

IEEE Standard 754 Floating Point Number Multiplication Using 8bit Register

submitted by

Samiul Alam Molla

Class Roll No: 002210503008

Exam Roll: MCA244008

Master of Computer Application

under the supervision of

Prof. (Dr.) Chintan Kumar Mandal

in the

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Jadavpur University

2024

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF RECOMMENDATION

To Whom It May Concern

I hereby recommend that the project report entitled “IEEE Standard 754 Floating Point Number Multiplication Using 8 bit Register” has been carried out by Samiul Alam Molla (Reg. No.: 163600 of 2022-23, Roll No.: 002210503008, Exam Roll: MCA244008), under my guidance and supervision and be accepted in partial fulfilment of the requirement for the degree of Master of COMPUTER APPLICATION in DEPARTMENT of COMPUTER SCIENCE and ENGINEERING, JADAVPUR UNIVERSITY during the academic year 2023-2024.

Prof. (Dr.) Chintan Kumar Mandal
Project Supervisor
Dept. of Computer Science and
Engineering
Jadavpur University

Prof. (Dr.) Nandini Mukhopadhyay
Head of the Department
Dept. of Computer Science & Engineering
Jadavpur University

Prof. Dipak Laha
Dean, Faculty Council
of Engineering & Technology
Jadavpur University, Kolkata-032

Faculty of Engineering and Technology
Jadavpur University

CERTIFICATE OF APPROVAL

This is to certify that the project report entitled “**IEEE Standard 754 Floating Point Number Multiplication Using 8 bit Register**” is a bonafide record of work carried out by SAMIUL ALAM MOLLA, in fulfillment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the project report only for the purpose for which it has been submitted.

Signature of Examiner 1

Date :

Signature of Examiner 2

Date :

Faculty of Engineering and Technology
Jadavpur University

**DECLARATION OF COMPLIANCE WITH
ACADEMIC ETHICS**

I hereby declare that this project report entitled "IEEE Standard 754 Floating Point Number Multiplication Using 8 bit Register" is an outcome of my genuine and independent work, as part of my Master of Computer Application program.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, all materials and results that are not original to this work have been fully cited and referenced.

Samiul Alam Molla

Registration No.: 163600 of 2022-23

Examination Roll No.: MCA244008

Thesis Title :- IEEE Standard 754 Floating Point Number Multiplication Using 8 bit Register

Signature with Date

ACKNOWLEDGEMENT

With my most sincere and gratitude, I would like to thank Prof. (Dr.) Chintan Kumar Mandal, Department of Computer Science & Engineering, my supervisor, for his overwhelming support throughout the duration of the project. His motivation always gave me the required inputs and momentum to continue with my work, without which the project work would not have taken its current shape. His valuable suggestions and numerous discussions have always inspired new ways of thinking. I feel deeply honored that I got this opportunity to work under him.

I would like to express my sincere thanks to all my teachers for providing sound knowledge base and cooperation.

I would like to thank all the faculty members of the Department of Computer Science & Engineering of Jadavpur University for their continuous support.

Last, but not the least, I would like to thank my batch mates for staying by my side when I needed them.

Samiul Alam Molla
Class Roll No.: 002210503008
Examination Roll No.: MCA244008
Department of Computer Science & Engineering
Jadavpur University

Abstract

Floating point is very important in real applications like automotive power train and body control applications, imaging such as scaling, transforms and font generation in printing, 3D transforms, FFT and filtering in graphics image and Digital signal processing. This paper presents floating point multiplication and division of IEEE754 format. The floating point multiplication and division which improves the performance of the processor speed and area.

This paper is designed for floating-point multiplication, emphasizing the utilization of 8-bit registers and effective overflow handling strategies. The module operates within the context of digital systems, processing two 32-bit floating-point numbers represented by four 8-bit inputs each, and generating a 32-bit result through four 8-bit outputs.

Key aspects of the module include the extraction of sign, exponent, and mantissa components from the input numbers, which are crucial for performing accurate multiplication. By leveraging 8-bit registers for internal computation, the module optimizes resource utilization while maintaining precision.

