A Novel 32 Bit RISC-V Based Pipelined Processor Design Using Verilog

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in ELECTRONICS AND COMMUNICATION ENGINEERING

by

SANDEEP.P (38130193)

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 I 12B Status by UGC I Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119

March – 2022

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC I 12B Status by UGC I Approved by AICTE

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in

DEPARTMENT OF ELECTRICAL AND ELECTRONICS

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of Sandeep.p(38130193) who have done the Project work as a team who carried out the project entitled "A Novel 32 Bit RISC-V Based Pipelined Processor Design Using Verilog" . Under my supervision from December 2022 to January 2022.

Internal Guide

(Dr.V.Vedanarayanan)

Head of the Department

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

Submitted for Viva voce Examination held on _____

Internal Examiner **External Examiner**

DECLARATION

I **SANDEEP.P(38130193)** hereby declare that the Project Report entitled "**A Novel 32 Bit RISC-V Based Pipelined Processor Design Using Verilog**" done by me under the guidance of Dr.V.Vedanarayanan at Sathyabama Institute of Science and Technology, Chennai- 600119 is submitted in partial fulfillment of the requirements for the award of Bachelor Engineering degree in Electronicsand Communication Engineering.

DATE:

PLACE: **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to Dr. N. M. Nandhita, Dean, School of Electrical and Electronics Engineering and Dr. T. RAVI, Head of the Department, Dept. of Electrical and Electronics Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr.V.Vedanarayanan for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of ELECTRICAL AND ELECTRONICS who were helpful in many ways for the completion of the project.

ABSTRACT

The main goal of this study is to develop a 32-bit pipelined processor with several clock domains based on the RISC-V (open source RV32I Version 2.0) ISA. To minimise the complexity of the instruction set and speed up the execution time per instruction, RISC (Reduced Instruction Set Computer) is a type of processor that uses less hardware than CISC (Complex Instruction Set Computer) is used. Furthermore, we constructed this processor with five levels of pipelining, resulting in parallelism in instruction execution. With the aid of necessary block diagrams, all of the processes are well described.

Multiple clock domains employing two clock sources are used to ensure that variable delays such as clock skew and metastability are avoided within the stage pipeline registers. Quartus Prime was used to design and synthesis this processor, which was written in Verilog HDL. ModelSim was used to verify this design, and all of the instructions have been thoroughly checked. Furthur the processor is implemented on the “ALTERA Cyclone 10 LP” board for calculating the device utilization.

TABLE OF CONTENTS

TITLE	PAGE NO
ABSTRACT	**V**
LIST OF ABBREVIATIONS	**X**
1. INTRODUCTION	1
1.1 Generations of microprocessors	1
1.2 Types of microprocessors	2
1.3 RISC vs CISC	3
1.4 Importance of CPI	5
1.5 Hazards Encountered	7
2. LITERATURE REVIEW	**8**
2.1 Implementation of a 32-bit MIPS based RISC processor using Cadence	8
2.2 Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL	9
2.3 Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL	10
2.4 Single cycle RISC-V micro architecture processor and its FPGA prototype	11

2.5 32-Bit RISC processor with floating point unit for DSP applications 12

2.6 A RISC-V instruction set processor-micro-architecture design and analysis 13

2.7 Advanced low power RISC processor design using MIPS instruction set 14

2.8 ASIC design of MIPS based RISC processor for high performance 15

2.9 Design and development of FPGA based low power pipelined 64-Bit RISC processor with double precision floating point unit 16

2.10 Design of an 8-bit five stage pipelined RISC microprocessor for sensor platform application 17

2.11 Design of FPGA based 8-bit RISC controller IP core using VHDL 18

3. INSTRUCTION SET ARCHITECTURE USED 19

3.1 RISC V 19

3.1.1 R-type RV32I Instruction Format 20

3.1.2. I-type RV32I Instruction Format 21

3.1.3. S-type RV32I Instruction Format 22

3.1.4. B-type RV32I Instruction Format 23

3.1.5. U-type & J-type RV32I Instruction Format 24

3.2 INSTRUCTIONS SUPPORTED 26

3.2.1 Arithmetic Operations 27

3.2.2 Logical Operations 28

3.2.3 Data Transfer Operations 29

3.2.4 Control Transfer Instructions 30

4. CONSTRUCTION OF THE PROCESSOR 31

4.1 PIPELINING OF A PROCESSOR 31

4.2 Organization Of The RISC-V Processor 33

4.2.1 Instruction memory 34

4.2.2 Register file 35

4.2.3 Data Memory 36

4.2.4 Instruction Fetch Stage 37

4.2.5 Decode Stage 38

4.2.6 Execute Stage 39

4.2.7 Memory Access Stage 40

4.2.8 Write Back Stage 41

4.3. ELIMINATION OF PIPELINE HAZARDS 42

4.3.1 STRUCTURAL HAZARDS 42

4.3.2 DATA HAZARDS 44

4.3.3 CONTROL HAZARDS 46

4.4. CONSTRUCTION OF MULTIPLE CLOCK DOMAINS 48

4.4.1 CAUSES OF METASTABLE CONDITIONS 48

4.4.2 PREVENTION OF METASTABLE CONDITIONS 51

5. SIMULATION AND SYNTHESIS RESULTS 53

5.1 SOFTWARE USED 53

5.2 SIMULATION RESULTS 58

5.2.1 ARITHMETIC TYPE INSTRUCTIONS 59

5.2.2 LOGICAL TYPE INSTRUCTIONS 60

5.2.3 IMMEDIATE TYPE INSTRUCTIONS 61

5.2.4 STORE TYPE INSTRUCTIONS 62

5.2.5 LOAD TYPE INSTRUCTIONS 63

5.2.6 BRANCH TYPE INSTRUCTIONS 64

5.2.7 JUMP TYPE INSTRUCTIONS 65

5.3 IMPLEMENTATION ON FPGA 66

6. CONCLUSION AND FUTURE SCOPE 67

6.1 CONCLUSION 67

6.2 FUTURE SCOPE 68

7. REFERENCES 69

LIST OF ABBREVIATIONS

S.NO	ABBREVIATION	FULL FORM
1	RISC	Reduced Instruction Set Computer
2	CISC	Complex Instruction Set Computer
3	LUT	Look Up Tables
4	FPGA	Field Programmable Gate Array
5	CPI	Cycles per instruction
6	CLK	Clock
7	ISA	Instruction Set Architecture

1.INTRODUCTION

Transistor was invented in 1948 (23 December 1947 in Bell lab). IC was invented in 1958 (Fair Child Semiconductors) By Texas Instruments J Kilby. The first microprocessor was invented by INTEL(INTegrated ELectronics).

1.1 Generations of microprocessors:

First-generation – From 1971 to 1972, the first generation of microprocessors appeared, including the INTEL 4004, Rockwell International PPS-4, and INTEL 8008 among others.

Second generation – From 1973 until 1978, the second generation of 8-bit microprocessors was developed. Processors such as the INTEL 8085, Motorola 6800, and 6801 were developed.

Third generation – The third generation saw the introduction of 16-bit processors such as the INTEL 8086/80186/80286, Motorola 68000 68010, and others. This generation employed the HMOS technology from 1979 to 1980.

Fourth generation – Between 1981 and 1995, the fourth generation existed. 32-bit CPUs based on HMOS manufacturing were created. This generation's prominent processors include the INTEL 80386 and Motorola 68020.

Fifth-generation –We've been in the fifth generation since 1995. There were 64-bit CPUs such as the PENTIUM, Celeron, twin, quad, and octa-core processors.

1.2 Types of microprocessors

Complex instruction set microprocessor –

The processors are built with the goal of reducing the number of instructions per programme while ignoring the number of cycles per instruction. Because the code is relatively short and extra RAM is utilised to hold the instructions, the compiler is used to translate a high-level language to assembly-level language.

These processors can download, upload, and recall data from memory, among other things. This microprocessor can also do sophisticated mathematical calculations in a single instruction, in addition to these functions.

Example: IBM 370/168, VAX 11/780

Reduced instruction set microprocessor –

These processors are built to do specific tasks. They are meant to use a reduced instruction set to reduce execution time. They are capable of carrying out minor tasks in response to particular commands. These processors are more efficient at completing commands. To implement a result at uniform execution time, they simply need one clock cycle. There are a lot of registers and a lot of transistors. The LOAD and STORE instructions are used to access the memory location.

Example: Power PC 601, 604, 615, 620

Superscalar microprocessor –

These processors are capable of handling multiple tasks at once. ALUs and multiplier-like arrays can both benefit from them. They feature several operating units and execute multiple orders to complete tasks.

Application-specific integrated circuit –

These processors, like personal digital assistant computers, are application-specific. They are created in accordance with strict guidelines.

Digital signal multiprocessor –

Signals such as analogue to digital and digital to analogue are converted using these processors. These processors' chips are found in a variety of devices, including RADAR SONAR home theatre systems.

1.3 RISC vs CISC

Over the past decades, microprocessors and microcontrollers has been constructed around two types of architectures, Reduced Instruction Set computer and Complex Instruction Set Computer.

CISC instructions are variable in length and are encoded for doing more micro operations per instruction. As a result, the complex architecture of CISC processors makes instructions take a longer time to execute. Since all instructions of a RISC processor have the same instruction length, the decoding process becomes easier compared to a CISC processor.

RISC is widely used due to its efficient architecture which can be used for low power and high speed processing application. It supports very few addressing modes, LOAD and STORE instructions are the only instructions that are used to access the external memory. Hence RISC processors are mainly dependent on software and CISC processors are mainly dependent on hardware for executing complex tasks.

Reduced Instruction Set Architecture (RISC) –

The fundamental goal is to make hardware simpler by employing an instruction set that consists of only a few basic loading, evaluating, and storing stages. A store command does the same thing as a load command: it stores the data.

Complex Instruction Set Architecture (CISC) –

The basic notion is that a single instruction will handle all loading, evaluating, and storing operations, similar to how a multiplication command will handle loading, evaluating, and storing data, which is why it's complicated.

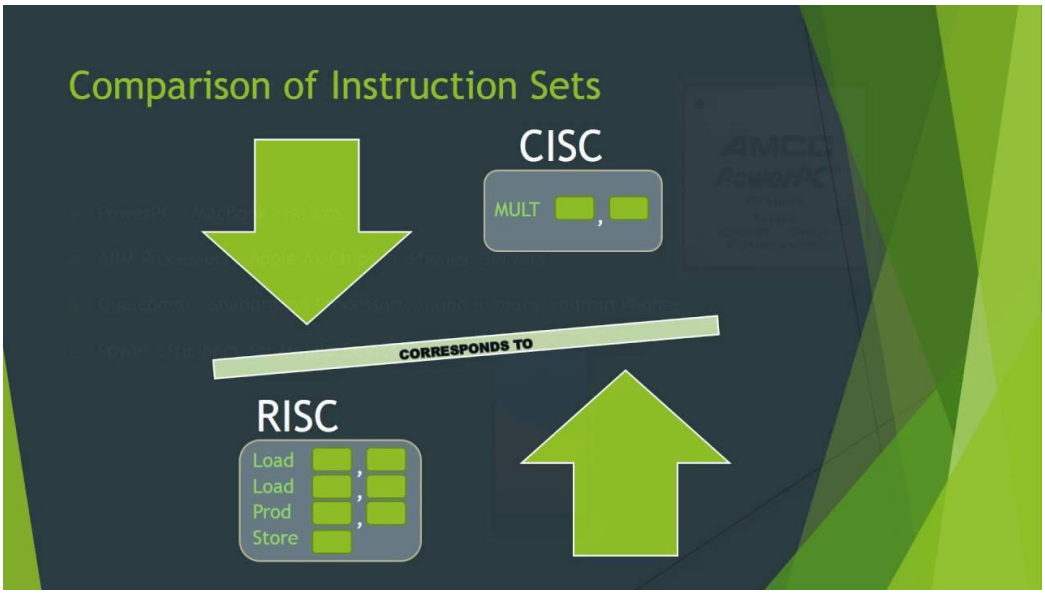

Fig 1.1: RISC vs. CISC

Cycles per instruction (also known as clock cycles per instruction, clocks per instruction, or CPI) is the average number of clock cycles per instruction for a programme or programme fragment in computer architecture. [1] It's the inverse of instructions per cycle multiplied by a factor of two.

1.3 Importance of CPI

The average of Cycles Per Instruction in a given process is defined by the following:

$$CPI = \frac{\Sigma_i (IC_i)(CC_i)}{IC}$$

Where ICi is the number of instructions for a given instruction type i, CCi is the clock-cycles or that instruction type and IC is the total instruction count. The summation sums over all instruction types for a given benchmarking process.

Every processor architecture design's primary aim is to keep Clock per Instruction (CPI) close to 1 which can be always challenging. To increase the throughput we have implemented the processor using 5 stages pipelined architecture, by implementing separate stages for Fetch instructions, Decoding, Arithmetic operations, Memory access and write back. Through pipelining on each cycle, an instruction can be executed.

Let us assume a classic RISC pipeline, with the following five stages:-

1) Instruction fetch cycle (IF).
2) Instruction decode/Register fetch cycle (ID).
3) Execution/Effective address cycle (EX).
4) Memory access (MEM).
5) Write-back cycle (WB).

Each step takes one clock cycle, and instructions are passed through the stages in the order they are received. In a multi-cycle processor without pipelining, a new instruction is fetched in stage 1 only after the preceding instruction has completed at stage 5, resulting in a clock cycle count of five (CPI = 5 > 1). The processor is considered to be subscalar in this instance.

Because one could theoretically have five instructions in the five pipeline stages at once (one instruction per stage), a different instruction would complete stage 5 in every clock cycle, and the average number of clock cycles it takes to execute an instruction is one (CPI = 1). The processor is said to be scalar in this situation.

The best CPI achievable with a single-execution-unit CPU is 1. With a multiple-execution-unit processor, however, CPI values can be improved even more (CPI 1). The processor is considered to be superscalar in this instance. Without pipelining, the number of execution units must be greater than the number of stages to achieve better CPI values.

With six execution units, for example, six new instructions are received in stage 1 only after the six preceding instructions have completed at stage 5, resulting in an average of 5/6 clock cycles per instruction (CPI = 5/6 1). Pipelining requires at least two execution units to achieve superior CPI values.

For example, by utilising instruction-level parallelism, two new instructions are fetched every clock cycle, resulting in two different instructions completing stage 5 every clock cycle, and the average number of clock cycles required to execute an instruction is 1/2 (CPI = 1/2 01).

The likelihood of pipeline risks during the execution of numerous instructions increases when instructions are executed in parallel.

1.4 Hazards Encountered

There are mainly three types of dependencies possible in a pipelined processor. These are:-

1) Structural Dependency

2) Control Dependency

3) Data Dependency

Structural Dependency causes Structural hazards which are encountered when multiple instructions use a common resource at the same time. This has been eliminated by implementing the processor using Harvard architecture with separate data and instruction memory, also a general-purpose memory with two read ports and one write port. Therefore, several data accesses can be performed simultaneously without conflict.

Control Dependency causes Control hazards that are encountered during the successful execution of branch and jump instructions. This can be prevented by a branch flag which goes HIGH when the branch is taken, thus following instructions following after the branch in memory and write backstage are terminated.

Data Dependency Data hazards which are encountered due to the usage of common source and destination resources in consecutive instructions, this occurs when the source for instruction is the destination for the previous instruction. Data hazards can be prevented by inserting dummy instructions during compiling using the compiler to create a gap between those instructions.

2. LITERATURE REVIEW

2.1 Implementation of a 32-bit MIPS based RISC processor using Cadence

By M. N. Topiwala and N. Saraswathi, 2014 IEEE International Conference on Avanced Communications, Control and Computing Technologies, 2014, pp. 979-983, doi: 10.1109/ICACCCT.2014.7019240.

The implementation of a 5-stage pipelined 32-bit High Performance MIPS based RISC Core is presented in this project. A RISC (Reduced Instruction Set Computer) architecture is a MIPS (Microprocessor without Interlocked Pipeline Stages). This microprocessor was created with the goal of enhancing the processor's overall speed by performing a small set of instructions. Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM), and Write Back (WB) are the five steps of the MIPS pipeline. Instruction Memory, Data Memory, ALU, Registers, and other modules are employed. The goal of this work is to incorporate a Hazard detection unit and a Data forwarding unit for a more efficient pipeline implementation. Verilog-HDL is used to create the design. The main goal is to complete the entire ASIC.

Advantages:-

1) It is a pipelined processor hence increased throughput.
2) This design has been synthesized using cadence.
3) Data forward unit.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V.

2.2 Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL

By S. P. Ritpurkar, M. N. Thakare and G. D. Korde, 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), 2014, pp. 1-6, doi: 10.1109/ICAETR.2014.7012843.

This paper's main goal is to use VHDL to develop and simulate a 32-bit MIPS (Microprocessor Interlocked Pipeline Stages) RISC (Reduced Instruction Set Computer) processor (Very High Speed Integrated Circuit Hardware Description Language). They examined the Instruction fetch module, Decoder module, Execution module, which includes 32Bit Floating point ALU, 32Bit Flag register, MIPS Instruction Set, and 32Bit general purpose registers, as well as design theory based on the 32Bit MIPS RISC Processor in this study. Furthermore, they used the pipeline approach, which involves the MIPS RISC processor's Instruction Fetch, Instruction Decode, Execution, Memory, and Write Back modules in a single clock cycle. VHDL is used to code all of the modules in the design because it is a very useful language with its notion.

Advantages:-

1) It is based on 32 bit ISA. Hence more data can be accesses reducing memory latency.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V
2) Non-pipelined hence less throughput.

2.3 Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL

By S. P. Ritpurkar, M. N. Thakare and G. D. Korde, 2015 International Conference on Advanced Computing and Communication Systems, 2015, pp. 1-6, doi: 10.1109/ICACCS.2015.7324067.

The design of a RISC (Reduced Instruction Set Computer) CPU architecture based on MIPS (Microprocessor Interlock Pipeline Stages) in VHDL is shown in this work. It also describes the processor's instruction set, architecture, and timing diagram. Converting a floating point number to a fixed number is the most common operation when working with numbers, and this may be done with the Float to Fixed Number Converter module. Finally, the suggested RISC Processor based on MIPS was designed, synthesised, and simulated using the Xilinx ISE 13.1i Simulator, with coding written in the VHDL language. Result forwarding is more efficient than stalling in resolving data hazards because it eliminates the time penalty associated with handling such disputes.

Advantages:-

1) It is based on 32 bit ISA. Hence more data can be accesses reducing memory latency.
2) It is a pipelined processor hence increased throughput.
3) It can do floating point computations.

Disadvantages:-

1) It is based on older ISA. Hence more LUT consumption than RISC-V.

2.4 Single cycle RISC-V micro architecture processor and its FPGA prototype

By D. K. Dennis et al 2017 7th International Symposium on Embedded Computing and System Design (ISED), 2017, pp. 1-5, doi: 10.1109/ISED.2017.8303926.

This work describes the creation of a fully synthesizable 32-bit processor using the open-source RISC-V (RV32I) ISA. This CPU was created with low-cost embedded devices in mind. This document also includes a RISC-V development and validation framework, as well as assembling tools and automated test suites. The result is a single-core, in-order, RISC-V processor with low hardware complexity that is not bus-based. The suggested processor is written in Verilog HDL and then prototyped on an FPGA board called the "Spartan 3E XC3S500E." The maximum functioning frequency was discovered to be 32MHz. Using the Xilinx Power Analyzer, the power is assessed to be 7.9mW.

Advantages:-

1) It is based on the latest 32 bit RISC V ISA. Hence consuming less LUTs.

Disadvantages:-

1) No hazard avoidance unit.
2) Non-pipelined hence less throughput.

2.5 32-Bit RISC processor with floating point unit for DSP applications

By S. Palekar and N. Narkhede, 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2016, pp. 2062-2066, doi: 10.1109/RTEICT.2016.7808202.

The goal of this study is to propose the design of a high-speed MIPS-based 32-bit RISC processor with a single-precision floating point unit for DSP applications. The entire architecture is geared at boosting the performance of the floating point arithmetic unit in order to improve the overall RISC processor performance. This suggested processor can do arithmetic, logical, floating point, data transmission, memory, shifting, and rotating operations. Because complex multiplication is commonly utilised in DSP applications, a separate instruction for complex multiplication has been included. When compared to ordinary complex multiplication, multiplication consumes the majority of the time, power, and area of any operation. As a result, the multiplier is reduced from four to two.

Advantages:-

1) It is based on 32 bit ISA. Hence more data can be accesses reducing memory latency.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V
2) Non-pipelined hence less throughput.

2.6 A RISC-V instruction set processor-micro-architecture design and analysis

By A. Raveendran, V. B. Patil, D. Selvakumar and V. Desalphine, 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, pp. 1-7, doi: 10.1109/VLSI-SATA.2016.7593047.

