Proceedings of the Second International Conference on Computing Methodologies and Communication (ICCMC 2018)
IEEE Conference Record # 42656; IEEE Xplore ISBN:978-1-5386-3452-3

IMPLEMENTATION OF THE STANDARD FLOATING POINT MAC USING IEEE 754 FLOATING POINT ADDER

Dr. R. Prakash Rao[1]
[1]*Professor, Electronics and Communication Engineering, St. Peter's Engineering College, Near Forest Academy, Dulapally, Medchal, Hyderabad, India.*
prakashiits@gmail.com

Dr. K. Naveen[2]
[2]*Professor, Electronics and Communication Engineering, St. Peter's Engineering College, Near Forest Academy, Dulapally, Medchal, Hyderabad, India.*
naveenkarunya@gmail.com

N. Dhanunjaya Rao[3]
[3]*Assistant Professor, Electronics and Communication Engineering, St. Peter's Engineering College, Near Forest Academy, Dulapally, Medchal, Hyderabad, India.*
dhanunjayaraonatha@gmail.com

P. Ramya[4]
[4]*Assistant Professor, Electronics and Communication Engineering, St. Peter's Engineering College, Near Forest Academy, Dulapally, Medchal, Hyderabad, India.*
ramyap@stpetershyd.com

Abstract— There are two types of processors. The first category is arithmetic processors and the second category is signal processing processors. For arithmetic processors in general precision is less where as for the signal processing processors precision is high because the signal is to be reconstructed at the receiving end. In the arithmetic processors the basic building block is Arithmetic Logic Unit (ALU). In the signal processing processors the basic building block is Multiplier Accumulator Content (MAC). For the Standard floating point MAC, more precision is essential compared with the traditional fixed point MAC because the former is to be designed by the IEEE 754 floating point arithmetic and the lateral is designed with fixed-point arithmetic i.e, the precision is limited to integer. But, in general if the precision is more, the accuracy is more. Hence, to improve the performance of the traditional fixed point MAC, in this work we implemented the standard floating point MAC using IEEE 754 floating point adder. This can be used to design all floating point DSP processors through the standard floating point MAC.

Keywords— IEEE 754; VHDL; MAC; fixed-point arithmeti; DSP processors.

I. INTRODUCTION

This work the implementation of the Standard floating point multiplier accumulator content (MAC) using IEEE 754 floating point adder in VHDL. For the Standard floating point MAC, more precision is there compared with the traditional MAC because the first one is designed by the IEEE 754 floating point arithmetic. But, as the traditional MAC is designed with fixed-point arithmetic, the precision is limited [1]. This is the main bottleneck of traditional MAC. In general, MAC is the fundamental building block of any digital signal processing (DSP) system. MAC consists of multiplier, adder and shifter. In a DSP system, the sampled signal is transformed well using the convolution operation by the MAC. Multiplier is used to multiply the input samples with the filter co-efficients and then the result is sent to the adder. The adder or accumulator adds the current result with the previous stored result and then the new result is stored again in the accumulator. The shifter shifts the various input samples throughout the filter length and then the final output of the MAC is taken at the end of the adder [2]. Like this, the input signal is accumulated sample by sample and then finally the signal is reconstructed well at the output of the receiver.

Hence, MAC is the basic building block of any DSP system. This explains initially the traditional MAC and its limitations and then IEEE 754 Standard floating point MAC to eliminate the limitations of the traditional MAC.

II. TRADITIONAL MULTIPLIER ACCUMULATOR CONTENT

Traditionally, the MAC is operated with the fixed point numbers for the fixed point DSP applications. The fixed point MAC is constituted by the fixed point adder, fixed point multiplier and a shifter. The sampled values which are x(n) will be given as input to the shifter. The shifter will shift the value of 'n' for different samples starting from first sample n=0 to the last sample i.e., n=N-1 where 'n' indicates number of samples and 'N' indicates length of the filter.