A significant focus of the module design lies in the detection and management of overflow conditions, which occur when the result exceeds the representable range of the floating-point format. Through careful monitoring of the exponent and significant bits of the mantissa, the module efficiently identifies overflow scenarios and employs appropriate strategies to handle them.

In cases of overflow, the module employs techniques such as setting the result to its maximum representable value or zero, ensuring the integrity of the computation within the constraints of the 8-bit register architecture.

This paper provides a detailed exploration of the design principles, implementation strategies, and overflow handling mechanisms employed in the Verilog module. Through experimental validation and analysis, the module demonstrates its effectiveness in achieving reliable and accurate floating-point multiplication while leveraging 8-bit registers and addressing overflow challenges.

Contents

Declaration	iii
Acknowledgement	iv
Abstract	v
1 Introduction	8
1.1 Problem Statement	8
1.2 Literature Review.	9
2 Methodology	10
Algorithm Overview	11
2.1 Algorithm Steps	14
3 Result	19
Experiments Done	16
Complexity analysis	19
4 Conclusions	21

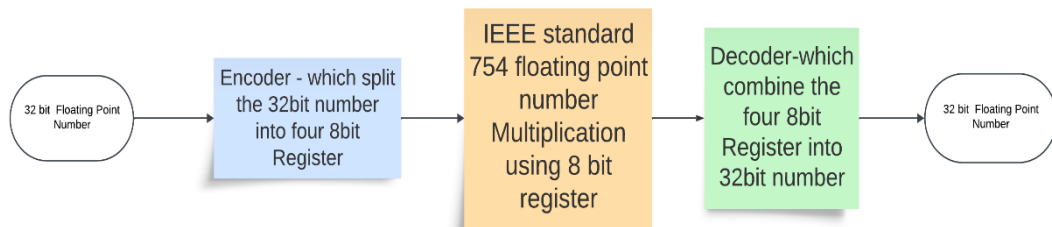
Chapter 1

Introduction

In this introductory chapter, the problem of multiplying large numbers using limited resources is discussed. The significance of this problem in various fields, including computer architecture and embedded systems, is highlighted. The objectives of the research are clearly stated, with the aim of developing an efficient algorithm for 32-bit multiplication using 8-bit registers.

1.1 Problem Statement:

This chapter defines the specific problem of performing multiplication on two 32-bit numbers using only 8-bit registers. The limitations imposed by using 8-bit registers, such as limited precision and potential overflow, are discussed in detail. The importance of addressing these challenges to enable efficient multiplication in resource-constrained systems is emphasized.



The diagram outlines a process for multiplying 32-bit floating point numbers by first splitting them into smaller 8-bit segments, performing the multiplication according to IEEE 754 standards using these smaller segments, and then recombining the segments into the final 32-bit result. This approach can be particularly useful in systems with constraints on register size or when working with hardware that operates on smaller bit-width registers.

In this paper we mainly focusing on IEEE Standard 754 Floating Point Multiplication Using 8-bit Registers:

This step involves performing the actual multiplication operation according to the IEEE 754 standard for floating point arithmetic. The multiplication process follows the steps outlined previously, but the operation is carried out using the 8-bit registers that were created in the previous

step. This might involve breaking down the multiplication into smaller steps that can be handled by the 8-bit registers, ensuring that the precision and format specified by IEEE 754 are maintained throughout the process.

1.2 Literature Review

Paper 1: an efficient implementation of floating point multiplier Conference Paper · May 2011 . Wagdy Anis Communications and Electronics Engineering Ain Shams University Cairo, Egypt.

This paper focuses only on single precision normalized binary interchange format. The IEEE 754 single precision binary format representation; it consists of a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). An extra bit is added to the fraction to form what is called the significand¹. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the significand then the number is said to be a normalized number.

