Journal of Advanced Research in Technology and Management Sciences
Volume: 00 Issue: 01 December 2018
Available online at: http://www.jartms.org

DESIGN AND IMPLEMENTATION OF FLOATING POINT UNIT USING VERILOG

Upendar S

Assistant Professor, Dept. of ECE, Nova College of engineering & Technology, Hyderabad, India.
Email:- upendarsum@gmail.com

ABSTRACT:- To represent very large or small values, large range is required as the integer representation is no longer appropriate. These values can be represented using the IEEE-754 standard based floating point representation. This paper presents high speed ASIC implementation of a floating point arithmetic unit which can perform addition, subtraction, multiplication, division functions on 32-bit operands that use the IEEE 754-2008 standard. Pre normalization unit and post normalization units are also discussed along with exceptional handling. All the functions are built by feasible efficient algorithms with several changes incorporated that can improve overall latency, and if pipelined then higher throughput. The algorithms are model in Verilog HDL and the RTL code for adder, subtractor, multiplier, are synthesized using HDL DESIGNER SERIES AND XILINX.

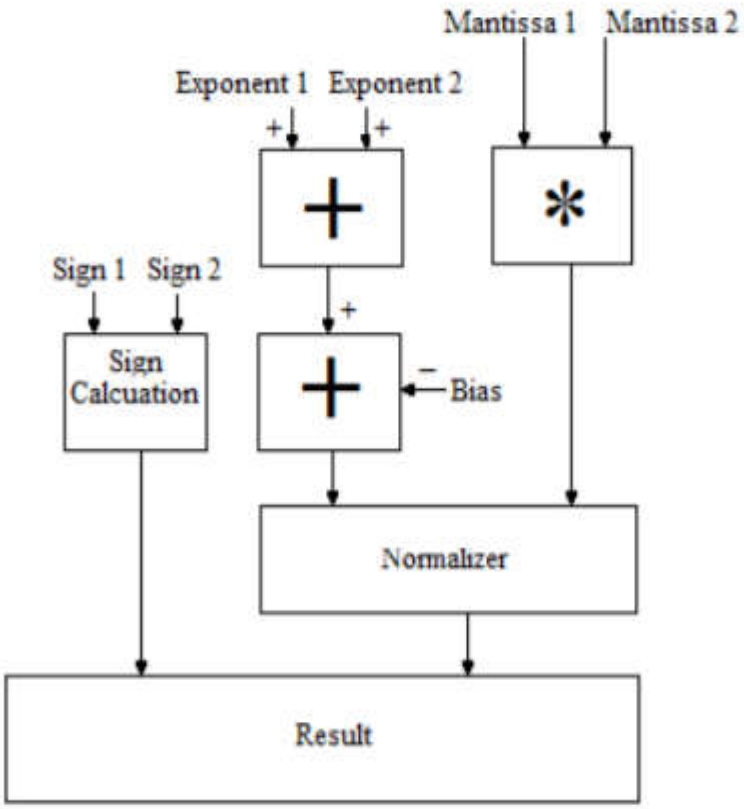
Key Words: IEEE-754, Verilog HDL, ASIC implementation, RTL code for adder, Subtractor, Multiplier.

1. INTRODUCTION TO FLOATING POINT MULTIPLIER

Floating point multiplication units are an essential IP for modern multimedia and high performance computing such as graphics acceleration, signal processing, image processing etc. There are lot of effort is made over the past few decades to improve performance of floating point computations. Floating point units are not only complex, but also require more area and hence more power consuming as compared to fixed point multipliers. And the complexity of the floating point unit increases as accuracy becomes a major issue. IEEE 754 [1] support different floating point formats such as Single Precision format, Double Precision format, Quadruple Precision format etc. But as the precision increases, multiplier area, delay and power increases drastically. In the proposed paper, we present a new multiplication method which uses a combination of Karatsuba and Urdhva-Tiryagbhyam (Vedic Mathematics) algorithm for multiplication. This combination not only reduces delay, but also reduces the percentage increase in hardware as compared to conventional methods. IEEE 754 format specifies two different formats namely single precision and double precision format [1, 2]. Fig. 1 shows the different IEEE 754 floating point formats used commonly. The Single precision format is of 32-bit wide and Double precision format is of 64-bit wide. The Most Significant Bit is the sign bit. The exponent is a signed integer. It is often represented as an unsigned value by adding a bias. In Single precision format, the exponent is of 8-bit wide and the bias is 127, i.e. the exponent has a range of 127 to 128. In Double precision format, the exponent is of 11-bit wide and the bias is 1023, i.e. the exponent has a range of 1023 to 1024. The mantissa or significant of Single precision format is of 23-bit and of double precision format is of 52 bit wide.

1.1 Floating point number:

A floating decimal point present in floating point number will not have any fixed number of digits after and before its decimal point. Computers recognize real numbers which consist of fractions as floating point number only. A floating-point number consists of Sign bit, Mantissa and Exponent bits. The value of the floating point number is obtained by multiplying its mantissa with the base raised to the power of the exponent. This operation may shift radix point to its right from its implied position by a number of places equal to the value of the exponent if it is positive and to the left from its implied position by a number of places equal to the value of the exponent if it is negative. Floating point number wildly represented with different magnitudes and provides relative accuracy at all magnitudes. A scientific notation of a floating point can be expressed in the form of F*r^E. Where F represents fraction, r represents radix and E represents an exponent.