A. Algorithm

If h(n) represents the filter co-efficients and x(n) represents the input signal samples then to get the output of the MAC which is the convolution operation, first replace the variable 'n' in h(n) and x(n) with some other variable like 'k' as h(k) and x(k) respectively.

Now shift right the input sample, x(k) with the variable 'n' so that the shifted input sample becomes x(n-k) and then multiply the x(n-k) with the h(k). Now add the current product with the previous product, after the accumulation of all the values obtain y(n). It can be simply concluded in two steps as-

1) Multiply the data sample x(n-k), with the co-efficient h(k).

2) Add the current product to the previous output and obtain y(n) at last

Figure 1 shows the block diagram of the fixed-point MAC. The sample by sample shifted values are sent to the fixed point multiplier. The filter co-efficients which are calculated or generated as per the user specifications are stored in the ROM location and sent as an input to the fixed–point multiplier. The filter co-efficients and input samples are multiplied to each other and the result will be sent to the fixed point adder [3].

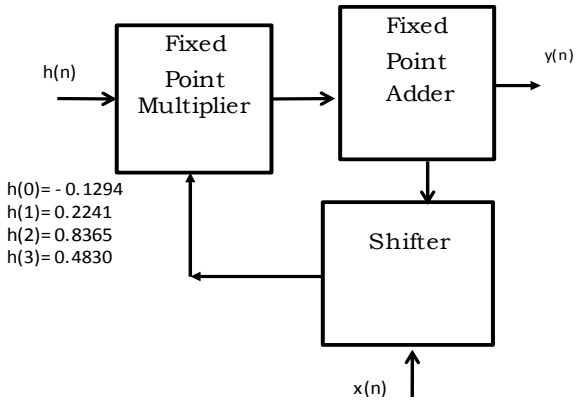

Figure 1: Fixed point MAC

The fixed point adder adds the current result with the previous result. Like this from n=0 to n=N-1 the operation will be performed and finally the output is available as y(n) at the output of the fixed point adder. As shown in Table 1, the filter co-efficients which are generated by giving different specifications are in the form of floating point, but the MAC is operated in the fixed point hence the floating point co-efficients are converted into fixed point using some scaling parameters [4]. In Table 1, the floating point co-efficients have been converted into fixed point by the multiplication of each through 2^3 scale factor.

Table 1: Scaling of floating point filter co-efficients into fixed point

Filter co-efficients	Scaled up by 2^3
h(3)= 0.4829629	h(3)≈4
h(2)= 0.8365163	h(2)≈7
h(1)= 0.2241439	h(1)≈2
h(0)=-0.1294095	h(0)≈ -1

This total operation of the MAC is called the "convolution". The convolution operation is the fundamental arithmetic operation of any DSP system [5]. The convolution output y(n) of the fixed-point MAC can be expressed mathematically by equation1.

$$y(n)=\sum_{k=0}^{N-1} h(k).x(n-k) \quad ---1$$

If the filter length N=4, equation 1 can be expressed as

$$y(n)=\sum_{k=0}^{4-1} h(k).x(n-k) \quad ---2$$

By expanding the summation of equation 2, it becomes

y(n)= h(0).x(n-0)+ h(1).x(n-1)+ h(2).x(n-2)+ h(3).x(n-3) –3

where h(o), h(1), h(2) and h(3) in equation 3 indicate filter co-efficients and x(n), x(n-1), x(n-2) and x(n-3) indicate various sample values at n=0,n=1, n=2 and n=3 respectively.

If n=0, equation 3 becomes
y(0) = h(0). x(0) +h(1).x(-1) +h(2).x(-2) +h(3).x(-3)
If n=1, equation 3 becomes
y(1) = h(1).x(1) +h(1).x(0) +h(2).x(-1) +h(3).x(-2)
If n=2, equation 3 becomes
y(2) = h(0).x(2) +h(1).x(1) +h(2) .x(0) +h(3).x(-2)
If n=3, equation 3 becomes
y(3) = h(0).x(3) +h(1).x(2) +h(2).x(1) +h(3).x(0)

A. Limitations of Traditional MAC

Basically, there are four limitations for the fixed point MAC [6]. These can be explained as given below:

1) ADC quantization error.
2) Co-efficient quantization error.
3) Over flow errors and
4) Round-off errors or Truncation errors

In the above, two problems of over flow errors and round off errors can be eliminated effectively with the floating point operation because the floating point arithmetic is having larger dynamic range. Over flow errors and round-off errors are very serious in the IIR filters than FIR filters. As feedback is there in the IIR filters these errors are continuously added up and finally limit cycle oscillations are produced due to which the system may become unstable.

