ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 4, Issue 10, October 2015

REDUCING THE COMPUTATION TIME IN FAST FLOATING POINT MULTIPLIER UNIT

C.M.KALAISELVI[1], V.P.DHIVYAPRIYA[2], P.NITHYA[3]

ABSTRACT- Floating point numbers are the quantities that cannot be represented by integers, either because they contain fractional values or because they lie outside the range re-presentable within the system's bit width. The growing market for fast floating-point co-processors, digital signal processing chips, and graphics processors has created a demand for high speed, area-efficient multipliers. The advantage of floating-point representation over fixed-point (and integer) representation is that it can support a much wider range of values. Architecture for a fast floating point multiplier compliant with the single precision IEEE 754 standard floating point multiplier has been designed. The floating point representation can preserve the resolution and accuracy compared to fixed point. Divide & Conquer technique and pipelining technique are used to design floating point multipliers. The multiplier implementation in floating point multiplication is done by Modified Booth Encoding (MBE) multiplier to reduce the partial products by half. Radix-8 Booth Encoder circuit generates n/3 the partial products in parallel. Use of the compressors permit the reduction of the vertical critical paths. This paper provides the techniques that optimize the area and speed up the multiplication process. In the proposed system, the area and delay is reduced to 15% and 20% when compared to radix-4 algorithm. The simulations have been carried out using the Xilinx ISE tool.

KEYWORDS- ***Floating point number, Kogge stone adder, Pipeline, radix-4 and radix-8 booth encoder, Wallace tree structure.***

.

I. INTRODUCTION

With the emerging need for high speed VLSI devices, there is a continuous demand for high speed multipliers. The multiplier is an essential part of the digital signal processing such as filtering and convolution [8]. Most digital signal processing methods use nonlinear functions such as discrete cosine transform (DCT) or discrete wavelet transform (DWT). As they are basically accomplished by repetitive application of multiplication and addition, their speed becomes a major factor which determines the performance of the entire calculation. Since the multiplier requires the longest delay among the basic operational blocks in digital system, the critical path is determined more by the multiplier. Furthermore, multiplier consumes much area and dissipates more power. Hence designing multipliers which offer either of the following design targets – high speed, low power consumption, less area or even a combination of them is of substantial research interest. However, area and speed are usually conflicting constraints hence improving speed results mostly in larger areas.

There are generally three phases in tree multiplier architecture, which are partial product generation phase, partial product reduction phase and finally the addition phase to obtain the final result. Among these three phases, the second phase - partial product reduction phase consumes most of the power and is responsible for overall critical path delay [8].

Multipliers using floating point numbers are in great demand because floating point numbers have good precision, since they never deliberately discard information. So a fast and energy-efficient floating point unit is always needed in electronics industry. Today the main applications of floating point numbers are in the field of medical imaging, biometrics, motion capture and audio applications [4]. Floating-point representation, in particular the standard IEEE format, is by far the most common way of representing an approximation to real numbers in computers because it is efficiently handled in most large computer processors. The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, that is, the decimal point can float. The Institute of Electrical and Electronics Engineer (IEEE) sponsored a standard format for 32-bit and larger floating point numbers, known as IEEE 754 standard. To reduce significant power consumption it is good to reduce the number of operation thereby reducing dynamic power which is a major part of total power consumption.

The multiplier makes use of the Radix-4 Booth Algorithm with 4:2 compressors. The number of partial products is n/2 in Radix-4 Booth algorithm while it gets reduced to n/3 in Radix-8 Booth algorithm. This can be accomplished with usage of 3-2 Compressors (Full-

Adders) and 4-2 Compressors. The pipelining is a popular technique to increase throughput of a high speed system, which divides total system into several small cascade stages and adds some registers to synchronize outputs of each stage [1].

The rest of this paper is organized as follows. Section II describes the literature review. Section III describes about the existing system. Section IV describes about the proposed system. Section V describes about the conclusion.

II. LITERATURE REVIEW

A high speed binary floating point multiplier is based on Dadda algorithm. By using Dadda Algorithm the number of half adders and full adders needed will be increased, so the area gets increased. Rounding is not implemented in order to be more precise [7].

A high speed floating point multiplier based on FPGA, modified booth Radix-4 algorithm is used. Unsigned binary bits are used, so the number of partial products gets increased by one row.

