DESIGNING OF A LOW LATENCY FLOATING POINT UNIT BASED ON RISC V ARCHITECTURE

Arit Kumar Ghosh

MASTER OF TECHNOLOGY IN VLSI DESIGN

ELECTRONICS & COMMUNICATION ENGINEERING DEPARTMENT

NATIONAL INSTITUTE OF TECHNOLOGY, AGARTALA

INDIA- 799046

December, 2022

M. Tech Thesis

DESIGNING OF A LOW LATENCY FLOATING POINT UNIT BASED ON RISC V ARCHITECTURE

Arit Kumar Ghosh

N I T Agartala

2022

DESIGNING OF A LOW LATENCY FLOATING POINT UNIT BASED ON RISC V ARCHITECTURE

Project report submitted to
National Institute of Technology, Agartala
for the award of the degree

of

Master of Technology in VLSI Design

By

Arit Kumar Ghosh (Enrolment No: 21PEC006)

Under the Guidence of

Dr. Kamalesh Debnath
Assistant Professor, Dept. of ECE, NIT Agartala

ELECTRONICS & COMMUNICATION ENGINEERING DEPARTMENT

NATIONAL INSTITUTE OF TECHNOLOGY, AGARTALA

December, 2022

APPROVAL SHEET

This project report entitled "**DESIGNING OF A LOW LATENCY FLOATING POINT UNIT BASED ON RISC V ARCHITECTURE"** by Arit Kumar Ghosh is approved for the degree of Master of Technology in VLSI Design.

_____

_____

_____

Examiners

_____

Dr. Kamalesh Debnath
Project Supervisor
Assistant Professor
Dept. of ECE, NIT Agartala

_____

Dr. Tamasi Moyra Panua
Head of the Department
Dept. of ECE, NIT Agartala

Date: 12.12.2022
Place: Agartala, Tripura

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

(Signature)

Arit Kumar Ghosh
(Name of the student)

21PEC006
(Roll No.)
Date: 12.12.2022

CERTIFICATE

It is certified that the work contained in the thesis titled "**DESIGNING OF A LOW LATENCY FLOATING POINT UNIT BASED ON RISC V ARCHITECTURE**" by Arit Kumar Ghosh has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

_____

Dr. Kamalesh Debnath

Project Supervisor

December, 2022

Assistant Professor

Dept. of Electronics & Communication Engineering

NIT Agartala

PREFACE

New open-source RISC V is created in response to processor advancements to assist research and instruction in computer architecture. As the complexity of the process rises daily, better algorithms are being devised to cut down on resource usage and increase latency. The Floating-Point Unit (FPU) is built to work as a co-processor with RISC V, completing all arithmetic floating-point operations more quickly. With the help of a technique that is an extension of an arithmetic logic unit, this project aims to shorten the processing time for a particular application.

The purpose of this paper is to examine several techniques for cutting down on space and resources and speeding up various floating-point operations on FPU. The approach has been to use an existing architecture as a starting point, pinpoint problem areas and bottlenecks in common applications, and then investigate several possible algorithm modifications that might have a positive performance impact.

ACKNOWLEDGEMENT

I would like to thank our mentors and guide for their constant guidance during the project. We would sincerely like to thank our guide **Dr. Kamalesh Debnath** for his guidance, wisdom, and time to time help.

I would like to express my deep sense of gratitude and indebtedness to my mentor and guide **Mr. Pranay Nath** for his invaluable encouragement, suggestions, and support from an early stage of this research and providing me extraordinary experiences throughout the work. Above all, his priceless and meticulous supervision at each phase of work inspired me in innumerable ways.

I am highly grateful to **Dr. Sambhu Nath Pradhan** for their kind support and permission to use the facilities available in the lab of the department. We would also like to extend our special thanks to **Dr. Biman Debbarma** for helping and encouragement during the project.

Finally, we would like to thank our family members, friends and peers for their constant motivation, support, and guidance during the hard times without which the accomplishment of theproject would have been difficult.

LIST OF FIGURES

Fig 1 – Single Precision IEEE 754 Floating-Point Standard 11

Fig 2 – Two floating point numbers to be multiplied 15

Fig 3 – Floating Point Multiplication Result 16

Figure 4: Floating Point Number Format 17

Fig 5 - Schematic of Addition 21

Fig 6 - Flow chart of the multiplication operation 23

Fig 7 - RTL schematic of the multiplication operation 24

Fig 8 – FMA schematic 25

Fig 9 – FMA Elaborated schematic 25

Fig 10 – FMA Simulation output case 1 27

Fig 11– FMA Simulation output case 2 27

Fig 12 – FMA Simulation output case 3 28

Fig 13 – FMA Resource Utilization Summary 28

LIST OF TABLES

Table 1: Sign Operation 18

Table 2: FMA output with various test cases 26

LIST OF SYMBOL AND ABBREVIATIONS

IP - Intellectual Property

RISC V - Reduced Instruction Set Computer V

ISA - Instruction Set Architecture

CISC - Complex Instruction Set Computer

ALU - Arithmetic Logical Unit

CPU - Central Processing Unit

FPU - Floating Point Unit

FP - Floating Point

IEEE - Institute of Electrical and Electronics Engineers

NaN - Not a Number

VHDL - Very High-Speed Integration Circuit HDL

VLSI - Very large-scale integration

DPFP - Double Precision Floating Point

FMA - Fused Multiplication Addition

FSUB - Fused Subtraction

MIPS - Million instructions per second

FPGA - Field Programmable Gate Arrays

CSA - Carry Save Adder

CLA - Carry Look Ahead Adder

HDL - Hardware Description Language

RTL - Register-Transfer Level

LUT - Look Up Table

FF - Flip Flop

ABSTRACT

One of the essential components of advanced processors is the floating-point unit. On floating point units, arithmetic operations are particularly challenging. Either 32-bit (single precision) or 64-bit IEEE 754 format is used to represent them (double precision). For a variety of applications, including mathematical analysis and formulation, signal processing, etc., they are widely utilized in high-end processors. This report outlines the precise steps involved in computing the FMA operation of floating-point integers. Verilog HDL was used in its design. To determine the space occupied and its performance in terms of delay, the design has been synthesized and simulated.

CONTENTS

TITLE PAGE 1		i
TITLE PAGE 2		ii
TITLE PAGE 3		iii
APPROVAL SHEET		iv
DECLARATION		v
CERTIFICATE		vi
PREFACE		vii
ACKOGDEMENT		viii
LIST OF FIGURES		ix
LIST OF TABLES		x
LIST OF SYMBOL AND ABBREVIATIONS		xi
ABSTRACT		xii
CONTENTS		xiii - xiv
CHAPTER 1 INTRODUCTION		1 - 6
	1.1 INTRODUCTION	1
	1.2 FLOATING POINT UNIT IEEE 754 STNDARDS	1
	1.3 IEEE 754 STNDARDS	1
	1.4 MOTIVATION AND OBJECTIVE	3
	1.5 RISC V INTRODUCTION	4
	1.6 RISC V ARCHITECTURE	4
	1.7 RISC V FEATURES	5
	1.8 RISC V ADVANTAGES	5
CHAPTER 2 LITERATURE REWVIEW		7 - 10
	2.1 INTRODUCTION	7
	2.2 PREVIOUS WORKS' REVIEW	7
CHAPTER 3 PROPOSED WORK AND METHODOLOGY		11 - 16
	3.1 INTRODUCTION	11
	3.2 SINGLE PRECISION FPN	11
	3.3 FP ADDITION METHOD	13

3.4 FP MULTIPLICATION MRTHOD 15
3.5 FMA METHOD 16

CHAPTER 4 FUSED-MULTIPLICATION ADDITION 17 - 25
4.1 INTRODUCTION 17
4.2 FP ADDITION 17
4.2.1 ADDITION STEPS 17
4.2.2 FLOW CHART OF ADDITION 19
4.2.3 ADDER BLOCK 19
4.2.4 STANDARDIZING BLOCK 20
4.2.5 DESIGN SUB MODULES 20
4.2.6 ALGORITHM OF FP ADDITION 20
4.2.7 RTL VIEW OF ADDITION 21
4.3 FP MULTIPLICATION 21
4.3.1 FLOW CHART OF MULTIPLICATION 23
4.3.2 RTL SCHEMATIC OF MULTIPLICATION 24
4.4 FMA 24
4.4.1 FMA RTL SCHEMATIC 25

CHAPTER 5 RESULT & ANALSIS 26 - 29
5.1 INTRODUCTION 26
5.2 SIMULATION RESULT & WAVEFORM 26
5.3 RESOURSE UTILIZATION 28
5.4 CONCLUSION

CAHPTER 6 CONCLUSION & FUTURE SCOPE 30
6.1 CONCLUSION 30
6.2 FUTURE SCOPE 30

REFERENCES 31 - 32

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Floating-point units (FPU) colloquially are a math coprocessor which is designed specially to carry out operations on floating point numbers. Typically, FPUs can handle operations like addition, subtraction, multiplication, and division. FPUs can also perform various transcendental functions such as exponential or trigonometric calculations, though these are done with software library routines in most modern processors. Our FPU is basically a single precision IEEE754 compliant integrated unit. In this chapter we have basically introduced the basic concept of what an FPU is, in the section 1.2. Following the section, we have given a brief introduction to the IEEE 754 standards in section 1.3. After describing the IEEE 754 standards, we have explained the motivation and objective behind this project in section 1.4.

