# ASIC Implementation of a High Speed Double (64bit) Precision Floating Point Arithmetic unit

**A. Srihitha[1], Syed KareemSaheb[2], J.Prasad Babu[3]**

**PG scholar, Dept of ECE(VLSI), ASIT, Gudur, AP state, E-mail:srihitha.arani@gmail.com**

**Research scholar, Dept of ECE, ANU, Guntur, AP state, E-mail:sd_kareem@yahoo.com**

**Associate Professor, Dept of ECE, ASIT, Gudur, AP state, E-mail:prasadsupriya1983@gmail.com**

*Abstract: To represent very large or small values, large range is re-quired 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 addi-tion, 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 la-tency, and if pipelined then higher throughput. The algo-rithms are modeled in Verilog HDL and the RTL code for adder, subtractor, multiplier, divider, square root are syn-thesized using Cadence RTL complier where the de-sign is targeted for 180nm TSMC technology with proper constraints.*

## I.INTRODUCTION

An arithmetic circuit which performs digital arithmetic operations has many applications in digital coprocessors, application specific circuits, etc. Because of the advance-ments in the VLSI technology, many complex al-gorithms that appeared impractical to put into practice, have become easily realizable today with desired perfor-mance 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 man-tissa [3]. The sign bit is of 1 bit where 0 refers to positive number and 1 refers to negative number [3]. The man-tissa, 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 $2val(E)$

## TABLE 1 BIT RANGE FOR SINGLE (32-BIT) AND DOUBLE (64-BIT) PRECISION FLOATING-POINT VALUES

|  | Sign | Exponent | Fraction | Bias |
|---|---|---|---|---|
| Single precision | 1[31] | 8[30-23] | 23[22-00] | 127 |
| Double precision | 1[63] | 11[62-52] | 52[51-00] | 1023 |

## TABLE 2    FLOATING POINT NUMBER REPRESENTATION

| 32 bits | | |
|---|---|---|
| sign | exponent | mantissa |
| 1 bit | 8 bits | 23 bits |

There are four types of exceptions that arise during float-ing 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 ex-ception 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].In this paper, ASIC implementation of a high speed FPU has been carried out using efficient addi-tion, subtraction, multiplication, division algorithms. Sec-tion II depicts the architecture of the floating point unit and methodology, to carry out the arithmetic operations. Section III presents the arithmetic operations that use ef-

**National Conference on Emerging Trends in Information, Digital & Embedded Systems(NC'e-TIDES-2016)**

*International Journal of Advanced Trends in Engineering, Science and Technology(IJATEST)Volume.4,Special Issue.1Dec.2016*

ficient algorithms with some modifications to improve la-tency. Section IV presents the simulation results that have been simulated in Cadence RTL compiler using 180nM process. Section V presents the conclusion.

## II.PROPOSED WORK

## ARCHITECTURE AND METHODOL-OGY:

The FPU of a single precision floating point unit that per-forms add, subtract, multiply, divide functions is shown in figure 1 [1]. Two pre-normalization units for addition/ subtraction and multiplication/division operations has been given [1]. Post normalization unit also has been giv-en that normalizes the mantissa part [2]. The final result can be obtained after post- normalization. To carry out the arithmetic operations, two IEEE-754 format single preci-sion operands are considered. Pre- normalization of the operands is done. Then the selected. handled using ex-ceptional handling. The executed operation depends on a three bit control signal (z) which will determine the arith-metic operation is shown in table 3.
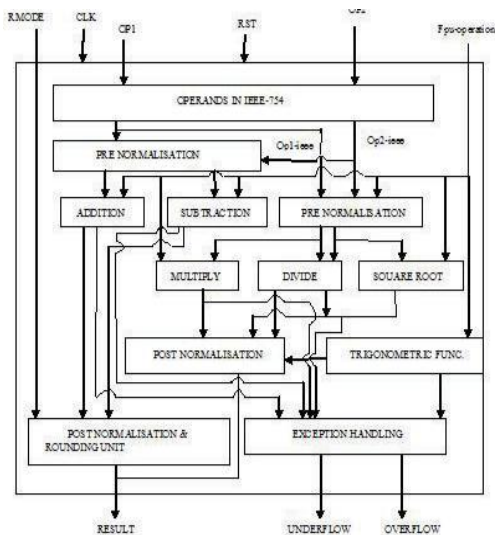