Table1. Comparison between multipliers [9]

Logic Utilization	Existing Method	Proposed Method	Available
No. of Slices	401	281	5472
No. of slice Flip-flops	72	32	10944
No. of 4-input LUT's	710	504	10944
No. of Bonded IOB's	99	97	240
No. of Global Clocks	2	1	32

III EXISTING SYSTEM

A. IEEE 754 Floating Point Standard

The representation of using floating point numbers has been bring out by IEEE is known as IEEE 754[] and used in all CPU implementation. For representing floating-point numbers which has negative numbers and de-normal numbers together with a set of floating-point operations that operate on IEEE 754 standard. It has 4 modes of rounding which are round to nearest, round to ∞, round to 0, round to even and five exceptions counting when the exceptions occur. The usability of a processor will be limited when dealing with fixed point arithmetic [1]. If operations on numbers with fractions, very tiny numbers (e.g. 0.000004), or very huge numbers (e.g. 62.445x 10^5) are required, then a different form of representation is used in the floating-point arithmetic. In order to get some notation away of the way, let us talk about a few floating-point -6.24x 10^3. The negative symbol represents the sign part of the number, while the '624' shows the significant digits part of the number, and to conclude the three shows the scale factor part of the number. The string of major digits is officially termed the mantissa part of the given number, while the factor is fittingly called the exponent part of the number. The broad representation of floating point is

$$(-1)S * M * 2E$$

Where S stands for sign bit, M stands for - mantissa bit and E stands for - exponent bit.

B Single Precision Floating point Numbers

Most of the DSP applications need floating point numbers multiplication. The possible ways to represent real numbers in binary format floating point numbers are; the IEEE 754 standard [1] represents two floating point formats, Binary interchange format and Decimal interchange format. Single precision normalized binary interchange format is implemented in this design. Representation of single precision binary format is shown in Figure 1; starting from MSB it has a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). Adding an extra bit to the fraction to form and is defined as significand1. If the exponent is greater than 0 and smaller than 255, and there is 1 in the MSB of the Significand then the number is said to be a normalized number; in this case the real number is represented by (1). Thus, a total of 32 bits is needed for single-precision number representation. To achieve a bias equal to 2n-1 -1 is added to the actual exponent in order to obtain the stored exponent. This equal 127 for an 8-bit exponent of the single precision format. The addition of bias allows the use of an exponent in the range from -126 to +127. The single precision format offers a range from 2-126 to2+127.

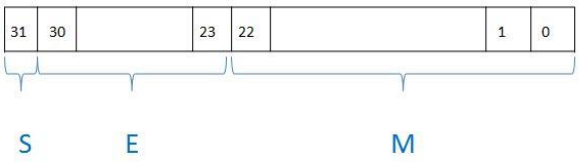

Fig.1 IEEE Single precision floating point format

$$Z = (-1^S) * 2^{(E - Bias)} * (1.M)$$

Where Bias=127

Table 2. Bit Range for Single (32-bit) and Double (64-bit) Precision Floating Point Values [5]

	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

C. FLOATING POINT MULTIPLIER UNIT

Floating point multiplication of two numbers is made in four steps [7]:

Step 1: Exponents of the two numbers are added directly, extra bias is subtracted from the exponent result.

Step 2: Significands multiplication of the two numbers using Dadda algorithm.

Step 3: To find the sign of result, XOR operation is done among sign bit of two numbers.

Step 4: Finally the result is normalized such that there should be 1 in the MSB of the result (leading one).

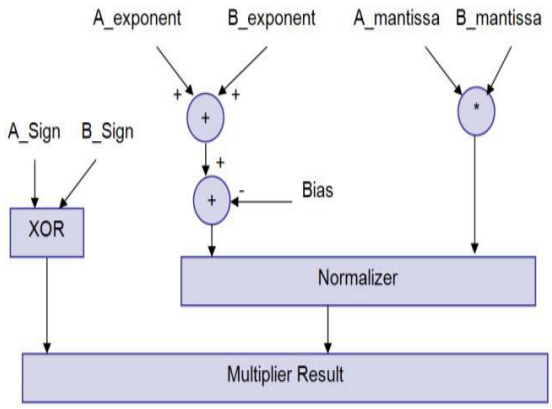

Fig.2 Floating point multiplier block diagram [10]

i) Floating Point Multiplication Algorithm

As stated in the introduction, normalized floating point numbers have the form of $Z= (-1^S) * 2^{(E - Bias)} * (1.M)$. To multiply two floating point numbers the following is done:

1. Multiplying the significand; i.e. (1.M1*1.M2)
2. Placing the decimal point in the result
3. Adding the exponents; i.e. (E1 + E2 – *Bias*)
4. Obtaining the sign; i.e. s1 xor s2
5. Normalizing the result; i.e. obtaining 1 at the MSB of the results' significand
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence

ii)Multiplier Flow

Multiplication of floating point number can be carried out in 3 parts [1].

In the 1st part, the sign product will be perform a XOR operation.

In the 2nd part, the exponent bits operands are passed to an adder stage and a bias 127 is subtracted from the output. 8-bit kogge-stone adder is used for implement the addition and 2s complement addition for subtraction operations.

a. Kogge-stone adder

The Kogge-Stone adder is a parallel prefix form Carry-look ahead adder. It generates the carry signals in (log n) time, and is widely considered the fastest adder design possible. It is the common design for high-performance adders. An example of 8-bit Kogge-Stone adder structures are shown in the figure.

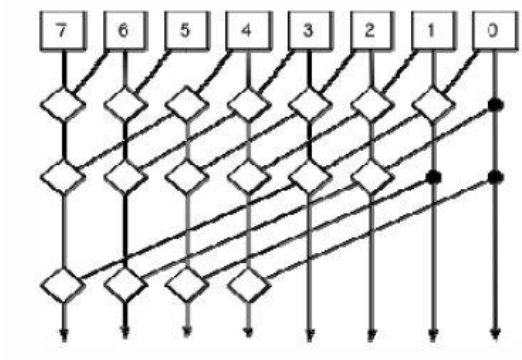

Fig 3. Koggestone adder

In the 3rd stage, find the product of the mantissa portion and the multiplication of mantissa portion is carry out in the following steps.

iii) Partial product generator:- For a given multiplier [6] there are many ways to generate partial products. The radix-4 booth programming was found to be quicker in which we had found out, so it will be put into operation in the final multiplier architecture. Twelve partial products are the output of this stage.

Radix 4 booth encoder

To recode the terms, divide it into block of three and in that every one block overlaps the prior block by one bit. The bits are grouped from the LSB, and 1 st block only takes 2 bits of the multiplier for grouping (no prior block to overlap): Two bits the multiplier have been in use by the least significant block, and consider a 0 for the third bit.

iv) Partial result accumulator:- The partial result obtained from the proposed multiplier will be used in the Wallace tree structure which is 4:2compressors. The propagation carry time is less in 4:2 compressor technique, used in carry save adders.

Wallace tree structure

Partial results is added in Wallace tree which is a new technique to propagate the carry that is used in the carry save adders [3]. The 4:2 compressor structures compresses five partial products bits into three. Figure _ shows the block diagram for the data distribution among a tree architecture that utilizes 4:2 compressors.

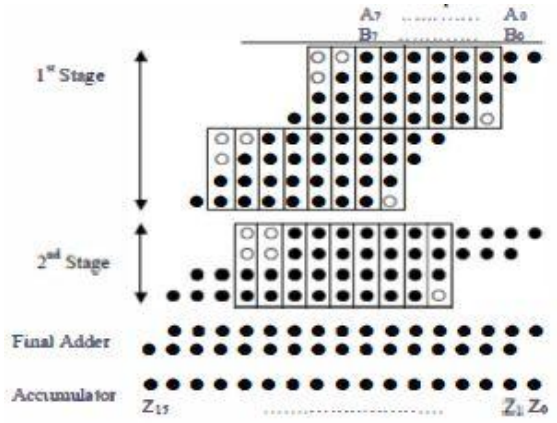

Fig. 4 Data distribution among tree structure

Each packet consists of the bits that fed into a 4:2 compressor group. The partial products result can be reduce by ratio of 2: 1 for two stages of 4:2 compressors. Figure _ show the cutback tree of 8 partial products to form two operands, added in to form a final product by using a speedy carry propagate adder.