III. STANDARD FLAOTING POINT MAC USING IEEE 754 FORMAT

The standard floating point MAC or MAC using IEEE754 standard consists of floating point adder, floating point multiplier along with a shifter to perform the convolution operation.

A. IEEE 754 Floating Point Standard:

There are various ways of representing the numbers in the computer among which fixed-point and floating point number representations are main. In order to represent the very large numbers or very small numbers, fixed point representation cannot be used [6] because in the fixed point representation, the numbers are represented with the "fixed window" where as such type of very large numbers or very small numbers can be represented very easily with the floating point number representation since floating point representation is having "sliding window".

For the floating point representation, different vendors follow different notations. In order to bring all the vendors under one umbrella, the Institute of Electrical and Electronics Engineers(IEEE) which is the recognized body started to represent the single notation called IEEE 754 in which the rules and regulations to operate the floating-point arithmetics' are illustrated [7]. IEEE 754 floating point standard has been found in every computer virtually for around 1980 to perform the floating point arithmetic operations.

Floating point representation:

In general the floating point numbers are represented as - ±d.ddd….d x b^e, where d.ddd….d is called the mantissa and has 'p' digits which are also called the precision of the number; 'b' is called the base which is 2 for binary,10 for decimal and 16 for hexa decimal; 'e' is the exponent which is equal to E_{max}-E_{min}. E_{max} represents maximum exponent and E_{min} represents minimum exponent.

If b=10 and p=4 then the number 0.1 is represented $1.000x10^{-1}$. If b=2 and p=24 then the decimal number 0.1 cannot be represented accurately, but it can be represented approximately as $1.10011001100110011001101x2^{-4}$. Hence, the most significant factor of the floating point arithmetic is precision or the number of bits used to represent the mantissa [8].

Basic Format:

There are various basic formats to represent the floating point numbers in IEEE754 standard. Depending upon the user requirement, the format is to be chosen. Different formats are- half precision, single precision and

double precision. The summary of different formats are illustrated in the below Table 2.

Table 2: The summary of different formats

Format	Precision	E_{max}	E_{min}	Exponent bias	Exponent width	Format width
Half	16	+7	-6	7	4	16
Single	24	+127	-126	127	8	32
double	53	+1023	-1022	1023	11	64

In half precision, the field of IEEE 754 standard can be represented as, for the sign, 1-bit; for the exponent, 4-bits; and for the mantissa, 11-bits.

The half precision floating point number can be calculated as $(-1)^S$ x (1.F)x $2^{Val(E)}$. The sign bit is ether zero for the positive numbers or 1 for the negative numbers which can be understood as $(-1)^S$ =1 when S=0 or sign is positive and $(-1)^S$ =-1 when S=1 or sign bit is negative. In general, the signed exponent can be represented in three formats like 'Sign magnitude' representation, 2's complement representation and 'Biased' representation [9].

IEEE 754 uses biased representation for exponent which is nothing but, Value of exponent = Val(E) = E-Bias, where Bias is a constant. Because, the exponent field is 4 bits for the half precision, it can be in the range of 0 to 15. E=0 and E=15 are kept for special case numbers such as zero, denormalized numbers, infinity and not a number; the remaining E=1 to 14 are used for normalized floating point numbers. The Bias is the half of the 14 values range of the exponent. Therefore, Bias = 7 which is half of 14. It can be understood that if the exponent field E=0001, then the Val(E=1)=1-7= -6 and the exponent field E=1110, then the Val(E=14)=14-7= 7.

