# Implementation of 32 Bit Floating Point MAC Unit to Feed Weighted Inputs to Neural Networks

Yadagiri Karri[#], Prof. Rajesh Misra[*]

[#]*Department of ECE, CUTM, JITM*

***Abstract*- This paper describes an FPGA implementation of IEE-754 format single precision floating point MAC unit that is used in artificial neural networks to feed the weighted inputs to the neurons. Use of floating point numbers improves the range of the representation of data from very small number to a very large number which is mostly recommended for Artificial Neuron Networks.**

***Keywords*- *FPGA, IEEE-754, floating point MAC, weighted inputs, Artificial Neural Networks.***

## I. INTRODUCTION

The main goal of this paper is designing of floating point MAC unit. Representing real numbers in binary format requires floating point numbers .In this paper floating point numbers are represented according to IEEE 754 standard format. In single precision, a Floating Point number consists of a 32-bit word divided in 1 bit for sign, 8 bits for exponent that constitutes a 127 bias, and 23 bits for the significand. This standard supports two types of formats binary interchanges format and decimal interchange format. In many applications computation is done using floating point arithmetic. Earlier floating point operations were mainly implemented as software while for main stream general purpose processor hardware implementation was an option because cost of the hardware was not reasonable. Today every microprocessor is hardware specific for handling various floating point operations. In artificial neural network applications floating point MAC unit is required in order to achieve desired performance. But because of the advancements in reconfigurable logic now these Mac units can be implemented on FPGA. The goal of this project is FPGA implementation of floating point MAC unit for ANN applications. Floating point number can be given by equation (1):

$$Z= (-1^{s}) *2^{(exp-bias)}*(1*M) \qquad (1)$$

Equation 1 represents IEEE 32 bit single precision floating point format. One very important requirement of the IEEE-754 representation is that the number should be represented with it closest equivalent for the precision chosen, which means that it is assumed that any operation is performed with infinite precision Any floating point number is first of all converted into this format (1) and further operations are performed. Floating-point (FP) addition is based on sequence of mantissa operations swap, shift, add, normalize, and round. A floating point adder first compares the exponents of the two input operands, swaps and shifts the mantissa of the smaller number to get them aligned. The number has to be adjusted if the incoming number is negative. Finally, the sum is renormalized, Exponents are adjusted accordingly, and resulting mantissas are truncated by an appropriate rounding scheme. If extra speed is required then FP adders use leading-zero anticipatory (LZA) logic to carryout pre-decoding for normalization shifts in parallel with the mantissa addition. Floating point multiplication basically involves xoring of the signs, multiplication of significands and adding exponents of both the numbers. After addition the exponent is called tentative result exponent. Then we have to subtract bias from added exponents. Result can be a normalized number if the MSB is 1. In this paper floating point multiplier and floating point adder/subtractor are designed and an accumulator is designed. Then floating point MAC unit is designed.MAC basically consists of adder, multiplier and an accumulator certain values are used for special number representation, as follows. If the exponent is 0 and mantissa is 0 then the number is a zero number. If the exponent is 0 and mantissa is greater than 0 then the number is a subnormal number. If the exponent lies in between 0 and 255 and mantissa is greater than 0 then the number is a normal number. If the exponent is 255 and mantissa is 0 then the number is an infinite number. If the exponent is 255 and the mantissa is greater than 0 then the number is not a number.

Table 1. Special Numbers

| S.No | Exponent | Mantissa | Output |
|---|---|---|---|
| 1 | =0 | =0 | Zero |
| 2 | =0 | >0 | Subnormal |
| 3 | 0<E<255 | >0 | Normal |
| 4 | =255 | =0 | Infinity |
| 5 | =255 | =0 | NAN |

The concept of ANN is basically introduced from the subject of biology where neural network plays a important and key role in human body. In human body work is done with the help of neural network. Neural Network is just a web of inter connected neurons which are millions and millions in number. With the help of these interconnected neurons all the parallel processing is done in human body and the human body is the best example of Parallel Processing. An Artificial Neuron is basically an engineering approach of biological neuron. It has device with many inputs and one output. ANN consists of large number of simple processing elements that are interconnected with each other and also layered. Similar to biological Neuron Artificial Neural Network also have neurons which are artificial and they also receive inputs from the other elements or other artificial neurons and then after the inputs are weighted and added, the result is then transformed by a transfer function into the output. The transfer function may be anything like Sigmoid, hyperbolic tangent functions or a step.

