ISSN: 2583-3286(Online)

IPEC Journal of Science & Technology
Vol. 01 (01), Feb. 2022, pp. 11-16

Single Precision Floating Point Unit (FPU) Based on IEEE 754 Standard Using Verilog

Uddeshya Kumar, Vikas Chauhan, Surya Kant Choudhary, Shubham Jain and Shafali Jagga

Department of Electronics and Communication Engineering
Inderprastha Engineering College, Ghaziabad, India

Accepted on 24 February 2022

Abstract: This research paper depicts single precision floating point operations such as addition, subtraction, multiplication, and division using hardware description language Verilog. Verilog is based on IEEE 754 standard initiated by Institute of Electrical and Electronics Engineers (IEEE) in 1985 [1]. The main aim is to build an efficient logic with reduced complexity. The functions performed for managing the floating point data are normalizing the data, converting data to IEEE - 754 format and performing mathematical operations like addition, subtraction, multiplication and division using single precision Floating Point Unit (FPU) [2]. All operations are made up of potentially efficient algorithms that may be modified to improve overall latency, and once pipelined, the output is high. Simulation and synthesis has been performed using Xilinx ISE 14.7 environment.

***Keywords* - Floating Point Unit, Overflow, Single Precision, Underflow**

.

I. INTRODUCTION

The real valued numerals can be expressed in the form of floating point numbers which can be both, a single precision and a double precision floating point numbers i.e., either it can be of 32- bits or of 64- bits. IEEE 754 standard is used as the regulated approach to portray floating point numbers. FPU has been evolved to execute arithmetic operations such as add, subtract, multiply and divide. The floating point number has three fields of representations given as:

A. Sign ($\pm$): Sign acquires the 31st bit of a single precision floating point number. It consists of a one bit data which decides the positivity or the negativity of any 32- bit floating point number i.e.

a. "0": for +ve numbers
b. "1": for -ve numbers

B. Mantissa(M): Mantissa must be from 0 to 22nd bits of a single precision floating point number. It must have 23-bits of data, if lesser then 23- bits, then zeroes can be appended on the empty bit places. These are the bits written just after the decimal point.

C. Exponent(E): The data of exponent must be from 23rd to 30th bits of a single precision floating point number. It must be of 8- bits signed exponent with excess of 127 bias, if lesser then zeroes can be filled to the left end of the number. Table 1 shows representation of single precision or double precision with 32 bits and 64 bits rrespectively as per IEEE 754 standard [3].

Table 1: Standard representation in terms of bits for single precision floating-point data[4]

	Sign	**Exponent**	**Mantissa**	**Bias**
Single Precision	1-bit (31st)	8-bits (23rd-30th)	23-bits (00-22nd)	127

Four exceptionalconditions may occur during or in the process of various operations, these are elaborated as:

a) **Overflow:** Overflow occurs when the result exceeds the limit and cannot be represented within the range of precision format.
b) **Underflow: U**nderflow condition is raised when the result is small enough in size to be calculated, or less to get normalized**,**
c) **Division by zero:** Divide by zero occurs when a non-zero number is divided by zero.
d) **Invalid operation:** If the operands are not valid then invalid operation condition occurs.

II. RESEARCH METHODOLOGY

Block diagram of a single precision FPU is shown in figure 1. Table 3 illustrates 2-bit signal control value for selecting various arithmetic operations. Result is obtained by

[1] Date of Submission: 11 Jan 2022
Corresponding Author: Shafali (e-mail: shafali.jagga@ipec.org.in)

Published online : 25 February 2022

performing pre-normalization, then corresponding operation is performed as per the given signal select value and finally post-normalization is done. After this, the condition for exception is checked and removed using exceptional handling [5]. The flow charts in figures 3 and 5 explain how all four fundamental arithmetic operations are handled, including overflow, underflow, rounding, and different exception scenarios.