This work describes the microarchitecture design and analysis of a 5-stage pipelined RISC-V ISA compliant processor, as well as the impact of the instruction set on the pipeline and microarchitecture design. This design was evaluated in terms of instruction encoding, functionality, instruction types, decoder logic complexity, data hazard detection, register file organisation and access, pipeline functionality, branch instruction effect, control flow, data memory access, operating modes, and execution unit hardware resources. This processor was micro-architected, simulated using Blue-spec System Verilog, synthesised, and analysed on an FPGA platform and ASIC nodes in the 65nm and 130nm technology nodes. Similar attempts on RISC-V ISA based processor cores are contrasted and analysed with the synthesis results.

Advantages:-

1) It is based on 32 bit ISA. Hence more data can be accesses reducing memory latency.
2) It is a pipelined processor hence increased throughput.
3) It can do floating point computations.
4) Has hazard detection and branch prediction unit.

Disadvantages:-

1) It is based on older ISA. Hence more LUT consumption than RISC-V.

2.7 Advanced low power RISC processor design using MIPS instruction set

By P. V. S. R. Bharadwaja, K. R. Teja, M. N. Babu and K. Neelima, 2015 2nd International Conference on Electronics and Communication Systems (ICECS), 2015, pp. 1252-1258, doi: 10.1109/ECS.2015.7124785.

This paper primarily focuses on resolving some of these challenges. They are proposing an upgraded version of MIPS to address these issues. The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture is a relatively new addition to the semiconductor industry. This study focuses solely on the architecture design in Verilog HDL. This design was simulated and synthesised using cadence Inc's Nc-launch and RTL-compiler, respectively. Socencounter worked on the physical design of the synthesised architecture using the TSMC Cmos 180nm technology node's slow.lib library.

Advantages:-

1) It is a pipelined processor hence increased throughput.
2) This design has been synthesized using socencounter.
3) Data forward unit.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V.

2.8 ASIC design of MIPS based RISC processor for high performance

By A. Ashok and V. Ravi, 2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), 2017, pp. 263-269, doi: 10.1109/ICNETS2.2017.8067945.

The main goal of this study is to use Verilog to create a 32-bit MIPS RISC processor. The suggested technique, which is based on a 32-bit MIPS RISC processor, examines the many phases of instruction decoding, including the Instruction fetch module, Decoder module, Execution module, and design theory. Furthermore, the algorithm employs the pipelining idea, which involves the MIPS RISC processor's Instruction Fetch, Instruction Decode, Execution, Memory, and Write Back modules in a single clock cycle. In general, the processor brings information from memory to work with a large number of instructions every second. Hardware interlocks occur when the processor speed does not correspond to the memory access speed. There is one more thing I'd like to add to this.

Advantages:-

1) It is a pipelined processor hence increased throughput.
2) This design has been synthesized using soc encounter.
3) Data forward unit.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V.

2.9 Design and development of FPGA based low power pipelined 64-Bit RISC processor with double precision floating point unit

By J. V. Kumar, B. Nagaraju, C. Swapna and T. Ramanjappa, 2014 International Conference on Communication and Signal Processing, 2014, pp. 1054-1058, doi: 10.1109/ICCSP.2014.6950008.

This work describes a low-power pipelined 64-bit RISC processor with Floating Point Unit based on FPGA. This processor is designed specifically for fixed and floating point arithmetic operations, as well as branch and logical functions. Because dynamic branch prediction is used, pipelining will not flush when a branch instruction occurs. This will improve the flow of instructions via the pipeline while also ensuring high efficiency. Clock gating is a technique used in RTL coding to reduce dynamic power. Double Precision floating point arithmetic operations such as addition, subtraction, multiplication, and division are also implemented in this article. By utilising floating point computations, this architecture has become vital and growing significant in various applications like as signal processing, graphics, and medicine. In the hardware description, the relevant code is written.

Advantages:-

1) It is based on 32 bit ISA. Hence more data can be accesses reducing memory latency.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V
2) Non-pipelined hence less throughput.

2.10 Design of an 8-bit five stage pipelined RISC microprocessor for sensor platform application

By R. J. L. Austria, A. L. Sambile, K. M. Villegas and J. N. T. Tabing, TENCON 2017 - 2017 IEEE Region 10 Conference, 2017, pp. 2110-2115, doi: 10.1109/TENCON.2017.8228209.

This study describes a low-power pipelined 64-bit RISC processor with Floating Point Unit based on FPGA technology. This processor is designed for fixed and floating point numerical arithmetic, as well as branch and logical tasks. Because dynamic branch prediction is used to achieve pipelining, it will not flush when a branch instruction is sent. This will improve the efficiency of the instruction stream. Using the clock gating technique in RTL coding, one can lower the dynamic power. Addition, subtraction, multiplication, and division are also implemented in this work using Double Precision floating point arithmetic. Because floating point operations are used in many applications, such as signal processing, graphics, and medicine, this architecture has become vital and increasingly important. The hardware description contains all of the necessary code.

Advantages:-

1) It is verified using UVM.
2) It is a pipelined processor hence more throughput.
3) It is synthesized using synopsys tool.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V.
2) It is a 8bit processor hence more memory latency,

2.11 Design of FPGA based 8-bit RISC controller IP core using VHDL

By R. Aneesh. and K. Jiju., 2012 Annual IEEE India Conference (INDICON), 2012, pp. 427-432, doi: 10.1109/INDCON.2012.6420656.

The design, development, and implementation of an 8-bit RISC controller IP core are described in this study. The controller was created using the Hardware Description Language for Very High Speed Integrated Circuits (VHDL). Speed, power, and area are the design restrictions. This controller is ideal for tiny applications and efficient for particular applications. Fetch, Decode, Execute, and a stage control unit make up this non-pipelined controller. It has a built-in data and programme memory. It also comes with four ports for connecting to other I/O devices. Basic units have been modelled using behavioural programming utilising a hierarchical approach. Structural programming is used to combine the fundamental parts. The ALTERA STRATIX II FPGA was used to implement the design.

Advantages:-

1) Less LUT consumption since it is a 8 bit processor.
2) It is based on a microcontroller.

Disadvantages:-

1) It is based on a older ISA. Hence more LUT consumption than RISC-V.
2) It is a 8bit processor hence more memory latency.
3) It is not pipelined hence less throughput.

3. INSTRUCTION SET ARCHITECTURE USED

3.1 RISCV

The RISC-V (RV32I) instruction set has a fixed length of 32 bits, which must be aligned to 32 bit boundaries. It is designed to form a sufficient compile target and support modern OS environments. It was constructed in a way that it reduces the hardware needed for minimum implementation. It follows a little endian format, where the lowest address contains the LSB part of the particular word.

The format used in this project is RV32I which is v2.0 of RISC V. Which is an optimized ISA for creating RISC machines. This ISA can support almost all modern operations and features. It has 32 general purpose registers reg0 to reg31 and reg0 is hardwired to the constant 0.

There is one additional user accessible register known as the program counter that holds the address of the next instruction to execute. The program counter is of 32 bits in length, at the positive edge of the clock the program counter is incremented. The number of the increment value is dependent on the instruction memory. In this project the instruction memory is word addressable so the program counter is incremented by one.

RISC V was mainly chose because it can be easily pipelined and consumes less hardware and power. Since it is mainly software oriented, we need techniques like loop unrolling and compiler scheduling to optimize the run time of this processor.

There are six instruction formats in the RV32I instruction set: R-type, U-type, I-type, B-type, J-type. and S-type shown in Figure 3.2. All of the types are explained in the following section

3.1.1. R-type RV32I Instruction Format

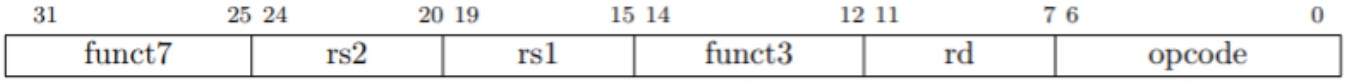

Fig 3.1: R-Type RV32I V 2.0 Instruction Format

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

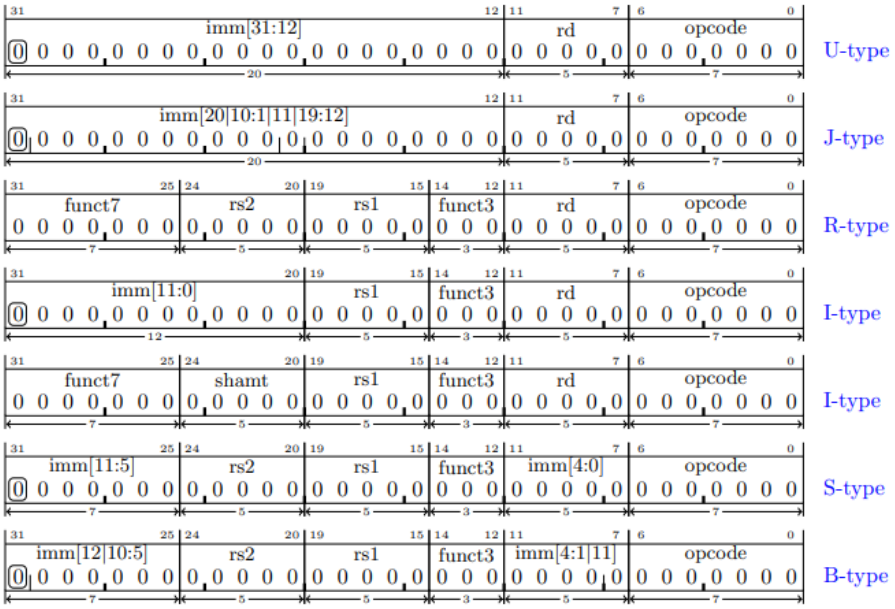

Fig 3.2: ***RISC-V instruction formats.***

3.1.2 I-type RV32I Instruction Format

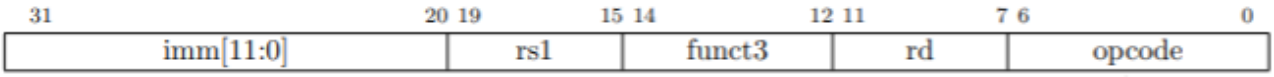

Fig 3.3: I-Type RV32I V 2.0 Instruction Format

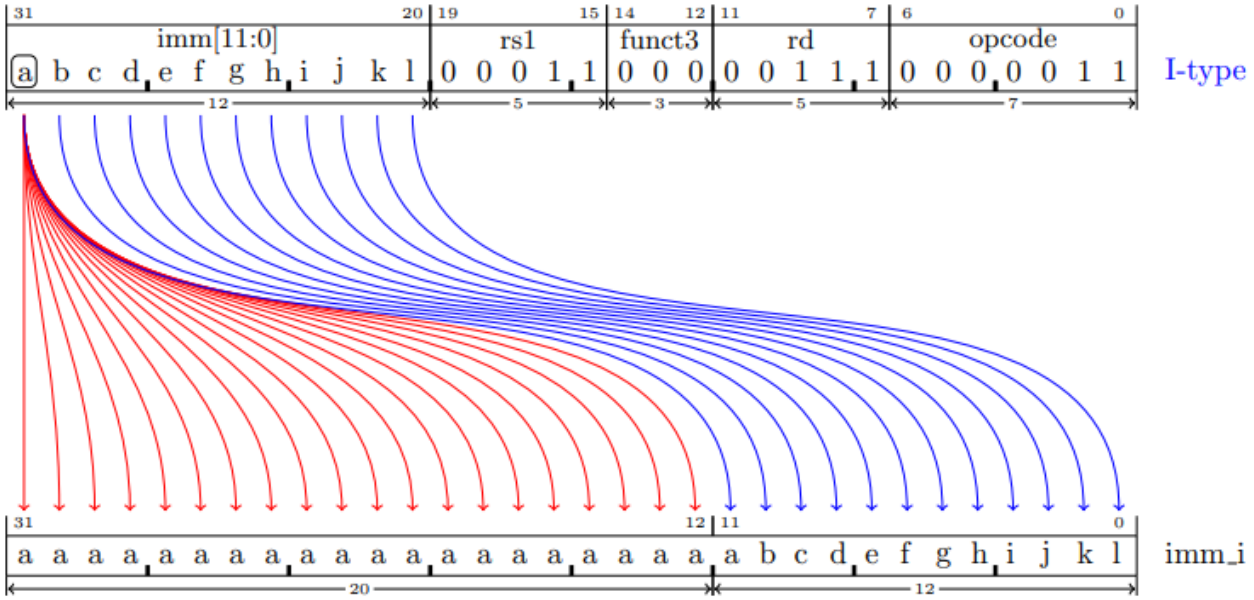

Fig 3.4: Decoding an I-type Instruction

Figure 2.3 depicts the Register-type RV32I ISA V 2.0. Similar to R-type, Opcode width is of 7 bits. Source registers (rs1) and destination register (rd) are indicated by five bit fields. The function field of 3 bits, is used for identifying the type of operation to be performed. It has a separate 12 bit field for holding the immediate operand, used for immediate data operations. The instructions which are supported by this format are jalr, lhu, lw, lb, lbu, lh, srai, srli, slli, slti, addi, andi, ori, xori and sltiu. Figure 2.4 shows the decoding logic of I type Instruction

3.1.3 S-type RV32I Instruction Format

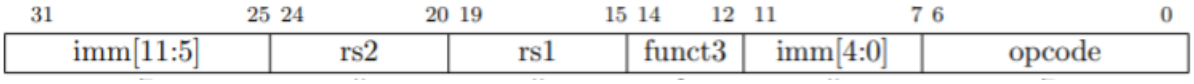

Fig 3.5: S-Type RV32I V 2.0 Instruction Format

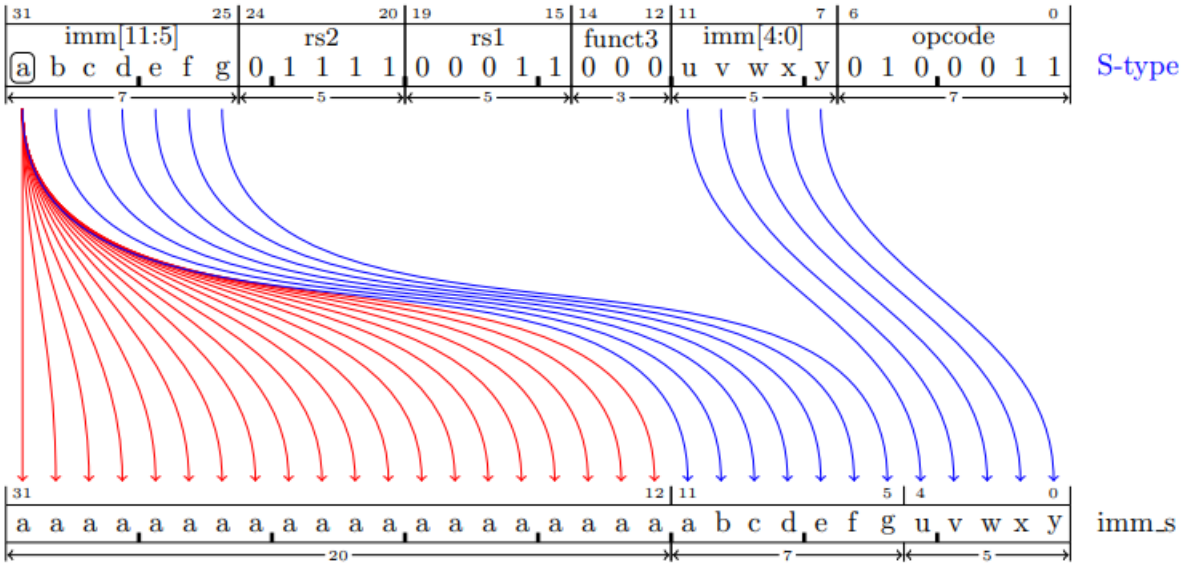

Fig: 3.6: Decoding an S-type Instruction

Figure 3.5 depicts the Store-type RV32I ISA V 2.0. Similar to R-type, Opcode width is 7 bits. Source registers (rs1 and rs2) are indicated by five bit fields. Function field is of 3 bits, which is used to indicate the size of the data need to be stored. It has a separate 7+5=12 bit field space for holding the immediate operand. This immediate operand is added with rs1 to calculate the address in which the value rs2 needed to be stored. The instructions which are supported by this format are sw, sb and sh. Figure 2.6 shows the decoding logic of s type Instruction. Figure 3.6 shows the decoding logic of S type Instruction

3.1.4 B-type RV32I Instruction Format

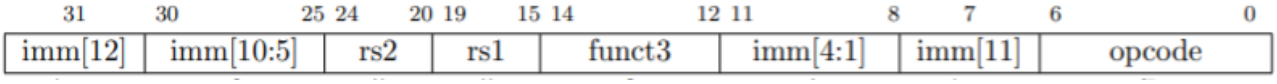

Fig 3.7: ***B-Type RV32I V 2.0 Instruction Format***

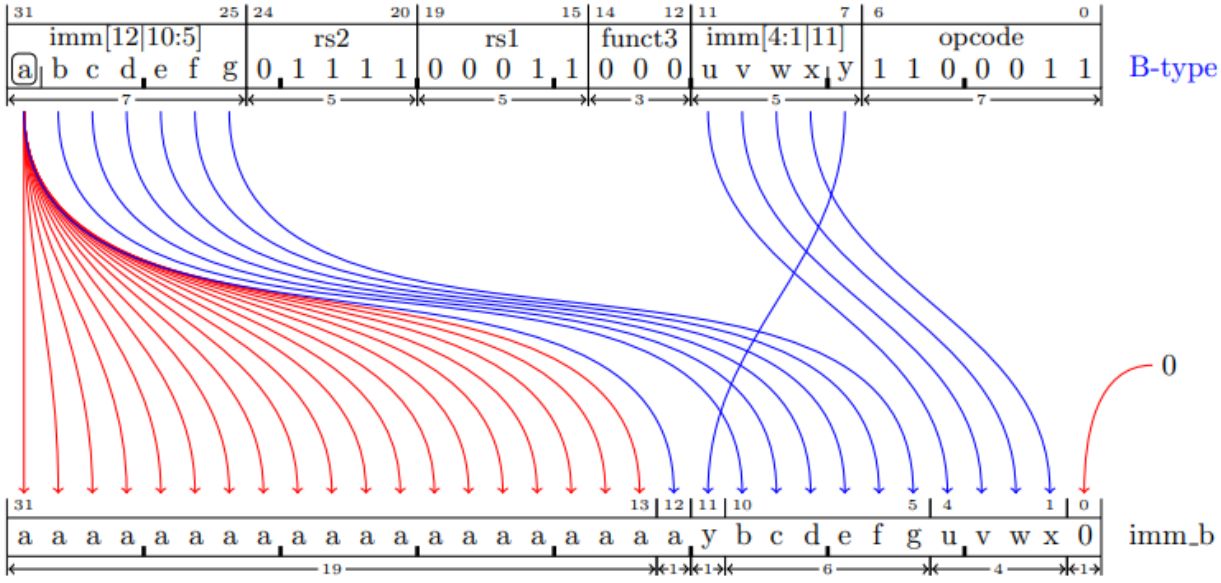

Fig 3.8: Decoding a B-type Instruction.

Figure 3.7 shows the Branch-type RV32I ISA V 2.0. Similar to other instructions, the Opcode width is of 7 bits. Source registers (rs1 and rs2) are indicated by five bit fields which are used for comparison for branching. The function field is of 3 bits, which is used to indicate the type of condition that need to be checked for branching. It has a separate 7+5=12 bit field space for holding the immediate operand, which is added to the program counter if a branch is taken. The instructions which are supported by this format are bne, bltu, blt, bgeu, bge and beq. Figure 3.8 shows the decoding logic of B type Instruction

`

3.1.5 U-type & J-type RV32I Instruction Format

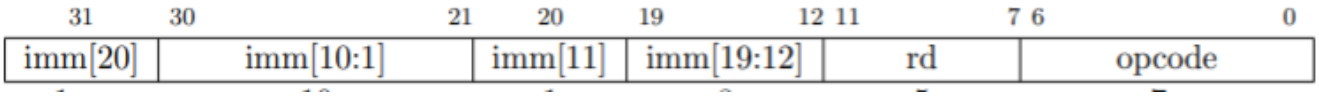

Fig 3.9: J-Type RV32I V 2.0 Instruction Format

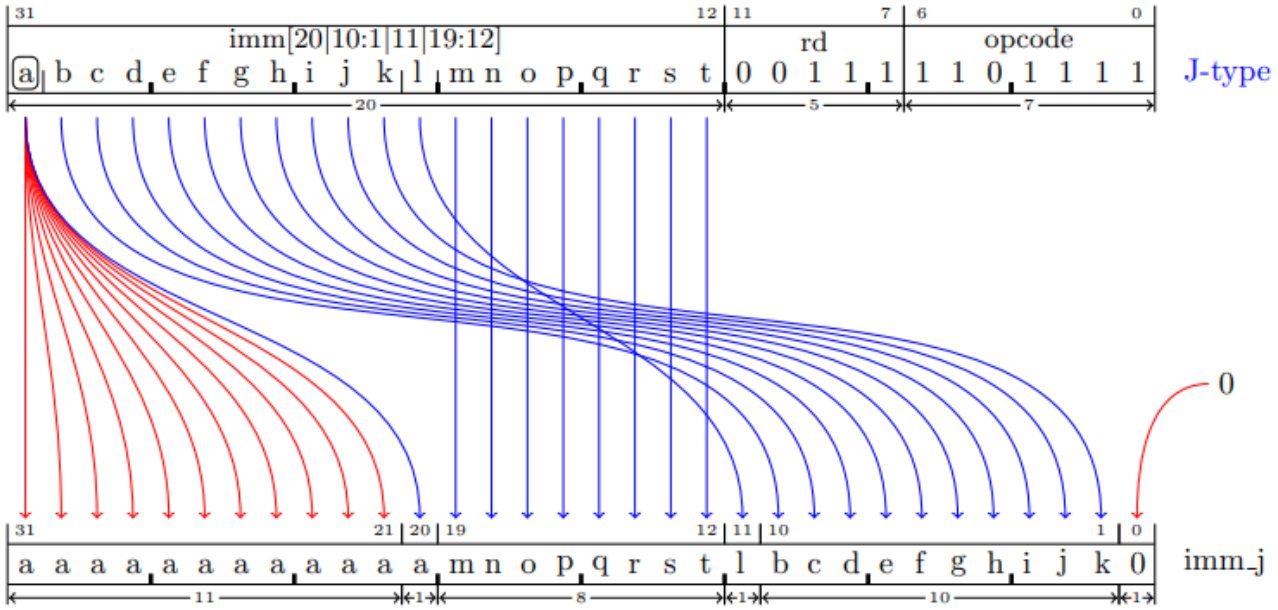

Fig 3.10: Decoding a J-type instruction.