**Figure 1. Block Diagram of floating point arithmeticunit [1]**

## TABLE 3 FLOATING POINT UNIT OPER-ATIONS

| z(control signal) | Operation |
|---|---|
| 2'b000 | Addition |
| 2'b001 | Subtraction |
| 2'b010 | Multiplication |
| 2'b011 | Division |
| 2'b100 | Square root |

**32 BIT FLOATING POINT ARITHME-**

## TIC 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 la-tency in order to improve performance. The floating point addition operation is carried out by first checking the zeros, then aligning the significand, followed by adding the two significands using an efficient architecture. The obtained result is normalized and is checked for excep-tions. 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.
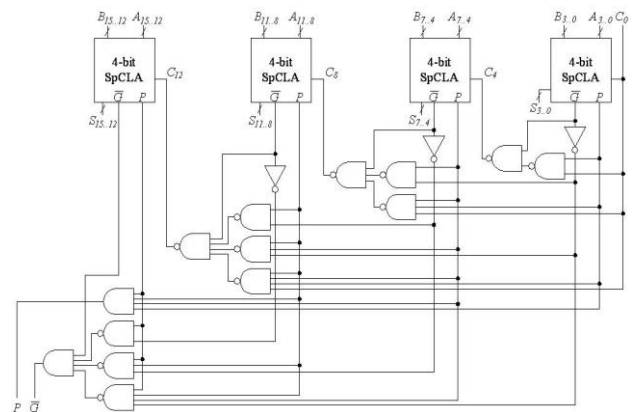


**Figure 2. 16 bit modified carry look ahead adder [4]**
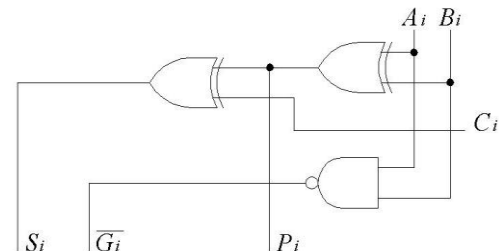**Figure 2. 16 bit modified carry look ahead adder [4]**



**Figure. 3 Metamorphosis of partial full adder [4]**

### B. Subtraction Unit:

**National Conference on Emerging Trends in Information, Digital & Embedded Systems(NC'e-TIDES-2016)**

*International Journal of Advanced Trends in Engineering, Science and Technology(IJATEST)Volume.4,Special Issue.1Dec.2016*

Subtraction operation is implemented by taking 2's com-plement of second operand. Similar to addition operation, subtraction consists of three major tasks pre nor-malization, 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 it-erative 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 algo-rithm and the output sign bit is determined by exoring the two input sign bits. The obtained result has been nor-malized and checked for exceptions.To multiply the man-tissas 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 effi-ciency of the algorithm and decrease the time complexity, Karatsuba algorithm can be paired with the bit pair recod-ing algorithm. One of the fastest multiplication algorithm is Karatsuba algorithm which reduces the multiplication of two n-digit numbers to $3n\log 32 \sim 3n1.585$ single-dig-it 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 followsLet x and y be represented as n-digit numbers with base B and m<n.

$$x = x1Bm + x0 \quad y = y1Bm + y0$$

Where x0 and y0 are less than Bm [11]. The product is then xy

$$= (x1Bm + x0)(y1Bm + y0)= c1B2m + b1Bm + a1 \text{ Where } c1 = x1y1$$

$$b1 = x1y0+ x0y1 \quad a1 = x0y0.$$

$$b1 = p1- z2 - z0$$
$$p1 = (x1 + x0)(y1 + y0)$$

Here c1, a1, p1 has been calculated using bit pair recoding algorithm. Radix-4 modified booth encoding has been used which allows for the reduction of partial product array by half [n/2]. The bit pair recoding table is shown in table 3. In the implemented algorithm for each group of three bits (y2iþ1, y2i, y2i_1) of multiplier, one partial product row is generated according to the encoding in table 3. Radix-4 modified booth encoding (MBE) signals and their respec-tive partial products has been generated using the figures4 and 5. For each partial product row, figure 4 generates the one, two, and neg signals.