TABLE 3: OPERATIONS & FUNCTIONS

OPERATION	FUNCTION	SIGNAL(Sel)
Addition	Performs addition	2’b00
Subtraction	Performs subtraction	2’b01
Multiplication	Performs multiplication and finally post normalization	2’b10
Division	Performs division, determines quotient and finally post normalized of the result.	2’b11

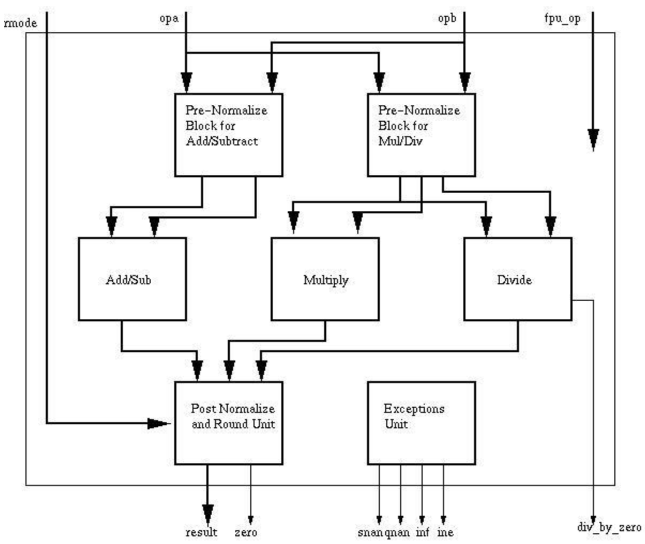

Figure 1: Block diagram of floating point arithmetic unit

Main modules of FPU are:

I. ADDITION MODULE:

Algorithm and flow chart for addition module is shown in figure 2 and 3.

Steps for executing operations are:

1. Compare the exponent’s magnitudes of the number on which the operation is to be performed.
2. Make appropriate alignment to the number with the lesser exponent magnitude until both exponents are equal.
3. Now add the mantissas of the numbers together.
4. By shifting the final resulting mantissa and adjusting the resulting exponent normalization can be performed.

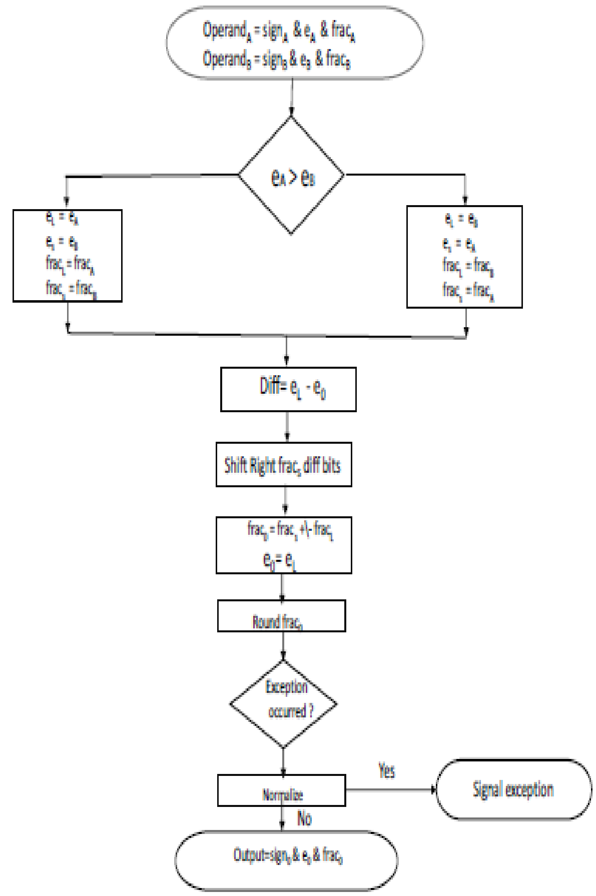

Figure 2: Algorithm for addition/subtraction [6] [8]

Algorithm:

Step 1: Determine if operation = 4‘‘b0000

Step 2: {C, Sum} = fracL[22:0] + fracr[22:0]

Step 3: If carry comes out to be 1, then

Resultant exponent = Larger exponent + 1 In other case if carry comes out to be 0, then
Resultant exponent = Larger exponent – (21- difference)

Step 4: Evaluate for exception conditions underflow and overflow.