Fig 3.11: U-Type RV32I V 2.0 Instruction Format

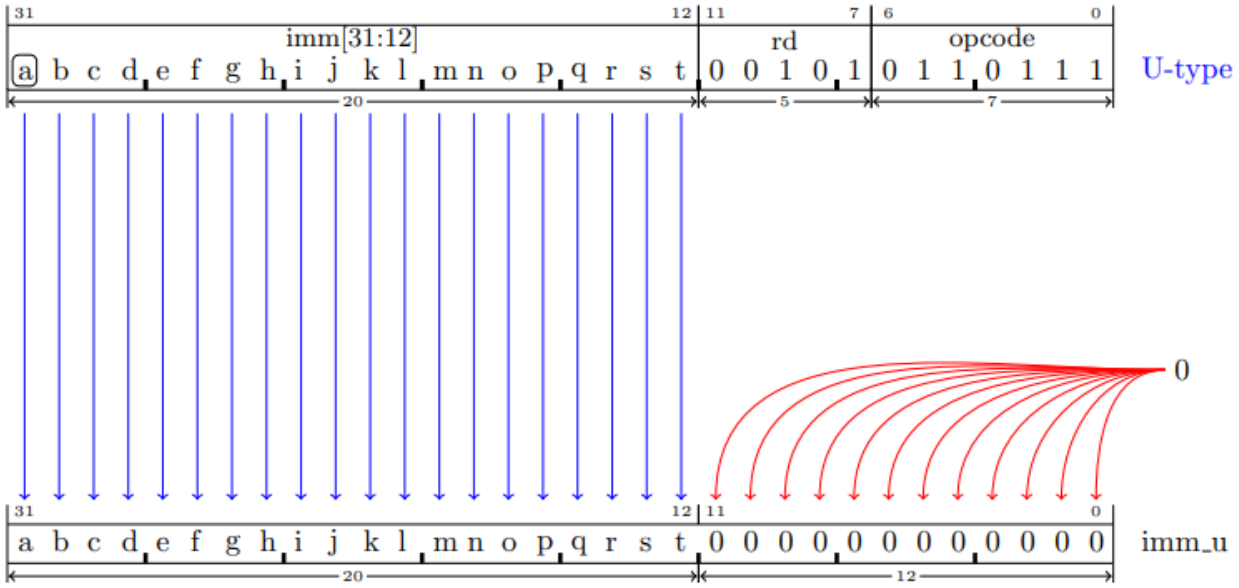

Fig 3.12: Decoding a U-type instruction

Figure 3.9 & 3.11 shows the U-type and J-type RV32I ISA V 2.0 which are similar to each other. It has a total of two fields. Opcode width is 7 bits, which is used to indicate the type of instruction format. The destination register (rd) is indicated by a five bit field. It has a 20 bit field for holding the immediate operand, used for immediate data operations. For J-type the immediate data is rearranged before branching. The instructions which are supported by this format are jal, lui and auipc. Figure 3.10 & 3.12 shows the decoding logic of J & U type Instruction.

3.2. INSTRUCTIONS SUPPORTED

The instructions supported by the RISC V RV32I processor supports a wide variety of operations. The types of operations can be mainly classified into:-

1) Arithmetic Operations
2) Logical Operations
3) Data Transfer operations
4) Control operations

Each of these operations requires the contributions of different combinations of stages to execute. Hence all stages are interdependent of each other to make a particular instruction to execute. The detailed operations of each instructions are listed in the tables below.

3.2.1 Arithmetic Operations

Instruction	Type	Example	Meaning
Add	**R**	`add rd, rs1, rs2`	`R[rd] = R[rs1] + R[rs2]`
Subtract	**R**	`sub rd, rs1, rs2`	`R[rd] = R[rs1] – R[rs2]`
Add immediate	**I**	`addi rd, rs1, imm12`	`R[rd] = R[rs1] + SignExt(imm12)`
Set less than	**R**	`slt rd, rs1, rs2`	`R[rd] = (R[rs1] < R[rs2])? 1 : 0`
Set less than immediate	**I**	`slti rd, rs1, imm12`	`R[rd] = (R[rs1] < SignExt(imm12))? 1 : 0`
Set less than unsigned	**R**	`sltu rd, rs1, rs2`	`R[rd] = (R[rs1] <u R[rs2])? 1 : 0`
Set less than immediate unsigned	**I**	`sltiu rd, rs1, imm12`	`R[rd] = (R[rs1] <u SignExt(imm12))? 1 : 0`
Load upper immediate	**U**	`lui rd, imm20`	`R[rd] = SignExt(imm20 << 12)`
Add upper immediate to PC	**U**	`auipc rd, imm20`	`R[rd] = PC + SignExt(imm20 << 12)`

Table 3.1: Arithmetic Operations

Table 3.1 shows the list of arithmetic operations supported by the processor. Arithmetic operations are carried out by the ALU in the execution stage of the processor. These operations are carried out on two source operands and the result is written back to the register file at the memory write back stage. The immediate data are sign extended to 32 bits, thus all operations are carried out with respect to 32 bits. All operations are done such that register 1 will always take the left hand side and the register 2 or the immediate data on the right hand side.

3.2.2 Logical Operations

Instruction	Type	Example	Meaning
AND	**R**	`and rd, rs1, rs2`	`R[rd] = R[rs1] & R[rs2]`
OR	**R**	`or rd, rs1, rs2`	`R[rd] = R[rs1] | R[rs2]`
XOR	**R**	`xor rd, rs1, rs2`	`R[rd] = R[rs1] ^ R[rs2]`
AND immediate	**I**	`andi rd, rs1, imm12`	`R[rd] = R[rs1] & SignExt(imm12)`
OR immediate	**I**	`ori rd, rs1, imm12`	`R[rd] = R[rs1] | SignExt(imm12)`
XOR immediate	**I**	`xori rd, rs1, imm12`	`R[rd] = R[rs1] ^ SignExt(imm12)`
Shift left logical	**R**	`sll rd, rs1, rs2`	`R[rd] = R[rs1] << R[rs2]`
Shift right logical	**R**	`srl rd, rs1, rs2`	`R[rd] = R[rs1] >> R[rs2]` *(logical)*
Shift right arithmetic	**R**	`sra rd, rs1, rs2`	`R[rd] = R[rs1] >> R[rs2]` *(arithmetic)*
Shift left logical immediate	**I**	`slli rd, rs1, shamt`	`R[rd] = R[rs1] << shamt`
Shift right logical imm.	**I**	`srli rd, rs1, shamt`	`R[rd] = R[rs1] >> shamt` *(logical)*
Shift right arithmetic immediate	**I**	`srai rd, rs1, shamt`	`R[rd] = R[rs1] >> shamt` *(arithmetic)*

Table 3.2: Logical Operations

Table 3.2 shows the list of Logical operations supported by the processor. Logical operations are carried out by the ALU in the execution stage of the processor. These operations are carried out on two source operands and the result is written back to the register file at the memory write back stage. The immediate data are sign extended to 32 bits, thus all operations are carried out with respect to 32 bits. All operations are done such that register 1 will always take the left hand side and the register 2 or the immediate data on the right hand side.

3.2.3 Data Transfer Operations

Instruction	Type	Example	Meaning
Load doubleword	I	ld rd, imm12(rs1)	R[rd] = Mem_8[R[rs1] + SignExt(imm12)]
Load word	I	lw rd, imm12(rs1)	R[rd] = SignExt(Mem_4[R[rs1] + SignExt(imm12)])
Load halfword	I	lh rd, imm12(rs1)	R[rd] = SignExt(Mem_2[R[rs1] + SignExt(imm12)])
Load byte	I	lb rd, imm12(rs1)	R[rd] = SignExt(Mem_1[R[rs1] + SignExt(imm12)])
Load word unsigned	I	lwu rd, imm12(rs1)	R[rd] = ZeroExt(Mem_4[R[rs1] + SignExt(imm12)])
Load halfword unsigned	I	lhu rd, imm12(rs1)	R[rd] = ZeroExt(Mem_2[R[rs1] + SignExt(imm12)])
Load byte unsigned	I	lbu rd, imm12(rs1)	R[rd] = ZeroExt(Mem_1[R[rs1] + SignExt(imm12)])
Store doubleword	S	sd rs2, imm12(rs1)	Mem_8[R[rs1] + SignExt(imm12)] = R[rs2]
Store word	S	sw rs2, imm12(rs1)	Mem_4[R[rs1] + SignExt(imm12)] = R[rs2](31:0)
Store halfword	S	sh rs2, imm12(rs1)	Mem_2[R[rs1] + SignExt(imm12)] = R[rs2](15:0)
Store byte	S	sb rs2, imm12(rs1)	Mem_1[R[rs1] + SignExt(imm12)] = R[rs2](7:0)

Table 3.3: Data Transfer Operations

Table 3.3 shows the list of Data transfer operations supported by the processor. Address calculation part is carried out by the ALU in the execution stage of the processor. These operations are carried out on two source operands and the result is written back to the next stages for memory access. The immediate data are sign extended to 32 bits, thus all operations are carried out with respect to 32 bits. All operations are done such that register 1 will always take the left hand side and the register 2 or the immediate data on the right hand side. Load operations are carried out on the write back stage and the store operations are carried on the memory access stage.

3.2.4 Control Transfer Instructions

Instruction	Type	Example	Meaning
Branch equal	**SB**	`beq rs1, rs2, imm12`	`if (R[rs1] == R[rs2])` `  pc = pc + SignExt(imm12 << 1)`
Branch not equal	**SB**	`bne rs1, rs2, imm12`	`if (R[rs1] != R[rs2])` `  pc = pc + SignExt(imm12 << 1)`
Branch greater than or equal	**SB**	`bge rs1, rs2, imm12`	`if (R[rs1] >= R[rs2])` `  pc = pc + SignExt(imm12 << 1)`
Branch greater than or equal unsigned	**SB**	`bgeu rs1, rs2, imm12`	`if (R[rs1] >=u R[rs2])` `  pc = pc + SignExt(imm12 << 1)`
Branch less than	**SB**	`blt rs1, rs2, imm12`	`if (R[rs1] < R[rs2])` `  pc = pc + SignExt(imm12 << 1)`
Branch less than unsigned	**SB**	`bltu rs1, rs2, imm12`	`if (R[rs1] <u R[rs2])` `  pc = pc + SignExt(imm12 << 1)`
Jump and link	**UJ**	`jal rd, imm20`	`R[rd] = PC + 4` `PC = PC + SignExt(imm20 << 1)`
Jump and link register	**I**	`jalr rd, imm12(rs1)`	`R[rd] = PC + 4` `PC = (R[rs1] + SignExt(imm12)) & (~1)`

Table 3.4: Control Transfer Operations

Table 3.4 shows the list of Control Transfer operations supported by the processor. The condition for branching is checked and carried out by the ALU in the execution stage of the processor. These operations are carried out on two source operands and the result is used for setting the taken branch flag. The immediate data are sign extended to 32 bits, thus all operations are carried out with respect to 32 bits. All operations are done such that register 1 will always take the left hand side and the register 2 or the immediate data on the right hand side. The taken branch flag does not allow the following two instructions to make changes to the memory and register file of the processor.

4. CONSTRUCTION OF THE PROCESSOR

4.1 PIPELINING OF A PROCESSOR

The process of gathering instructions from the processor through a pipeline is known as pipelining. It provides for the systematic storage and execution of instructions. Pipeline processing is another name for it.

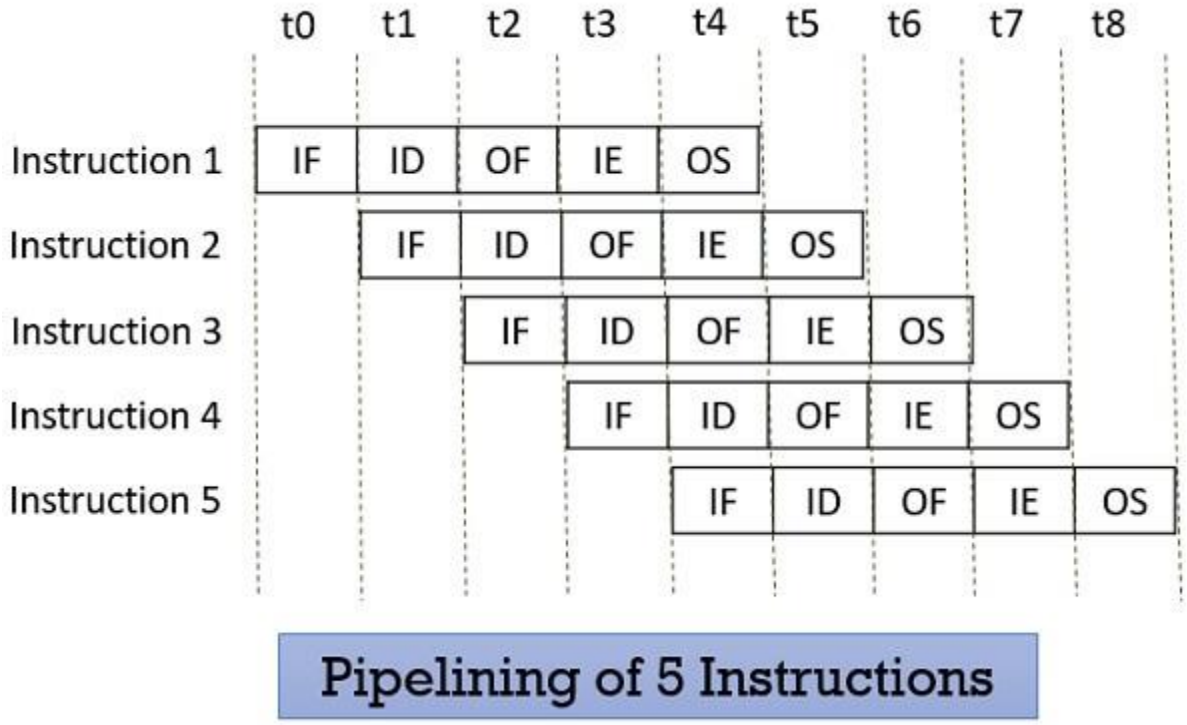

Fig 4.1: Pipelining scheduling

Pipelining is a the process of gathering instructions from the processor via a pipeline is known as pipelining. It enables the systematic storage and execution of instructions. Processor processing is another name for it. Figure 4.1 shows pipelining scheduling where 5 instructions can use the same part of the processor at the same time.

Pipelining increases the overall instruction throughput. In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment. In Figure 4.2 S1 indicated a combinational circuit and R! shows the latch.

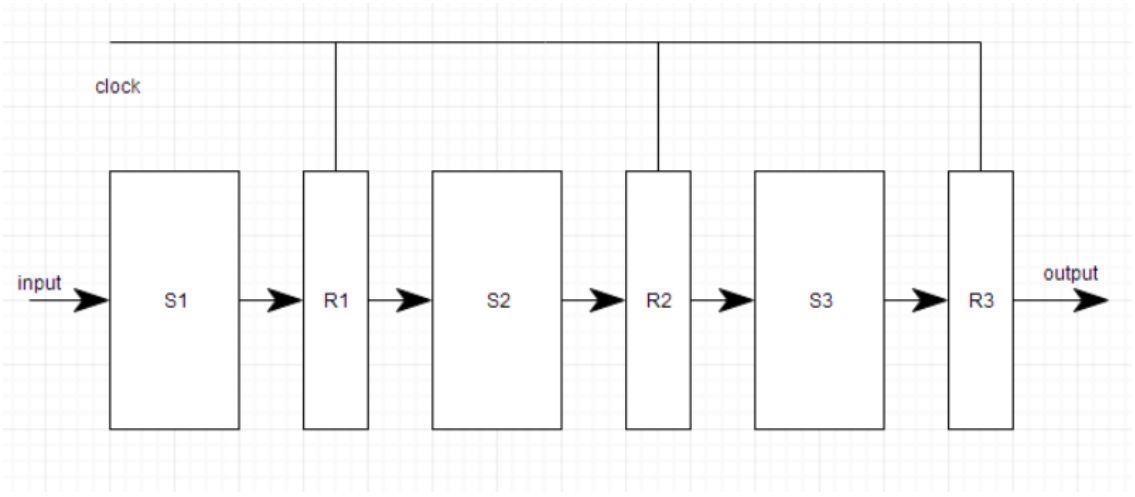

Fig 4.2: Latch configuration of a pipelined system

The process of accumulating instructions from the processor through a pipeline is known as pipelinelining. It enables for the orderly storage and execution of instructions. It's also referred to as pipeline processing**.**

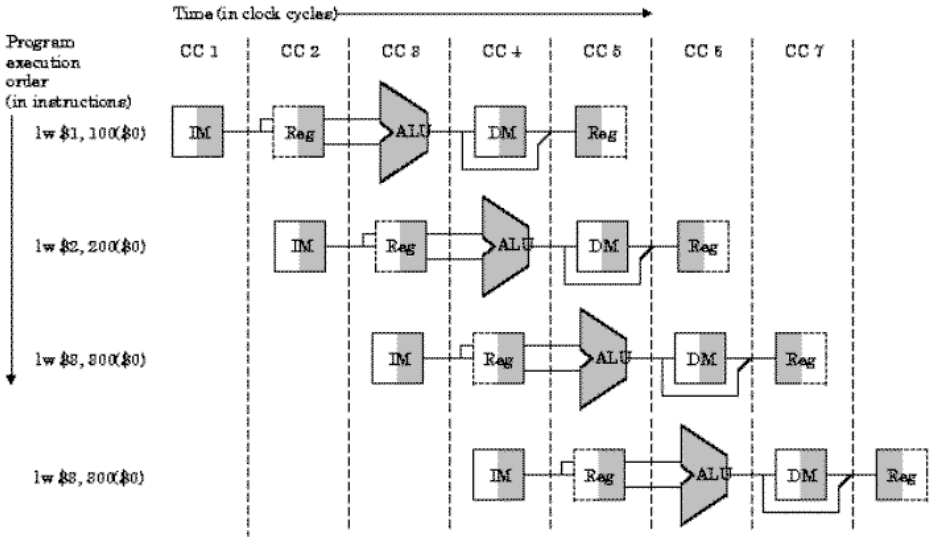

Fig 4.3: Resource utilization per clock cycle

4.2. ORGANIZATION OF THE RISC-V PROCESSOR

This RISC processor design has been constructed using five pipeline stages. The used pipeline stages are the Instruction Fetch stage (IF), Instruction Decode stage (ID), Execution stage (EX), Memory Access stage (MEM) and Write Back stage (WB). Pipeline registers or latches are used to separate the stages of the processor into 5 parts, so there is no contradictory data due to the execution of multiple instructions. They are named with the prefix as IF_ID, ID_EX, EX_MEM, MEM_WB, and WB_END. They are asserted with two different clock sources for alternate stages, the working of multiple clock domains is discussed in section IV. Other blocks include instruction memory (IR_MEM), Data memory (DATA_MEM), and General purpose registers. The working of all memory units and stages are explained here.