1.2 FLOATING POINT UNIT

Then a CPU executes a program that is calling for a floating-point (FP) operation, there are three ways by which it can carry out the operation. Firstly, it may call a floating-point unit emulator, which is a floating-point library, using a series of simple fixed-point arithmetic operations which can run on the integer ALU. These emulators can save the added hardware cost of a FPU but are significantly slow. Secondly, it may use an add-on FPUs that are entirely separate from the CPU, and are typically sold as an optional add-on which are purchased only when they are needed to speed up math-intensive operations. Else it may use integrated FPU present in the system.

The FPU designed by us is a single precision IEEE754 compliant integrated unit. It can handle not only basic floating-point operations like addition, subtraction, multiplication and vision but can also handle operations like shifting, square root determination and other transcendental functions like sine, cosine and tangential function.

1.3 IEEE 754 STNDARDS

IEEE754 standard is a technical standard established by IEEE and the most widely used standard for floating-point computation, followed by many hardware (CPU and FPU) and software implementations [3]. Single-precision floating-point format is a computer number

format that occupies 32 bits in a computer memory and represents a wide dynamic range of values by using a floating point. In IEEE 754-2008, the 32-bit with base 2 format is officially referred to as single precision or binary32. It was called single in IEEE 754-1985. The IEEE 754 standard specifies a single precision number as having sign bit which is of 1 bit length, an exponent of width 8 bits and a significant precision of 24 bits out of which 23 bits are explicitly stored and 1 bit is implicit 1.

Sign bit determines the sign of the number where 0 denotes a positive number and 1 denotes a negative number. It is the sign of the mantissa as well. Exponent is an 8-bit signed integer from −128 to 127 (2's Complement) or can be an 8 bit unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 single precision definition. In this case an exponent with value 127 represents actual zero. The true mantissa includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1 unless the exponent is stored with all zeros. Thus only 23 fraction bits of the mantissa appear in the memory format but the total precision is 24 bits.

For example:

S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF

31 30 23 22 0

IEEE754 also defines certain formats which are a set of representation of numerical values and symbols. It may also include how the sets are encoded. The standard defines:

- Arithmetic formats which are sets of binary and decimal floating-point numbers, which consists of finite numbers including subnormal number and signed zero, a special value called "not a number" (NaN) and infinity.
- Interchange formats which are bit strings (encodings) that are used to exchange a floating-point data in a compact and efficient form.
- Rounding rules which are the properties that should be satisfied while doing arithmetic operations and conversions of any numbers on arithmetic formats.
- Exception handling which indicates any exceptional conditions (like division by zero, underflow, overflow, etc.) occurred during the operations.

The standard defines the following five rounding rules:

- Round to the nearest even which rounds to the nearest value with an even (zero) least significant bit.
- Round to the nearest odd which rounds to the nearest value above (for positive numbers) or below (for negative numbers).
- Round towards positive infinity which is a rounding directly towards a positive infinity and it is also called rounding up or ceiling.
- Round towards negative infinity which is rounding directly towards a negative infinity and it is also called rounding down or floor or truncation.

The standard also defines five exceptions, and all of them return a default value. They all have a corresponding status flag which are raised when any exception occurs, except in certain cases of underflow. The five possible exceptions are:

- Invalid operation is like square root of a negative number, returning of qNaN by default, etc., output of which does not exist.
- Division by zero is an operation on a finite operand which gives an exact infinite result for e.g., 1/0 or log (0) that returns positive or negative infinity by default.
- Overflow occurs when an operation results a very large number that can't be represented correctly i.e., which returns ±infinity by default (for round-to-nearest mode).
- Underflow occurs when an operation results very small i.e., outside the normal range and inexact (deformalized value) by default.
- Inexact occurs when any operation returns correctly rounded result by default.

1.4 MOTIVATION AND OBJECTIVE

Floating-point calculation is an esoteric subject in the field of Computer Science. This is obviously surprising, because floating-point is omnipresent in computer systems. Floating-point (FP) data type is almost present in every language. From PCs to supercomputers, all have FP accelerators in them. Most compilers are called from time to time to compile the floating-point algorithms and virtually every OS must respond to all FP exceptions during operations such as overflow. Also, FP operations have a direct effect on designs as well as designers of computer systems. So, it is very important to design an efficient FPU such that the computer system becomes efficient. Further, FPU can be improvised by using efficient algorithm for the basic as well as transcendental functions, which can be handled by any

FPU, with reduced complexity of the logic used. This FPU further can be worked upon to improvise further complex operations-viz. exponent, etc. It can be designed so that it can handle different data types like character, strings etc. can serve as a backbone for designing a fault tolerant IEEE754 compliant FPU on higher grounds and such that pipeline can be implemented.

Motivated by the need of efficient FPU for different kind of operations, the objective of the proposed work are as follows:

- To develop an efficient algorithm for FP operations like addition, subtraction, division, multiplication and few transcendental functions.
- To implement the proposed algorithm using Verilog.
- To synthesize the above proposed algorithm.

1.5 RISC V INTRODUCTION

In the RISC V processor, the term RISC stands for "reduced instruction set computer" which executes few computer instructions whereas 'V' stands for the 5th generation. It is an open-source hardware ISA (instruction set architecture) based on the established principle of RISC. As compared to other ISA designs, this ISA is available with an open-source license. So, a few manufacturing companies have announced and provided RISC-V hardware, with open-source operating systems. This is a new architecture and is available in open, non-restrictive & free licenses. This processor has extensive support from chip & device makers industries. So, it is mainly designed to be freely extensible & customizable to use in many applications.

1.6 RISC V ARCHITECTURE

RV12 is highly configurable with a single-core RV32I and RV64I compliant RISC CPU which is used in embedded fields. The RV12 is also from a 32 or 64-bit CPU family depending on the industrial standard RISC-V instruction set. The RV12 simply executes a Harvard architecture for simultaneous access to instruction as well as data memory. It also includes a 6-stage pipeline which helps in optimizing overlaps in between the execution as well as memory accesses to improve efficiency. This architecture mainly includes Branch Prediction, Data Cache, Debug Unit, Instruction Cache, & optional Multiplier or Divider Units.

1.7 RISC V FEATURES

The main features of RV12 RISC V include the following.

- It is an Industry standard instruction set.
- Parameterized with 32 or 64bit data.
- It has precise and fast interrupts.
- Custom instructions allow the addition of proprietary hardware accelerators.
- Execution of single cycle.
- Six-stage pipeline with optimizing folded.
- Support with memory protection.
- Optional or Parameterized caches.
- Extremely Parameterized.
- Users can select 32/ 64-bit data & Branch Prediction Unit.
- Users can select instruction/data caches.
- User selectable structure, size & architecture of cache.
- Hardware Divider or Multiplier Support by user-defined latency.
- The bus architecture is flexible which supports Wishbone & AHB.
- This design optimizes the power & size.
- Design is completely parameterized which provides performance or power tradeoffs.
- Gated CLK design to decrease power.
- Software support by Industry standard.
- Architectural simulator.
- Eclipse IDE is used for Linux/ Windows.

1.8 RISC V ADVANTAGES

The advantages of the RISC V processor include the following,

- By using RISCV, we can save development time, software development, verification, etc.
- This processor has many pros like simplicity, openness, modularity, clean-slate design, and extensibility.
- This is supported by several language compilers like the GCC (GNU Compiler Collection), a free-software compiler & through the Linux OS.