For any of the operands condition is checked, if it meets the requirement i.e. (sign(operand with greater exponent) = = 0 & (exp_greater + 1 > 255)) then, the overflow flag is set to 1.

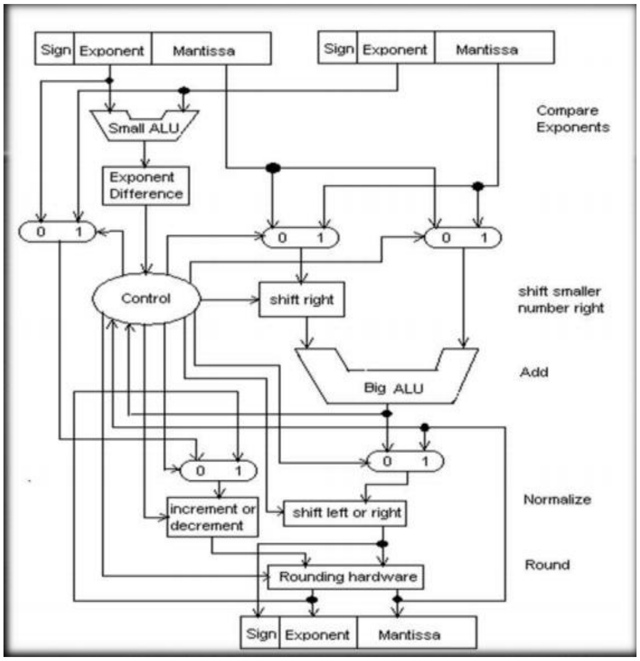

Figure 3: Flow chart for addition/subtraction module [7]

It can easily be understood with an example as follows:

Example: Take two 5-digits binary FP numbers:

$2^4 \times 1.1001 + 2^2 \times 1.0010$

a) Get the number with the larger exponent and subtract it from the smaller exponent. Let the two numbers are:

eL & eS are given as : = 2^4 *and* 2^2
d equals 4 – 2 i.e. 2

b) Shift the fraction with the smaller exponent difference positions to the right. We can leave out the exponent since they are both equal.
1.1001000 + 0.0100100

c) Add both fractions:
1.1001000 + 0.0100100 = 1.1101100

d) It is rounded to nearest even number as: 1.1110

e) Result comes out to be $2^4 \times 1.1110$

II. SUBTRACTION MODULE

Subtraction can be understood as adding one negative number and one positive number. Same algorithm as that of addition can be applied to subtraction too. Subtraction can be performed by taking one's complement of subtrahend and adding one to it. Its equivalent to taking two's complement of the negative number. Doing this interpretation, the negative number is made as positive and further addition operation is carried out. The subtraction operation is similar to the addition process i.e., by comparing exponents of the two operand and shifting until the exponent becomes equal, adding of mantissas andnormalizing.

Algorithm:

<u>Step</u> 1: Determine if oper comes out to be 4‟b0001

<u>Step</u> 2: Borrow and difference is given as {borrow.diff} = Temp_op1_ieee[22:0] - Temp_op2_ieee[22:0]

<u>Step</u> 3: Resultant_exponent = Larger_exponent + (21-difference)

<u>Step</u> 4: Finally evaluate for exception conditions underflow and overflow.

For any of the operands overflow condition is checked, if it meets the condition overflow is set to 1. It can be expreseed as: (sign (operand with greater exponent) == 1& (exp_greater + 1 < 0)) the overflow flag is set to 1. Code has been written in Verilog and synthesized register transfer logic (RTL) view of addition/subtraction is shown in Figure 4.