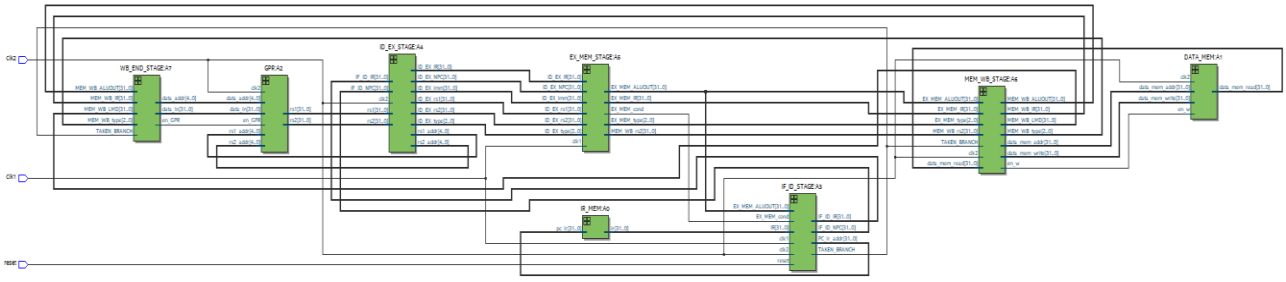

Fig 4.4: Netlist (Block diagram) of the proposed processor

4.2.1 Instruction memory

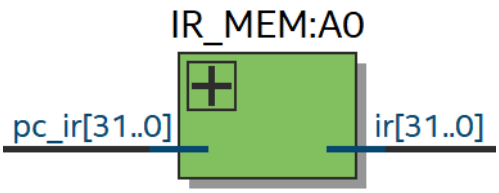

Fig 4.5: Instruction Memory Block

All instructions to be performed are stored in the ROM that acts as the instruction memory. The program counter (pc_ir) points to the location address of the next instruction to be executed as shown in Fig. The output is the 32-bit instruction, which is sent to the instruction fetch stage. Here the pc address length is of 32 bits hence it can point up to 2^32 locations. Figure 4.6 shows the Logic Diagram of Instruction Memory Block

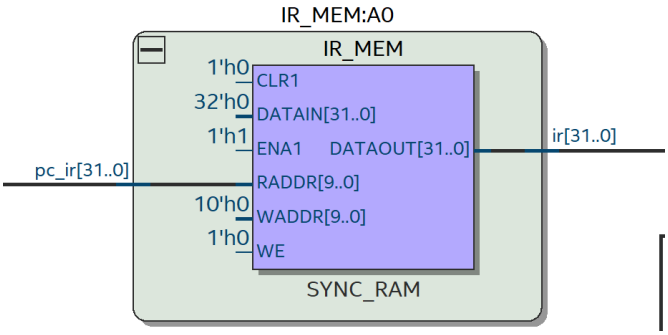

Fig 4.6: Logic Diagram of Instruction Memory Block

4.2.2 Register file

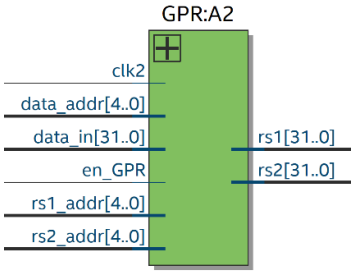

Fig 4.7: General Purpose Register file Block

This module consists of 32 registers each of 32 bit in length. The values stored in the General Purpose registers can be read simultaneously twice and written once at the same time. Data can only be written at negative edge of clock 2 if en_GPR is HIGH. The registers in this unit can be used in arithmetic and logical operations either as a source or destination [5]. The rs1_addr[4:0] is used to point to the location of source register 1 (rs1[31:0]), whereas the rs2_addr[4:0] is used to point to the location of source register 2 (rs2[31:0]). The address is of length 5 bits because there are a total of 32 registers (2^5=32). The data_in[31:0] is used to feed the input data to be written and data_addr[4:0] is used to point to the register that needs to be written. Figure 4.8 shows the Logic Diagram of Register file.

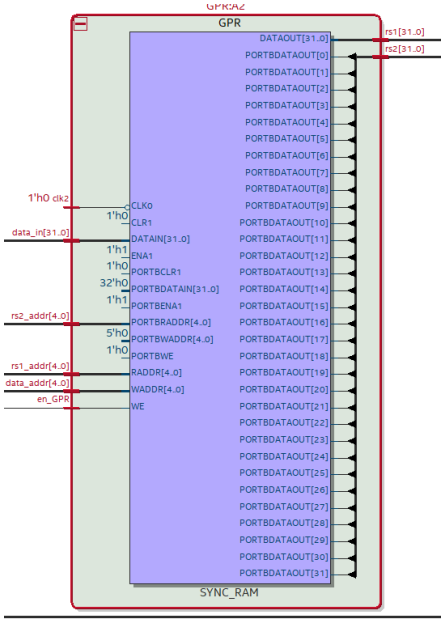

Fig 4.8: Logic Diagram of Register file

4.2.3 Data Memory

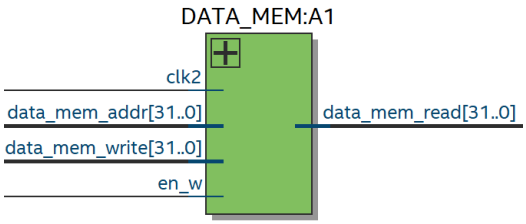

Fig 4.9: Data Memory Block

Data memory in this processor functions as RAM. This memory can only be accessed by store and load instructions. The store instruction enables the signal en_w HIGH, so at negative edge of clock 2 data can be written into the memory. Data can be read by using load instructions by setting en_w as LOW. Figure 4.10 shows the Logic Diagram of Data Memory

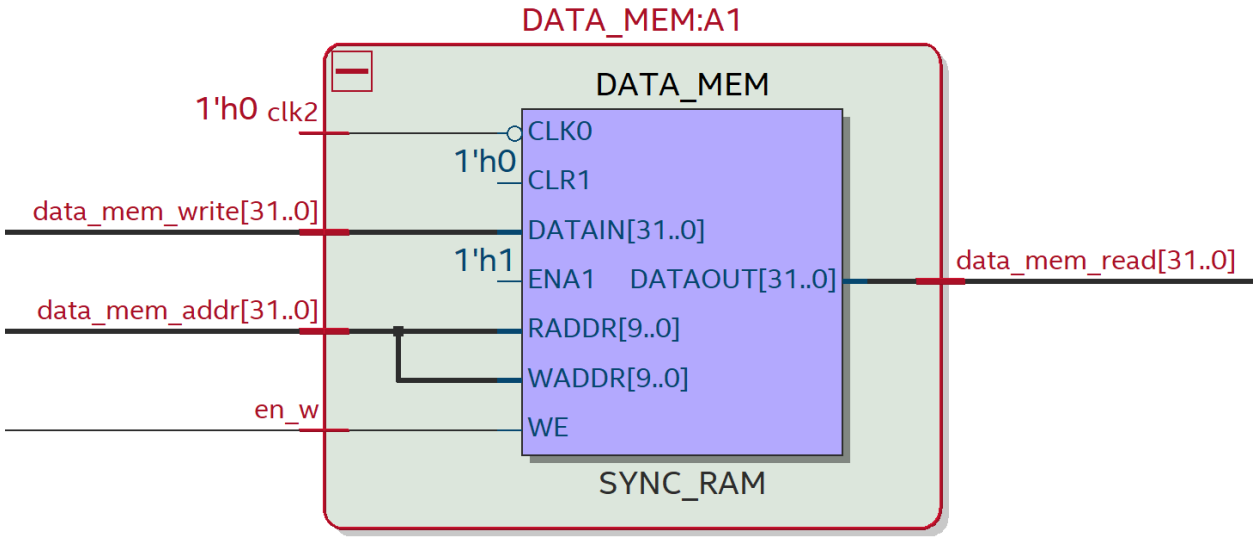

Fig 4.10: Logic Diagram of Data Memory

4.2.4 Instruction Fetch Stage

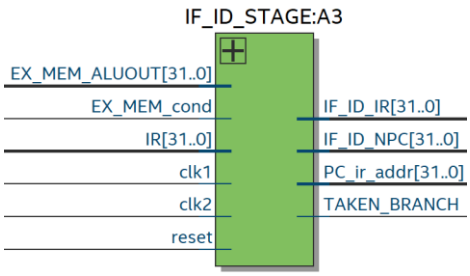

Fig 4.11: Instruction Fetch Stage Module

This Stage contains the program counter (PC) which points to the next instruction address to be executed in the instruction memory. Branch condition from execution stage is given as input to this stage, if branch taken the TAKEN_BRANCH signal is asserted HIGH at the positive edge of clock 1. This is the only stage in the processor that needs both the clocks to operate. At the positive edge of clock 1, the Program counter is incremented and if the branch is taken, the new address is written to the program counter. Clock 2 is used for resetting the status of TAKEN_BRANCH. The instruction fetched and its memory address are forwarded to the next stage. Figure 4.12 shows the Logic Diagram of Fetch Stage.

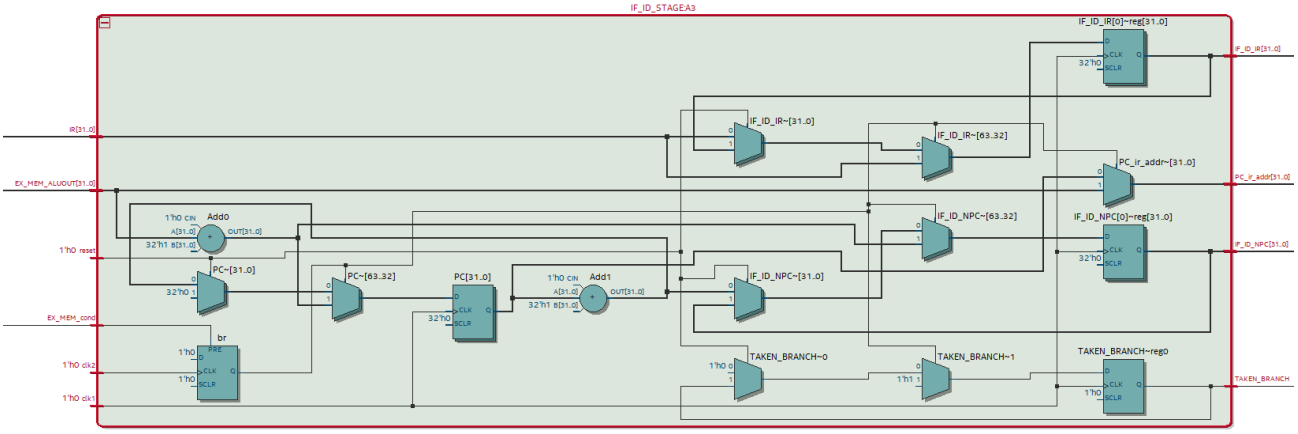

Fig 4.12: Logic Diagram of Fetch Stage

4.2.5 Decode Stage

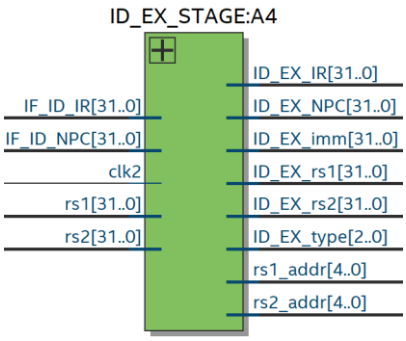

Fig 4.13: Decode Stage Module

The instruction is decoded in this stage and decoded information is forwarded at the positive edge of clock 2. Source operands from general-purpose registers are fetched. The immediate data from the instruction is rearranged according to the opcode and are sign-extended to 32 bits. The function field is decoded to find the operation needed to be performed. All the data from the decoder is forwarded to the next stage for further processing. Figure 4.14 shows the Logic Diagram of Decode Stage.

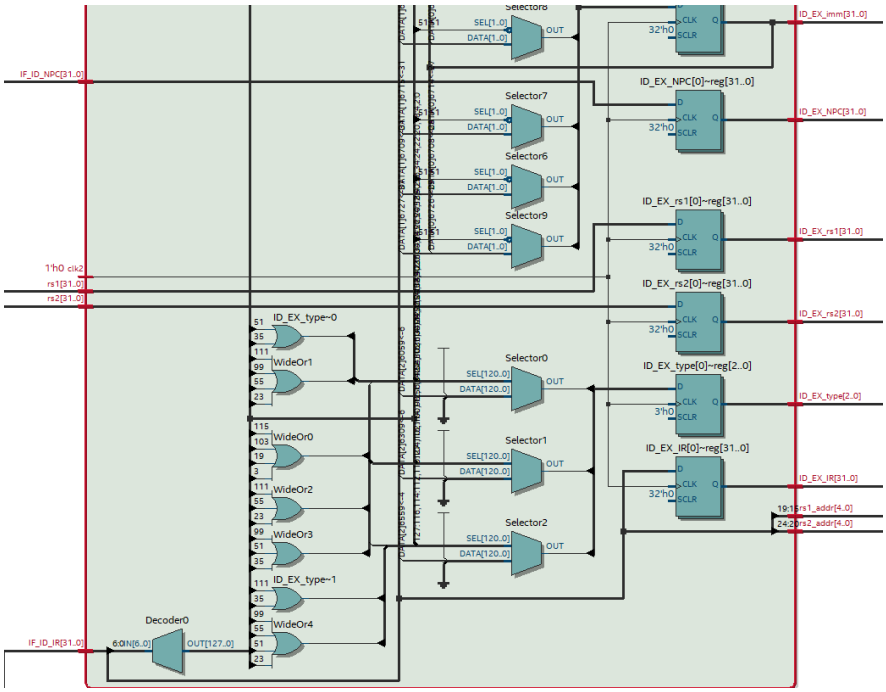

Fig 4.14: Logic Diagram of Decode Stage

4.2.6 Execute Stage

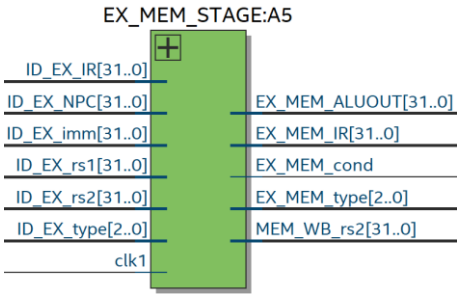

Fig 4.15: Execute Stage Module

This stage contains an ALU which is used to do all the arithmetic and logical operations. If a successful branch is taken then EX_MEM_cond is set HIGH alerting the Instruction fetch stage to update the program counter with EX_MEM_ALUOUT value. At the positive edge of clock1 the computed data is forwarded to the next stage. Figure 4.16 shows the Logic Diagram of Execute Stage.

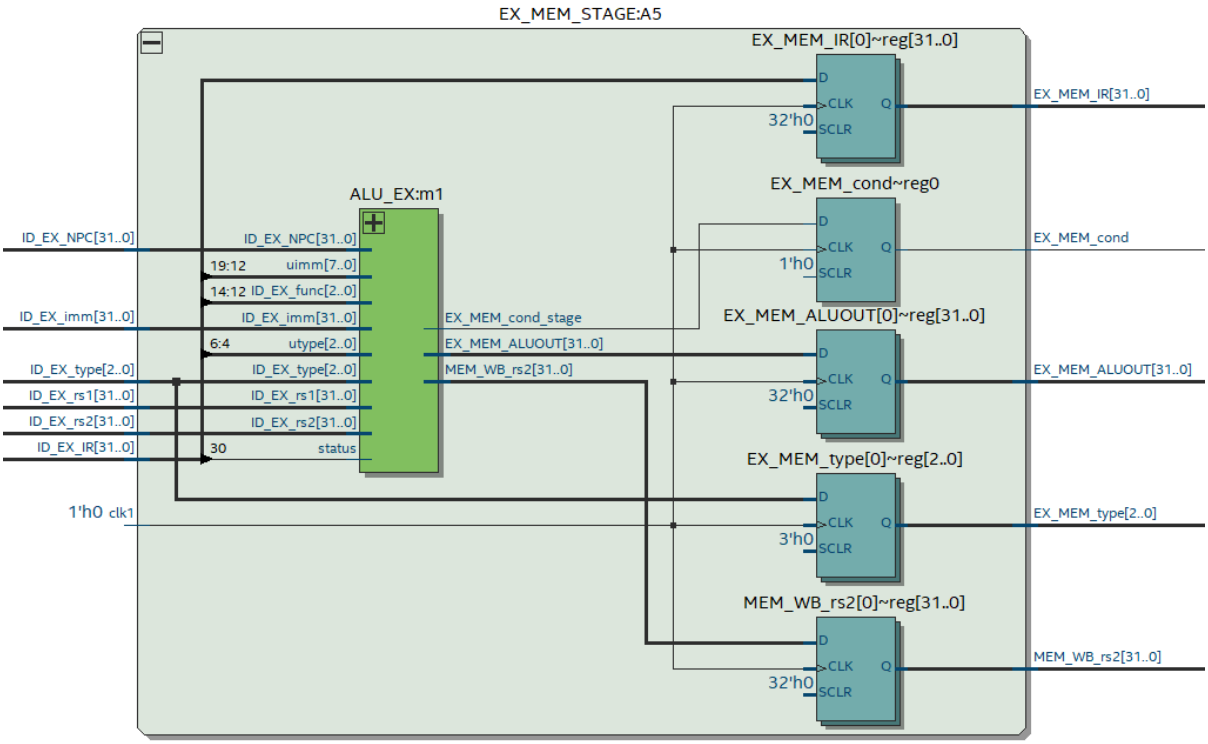

Fig 4.16: Logic Diagram of Execute Stage

4.2.7 Memory Access Stage

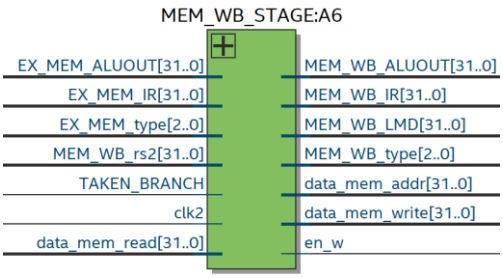

Fig 4.17: Memory Access Stage Module

Load and Store operations are performed in this stage. If the instruction is not a memory access instruction then it is ignored and en_w is set to LOW. MEM_WB_LMD is the data read from memory for Load instructions. If TAKEN_BRANCH is HIGH then all write operations are terminated, since the instruction before this has taken the branch. At the positive edge of clock 2, the output is forwarded to data memory or the next stage for write back. Figure 4.18 shows the Logic Diagram of Memory Access Stage.

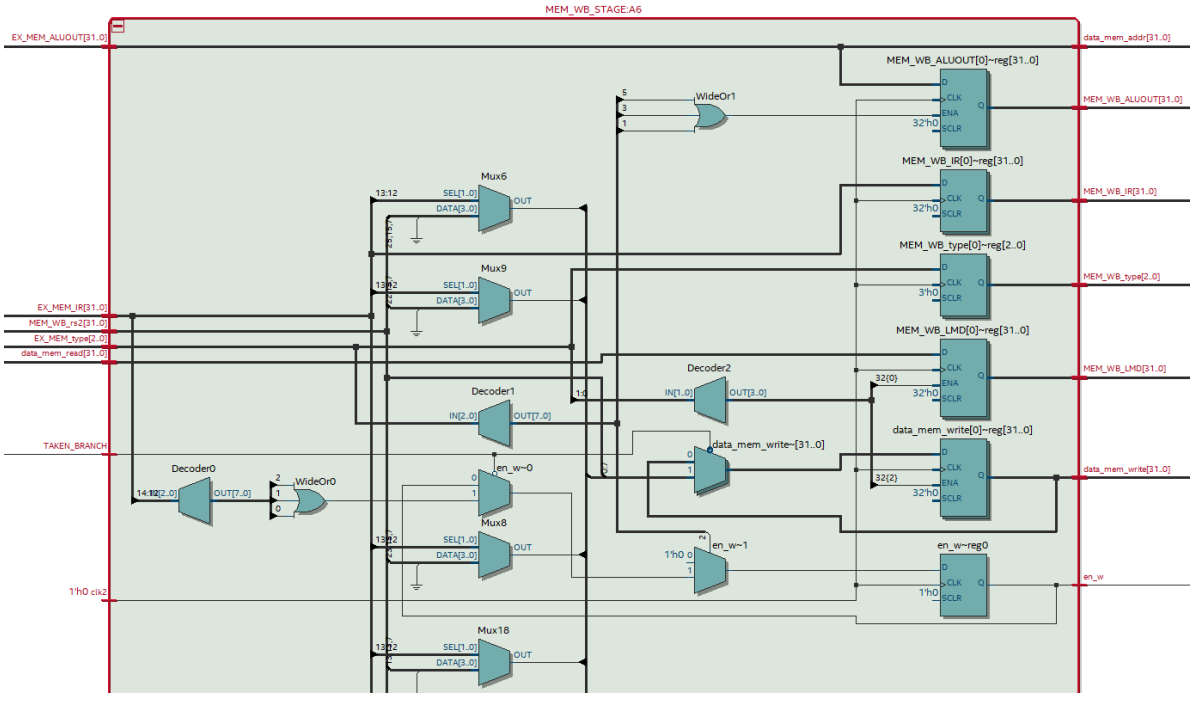

Fig 4.18: Logic Diagram of Memory Access Stage

4.2.8 Write Back Stage

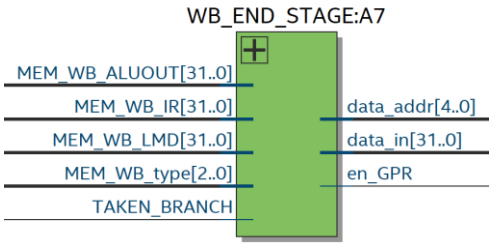

Fig 4.19: Write Back Stage Module

In this stage, the output data from the previous stage is uploaded to the address in the destination register. If TAKEN_BRANCH is HIGH then all write operations are terminated. Figure 4.20 shows the Logic Diagram of Write Back Stage.

