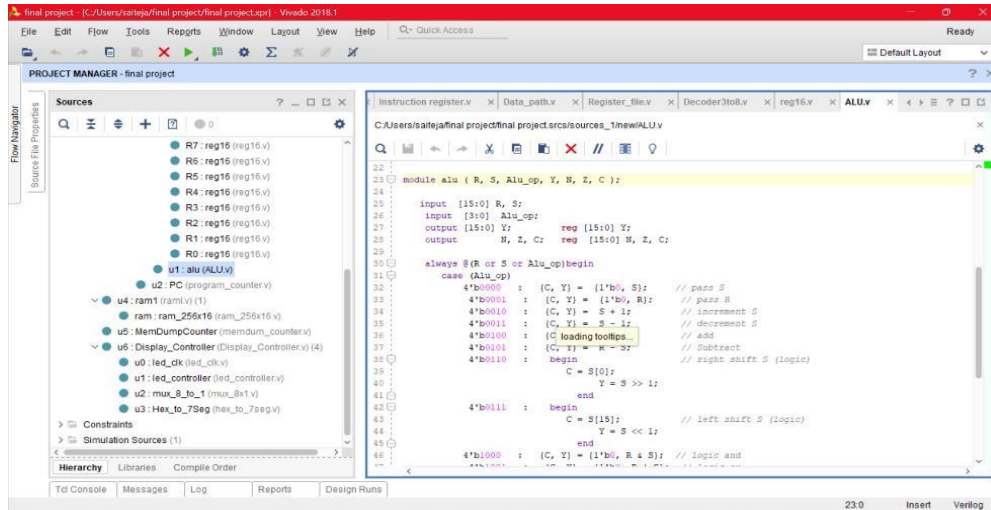
7.2 SUMMARY

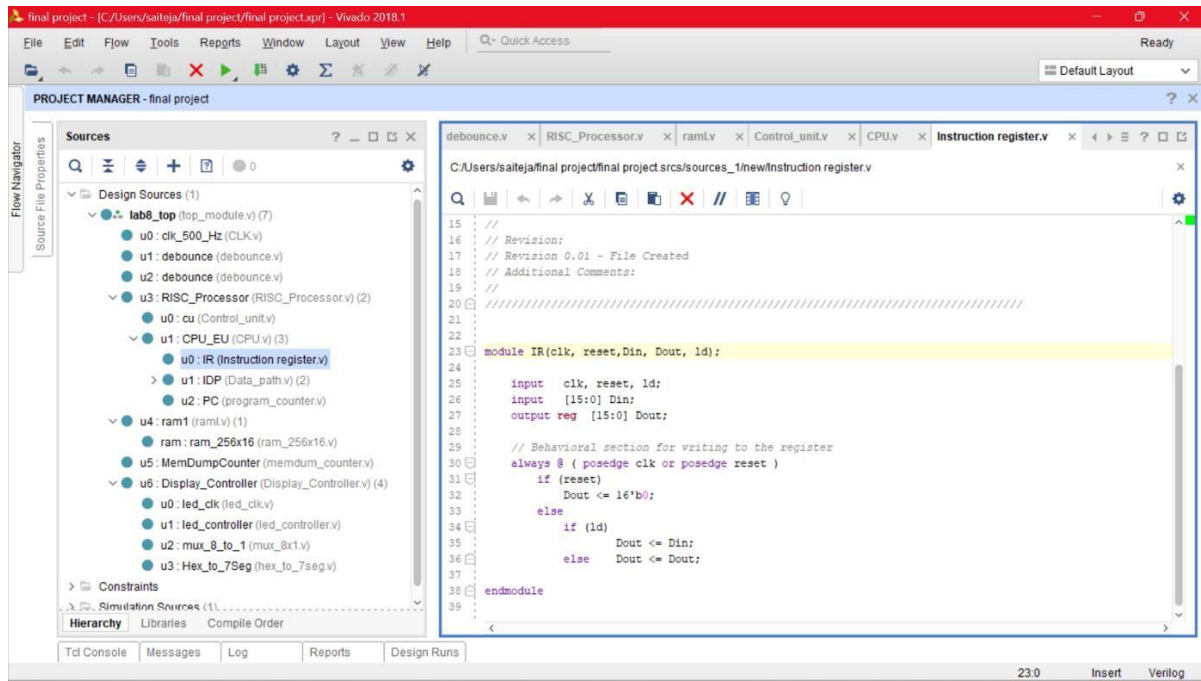
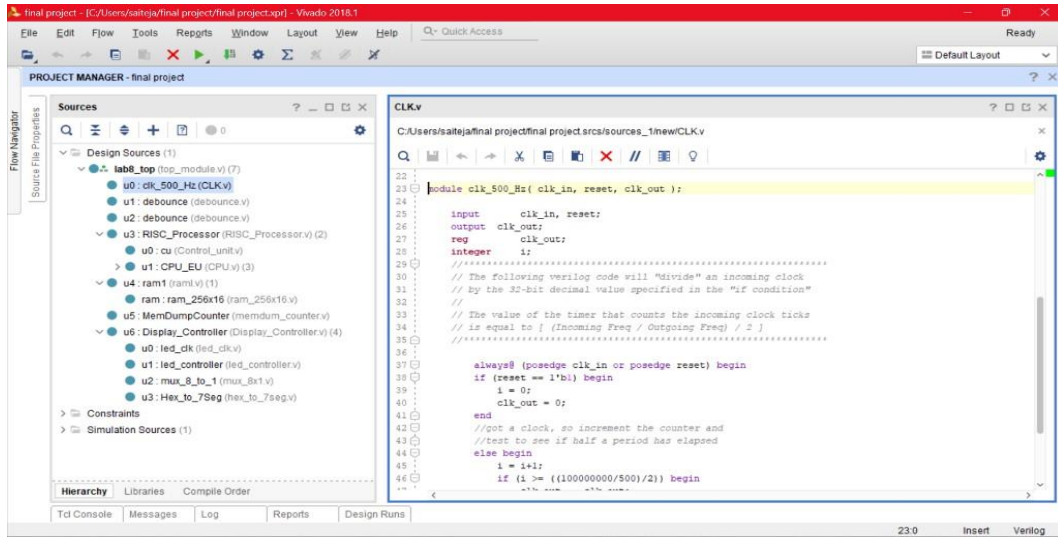
The overall design of the 16-bit processor is simulated and is shown . The synchronization of various operations is done using signal.CLK The processor is designed with a control signal RST. If RSTis high, then the processor will not perform any operation and it will remain in idle state. If the RST is low and CLK is high, then the data is transferred to the data bus and the corresponding values are loaded into the general purpose registers. The signal write loads the data of bus1 to memory location indicated by Address register. Thus the processor interacts with memory. It operates at a max. frequency range of 166.522 Mhz. Total power consumption is 0.356w which is very less. The resource utilization is shown in table 1. The computational instruction takes four cycles to execute.