- This can be used by companies freely due to no royalties, no licensing fees & no strings connected.
- RISC-V processor doesn't include any new or innovative features because it simply follows established principles of RISC.
- Similar to several other ISAs, this processor specification simply defines various instruction set levels. So this contains 32 & 64-bit variants as well as extensions to give support for floating point instructions.
- These are free, simple, modular, stable, etc.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

FPU is a math coprocessor which is designed specially to carry out operations on floating point numbers. The FPUs will perform operations like addition, subtraction, multiplication and division. Main function of FPUs can execute different functions such like as exponential or trigonometric calculations, although these are done with software library routine in nearly all recent processors. Our FPU is basically a 32-bit (single precision) IEEE754. In this chapter, investigation of how the classic approaches to numerical computations using floating-point (FP) arithmetic can be improved upon to achieve the fast computation required for flexible computing.

2.2 PREVIOUS WORKS' REVIEW

Jain, Jenil, and Rahul Agrawal et al. [2] this paper grants design of high-speed floating point unit using reversible logic. There are various alterable implementations of logical and arithmetic units have been proposed in the existing research, but very few reversible floating-point designs has been designed. Floating- point processes are used very often in nearly all computing disciplines. The analysis of projected reversible circuit can be done in terms of quantum cost, garbage outputs, constant inputs, power consumption, speed and area.

Gopal, Lenin, Mohd Mahayadin et al. [3] in the newspaper, eight arithmetic and four logical operations has been presented. In the intended design 1, Peres Full Adder Gate (PFAG) is use in reversible ALU plan and HNG gate is used as an adder logic circuit in the planned ALU design 2. Both planned designs are analyze and compare in terms of number of gates calculate, garbage output, quantum price and propagation interruption. The model results show that the proposed reversible ALU design 2 outperforms the proposed reversible ALU design 1 and conventional ALU design.

Nachtigal, Michael ,Himanshu Thapliyaletal.[4] In this work, a innovative design of single precision floating point multiplier has been proposed based on operand decomposition approach. Moreover, a new mutable design of the 8x8 bit Wallace tree multiplier has proposed that is accustomed in terms of quantum cost, delay, and number of garbage outputs.

Wallace tree multiplication involves of three intangible steps: Partial product generation, partial product compression spending 4:2 compressors, full adders, and half adders, and then the ultimate accumulation stage to produce the product. In this slog we perform optimization at each of these three stages.

Dhanabal, R., Sarat Kumar Sahoo et al. [5] present a design using reversible gates. Alterable gates namely TSG gate performs 1-bit addition with carry. This is the first alterable gate which alone can acts as full adder. Gate is used to perform logical actions like AND, OR. In this works, designing 1-bit alum has also been presented using pass transistor with virtuoso tool of cadence. Based on examination of the result, this design using reversible gates is better than that using the irreversible gates.

Nachtigal, Michael, Himanshu Thapliyal, and Nagarajan Ranganathan [6] Floating-point actions are needed very frequently in nearly all computing disciplines, and studies have shown floating-point addition to be the most often used floating-point operation. These paper offerings for the first time a reversible floating-point adder that closely follows the IEEE754 specification for binary floating-point arithmetic. This design requires reversible designs of a controlled swap unit, a subtractor, an alignment unit, signed integer representation conversion units, an integer adder, a normalization unit, and a rounding unit.

Alaghemand, Fatemeh et al. [7] presented a reversible floating-point adder design, because the fixed-point adder is less precise in the representation of numbers. The planned design is made up of several parts, including: Conditional swap, Alignment unit, Converter, Addition and Normalization. We tried to improve the parameters of quantum cost, garbage outputs and constant inputs for these parts and finally compared this design with the existing designs. This planned design has reduced 78% and 30% of the quantum cost, 78% and 26% of the garbage output and 79% and 30% of the constant input in compared with other approaches.

Kahanetal.[8] proposed a dozen commercially vital arithmetic's boasted various word sizes, precisions, misestimating procedures and over/underflow behaviors', and additional were within the works. Suitable software system meant to reconcile that numerical diversity had become unbearably expensive to develop. 13years previous, once IEEE 754 became official, major microchip makers had already adopted it despite the challenge it exhibits to implementers. With new selflessness, hardware designers had up to its challenge within the belief that they might ease and encourage a huge burgeoning of numerical software system.

They did succeed to a substantial extent. Anyway, misestimating a normalizes that preoccupied all folks within the Seventies afflict solely CRAY X-MPs — J90scurrently.

Ykuntam et al. [9] Planned Addition is that the heart of arithmetic unit and the arithmetic unit is commonly the work horse of a machine circuit. Therefore, adders play a key role in planning Associate in Nursing arithmetic unit and additionally several digital integrated circuits. Carry pick Adder is one amongst the quickest adders employed in several information processors and in digital circuits to perform arithmetic operations. However, CSLA is area consuming as a result of it consists of twin ripple carry adder within the structure. To cut back the world of CSLA, a CSLA with Binary to Excess-1 converter is already designed that reduces the world of adder. However, there are unit different techniques to style a CSLA to cut back its space. One amongst such technique is victimization Associate in Nursing add one circuit technique. This paper offers the planning of root CSLA victimization add one circuit with vital reduction in space.

Quinnell et al. [10] planned several new architectures for floating-point amalgamate multiplier adders employed in the x87 units of microprocessors. These fresh architectures are designed to produce solutions to the implementation issues found in modern amalgamate multiply-add units, at the same time increasing their performance and decreasing their power consumption. All new design, additionally as a group of contemporary floating- point arithmetic units used as reference styles for comparison, are styled and enforced victimization the Advanced small Devices sixty-five micro-millimeter atomic number 14 on dielectric junction transistor technology logic gate design tool set.

[11] This paper show evaluation of IEEE floating point unit (FPU) which will carry out multiplication, addition, subtraction and division reason on 32bit operand that uses the IEEE-754 regulation. Floating point numbers representation can support a much wider range of values than fixed point representation. The work is to tool and analyses floating point unit operation and hardware module were implemented using VHDL and synthesized using Xilinx.

[12] Kodali, R.K., Gundabathula, S.K. and Boppana, L investigated the floating point arithmetic, specifically multiplication, is a widely used computational operation in many scientific and signal processing applications. In general, the IEEE-754 single-precision multiplier requires a 23 × 23 mantissa multiplication and the double-precision multiplier requires a large 52 × 52 mantissa multiplier to obtain the final result. This computation exists

as a limit on both area and performance bounds of this operation. A lot of multiplication algorithms have been developed during the past decades. In this research, the two of the popular algorithms, namely, Booth and Karatsuba (Normal and Recursive) multipliers have been implemented, and a performance comparison is also made. The algorithms have been implemented on an uniform reconfigurable FPGA platform providing a comparison of FPGA resources utilized and execution speeds. The recursive Karatsuba is the best performing algorithm among the algorithms.

[13] Mamatha M1 S Pramod Kumar2 In this paper we depict an effective usage of an IEEE 754 single precision floating point multiplier focused for Xilinx Spartan 3E FPGA. Verilog HDL is utilized to actualize an innovation. The multiplier execution handles the overflow and underflow cases. Adjusting is not actualized to give more accuracy when utilizing the multiplier as a part of a Multiply and Accumulate (MAC) unit. The multiplier was confirmed against Xilinx floating point multiplier center produced by Xilinx coregent.Key words: Floating Point; Multiplication; FPGA.

[14] KV Gowree srinivas P.Samundiswary Floating-point arithmetic plays major role in computer systems. Many of the digital signal processing applications use floating-point algorithms for execution of the floating-point computations and every operating system is answerable practically for floating-point special cases like underflow and overflow. The single precision floating point arithmetic operations are multiplication, division, addition and subtraction among all these multiplications is extensively used and involves composite arithmetic functions. The single precision (32-bit) floating point number split into three parts, Sign part, and Exponent part and Mantissa part. The most significant bit of the number is a sign bit and it is a 1- bit length. Next 8-bits represent the exponent part of the number and next 23-bits represent the mantissa part of the number. Mantissa part needs large 24-bit multiplication. The performance of the single-precision floating point number mostly based on the occupied area and delay of the multiplier. In this paper, a novel approach for single-precision floating multiplier is developed by using Urdhva Tiryagbhyam technique and different adders to decrease the complexity of mantissa multiplication. This requires less hardware for multiplication compared to that conventional multipliers and used different regular adders like carry select, carry skip adders and parallel prefix adders for exponent addition.

