INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 6.078
(Volume 6, Issue 3)
Available online at: www.ijariit.com

Design and implementation of floating point processor

Harshita Nair
harshitanair33@yahoo.com
Usha Mittal Institute of Technology, Mumbai, Maharashtra

Keerti Puthran
keertiputhran@gmail.com
Usha Mittal Institute of Technology, Mumbai, Maharashtra

Shivangi Srivastava
shivangionlinematlab@gmail.com
Usha Mittal Institute of Technology, Mumbai, Maharashtra

Sanchali Kshirsagar
sanchali.kshirsagar@umit.sndt.ac.in
Usha Mittal Institute of Technology, Mumbai, Maharashtra

ABSTRACT

The floating point processor is the backbone of digital signal processing units. In real time applications it is necessary to have a powerful processor which has a very high level of precision. This can only be achieved using a floating point unit. We are going to incorporate various instructions which will be serving for a dynamic range of applications. The floating point unit needs to be fast enough to process real time signals. An efficient design of the floating point processor can help in reducing the area and increasing the speed.

Keywords— *Floating point, FPGA, Xilinx*

1. INTRODUCTION

Floating point number notation is used to represent real numbers and do computations with them. The “floating point” or the decimal point is not fixed and can be placed anywhere in relation to the mantissa part of the number. This position can be represented as an exponent value. In the past few years a standard was set for floating point numbers representation by IEEE. It is called the IEEE 754 standard and is the most commonly used standard for representing floating point numbers. High speed processors which use complex tasks for arithmetic operations use the operation of these numbers.

In the present century high speed processors are required to perform high computational tasks which involve floating point arithmetic. In order to design an efficient floating point processor, there is a need to improve on constraints like power, area or speed. The floating point processor can be implemented using VHDL and on an FPGA board.

2. FLOATING POINT NUMBER REPRESENTATION

The IEEE 754 is the set technical standard for Floating-Point Arithmetic established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE). This standard is widely used in hardware units of computers. The 3 main components of the IEEE 745 floating point representation are Sign, Exponent and Mantissa.

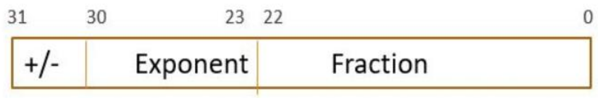


Fig. 1: IEEE standard 32- bit floating point number representation

- Sign: The Sign bit is used to represent if the number is positive or negative. If it is 0 the number is positive and if 1 then the number is negative.
- Exponent: The Exponent bit represents both positive and negative exponents. An exponent bias is added to the actual exponent to get the stored exponent.
- Fraction (Mantissa): The Mantissa bit is used to store the numbers in floating point format. So, a normalized mantissa is with a leading 1 to the left of the decimal.

3. LITERATURE SURVEY

This paper [1] contains comparison of three algorithms for adder circuit namely Standard Algorithm, LOP Algorithm and Far and close data path algorithm. The Standard Algorithm is area efficient, the LOP Algorithm does not affect the overall latency, Far- and close data path algorithm has significant drop in overall latency. All the Algorithms contribute significantly in terms of performance, area efficiency and reduced latency but in all the cases one or the other attribute needs to be compromised.

In this paper [2] A 32-bit floating point unit is implemented with an efficient algorithm for basic operation such addition, subtraction, multiplication and division. The algorithms for basic operations are simulated using ModelSim in Verilog HDL. The addition and subtraction modules are modified in such a way that no. of gates used are reduced from 2539 to 581.

In this paper [3] the multiplier is a digit serial multiplier. Area and speed trade-offs are made for accuracy. There are two approaches for multiplication: one is the digit serial multiplier which takes 12 clock cycles for complete result to be seen and an alternative approach would be to use a parallel adder, in this case 7 clock cycles in total but it also comes with a downside of more logic cells, Maximum performance of 7MFlops were obtained on an Altera FLEX8000 prototype for addition and 2.3MFlops for multiplication were obtained on our Altera FLEX8000 prototype. There was also an area reduction from 72% to 47%. In this paper there is no rounding of the result for adder while it can be done for future implementations. The main reason for not including rounding is that there are certain area limitations with the FPGA.