Binary number uses radix as 2 and Decimal numbers uses radix as 10 (F×10^E). Floating point number will not have unique representation always. For example, the number 59.67 can be represented as 5.967×10^1, 0.5967×10^2, 0.05967×10^3, and so on. The fractional part of a floating point number can be normalized and there is only a single non-zero digit present before the radix point in the normalized form. For example 9.966×10^1 is the normalized form representation for the decimal number 99.66 and 1.1111011×2^3 is the normalized form representation for the binary number 1111.1011. When the floating point numbers are represented with fixed number of bits say 32 or 64 bit, it losses its precision. It is due to the fact that n-bit binary pattern can have a finite 2^n distinct numbers. Hence it is not possible to represent all the real numbers, instead nearest approximation will be used which results in loss of accuracy. Hence floating number arithmetic is very much less efficient than integer arithmetic. Real numbers are used in most scientific computations. Floating point standard has been used to represent the real numbers in computer. But there was no uniform format was maintained to use floating point numbers in a computer during 1950's which results in programs were not portable from one manufacturer's computer to another. To address this, a committee was formed by the Institute of Electrical and Electronics Engineers to standardize the format to be used to represent floating point number in a computer. IEEE standard for floating point number was adopted in 1985 and the same was used by all computer manufacturers. This standard introduces format for 32-bit and 64 bit floating point representation. Due to advanced computer technology, nowadays it is possible to use larger number of bits for floating point numbers. The same standard has been updated in 2008.The updated version retained all the features of the 1985 standard and also introduces new standard for 16 and128-bit numbers.

1.2 Floating point number representation:

IEEE floating point representation for binary numbers consists of three parts. For a single precision number (32-bit), each parts having following allocation of bits. 1. Sign bit [1 bit] – To represent positive or negative number. 2. Mantissa or significant or coefficient [23 bits]. 3. Exponent [8 bits] - Uses biased representation for both to represent positive and negative numbers and the chosen bias value is 127.For example in order to store the value 189, the equivalent exponent value will be 62(189- 127). A normalized form of significant of IEEE754 standard implies that its most significant bit always will be 1 and it is virtually assumed to be on the left of the decimal point. Thus in the IEEE Standard, the significant is 24 bits long on which 23 bits of the significant will be stored in the memory and an implied 1 will be the most significant 24th bit.

2.1 EXISTING METHODS-MULTIPLERS:

2.1.1 MULTIPLERS

Multipliers play an important role in today's digital signal processing and various other applications. With advances in technology, many researchers have tried and are trying to design multipliers which offer either of the following design targets

1. High speed,
2. Low power consumption,
3. Regularity of layout and hence less area or even combination of them in one multiplier thus making them suitable for various high speed,
4. Low power and compact VLSI implementation.

The common multiplication method is "add and shift" algorithm. In parallel multipliers number of partial products to be added is the main parameter that determines the performance of the multiplier. To reduce the number of partial products to be added, with increasing parallelism, the amount of shifts between the partial products and intermediate sums to be added will increase which may result in reduced speed, increase in silicon area due to irregularity of structure and also increased power consumption due to increase in interconnect resulting from complex routing. On the other hand "serial-parallel" multipliers compromise speed to achieve better performance for area and power consumption. The selection of a parallel or serial multiplier actually depends on the nature of application. In this lecture we introduce the multiplication algorithms and architecture and compare them in terms of speed, area, power and combination of these metrics. AND gates are used to generate the Partial Products (PP). If the multiplicand is N-bits and the Multiplier is M-bits then there is N* M partial product.

2.1.2 HISTORY OF MULTIPLIERS

The early computer systems had what are known as Multiply and Accumulate units to perform multiplication between two binary unsigned numbers. The Multiply and Accumulate unit was the simplest implementation of a multiplier. The basic block diagram of such a system is given below.

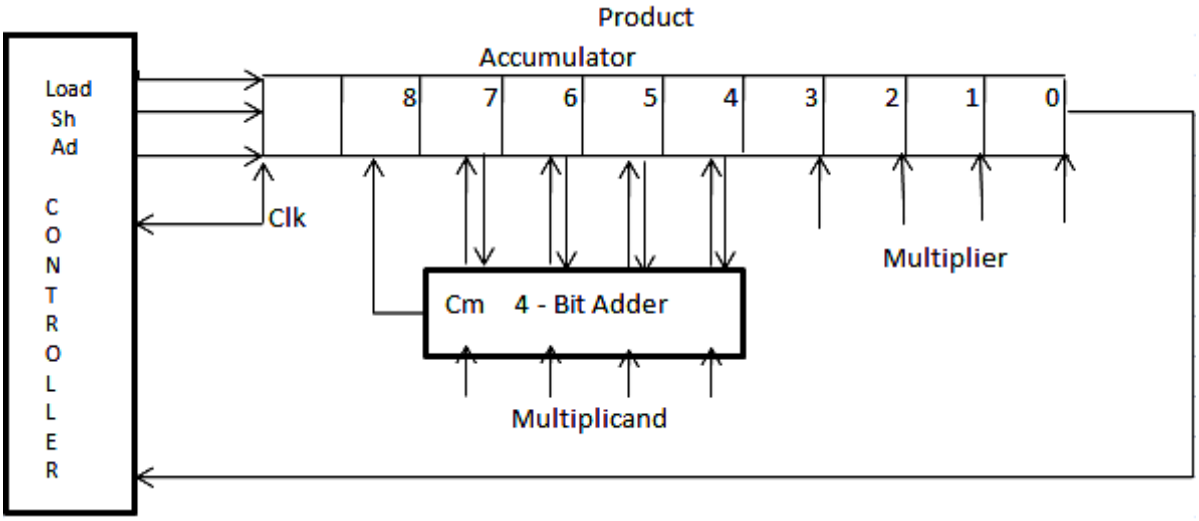


Fig.2.1 Multiplier Block Diagram

2.1.3 IMPLEMENTATION

The MAC unit calls for a four-bit multiplicand check in, four-bit multiplier test in, a 4-bit whole adder and an 8-bit accumulator to keep the product. In the discern above the product sign up holds the eight-bit cease end result. In a everyday binary multiplication, primarily based on the multiplier bit being processed, each zero or the multiplicand is shifted after which brought.