CHAPTER 3

PROPOSED WORK AND METHODOLOGY

3.1 INTRODUCTION

Designing of floating-point unit (FPU) which utilize less resource and to work on high operating frequency, so that it can act as a co-processor for open-source RISC V to perform different floating-point operations. The proposed work is divided into multiples sub-modules (addition and division). These different sub-modules will create as a single IP, under a one umbrella of floating-point unit (FPU). This floating-point unit can be used to perform different floating-point operation supported by RISC V. The architecture and improved algorithm is developed for efficient use of available resources. To design modules, which support IEEE -754 format and check for all the various possible floating-point numbers as inputs.

3.2 SINGLE PRECISION FLOATING POINT NUMBERS

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.

There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

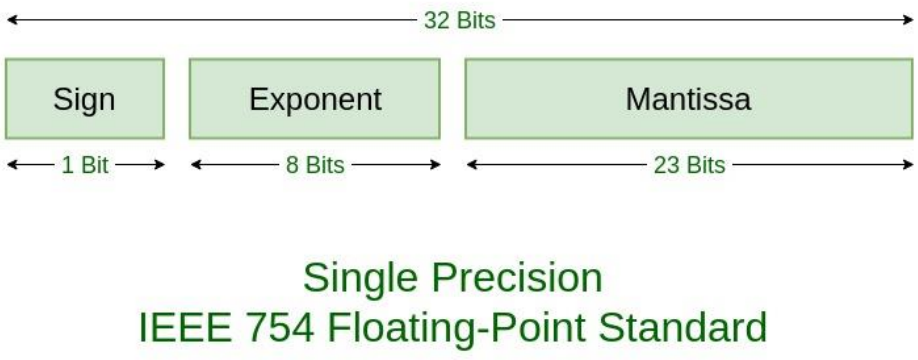

Fig 1 – Single Precision IEEE 754 Floating-Point Standard

- **The Sign of Mantissa –** This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
- **The Biased exponent –**The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
- **The Normalized Mantissa –** The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e., 0 and 1. So a normalized mantissa is one with only one 1 to the left of the decimal.

Special Values: IEEE has reserved some values that can ambiguity.

- **Zero** – Zero is a special value denoted with an exponent and mantissa of 0. -0 and +0 are distinct values, though they both are equal.
- **Denormalised** – If the exponent is all zeros, but the mantissa is not then the value is a denormalized number. This means this number does not have an assumed leading one before the binary point.
- **Infinity** – The values +infinity and -infinity are denoted with an exponent of all ones and a mantissa of all zeros. The sign bit distinguishes between negative infinity and positive infinity. Operations with infinite values are well defined in IEEE.
- **Not A Number (NAN)** – The value NAN is used to represent a value that is an error. This is represented when exponent field is all ones with a zero-sign bit or a mantissa that it not 1 followed by zeros. This is a special value that might be used to denote a variable that does not yet hold a value.

The range of positive floating-point numbers can be split into normalized numbers, and denormalized numbers which use only a portion of the fractions's precision. Since every floating-point number has a corresponding, negated value, the ranges above are symmetric around zero. There are five distinct numerical ranges that single-precision floating-point numbers are not able to represent with the scheme presented so far:

- Negative numbers less than – (2 – 2-23) × 2127 (negative overflow)
- Negative numbers greater than – 2-149 (negative underflow)
- Zero
- Positive numbers less than 2-149 (positive underflow)

- Positive numbers greater than (2 – 2-23) × 2127 (positive overflow)

Overflow generally means that values have grown too large to be represented. Underflow is a less serious problem because is just denotes a loss of precision, which is guaranteed to be closely approximated by zero.

3.3 FLOATING POINT ADDITION METHOD

To understand floating point addition, first we see addition of real numbers in decimal as same logic is applied in both cases. For example, we have to add 1.1 * 103 and 50.We cannot add these numbers directly. First, we need to align the exponent and then, we can add significant.

After aligning exponent, we get 50 = 0.05 * 103

Now adding significant, 0.05 + 1.1 = 1.15

So, finally we get (1.1 * 103 + 50) = 1.15 * 103

Here, notice that we shifted 50 and made it 0.05 to add these numbers.

Now let us take example of floating-point number addition.

We follow these steps to add two numbers -

1. Align the significant
2. Add the significant
3. Normalize the result

Let the two numbers be

x = 9.75

y = 0.5625

Converting them into 32-bit floating point representation -

9.75's representation in 32-bit format = 0 10000010 00111000000000000000000

0.5625's representation in 32-bit format = 0 01111110 00100000000000000000000

Now we get the difference of exponents to know how much shifting is required.

(10000010 – 01111110)2 = (4)10

Now, we shift the mantissa of lesser number right side by 4 units.

Mantissa of 0.5625 = 1.00100000000000000000000

(Note that 1 before decimal point is understood in 32-bit representation)

Shifting right by 4 units, we get 0.00010010000000000000000

Mantissa of 9.75 = 1. 00111000000000000000000

Adding mantissa of both,

0. 00010010000000000000000

+ 1. 00111000000000000000000

———————————————————-

1. 01001010000000000000000

In final answer, we take exponent of bigger number

So, final answer consists of -

Sign bit = 0

Exponent of bigger number = 10000010

Mantissa = 01001010000000000000000

32-bit representation of answer = x + y = 0 10000010 01001010000000000000000

3.4 FLOATING POINT MULTIPLICATION METHOD

Here, we have discussed an algorithm to multiply two floating point numbers, x and y.

Algorithm -

1. Convert these numbers in scientific notation, so that we can explicitly represent hidden 1.
2. Let ‘a’ be the exponent of x and ‘b’ be the exponent of y.
3. Assume resulting exponent c = a+b. It can be adjusted after the next step.
4. Multiply mantissa of x to mantissa of y. Call this result m.
5. If m does not have a single 1 left of radix point, then adjust radix point so it does, and adjust exponent c to compensate.
6. Add sign bits, mod 2, to get sign of resulting multiplication.
7. Convert back to one-byte floating point representation, truncating bits if needed.

Note

Negative values are simple to take care of in floating point multiplication. Treat sign bit as 1-bit unsigned binary, add mod 2. This is the same as XORing the sign bit.

Example

Suppose you want to multiply following two numbers –

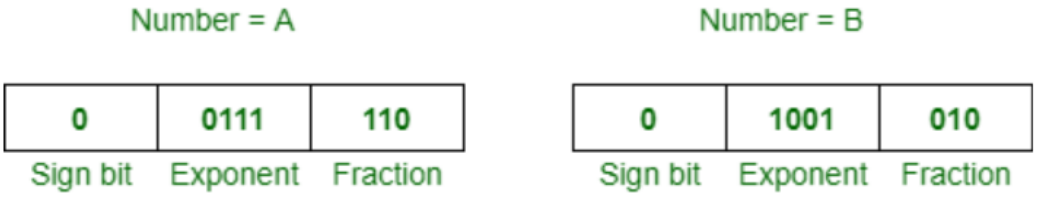

Fig 2 – Two floating point numbers to be multiplied

Now, these are steps according to above algorithm:

1. Given, A = 1.11 x 2^0 and B = 1.01 x 2^2
2. So, exponent c = a + b = 0 + 2 = 2 is the resulting exponent.
3. Now, multiply 1.11 by 1.01, so result will be 10.0011
4. We need to normalize 10.0011 to 1.00011 and adjust exponent 1 by 3 appropriately.
5. Resulting sign bit 0 (XOR) 0 = 0, means positive.
6. Now, truncate and normalize it 1.00011 x 2^3 to 1.000 x 2^3.

Therefore, resultant number is,

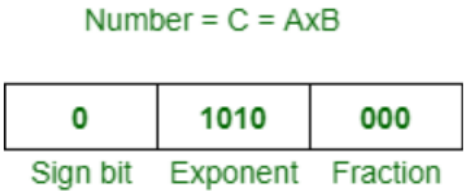

Fig 3 – Floating Point Multiplication Result

3.5 FMA METHOD

This method allows great throughput by combining two operations that would otherwise be separate and put them into the same pipeline, producing one result every clock cycle instead of having to store the final multiplication result before using it in an addition. The computation performed can be as described by the equation where A, B, C are all 32-bit floating-point numbers & result is also a 32-bit floating-point number.

$$\text{Result} = A * B + C$$

CHAPTER 4

FUSED-MULTIPLICATION ADDITION

4.1 INTRODUCTION

Once the standard IEEE 754 is discussed, now it is time to start with the implementation of the design of the respective modules. First, thinking about the different steps we should do to perform the operation required in the development of the codes and the design. In this section will talk about the procedure in addition and multiplication operations in flow chart and block diagram way to get the FMA by clubbing them together in a pipelined architecture.