Regarding mantissa which is composed of an implicit leading bit and the fraction bits that represent the precision bits of the number. With the above analysis, the smallest number of 1 0100 00001001000 can be converted into decimal as given below. In this, the sign bit is 1 and the exponent is 4. Therefore, Val(E)=E-7 = 4-7=-3. Fraction = $(1/2^5 + 1/2^8)$ x 2^{-3} = 0.0040283203 x $1/2^3$ =0.00050354 =5.0354x10^{-4}.

Finally, the decimal value of 1 0100 00001001000 is +5.0354x10^{-4}. The complete analysis of the IEEE754 half precision floating-point encoding is shown below in Table 3

Table 3: The complete analysis of the IEEE754 half precision floating-point encoding

Sign	Exponent	Fraction	Value	Description
S	0xF	0x0000	$(-1)^S \infty$	Infinity
S	0xF	F≠0	NaN	Not a Number
S	0x0	0x0000	0	Zero
S	0x0	F≠0	$(-1)^S$ x (0.F)x $2^{(E-6)}$	Denormalized Number
S	0x00 <E< 0xFF	F	$(-1)^S$ x (1.F)x $2^{(E-7)}$	Normalized Number

The brief description of the Normalized Number, Denormalized Number, Infinity, Not a Number and Zero can be explained in the below sections.

Normalized numbers:

A floating point number is supposed to normalized only when the exponent field consists of the real exponent plus the bias other than 0xF and 0x00. For all the normalized numbers, there is a hidden bit '1' before the decimal point of the fraction which is also called the implicit bit. Hence, the half precision encodes only the lower 11 bits which are the bits after the decimal point.

Denormalized numbers:

A floating point number is said to denormalized only when the exponent field is 0x00 and the fraction filed does not contain all 0's. The hidden bit is set to '0'.

Infinity:

In a half precision representation, infinity is represented by the exponent field of 0xF and whole fraction filed of 0's.

Not a Number(NaN):

In a half precision representation, NaN is represented by exponent field of 0xF and the fraction field that doesn't include all 0's.

Zero:

In a half precision representation, zero is represented by exponent field of 0x0 and the whole fraction field of 0's. The sign bit represents -0 and +0.

Rounding modes:

Rounding takes a number regarded as infinitely precise and if necessary modifies it in the destination's format while signaling the inexact exception. There are four rounding modes in IEEE754 format which are nothing but, round towards nearest even(REN), round towards -∞(RN), round towards +∞(RP) and round towards 0(RZ).

Exceptions:

Exceptions are signaled by setting the flag or setting a trap. In IEEE 754, there are five types of exceptions: overflow, underflow, invalid operation, inexact result, division-by-zero. Because, the adder and the multiplier are implemented in the floating point, only two exceptions like overflow and underflow occur during the operation.

B. Implementation of Standard Floating Point MAC Using Floating Point Arithmetic:

To implement the Standard floating point MAC using IEEE 754 floating point arithmetic, the user defined floating point arithmetic package has been developed first because the digitally defined simulation tool like Modelsim is having the fixed point IEEE library packages only but it does not have floating point library package. The developed user defined floating point arithmetic package has been compiled with the fixed point IEEE library packages of the digitally defined simulation tool after that the floating point MAC is simulated on the updated tool.

The Standard floating point MAC is constituted by the floating point adder, floating point multiplier and a shifter. The block diagram of the Standard floating-point MAC is shown in below Figure 2. The sampled values which are x(n) as an input will be given to the shifter. The shifter will shift the value of 'n' for different samples starting from first sample n=0 to the last sample i.e., n=N-1 where n=number of samples and N=length of the filter.