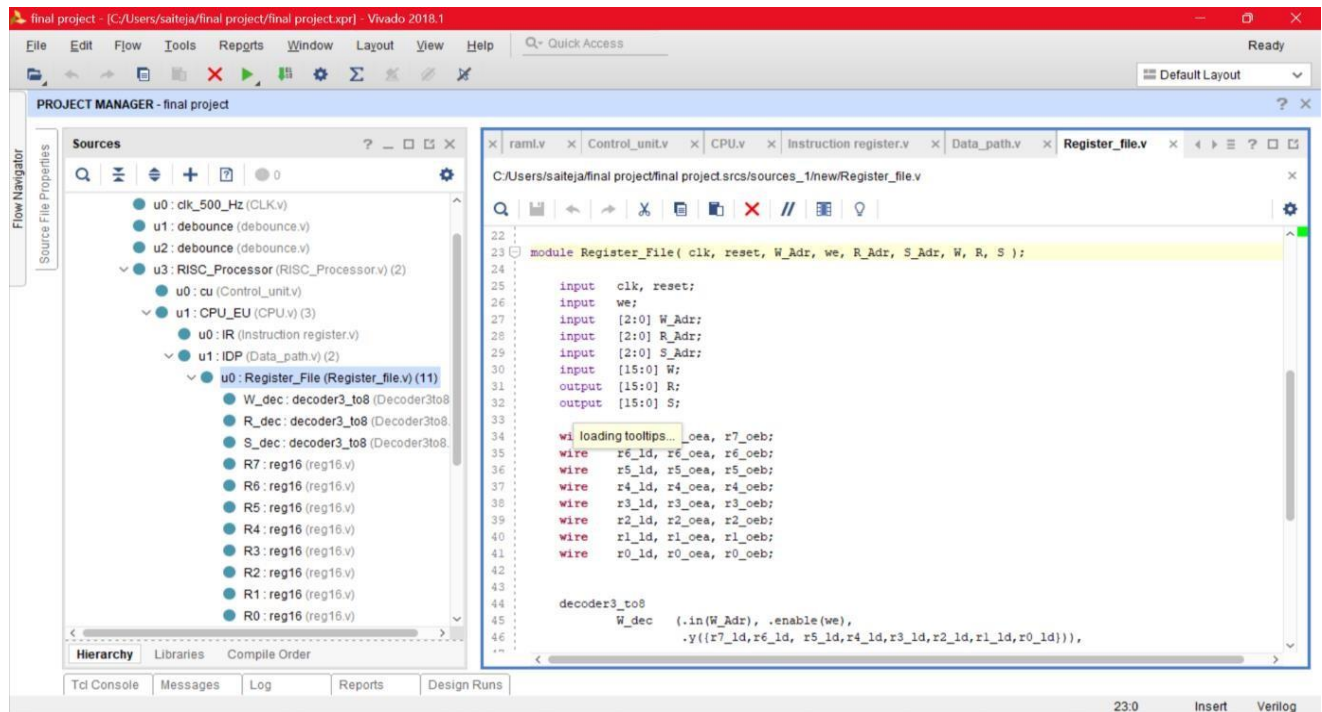
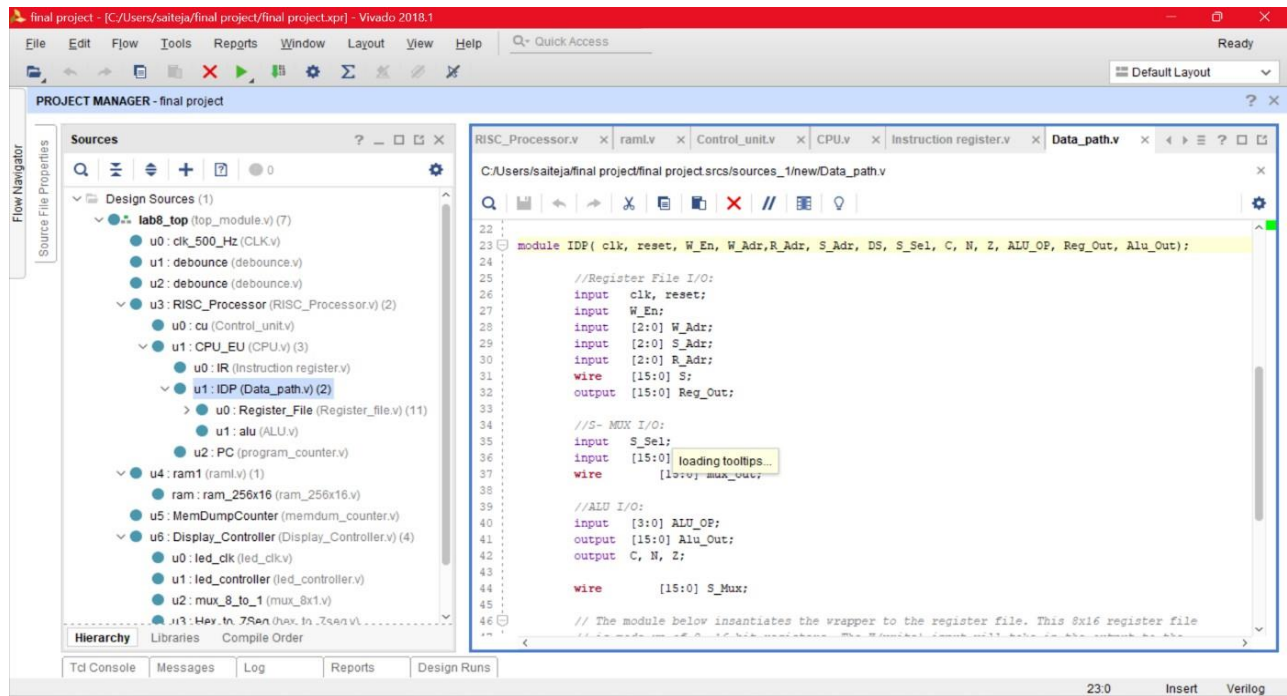
REFERENCES