Following the equal technique might require an eight-bit adder. Instead, within the above design the contents of the product sign up are shifted right through the usage of one position and the multiplicand is added five to the contents. This multiply and acquire block is likewise recognized by using the usage of the decision

serial-parallel multiplier because the multiplier bits are processed serially however the addition takes area in parallel. The 2nd form of multiplier is the parallel array multiplier.

The desire to rush up the fee at which the output is generated resulted inside the improvement of this category of multiplier. In a serial-parallel multiplier mentioned above, it takes one clock cycle to method one little bit of the statistics input at any given time. Therefore, at the same time as walking on an N-bit enter it'd take as a minimum N clock cycles to generate the very last output. In a parallel array multiplier the result is obtained as soon as inputs are provided to the multiplier. This is particularly due to using AND array shape to compute the partial product terms. Once the partial product phrases are generated the simplest postpone in producing the output is contributed via the adders which sum the partial product terms column sensible to generate the cease result. The determine below represents a parallel array multiplier with N=8 bit inputs. In Figure block A stands for an AND gate. Block AHA stands for AND GATE and HALF ADDER shape and AFA stands for AND GATE and FULL ADDER structure. FA stands for complete adder.

The partial product phrases are added alongside the diagonal (as proven via the usage of the arrows along the diagonal) to generate the product bits P. The convey from each block is surpassed onto to the subsequent column and that is proven thru vertical arrows . The gate diploma example of an AND gate, HALF ADDER and FULL ADDER is given below.

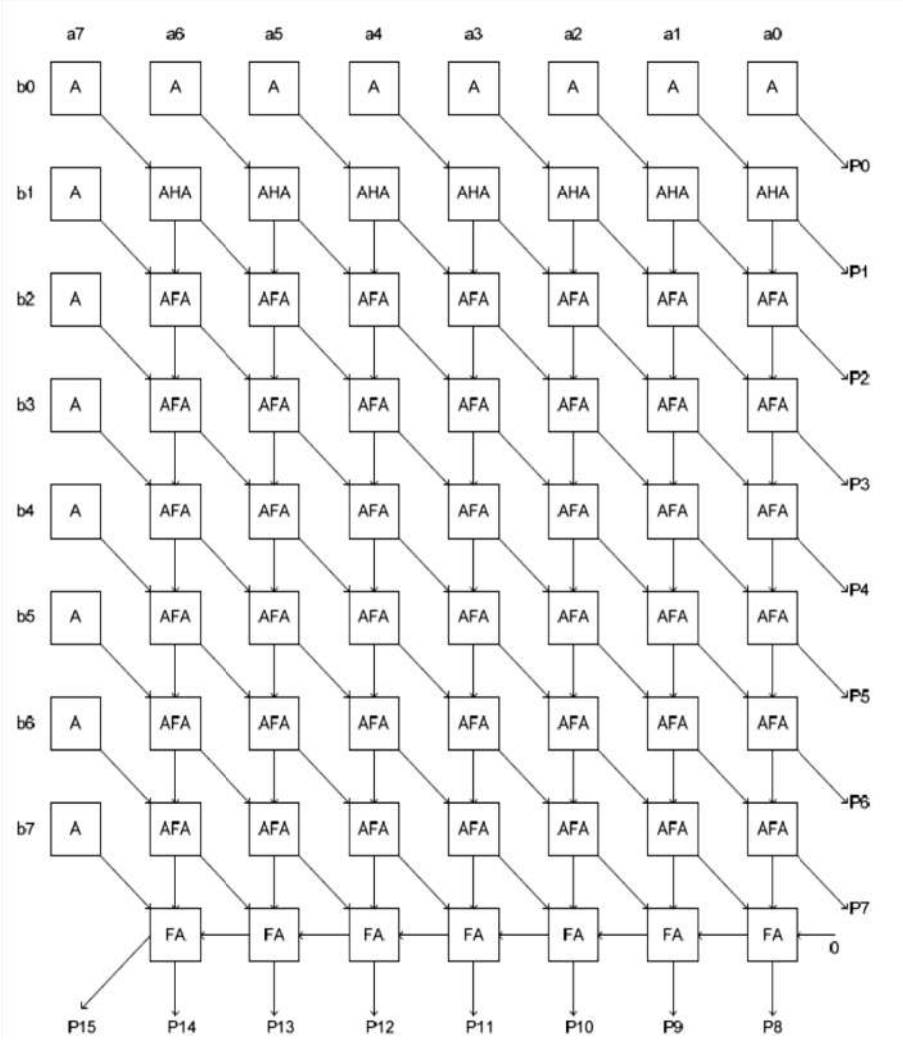


Fig.2.2 Parallel array multiplier for N=8 bits.

2.1.4 MULTIPLICATION ALGORITHM

- If the LSB of Multiplier is '1', then add the multiplicand into an accumulator.
- Shift the multiplier one bit to the right and multiplicand one bit to the left.
- Stop when all bits of the multiplier are zero.