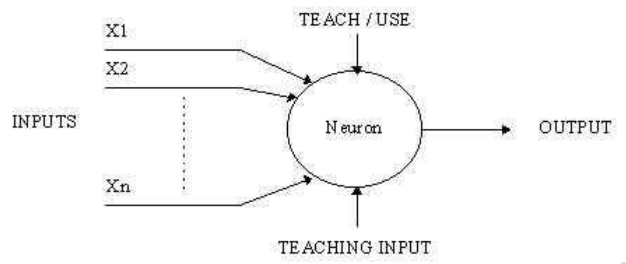

Fig 1 Artificial Neuron

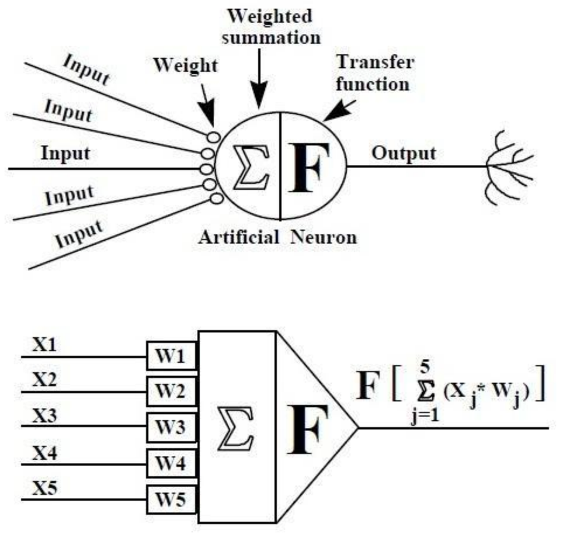

Fig 2 Functions of an Artificial Neuron

## II. RELATED WORK

Guillermo Marcus presents a multiplier and an adder/subtractor for single precision floating point numbers in IEEE format. They have pipelined architecture which are implemented in VHDL. Mohamed Al-Ashrafy presents a floating point multiplier In IEEE single precision floating point format. The multiplier does not implement rounding and it just presents the significand multiplied result. Carlos Minchola has presented FPGA implementation of a Decimal Floating Point (DFP) Adder/Subtractor. Lamiaa S. A. Hamid [10] has presented a high speed generic Floating Point Unit (FPU) consisting of a multiplier and adder/Subtractor units is proposed. A novel multiplication algorithm is proposed and used in the multiplier implementation.

## III. ORGANIZATION OF WORK

In this section IEEE 754 single precision format, floating point adder, Floating point multiplier and floating point MAC unit are explained. MAC consists of a multiplier and an accumulator unit. Multiplier will multiply two numbers and result will be added to the number already stored in the accumulator.

### *3.1. STANDARD IEEE 754 FORMAT*

The standard binary floating point format was issued by IEEE in 1985 [6]. It covers different types of floating-point formats (e.g. single, double), special coding representations (e.g. 0, $+\infty$, $-\infty$), rounding mechanisms, arithmetic operations, etc. The standard radix-2 binary floating-point representation can be written as in equation1 with s as the sign bit, M as the mantissa or fraction.

| S Bit | Exponent | Mantissa |  |
|---|---|---|---|
| 32 | 31 | 23 | 0 |

Figure 3. IEEE single precision floating point format

### *3.2. FLOATING POINT ADDER*

The computation is done in four major steps:

1. *Sorting:* puts the number with the larger magnitude on the top and the number with the smaller magnitude on the bottom .

**2.** *Alignment:* aligns the two numbers so they have the same exponent. This can be done by adjusting the exponent of the small number to match the exponent of the big number. The significand of the small number has to shift to the right according to the difference in exponents.