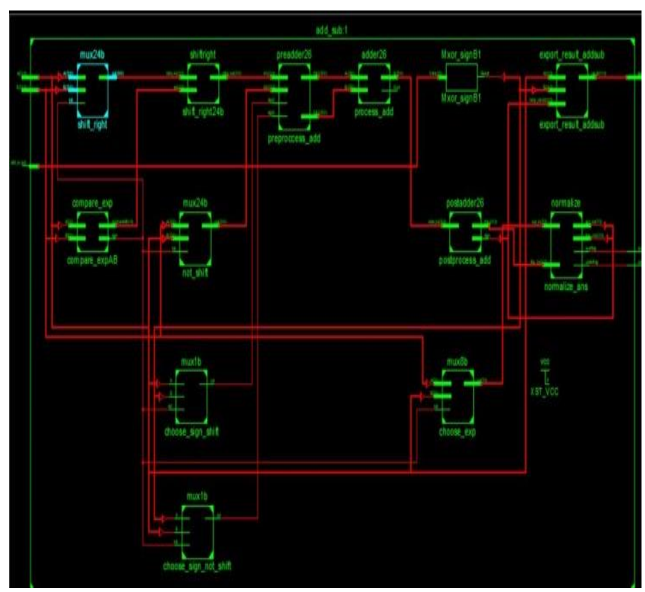

Figure 4: Addition/subtraction schematic (RTL view)

III. MULTIPLICATION MODULE

In the multiplication operation, the mantissas of two inputs are multiplied using algorithm given in Figure 6 with addition of their exponents along with the removal of bias (127). It can be illustrated as: eR = eA + eB - bias (127). Ex-ORing the two input sign bits determines the output sign

Published online : 25 February 2022

bit. After normalization, exception handling is performed. The algorithm for floating point multiplication is explained through the flow chart using "*" operator. Algorithm and flow chart for multiplication module is shown in figure 5, 6 and 8.

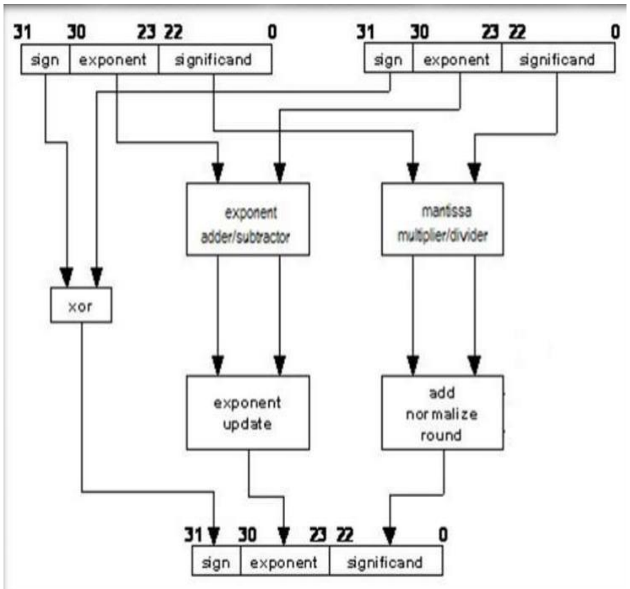

Figure 5: Multiplication and division flow chart

Algorithm:

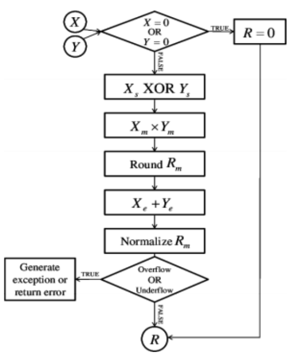

Figure 6: Multiplication algorithm

Step 1: Determine for oper equals to 4‟b0010

Step 2: Product is found as fa[22:0] * fb[22:0]

Step 3: Resultant exponent is geven by

r[30:23] = ea[30:23] + eb[30:23] – 127

Step 4: Finally evaluate for exception conditions overflow. If for product (r[30:23] >255), then do, set the overflow flag to 1

Step 5: Also calculate sign bit as

r[31] = sa[31] xor sb[31]

Step 6: Aggregate the result as concatenation of

{ r[31], R_exponent, product }

Code is written in Verilog and synthesized register transfer logic (RTL) view of multiplication logic is shown in Figure 7.