4.2 FLOATING POINT ADDITION

The main goal of this chapter is the implement 32-bit Floating Point Adder with Verilog code. The steps to achieve this target have been explained here for the adder, it consists of two 32-bit input data bus and one 32-bit output. It gives addition result as an output in IEEE 754 format. Along with this, it will give four more exceptional signals namely overflow, underflow, and invalid.

4.2.1 ADDITION STEPS

Determination of operation to do (Addition/subtraction) can be logically find out. The first logical step is trying to specify what operations should be done to obtain a proper addition or subtraction.

The different steps are as follows:

1. Extracting signs, exponents and mantissas of both A and B numbers. The numbers format is as follows:

Figure 4: Floating Point Number Format

2. Operations the special cases:

• Operations with A or B equal to zero

• Operations with $\pm\infty$

• Operations with NaN

3. Finding out what type of numbers are given:

   • Normal

   • Subnormal

   • Mixed

4. Shifting the lower exponent number mantissa to the right $[Exp1 - Exp2]$ bits. Setting the output exponent as the highest exponent.
5. Working with the operation symbol and both signs to calculate the output sign and determine the operation to do.

Table 1: Sign Operation

A'Sign	Symbol	B'Sign	Operation
+	+	+	+
+	+	-	-
-	+	+	-
-	+	-	+

6. Addition/Subtraction of the numbers and detection of mantissa overflow (carry bit)

   If carry bit is encountered then shift the decimal left side and increase the exponent value to get a normalized result.
7. To standardizing mantissa, shifting it to the left up the first one will be at the first position and updating the value of the exponent according with the carry bit and the shifting over the mantissa.
8. Detecting exponent overflow or underflow (result NaN or $\pm\infty$). This is the

way forward to proper operation.

4.2.2 FLOW CHART OF ADDITION OPERATION

First the numbers should be treated (pre-adder) in order to perform the operation properly (adder) and finally, standardizing the result according with the standard IEEE 754 (standardizing).

Function of each block is given below

- **Pre-adder block**
  - Extracting the numbers to make the operations.
  - Identify the numbers as normal, special numbers.
  - Setting the exponents.
  - Shifting the mantissa.
- **Adder block**
  - Calculate sign of the operation
  - Making the operations (addition/subtraction).
- **Standardizing/normalization block**
  - Standardize the result according with IEEE754 standard.
  - Increase exponent in case of operation overflow.
  - Standardize mantissa.
  - Recalculating exponent.

4.2.3 ADDER BLOCK

Adder is the easiest part of the blocks. This block only implements the operation (addition or subtraction). It can be said the adder block is the ALU (Arithmetic Logic Unit) of the project because it oversees the arithmetic operations.

Two functions are implemented in this part of the code:

1. Obtaining the output's sign
2. Implementing the desired operation.

4.2.4 STANDARDIZING BLOCK

The main objective of this block is to normalize the operation output, as well as give in the proper IEEE-754 format. Finally, the Standardizing Block takes addition/subtraction and gives it an IEEE 754 format.

The procedure is as follows:

1. Shifting the mantissa to standardize the result.
2. The result of the calculating the new exponent according with the addition/subtraction overflow (carry out bit) and the displacement of the mantissa.

4.2.5 DESIGN OF SUB MODULES

As it has been said, in addition to normal and subnormal numbers, infinity, NaN and zero are represented in IEEE 754 standard. Some possible combinations have a direct result, for example, if a zero and a normal number are introduced the output will be the normal number directly. Time and resources are saved implementing this block. Both number A and number B are introduced as inputs.

- **1-Bit Carry Look Ahead Structure -** A Carry Look Ahead structure has been implemented. This structure allows a faster addition than other structures. The implementation of the Carry Look Ahead structure is shown at the figure 4.4. The idea is to obtain the carry generation and the carry propagation independently of each bit in order to obtain last carry faster.

4.2.6 ALGORITHM FLOATING POINT ADDITION

Given two numbers A and B represented in IEEE-754 single precision format this algorithm performs the addition of the two numbers and represents the result in IEEE-754 single precision format. Assume the exponent in B is less than or equal to the exponent in A. Let the exponent of B be b and let the exponent of A be a.

1. First, convert the two representations to scientific notation. Thus, the hidden one

is explicitly represented.

2. In order to add, the exponents of the two numbers need to be the same. The operand B is rewritten for this. This will result in B being not normalized, but value is equivalent to the normalized B. Add a - b to B's exponent. Shift the radix point of the mantissa (signficand) B left by a - b to compensate for the change in exponent.
3. Add the two mantissas of A and the adjusted B together.

4. If the sum in the previous step does not have a single bit of value 1, left of the radix point, and then adjust the radix point and exponent until it does.
5. Convert back to the one-byte floating point representation.
6. Stop.

4.2.7 RTL VIEW OF FLOATING-POINT ADDITION

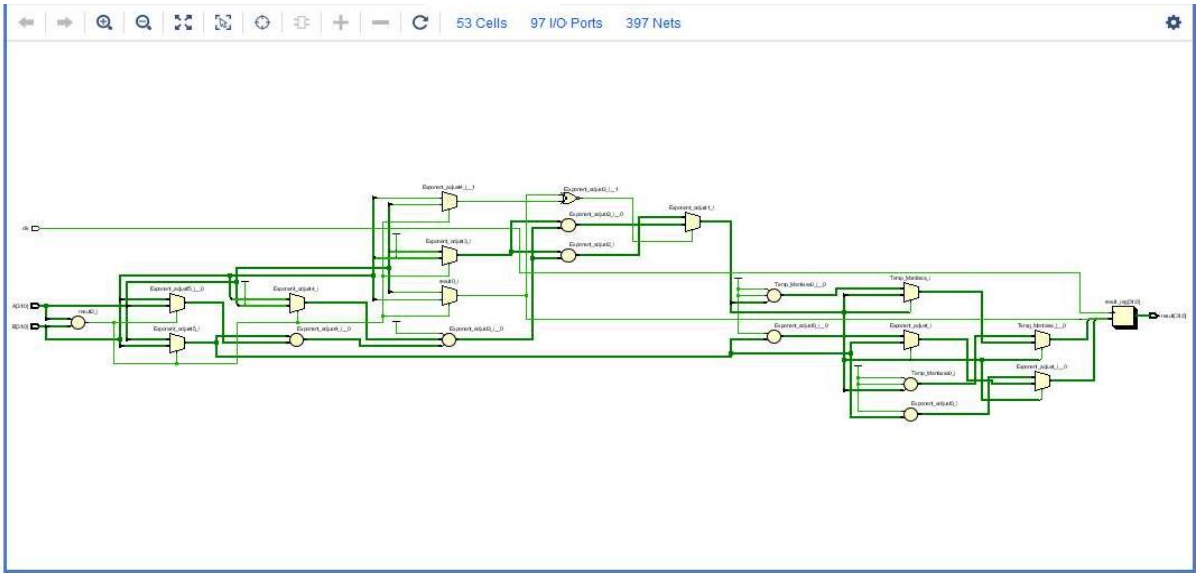

Fig 5 - Schematic of Addition

4.3 FLOATING POINT MULTIPLICATION

The figure 3 shows the flowchart of multiplication algorithm of multiplication is demonstrated by flowchart. In1 and in2 are two numbers sign1, expo1, S1 and sign2, expo2, S2 are sign bit, exponent and significand of in1 and in2 respectively.

Steps for multiplication are as follows -

1. calculate sign bit. sign_f= sign1 XOR sign2, sign_f is sign of final result.add the exponents and subtract 127 to make adjustment in exponent (expo1+127+expo2+127)-127.
2. Multiply the significand. S=S1*S2, check for overflow and underflow and special flag. When the value of biased exponent is less than 1 it shows occurrence of underflow, if exponent is greater than 254 then it shows overflow of floating-point operation.
3. Take the first 23 bits of 'S' and from left side and discard remaining bits.
4. Arrange the results in 32-bit format. 1 sign bit followed by eight bits exponent followed by 23-bits mantissa/significand.