3. *Addition/subtraction:* adds or subtracts the significands of two aligned numbers.

4. *Normalization:* adjusts the result to normalized format. Three types of normalization procedures may be needed:

i) After a subtraction, the result may contain leading zeros in front.
***ii)*** After a subtraction, the result may be too small to be normalized and thus needs to be converted to zero.
***iii)*** After an addition, the result may generate a carry-out bit.

During alignment and normalization, the lower bits of the significand will be discarded when shifted out. The design is divided into four stages, each corresponding to a step in the foregoing algorithm. The circuit in the first stage compares the magnitudes and find the larger number and the smaller number.. The comparison is done between expl&fracl and exp2&f rac2. It implies that the exponents are compared first, and if they are the same, the significands are compared.

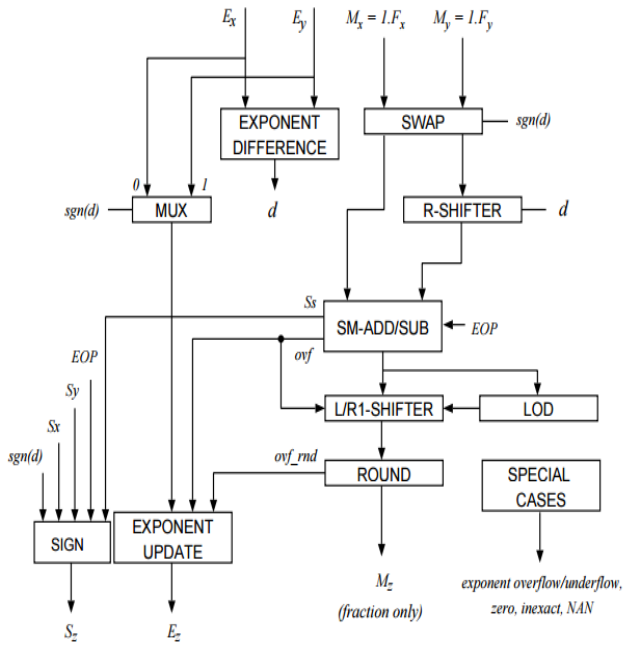

Figure 4 Block Diagram for floating point addition

The circuit in the second stage performs alignment. It first calculates the difference between the two exponents, which is expb - exps, and then shifts the significand, fracs, to the right by this amount. The circuit in the third stage performs sign-magnitude addition.Note that the operands are extended by 1 bit to accommodate the carry-out bit.

The circuit in the fourth stage performs normalization, which adjusts the result to make the final output conform to the normalized format. The normalization circuit is constructed in three segments. The first segment counts the number of leading zeros. It is somewhat like a priority encoder. The second segment shifts the significands to the left by the amount specified by the leading-zero counting circuit. The last segment checks the carry-out and zero conditions and generates the final normalized number.

## *3.3. FLOATING POINT MULTIPLIER*

Given two floating point numbers n1, n2 and after multiplication the result is n.

$$\begin{aligned} n &= n1*n2 \\ &= (-1)^{s1}.n1.2^{e1}*(-1)^{s2}.n2.2^{e2} \\ &= (-1)^{s1}+s2.p1.p2.2^{e1+e2} \end{aligned}$$

In Figure 5 we present a general multiplier block diagram. The sign, exponent and mantissas are extracted from both the numbers respectively. Pipelining has been used for designing multiplier. The sign bits of both the numbers are xored. The 8 bit exponents are added and then bias is subtracted from it. Subtraction is easily achieved by adding carry in to the sum and then subtracting 128 from it by complementing most significant bit. For multiplying significands 48 bit multiplier is used. The 28 bits are considered as the most significand bits out of which 24 bits are the mantissa bits, 3 bits are for proper rounding, 1 bit is for range overflow. The result is then normalized for proper approximation to closest value. The approximation consists of a possible single bit right shift and corresponding exponent is incremented depending on b1 bit. The resultant sign, exponent and mantissas are obtained. The resultant sign, exponent and mantissas are then obtained. The figure shown below is simple floating point multiplier. In this paper three floating point multipliers have been designed using carry save, carry look ahead, ripple carry adder. Same flow is used for all of them .only for addition of exponents different adders are used.