This paper [4] provides an IEEE 754 single precision floating point multiplier in Xilinx Virtex-5 FPGA. The implemented multiplier supports binary interchange format and has 3 pipelining stages. The multiplier handles overflow and underflow conditions. In order to provide more precision when being used for a MAC unit that is, Multiply and Accumulate, the rounding block is not implemented to. It has achieved 301 MFLOPs.

The latency of division operation is normally between 6-61 cycles hence in this paper's [5] design the authors have designed a sequential double precision floating point divider to achieve a low area and moderate latency and more throughput through increased pipelining. The design was mapped onto a Cyclone-II FPGA. The whole circuit operation took 62 clock cycles. With the help of more pipelining which increased area overhead and partial unrolling of the circuit, the overall throughput of the circuit was increased. The area usage was 2% for total logic registers, static power dissipation was 47.41mW and clock frequency was 265 MHz. This design showed an improvement in comparison to a functional iteration divider unit.

This paper [6] presents 3 division algorithms and their respective implemented model results. The algorithms are as follows Newton Raphson division, Goldschmidt division and Goldschmidt division with binomial simplification. Multiplicative inverse approach is used for division that is, there is continuous subtraction of the divisor from the dividend requiring a lot of iterations. The total LUTs used for all three models are 2448,2441,2453. No of flip flops used are 291,161,195 and the delay for Newton Raphson division and Goldschmidt division with binomial simplification is 13*clk while for Goldschmidt division it is 11*clk. The result shows that the Goldschmidt implementation is the best from all presented implementations

4. ARCHITECTURE DESIGN OF THE FPU

The figure 2 shows the basic architecture diagram.

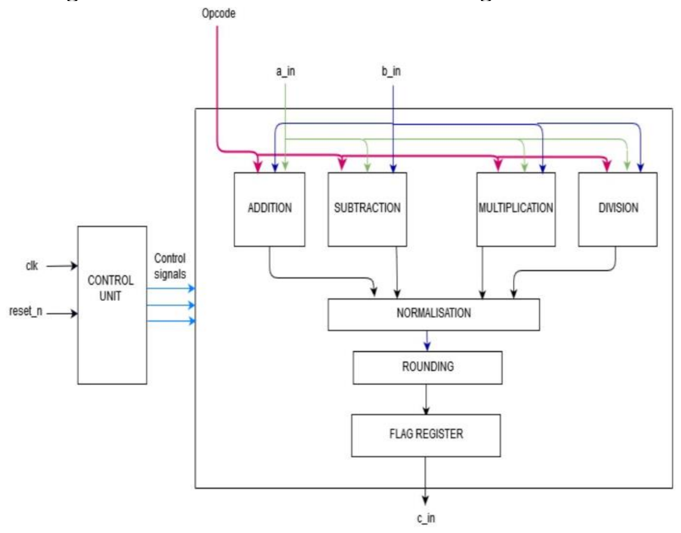


Fig. 2: Basic Architecture of Floating-Point Unit

The control unit generates the necessary control signals. It directs the operation of the other units by providing the timing and control signals. Two operands a_in and b_in having normalized values are taken as input. The opcode decides upon which operation needs to be performed from addition, subtraction, multiplication and division. After the operation is formed, the normalisation unit converts the mantissa in the required format. The result is rounded or truncated and checked for exceptions like overflow, underflow etc. which changes the values of the flag registers. The result c_in is taken as output from the floating-point unit and displayed on the display.

5. INPUT AND OUTPUT SIGNALS IN ARCHITECTURE DESIGN OF FLOATING-POINT PROCESSOR

The different input and output signals that are used in our floating point processor are shown in the figure 3

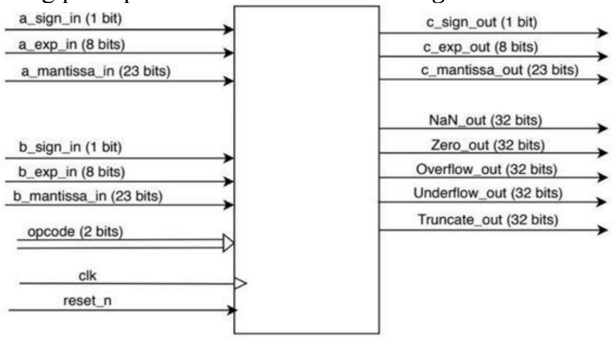


Fig. 3: Input/Output signals for floating point processor

The different inputs a in and b in are taken and the output c in is shown in the standard format. The clock signal for the processor is shown as a clock signal. The reset n signal is used to reset the processor. Overflow out when an overflow condition occurs i.e. operation results in a larger magnitude value than the largest finite value. Underflow out is when the underflow condition occurs i.e. when the operation results in a value smaller than the smallest finite value. Table 1 shows the description of the different signals used in the floating-point processor.