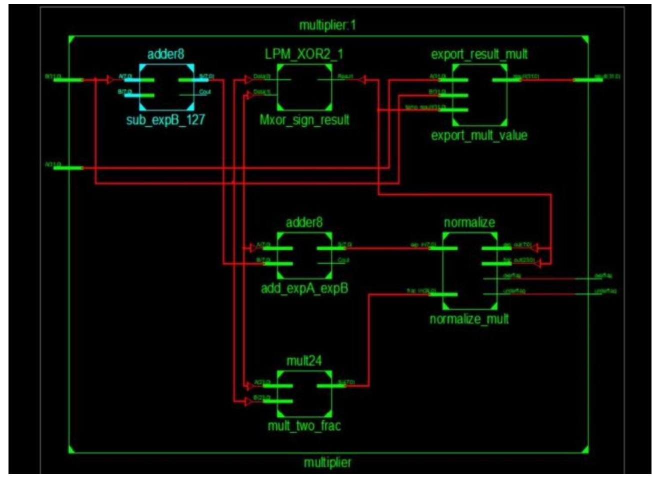

Figure 7: Multiplication schematic (RTL view)

IV. DIVISION MODULE

In the division operation, subtraction of exponents take place along with addition of bias (127). The mantissas of two inputs i.e. one dividend and other divisor are divided and sign is evaluated followed by normalization, if required. It is illustrated using the equation, eR = eA – eB + bias (127).

The sign of result was computed by XORing the signs of two operands. Division of two numbers is performed using in-built "/" operator available in Verilog library. 32- bit result_div_ieee register is used to store the quotient and register named as remainder is used to store the remainder of the division operation. The algorithm for floating point division is explained through the flow chart.

Algorithm:

<u>Step</u> 1: Determine for oper equal to 4‟b0010

<u>Step</u> 2: Evaluate result as dividing the mantissa

result_div = fracL[22:0] / fracs[22:0]

<u>Step</u> 3: Evaluate result by subtracting exponents
It can be expressed as

r[30:23] = eA[30:23] - eB[30:23] + 127

<u>Step</u> 4: Evaluate for the exception condition divide by zero..

If fracs [30:0] is all 0.
Set div_b by_zero flag to 1.

<u>Step</u> 5: Also calculate sign bit as

r [31] = sa [31] xor sb[31]

<u>Step</u> 6: Aggregate the result as concatenation of

{r[31], R_exponent, product }

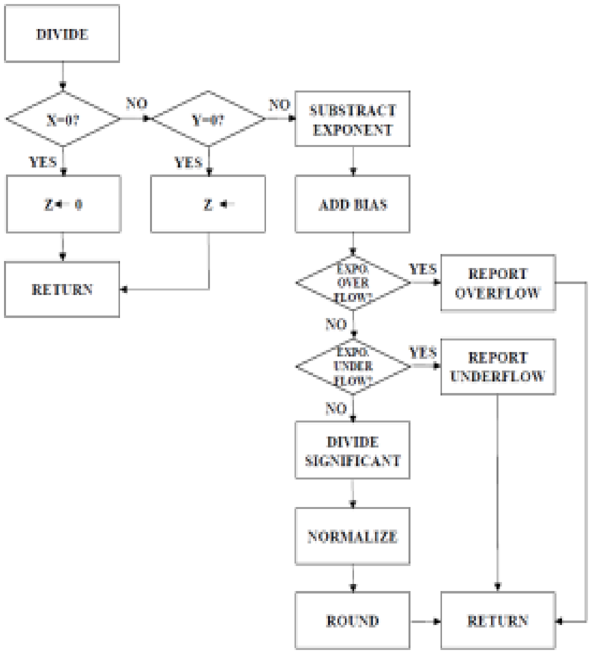

Figure 8: Division flow chart [7]

Code is written in Verilog and synthesized register transfer logic (RTL) view of division logic is shown in Figure 9.