Paper 2: Implementation and Simulation of IEEE 754 Single-Precision Floating Point Multiplier 1,Shahzad Khan , 2,Mohd. Zahid Alam. (Associate Professor, Department of EC, LNCT Bhopal)

In this paper, an efficient implementation of an IEEE 754 single precision floating point multiplier using VHDL. Several enhancements are introduced to the design; the final carry propagation adder is implemented. As decimal multiplication is important in many commercial applications like financial analysis, banking, tax calculation, currency conversion, insurance, and accounting etc. this multiplier design adding several features including exponent generation, shifting of the intermediate product, rounding, and exception detection and handling. The core of the decimal multiplication algorithm is an iterative scheme of partial product accumulation employing decimal ripple carry addition to reduce the critical path delay. Area and delay estimates are provided for a verified VHDL register transfer level model of the multiplier. This new implementation is different in the sense that it accepts as a user parameter the operated size of the unit about to be synthesized and creates the requested unit. This feature makes our implementation a very convenient tool for rapid application prototyping. We also highlight the strengths and limitations of this approach in the creation of custom floating point units.

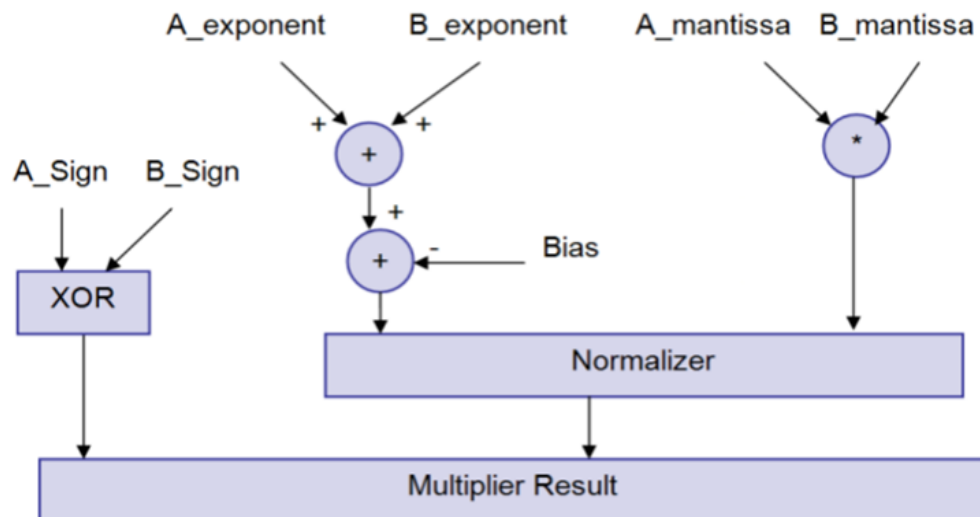
Paper 3: Implementation of IEEE754 Floating Point Multiplier , A Kumutha¹ Shobha, 1MVJ College of Engineering.