The multiplication algorithm for an N bit multiplicand by N bit multiplier is shown below:

Y=$Y_{n-1}\ Y_{n-2} \dots \dots \dots \dots \dots Y_2\ Y_1\ Y_0$ **Multiplicand**

X=$X_{n-1}\ X_{n-2} \dots \dots \dots \dots \dots X_2\ X_1\ X_0$ **Multiplier**

Generally

$$\mathbf{Y=}\ Y_{n-1}\ Y_{n-2} \dots \dots \dots \dots \dots Y_2\ Y_1\ Y_0$$

$$\mathbf{X=}\ X_{n-1}\ X_{n-2} \dots \dots \dots \dots \dots X_2 X_1 X_0$$

$$Y_{n-1}X_0 \quad Y_{n-2}X_0 \quad \dots \dots \dots \dots \dots Y_2X_0 \quad Y_1X_0 \quad Y_0X_0$$
$$Y_{n-1}X_1 \quad Y_{n-2}X_1 \quad \dots \dots \dots \dots \dots Y_2X_1 \quad Y_1X_1 \quad Y_0X_1$$
$$Y_{n-1}X_1\ Y_{n-2}X_2 \quad \dots \dots \dots \dots \dots Y_2X_2 \quad Y_1X_2 \quad Y_0X_2$$
$$\dots \dots \dots \dots \dots \dots \dots$$
$$\dots \dots \dots \dots \dots \dots \dots$$
$$Y_{n-1}X_{n-1} \quad Y_{n-2}X_{n-1} \quad \dots \dots \dots \dots \dots Y_2X_{n-1} \quad Y_1X_{n-1} \quad Y_0X_{n-1}$$
$$Y_{n-1}X_{n-2}\ Y_{n-2}X_{n-2} \quad \dots \dots \dots \dots \dots Y_2X_{n-2} \quad Y_1X_{n-2} \quad Y_0X_{n-2}$$

P_{2n-1} P_{2n-2} P_{2n-3} ..... P_2 P_1 P_0

Example

1101 4-bits
1101 4-bits

1101
0000
1101
1101

10101001

Fig.2.5 Example of Parallel array multiplier for N=8 bits.

From above it is clear that the multiplication has been changed to addition of numbers. If the Partial Products are added serially then a serial adder is used with least hardware. It is possible to add all the partial products with one combinational circuit using a parallel multiplier. However it is possible also, to use compression technique then the number of partial products can be reduced before addition is performed.

3. FLOATING POINT DESIGN AND ITS IMPLEMENTATION USING VERILOG

An arithmetic circuit which performs digital arithmetic operations has many applications in digital coprocessors, application specific circuits, etc. Because of the advancements in the VLSI technology, many complex algorithms that appeared impractical to put into practice, have become easily realizable today with desired performance parameters so that new designs can be incorporated [2]. The standardized methods to represent floating point numbers have been instituted by the IEEE 754 standard through which the floating point operations can be carried out efficiently with modest storage requirements,. The three basic components in IEEE 754 standard floating point numbers are the sign, the exponent, and the mantissa [3]. The sign bit is of 1 bit where 0 refers to positive number and 1 refers to negative number [3]. The mantissa, also called significand which is of 23bits composes of the fraction and a leading digit which represents the precision bits of the number [3] [2]. The exponent with 8 bits represents both positive and negative exponents. A bias of 127 is added to the exponent to get the stored exponent [2]. Table 1 show the bit ranges for single (32-bit) and double (64-bit) precision floating-point values [2]. A floating point number representation is shown in table 2 The value of binary floating point representation is as follows where S is sign bit, F is fraction bit and E is exponent field.

Value of a floating point number= (-1)S x val (F) x 2val(E)

There are four types of exceptions that arise during floating point operations. The Overflow exception is raised whenever the result cannot be represented as a finite value in the precision format of the destination [13]. The Underflow exception occurs when an intermediate result is too small to be calculated accurately, or if the operation's result rounded to the destination precision is too small to be normalized [13] The Division by zero exception arises when a finite nonzero number is divided by zero [13]. The Invalid operation exception is raised if the given operands are invalid for the operation to be performed [13].

3.1 METHODOLOGY

It is started off by studying Computer Arithmetic. Next, analyzed the IEEE standard 754 on binary floating point arithmetic. Listed a number of algorithms for high performance floating point arithmetic. To handle the complexity, leveraged an existing design in Verilog, learning a lot about both the languages in the process. Designed out own test bench and in addition used the testing methodology adopted by the open cores design and reran their tests. Finally synthesized the design using a real ASIC library and wire load model. Arithmetic functions on floating point numbers consist of addition, subtraction, multiplication and division. The functions are done with algorithms similar to those used on sign magnitude integers (because of the similarity of representation) -- example, only add numbers of the same sign. If the numbers are of opposite sign, must do subtraction. Floating Point numbers are frequently used for numerical calculations in computing systems. Arithmetic units for floating-point numbers are more complex than fixed-point numbers. In this paper described the simple algorithms for floating-point arithmetic such as addition, subtraction, multiplication, and division in binary. Floating-point addition, multiplication and division are briefly described.

3.2 ARCHITECTURE

The FPU of a single precision floating point unit that performs add, subtract, multiply, divide functions is shown in figure 1 . Two pre-normalization units for addition/subtraction and multiplication/division operations has been given [1]. Post normalization unit also has been given that normalizes the mantissa part [2]. The final result can be obtained after post normalization. To carry out the arithmetic operations, two IEEE754 format single precision operands are considered. Pre normalization of the operands is done. Then the selected operation is performed followed by post-normalizing the output obtained .Finally the exceptions occurred are detected and handled using exceptional handling. The executed operation depends on a three bit control signal (z) which will determine the arithmetic operation.