Calculation of sign bit - when both numbers have same sign. The sign of result is positive else sign will be negative the sign in calculated by XORing both sign bits of inputs. The multiplication result is 48 bits. If 47th bit is '1' then right shift the result and add '1' in exponent to normalize the product. 46th to 23th bits are actual significand product. Exponent addition is done by unsigned 8-bit adder and to bias properly subtract 127 from the addition result for that purpose unsigned 8-bit subtractor is used. In any of the cases either addition of exponent in the beginning or while adjusting result the exponent must in the range 1 to 254. When overflow occur the result of multiplication goes to ± Infinity (+ or – sign is determined by the sign of two input numbers). When underflow occur, it makes underflow flag high and the result goes to ±0 (+ or – signed is determined by the sign of two input numbers).

4.3.1 FLOW CHART OF THE MULTIPLICATION OPERATION

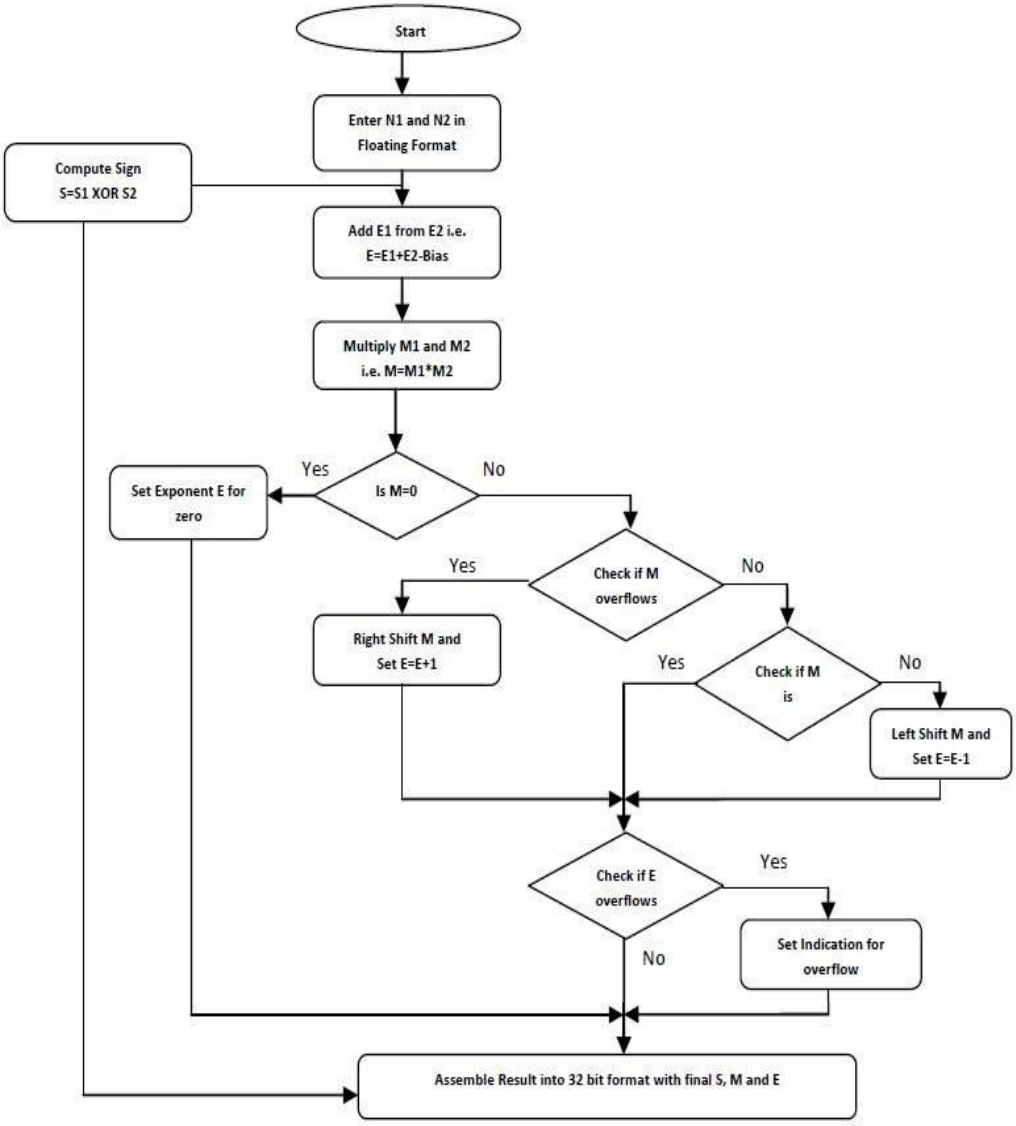

Fig 6 - Flow chart of the multiplication operation

4.3.2 RTL SCHEMATIC OF THE MULTIPLICATION OPERATION

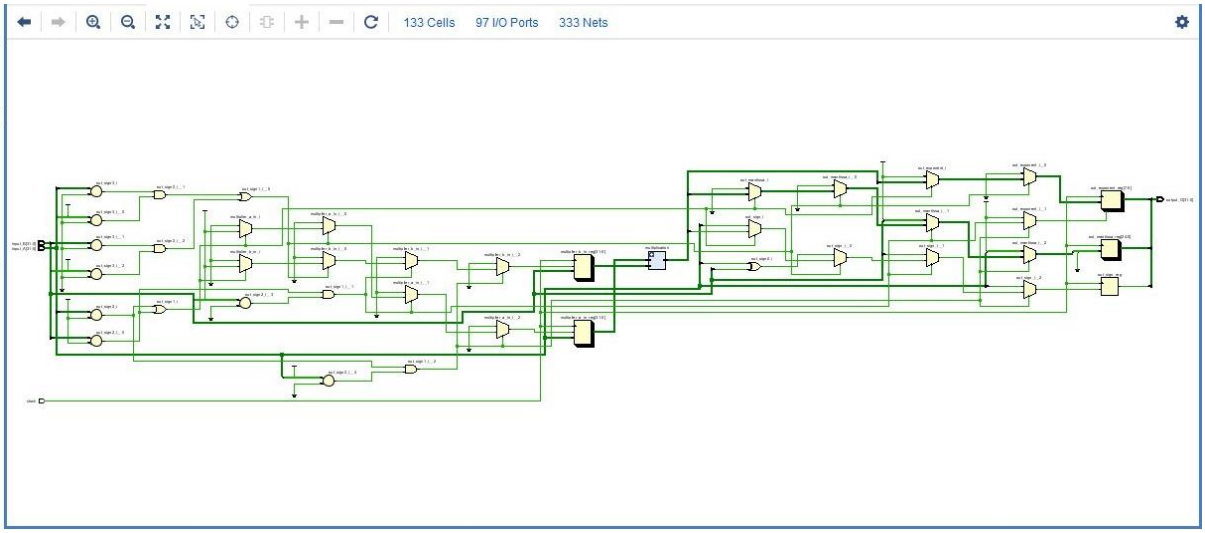

Fig 7 - RTL schematic of the multiplication operation

4.4 FMA

This method allows great throughput by combining two operations that would otherwise be separate and put them into the same pipeline, producing one result every clock cycle instead of having to store the final multiplication result before using it in an addition. The computation performed can be as described by the equation.

$$\text{Result} = (A * B) + C$$

Although a bit more complicated than a setup of simple multipliers and adders, these fused units can perform all operations involving addition and multiplication by simply setting the correct operand to 1 or 0 - addition is done by setting A or B to 1, and multiplication by setting C to 0. Aside from achieving higher throughput in suited applications, combining the two operations will usually also reduce the rounding error. This is a result of the multiplication product being fed directly into the adder without rounding the intermediate result first. One obvious cost of the FMA compared to using the simpler CMA architecture is increased complexity and area; the data path needs to be significantly wider for FMA - 72 bits for single-precision compared to 48 for a CMA implementation. For double-precision it is even more - 161 bits according to Schwarz. An FMA needs to align the multiplication product before performing the addition, and this could in the worst-case scenario require shifting it all the way from one end to the other. A more thorough introduction to FMAs

including some hints on how to optimize the design can be found in a paper by Eric M. Schwarz.