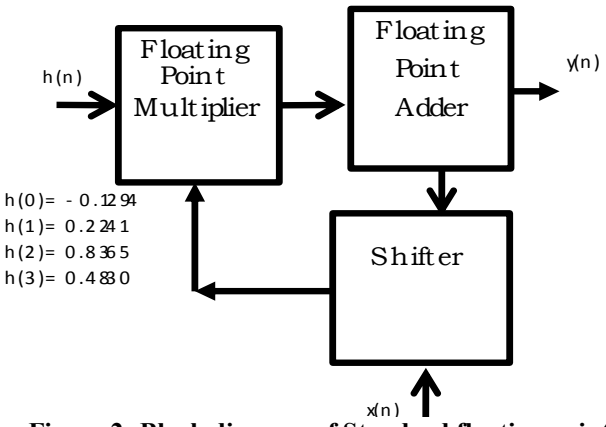

Figure 2: Block diagram of Standard floating point MAC

In the above Figure 5.5, to get output y(n); in x(n) and h(n), 'n' is replaced with other variable [3] say 'k', hence they become h(k) and x(k); then the output y(n) could be obtained as shown in equation.4.

$$y(n)= \sum_{k=0}^{N-1} h(k).x(n-k) \quad \text{---------- 4}$$

For N=4, k becomes as k=0, k=1, k=2 and k=3. Substitute 'k' values in expression 5.4.

y(n)= h(0)x(n-0)+ h(1)x(n-1)+ h(2)x(n-2)+ h(3)x(n-3). When n=0,1,2 and 3, y(n) becomes

y(0)= h(0)x(0)+ h(1)x(-1)+ h(2)x(-2)+ h(3)x(-3);
y(1)= h(0)x(1)+ h(1)x(0)+ h(2)x(-1)+ h(3)x(-2);
y(2)=h(0)x(2)+h(1)x(1)+h(2)x(0)+h(3)x(-1) and
y(3)=h(0)x(3)+h(1)x(2)+h(2)x(1)+h(3)x(0).

In the above example, x(0) x(1) x(2)…. values are the floating point values. But n=0,1,2,3 ……are integers only. Similarly, h(0),h(1),h(2),h(3) values are the floating point values. But, n=0,1,2,3… act as integers only. To store such 16 bit length of the floating point values, the length of the internal registers of the DSP system should be 16. Using this floating point operators; overflow errors, round off errors or truncation errors can be minimized.

1) Standard Floating point Adder Architecture

Because, the 16 bit floating point number consists of 11 bits for mantissa, 4 bits for exponent and 1 bit for sign; to add such two 16 bit floating point numbers, the floating point adder architecture requires two 16 bit registers to store the two 11 bit mantissas, two 4 bit registers to store the two exponents and two 1 bit registers to store the sign bit.

While adding the two floating point numbers the smallest number is to be identified so that eight bit subtractor is required for the exponent. Similarly, it requires one 2x1 multiplexer to select the input data depending upon the status of the select line; one 32 bit swap register to swap the smallest mantissa with the highest mantissa; one 16 bit register to shift right the smallest mantissa to the right side bit by bit to make equal the smallest mantissa to the biggest mantissa value depending upon the difference value of the two exponents [10].

To add the two mantissas, 32 bit significant adder is required. Similarly, after addition if carry is generated, to detect that carry, one bit register and to normalize the result one 16 bit register are required. At the end to round the result one sixteen bit register, to adjust the value of the exponent one four bit register are required. Finally, to store the resultant value of the addition of two floating point numbers, it requires 1 bit register for sign, 4 bit register for the exponent and 11 bit register for the mantissa.

2)Algorithm for Standard Floating point adder

Let S1, E1, M1 are the sign, exponent and mantissa of the first floating point operands of N1 and S2, E2, M2 are the sign, exponent and mantissa of the second floating point operands N2, then for the standard floating point adder, the explanation of the algorithm is as follows:

a) Initially, the system reads the two operands of N1 and N2 for denormalization and infinity. Set the hidden bit of the fraction to 0 if numbers are denormalized otherwise set to 1.
b) Using the 4-bit subtractor, the two exponents E1, E2 are compared. If E1 is less than E2, N1 and N2 are swapped which means that previous M2 is now referred to as M1 and vice versa.
c) The smaller fraction, M2 is shifted right by the absolute difference result of the two exponents' subtraction. Now both the numbers have the same exponent.
d) Now the two mantissas of M1 and M2 are added.
e) For the normalization, after addition the result is then passed through a leading one detector.
f) Using the results from the leading one detector, if it is needed, the result is then shifted right by 1 bit to complete the normalization process.
g) After normalization, using the default rounding mode the result is rounded to the nearest value.