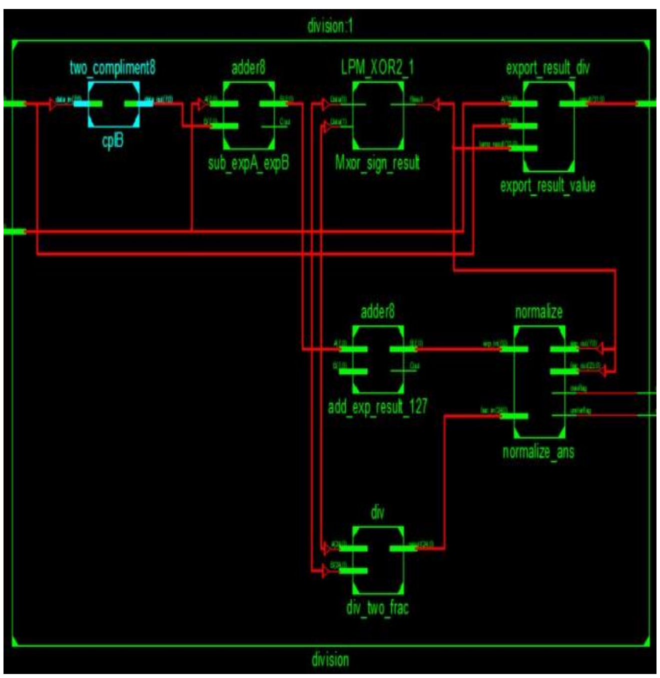

Figure 9: Division schematic (RTL view)

III RESULTS AND DISCUSSION

All operations i.e. addition, subtraction, multiplication and division are simulated and results are shown in Figure 10, 11, 12 and 13. Simulation results and theoretical results of all operations are summarized in Table 4 also.

Table 4: Simulated and theoretical results of all operations

Operation	**Sel (Binary)**	**A (Hex)**	**B (Hex)**	**Result Obtained (Hex)**	**Result Expected (Hex)**
Addition	00	42c80000	3e800000	42c88000	42c88000
Subtraction	01	42c80000	3e800000	42c78000	42c78000
Multiplication	10	42c80000	3e800000	41c80000	41c80000
Division	11	42c80000	41c80000	40800000	40800000