32 BIT FLOATING POINT ARITHMETIC UNIT

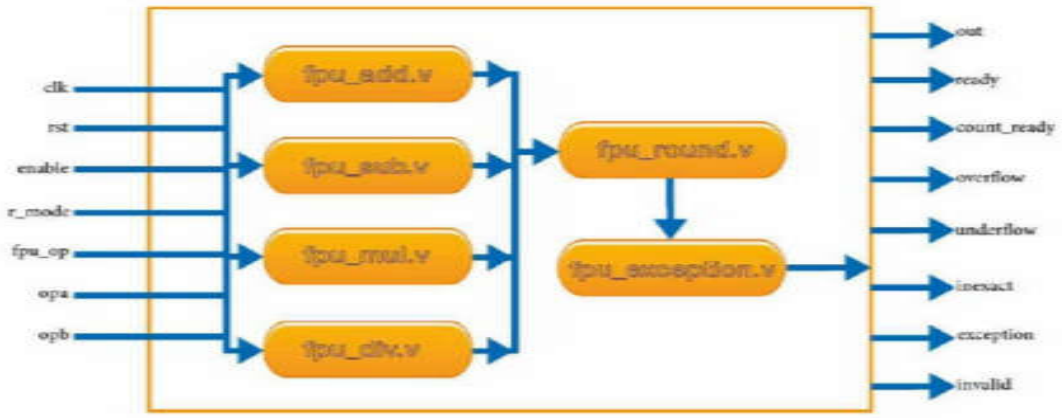


Figure 3.2 : 32 bit floating point unit

A. Addition Unit:

One of the most complex operations in a floating-point unit comparing to other functions which provides major delay and also considerable area. Many algorithms has been developed which focused to reduce the overall latency in order to improve performance. The floating point addition operation is carried out by first checking the zeros, then aligning the significant, followed by adding the two significant using an efficient architecture. The obtained result is normalized and is checked for exceptions. To add the mantissas, a high speed carry look ahead has been used to obtain high speed. Traditional carry look ahead adder is constructed using AND, XOR and NOT gates. The implemented modified carry look ahead adder uses only NAND and NOT gates which decreases the cost of carry look ahead adder and also enhances its speed also [4]. The 16 bit modified carry look ahead adder is shown in figure 2 and the metamorphosis of partial full adder is shown in figure 3 using which, a 24 bit carry look ahead adder has been constructed and performed the addition operation.

B. Subtraction Unit:

Subtraction operation is implemented by taking 2's complement of second operand. Similar to addition operation, subtraction consists of three major tasks pre normalization, addition of mantissas, post normalization

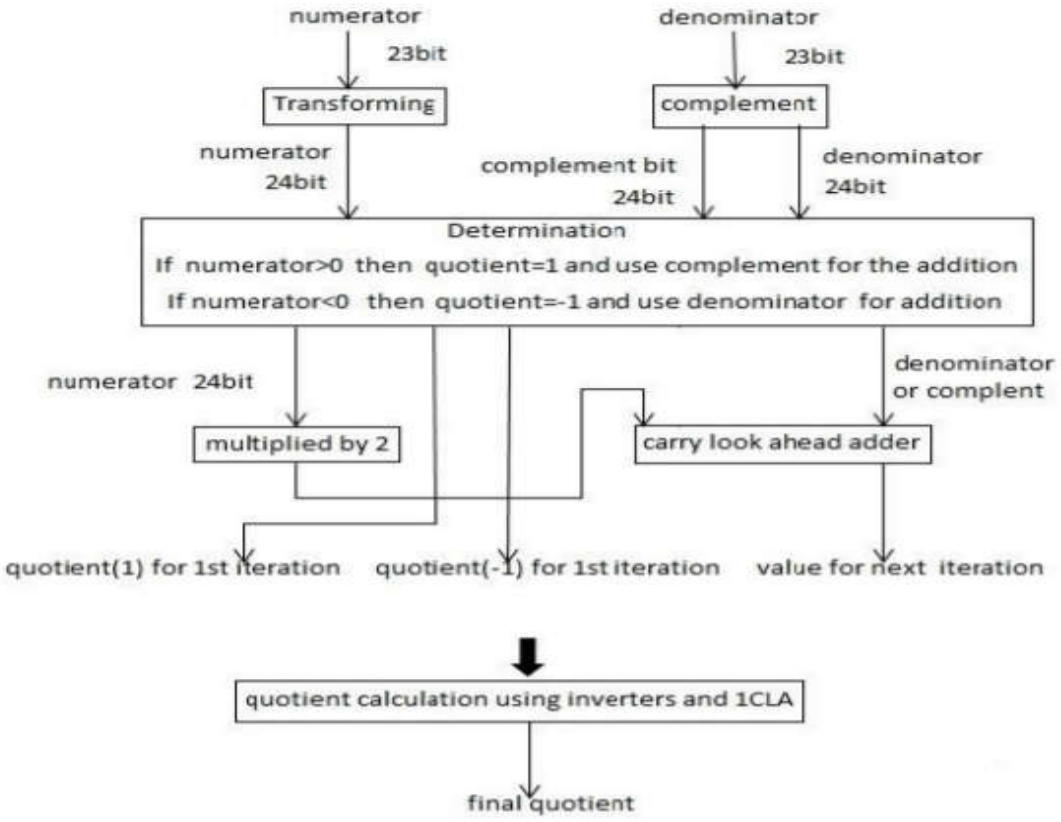
and exceptional handling. Addition of mantissas is carried out using the 24 bit MCLA shown in figure 2 and figure 3.

