International Journal of Research Studies in Science, Engineering and Technology
Volume 2, Issue 12, December 2015, PP 15-21
ISSN 2349-4751 (Print) & ISSN 2349-476X (Online)

Implementation of Area Efficient IEEE-754 Double Precision Floating Point Arithmetic Unit Using Verilog

Bhaskar Chittaluri

M. Tech, ECE department, Malla Reddy Engineering College (Autonomous), Hyderabad, India.
bhaskar.chittaluri@gmail.com

Abstract: *This methodology considered for the many applications related to the different criterions of basic characteristic requirements. This can be carried via 64- bits, which employs the IEEE 754-2008 standard format for floating point unit design. In our proposed method we apply the novel constructs related to the different considerations of the VLSI. Unlike the traditional methods we adopted betterment multiplication algorithm and the addition of the partial product. For accomplishment of floating point operational unit the standard IEEE format can be acquired. The proposed concept provides the additional solution that able to perform the many operations such as addition, subtraction multiplication and division with accurate response and effectiveness. From out of the previous methods this proposal may affords the less footprint with more accuracy.*

Keywords: *Double Precision, Floating Point number, Normalization, Overflow, Underflow, etc.*

1. Introduction

At this level no other promising approaches for the real number operations. By this only IEEE 754-2008 can be used in this format include the two edicts. They are single precision and double precision and the uses the two lengths of bits, 32 and 64 respectively.

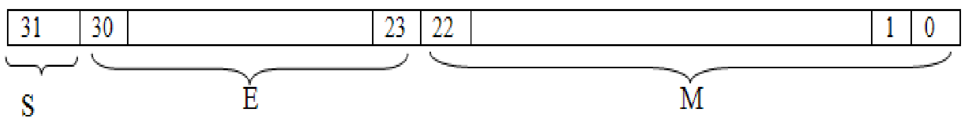


Figure1. *IEEE single precision floating point format*

S = Sign , E = Exponent, M = Mantissa

$$X = (-1)^{s} * 2^{(E-bias)} * (1.F)$$

As from the previous methods that uses the IEEE 754-2008 binary interchange format with the 32 format. In this the multiplication of the significant can be carried out by the 24 bits that results the 48 bit output. Which will able to gives the high end output when they use the whole bits for the operation. For the improvement thesis utilizes the rounding of the numbers can also be considered. When we concentrated on the high accuracy the area and the operational power also increases. Moreover, when we use the larger bit count (precision) i.e. for the longer real numbers, this method cannot provide that much good result with this concept.

Many advanced methods are increasing by the progress of requirements of the user. By this aspect user cannot able to accept the huge truncation error with reference of the factorization.

Sign	Exponent	Mantissa
63	62………..52	51…………………………..0

Figure2. *IEEE double precision floating point format*

From the above format ‘63’ bit concerns for the sign of the number if this is high the number considered as the negative, if zero, the number is treated as positive. Consecutive ‘11’upper bits from 62 bit considered as the exponent and remaining all bits can be considered as the mantissa i.e. 52 bits from 51 to 0.

The literal value of the double precision floating point number is as follows:

$$\text{Value} = -1\text{^} (\text{sign bit}) * 2\text{^} (\text{exponent} - 1023) * 1. (\text{Mantissa})$$

(1.mantissa) being a base 2 representation of a number between 1 and 2, with 1 followed by a decimal point and the 52 bits of the mantissa. There by the large values can be able to represent. This will increase the accuracy. That causes the reduction of the truncation error.

This paper organizes as follows section I deals with basic introduction about the floating formats and previous methods, section II describes the proposed floating point unit, section III provides the results and discussion and followed by the conclusion and references.

2. Floating Point Unit

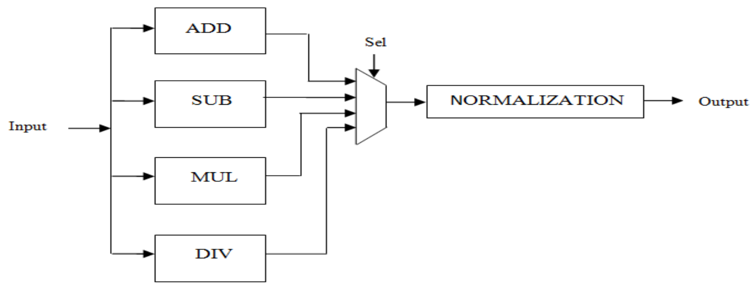


Figure3. *Floating point unit*

In our proposed floating point architecture consists of the four sub units namely addition, subtraction, multiplication and division. For these units receive the double precision formatted outputs and performs the operation produces the outputs, those outputs are selected by the selection multiplexer, that multiplexer will controlled by using the ‘sel’ signal given by the user.