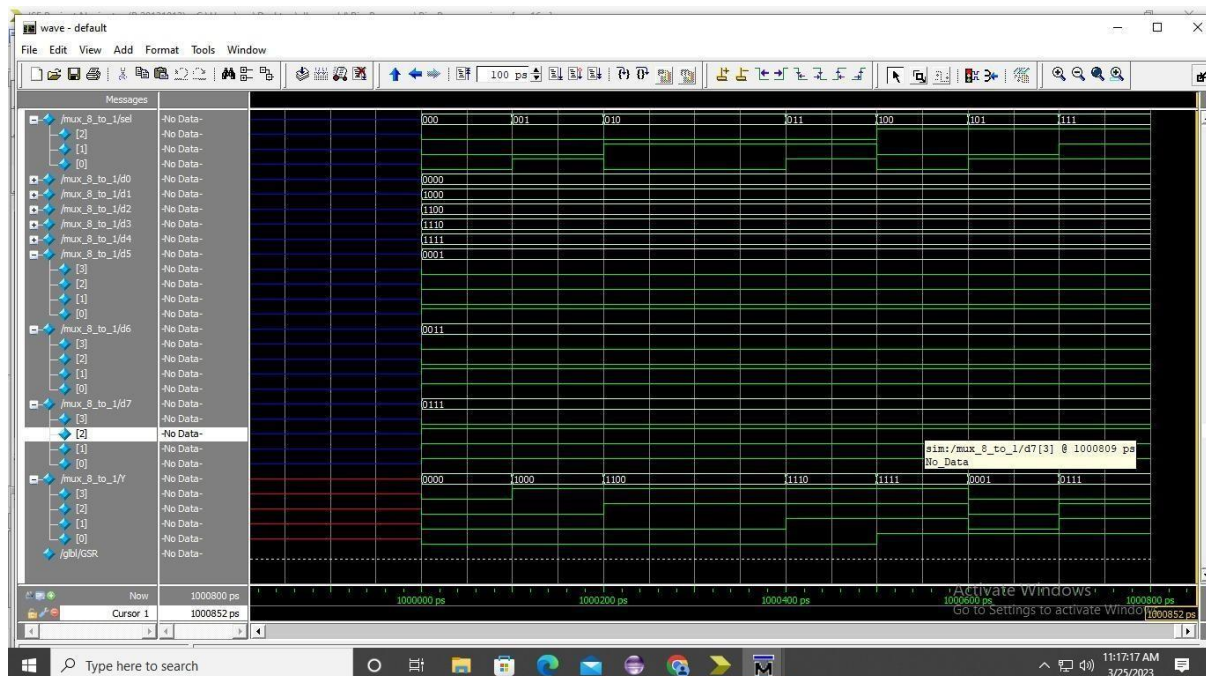
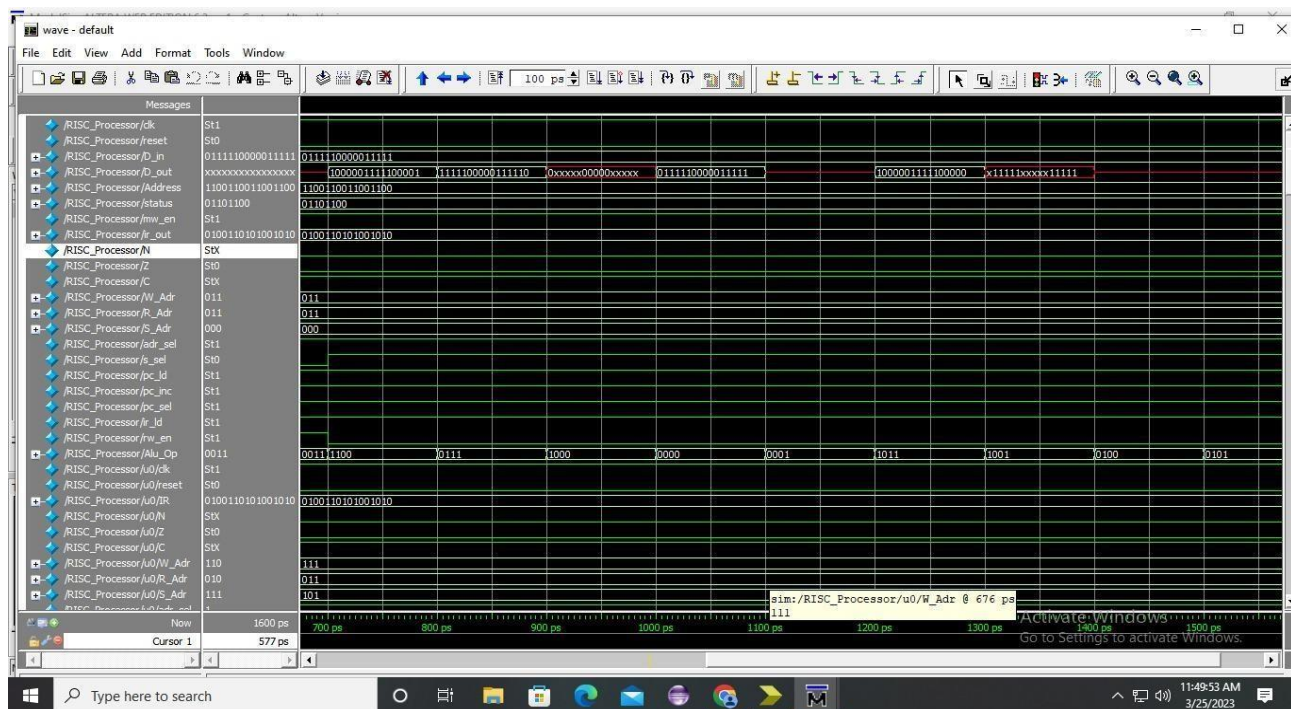
- [1] Mamun B., Shabiul I. and Sulaiman S. 2002. A Single Clock Cycle MIPS RISC Processor Design using VHDL. Penang, Malaysia, pp.199- 203, 2002.
- [2] Hamblen J. Using Synthesis, Simulation, and Hardware Emulation to Prototype a Pipelined RISC Computer System. Atlanta, Georgia.
- [3] Zainalabedin N. Using VHDL for Modeling and Design of Processing Units. Pp.315- 326, Boston, Massachusetts.
- [4] Outline of OROCHI. 2007. A Multiple Instruction Set Executable SMT Processor., International Workshop on Innovative Architecture for Future generation Processors and Systems.
- [5] Takanori M., Satoshi A., and Masaaki I. 2005. A Multithreaded Processor Architecture Based on the Continuation Model. Proceedings of the Innovative Architecture for Future Generation High-Performance Processors and Systems. Kasuga-Koen, Kasuga, Fukuoka, Japan.
- [6] Virendra S. and Michiko I. 2006. Instruction-Based SelfTesting of Delay Faults in Pipelined Processors., IEEE Transaction on VLSI systems, vol. 14, no.11, pp.1203-1215.
- [7] Hamacher V. and Zaky. 2002. Computer Organization. McGraw-Hill Companies, New York, 5th edition.
- [8] Frank V. and Tony G. 1999. Embedded System Design, A Unified Hardware/Software Approach., Department of Computer Science and Engineering University of California, Draft version.
- [9] Patterson A. and Hennessy J. 1999. Computer Organization & Design., Morgan Kaufmann Publishers.
- [10] Weijun Z. 2001. VHDL Tutorial, Learn by Example.
- [11] Data sheet of Zunc FPGA Family. 2003. XILINX, DS001-2 (V2.2)