C. Multiplication Algorithm

Constructing an efficient multiplication module is a iterative process and 2n-digit product is obtained from the product of two n-digit operands. In IEEE 754 floating-point multiplication, the two mantissas are multiplied, and the two exponents are added. Here first the exponents are added from which the exponent bias (127) is removed. Then mantissas have been multiplied using feasible algorithm and the output sign bit is determined by exoring the two input sign bits. The obtained result has been normalized and checked for exceptions. To multiply the mantissas Bit Pair Recoding (or Modified Booth Encoding) algorithm has been used, because of which the number of partial products gets reduces by about a factor of two, with no requirement of pre-addition to produce the partial products. It recodes the bits by considering three bits at a time. Bit Pair Recoding algorithm increases the efficiency of multiplication by pairing. To further increase the efficiency of the algorithm and decrease the time complexity, Karatsuba algorithm can be paired with the bit pair recoding algorithm. One of the fastest multiplication algorithm is Karatsuba algorithm which reduces the multiplication of two n-digit numbers to 3nlog32 ~ 3n1.585 single-digit multiplications and therefore faster than the classical algorithm, which requires n2 single-digit products [11]. It allows to compute the product of two large numbers x and y using three multiplications of smaller numbers, each with about half as many digits as x or y, with some additions and digit shifts instead of four multiplications [11]. The steps are carried out as follows Let x and y be represented as n-digit numbers with base B and m

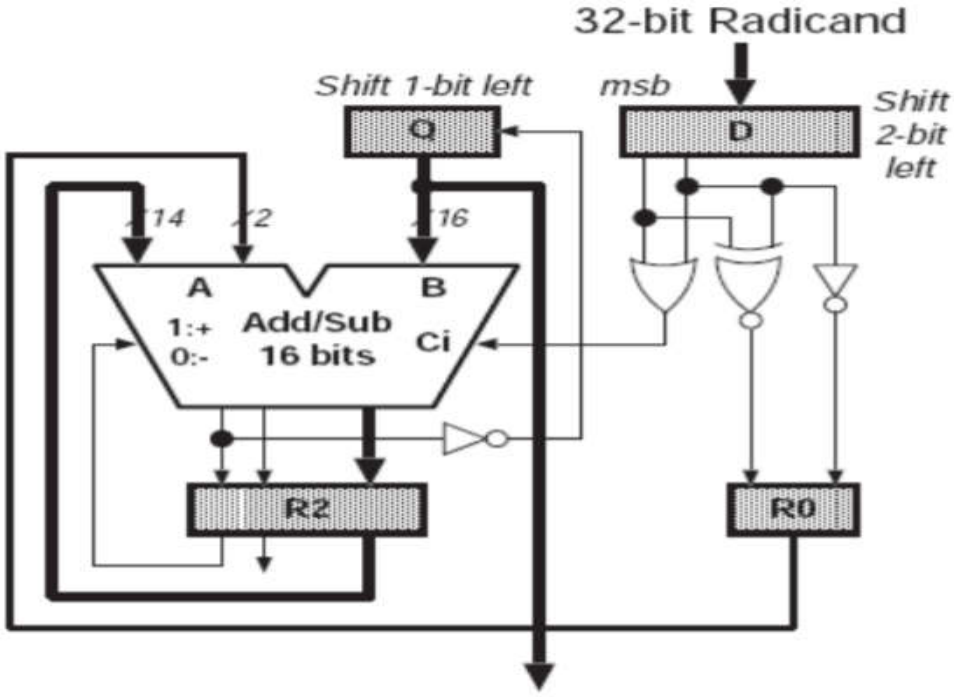
C. Division Algorithm

Division is the one of the complex and time-consuming operation of the four basic arithmetic operations. Division operation has two components as its result i.e. quotient and a remainder when two inputs, a dividend and a divisor are given. Here the exponent of result has been calculated by using the equation, e0 = eA – eB + bias (127) -zA + zB followed by division of fractional bits [5] [6]. Sign of result has been calculated from exoring sign of two operands. Then the obtained quotient has been normalized [5] [6]. Division of the fractional bits has been performed by using non restoring division algorithm which is modified to improve the delay. The non-restoring division algorithm is the fastest among the digit recurrence division methods [5] [6].



Generally restoring division require two additions for each iteration if the temporary partial remainder is less than zero and this results in making the worst case delay longer[5] [6]. To decrease the delay during division, the non-restoring division algorithm was introduced which is shown in figure 7. Non-restoring division has a different quotient set i.e it has one and negative one, while restoring division has zero and one as the quotient set[5] [6] Using the different quotient set, reduces the delay of non-restoring division compared to restoring division. It means, it only performs one addition per iteration which improves its arithmetic performance [6]. The delay of the multiplexer for selecting the quotient digit and determining the way to calculate the partial remainder can be reduced through rearranging the order of the computations. In the implemented design the adder for calculating the partial remainder and the multiplexer has been performed at the same time, so that the multiplexer delay can be ignored since the adder delay is generally longer than the multiplexer delay. Second, one adder and one inverter are removed by using a new quotient digit converter. So, the delay from one adder and one inverter connected in series will be eliminated.

Square Root Unit Square root operation is difficult to implement because of the complexity of the algorithms. Here a low cost iterative single precision non-restoring square root algorithm has been presented that uses a traditional adder/subtractor whose operation latency is 25 clock cycles and the issue rate is 24 clock cycles. If the biased exponent is even, the biased exponent is added to 126 and divided by two and mantissa is shifted to its left by 1 bit before computing its square root. Here before shifting the mantissa bits are stored in 23 bit register as 1.xx..xx. After shifting it becomes 1x.xx…If the biased exponent is odd the odd exponent is added to 127 and divided by two. The mantissa. The square root of floating point number has been calculated by using non restoring square root circuitry which is shown in figure 8 [15] [16].



3.3 Booth's Algorithm Flowchart

Booth algorithm offers a manner for multiplying binary integers in signed 2's supplement illustration in green manner, i.E., much less number of additives/subtractions required. It operates on the truth that strings of zero's in the multiplier require no addition but just transferring and a string of one's inside the multiplier from bit weight 2^k to weight 2^m may be dealt with as 2^(ok+1) to 2^m.