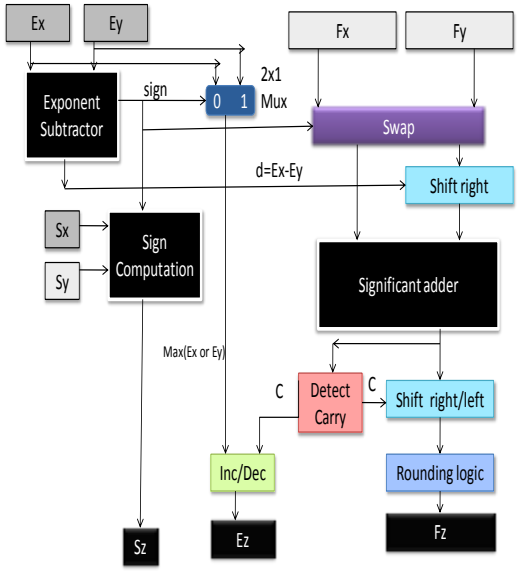

Figure 3: 16 bit floating point adder

h) The exponent is adjusted using the results from the leading one detector.
i) The sign is computed depending on the value of exponents of E1 and E2 which means that whichever the exponent is the maximum, that sign is computed. The result is registered after the overflow and underflow check.

Based on the above description, the micro architecture of the 16 bit floating point adder has been implemented as shown in Figure 3.

C. Standard Floating point Multiplier Architecture

Unlike the design of 16 bit floating point adder, the design of 16-bit floating point multiplier is simple [11], [73]. Because, the 16 bit floating point number consists of 11 bits for mantissa, 4 bits for exponent and 1 bit for sign; to multiply such two 16 bit floating point numbers, the floating point multiplier architecture requires two 16 bit registers to store the 11 bit mantissas, two 4 bit registers to

store the two exponents and two 1 bit registers to store the sign bit.

Similarly, it requires 8-bit register to add the two exponents, 32-bit register to multiply the two 16 bit mantissas, a 32 bit register for normalization and a 16 bit register for the result storage.

1) Algorithm for Standard Floating point Multiplier

Let S1, E1, M1 and S2, E2, M2 be the signs, exponents and mantissas of the two input floating point operands of N1 and N2 respectively. For the standard floating point multiplier, the description of the algorithm is as follows:

a) For the floating-point multiplication, to get the sign bit, S1 and S2 of the two operands of N1 and N2 are exclusive-OR operated.

b) The exponents, E1and E2 of the two operands of N1 and N2 are added up from which the bias of 7 for half precision is subtracted and the exponent value is finalized based on the carry propagated from the result of multiplication of the two mantissas.