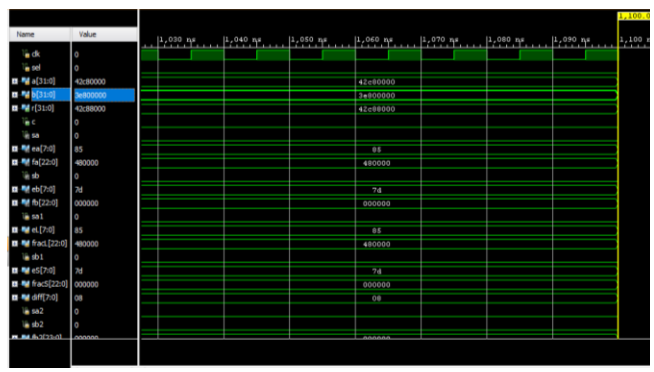

Figure 10: Simulation result of addition module

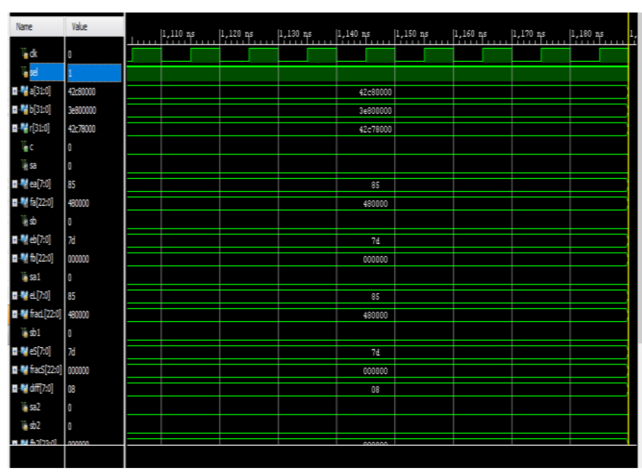

Figure 11: Simulation result of subtraction module

Published online : 25 February 2022

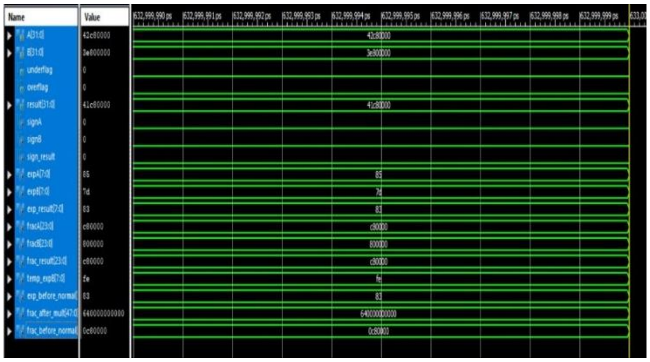

Figure 12: Simulation result of multiplication module

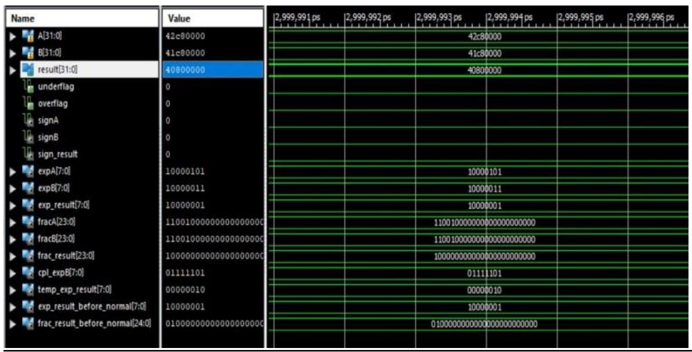

Figure 13: Simulation result of division module

IV CONCLUSIONS

In this paper IEEE-754 standard based single precision FPU capable to perform single precision addition, subtraction, multiplication and division is presented. The successful simulation and synthesis of the Verilog code of FPU has been presented. And the simulation results with RTL schematic given here are performed using Xilinx Design suit 14.7.

REFERENCES

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.

[2] L. Gangwar and R. Chaudhary, "Floating Point Arithmetic Unit Using Verilog", *Advance in Electronic and Electric Engineering* ISSN 2231-1297, vol 3 8 , pp. 1013-1018, 2013

[3] D. Saini, B. M'dia ,"Floating Point Unit Implementation on FPGA", *International Journal Of Computational Engineering Research* , ISSN: 2250–3005,| vol. 2, Issue 3, pp 972-976, 2012

[4] G. Ushasree, R. Dhanabal and S. Kumar sahoo, "VLSI implementation of a high speed single precision floating point unit using verilog," 2013 IEEE Conference on Information & Communication Technologies, 2013, pp. 803-808, doi: 10.1109/CICT.2013.6558204.issue 02, 2017

[5] Swathi. G. R and Veena. R , "Design of IEEE-754 Double Precision Floating Point Unit Using Verilog" *International Journal of Engineering Research & Technology (IJERT)*, ISSN:2278-0181, Vol. 3 Issue 4, April – 2014

[6] M. Ziaullah and A. Munaff, "Design and Implementation of Floating Point ALU with Parity Generator Using Verilog HDL", *IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)* Volume 5, Issue 5, Ver. I, pp 54-59, 2015

[7] Y. Savaliya, J. Rudani, "Design and Simulation of 32-Bit Floating Point Arithmetic Logic Unit using Verilog HDL", *International Research Journal of Engineering and Technology (IRJET)*, vol. 07, issue: 12, 2020

[8] P. Singh, and K. Bhole," Optimized Floating Point Arithmetic Unit", *Annual IEEE India Conference (INDICON, 2014*