These values are then given to the logic in figure 5 with the bits of the multiplicand, to produce the whole partial product array. To prevent the sign extension the obtained partial products are extended and the the product has been calcu-lated using carry save select adder.

## TABLE 3:BIT-PAIR RECODING [11]
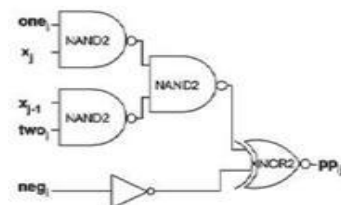


**Figure 4 MBE signal generation [10]**

| BIT PATTERN | | | | OPERATION |
|---|---|---|---|---|
| 0 | 0 | 0 | NO OPERATION | |
| 0 | 0 | 1 | 1xa | prod=prod+a; |
| 0 | 1 | 0 | 2xa-a | prod=prod+a; |
| 0 | 1 | 1 | 2xa | prod=prod+2a; |
| 1 | 0 | 0 | -2xa | prod=prod-2a; |
| 1 | 0 | 1 | -2xa+a | prod=prod-a; |
| 1 | 1 | 0 | -1xa | prod=prod-a; |
| 1 | 1 | 1 | NO OPERATION | |
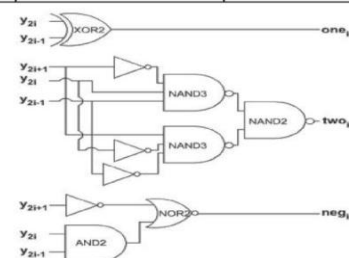


**Figure 5 Partial Product Generation [10]**

## D. Division Algorithm:

**National Conference on Emerging Trends in Information, Digital & Embedded Systems(NC'e-TIDES-2016)**

*International Journal of Advanced Trends in Engineering, Science and Technology(IJATEST)Volume.4,Special Issue.1Dec.2016*

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. quo-tient 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 ex-oring 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 re-storing 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 divi-sion algorithm was introduced which is shown in figure 6. 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 select-ing 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 ig-nored 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.
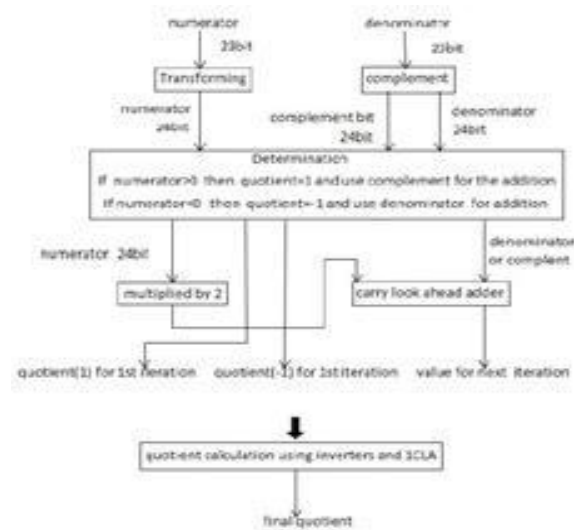


**Figure 6. Non Restoring Division algorithm**

### E. 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/sub-tractor 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 itsleft 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 7 [15] [16].

**National Conference on Emerging Trends in Information, Digital & Embedded Systems(NC'e-TIDES-2016)**

*International Journal of Advanced Trends in Engineering, Science and Technology(IJATEST)Volume.4,Special Issue.1Dec.2016*

**Figure 7. Non Restoring square root circuitry [15] [16]**

### III.RESULTS

## A. Addition Unit:

The single precision addition operation has been imple-mentation in cadence RTL compiler. Here for the inputs, input1=25.0 and input2=4.5 the result has been obtained as 29.5 and is shown in figure 8.