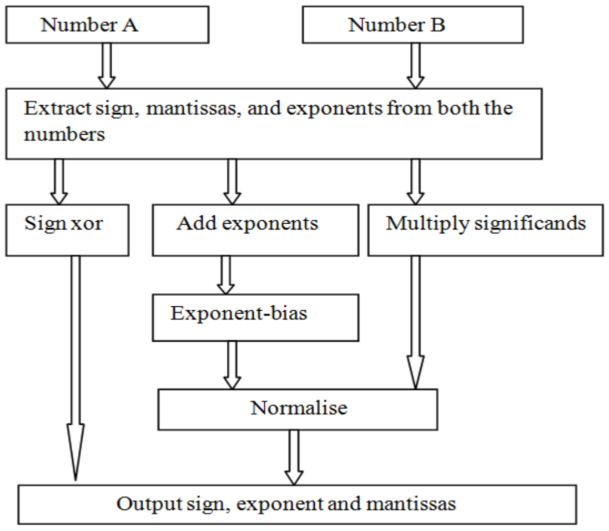
Number A
Number B
Extract sign, mantissas, and exponents from both the numbers
Sign xor
Add exponents
Multiply significands
Exponent-bias
Normalise
Output sign, exponent and mantissas

Figure 5 Block Diagram for floating point multiplier

## *3.4. FLOATING POINT MAC UNIT*

Basically composed of adders, multiplier and an accumulator. The inputs which are given to the MAC are fetched from memory location and fed to the multiplier of block of MAC which performs multiplication and gives the result back to adder which will accumulate the result and store it in a memory location. Complete process is achieved in a single cycle. The design consists of 32 bit floating point adder and 1 register for memory location. A typical MAC unit consists of multiplier, adder and accumulator. The most important feature that differentiates general processor from digital signal processor is it's multiply and accumulate unit. Each DSP algorithm would require some form of Multiplication and accumulation system. This one is the most important block in DSP systems. Usually adders that are used are carry save, carry select, ripple carry adders because of their speed. The inputs of MAC are supposed to be fetched from memory location and then they are fed to the multiplier. Multiplier will multiply the inputs and it will give the results back to the adder and then the results of the multiplier are added to the previously accumulated results. Computation of most important formula i.e. b (n) x (n-k) is easily solved by this operation.

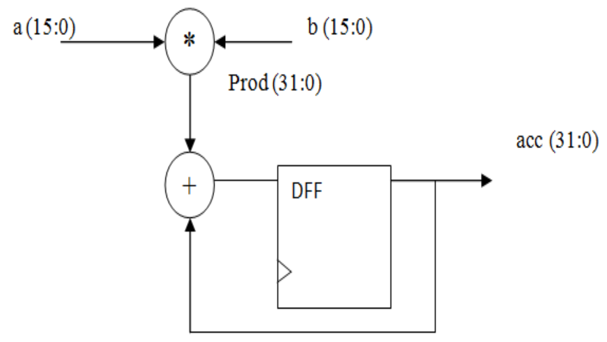

Figure 6 Block Diagram for floating point MAC

## IV. RESULTS

The proposed Mac unit is implemented on Xilinx ISE design suite 9.2 in Vertex 2P family. The device utilization factor for the floating point MAC unit is given below.

Table 2. Device Utilization Summary of FPMAC

| Logic utilization | Used | Available | Utilization |
|---|---|---|---|
| Number of slices | 610 | 1408 | 43% |
| Number of 4 input LUTs | 1123 | 2816 | 39% |
| Number of bonded IOBs | 99 | 140 | 70% |
| Number of MULT 18x18s | 4 | 12 | 33% |

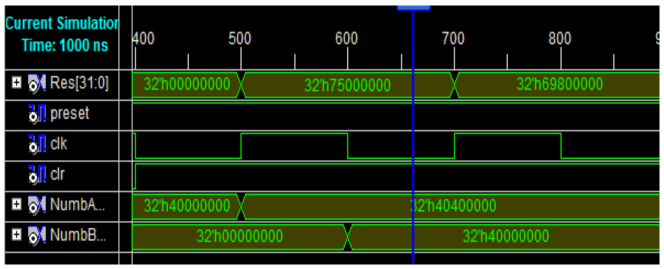

Figure 7.Simulation of Floating point MAC unit.