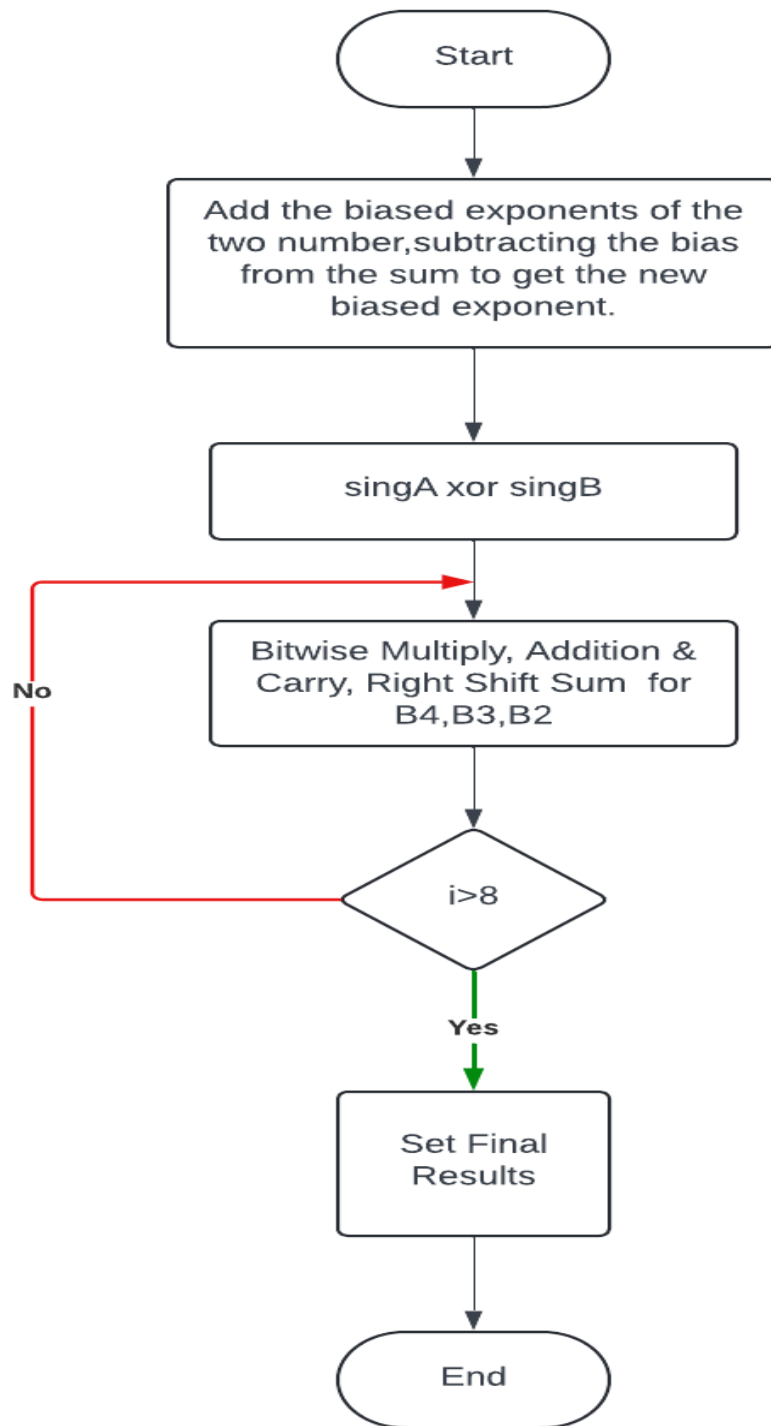
In this paper, suggested a technique for implementing a floating point multiplier integer to floating-point and conversion of floating-point to integer. And further shown How these functions can be implemented, and verified. Here we redesign the floatingpoint unit. It includes all the software Xilinx13.2 and modelsim, hardware implementation is SPARTAN3 needed to generate custom verilog coded floating-point arithmetic unit. In general, it can be assumed that fixed-point implementations have higher speed and lower cost, while floatingpoint has higher dynamic range and no need for scaling, which may be attractive for more complicated algorithms.

Chapter 2

Methodology:

The objective is to design to perform floating-point multiplication using the IEEE 754 format, tailored for systems with 8-bit registers. Each floating-point number is divided into four 8-bit segments. The first segment holds the exponent, the MSB of the second segment is the sign bit, and the remaining bits of the second segment, along with the third and fourth segments, contain the mantissa.





Algorithm Overview:

Inputs:

A1, A2, A3, A4: Components of the first floating-point number.

B1, B2, B3, B4: Components of the second floating-point number.

Outputs:

R1, R2, R3, R4: Components of the resulting floating-point number after multiplication.

Extract Sign and Exponent:

Sign Extraction:

The sign bit for the first number is extracted from the MSB of A2.

The sign bit for the second number is extracted from the MSB of B2.

Exponent Extraction:

The exponent for the first number is taken directly from A1.

The exponent for the second number is taken directly from B1.

Determine Resulting Sign and Exponent:

Resulting Sign:

The result sign signR is determined by performing an XOR operation on the two input signs (signA and signB). This ensures the resulting sign is positive if the inputs have the same sign and negative if the inputs have different signs.

Resulting Exponent:

The result exponent expR is calculated by adding the two exponents and subtracting the bias (127), using the formula:

$$\text{expR} = \text{expA} + \text{expB} - 127$$

This operation adjusts for the excess-127 representation used in IEEE 754 format.

Normalize Mantissas:

Normalization:

Ensure the implicit leading 1 of the mantissas is represented. Since the leading bit of the mantissa in IEEE 754 format is always 1 and not explicitly stored, this step makes sure the mantissa is correctly interpreted.