4.4.1 FMA RTL SCHEMATIC

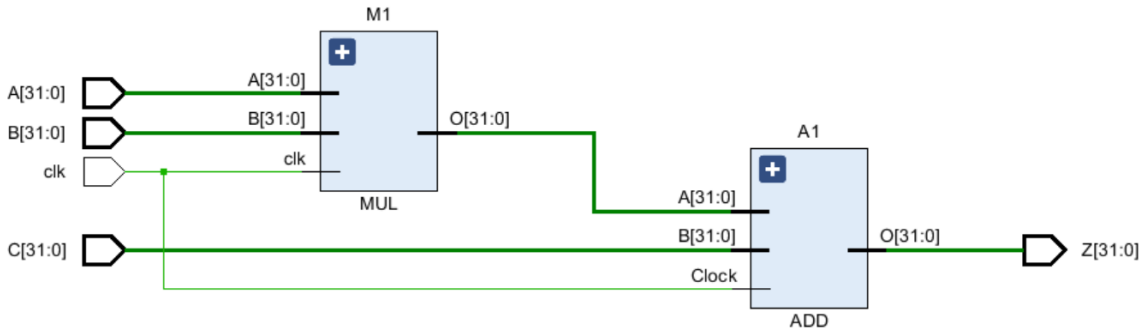

Fig 8 – FMA schematic

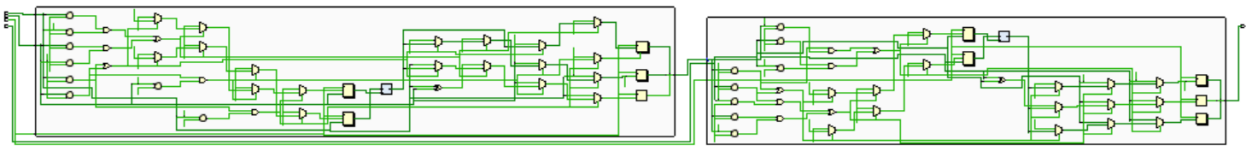

Fig 9 – FMA Elaborated schematic

CHAPTER 5

RESULT & ANALYSIS

5.1 INRODUCTION

Implemented algorithm on Xilinx Vivado and design is tested with different test cases, is run on Xilinx Vivado simulator of floating-point addition and floating-point division and result is listed below.

5.2 SIMULATION RESULT AND WAVEFORM

Simulation waveform output with various test cases is shown below.

Table 2: FMA output with various test cases

Input **A** (32-bit) (Decimal/ Hexadecimal)	Input **B** (32-bit) (Decimal/ Hexadecimal)	Input **C** (32-bit) (Decimal/ Hexadecimal)	Multiplication Output **OP** (32-bit) = (A*B) (Decimal /Hexadecimal)	Addition Output **Z** (32-bit) = (A*B) + C (Decimal/ Hexadecimal)
18.4576/ 4193A92A	6353.373/ 45C68AFB	290.570/ 439148F5	117268.0174848/ 47E50A00	117558.5874848/ 47E59B48
74.000/ 42940000	234.009/ 436A024D	45.9801 /4237EB9F	17316.666/ 46874954	17362.6461/ 4687A54A
59.36/ 426D70A3	35.345/ 420D6147	107.34/ 42D6AE14	2098.0792/ 45032144	2205.4192/ 4509D6B5
-45.89/ C2378F5C	80.076/ 42A026E9	-3.67/ C06AE147	-3674.6872/ C565AAFF	-3678.35717/ C565E5B7
-21.9/ C1AF3333	-49.31/ C2453D70	-234.981/ C36AFB22	1079.89/ 4486FC72	844.91/ 44533A1C