**Figure 8. Implementation of 32 bit Addition operation**

## B. Subtraction Unit:

The single precision addition operation has been imple-mentation in cadence RTL compiler. Here for the inputs, input1=25.0 and input2=4.5 the result has been obtained as 20.5 and is shown in figure 9



**Figure 9. Implementation of 32 bit Subtraction operation**

## C. Multiplication Unit:

The single precision multiplication operation has been implementation in cadence RTL compiler. For the inputs in_sign1=1'b0,in_sign2=1'b0;in_exp1=8'b10000011,in_exp2=8'b1000 0010,in_mant1=23'b00100,in_mant2=23'b00110, the output obtained is out_sign=1;b0,out_exp=8'b10000011,out_mant=23'b00101011and the simulation waveforms are shown in figure 10.



**Figure 10. Implementation of 32 bit Multiplication operation**

## D .Division Unit:

The single precision division operation has been implementation in Cadence RTL compiler. For the inputs, input1=100.0 and input2=36.0 quotient has been obtained as 2.0 and the remainder as 28.0 andis shown in figure 11.

**National Conference on Emerging Trends in Information, Digital & Embedded Systems(NC'e-TIDES-2016)**

*International Journal of Advanced Trends in Engineering, Science and Technology(IJATEST)Volume.4,Special Issue.1Dec.2016*

**Figure 11. Implementation of 32 bit Division operation**

## D.Square Root Unit:

The single precision square root operation has been implementation in Cadence RTL compiler. For the input, input=144.0, square root has been obtained as 12.0 and is shown in figure 12.
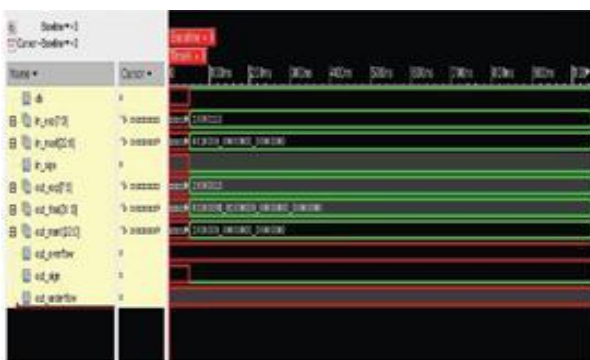


**Figure 12. Implementation of 32 bit Division operation**

## IV. CONCLUSION

The implementation of a high speed single precision FPU has been presented. . The design has been synthesized with TSMC 0.18 I-lm Logic Salicide 1.8V/3.3V 1 P6M process technology. 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 Cadence RTL compiler with 180nM process technology. 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] (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 AboutFloating-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 Engineer-ing, Huafan University.

[5] Prof. Kris Gaj, Gaurav, Doshi, Hiren Shah, "Sine/Co-sine using CORDICAlgorithm".

[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: Digit- Recurrence Algorithms and Imple-mentations, Boston: Kluwer Academic Publishers, 1994.

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

[9] Behrooz Parhami, Computer Arithmetic - Algo-rithms and Hardware Designs, Oxford: Oxford Univer-sity Press, 2000.

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

[11] D. J. Bernstein. Multidigit Multiplication for Mathe-maticians. Advances in Applied Mathemat-ics, to appear

[12] A. Karatsuba and Y. Ofman. Multiplication of Mul-tidigit Numbers onAutomata. Soviet Physics- Doklady, 7 (1963), 595-596.

### BIOGRAPHY

A. Srihitha , B tech: Brahmaiah College of Engineering (ECE), M tech(VLSI): Audisankara institute of technology, Gudur.

Syed Kareem Shaheb, Research scholar, Dept of ECE, ANU, Guntur, AP state.

*J. PrasadBbabu, Professor,Dept of ECE, ASIT, Gudur, AP state.*