## V. CONCLUSION

A FP adder and a FP multiplier are presented in this paper. Both are available in pipeline architectures and they are implemented in VHDL, are fully synthesizable.

Then a complete MAC unit is designed using floating point adder and multiplier and its FPGA implementation is done. This MAC Unit is used to feed the inputs to a neuron in Artificial Neural Networks.

## REFERENCES

[1]. Mohamed Al-Ashrafy, Ashraf Salem and Wagdi Anis, “An Efficient Implementation of Floating Point Multiplier, “proceeding of 2011 IEEE.

[2]. Guillermo Marcus, Patricia Hinojosa, Alfonso Avila and Juan Nolazco-Flores, “A Fully Synthesizable Single-Precision, Floating-Point Adder/Subtractor and Multiplier in VHDL for General and Educational Use”, Proceedings of the Fifth IEEE International Caracas Conference on Devices, Circuits and Systems, Dominican Republic.

[3]. Xilinx Inc, ISE, at http://www.xilinx.com.

[4]. Behrooz Parhami, Computer Arithmetic: Algorithms and Hardware Designs, 1st ed. Oxford: Oxford University Press, 2000

[5]. John G. Proakis and Dimitris G. Manolakis (1996), “Digital Signal Processing: Principles. Algorithms and Applications”, Third Edition.

[6]. Patterson, D. & Hennessy, J. (2005), Computer Organization and Design: The Hardware/software Interface, Morgan Kaufmann.

[7]. Mentor Graphics Inc, FPGA Advantage, at http://www.mentor.com/fpgaadvantage.

[8]. IEEE Standards Board, IEEE-754, IEEE Standard for Binary Floating-Point Arithmetic, New York: IEEE, 1985.

[9]. Lamiaa S.A.Hamid, Khaled A.Sheata, Hassan El-Ghitani, Mohamed Elsaid (2010),“ Design of Generic Floating Point Multiplier and Adder/Subtractor Units”, in proceedings of the 12th IEEE international Conference on computer modeling and Simulation.

[10]. YAJUAN CH. and Q. WU. Design and implementation of PID controller based on FPGA and genetic algorithm. In: Proceedings of 2011 International Conference on Electronics and Optoelectronics. Dalian: IEEE, 2011, pp. 308–311. ISBN 978-1-61284-275-2.DOI: 10.1109/ICEOE.2011.6013491

[11]. ZHENBIN G., X. ZENG, J. WANG and J. LIU. FPGA implementation of adaptive IIR filters with particle swarm optimization algorithm. In: 11th IEEE Singapore International Conference on Communication Systems. Guangzhou: IEEE, 2008, pp. 1364–1367. ISBN 978-1-4244-2424-5. DOI: 10.1109/ICCS.2008.4737406.

[12]. OTSUKA, T., T. AOKI, E. HOSOYA and A. ONOZAWA. An Image Recognition System for Multiple Video Inputs over a Multi-FPGA System. In: IEEE 6th International Symposium on Embedded Multicore SoCs. Aizu-Wakamatsu: IEEE, 2012, pp. 1–7. ISBN 978-0-7695-4800-5. DOI: 10.1109/MCSoC.2012.33.

[13]. RAMAKRISHNAN, A. a J. M. CONRAD. Analysis of floating point operations in microcontrollers. In: Proceedings of IEEE Southeastcon. Nashville: IEEE, 2011, pp. 97–100. ISBN: 978-1-61284-739-9. DOI: 10.1109/SECON.2011.5752913.

[14]. UNDERWOOD, K. FPGAs vs. CPUs. In: Proceeding of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays. New York: ACM Press, 2004, pp. 171–180. ISBN 1-58113-829-6.DOI: 10.1145/968280.968305.