1. When A, B, C all the 32-bit inputs are positive.

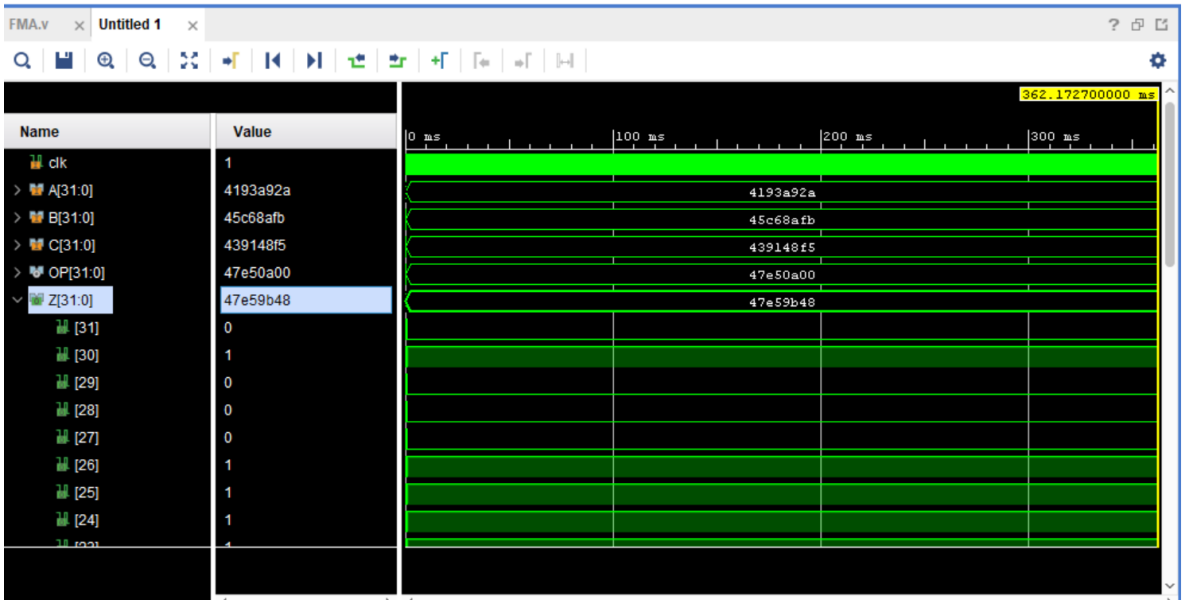

Fig 10 – FMA Simulation output case 1

2. When A, C are negative & B is positive 32-bit inputs.

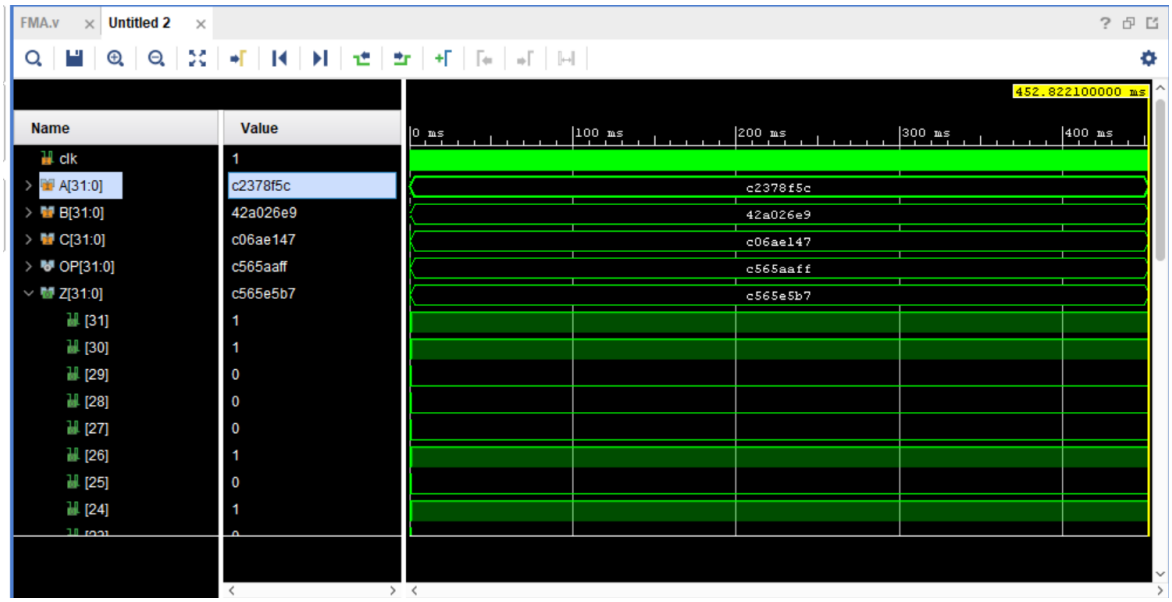

Fig 11 – FMA Simulation output case 2

3. When A, B, C are negative is positive 32-bit inputs.

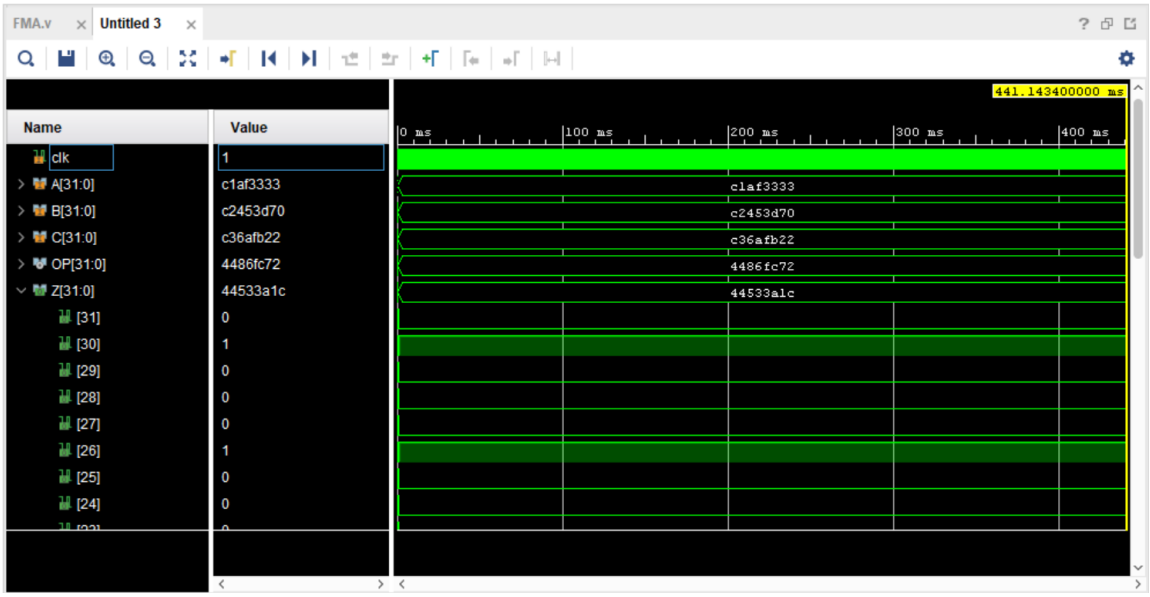

Fig 12 – FMA Simulation output case 3

5.3 RESOURCE UTILIZATION

FPGA Board device the FMA is implemented – **Artix 7**

The part of the device the FMA is implemented - **xa7a15tcsg324-2**

Resource	Utilization	Available	Utilization %
LUT	726	10400	6.98
FF	341	20800	1.64
DSP	2	45	4.44
IO	129	210	61.43

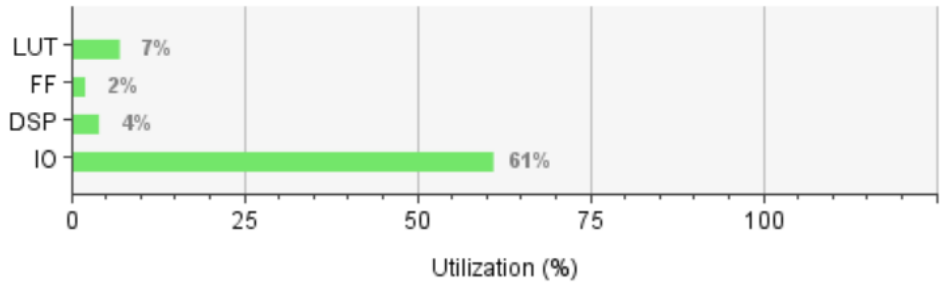

Fig 13 – FMA Resource Utilization Summary

5.4 CONCLUSION

The FMA IP is designed and are tested with all possible different combination of inputs, result of all inputs are verified and checked for correctness. Simulation waveform is generated with results tables. The utilization of resources is also analysis and compare with the existed Xilinx IP, proposed design showed improvement in utilization of resources.

CHAPTER 6

CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

The proposed algorithms are created using Verilog HDL and implemented on a floating-point coprocessor using an open-source RISC V ISA. The IEEE754-2008 standard is compatible with it. It supports the rounding mode and popular floating-point operations like addition, multiplication & FMA. With the exception of the inexact exception, it can handle any exception listed in the standard. Each module's correctness is evaluated using a wide range of input values and contrasted with the Xilinx Floating-Point Unit (FPU). Modern algorithms are implemented in each individual floating-point unit to maximize resource consumption and increase operating frequency.

6.2 FUTURE SCOPE

- With the right improved method, resources of addition, multiplication & FMA IPs can still be decreased.
- Double precision design is an option for FPUs, which improves results and broadens the scope of calculation.
- More floating-point modules, such as FMSUB, can be created and added to the FPU.
- Combine all the various IPs to create a reliable generic FPU and interface with RISC V.
- If the design's latency can be made better, operational frequency can also be increased.

REFERENCES

[1] "IEEE Standard for Floating-Point Arithmetic", in IEEE STD 754-2008, vol., no., pp.1-70, Aug. 292008.

[2] Jain, Jenil, and Rahul Agrawal. "Design and Development of Efficient Reversible Floating Point Arithmetic unit." In Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on, pp. 811-815. IEEE, 2015.

[3] Gopal, Lenin, Mohd Mahayadin, Syahira, Adib Kabir Chowdhury, Alpha Agape Gopalai, and Ashutosh Kumar Singh. "Design and synthesis of reversible arithmetic and Logic Unit (ALU)." In Computer, Communications, and Control Technology (I4CT), 2014 International Conference on, pp. 289-293.IEEE, 2014.

[4] Nachtigal, Michael, Himanshu Thapliyal, and Nagarajan Ranganathan. "Design of a reversible single precision floating point multiplier based on operand decomposition." In Nanotechnology (IEEE-NANO), 2010 10th IEEE Conference on, pp. 233-237. IEEE, 2010.

[5] Dhanabal, R., Sarat Kumar Sahoo, V. Bharathi, V. Bhavya, Patil Ashwini Chandrakant, and K. Sarannya. "Design of Reversible Logic Based ALU." In Proceedings of the International Conference on Soft Computing Systems, pp. 303-313. Springer India, 2016.

[6] Nachtigal, Michael, Himanshu Thapliyal, and Nagarajan Ranganathan. "Design of a reversible floating-point adder architecture." In Nano technology (IEEE-NANO), 2011 11th IEEE Conference on, pp. 451-456. IEEE, 2011.

[7] Alaghemand, Fatemeh, and Majid Haghparast. "Designing and Improvement of a New Reversible Floating Point Adder." (2015)

[8] Kahan, William."IEEEstandard754forbinaryfloatingpoint arithmetic. "Lecture Notes on the Status of IEEE 754.94720-1776 (1996):11.

[9] Ykuntam, Yamini Devi, MV Nageswara Rao, and G. R. Locharla. "Design of 32-bit Carry Select Adder with Reduced Area." International Journal of Computer Applications 75.2(2013).

[10] Quinnell, Eric, Earl E. Swartzlander Jr, and Carl Lemonds. "Floating-point fused multiply-add architectures." Signals, Systems and Computers, 2007 ACSSC 2007. Conference Record of the Forty- First Asilomar Conference on. IEEE, 2007

[11] Kahan, William."IEEE standard 754 for binary floating- point arithmetic."Lecture Notes on the Status of IEEE 754.94720-1776 (1996):11.

[12] Kodali, R.K.; Gundabathula, S.K.; Boppana, L., "FPGA implementation of IEEE-754 floating point Karatsuba multiplier," Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014 International Conference on , vol., no., pp.300,304, 10-11 July 2014.

[13] An Implementation of Single Precision Floating Point Multiplier Mamatha M1 S Pramod Kumar2. [14] Mamatha M1 S Pramod Kumar 2, Comparative Study on Performance of Single Precision Floating Point Multiplier using Vedic Multiplier and different types of Adders.

[15] Ms. Pallavi Ramteke*Dr. N. N. Mhala Prof. P. R. Lakhe, SINGLE PRECISION FLOATING POINT MULTIPLIER USING SHIFT AND ADD ALGORITHM.

[16] Aniruddha Kanhe ,Shishir Kumar Das, Ankit Kumar Singh, Design and Implementation of Floating Point Multiplier based on Vedic Multiplication Technique

[17] Arish S, R.K.Sharma, An efficient floating point multiplier design for high speed applications using Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm

[18] Sushma Wadar 1, Y. V. Chavan2, Sr. IEEE-Member, Avinash Patil 3, Improved Algorithm for Floating Point Multiplication

[19] K. Lavanya, K. Navatha, IEEE 754 Floating Point Multiplier using Carry Save Adder and Modified Booth Multiplier