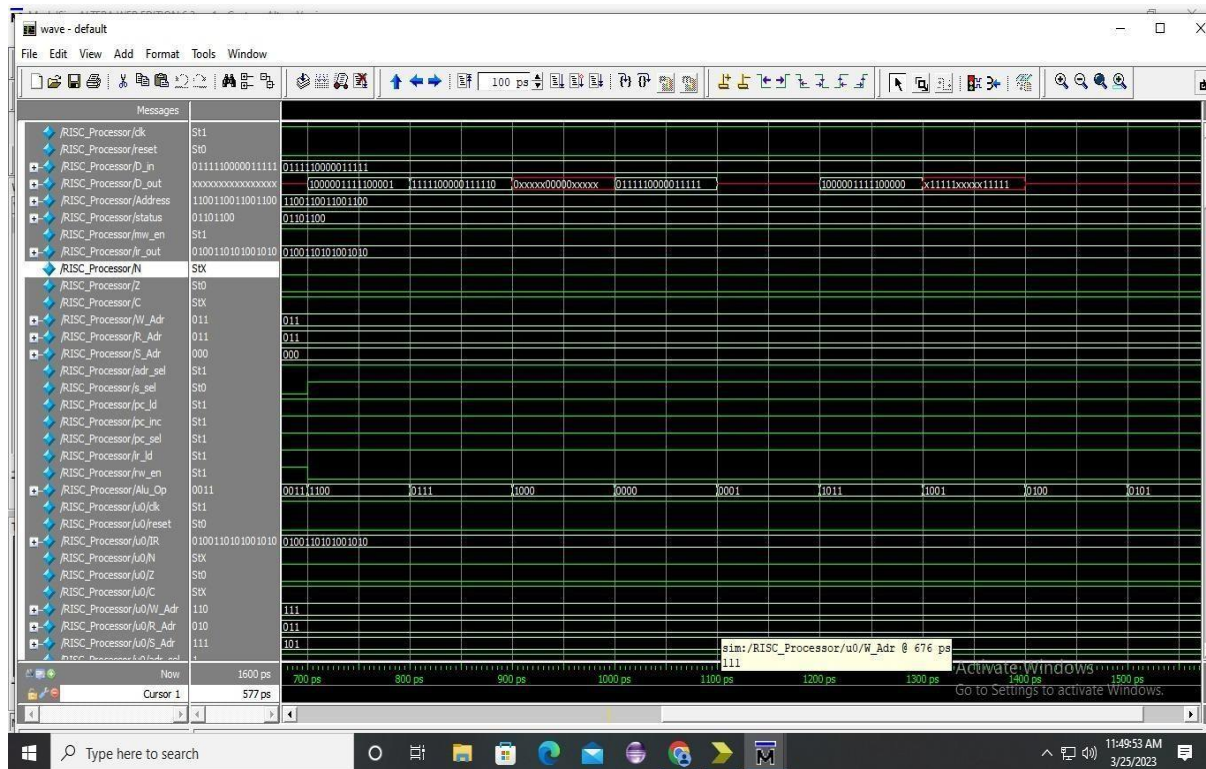
A.SCREENSHOTS











Performance Metrics of 16-Bit VLSI RISC Architecture

Jarnpula Saiteja¹
Student

Department of Electronics and
Communication Engineering,
Sathyabarna Institute of Science and
Technology,
Chennai,India.
Saitejjarnpula143@gmail.com

Gundlapudi Surya Teja²
Student

Department of Electronics and
Communication Engineering,
Sathyabarna Institute of Science and
Technology,
Chennai,India.
Suryatej0476@gmail.com

M Subrarnaniarn³
Associate Professor

Department of Electronics and
Communication Engineering,
Sathyabarna Institute of Science and
Technology,
Chennai,India.
Subrarnoniam.viru@gmail.com

Abstract-

The Reduced Instruction Set Computer is what it's called. The Reduced Instruction Set Computer (RISC) architecture is used for fast execution

.RISC instruction uses register-to-register operations. The results are stored in registers for different operations and stores. Instructions are used to move the results to memory. Each and every step is done inside the registers. All RISC instructions have their clear register addressing, reducing the number of addressing modes that are needed. RISC effortlessly decodes and uses easy and flexible syntax. Word boundaries can be utilized to align the instruction and to set the instruction's length in this case. To accomplish this, using overlying the fetch, decode, and execute stages of one or more instructions is known as pipelining. RISC runs programs faster than CISC because it uses a large percentage of processor unit's registers. This project involves constructing a 16-bit RISC processor and utilizing Verilog HDL to model its component parts. Based on Harvard architecture, the processor. The incredibly basic instruction set used in this case provides understanding into the type of hardware that should be able to correctly execute the instructions. More sophisticated building components, like ALU and memory, have been built and simulated in addition to the sequential and combinational building blocks of a processor, including adders and registers.

KEYWORDS - RISC Processor,CISC Processor, CPU, Fetch, Decode, Execute, ALU(Arithmetic logical unit),Harvard Architecture, Von Neumann Architecture, Operations, Pipelining.

I. INTRODUCTION

People started looking at other options when the controller architecture in CISC became more difficult and the performance was also below expectations. It has been discovered that speed is eliminated when a CPU communicates with memory. The only way to improve CPI was to keep the instruction set as basic as possible. Simple in terms of appearance rather than functionality. Because of this, very few instructions-probably just load and store-in a typical RISC architecture require the CPU to access data from memory. In the end, pipelining increases speed by a new dimension just by adding a few extra registers, which lowers CPI and raises throughput. As a result, the instruction can be efficiently operated out in a single clock cycle.

The misconception that instructions are simply removed to produce a reduced set of instructions is a frequent misinterpretation of the phrase "Reduced Instruction Set Computer." In fact, RISC instruction sets have expanded over time, and many of them now have more instructions than many CISC CPUs. The word "Reduced" in that phrase refers to the fact that, in contrast to the "complex instructions" of CISC CPUs, which may require multiple data memory cycles to execute a single instruction [2], the amount of work that any single instruction accomplishes is reduced to at most one data memory cycle. The majority of microprocessors on the market today are built using the RISC or CISC designs. According to research, RISC architecture significantly increases computer speed by implementing streamlined machine instructions for often used functions. The following characteristics are frequently present in RISC-based systems.

1) Pre-fetching: Pre-fetching is fetching the next instruction to be executed and keeping it in a queue in advance before the current instruction is finished.

2) Pipelining: Pipelining enables issuing an instruction before the one that is presently being executed has finished.

3) Superscalar operation: A processor that can send out several instructions at once is said to be using superscalar operation.

II. LITERATURE SURVEY

Presented in the 2015 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT) by Jikku Jeemon. The paper available on IEEE Xplore, presents a low power and simple implementation of an 8-bit RISC processor design on a Spartan-6 SP605 Evaluation Platform FPGA using Verilog HDL. The processor is designed using Harvard architecture, having separate instruction and data memory. It has an instruction set of only 29 instructions. The proposed processor is physically verified on Xilinx Spartan-6SP605 Evaluation Platform.