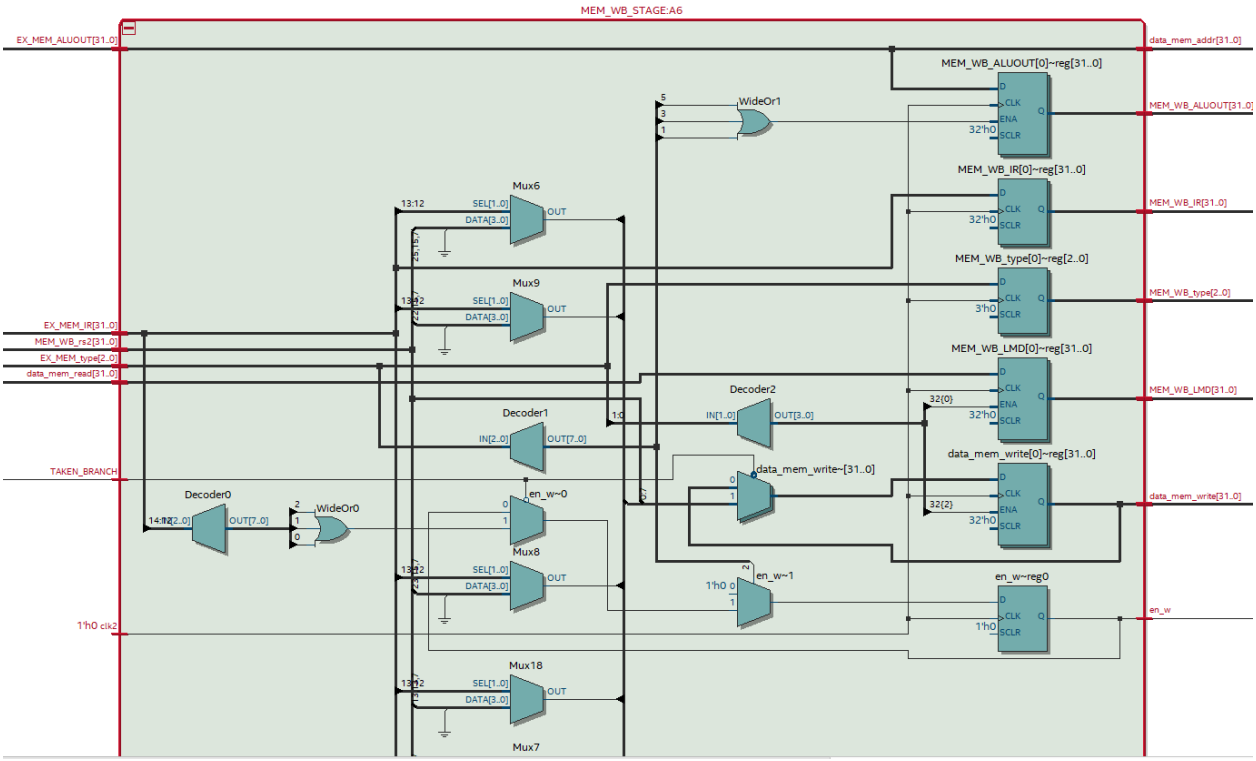

Fig 4.20: Logic Diagram of Write Back Stage

4.3 ELIMINATION OF PIPELINE HAZARDS

There are possibly three types of hazards that arise in a pipelined processor:-

1) Structural Hazards

2) Data Hazards

3) Control Hazards

4.3.1 STRUCTURAL HAZARDS

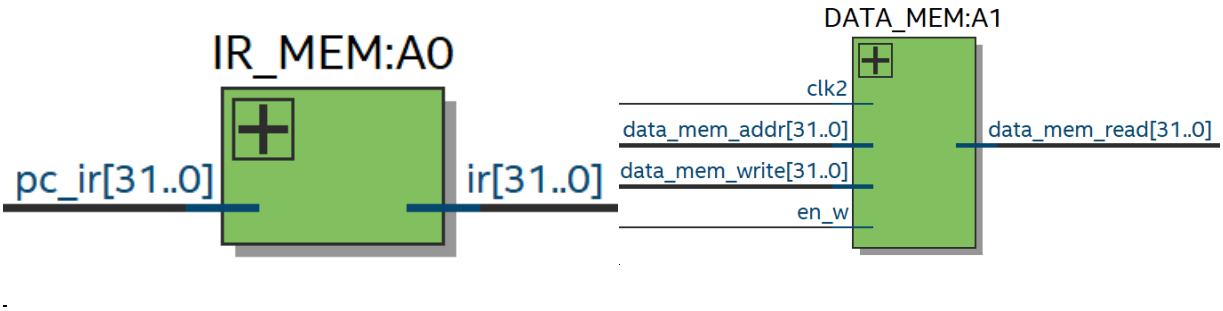

Fig 4.21: Separate IR memory and date memory

The process of gathering instructions from the processor through a pipeline is known as pipelining. It provides for the systematic storage and execution of instructions. Pipeline processing is another name for it.

Example:-

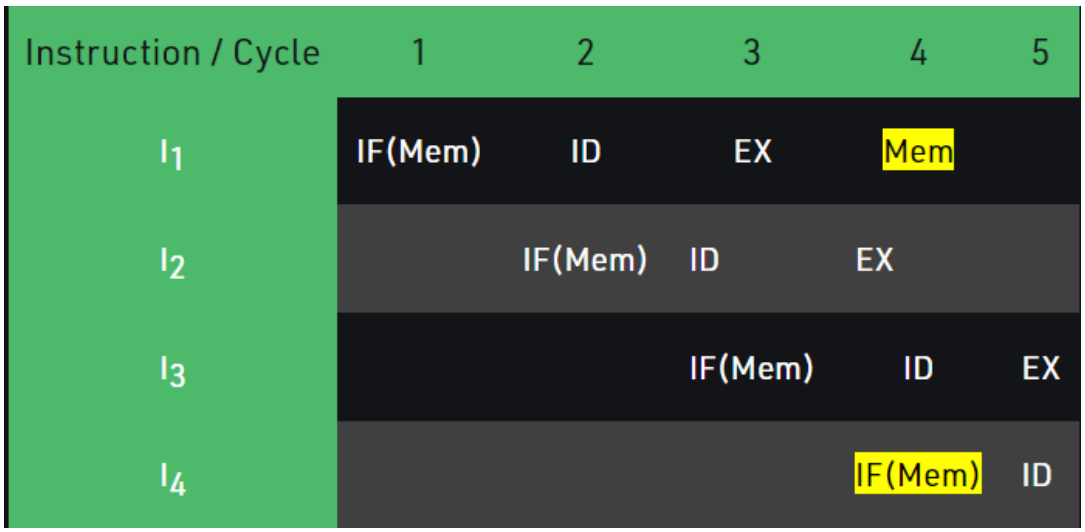

Fig 4.22. Example of Structural Hazard

In the above scenario in Figure 4.22, in cycle 4, instructions I1 and I4 are trying to access same resource (Memory) which introduces a resource conflict.

To avoid this problem, we have to keep the instruction on wait until the required resource (memory in our case) becomes available

Execution of instructions in parallel will also introduce the risk of pipeline hazards during the execution of multiple instructions. Structural hazards are encountered when multiple instructions use a common resource at the same time.

This has been eliminated by implementing the processor using Harvard architecture with separate data and instruction memory, so instruction can be fetched and data can be accessed at the same time.

Also the general-purpose memory with two read ports and one write port. Hence both the decode stage and write back stage can access the register file at the same time. Therefore, several data accesses can be performed simultaneously without conflict.

4.3.2 DATA HAZARDS

Data hazards can be classified in to three types:-

A) Read After Write (RAW) Hazard [Flow/True data dependency].
B) Write After Read (WAR) Hazard [Anti-Data dependency] .
C) Write After Write (RAW) Hazard [Output data dependency] .

Let there be two instructions I and J, such that J follow I. Then,

RAW hazard occurs when instruction J tries to read data before instruction I writes it.
Eg:
I: R2 <- R1 + R3
J: R4 <- R2 + R3
WAR hazard occurs when instruction J tries to write data before instruction I reads it.
Eg:
I: R2 <- R1 + R3
J: R3 <- R4 + R5
WAW hazard occurs when instruction J tries to write output before instruction I writes it.
Eg:
I: R2 <- R1 + R3
J: R2 <- R4 + R5
WAR and WAW hazards occur during the out-of-order execution of the instructions.

In RISC V RAW hazards are only possible, Due to the organization of stages. Since write back stage is after read and instructions are executed in order.

RAW Data hazards are encountered due to the usage of common When the source for one instruction is also the destination for the previous instruction, this is known as source and destination resources in successive instructions.

For example in the figure ,R3 must be written before reading from it. But due the organization of the stages the written is taken in the last stage which causes the error. Data hazards can be prevented by inserting dummy instructions during compiling using the compiler to create a gap between those instructions.

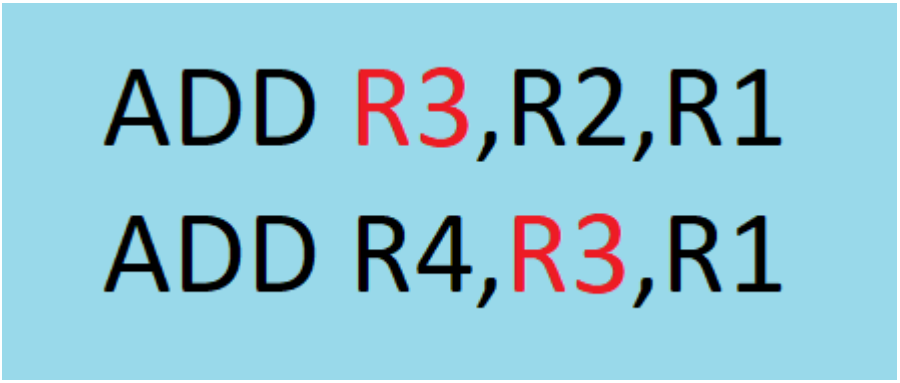

Fig 4.23: Example for RAW Hazard

4.3.3 CONTROL HAZARDS

This form of dependency happens when control instructions such as BRANCH, CALL, JMP, and others are transferred. When the processor wants to introduce a new instruction into the pipeline, it will not know the target address of these instructions on many instruction architectures. Unwanted instructions are fed into the pipeline as a result of this.Consider the following sequence of instructions in the program:

100: I1

101: I2 (JMP 250)

102: I3

.

250: BI1

Expected output: I1 -> I2 -> BI1

NOTE: Generally, the target address of the JMP instruction is known after ID stage only.

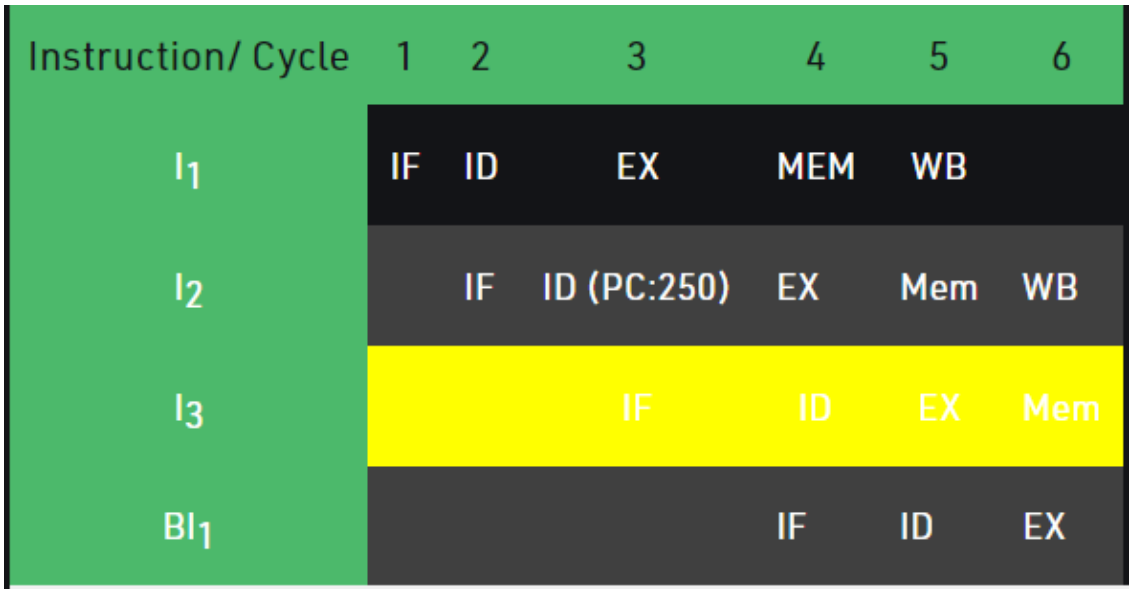

Fig 4.24: Example for RAW Hazard

Output Sequence: I1 -> I2 -> I3 -> BI1

So, the output sequence is not equal to the expected output, that means the pipeline is not implemented correctly as shown in Figure 4.24.

These hazards are encountered during the successful execution of branch and jump instructions. This can be prevented by a branch flag which goes HIGH when the branch is taken, thus following instructions following after the branch in memory and write backstage are terminated.

For example in figure 4.25, when the branch enters the pipeline the result of whether to take branch or not will be known at the execution stage only. Within that period of time the next two instruction would have already entered the pipeline. If the branch condition is satisfied the next two instruction execution must not take place. Hence with the help of the taken branch flag the next following two instructions are terminated.

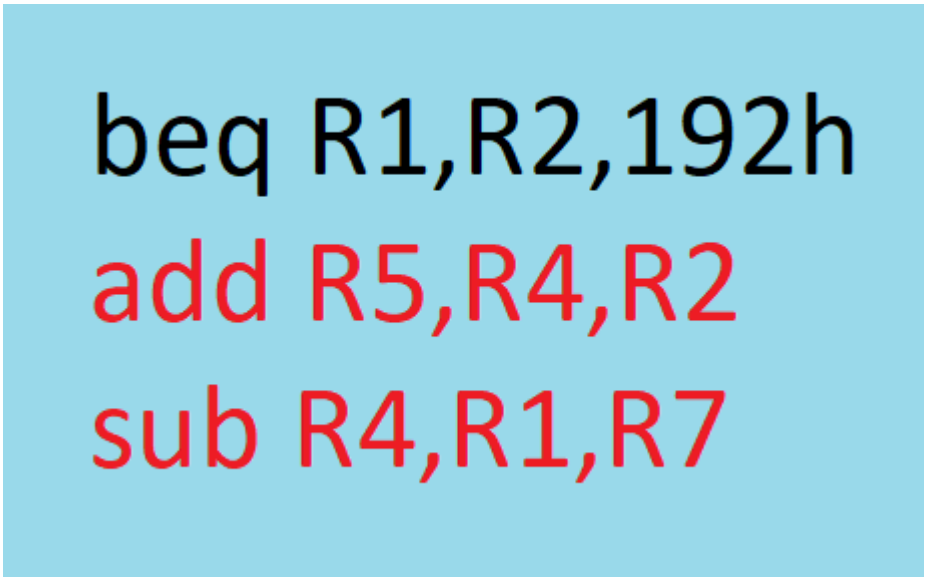

Fig 4.25: Example for Control Hazard

4.4. CONSTRUCTION OF MULTIPLE CLOCK DOMAINS

4.4.1 CAUSES OF METASTABLE CONDITIONS

The outputs of registers (or clocked flip-flops in old money) in digital circuits are subject to metastability, which refers to the possibility of an output terminal entering a'metastable condition.' D-type flip-flops are commonly used in FPGA devices. Before we look at how such a state might be entered, it's a good idea to refresh our memories on some of the most important timing factors in register operation:

'Set-up time' – The outputs of registers (or clocked flip-flops in old money) in digital circuits are concerned with metastability, which refers to the possibility of a'metastable state' for an output terminal.

'Hold time' – The outputs of registers (or clocked flip-flops in old money) in digital circuits are subject to metastability, which refers to the possibility of an output terminal entering a'metastable condition.'

'Clock-to-Output Delay time' – This is the amount of time that passes after the clock edge before the register's output changes. This is also known as the'settling time' or 'propagation delay' of the register.

There is the chance of encountering metastability whenever a signal travels between two asynchronous clock domains — digital sub-circuits within the overall design that are running on distinct, or unrelated clocks. Data transfer from an unclocked part of a design into a synchronous system – for example, external (outside) signals fed into an FPGA – is also true.

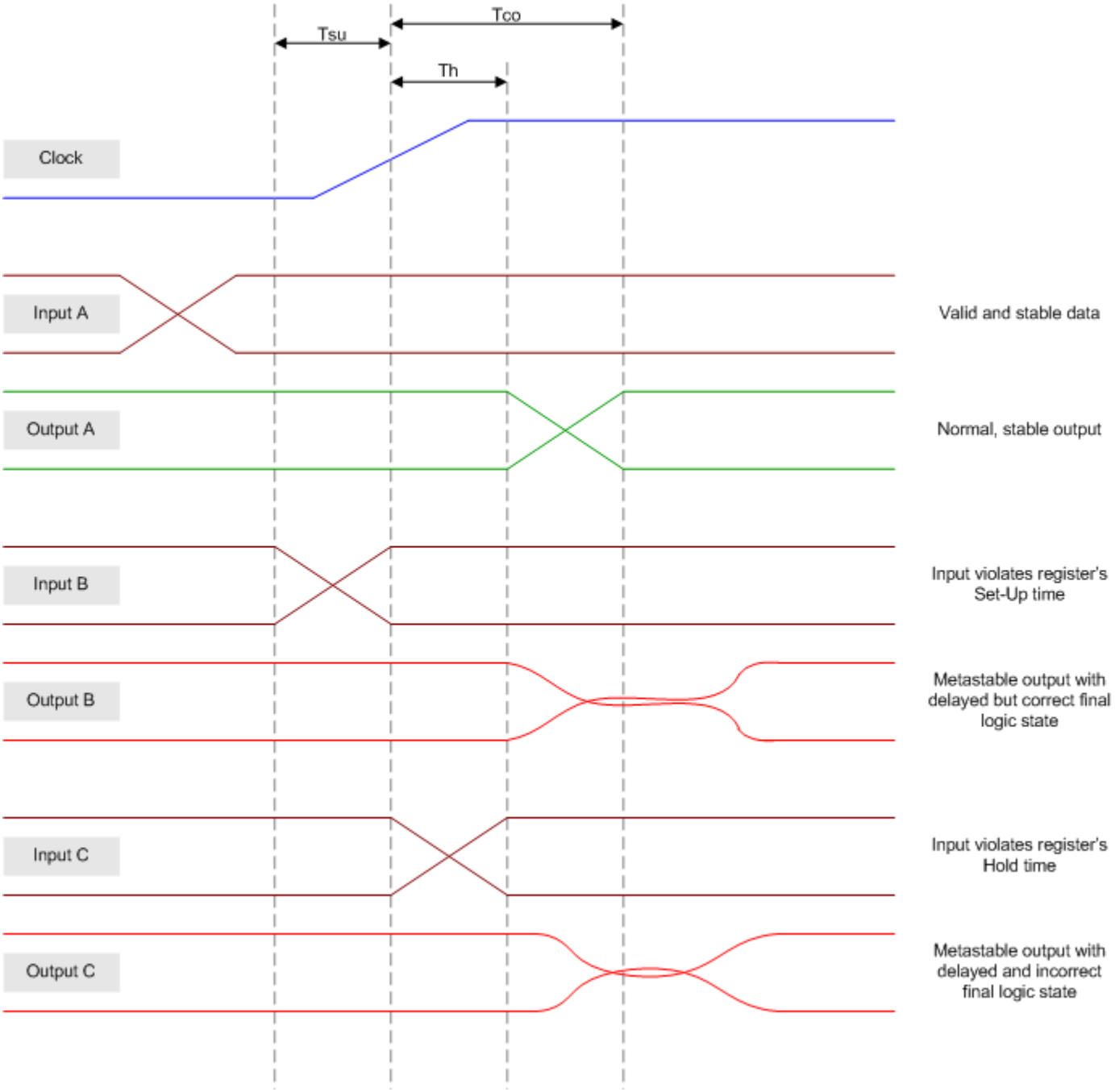

Fig 4.26: Outputs of metastable conditions

Figure 4.26 shows the various types of output results in case of violation of set up and hold time.