2.1. Addition

From the addition algorithm inputs are taken, large exponents can be determined from two they are namely exponent large (EL) and exponent small (ES). EL and ES subtracted and that many respected shifts performed with Mantissa small, and then shifted mantissa and, mantissa large will be added to find the mantissa output. Exponent large will be the exponent output.

S	Exp large (EL)	TM

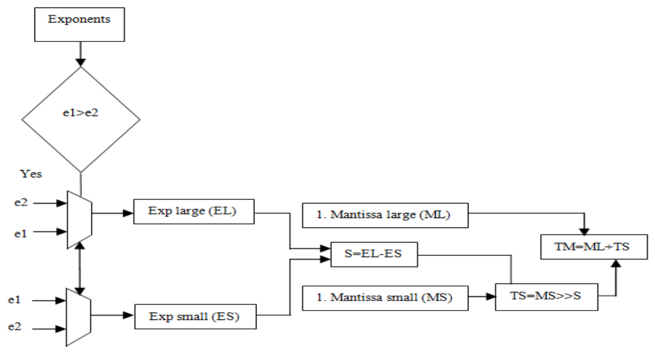


Figure4. *Addition algorithm*

2.2. Subtraction

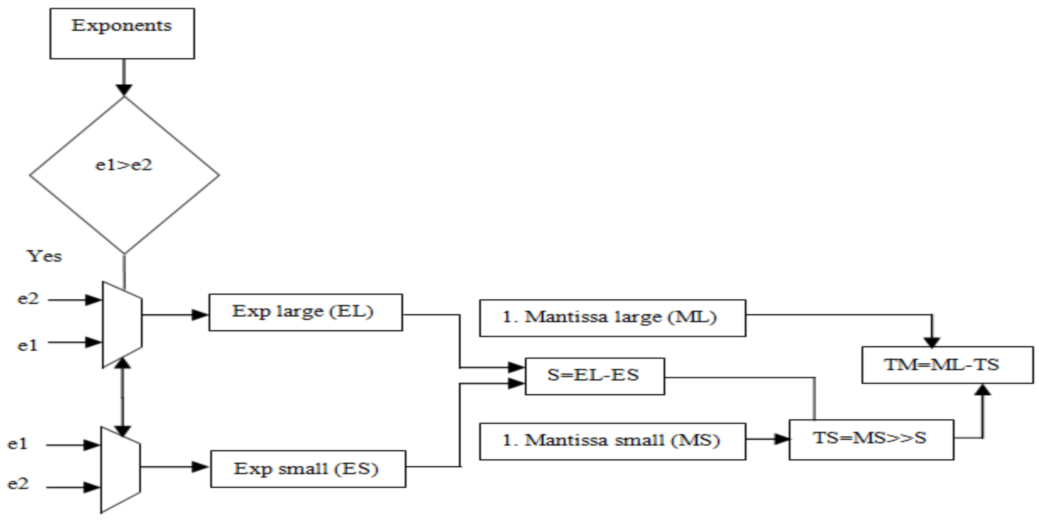


Figure5. *Subtraction algorithm*

Subtraction algorithm is similar to that of the addition, the only difference is that rather than performing shifted mantissa and mantissa large addition here we perform subtraction, remaining will be the same.

2.3. Multiplication

For performing the floating point multiplication there are some basic steps to be followed those steps are:

1. Multiplying the significand; i.e. (1.M1 * 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^S2
5. Normalizing the result; i.e. obtaining 1 at the MSB of the results significant
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence

Unlike the previous, for significant multiplication we use the radix-4 unsigned booth algorithm because, in general we don't deals with the negative number multiplication method because in our multiplication, MSB is not concern with the sign of that particular numbers. There by we can reduce the area and bit count that related to the signed multiplication.

The multiplication is segmented as the partial product generation and tree addition and the final addition, here as discussed above, the radix-4 unsigned algorithm used for the partial product generation and compressor can be used for the tree addition and the Normal adder can be used for the final addition.

In this the booth recoding can be of three bits each starts for either LSB to MSB or MSB to LSB by the over lapping each group numbers that are multiplied. By this we can reduce the number of partial products by half, by radix-Booth recoding technique can be considered as the partial product generation.