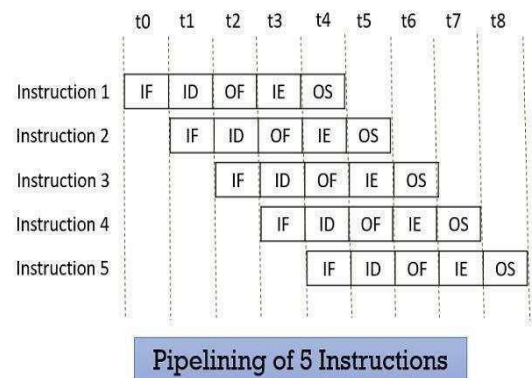
Published in IJERT Vol. 2 Issue 7, July 2013 by Supraj Gaonkar and Anitha M. The behavioral design and characteristics of a 16-bit RISC processor is designed by using minimum units without minimizing the performance. It is Harvard based architecture. All the modules are designed using Verilog. The separate modules are designed and tested at every level of execution and integrated in a top-level module. Xilinx ISE 10.1 and ModelSim 10.2

Presented in the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA) by Shraddha M. Bhagat and Sheetal U. Bhandari This paper is available on IEEE Xplore and the design of a 16-bit RISC processor is explained using Verilog HDL. It comprises various blocks such as ALU, Controller, register files and data memory unit. It is based on the simple Von Neumann architecture which has a single shared memory for instructions and data. It consists of 15 instructions. The processor is analyzed using the Xilinx ISE tool.

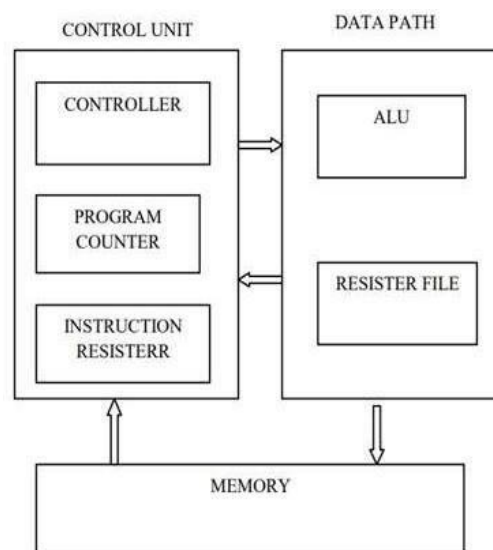
III. ARCHITECTURE

The project's main aim is to design a 16-bit RISC processor based on Harvard architecture that makes use of the fewest possible functional components. The structure of the suggested In Fig. 1, a 16-bit

processor is depicted. The processor has a 16-bit ALU with 11 arithmetic and logical operations, a 3-bit flag register for carry, zero, and parity, a 24-bit instruction register, a 16-bit program counter, and sixteen 16-bit general purpose registers. The four states of the CPU are execute, fetch, decode, and idle. The control unit offers the essential signal interaction for all states' expected functions to be carried out. The address of the memory location from which the instruction is to be fetched is indicated by the 16-bit program counter. Unless a "JUMP" instruction is encountered, the program counter is increased by one after the current instruction has been completed.



The program counter is increased or decreased by the amount indicated by the offset when the "JUMP" instruction is carried out.



Instruction memory contains the written instructions. The program counter's result is given to instruction memory as input. The program counter's output

points to an address, and an instruction memory location is fetched for that instruction. The decoded and stored instruction is sent to the instruction register. The opcode is used to assign the destination register, source register, memory location address, or instantaneous value during the decoding process. The opcode is employed by the ALU to carry out the required operation and by the control unit to produce the required control signals.

RO through R15 are the sixteen 16-bit general purpose registers. Two read(Rd) ports and one write(Wr) port are on the register array. Data is written into a register indicated by the write address when the "reg wr" signal is enabled. Otherwise, the read addresses point to two registers that are read. 11 arithmetic and logical operations are carried out by the ALU. While some of the operations call for two operands, others just require one. Registers serve as the source of all operands for ALU operations, and the multiplexer writes the operation's outcome back into the desired destination register.

Fetch Unit: Using the programme counter's current value to retrieve an instruction from the instruction memory and increase the programme counter value for the subsequent instruction is the function of the fetch unit. The fetch unit is made up of an instruction memory and a 16-bit programme counter.

Decode Unit: The purpose of the instruction decode unit is to arrange the register file and get the register data values using the 24-bit instruction delivered by the preceding fetch unit. The decode unit is made up of the instruction register, control module, and registers.

Execution Unit: The ALU, carries out the operation indicated by the opcode, makes up the execution unit of the processor.

IV. INSTRUCTION SET

The 24-bit instruction word is broad. The opcode is determined by the first four bits [23-20]. Destination register is represented by the next four bits [19-16]. The remaining 16 bits [15-0] hang down to the type of instruction which effects the immediate value, the origin registers Rx and Ry.

The following table describes the different instruction operations.

Mnemonic	Name and Format	Opcode (binary)	Assembly Format	Action
add	Add RRR-type	000	add rA, rB, rC	Add contents of regB with regC, store result in regA.
addi	Add Immediate RRI-type	001	addi rA, rB, imm	Add contents of regB with imm, store result in regA.
nand	Nand RRR-type	010	nand rA, rB, rC	Nand contents of regB with regC, store results in regA.
lui	Load Upper Immediate RI-type	011	lui rA, imm	Place the 10 ten bits of the 16-bit imm into the 10 ten bits of regA, setting the bottom 6 bits of regA to zero.
sw	Store Word RRI-type	101	sw rA, rB, imm	Store value from regA into memory. Memory address is formed by adding imm with contents of regB.
lw	Load Word RRI-type	100	lw rA, rB, imm	Load value from memory into regA. Memory address is formed by adding imm with contents of regB.
beq	Branch If Equal RRI-type	110	beq rA, rB, imm	If the contents of regA and regB are the same, branch to the address PC+1+imm, where PC is the address of the beq instruction.
jair	Jump And Link Register RRI-type	111	jair rA, rB	Branch to the address in regB. Store PC+1 into regA, where PC is the address of the jair instruction.

V. DESIGN METHODOLOGY

Typically, each PCB (printed circuit board) used in a traditional design process includes a number of chips in addition to other components. The overarching structure is first defined before these items are developed. Following the selection of the necessary integrated circuit chips, the PCBs that contain and fix the chips are designed. It is imperative to use Xilinx software because the difficult circuits carried out on separate chips and development circuit boards is typically great.

The suggested 16-bit processor's design process is depicted in the figure below. Field programmable and software configurable are FPGA devices. Hence, there is a considerable cost reduction in production and design. The processor's design entry is done by Verilog HDL code. On the basis of the data entered during the design entry step, initial synthesis creates an initial circuit. Based on inputs supplied by the designer, functional simulation is performed to confirm the functionality of the circuit. In order to create efficient circuits, logic synthesis and optimization use optimization techniques. How the optimized circuit is implemented in an FPGA chip is determined by the physical design. The estimated propagation delays in the implemented circuit are determined using timing simulation. The actual chip is set up through chip setup in order to implement the intended circuit.