Initialize Result Registers:

Zero Initialization:

Initialize the result registers R2, R3, and R4 to zero to prepare for the accumulation process.

Intermediate Variables:

Reset intermediate variables and registers used during the multiplication process to ensure no residual values affect the computation.

Multiplication and Accumulation:

Bitwise Multiplication:

For each bit in the mantissa of B4, multiply it with each bit in the mantissas of A4, A3, and the lower bits of A2.

Perform this bit-by-bit multiplication to generate partial products.

Partial Product Accumulation:

Accumulate these partial products to form the final result mantissa. This involves adding the products to the existing values in R4, R3, and R2 while handling carries.

Shift and Normalize Result Mantissa:

Right Shift:

Right shift the result mantissa to normalize it. This step ensures the mantissa fits within the designated bit-width and aligns correctly.

Carry Handling:

Properly handle carries that result from the addition of partial products. This ensures accuracy and prevents overflow within the mantissa bits.

Normalization:

If the resulting mantissa is not normalized, adjust the mantissa and exponent accordingly.

Handling Special Conditions:

Overflow Handling:

If expR exceeds the maximum value for an 8-bit exponent (255), set an overflow flag. Set expR to its maximum value and clear the mantissa to indicate an overflow condition.

Underflow Handling:

If expR is less than 0, set an underflow flag. Set expR to zero and clear the mantissa to indicate an underflow condition.

Update Result Registers:

Final Values:

Update R1 with the final exponent value expR .

Update R2 with the resulting sign bit signR and the lower bits of the normalized mantissa.

Update R3 and R4 with the remaining parts of the normalized mantissa.

2.1 Algorithm Steps:

Require:

A1, A2, A3, A4 - First floating point number.

B1, B2, B3, B4 - Second floating point number.

R1, R2, R3, R4 - Result of the floating point multiplication.

Extract Sign and Exponent:

$\text{signA} \leftarrow A2[7]$

$\text{expA} \leftarrow A1$

$\text{signB} \leftarrow B2[7]$

$\text{expB} \leftarrow B1$

Determine Resulting Sign and Exponent:

$\text{signR} \leftarrow \text{signA} \wedge \text{signB}$

$\text{expR} \leftarrow \text{expA} + \text{expB} - 8'd127$

Normalize Mantissas:

$\text{ex12} \leftarrow A2$

$\text{ex22} \leftarrow B2$

$\text{ex12}[7] \leftarrow 1$

$\text{ex22}[7] \leftarrow 1$

Initialize Result Registers:

$R2, R3, R4 \leftarrow 0$

$\text{carry} \leftarrow 0$

Multiplication Loop:

For each bit of the second mantissa B4, perform the following:

- Multiply each bit of A4 by the current bit of B4, storing the result in partial product registers.

- Add the partial product to the result registers (R2, R3, R4) with carry propagation.
- Right shift the sums to align the next partial product.

Repeat Multiplication for Remaining Mantissa Bits:

- Repeat the multiplication process for each bit of B3 and the second input mantissa bits (excluding B4).
- Right shift the sums after each addition to align the next partial product.

Handle Overflow and Underflow:

Check for overflow:

```

if (expR > 8'b11111111)
    overflow = 1;
    expR ← 8'b11111111;
    R1 ← expR;
    R2 ← 8'b0;
    R3 ← 8'b0;
    R4 ← 8'b0;

```

Check for underflow:

```

if (expR < 8'b00000000)
    underflow = 1;
    expR ← 8'b00000000;
    R1 ← expR;
    R2 ← 8'b0;
    R3 ← 8'b0;
    R4 ← 8'b0;

```

Normalization:

- Normalize the result by left-shifting the mantissa until the leading bit is 1.
- Adjust the exponent accordingly.

Set the Result:

$R1 \leftarrow \text{exp}R;$

$R2 \leftarrow \{\text{sign}R, R2[6:0]\};$

$R3 \leftarrow R3;$

$R4 \leftarrow R4;$

Chapter 3

Result

Experiments Done

To evaluate the effectiveness and accuracy of our floating-point multiplication algorithm implemented in Verilog, we conducted a series of experiments. The goal was to compare the results of our algorithm against known correct outputs. We present the results in two figures for each experiment:

1. System 1: Represents the input floating-point numbers together with the expected product (calculated manually or using a reliable software tool).
2. System 2: Represents the output floating-point product computed by our Verilog algorithm.