Table 1: Input and Output Signals

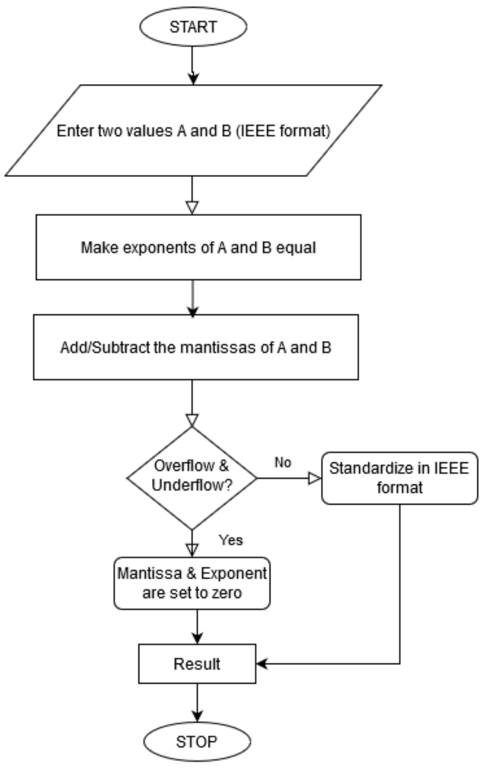
Name of the signal	Direction	Width	Description
a_sign	input	1 bit	Input operand a sign
a_mantissa	input	23 bit	Input operand a mantissa
a_exp	input	8 bit	Input operand a exponent
b_sign	input	1 bit	Input operand b sign
b_mantissa	input	23 bit	Input operand b mantissa
b_exp	input	8 bit	Input operand b exp
clk	input	-	Clock signal for floating point unit
reset_n	input	1 bit	Processor reset
Nan_out	output	32 bits	Undefined output
Zero_out	output	32 bits	Output is zero
Overflow_out	output	32 bits	Result exceeds max limit
Underflow_out	output	32 bits	Result is below min limit
Truncate_out	output	32 bits	Truncated output result

6. ALU UNITS

6.1 Adder/Subtractor

The flowchart of the adder/subtractor unit is given as below in the figure 3. The below algorithm is used to add/subtract the two floating point numbers

1. Entering two operand values A and B in IEEE format.
2. Checking if the two exponents are the same. If not, shift the mantissas to match the exponents.
3. Perform mantissa addition or subtraction operation.
4. Check for overflow and underflow conditions
5. If overflow condition occurs then set both mantissa and exponent values to zero.
6. If not then normalise the mantissa to bring it in normal range.
7. Final result is obtained in standard IEEE format.



START
Enter two values A and B (IEEE format)
Make exponents of A and B equal
Add/Subtract the mantissas of A and B
Overflow & Underflow?
No
Standardize in IEEE format
Yes
Mantissa & Exponent are set to zero
Result
STOP

Fig. 3: Flowchart of floating point adder

The block diagram of the adder/subtractor unit is given in figure 4. The exponent difference goes to the right shifter. The result of exponent difference and MUX goes into the add/sub unit. The last step is to normalise the output of add/sub block.

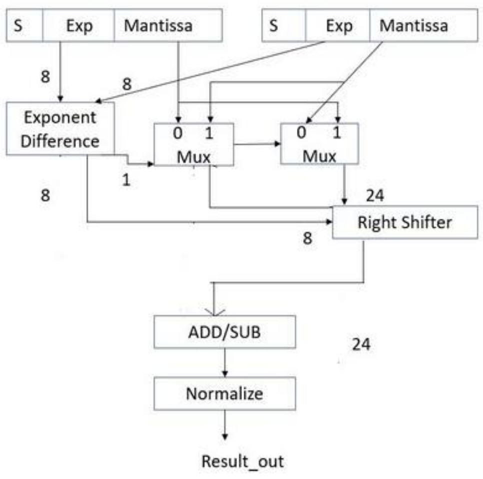


Fig. 4: Block diagram of floating-point adder

6.2 Multiplier/Divider

The flowchart of the multiplier/divider unit is given in the figure.