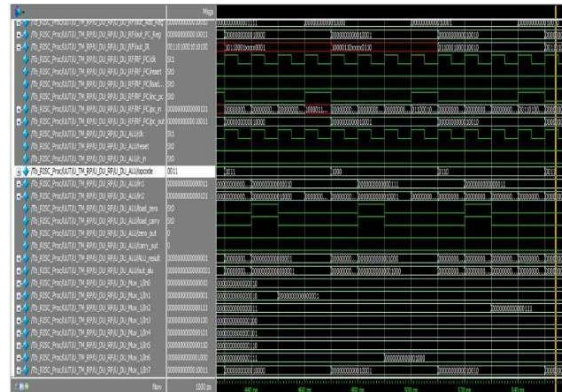
The "register file" that contains the machine's register state contains the 16-bit registers for the processor. There are three register operands for R-format instructions. One data word is written into and two data words are read from the register file. Verilog HDL code is used. On the basis of the data

entered during the design entry step, initial synthesis creates a temporary circuit. Based on inputs supplied by the designer, manageable simulation is performed to confirm the working of the circuit. In order to create efficient circuits, logic synthesis and optimization use optimization techniques. How the optimized circuit is implemented in an FPGA chip is determined by the physical design. The estimated propagation delays in the implemented circuit are determined using timing simulation. The actual chip is set up through chip setup in order to implement the intended circuit. The 16-bit registers for the CPU are contained in the "register file" that houses the machine's register state. R-format instructions, consists of three register operands available. Each instruction's One data word is written to and two are read from the register, which is for a file. The register number inputs, which identify one of the 16 registers, are only 4 bits wide, although one input and two output buses are 16 bits wide. When an instruction is fetched from the fetch stage, the opcode is sent to the control unit. The instruction's register address is used to address the two-port register file. In one clock cycle, the two-port register can handle two independent reads and one write. The instructions are decoded during the decode step, after which the register file is opened to read the registers. Registers R1 and R2, are read using the outputs of the general-purpose registers. The execution step follows the instructions decoding stage with calculations.

VI. SIMULATION RESULTS

The simulation outcome of the suggested 16-bit RISC CPU. The two 16-bit inputs from the processor are memory add and CPU output (15:0). ALU result and memory address are represented by the variables cpu output (15:0) and memory add (15:0), respectively. Originally, the 16-bit operand's decimal equivalent of "7" is copied into register address R1. Similar to R1, R2 contents are kept in memory at location 20. Now, the elements of R1 and R2 are combined to create the result "12," which is then placed in R1. The memory address '21' receives a copy of the additional result from R1. Similar to this, the contents of R2's "5" are deducted from R1's "12" modified contents to create the result "7," which is then saved in R1. The output

of the sub(-) in R1 is copied to memory at address '22'. The result of multiplying the adjusted contents of R1 and R2, "7" and "5," is "35," which is then put in R1. The memory address '23' receives a copy of the multiplication result from R1. The result of dividing the "35" updated contents of R1 by the "5" original contents of R2 is "12," which is then kept in R1. The memory address '24' contains a copy of the division result from R 1.



VII. REFERENCES

- [1] Mamun, B., I. Shabiul, and Sulaiman (2002). A VHDL-based single-cycle MIPS RISC processor design. Penang, Malaysia, 2002, pp. 199-203.
- [2] Hamblen J. A Pipelined RISC Computer System Prototype Utilizing Hardware Emulation, Simulation, and Synthesis. Georgia's Atlanta.
- [3] Zainalabedin N. Utilizing VHDL for Processor Unit Modeling and Design. Boston, Massachusetts, pp. 315-326.
- [4] OROCHT's 2007 outline. International Workshop on Innovative Architecture for Next generation Processors and Systems, A Multiple Instruction Set Executable SMT Processor.
- [5] Takanori, Masaaki, and Satoshi, 2005. An architecture for multi-threaded processors based on the continuation model. Inventive Architecture for Next-Generation High-Performance Processors and

Systems Proceedings. Kasuga-Koen, Fukuoka, Japan; Kasuga.

[6] Instruction-Based Selftesting of Delay Faults in Pipelined Processors, IEEE Transaction on VLSI Systems, vol. 14, no. 11, pp. 1203-1215, 2006. Virendra S. and Michiko I.

[7] Computer Organization, Hamacher V. and Zaky, 2002. Fifth edition, McGraw-Hill Companies, New York.

[8] Tony G. and Frank V., 1999. Unified Hardware/Software Methodology for Embedded System Design, Department of Computer Science and Engineering, University of California, draught.

[9] Computer Organization & Design, Morgan Kaufmann Publishers, 1999, Patterson A. and Hennessy J. VHDL Tutorial, Learn by Example. Weijun Z. 2001.

[11] Zynq FPGA Family data sheet, 2003. XILINX, DS001-2 (V2.2)

PERFORMANCE METRICS OF 16

by Marreddy Anuhya

Submission date: 14-Apr-2023 07:40AM (UTC-0500)

Submission ID: 2064364341

File name: Performance_Metrics_of_16_3.pdf (535.1K)

Word count: 2494

Character count: 13757

Performance Metrics of 16-Bit VLSI RISC Architecture

Jampula Saiteja¹
Student
Department of Electronics and
Communication Engineering,
Sathyabama Institute of Science and
Technology,
Chennai, India.
Saitejjampula143@gmail.com

Gundlapudi Surya Teja²
Student
Department of Electronics and
Communication Engineering,
Sathyabama Institute of Science and
Technology,
Chennai, India.
Suryatej0476@gmail.com

M. Subramoniam³
Associate Professor
Department of Electronics and
Communication Engineering,
Sathyabama Institute of Science and
Technology,
Chennai, India.
Subramoniam.viru@gmail.com

Abstract---

Reduced Instruction Set Computer. For quick execution, the Reduced Instruction Set Computer (RISC) architecture is used. Register-to-register operations are used in RISC instructions. Registers for various operations and storage are where the results are kept. The findings are transferred to memory via instructions. Every single step is completed inside the registers. Since all RISC instructions provide unambiguous register addressing, fewer addressing modes are required. RISC can utilize simple, flexible syntax and decode it with ease. In this situation, word boundaries can be used to set the instruction's alignment and length. The RISC processor tries to carry out one instruction for each clock signal. Pipelining is the process of using one or more instructions' fetch, decode, and execute phases on top of one another to achieve this. Because RISC utilizes a significant portion of the registers on the processor unit, it runs programs faster than CISC. Building a 16-bit RISC processor and modelling each component in Verilog HDL are required for this project. Harvard architecture is the foundation of the CPU. The very simple instruction set used in this circumstance makes it possible to comprehend the type of hardware that should be able to correctly execute the instructions. In addition to the adders, registers, and other sequential and combinatorial building blocks of a processor, more complex construction elements like ALU and memory have been constructed and simulated.

1. INTROCUCTION

People started looking at other options when the controller architecture in CISC became more difficult and the performance was also below expectations. It had been discovered that speed is eliminated when a CPU communicates with memory. The only way to improve CPI was to keep the instruction set as basic as possible. Simple in

terms of appearance rather than functionality. Because of this, very few instructions—probably just load and store—in a typical RISC architecture require the CPU to access data from memory. In the end, pipelining increased speed by a new dimension just by adding a few extra registers, which lowers CPI and raises throughput. As a result, the instruction can be efficiently operated out in a single clock cycle.