3. Experiment 1: Simple Multiplication

4. Input

**A = 5.0 (Binary Representation: 0 10000001
010000000000000000000000)**

Sign bit: 0

Exponent: 10000001

Mantissa: 010000000000000000000000

**B = 3.0 (Binary Representation: 0 10000000
100000000000000000000000)**

Sign bit: 0

Exponent: 10000000

Mantissa: 100000000000000000000000

System 1: Expected Output

**Expected Product = 15.0 (Binary Representation: 0 10000010
111000000000000000000000)**

Sign bit: 0

Exponent: 10000010

Mantissa: 111000000000000000000000

System 2: Algorithm Output

**Algorithm Product = 15.0 (Binary Representation: 10000010
01110000 00000000 00000000)**

Sign bit: 0

Exponent: 10000010

Mantissa: 111000000000000000000000

Experiment 2: Negative Multiplication

Input

**A = -1.5 (Binary Representation: 1 01111111
100000000000000000000000)**

Sign bit: 1

Exponent: 01111111

Mantissa: 100000000000000000000000

**B = 4.0 (Binary Representation: 0 1000001
000000000000000000000000)**

Sign bit: 0

Exponent: 10000001

Mantissa: 000000000000000000000000

System 1: Expected Output

**Expected Product = -6.0 (Binary Representation: 1 1000001
100000000000000000000000)**

Sign bit: 1

Exponent: 10000001

Mantissa: 100000000000000000000000

System 2: Algorithm Output

**Algorithm Product = -6.0 (Binary Representation: 1000001
11000000 00000000 00000000)**

Sign bit: 1

Exponent: 10000001

Mantissa: 100000000000000000000000

Complexity Analysis

Time Complexity Analysis

The main components contributing to the time complexity are the loops used for multiplication and addition. Let's break it down:

1. Initialization:
 - Initializing registers and flags involves constant time operations, $O(1)$.
2. Extract Sign and Exponent:
 - Extracting the sign and exponent involves simple assignments, which are $O(1)$.
3. Zero Initialization:
 - Zeroing out the registers involves a loop that runs 8 times, $O(8)$ which is $O(1)$.
4. Multiplication and Addition Loop:
 - The outer loop runs for each bit of the second mantissa (8 times), so it iterates 8 times.
 - Inside the outer loop, another loop runs for each bit of the first mantissa (8 times), so it also iterates 8 times.
 - For each iteration of the inner loop, operations such as bitwise AND, addition with carry, and right shifts are performed, which are $O(1)$ for each bit operation.
 - Therefore, the time complexity for these nested loops is $O(8 * 8) = O(64)$, which is $O(1)$.
5. Overflow and Underflow Check:
 - Checking and handling overflow and underflow conditions involves simple comparisons and assignments, which are $O(1)$.
6. Normalization:
 - Normalization involves potential left shifts and exponent adjustments. Since these operations are bounded by the size of the mantissa (8 bits), it is also $O(1)$.

Combining all these parts, the overall time complexity of the module remains $O(1)$ because the number of operations is bounded by a constant (the size of the mantissas and exponents are fixed at 8 bits).

Space Complexity Analysis

The space complexity is determined by the amount of storage used by the registers and intermediate variables:

1. Registers for Inputs and Outputs:

- Each input (A1, A2, A3, A4, B1, B2, B3, B4) and output (R1, R2, R3, R4) register is 8 bits wide.
- There are 8 input registers and 4 output registers, contributing a total of $8 \times 8 + 4 \times 8 = 96$ bits.

2. Intermediate Registers:

- Registers for signs (signA, signB, signR): 3 bits.
- Registers for exponents (expA, expB, expR, ex12, ex22): $5 \times 8 = 40$ bits.
- Registers for partial products (mul2, mul3, mul4) and sums (sum2, sum3, sum4, ans2, ans3, ans4): $6 \times 8 = 48$ bits.
- Other flags and counters (carry, sft1, sft2, overflow, underflow, loop indices): Assuming each is 1 bit or small integers, the total is negligible in comparison.