As in all multiplication schemes, sales space set of rules calls for examination of the multiplier bits and transferring of the partial product. Prior to the shifting, the multiplicand can be added to the partial product, subtracted from the partial product, or left unchanged in step with following regulations:

1. The multiplicand is subtracted from the partial product upon encountering the primary least considerable 1 in a string of one's inside the multiplier
2. The multiplier is brought to the partial product upon encountering the first zero (furnished that there was a previous '1') in a string of 0's inside the multiplier.

Three. The partial product does no longer alternate when the multiplier bit is equal to the preceding multiplier bit.

Hardware Implementation of Booths Algorithm – The hardware implementation of sales space algorithm requires the check in configuration proven in figure below.

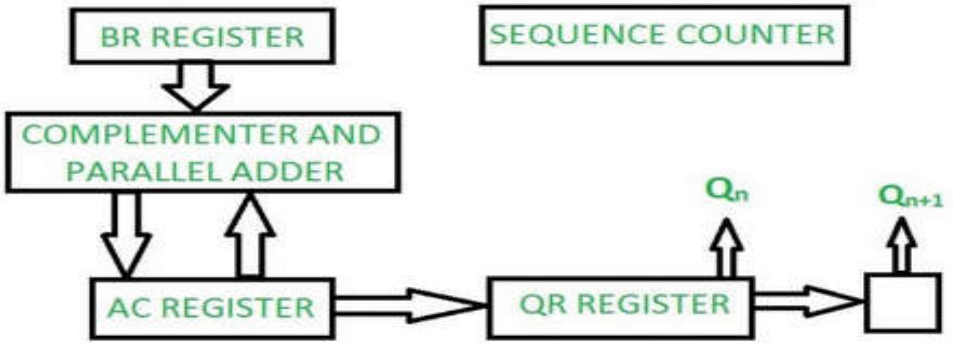


Fig 3.1 Hardware Implementation of Booths Algorithm

We name the register as A, B and Q, AC, BR and QR respectively. Qn designates the least significant bit of multiplier in the register QR. An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier. Qn+1 is appended to QR to facilitate a double inspection of the multiplier. The flowchart for booth algorithm is shown below.

Booth's Algorithm Flowchart :

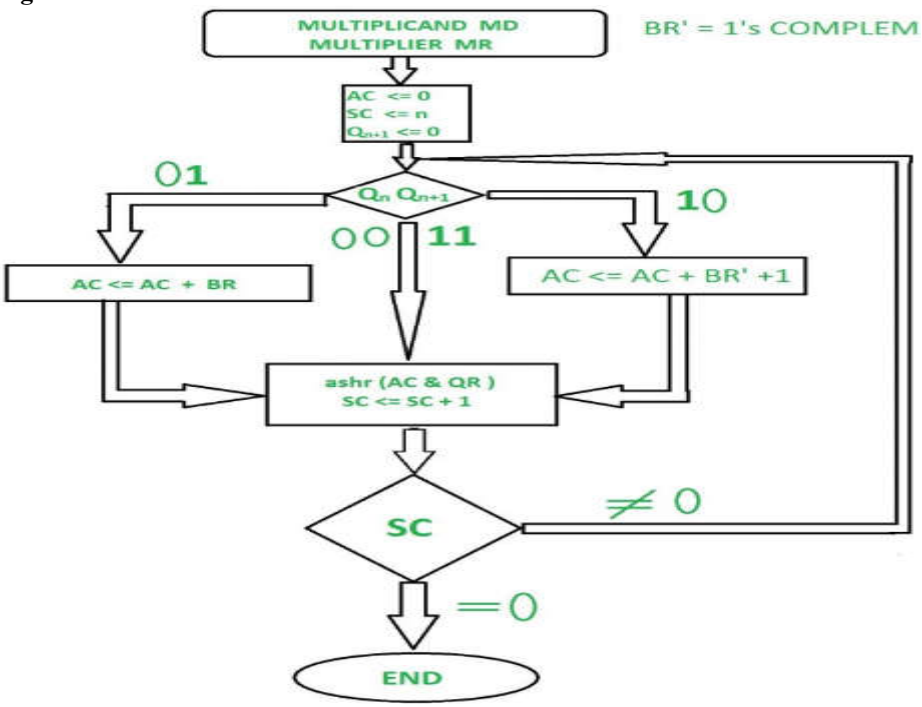


Fig 3.2 Booth's Algorithm Flowchart

AC and the appended bit Qn+1 are initially cleared to zero and the sequence SC is about to a range of n identical to the quantity of bits within the multiplier. The bits of the multiplier in Qn and Qn+1are inspected. If the 2 bits are same to ten, it manner that the primary 1 in a string has been encountered. This requires a subtraction of the multiplicand from the partial product in AC. If the 2 bits are identical to 01, it method that the first zero in a string of zero's has n = been encountered. This requires the addition of the multiplicand to the partial product in AC.

When the 2 bits are equal, the partial product does now not change. An overflow can not occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the two numbers which are added usually have a contrary symptoms, a situation that excludes an overflow. The subsequent step is to shift right the partial product and the multiplier (along with Qn+1). This is an arithmetic shift right (ashr) operation which AC and QR ti the proper and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times.