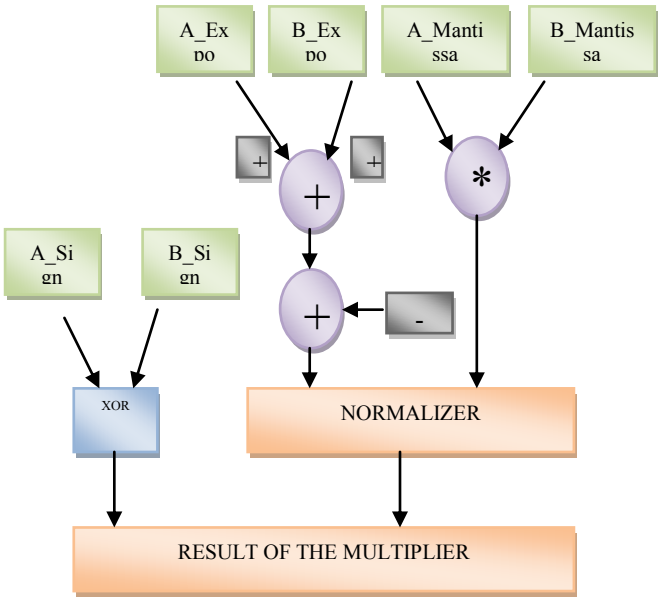

Figure4: 16 bit floating point multiplier

c) The mantissas, M1 and M2 of the two operands of N1 and N2 are multiplied by the elementary school algorithm and if carry is generated from the result, it has to be normalized by right shift of the result bits to the decimal point.

d) The result bits are doubled after the multiplication of two mantissas and hence the truncation mode is applied using which the right side bits are truncated to fit the result with the specified bit length of the register.

Based on the above description, the micro architecture of the 16 bit floating point multiplier has been implemented as shown in Figure 4.

IV .RESULTS

This section gives the simulation as well as synthesis results of Standard floating point adder and Standard floating point MAC.

1) Simulation results:

The simulation for the Standard floating point adder and MAC have performed by the Modelsim 10.3c tool. Because, the Modelsim tool is not having the floating point packages, the user defined floating point library package is developed and compiled with the standard library of the Modelsim tool 10.3c. Now by porting the 16-bit floating point input values through the test-bench, the output could be obtained .

a) FOR STANDARD FLOATING POINT ADDER:

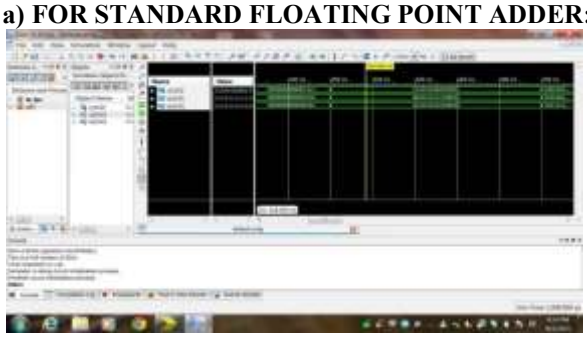

Figure 5: Simulation results of Standard Floating Point Adder

The synthesis is carried out using Xilinx Synthesis Technology (XST) tool. The selected hardware device is Xc2s50e-ft256-6.The speed grade is -6. In this device, the maximum number of IOs are 182 and the maximum number of BELs are 1728.

Table 4: Synthesis report of floating point Adder

Hardware Resources/Parameters	Standard floating point Adder
Number of IOs	31 out of 182(17%)
Number of BELs	59 out of 1728(3.4%)
Minimum period	0.983 ns
Max. Frequency (speed)	1017.2 MHz
Power consumption	9.726mw

Figure 5 and Table 4 show the simulation results and synthesis report of Standard floating point adder. It is understood that the obtained speed of the Standard floating point adder is 1017.2 MHz with the power consumption and delay of 9.726mw and 0.983 ns respectively. The hardware resources which were taken by the Standard floating point adder are 17% of IOs and 3.4% of BELs.

b)FOR STANDARD FLOATING POINT MAC:

Figure 6 and Table 5 show the simulation results and synthesis report of Standard floating point MAC respectively.

Figure 6: Simulation results of Standard Floating Point MAC

Table5: Synthesis report of floating point MAC

Hardware Resources/Parameters	Standard floating point MAC
Number of IOs	54 out of 182(29%)
Number of BELs	103 out of 1728(6%)
Minimum period	1.286 ns
Max. Frequency (speed)	793.65 MHz
Power consumption	12.493mw

With the above Table 5, it is understood that the obtained speed of the Standard floating point MAC is 793.65 MHz with the power consumption and delay of 12.493mw and 1.286 ns respectively. The hardware resources which were taken by the Standard floating point MAC are 29% of IOs and 6% of BELs.