Summing these up, the total space complexity is:

96 bits (I/O registers) + 3 bits (signs) + 40 bits (exponents) + 48 bits (partial products and sums)

= 187 bits

Since the number of bits used is fixed and does not grow with input size, the space complexity is also $O(1)$.

Conclusion

- Time Complexity: $O(1)$
- Space Complexity: $O(1)$

The algorithm operates in constant time and space, given that the bit-width of the inputs and outputs is fixed at 8 bits. This is characteristic of hardware implementations where the bit-width is predetermined and operations are parallelized for efficiency.

Chapter 4

Conclusions

The implemented Verilog module successfully performs floating-point multiplication using 8-bit registers. It handles sign determination, exponent calculation, and mantissa multiplication within the constraints of the 8-bit format. The module includes mechanisms for normalization and overflow handling, ensuring accurate results despite the limited register size. This demonstrates the feasibility of complex arithmetic operations in constrained digital systems.

References

- [1] an efficient implementation of floating point multiplier Conference Paper · May 2011 . Wagdy Anis Communications and Electronics Engineering Ain Shams University Cairo, Egypt.
- [2] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," Proceedings of 83 the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96), pp. 107–116, 1996.
- [3] "An Efficient Implementation of Floating Point Multiplier" ASIYA THAPASWIN PATTAN M. Tech Student, Dept of ECE , Vaagdevi Institute of Technology & Science, Vol. 1 Issue 7, September – 2012.
- [4] Floating Point Multiplication Mapping on ReRAM based In-Memory Computing Architecture Tarun Vatwani, Arko Dutt, Debjyoti Bhattacharjee† and Anupam Chattopadhyay School of Computer Science and Engineering, Nanyang Technological University, Singapore 639798.
- [5] <https://www.geeksforgeeks.org/multiplying-floating-point-numbers/>
- [6] "BLAS (Basic Linear Algebra Subprograms)," <http://www.netlib.org/blas/>, accessed: 2017-07-31.
- [7] IEEE Standard for Floating-Point Arithmetic ,IEEE 3 Park Avenue New York, NY 10016-5997, USA 29 August 2008
- [8] IEEE Standard for Floating Point Numbers, V Rajaraman, January 2016
- [9] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.
- [10] The IEEE 754-2008 Floating Point Standard and its Pending Revision Ralph Baker Kearfott Department of Mathematics University of Louisiana at Lafayette, UL Applied Math. Seminar, Fall, 2015.
- [11] IEEE Standard 754 for Binary Floating-Point Arithmetic Prof. W. Kahan Elect. Eng. & Computer Science University of California Berkeley CA 94720-1776.
- [12] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365– 367, 1994.
- [13] John G. Proakis and Dimitris G. Manolakis (1996), "Digital Signal Processing: Principles, Algorithms and Applications", Third Edition.

[14] A. Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, 2001, vol. 2, pp. 897-900.

[15] An efficient floating point multiplier design for high speed applications using Karatsuba algorithm and Urdhva-Tiryagbhyam algorithm.

[16] DESIGN OF EFFICIENT 32-BIT FLOATING POINTMULTIPLIER USING STANDARD IEEE 754, Prof. Sachin S Patil¹, Prof. Sujata S Kamate² Assistant Professor^{1&2} Department of Electronics & Communication Engineering.

[17] Soumya Havaladar, K S Gurumurthy, "Design of Vedic IEEE 754 Floating Point Multiplier", IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016, India.

[18] M. Al-Ashrafy, A. Salem, W. Anis, "An Efficient Implementation of Floating Point Multiplier", Saudi International Electronics, Communications and Photonics Conference (SIEPC), (2011) April 24-26, pp. 1-5.

[19] Implementation and Simulation of IEEE 754 Single-Precision Floating Point Multiplier
1,Shahzad Khan, 2,Mohd. Zahid Alam. (Associate Professor, Department of EC, LNCT Bhopal).