Input A: The output is available after the device's Clock-to-Output Delay time, and the input observes the register's Set-up and Hold durations.

Input B: During the Set-up time of the register, the input transitions, and the output becomes metastable until it settles to the correct stable level beyond the Clock-to-Output Delay time.

Input C: During the Hold period of the register, the input transitions, and the output becomes metastable. Not only does the output stabilise after the Clock-to-Output Delay time, but it also stabilises at the incorrect logic level!

If the register's output feeds into more than one subsequent register in the circuit in parallel, these destination registers may capture the data at different logic levels, depending on whether the source register's metastable output has settled to a stable state before each destination register is clocked over to capture the next data. The problem is exacerbated by path delays between the source and destination registers, which are added to the time it takes for the metastable output to become stable.

4.4.2 PREVENTION OF METASTABLE CONDITIONS

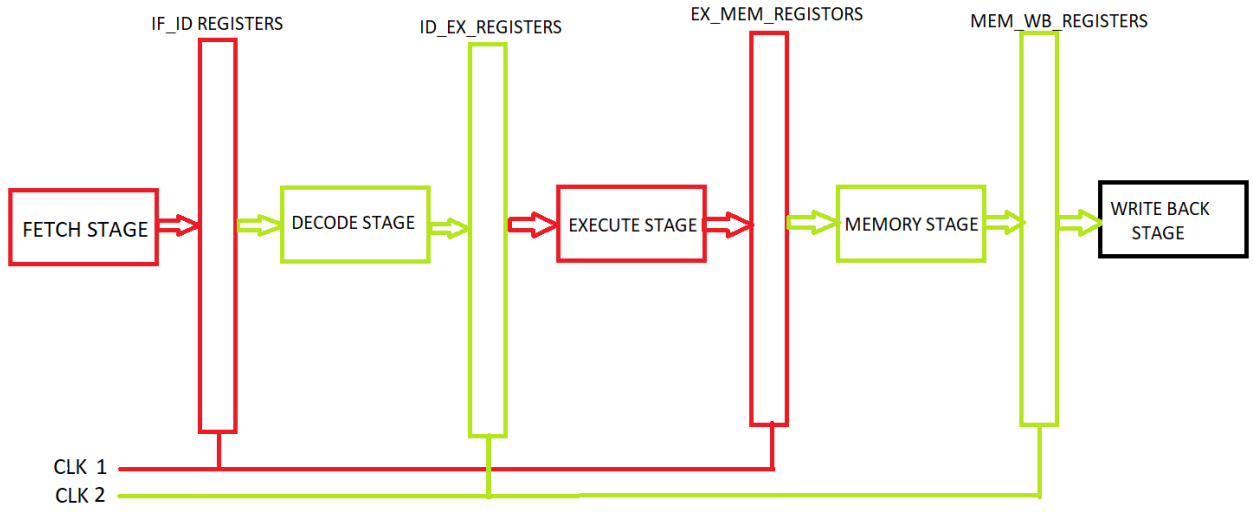

Fig 4.24: Clock distribution to the stages

Two Clock sources (clk1, clk2) are used for consecutive stages as shown in Figure 4.24. Both clock sources are non-Overlapping as shown in Figure 4.25.

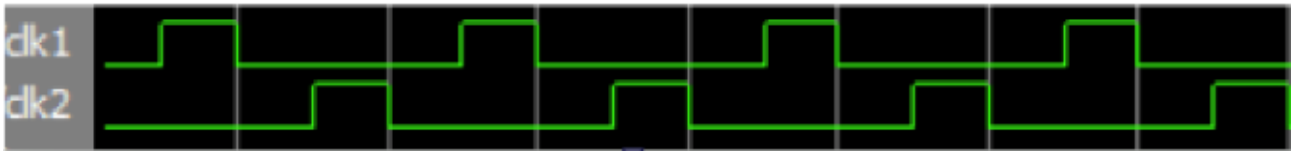

Fig 4.25: Non–Overlapping clock sources

They are non-overlapped to take care of variable delays like clocks skew, there is also a gap in between them where both clocks are LOW. This is a very safe kind of clocking scheme where we have non-overlapping clocks with a safe margin in between.

For example: since there is no overlap, at the positive edge clock 1, it is guaranteed that EX_MEM is active and the previous stage ID_EX is inactive. Hence the inputs to EX_MEM are held constant so setup & hold time violation does not occur, thus it prevents unstable output or metastable output. Thus by using two-phase clocks isolation of one stage from the other can be achieved.

5. SIMULATION AND SYNTHESIS RESULTS

5.1 SOFTWARES USED

There are mainly two softwares used in this project :-

1) ModelSim
2) Quartus Prime

5.1.1 ModelSim

Mentor's ModelSim is the best HDL language simulation software on the market. It is the industry's first single-core simulator that supports VHDL and Verilog mixed simulation and can provide a nice simulation environment. It makes use of single-core simulation technology, Tcl/Tk technology, and directly optimised compilation technology.

The compilation simulation is quick, and the produced code is platform agnostic. It's simple to secure the IP core, create a personalised graphical and user interface, and speed up debugging for users. Is the first choice of simulation software for FPGA/ASIC design because it provides a powerful means.

Features:-

- RTL and gate-level optimization, local compilation structure, fast compilation simulation speed, cross-platform and cross-version simulation;
- Single core VHDL and Verilog mixed simulation;
- Source code templates and assistants, project management;
- Integrated performance analysis, waveform comparison, code coverage, and data flow ChaseX, Signal Spy, Virtual Object, Memory window, Assertion window, source window display signal value, signal condition breakpoint, and many other debugging functions;C and Tcl/Tk interface, C debugging;
- Direct support for SystemC, arbitrarily mixed with HDL;
- Support the design function of SystemVerilog;

- The most comprehensive support for system-level description languages, SystemVerilog, SystemC, PSL;
- ASIC Sign off.
- Behavioral, RTL level, and gate-level codes can be performed individually or simultaneously.

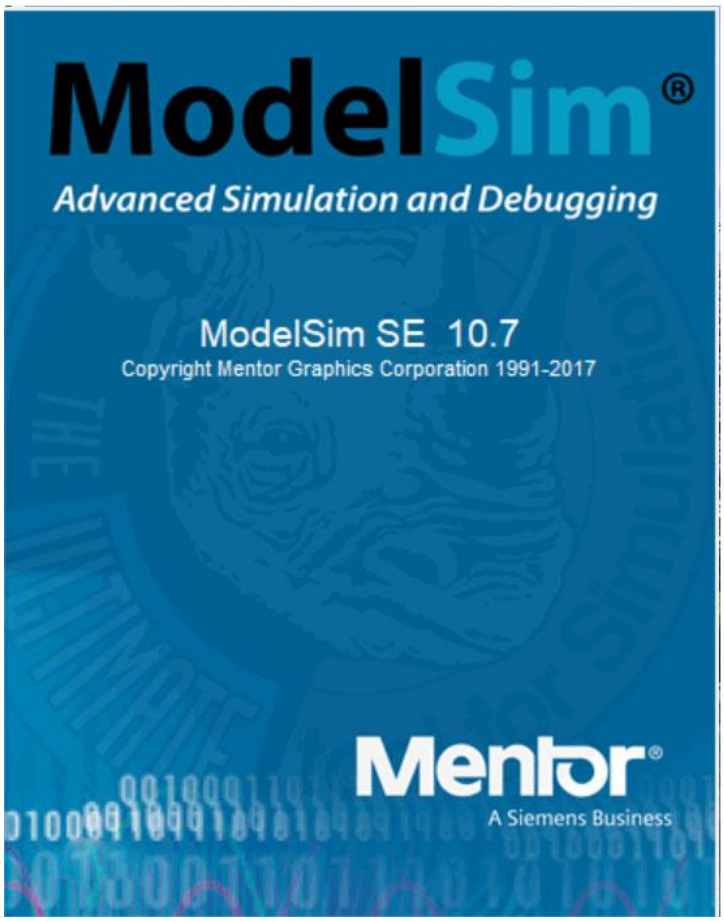

Fig 5.1: ModelSim Logo

5.1.2 Quartus Prime

In the system-on-a-programmable-chip (SOPC) design environment, the Quartus II design is the most advanced and complex. Quartus II design offers a complete temporal closure as well as a LogicLockTM block-based design flow. Quartus II design is the only software that provides fundamental capabilities such as a programmable logic device (PLD) with timing closure and a block-based design flow. The Quartus II design software boosts performance, adds functionality, and eliminates design delays. It is the first in the industry to provide a unified process for the creation of FPGA and mask-programmed devices.

Fig 5.2: QuartusLogo

Due to its extensive design capabilities and simple and easy-to-use interface, Altera Quartus II is becoming increasingly popular among digital system designers as a programmable logic design environment. The most recent version is v17.0, which is now available for official download.

Altera Quartus II (3.0 and higher) is the industry's only design tool that combines FPGAs and fixed-function HardCopy devices into a single design flow. Engineers may design HardCopy Stratix devices for mass production using the same low-cost tools they use for functional verification and prototyping of Stratix FPGAs.

System designers may now assess the performance and power consumption of HardCopy Stratix devices using the Quartus II software, and design the maximum throughput accordingly.

The fourth-generation PLD development platform is Altera's Quartus II programmable logic software. The platform accommodates design needs in a workgroup setting, including Internet-based collaborative design. Cadence, ExemplarLogic, MentorGraphics, Synopsys, and Synplicity's development tools are all compatible with the Quartus platform.

The LogicLock module design function has been enhanced, FastFit compilation options have been added, network editing performance has been enhanced, and debugging features have been enhanced.

I. The software is smaller and faster

The installation software for QuartusII2.0 is 290M, while the entire installation is 700M. If you tailor the installation and don't use the Excalibur embedded processor, the space required for installation is 460M, which is more than half as much as the QuartusII1.1 version.

But it can still handle all ALTERA chips. Development. At the same time, the software is substantially faster than version 1.1 in terms of loading, compilation, and simulation.

II. LogicLock design process improves performance by 15%

By upgrading the hierarchical LogicLock module-level architecture, QuartusII2.0 design software boosts performance by an average of 15%. The LogicLock design method allows the designer to manage the placement of the entire module, and an auxiliary layout can be employed if necessary. During the building of big SOPC systems, the LogicLock design method allows designers to optimise and lock the performance of each module independently while retaining the overall system performance.

In future Altera devices, the new LogicLock design flow algorithm is integrated into the Quartus II design software version 2.0. This algorithm makes use of module-level design to its greatest potential.

III. Reduce compilation time with quick adaptation options

QuartusII2.0 adds a new quick adaptation compilation option. Selecting this option will shorten the compilation time by 50% compared to the default setting. The quick adaptation function retains the best performance settings and speeds up the compilation process.

In this way, the layout adaptation algorithm has fewer iterations, faster compilation speed, and minimal impact on design performance.

IV. New features reduce system-level verification

Version 2.0 of the Quartus II design software adds additional features to help speed up the verification step of the SOPC design process, which is normally the most time-consuming. The new SignalProbe technology allows users to route internal nodes to unused pins for analysis during the initial compilation period, while keeping the design's original wiring, time limit, and design files.

The SignalProbe technology extends the capabilities of the SignalTap embedded logic analysis. Designers can also use the HDL test templates included in the new version to create HDL simulation vectors fast.

From the Quartus II simulator waveform file, version 2.0 of the Quartus II design programme can automatically produce a complete HDL test platform.

The Quartus II design programme now supports high-speed I/O design in version 2.0.

5.2. SIMULATION RESULTS

The proposed microprocessor design is developed using Verilog and simulated using ModelSim, The design is simulated by providing loaded instructions in the instruction memory.

The following examples of instruction types are shown as examples with respect to pipelining in the upcoming sessions

1 ARITHMETIC TYPE INSTRUCTIONS
2 LOGICAL TYPE INSTRUCTIONS
3 IMMEDIATE TYPE INSTRUCTIONS
4 STORE TYPE INSTRUCTIONS
5 LOAD TYPE INSTRUCTIONS
6 BRANCH TYPE INSTRUCTIONS
7 JUMP TYPE INSTRUCTIONS

During the following examples assume that in the following simulations the general purpose registers R4,R6,R7 are preloaded with hexa values of 3,10,f.

5.2.1 ARITHMETIC TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

add r1,r4,r6

sub r2,r6,r4

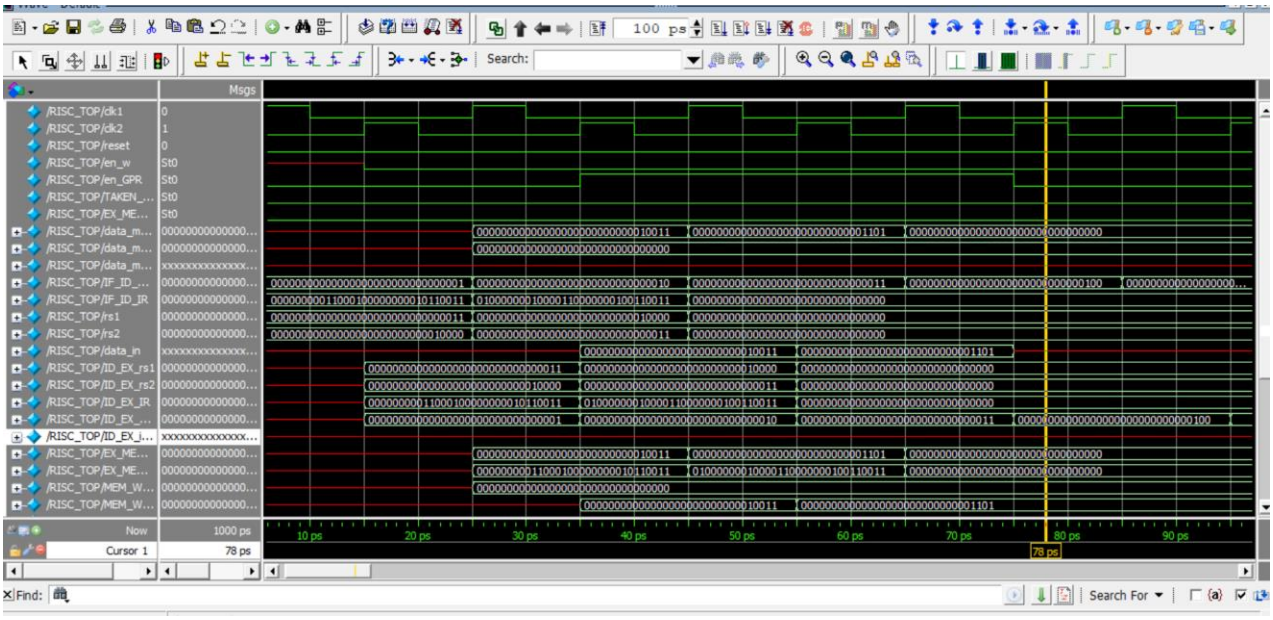

Fig 5.3: Simulation Example of ARITHMETIC TYPE INSTRUCTIONS

First the contents of r4 and r6 is fetched and added, then stored into r1. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r6 and r4 values and subtracted, then stored into r2.

```
GPR[1]=00000013,

GPR[2]=0000000d,
```

Fig 5.4: GPR value after execution of above instructions

5.2.2 LOGICAL TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

or r1,r4,r6

and r2,r6,r4

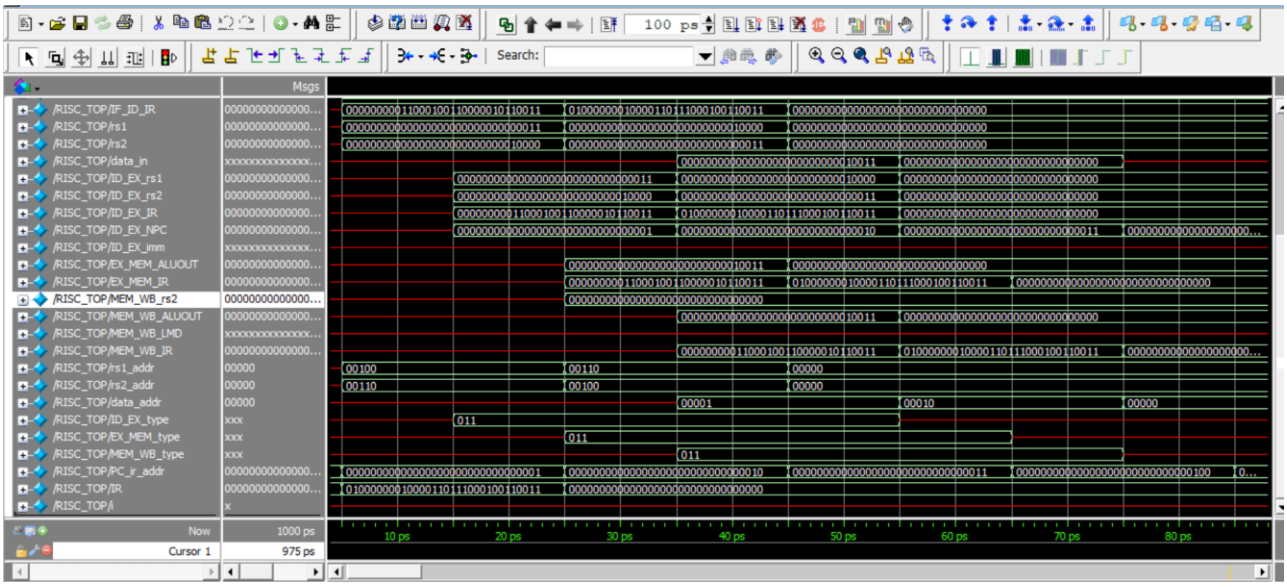

Fig 5.5: Simulation Example of LOGICAL TYPE INSTRUCTIONS

First the contents of r4 and r6 is fetched and computed, then stored into r1. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r6 and r4 values and computed, then stored into r2.

```
GPR[1]=00000013,

GPR[2]=00000000,
```

Fig 5.6: GPR value after execution of above instructions

5.2.3 IMMEDIATE TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

addi r1,r4,7

andi r2,r7,3

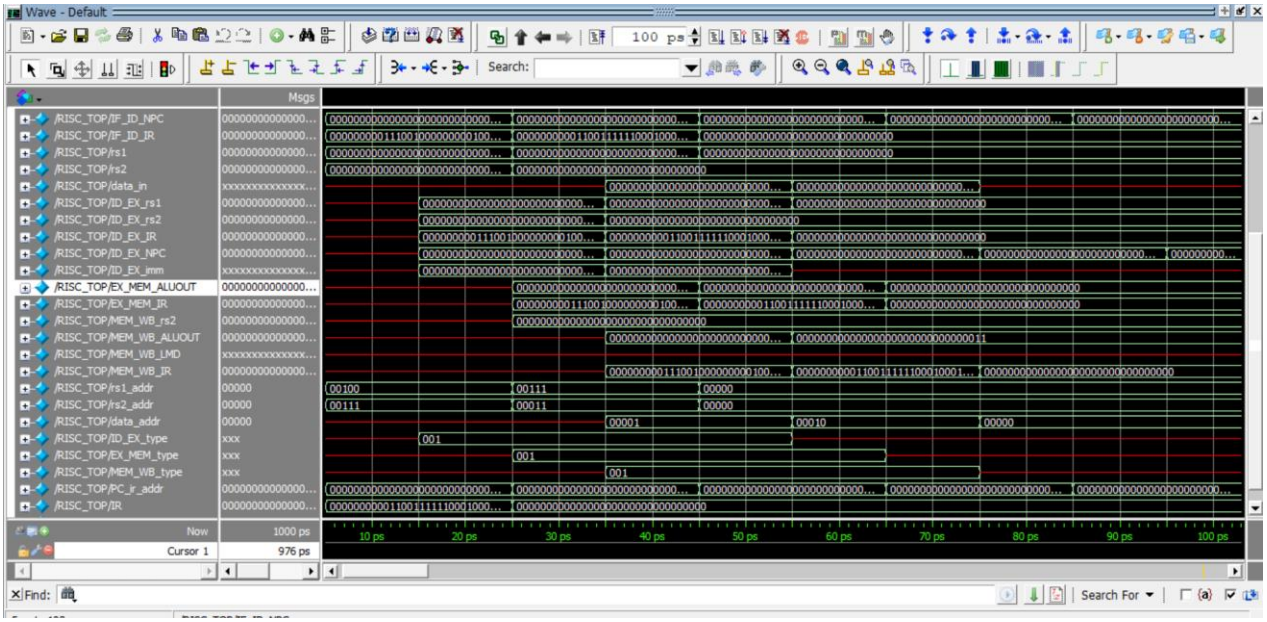

Fig 5.7: Simulation Example of IMMEDIATE TYPE INSTRUCTIONS

First the contents of r4 is fetched and effective value is computed, then stored into r1. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r7 value and and to compute the effective value, then stored into r2.

```
GPR[1]=0000000a,

GPR[2]=00000003,
```