Many people mistakenly believe that "Reduced Instruction Set Computer" (RISC) means that instructions are only reduced in number. However, RISC instruction sets have actually grown over time and some RISC CPUs now have more instructions than certain Complex Instruction Set Computers (CISC) CPUs. Such project By limiting each instruction to a maximum of one data memory cycle, RISC CPUs simplify instruction execution in contrast to the "complex instructions" of CISC CPUs, which may need numerous cycles. This is what is meant by the name "Reduced" in this context. At the moment, RISC or CISC designs are used by the majority of microprocessors on the market. Research has demonstrated that RISC design, which uses simplified machine instructions for commonly used processes, can greatly improve computer performance. The expert RISC-based systems commonly exhibit the following traits.

1) Pre-fetching: Pre-fetching is the process of retrieving the subsequent instruction to be carried out and holding it in a queue before the current instruction is completed.

2) Pipelining: Pipelining enables issuing an instruction before the one that is presently being executed has finished.

3) Superscalar operation: A processor that can send out several instructions at once is said to be using superscalar operation.

II. LITERATURE SURVEY

At the 2015 International Conference on Control, Instrumentation, Communication, and Computational Technologies (ICICCT), Jikku Jeemon presented this study. The task entails designing a 16-bit RISC processor and utilising Verilog HDL to model each of its component parts. The Harvard architecture-based processor uses a simple instruction set and reveals the hardware requirements for effective instruction execution. Advanced components like ALU and memory were built and simulated in addition to sequential and combinational building blocks like adders and registers.

Published in (IJCRT) [International Journal of creative Research thoughts] In this study, the ALU modelling that was completed for this project is fully structural and begins with half adders. Finally, ALU's semi-custom layout was created. A behavioural method has been used to describe complex building components like memory, as opposed to a structural approach. ModelSim and Intel Quartus Prime 18.1 have been used as the project's primary tools. The FPGA device technology that is being pursued for synthesis is ALTERA.

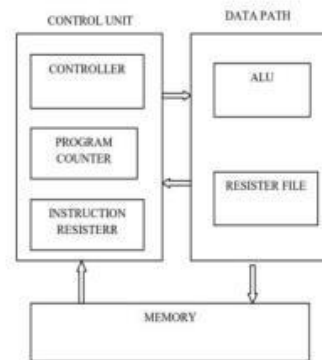
Low-power 16-bit RISC processor using VHDL and FPGA. The proposed processor architecture is based on the Harvard architecture, has a three-stage pipeline, a unique instruction set, and uses power-saving techniques including clock gating and data gating. It also has separate instruction and data memory. The processor was implemented on a Xilinx Spartan-3 FPGA, and the results show that the processor operates at a clock frequency of 50MHz and consumes only 32mW of power. The performance of the processor was evaluated using a set of benchmark programs, and the results demonstrate that the processor can execute the benchmark programs with a reasonable speed and efficiency, while consuming significantly less power compared to other existing processors. The proposed low-power 16-bit RISC processor is suitable for use in low-power embedded systems, such as wireless sensor networks, portable devices, and Internet of Things (IoT) applications. In the study, a 16-bit RISC processor design using Verilog HDL and the RISC (Reduced Instruction Set Computer) paradigm is presented.

The ALU, Controller, Register files, and Data memory unit are some of the different building parts that make up the CPU. It uses a single shared memory for both data and instructions in the van Neumann architecture. The RISC processor design is kept simple with only 15 instructions for investigation purposes. Logical verification is performed using Modelsim, and the design is analyzed using Xilinx ISE tool.

Presented in [International Journal of Engineering Trends in Electrical and Electronics (IJETEE)]. The paper addresses the widespread use of RISC, or Reduced Instruction Set Computers, in applications related to science and engineering. The development and implementation of a 32-bit RISC processor for embedded and portable applications utilising the XILINX VIRTEX4 Tool is the major objective of the article. The design's main objective is to boost the processor's overall performance and speed. One key feature of RISC processors is their simplicity and support for load/store architecture.

III. ARCHITECTURE

The primary objective of the project is to develop a



The proposed design for a 16-bit RISC processor with a Harvard architecture aims to minimize the number of functional components needed. Illustrated in Figure 1, The processor structure consists of sixteen 16-bit general purpose registers, a 16-bit programme counter, a 24-bit instruction register, a 3-bit flag register for carry, zero, and parity, and a 16-bit ALU performing 11 arithmetic and logical operations. The CPU operates in four states: execute, fetch, decode, and idle, each with specific functions controlled by the control unit.

The program counter points to the memory address from which instructions are fetched, and is incremented by one after each instruction, unless a "JUMP" command is encountered. With the "JUMP" instruction, the programme counter can be modified by a given offset. The result of the programme counter is used as input while instructions to be executed are placed in the instruction memory. The program counter's output indicates the memory address from which an instruction is fetched and stored in the instruction register. During the decoding process, the opcode is used to assign the destination register, source register, memory location address, or instantaneous value. The control unit generates the necessary control signals while the ALU uses the opcode to carry out the required operation.

The sixteen 16-bit general purpose registers are R0 through R15. The register array has two read ports (Rd) and one write port (Wr). When the "reg wr" signal is enabled, data is loaded into a register that has been designated using the write address. If not, two registers are pointed to by the read addresses. 11 arithmetic and logical operations are carried out by the ALU. While some of the operations call for two operands, others just require one. Registers serve as the source of all operands for ALU operations, and the multiplexer writes the operation's outcome back into the desired destination register.

Fetch Unit: The fetch unit's job is to retrieve an instruction from memory using the current value of the programme counter and to raise the value of the programme counter for the next instruction. An instruction memory and a 16-bit programme counter are the two components that make up the fetch unit.

Decode Unit: To store and retrieve register data values, the register file comprises 24 units. To decode instructions acquired from the fetch unit, the instruction register, control module, and registers work as a unit.

Execution Unit: The ALU, carries out the operation indicated by the opcode, makes up the execution unit of the processor.

IV. INSTRUCTION SET

The 24-bit instruction word is broad. The opcode is determined by the first four bits [23–20]. Destination register is represented by the next four bits [19–16]. The remaining 16 bits [15–0] hang

down to the type of instruction which effects the immediate value, the origin registers Rx and Ry.

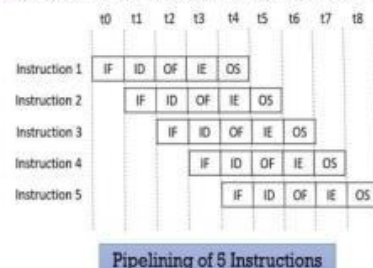
The following table describes the different instruction operations.