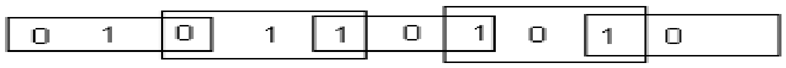
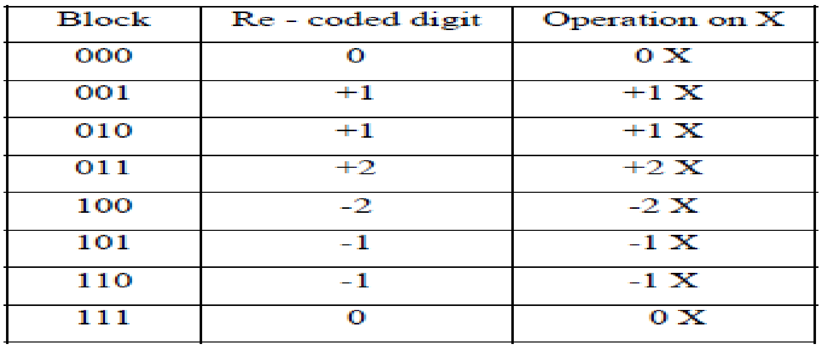


Figure6. *Grouping of bits from the multiplier term*

Table1. *Redundant value for group pairing*



Block	Re - coded digit	Operation on X
000	0	0 X
001	+1	+1 X
010	+1	+1 X
011	+2	+2 X
100	-2	-2 X
101	-1	-1 X
110	-1	-1 X
111	0	0 X

Each block is decoded to generate the correct partial product. The encoding of the multiplier Y, using the modified booth algorithm, generates the following five signed digits, -2, -1, 0, +1, +2. Each encoded digit in the multiplier performs a certain operation on the multiplicand, X, as illustrated in Table 1

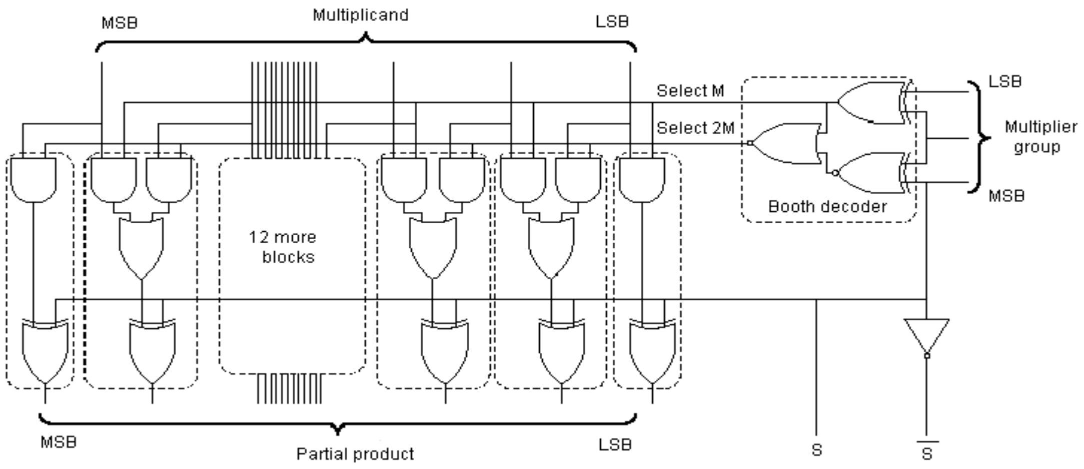


Figure7. *Radix-4 partial product selector logic*

Exponent addition can be carried out by the adder but this resultant added output consists of the two bias numbers so the subtractor circuit can be used to subtract the one bias from the resultant exponent add out.

Sign determination can be carried out by the XOR gate.

The normalization operation can be performed to prevent ‘1’ at most significant position. Condition for the normalization can be shown below

Table2. *Normalization conditions*

104	103	Normalization
0	0	No
0	1	No
1	0	Yes
1	1	No

As part of normalization if normalization condition occurs then the mantissa will shift one position right and exponent will be incremented by one. Before going to the normalization itself the overflow/under flow can be determined for the used identification purpose. Below Figure shows the table of under flow/overflow.