Fig 5.8: GPR value after execution of above instructions

5.2.4 STORE TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

sb r4,01(r0)

sw r7,02(r0)

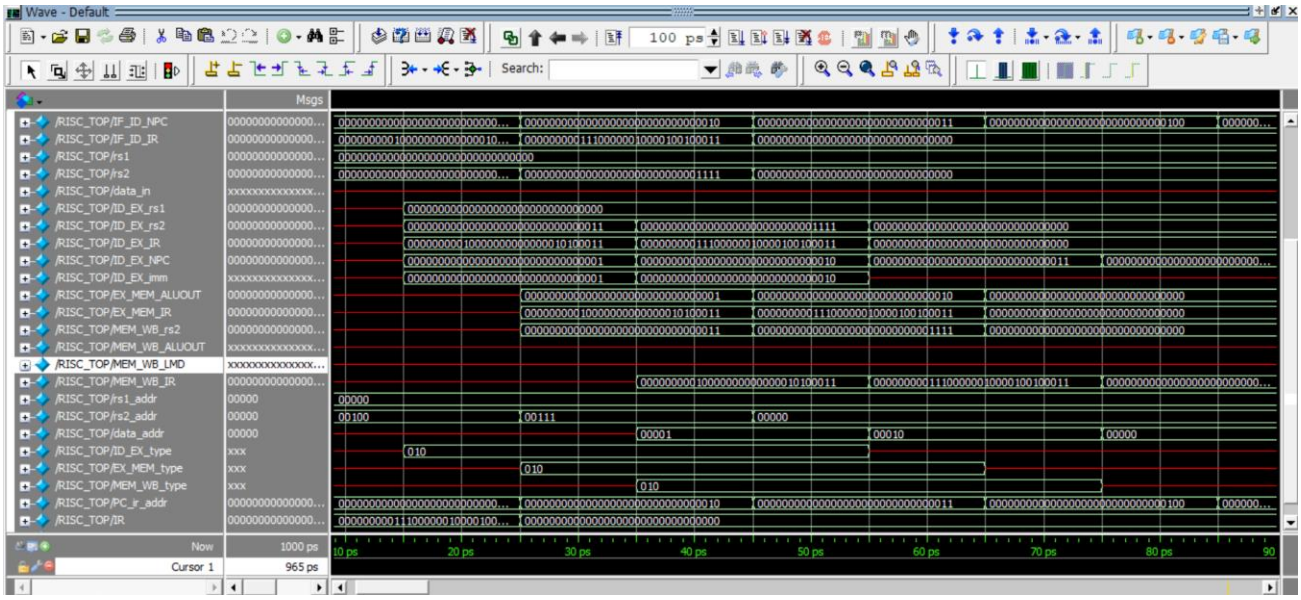

Fig 5.9: Simulation Example of STORE TYPE INSTRUCTIONS

First the contents of r4 is fetched and effective address is computed, then stored. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r7 and the effective address is computed, then stored.

```
Data[1]=00000003

Data[2]=0000000f
```

Fig 5.10: GPR value after execution of above instructions

5.2.5 LOAD TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

lw r6,1(r0)

lb r7,2(r0)

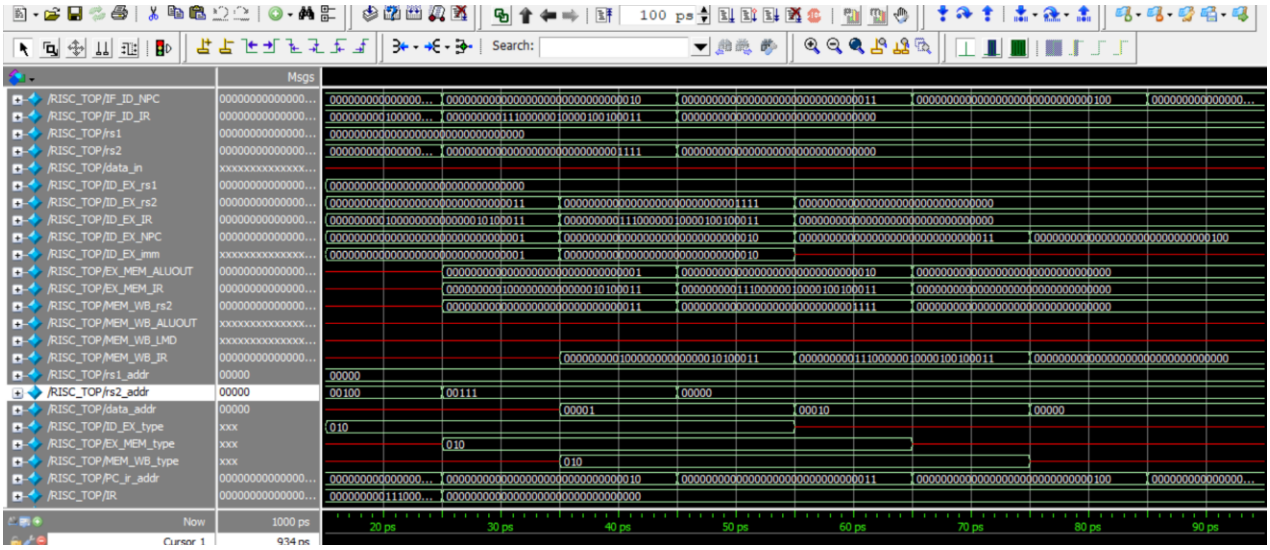

Fig 5.11: Simulation Example of LOAD TYPE INSTRUCTIONS

First the contents of r4 is fetched and effective address is computed, then stored. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r7 and the effective address is computed, then stored.

```
GPR[6]=00000000,

GPR[7]=00000000,
```

Fig 5.12: GPR value after execution of above instructions

5.2.6 BRANCH TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

beq r1,r2,12

bne r1,r2,5

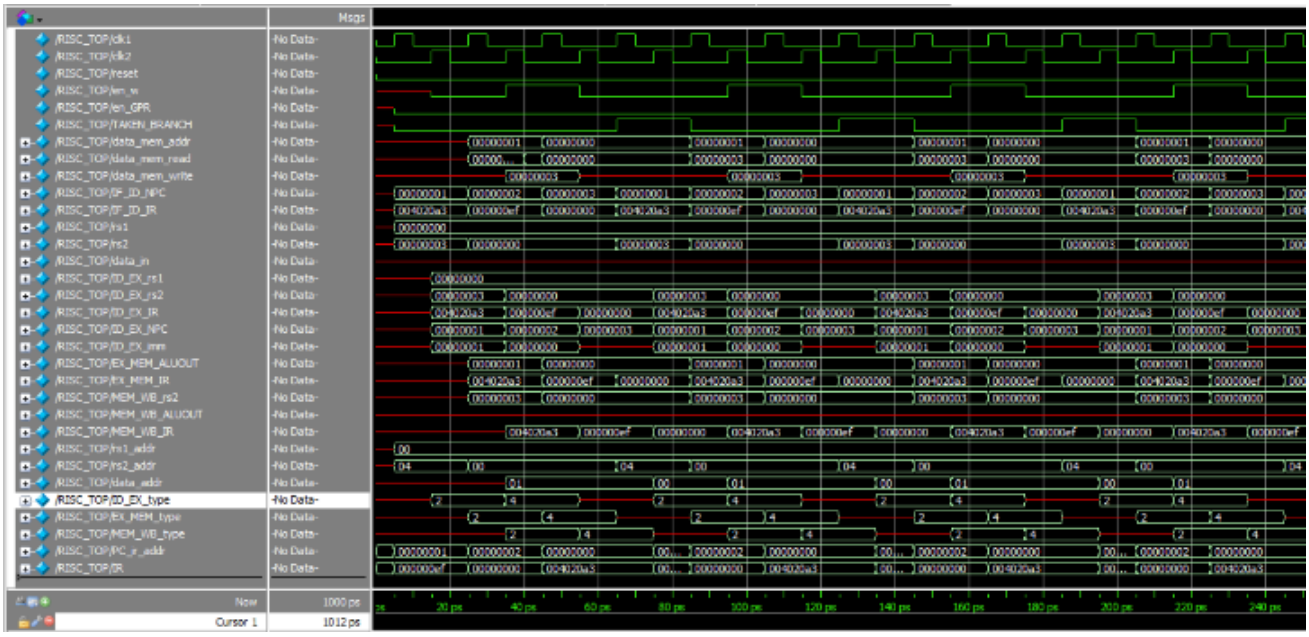

Fig 5.13: Simulation Example of LOAD TYPE INSTRUCTIONS

First the contents of r1 and r2 is fetched and compared. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r1 and r4. Then branch is decided based on the condition.

5.2.7 JUMP TYPE INSTRUCTIONS

Consider the following instructions loaded into the instruction memory from position 0. The instructions are executed in a pipeline fashion as shown using modelsim in the figure below.

Eg:-

jal r1,12

jalr r1,5(r3)

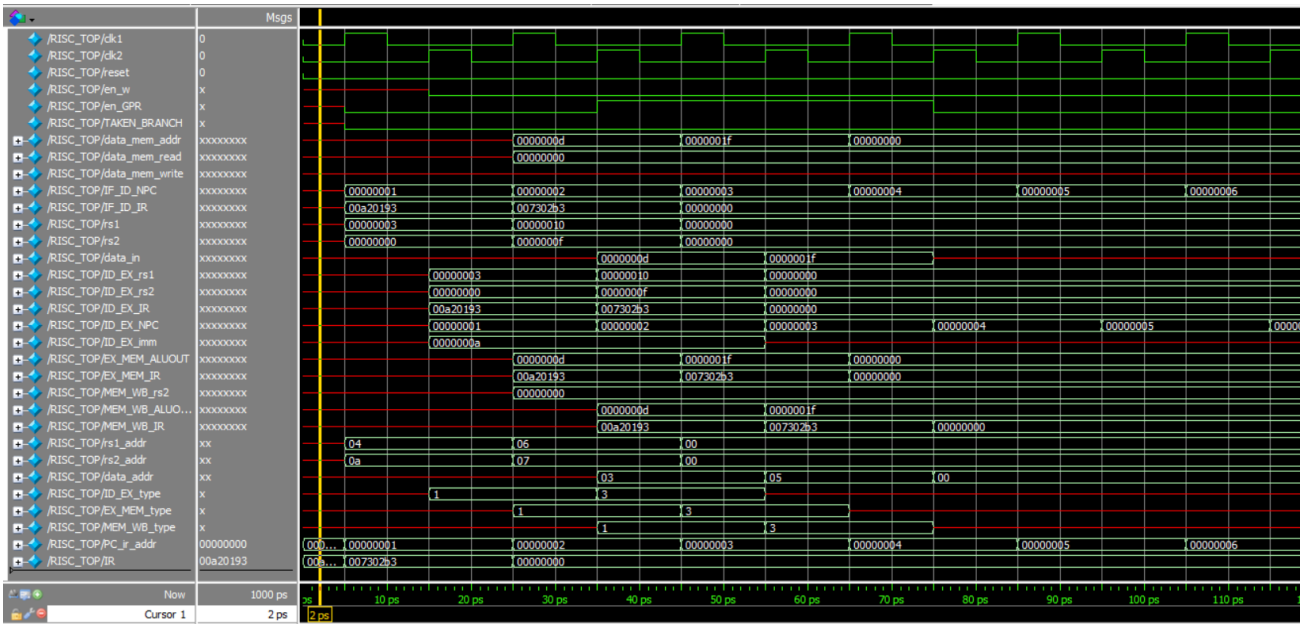

Fig 5.14: Simulation Example of LOAD TYPE INSTRUCTIONS

First the contents of r1 is fetched and effective address is computed, then jump occurs. At the same time in the continuing cycle the hardware is used paralleled by the next instruction to fetch r1 and the effective address is computed then jump occurs.

5.3 IMPLEMENTATION ON FPGA

This processor is synthesized using Quartus Prime. Figure 16 shows the Register Transfer Logic diagram of the proposed 32-bit RISC-V processor. The processor is implemented on ALTERA Cyclone 10 LP 10CL080ZF484I8G FPGA. Table 1: shows the device utilization of the proposed processor.

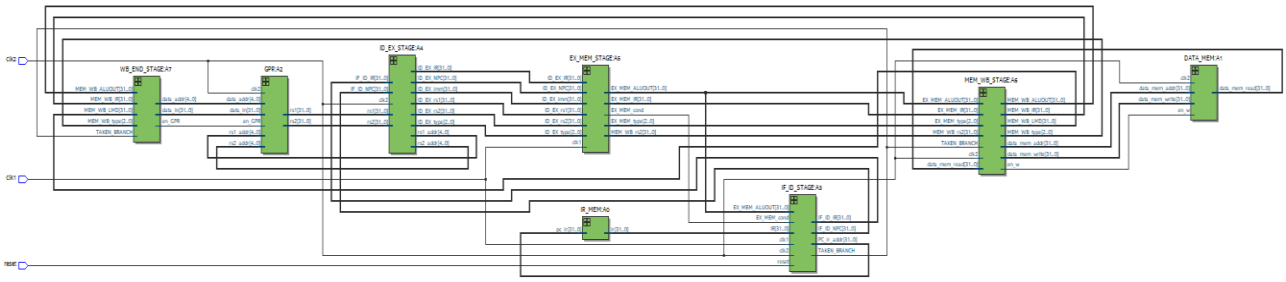

Fig 5.15: RTL View of the Processor Design

Parameter	Utilization
Combinational LUTs	1,599 / 81,264 (2 %)
Logic Registers	410
Max .Frequency	250 MHZ

Table 5.1: Device Utilization

6. CONCLUSION AND AND FUTURE SCOPE

6.1 CONCLUSION

THE DESIGN OF A MULTIPLE CLOCK DOMAIN PIPELINED RISC PROCESSOR IS DISCUSSED IN THIS WORK. ON THE BASIS OF RV32I ISA V 2.0, WE HAVE INCORPORATED OVER 30 INSTRUCTIONS INTO THE ARCHITECTURE. WE INCREASED THROUGHPUT BY LOWERING THE NUMBER OF CLOCKS PER INSTRUCTION VIA PIPELINING.

QUARTUS PRIME WAS USED TO SYNTHESIS THE DESIGN, AND MODELSIM WAS USED TO MODEL IT. EXTENSIVE SIMULATIONS WERE USED TO VERIFY EACH INSTRUCTION SEPARATELY. METASTABILITY AND UNPREDICTABLE DELAYS LIKE CLOCK SKEW BY USING NON-OVERLAPPING TWO-PHASE CLOCKS.

The future depends on the computing speed of these RISC processors. IOT devices has already started emerging in this century. More and more RISC based processors will be designed and verified for the consumers. Thus more flexible designs are needed like the one being projected in this report.

The processor can be further converted from RTL to GDSI for a specific technology node. Where the power and area can be optimized for tape out.

After fabrication the processor can be installed with an OS to make it work as applicable to a application.

Then further it can be loaded with applications and output devices to make to consumer friendly to use.

6.2 FUTURE SCOPE

The existing processor can be enhanced by the following ways:-

- Branch prediction and Branch history table can be added for better branching and for reducing stalls by branch
- Data forwarding unit can be added to reduce data delay slots by the compiler and to make the throughput high.
- Multiple cycle units can be added to increase the throughput.
- Multiple issue and multiple execution can be done by implementing the same concept by using tomasulos algorithm.
- Interrupt error checker can be added.
- Error information from interrupts can be forwarded by registers by pipelining for finding the fault.

The processor can be further converted from RTL to GDSI for a specific technology node. Where the power and area can be optimized for tape out.

After fabrication the processor can be installed with an OS to make it work as applicable to a application.

Then further it can be loaded with applications and output devices to make to consumer friendly to use.

7. REFERENCES

[1] M. N. Topiwala and N. Saraswathi, "Implementation of a 32-bit MIPS based RISC processor using Cadence," 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, 2014, pp. 979-983, doi: 10.1109/ICACCCT.2014.7019240.

[2] S. P. Ritpurkar, M. N. Thakare and G. D. Korde, "Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL," 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), 2014, pp. 1-6, doi: 10.1109/ICAETR.2014.7012843.

[3] S. P. Ritpurkar, M. N. Thakare and G. D. Korde, "Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL," 2015 International Conference on Advanced Computing and Communication Systems, 2015, pp. 1-6, doi: 10.1109/ICACCS.2015.7324067.

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

[5] S. Palekar and N. Narkhede, "32-Bit RISC processor with floating point unit for DSP applications," 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2016, pp. 2062-2066, doi: 10.1109/RTEICT.2016.7808202.

[6] A. Raveendran, V. B. Patil, D. Selvakumar and V. Desalphine, "A RISC-V instruction set processor-micro-architecture design and analysis," 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, pp. 1-7, doi: 10.1109/VLSI-SATA.2016.7593047.

[7] P. V. S. R. Bharadwaja, K. R. Teja, M. N. Babu and K. Neelima, "Advanced low power RISC processor design using MIPS instruction set," 2015 2nd International Conference on Electronics and Communication Systems (ICECS), 2015, pp. 1252-1258, doi: 10.1109/ECS.2015.7124785.

[8] A. Ashok and V. Ravi, "ASIC design of MIPS based RISC processor for high performance," 2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), 2017, pp. 263-269, doi: 10.1109/ICNETS2.2017.8067945.

[9] J. V. Kumar, B. Nagaraju, C. Swapna and T. Ramanjappa, "Design and development of FPGA based low power pipelined 64-Bit RISC processor with double precision floating point unit," 2014 International Conference on Communication and Signal Processing, 2014, pp. 1054-1058, doi: 10.1109/ICCSP.2014.6950008.

[10] R. J. L. Austria, A. L. Sambile, K. M. Villegas and J. N. T. Tabing, "Design of an 8-bit five stage pipelined RISC microprocessor for sensor platform application," TENCON 2017 - 2017 IEEE Region 10 Conference, 2017, pp. 2110-2115, doi: 10.1109/TENCON.2017.8228209.

[11] R. Aneesh. and K. Jiju., "Design of FPGA based 8-bit RISC controller IP core using VHDL," 2012 Annual IEEE India Conference (INDICON), 2012, pp. 427-432, doi: 10.1109/INDCON.2012.6420656.

Sairam INSTITUTIONS

DEPARTMENT OF
ELECTRONICS AND
COMMUNICATION ENGINEERING

IEEE

Sri SAI RAM ENGINEERING COLLEGE
An Autonomous Institution
West Tambaram, Chennai - 44
www.sairam.edu.in

2022 INTERNATIONAL CONFERENCE ON
COMMUNICATION, COMPUTING & INTERNET OF THINGS
(IC3IoT- 2022)

Certificate

of Presentation

This certificate is presented to Dr./Prof./ Mr./Ms. Sandeep Prabhakaran

has presented a paper titled Design and Analysis of a Multi Clocked Pipelined Processor Based on RISC-V

in the ***2022 International Conference on Communication Computing & Internet of Things***

(IC3IOT-2022) *held on 10th and 11th March 2022.*

Dr. S. Brindha	Dr. J. Raja	Dr. K. Porkumaran	Sai Prakash LeoMuthu
Conference Chair	General Chair	Principal	Chairman & CEO

Sairam RAISE Sairam EOMS ARIIA ATAL RANKING OF INSTITUTIONS ON INNOVATION ACHIEVEMENTS EXCELLENT BAND nirf 109 RANK INSTITUTION'S INNOVATION COUNCIL SUSTAINABLE DEVELOPMENT GOALS

Design and Analysis of a Multi Clocked Pipelined Processor Based on RISC-V

Sandeep Prabhakaran
Dept. of ECE,
Sathyabama Institute of Science and Technology
Chennai, India

Mathan N
Dept. of ECE,
Sathyabama Institute of Science and Technology
Chennai, India

V Vedanarayanan
Dept. of ECE,
Sathyabama Institute of Science and Technology
Chennai, India

***Abstract*— The main goal of this study is to develop a 32-bit pipelined processor with several clock domains based on the RISC-V (open source RV32I Version 2.0) ISA. To minimise the complexity of the instruction set and speed up the execution time per instruction, RISC (Reduced Instruction Set Computer) is a type of processor that uses less hardware than CISC (Complex Instruction Set Computer) is used. Furthermore, we constructed this processor with five levels of pipelining, resulting in parallelism in instruction execution. With the aid of necessary block diagrams, all of the processes are well described. Multiple clock domains employing two clock sources are used to ensure that variable delays such as clock skew and metastability are avoided within the stage pipeline registers. Quartus Prime was used to design and synthesis this processor, which was written in Verilog HDL. ModelSim was used to verify this design, and all of the instructions have been thoroughly checked. Furthur the processor is implemented on the "ALTERA Cyclone 10 LP" board for calculating the device utilization.**

Keywords—RISC, Multiple Clock Domains, Pipeline, Verilog, Processor.