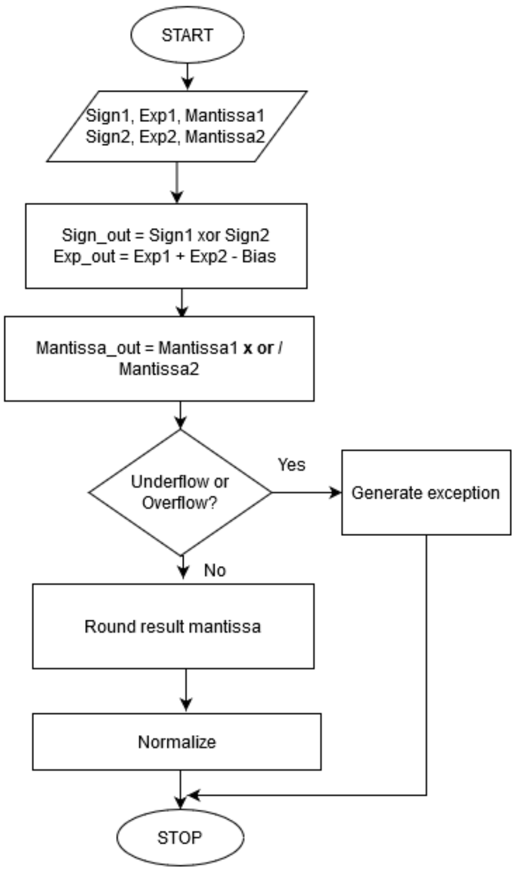


Fig. 5: Flowchart of multiplication and division unit

The following algorithm is used to multiply the two floating point numbers.

1. Multiplying the significant i.e. (1.M1 * or / 1.M2).
2. Placing the decimal point in the result.
3. Adding the exponents i.e. (E1+ E2 - Bias)
4. Obtaining the sign i.e. s1 XOR s2
5. Normalizing the result
6. Rounding the result to fit in the available bits.
7. Checking for underflow/overflow occurrence.

The block diagram of the multiplier/divider unit is given below in the figure.

It shows two numbers taken as A and B. Their signs are XORed and taken as sign_out. The exponents are added and the bias value is subtracted. The two mantissas are multiplied or divided as per the operation and the result is normalised to give a final result in standard IEEE format.

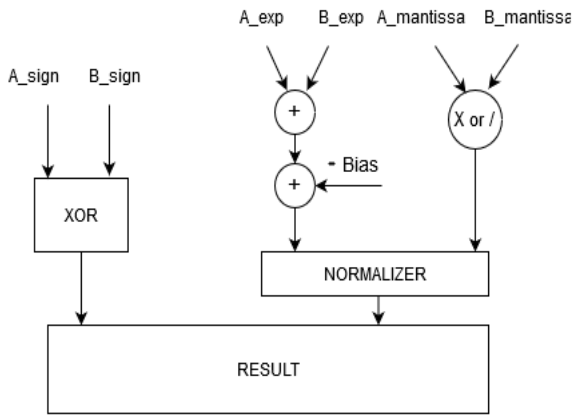


Fig. 6: Block diagram for floating point multiplier/divider

7. SIMULATION RESULTS

Shown below are the different simulation examples and their results for different operations.