Table3. *Overflow/under flow*

Eout	Condition	Value
Eout < 0	Under flow	00
Eout =0	Zero	01
Eout<1023	Normalized number	10
Eout>1024	Over flow	11

2.4. Division

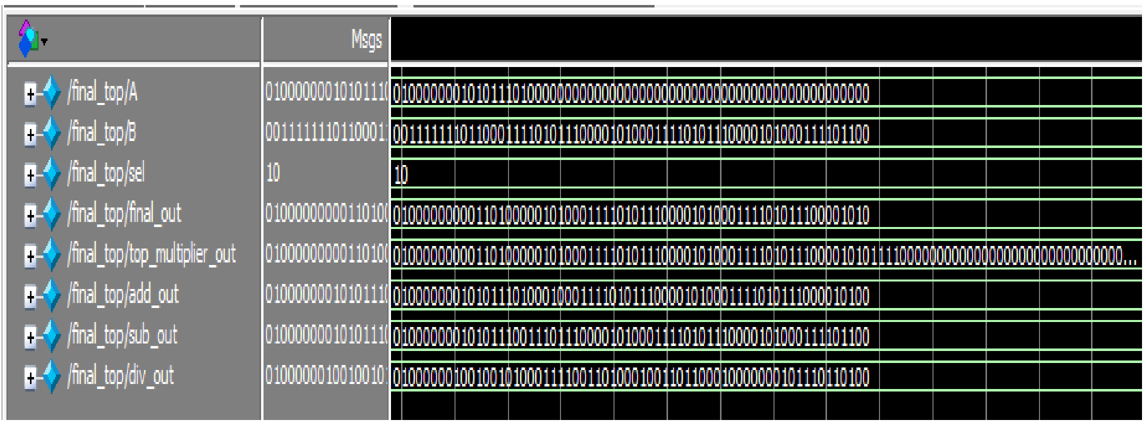
Division operation is similar to that of the multiplication. The divide is executed long hand style, with one bit of the quotient calculated each clock cycle based on a comparison between the dividend register and the divisor register.

If the dividend is greater than the divisor, the quotient bit is '1', and then the divisor is subtracted from the dividend, this difference is shifted one bit to the left, and it becomes the dividend for the next clock cycle.

If the dividend is less than the divisor, the dividend is shifted one bit to the left, and then this shifted value becomes the dividend for the next clock cycle.

The exponent can be calculated from the subtracting the dividend exponent can be subtracted from the divisor and bias will be added to it to produce the exponent out.

3. Results and Discussion



```
                              Msgs
/final_top/A                  01000000010101110   0100000001010111010000000000000000000000000000000000000000000000
/final_top/B                  00111111101100011   0011111110110001111010111000010100011110101110000101000111101100
/final_top/sel                10                  10
/final_top/final_out          01000000000110100   0100000000011010000010100011110101110000101000111101011100001010
/final_top/top_multiplier_out 01000000000110100   0100000000011010000010100011110101110000101000111101011100001010111000000000000000000000000000000...
/final_top/add_out            01000000010101110   0100000001010111010001000111101011100001010001111010111000010100
/final_top/sub_out            01000000010101110   0100000001010111001110111000010100011110101110000101000111101100
/final_top/div_out            01000000100100101   010000001001001010001111001101000100110110001000000101110110100
```

Figure8. *Simulation of proposed ALU unit*

The double precision floating point arithmetic unit design was simulated in Modelsim 6.6d and synthesized using Xilinx ISE 14.3. The simulation results of 64-bit double precision floating point arithmetic unit are shown in above figure8.

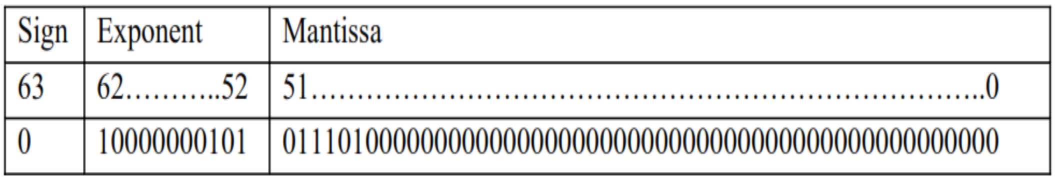
Addition:

The exponent value for A (93) is 1029. For B (.07), the exponent is 1019. The differences between the Exponents are 10. So the mantissa of B will be shifted to the right by 10 bits, and then added to the mantissa of A.

Also, because both numbers are normalized, it means their exponents are greater than 0, the leading '1' needs to be included in the addition.

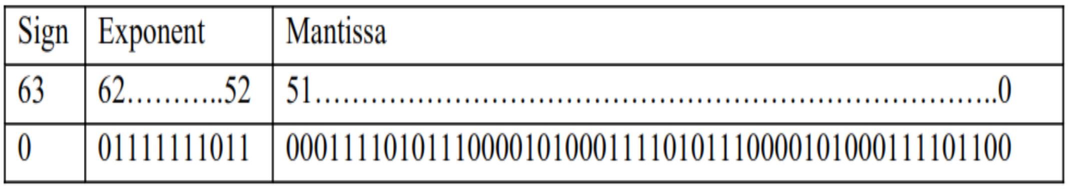
For addition operation, suppose we are adding 93 and .07.