IV. CONCLUSION

Because, there are various drawbacks in the fixed point MAC, mainly precision and accuracy the Standard floating point MAC has been implemented in this work by developing the user defined floating point library package with IEEE 754 standard. For the fixed point MAC in order to convert the speech signal into digital, the physical connection of the ADC and DAC are used at the input and output of the system respectively. But by developing the 16-bit floating point packages, a logical connection has been established instead of the physical connection of the ADC and DAC so that the design difficulty of high precision ADC and DAC has been eliminated with this work.

REFERENCES

[1] M. J. Flynn, S. F. Oberman, AdvancedComputer Arithmatic Design. John Wiley & Sons, Inc, 2001.

[2] Israel Koren, Computer Arithmetic Algorithms, A K Peters, second edition, 2002.

[3] J. Hennessy and D. A. Peterson, Computer Architecture a Quantitative Approach, Morgan Kauffman Publishers, second edition, 1996.

[4] Tong Ziquan, Yang Shaojun, Jiang Yueming and Dou Naiying, "The Design of a Multi-bit Quantization Sigma-delta Modulator", International Journal of Signal Processing, Image Processing and Pattern Recognition, Vol.6, No.5 (2013), pp.265-274.

[5] Dhanabal, Sarat kumar sahoo, Barathi, Naamatheertham, Samhitha, Neethu acha cherian, Pretty mariam jacob, "Implementation of floating point MAC using residue number system", Journal of Theoretical and Applied Information Technology, Vol. 62 No.2, ISSN: 1992-8645.

[6] Deepika Setia, Charu Madhu, "Novel Architecture of High Speed Parallel MAC using Carry Select Adder", International Journal of Computer Applications (0975 – 8887) Volume 74– No.1, July 2013.

[7] L.-H. Chen, O. T. C. Chen, T.-Y. Wang, and Y.-C. Ma, "A multiplication-accumulation computation unit with optimized compressors and mini-mized switching activities," inProc. IEEE International Symposium onCircuits and Systems, 2005 (ISCAS 2005), vol. 1, May 2005, pp. 6118–6121

[8] Pradnya A. Shengale, Vidya Dahake, Mithilesh Mahendra,, "Single precision Floating point ALU", International Research Journal of Engineering and Technology, Volume: 02 Issue: 02 | May-2015.

[9] Vladik Kreinovich and Siegfried Rump, "Towards Optimal Use of Multi-Precision Arithmetic: A Remark", published in Reliable Computing, 12:365-369, 2006.

[10] L. Fousse, G. Hanrot, V. Lefμevre, P. Pelissier, and P. Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding. Research Report RR-5753, INRIA, 2005.

[11] Kari Kalliojarvi and Yrjo Neovo, "Distribution of RoundoffNoise in Binary Floating - Point Addition", Proceedings of ISCAS92, pp. 1796-1799.

[12] M. Schulte, E. Swartzlander, "Hardware Design and Arithmetic Algorithms for a Variable-Precision, Interval Arithmetic Coprocessor", Proc. IEEE 12th Symposium on Computer Arithmetic (ARITH-12), pp 222-230, 1995.

[13] Nabeel S hirazi, A1 Walters, an d P eter Ath anas."Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines". In IEEE Symposium on FPGAs for Custom Computing Machines, pages 155-162, April 1995

[14] M.Karthik kumar, D.Manoranjitham, K.Praveen kumar, "Implementation of Efficient 16-Bit MAC Using Modified Booth Algorithm and Different Adders", International Journal Scientific and Research Publications, Volume 4, Issue 3, March 2014, ISSN 2250-3153.

[15] Dhiraj Sangwan, Mahesh K. Yadav, "Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic", International Journal of Electronics Engineering, 2(1), 2010, pp. 197-203.

[16] G. Bewick, "Fast Multiplication : Algorithms and Implementation", PhD. Thesis, Stanford University, 1992-CH6

[17] IEEE Standard Board and ANSI, "IEEE Standard for Binary Floating- Point Arithmetic," 1985, IEEE Std 754-1985.