7.1 Addition operation

EG. 0.5 + 0.4
OP A=0.5= 3F000000 (HEX)
OP B=0.4= 3ECCCCCC (HEX)

The result of addition is 0.5 + 0.4 = 0.9 = 3F666666
= 0-00111111-01100110011001100110
which is shown in the binary form in the simulator fig. 7

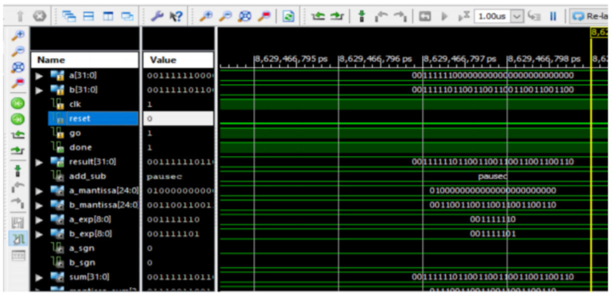


Fig. 7: Simulation for floating point addition

7.2 Subtraction operation

EG. 21.5 - 8.26
OP A = 21.5 = 41AC0000 (HEX)
OP B = - 8.26 = C10428F5 (HEX)

The result of subtraction is 21.5 - 8.26 = 13.24 = 4163D70A
= 0-10000010-10100111101011100001010 which is shown in the binary form in the simulator figure 8.

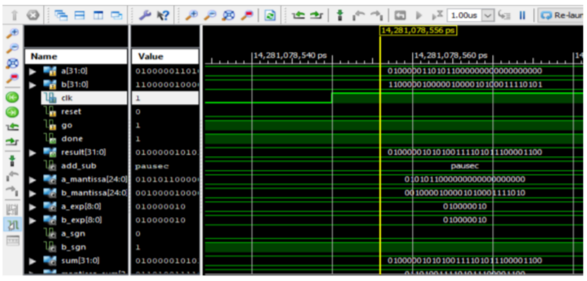


Fig. 8: Simulation for floating point subtraction

7.3 Multiplication operation

Eg: 4.25 x 2.5

Op A = 5.7 = 40B66666 (HEX)
Op B = 3.4 = 40599999 (HEX)

The result of multiplication 5.7 x 3.4 is 19.38
= 419B0A3D = 0-10000011-00110110000101000111101
which is shown in the binary form in the simulator figure 9.

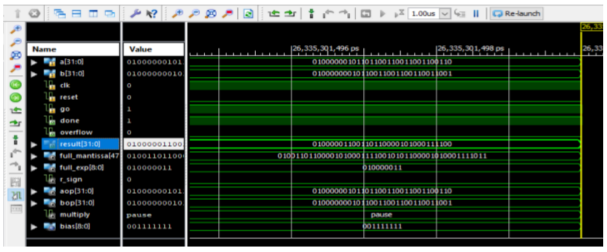


Fig. 9: Simulation for floating point multiplication

7.4 Division operation

Eg: 7.3/ 2.4

Op A = 7.3 = 40E99999
Op B = 2.4 = 40199999

The result of division 7.3 / 2.4 is 3.04166666667
= 4042AAAA = 0-10000000-10000101010101010101010
which is shown in the binary form in the simulator figure 10.

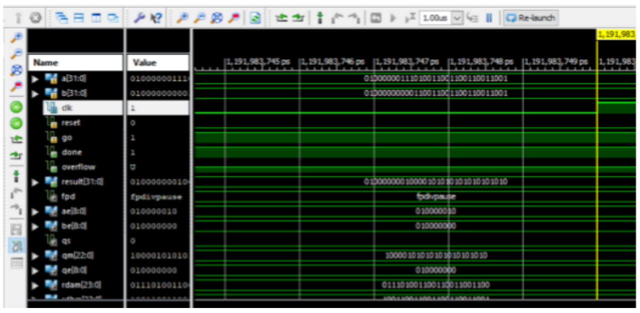


Fig. 10: Simulation for floating point divider

The operation of the floating-point processor happens by entering different 3-bit opcode values like 000 for add, 001 for sub, 010 for mul and 011 for div.

The table 2 below shows the device utilisation of different units of FPU and the floating point processor.

Table 2: Device Utilisation Summary

ALU Unit	Slice Registers	Slice LUTs	LUT-FF pairs	Delay
Adder/Subtractor	1%	21%	10%	5.591ns
Multiplier/ Divider	1%	3%	40%	3.643ns
FPU	13%	65%	24%	10.004ns

Minimum period (delay) obtained for the floating-point processor was 10.004ns (i.e. Maximum Frequency: 99.960MHz).

8. CONCLUSION AND FUTURE SCOPE

In this paper, a single precision floating point processor is implemented that can do four operations addition, subtraction, multiplication and division. The different operations performed showed an 88% accuracy of result on comparing to the manually calculated results.