I. INTRODUCTION

Over the past decades, microprocessors and microcontrollers has been constructed around two types of architectures, Reduced Instruction Set computer and Complex Instruction Set Computer [1]. CISC instructions are variable in length and are encoded for doing more micro operations per instruction. As a result, the complex architecture of CISC processors makes instructions take a longer time to execute [2]. Since all instructions of a RISC processor have the same instruction length, the decoding process becomes easier compared to a CISC processor. RISC is widely used due to its efficient architecture which can be used for low power and high speed processing application. It supports very few addressing modes, LOAD and STORE instructions are the only instructions that are used to access the external memory. Hence RISC processors are mainly dependent on software and CISC processors are mainly dependent on hardware for executing complex tasks.

Every processor architecture design's primary aim is to keep Clock per Instruction (CPI) close to 1 which can be always challenging. To increase the throughput we have implemented the processor using 5 stages pipelined architecture, by implementing separate stages for Fetch instructions, Decoding, Arithmetic operations, Memory access and write back. Through pipelining on each cycle, an instruction can be executed. Execution of instructions in parallel will also introduce the risk of pipeline hazards during the execution of multiple instructions. Structural hazards are encountered when multiple instructions use a common resource at the same time. This has been eliminated by implementing the processor using Harvard architecture with separate data and instruction memory, also a general-purpose memory with two read ports and one write port. Therefore, several data accesses can be performed simultaneously without conflict. Data hazards are encountered due to the usage of common source and destination resources in consecutive instructions, this occurs when the source for instruction is the destination for the previous instruction [3]. Data hazards can be prevented by inserting dummy instructions during compiling using the compiler to create a gap between those instructions. Control hazards are encountered during the successful execution of branch and jump instructions. This can be prevented by a branch flag which goes HIGH when the branch is taken, thus following instructions following after the branch in memory and write backstage are terminated.

The organization of the paper is as follows. Section II explains the ISA of RISC-V RV32I in detail. Section III presents the working of individual stages and memory units of the processor. Sections IV describes the construction of multiple clock domains. Section V shows the simulated waveforms in ModelSim. Section VI shows the RTL view and Device utilization summary of the design. The final section provides the Conclusion and References.

II. INSTRUCTION SET ARCHITECTURE (RV32I)

The RISC-V (RV32I) instruction set has a fixed length of 32 bits. It is designed to form a sufficient compile target and support modern OS environments [4]. It was constructed in a way that it reduces the hardware needed for minimum implementation. It has 32 general purpose registers reg0 to reg31 and reg0 is hardwired to the constant 0. There are six instruction formats in the RV32I instruction set: R-type, U-type, I-type, B-type, J-type. and S-type. All of the types are explained in the following section.

A. R-type RV32I Instruction Format

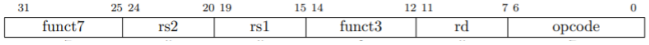

Fig. 1. R-Type RV32I V 2.0 Instruction Format

Figure 1 depicts the Register-type RV32I ISA V 2.0. It has a total of six fields. Opcode width is 7 bits, which is used to indicate the type of instruction. Source registers (rs1, rs2) and destination register (rd) are indicated by five-bit fields. The function field is of a total of 10 bits, which is used for identifying the type of operation to be performed. The instructions which are supported by this format are add, sub, sltu, sll, xor, and, sra, srl, or, and slt.

B. I-type RV32I Instruction Format

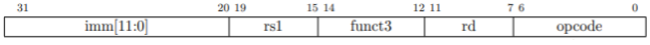

Fig. 2. I-Type RV32I V 2.0 Instruction Format

Figure 1 depicts the Register-type RV32I ISA V 2.0. Similar to R-type, Opcode width is of 7 bits. Source registers (rs1) and destination register (rd) are indicated by five bit fields. The function field of 3 bits, is used for identifying the type of operation to be performed. It has a separate 12 bit field for holding the immediate operand, used for immediate data operations. The instructions which are supported by this format are jalr, lhu, lw, lb, lbu, lh, srai, srli, slli, slti, addi, andi, ori, xori and sltiu.

C. S-type RV32I Instruction Format

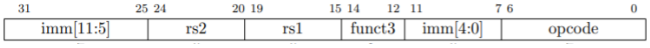

Fig. 3. S-Type RV32I V 2.0 Instruction Format

Figure 3 depicts the Store-type RV32I ISA V 2.0. Similar to R-type, Opcode width is 7 bits. Source registers (rs1 and rs2) are indicated by five bit fields. Function field is of 3 bits, which is used to indicate the size of the data need to be stored. It has a separate 7+5=12 bit field space for holding the immediate operand. This immediate operand is added with rs1 to calculate the address in which the value rs2 needed to be stored. The instructions which are supported by this format are sw, sb and sh.

D. B-type RV32I Instruction Format

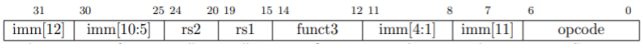

Fig. 4. B-Type RV32I V 2.0 Instruction Format

Figure 4 shows the Branch-type RV32I ISA V 2.0. Similar to other instructions, the Opcode width is of 7 bits. Source registers (rs1 and rs2) are indicated by five bit fields which are used for comparison for branching. The function field is of 3 bits, which is used to indicate the type of condition that need to be checked for branching. It has a separate 7+5=12 bit field space for holding the immediate operand, which is added to the program counter if a branch is taken. The instructions which are supported by this format are bne, bltu, blt, bgeu, bge and beq.

E. U-type & J-type RV32I Instruction Format

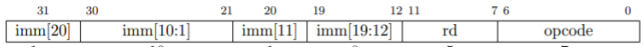

Fig. 5. J-Type RV32I V 2.0 Instruction Format

Fig. 6. U-Type RV32I V 2.0 Instruction Format

Figure 5 shows the U-type and J-type RV32I ISA V 2.0 which are similar to each other. It has a total of two fields. Opcode width is 7 bits, which is used to indicate the type of instruction format. The destination register (rd) is indicated by a five bit field. It has a 20 bit field for holding the immediate operand, used for immediate data operations. For J-type the immediate data is rearranged before branching. The instructions which are supported by this format are jal, lui and auipc.

III. Organisation Of RISC-V Processor

This RISC processor design has been constructed using five pipeline stages. The used pipeline stages are the Instruction Fetch stage (IF), Instruction Decode stage (ID), Execution stage (EX), Memory Access stage (MEM) and Write Back stage (WB). Pipeline registers or latches are used to separate the stages of the processor into 5 parts, so there is no contradictory data due to the execution of multiple instructions. They are named with the prefix as IF_ID, ID_EX, EX_MEM, MEM_WB, and WB_END. They are asserted with two different clock sources for alternate stages, the working of multiple clock domains is discussed in section IV. Other blocks include instruction memory (IR_MEM), Data memory (DATA_MEM), and General purpose registers. The working of all memory units and stages are explained here.

A. Instruction memory

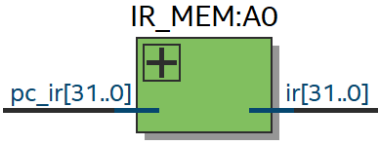

Fig. 7. Instruction Memory Block

All instructions to be performed are stored in the ROM that acts as the instruction memory. The program counter (pc_ir) points to the location address of the next instruction to be executed. The output is the 32-bit instruction, which is sent to the instruction fetch stage.

B. Register file

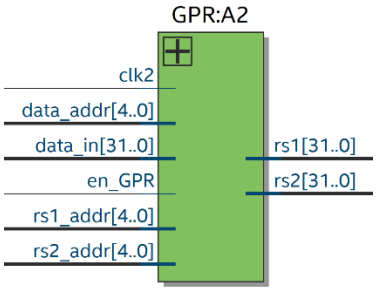

Fig. 8. General Purpose Register file Block

This module consists of 32 registers each of 32 bit in length. The values stored in the General Purpose registers can be read simultaneously twice and written once at the same time. Data can only be written at negative edge of clock 2 if en_GPR is HIGH. The registers in this unit can be used in arithmetic and logical operations either as a source or destination [5].

C. Data Memory

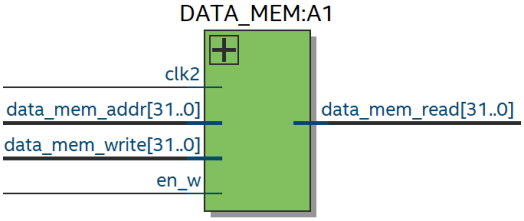

Fig. 9. Data Memory Block

Data memory in this processor functions as RAM. This memory can only be accessed by store and load instructions. The store instruction enables the signal en_w HIGH, so at negative edge of clock 2 data can be written into the memory. Data can be read by using load instructions by setting en_w as LOW.

D. Instruction Fetch Stage

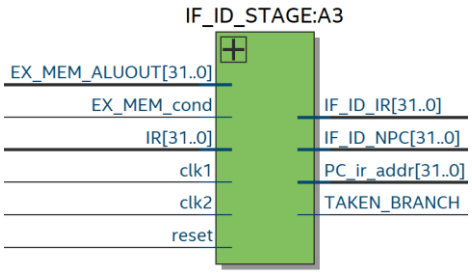

Fig. 10. Instruction Fetch Stage Module

This Stage contains the program counter (PC) which points to the next instruction address to be executed in the instruction memory. Branch condition from execution stage is given as input to this stage, if branch taken the TAKEN_BRANCH signal is asserted HIGH at the positive edge of clock 1. This is the only stage in the processor that needs both the clocks to operate. At the positive edge of clock 1, the Program counter is incremented and if the branch is taken, the new address is written to the program counter. Clock 2 is used for resetting the status of TAKEN_BRANCH. The instruction fetched and its memory address are forwarded to the next stage.

E. Decode Stage

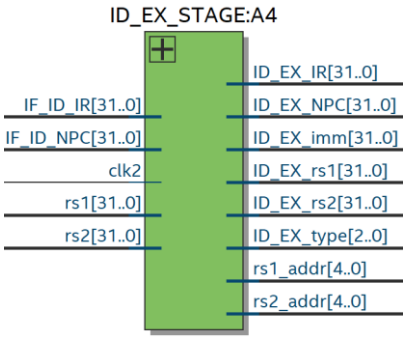

Fig. 11. Decode Stage Module

The instruction is decoded in this stage and decoded information is forwarded at the positive edge of clock 2. Source operands from general-purpose registers are fetched. The immediate data from the instruction is rearranged according to the opcode and are sign-extended to 32 bits. The function field is decoded to find the operation needed to be performed. All the data from the decoder is forwarded to the next stage for further processing.

F. Execute Stage

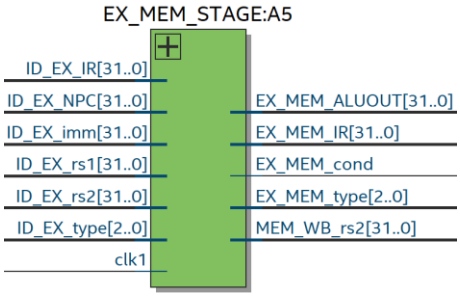

Fig. 12. Execute Stage Module

This stage contains an ALU which is used to do all the arithmetic and logical operations. If a successful branch is taken then EX_MEM_cond is set HIGH alerting the Instruction fetch stage to update the program counter with EX_MEM_ALUOUT value. At the positive edge of clock1 the computed data is forwarded to the next stage.

G. Memory Access Stage

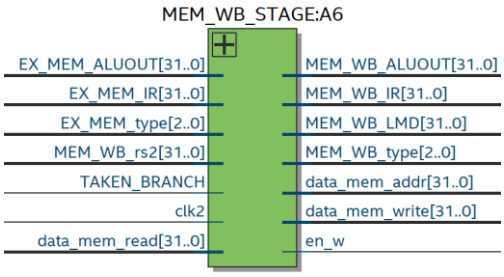

Fig. 13. Memory Access Stage Module

Load and Store operations are performed in this stage. If the instruction is not a memory access instruction then it is ignored and en_w is set to LOW. MEM_WB_LMD is the data read from memory for Load instructions. If TAKEN_BRANCH is HIGH then all write operations are terminated, since the instruction before this has taken the branch. At the positive edge of clock 2, the output is forwarded to data memory or the next stage for write back.

H. Write Back Stage

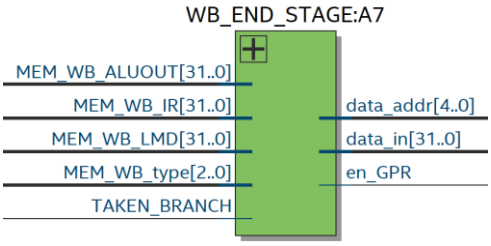

Fig. 14. Write Back Stage Module

In this stage, the output data from the previous stage is uploaded to the address in the destination register. If TAKEN_BRANCH is HIGH then all write operations are terminated.

IV. Construction of Multiple Clock Domains

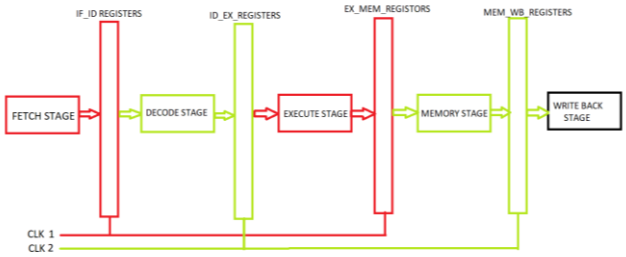

Fig. 15. Clock distribution to the stages

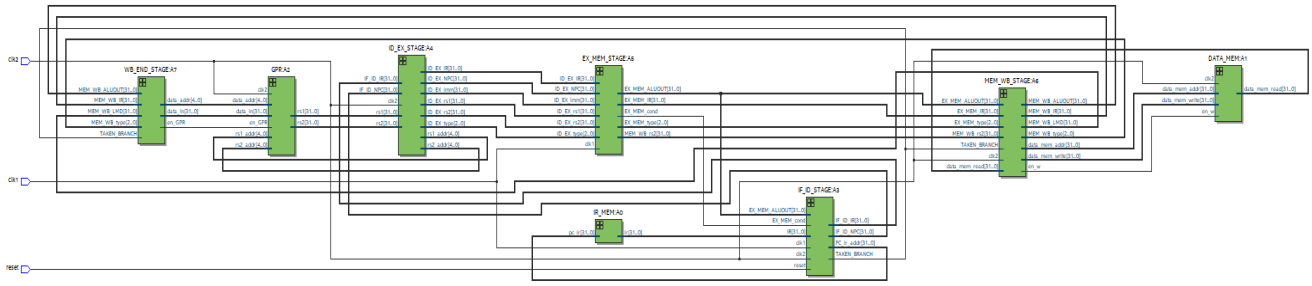

Fig. 16. RTL View of the Processor Design

Two Clock sources (clk1, clk2) are used for consecutive stages as shown in Figure 15. Both clock sources are non-Overlapping as shown in Figure 17.

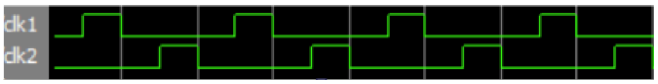

Fig. 17. Non –Overlapping clock sources

They are non-overlapped to take care of variable delays like clocks skew, there is also a gap in between them where both clocks are LOW. This is a very safe kind of clocking scheme where we have non-overlapping clocks with a safe margin in between. For example: since there is no overlap, at the positive edge clock 1, it is guaranteed that EX_MEM is active and the previous stage ID_EX is inactive. Hence the inputs to EX_MEM are held constant so setup & hold time violation does not occur, thus it prevents unstable output or metastable output. Thus by using two-phase clocks isolation of one stage from the other can be achieved.

V. Simulation Results

The proposed microprocessor design is developed using Verilog and simulated using ModelSim, The design is simulated by providing loaded instructions in the instruction memory. Figure 18 shows the simulated output of I-type and R-type instructions, addi R3, R4, 10, add R5, R6, R7.

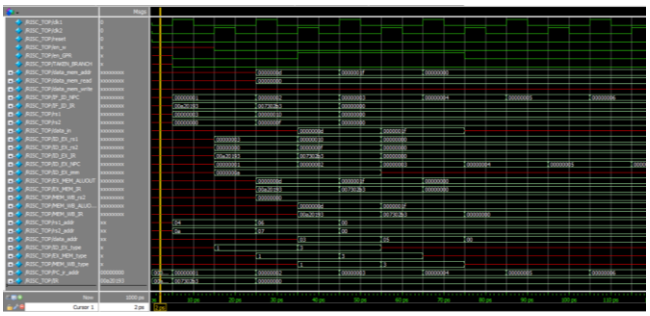

Fig. 18. Simulation Waveform of R-type and I-type Instructions.

Figure 19 shows the simulated output of S-type and J-type instructions, sw R4, 1(R0) and jal R1,0

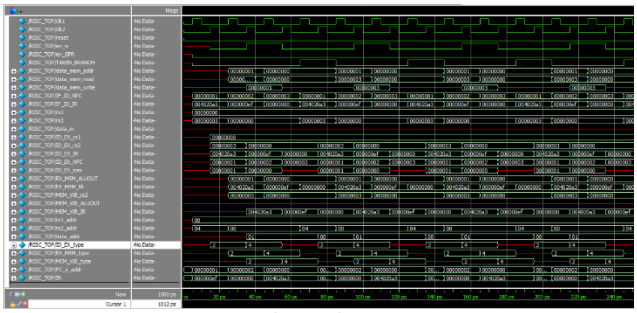

Fig. 19. Simulation Waveform of S-type and J-type Instructions

VI. Implementation On FPGA

This processor is synthesized using Quartus Prime. Figure 16 shows the Register Transfer Logic diagram of the proposed 32-bit RISC-V processor. The processor is implemented on ALTERA Cyclone 10 LP 10CL080ZF484I8G FPGA. Table 1: shows the device utilization of the proposed processor.

Table 1: Device Utilization

Parameter	Utilization
Combinational LUTs	1,599 / 81,264 (2 %)
Logic Registers	410
Max .Frequency	250 MHZ

VII. Conclusion

The design of a multiple clock domain pipelined RISC processor is discussed in this work. On the basis of RV32I ISA V 2.0, we have incorporated over 30 instructions into the architecture. We increased throughput by lowering the number of clocks per instruction via pipelining. Quartus Prime was used to synthesis the design, and ModelSim was used to model it. Extensive simulations were used to verify each instruction separately. Metastability and unpredictable delays like clock skew by using non-overlapping two-phase clocks.

REFERENCES

[1] M. N. Topiwala and N. Saraswathi, "Implementation of a 32-bit MIPS based RISC processor using Cadence," 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, 2014, pp. 979-983, doi: 10.1109/ICACCCT.2014.7019240.

[2] S. P. Ritpurkar, M. N. Thakare and G. D. Korde, "Synthesis and Simulation of a 32Bit MIPS RISC Processor using VHDL," 2014 International Conference on Advances in Engineering & Technology Research (ICAETR - 2014), 2014, pp. 1-6, doi: 10.1109/ICAETR.2014.7012843.

[3] S. P. Ritpurkar, M. N. Thakare and G. D. Korde, "Design and simulation of 32-Bit RISC architecture based on MIPS using VHDL," 2015 International Conference on Advanced Computing and Communication Systems, 2015, pp. 1-6, doi: 10.1109/ICACCS.2015.7324067.

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

[5] S. Palekar and N. Narkhede, "32-Bit RISC processor with floating point unit for DSP applications," 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), 2016, pp. 2062-2066, doi: 10.1109/RTEICT.2016.7808202.

[6] A. Raveendran, V. B. Patil, D. Selvakumar and V. Desalphine, "A RISC-V instruction set processor-micro-architecture design and analysis," 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, pp. 1-7, doi: 10.1109/VLSI-SATA.2016.7593047.

[7] P. V. S. R. Bharadwaja, K. R. Teja, M. N. Babu and K. Neelima, "Advanced low power RISC processor design using MIPS instruction set," 2015 2nd International Conference on Electronics and Communication Systems (ICECS), 2015, pp. 1252-1258, doi: 10.1109/ECS.2015.7124785.

[8] A. Ashok and V. Ravi, "ASIC design of MIPS based RISC processor for high performance," 2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), 2017, pp. 263-269, doi: 10.1109/ICNETS2.2017.8067945.

[9] J. V. Kumar, B. Nagaraju, C. Swapna and T. Ramanjappa, "Design and development of FPGA based low power pipelined 64-Bit RISC processor with double precision floating point unit," 2014 International Conference on Communication and Signal Processing, 2014, pp. 1054-1058, doi: 10.1109/ICCSP.2014.6950008.

[10] R. J. L. Austria, A. L. Sambile, K. M. Villegas and J. N. T. Tabing, "Design of an 8-bit five stage pipelined RISC microprocessor for sensor platform application," TENCON 2017 - 2017 IEEE Region 10 Conference, 2017, pp. 2110-2115, doi: 10.1109/TENCON.2017.8228209.

[11] R. Aneesh. and K. Jiju., "Design of FPGA based 8-bit RISC controller IP core using VHDL," 2012 Annual IEEE India Conference (INDICON), 2012, pp. 427-432, doi: 10.1109/INDCON.2012.6420656.