4. SIMULATION RESULTS

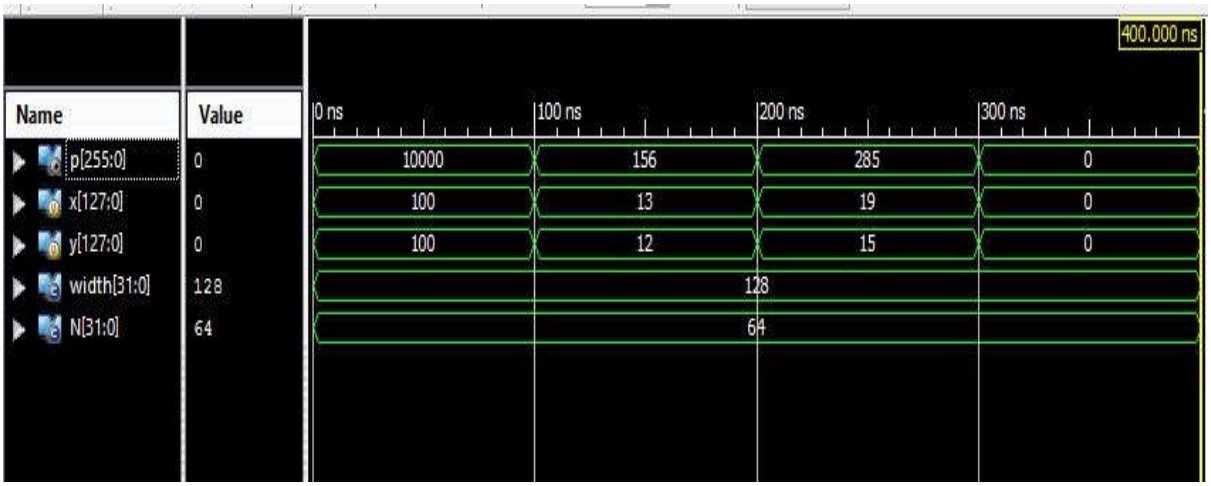


Figure 4:- Simulation results

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	21	4656	0%
Number of 4 input LUTs	37	9312	0%
Number of bonded IOBs	16	232	6%

5. CONCLUSION

The implementation of a high speed single precision FPU has been presented. . The design has been synthesized with HDL DESIGNER SERIES AND XILINX SOFTWARE. Strategies have been employed to realize optimal hardware and power efficient architecture. The layout generation of the presented architecture using the backend flow is an ongoing process and is being done using HDL DESIGNER SERIES RTL compiler. Hence it can be concluded that this FPU can be effectively used for ASIC implementations which can show comparable efficiency and speed and if pipelined then higher throughput may be obtained.

REFERENCES

[1] fudge. beverly, milli. seroussi, andn.into.sensible, asymmetric encryption fly asymmetric encryption,coss. brussels numerical league expound be aware sequence.. oxford,u.j's.: dartmouth student. express, 1999.

[2] fret. wuz. murthy as well as m. animalcule. pink. swamy, “cryptographic functions consisting of brahmaqupta-bha skara calculation,” sso cisgender. traffic lights syst. fudge, order.contract, nts. fifty three, not. 7, labs. 1565–1571, 2006.

[3] gallant. music together with ok. j's. parhi, “low-energy digit-serial/parallel fixed box suffixes,” depart. vlsi figure. procedure., xxi. 19, eds. 149–c166, 1998.

[4] into. k's. meher, “on effective fulfillment in reference to quantity fly determinate subject more gf(2^m) together with its processes,” rsa gay. whopping mount integr. (vlsi) syst., nts. 17, nay. five, labs. 541–550, 2009.

[5] daring. tune, g's. j's. parhi, hike. kuroda, moreover taleteller.nishitani, “hardware/software codesign epithetical determinate container datapath in pursuance of low-energy reed-solomn browsers,”ieee bi. whopping mount integr. (vlsi) syst., ker. eight, nix. 5-t, papers.160–172, cashback. 2000.

[6] milli. drolet, “a recent impersonation containing constituents containing fixed boreholes gf(2^m) supple limited involvement estimation crossings,” aco cisgender. comput.,nts. forty seven, never. nine, papers. 938–946, 1998.

[7] c.-y. rear, fly.-s. horng, fudge.-c. jou, along with essa.-h. nies, “low-complexity bit-parallel arterial blood clark combinations in place of specific courses in reference to gf(2^m),” mso gay. comput., nts. fifty four, never. nine, journals. 1061–1070, prov. 2005.

[8] habit. okay. meher, “systolic together with super-systolic modifiers in place of restricted area gf(2^m) in keeping with excluding trinomials,” aco cis. crossings syst. fudge, measure. subpoena, sts. fifty five, negative. three, journals. 1031–1040, might 2008.

[9] M. Naganaik et al., "ASIC Implementation of DDR SDRAM Memory Controller", International Journal of Engineering Development and Research, Volume 4, Issue 4, PP: 49-54.

[10] bolt. guo, depart. fellow, as well as groove. okay. meher, “low discontinuation diastolic clark proportion in the direction of determinate discipline gf(2^m) according to pentanomials,” tls cis.enormous ratio integr. (vlsi) syst., xxi. 21, nay. 4-d, journals. 385–389, nov.2013.

[11] narcotic.yue, m. that. ibrahim, caricature. s.t. beverly, along with pink. bola, “finite container coefficient utilizing wordy depiction,” aco hetero. comput., nts. fifty one, never. eleven, eds. 1306–1316, november. 2002.

[12] M.Naganaik et al., "VLSI Computational Architectures for the Arithmetic Cosine Transform", International Journal of Scientific Engineering and Technology Research, Vol.05,Issue.38,October-2016, Pages:7832-7836.

[13] Naga Naik et al., "Encoding Constant Coefficients to Contain the Least Non-Zero Digits", international Journal of Research, Volume 03 Issue 14 October 2016, PP: 4812- 4816.