The device utilisation in terms of number of slice registers utilisation for adder/subtractor unit is coming 2%, slice LUTs is 5%, LUT-FF pairs is 41%. The delay for the addition unit was 5.591ns and CPU execution time was 7.44 sec. For the multiplier unit the device utilisation comes 3% for IOBs. The delay for multiplier comes 3.643 ns. For the divider unit the slice register utilisation is 1% and LUT-FF pairs as 3%. In case of the full processor working the total CPU time for execution completion comes out as 47.97 sec, delay as 10.004ns.

For future scope of the project, the floating point unit can be maximised in speed, minimized in area, more power efficiency can be added for efficient and fast operation performance. Different specific applications can be used for the floating point processor like signal processing, image processing, cryptography etc. Other scope includes:

- Extension for double precision floating point numbers
- Different implementations of operations like vedic addition and multiplication
- High throughput multipliers can be used for video signal processing

9. ACKNOWLEDGEMENT

The authors would like to acknowledge their mentor Ms. Sanchali Kshirsagar and Usha Mittal Institute of Technology for the support of their work.

10. REFERENCES

[1] A. Malik and S. Ko. A study on the floating-point adder in fpgas, 2006. Available; accessed 14-Jan-2020.

[2] Nisha Singh and R Dhanabal. Design of single precision floating point arithmetic logic unit. In 2018 4th International Conference on Electrical Energy Systems (ICEES), pages 133–137. IEEE, 2018.

[3] Loucas Louca, Todd A Cook, and William H Johnson. Implementation of ieee single precision floating point addition and multiplication on fpgas., 1996.

[4] Mohamed Al-Ashrafy,Ashraf Salem,Wagdy Anis, An Efficient Implementation of Floating Point Multiplier,2011 IEEE.

[5] Shamna.K and S.R Ramesh, Design and implementation of an optimised double precision divider on FPGA ,International Journal International Journalof Advanced Science and Technology of Advanced Science and Technology of Advanced Science and Technology Vol. 18 Vol. 18, May, 2010

[6] Peter Malik,High Throughput floating point dividers implemented in FPGA,2015 IEEE.

[7] Tarun Kumar Rawat Abhay Sharma. Truncated wallace based single precision floating point multiplier, 2018.

[8] Nihal Nitnaware Adil Shaikh, Akash Ninave and Prof. Shaiwali Ballal. Ieee 754 single precision floating point arithmetic unit using vhdl, 2017. Available; accessed 15-Jan-2020.

[9] Ayusharma0698. Ieee standard 754 floating point numbers. https://www.geeksforgeeks.org/ieee-standard-754-floating-point-numbers/, Sep 2019. Online; accessed 09-Oct-2019.

[10] Anand Burud and Pradip Bhaskar. Design and implementation of fpga based 32-bit floating-point processor for dsp application. In 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), pages 1{5. IEEE, 2018.

[11] Karan Gumber and Sharmelee Thangjam. Performance analysis of floating-point adder using VHDL on reconfigurable hardware, 2012. Available; accessed 15-Jan-2020.

[12] Z. Abid Jasbir N. Patel and Wei Wang. VLSI implementation of a floating-point divider, 2004.

[13] Lokesh Kamble, Prachi Palsodkar, and Prasanna Palsodkar. Research trends in development of floating-point computer arithmetic. In 2017 International Conference on Communication and Signal Processing (ICCSP), pages 0329{0333. IEEE, 2017.

[14] Mohamed ElSaid Khaled A. Shehata, Hassan El-Ghitani. Design of generic floating-point multiplier and adder/subtractor units, 2010.

[15] S. Koteswara Rao and T. Srinivasa Rao. Design and implementation of 32-bit inexact floating-point arithmetic unit, 2018. Available; accessed 15-Jan-2020.48

[16] William H. Johnson Loucas Louca, Todd A. Cook. Implementation of IEEE single precision floating point addition and multiplication on FPGAS, 1996.

[17] P Narayana Murty Molleti Rajani. Verilog implementation of double precision floating point division using vedic paravartya sutra, 2015.

[18] Khushbu Naik and Prof. Tarun Lad. Implementation of IEEE 32-bit single precision floating point addition and subtraction, 2015. Available; accessed 15-Jan-2020.