Operand A (93) is represented in the double precision floating point format as:



Sign	Exponent	Mantissa
63	62……….52	51………………………………………………………………..0
0	10000000101	0111010000000000000000000000000000000000000000000000

Operand B (.07) is represented as:



Sign	Exponent	Mantissa
63	62……….52	51………………………………………………………………..0
0	01111111011	0001111010111000010100011110101110000101000111101100

Subtraction:

Subtraction is similar to addition except addition of mantissa here we perform subtraction.

Multiplication:

The exponent value for A (93) is 1029. For B (.07), the exponent is 1019, and then exponent are added. The leading '1' needs to be included in the multiplication of mantissas, Also the result is normalized.

Division:

The exponent value for A (93) is 1029. For B (.07), the exponent is 1019, and then exponent are subtracted. The leading '1' needs to be included in the dividing of mantissas.

All operations will be performed every time we give input, but based on the 'sel' input only one assigned to output.

Table4. *Device Utilization Summary of Double Precision Floating Point Arithmetic Unit*

Logic Utilization	Used
Number of Slice Registers	104
Number of Slice LUTs	13313
Number of fully used LUT-FF pairs	104
Number of bonded IOBs	194

Table5. *Area of Double Precision Floating Point Multiplier, Double Precision Floating Point Multiplier[14]*

Device parameters	Present work	Addanki Purna Ramesh,A.V.N Tilak And Dr.A.Mallikarjuna Prasad[14]
	Double precision	Double precision
No .of slices	4425	5648

Table4 shows the device utilization of double precision floating point arithmetic unit and it occupies an area of 13313 slices.Table5 shows the area of double precision floating point multiplier. Addanki Purna Ramesh,A.V.N Tilak And Dr.A.Mallikarjuna Prasad[14] implemented double precision floating point multiplier and it occupies an area of 5648 Slices. So the implemented design provides less area with more accuracy.

4. Conclusion

Efficient floating point unit can be designed using Verilog HDL and simulated with Modelsim6.6d and synthesized using the XILINX ISE 14.3. In our proposed design we achieve the less partial product concern to the traditional multiplication algorithm and we also concentrated on the advanced method for the partial product addition many cases some improvement towards the speed and also the pipelined operation also can be achieved by using our design. By this we concluded that this is well suited for the high speed application and also less die requirements.

References

[1] Rudolf Usselmann, “Open Floating Point Unit, The Free IP Cores Projects”.

[2] Edvin Catovic, Revised by: Jan Andersson, “GRFPU – High Performance IEEE754 Floating Point Unit”, Gaisler Research, Första Långatan 19, SE413 27Göteborg, and Sweden.

[3] David Goldberg, “What Every Computer Scientist Should Know About Floating-Point Arithmetic”, ACM Computing Surveys, Vol 23, No 1, March1991, Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, California 94304.

[4] Yu-Ting Pai and Yu-Kumg Chen, “The Fastest Carry Lookahead Adder”, Department of Electronic Engineering, Huafan University.

[5] Prof. Kris Gaj, Gaurav, Doshi, Hiren Shah, “Sine/Cosine using CORDIC Algorithm”.

[6] S. F. Oberman and M. J. Flynn, “Division algorithms and implementations,” IEEE Transactions on Computers, vol. 46, pp. 833–854, 1997.

[7] Milos D. Ercegovac and Tomas Lang, Division and Square Root: DigitRecurrence Algorithms and Implementations, Boston: Kluwer Academic Publishers, 1994.

[8] ANSI/IEEE Standard 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[9] Behrooz Parhami, Computer Arithmetic - Algorithms and Hardware Designs, Oxford: Oxford University Press, 2000.

[10] Steven Smith, (2003), Digital Signal Processing-A Practical guide for Engineers and Scientists, 3rd Edition, Elsevier Science, USA.

[11] D. J. Bernstein. Multidigit Multiplication for Mathematicians. Advances in Applied Mathematics, to appear

[12] A. Karatsuba and Y. Ofman. Multiplication of Multidigit Numbers on Automata. Soviet Physics-Doklady, 7 (1963), 595-596.

[13] D. E. Knuth. The Art of Computer Programming. Volume 2: Seminumerical Algorithms. Addison-Wesley,Reading, Massachusetts, 3rd edition, 1997.

[14] Addanki Purna Ramesh,A.V.N Tilak And Dr.A.Mallikarjuna Prasad[2] Implemented “FPGA Based High Speed IEEE-754 Double Precision Floating Point Multiplier using verilog” pp.978-1-4673-5301-4/13/$31.00 ©2013 IEEE.