Mnemonic	Name and Format	Opcode (Binary)	Assembly Format	Action
add	Add RRR-type	000	add rA, rB, rC	Add contents of regB with regC, store result in regA.
addi	Add Immediate RRR-type	001	addi rA, rB, imm	Add contents of regB with imm, store result in regA.
rand	Rand RRR-type	010	rand rA, rB, rC	Rand contents of regB with regC, store result in regA.
li	Load Upper Immediate RRR-type	011	li rA, imm	Place the 16-bit value of the 16-bit imm into the 16 bits of regA, setting the bottom 8 bits of regA to zero.
sw	Store Word RRR-type	101	sw rA, rB, imm	Store value from regA into memory. Memory address is formed by adding imm with contents of regA.
lw	Load Word RRR-type	100	lw rA, rB, imm	Load value from memory into regA. Memory address is formed by adding imm with contents of regA.
beq	Branch if Equal RRR-type	110	beq rA, rB, imm	If the contents of regA and regB are the same, branch to the address PC+imm, where PC is the address of the first instruction.
jlr	Jump And Link Register RRR-type	111	jlr rA, rB	Branch to the address in regA. Store PC+1 into regA, where PC is the address of the first instruction.

V. DESIGN METHODOLOGY

Typically, each PCB (printed circuit board) used in a traditional design process includes a number of chips in addition to other components. The overarching structure is first defined before these items are developed. The PCBs that hold and fix the integrated circuit chips are designed after the appropriate integrated circuit chips have been chosen. Because complex circuits are frequently implemented on separate chips and development circuit boards, it is essential to use Xilinx software.

The suggested 16-bit processor's design process is depicted in the figure below. Field programmable and software configurable are FPGA devices. Hence, there is a considerable cost reduction in production and design. The processor's design entry is done by Verilog HDL code is used. On the basis



of the values entered during the first step of design, A preliminary circuit is made during first synthesis. Functional simulation is used to verify the circuit's functionality using inputs provided by the designer. Circuits that are effective must be designed, logic

synthesis and optimization use optimization techniques. How the optimised circuit is implemented in an FPGA chip is determined by the physical design. The estimated propagation delays in the implemented circuit are determined using timing simulation. The actual chip is set up through chip setup in order to implement the intended circuit.

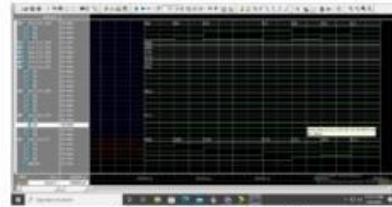
The "register file" that contains the machine's register state contains the 16-bit registers for the processor. There are three register operands for R-format instructions. One data word is written into and two data words are read from the register file. Verilog HDL code is used. On the basis of the data entered during the design entry step, initial synthesis creates a temporary circuit. Based on inputs supplied by the designer, manageable simulation is performed to confirm the working of the circuit. In order to create efficient circuits, logic synthesis and optimization use optimization techniques. How the optimised circuit is implemented in an FPGA chip is determined by the physical design. The estimated propagation delays in the implemented circuit are determined using timing simulation. The actual chip is set up through chip setup in order to implement the intended circuit.

R-format instructions make use of three register operands in the register file, which houses the CPU's 16-bit registers and keeps track of the register state. Each instruction's data word is written to the register file, while two data words are read from it. Despite the input and output buses being 16 bits wide, the register number inputs used to identify registers are only 4 bits wide. The register addresses of the instruction are utilised to access the two-port register file when it is fetched and the opcode is transmitted to the control unit. In a single clock cycle, the two-port register file can handle two independent reads and one write. Instructions are decoded during the decode step, and the outputs of the general-purpose registers are used to read the contents of registers R1 and R2 from the register file. The execution step then proceeds with the calculations.

VI. SIMULATION RESULTS

The suggested 16-bit RISC CPU's simulation output. The processor's two 16-bit inputs are the CPU output (15:0) and memory add. The variables cpu output (15:0) and memory add (15:0), respectively, stand in for the ALU result and memory address. Originally, the 16-bit operand's decimal equivalent of "7" is copied into register address R1.. Similar to R1, R2

contents are kept in memory at location 20. Now, the elements of R1 and R2 are combined to create the result "12," which is then placed in R1. The memory address '21' receives a copy of the addition result from R1. Similar to this, the contents of R2's "5" are deducted from R1's "12" modified contents to create the result "7," which is then saved in R1.



The output of the sub(-) in R1 is copied to memory at address '22'. The result of multiplying the adjusted contents of R1 and R2, "7" and "5," is "35," which is then put in R1. The memory address '23' receives a copy of the multiplication result from R1. The result of dividing the "35" updated contents of R1 by the "5" original contents of R2 is "12," which is then kept in R1. The memory address '24' contains a copy of the division result from R1.



VII. REFERENCES

- [1]Mamun, B., I. Shabiul, and Sulaiman (2002). A VHDL-based single-cycle MIPS RISC processor design. Penang, Malaysia, 2002, pp. 199–203.
- [2] Hamblen J. A Pipelined RISC Computer System Prototype Utilizing Hardware Emulation, Simulation, and Synthesis. Georgia's Atlanta,

- [3] Zainalabedin N. Utilizing VHDL for Processor Unit Modeling and Design. Boston, Massachusetts, pp. 315–326.
- [4] OROCHI's 2007 outline. International Workshop on Innovative Architecture for Next generation Processors and Systems, A Multiple Instruction Set Executable SMT Processor.
- [5] Takanori, Masaaki, and Satoshi, 2005. An architecture for multi-threaded processors based on the continuation model. Inventive Architecture for Next-Generation High-Performance Processors and Systems Proceedings. Kasuga-Koen, Fukuoka, Japan; Kasuga.
- [6] Instruction-Based SelfTesting of Delay Faults in Pipelined Processors, IEEE Transaction on VLSI Systems, vol. 14, no. 11, pp. 1203–1215, 2006. Virendra S. and Michiko I.
- [7] Computer Organization, Hamacher V. and Zaky, 2002. Fifth edition, McGraw-Hill Companies, New York.
- [8] Tony G. and Frank V., 1999. Unified Hardware/Software Methodology for Embedded System Design, Department of Computer Science and Engineering, University of California, draught.
- [9] Computer Organization & Design, Morgan Kaufmann Publishers, 1999, Patterson A. and Hennessy J. VHDL Tutorial, Learn by Example. Weijun Z. 2001.
- [11] Zunq FPGA Family data sheet, 2003. XILINX, DS001-2 (V2.2)

PERFORMANCE METRICS OF 16

ORIGINALITY REPORT

12%	10%	4%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.ijert.org Internet Source	4%
2	www.ijcaonline.org Internet Source	3%
3	R. M. Gomathi, Malle Rahulkumar Reddy, Madithati Veera Bhadra Reddy, T. Anandhi, A. Sivasangari, P. Ajitha. "Spam Review Detection using Linguistic Methods for Specified user in Twitter", 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), 2022 Publication	1%
4	Submitted to Open University of Mauritius Student Paper	1%
5	www.mitpublications.org Internet Source	1%
6	Submitted to University of Southampton Student Paper	1%
7	www.semanticscholar.org Internet Source	<1%

8

www.coursehero.com

Internet Source

<1 %

9

Soliman, Mostafa I.. "Design, implementation, and evaluation of a low-complexity vector-core for executing scalar/vector instructions", Journal of Parallel and Distributed Computing, 2013.

Publication

<1 %

Exclude quotes On

Exclude matches Off

Exclude bibliography On