C. Final stage adder: - The products of mantissas are specified by the 48-bit sum and carry outputs obtained from the partial product are added in the final stage adder. This stage adders should have a small amount delay and high speed. After research, comparing and implementing the power and delay uniqueness of various adders, we found out that the KoggeStone adder is the fastest among all the adder.

D. Normalization and rounding: - The product of mantissas is normalized and round off. The excess one is detected and the exponent is adjusted for normalization. We are reducing the implied bit which is foremost one [3]. The left over bits are reduced to a 26-bit value. For precision a few extra bits is added to the reduced value. The reduced value is finally rounded off using the rounding to nearest value to give the 23 bit mantissa of the product. A zero detect block is used in the multiplier architecture to avoid unnecessary calculations of zero in the input**.**

D.PIPELINING

Pipelining is a method where multiple instructions are overlapped. It separated into segments. Each segment will carry out its operation all together with other segments. After finishing of one step of operation outcome of that step will be passes to next step in channel, carries the next operation into next segment. The final outcome of each instruction will be produced at end. The pipeline method techniques are widely used for making better performance for digital circuits. The overall performance can be enlarged by increasing the pipelining stages which decreases the path delay in each and every stage.

a) 4-Stage Pipelining

The multiplier structure is prearranged as a four-stage pipeline. This arrangement allows to generate one result in each clock cycle, after make the first three values, outcome have been entered into the unit.

To increase the performance of multiplier, four stage pipelining is used. In order to increase the performance of multiplier operating frequency of multiplier is increased, pipeline stages are inserted in the critical path. Pipelining stages reduce the latency in the output by four clocks [7].

The pipelining stages are followed in following steps:

i. Pre-processing the input data.
ii. Addition of partial products
iii. Subtracting the Bias and Compressing the partial result in Wallace tree.
iv. Normalization and final carry is adder.

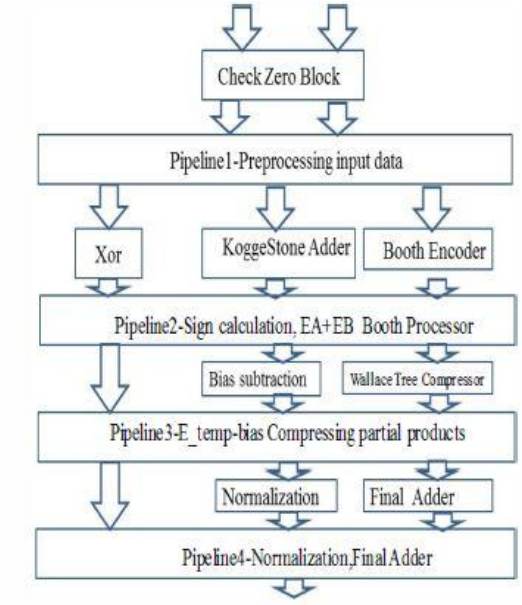

Fig.5 Multiplier unit

Pipeline registers is implement with least number of register in pipeline, positions of the pipeline register is inserted in Koggestone Adder in order to get the high speed. Koggestone adder have three pipeline stages as shown in figure 5. Implementing the pipelining register is made at three stages. First stage is where two inputs of the adder is set, next stage is after the estimate of partial product stage 1 in Koggestone adder, and final stage is after the stage 2 of adder.

IV. PROPOSED SYSTEM

A.Radix-8 Modified Booth's Algorithm

The speed of multiplication can be increased by reducing the number of partial products and/or accelerating the accumulation of partial products. Recoding extended to 3 bits at a time – overlapping

groups of 4 bits each. Radix-8 recoding applies the same algorithm as radix-4, but now we take quartets of bits instead of triplets. Consequently, a multiplier based on this radix-8 scheme generates fewer partial products than a radix-4 multiplier, but the computation of each partial product is more complex [11]. In particular, a partial product corresponding to an encoding x=+3 requires the computation of 3x, and therefore a full addition.

Table 3.Recoding in Booth Radix-8 Algorithm

Quartet value	Signed-digit value	Quartet value	Signed-digit value
0000	0	1000	-4
0001	+1	1001	-3
0010	+1	1010	-3
0011	+2	1011	-2
0100	+2	1100	-2
0101	+3	1101	-1
0110	+3	1110	-1
0111	+4	1111	0

B. KOGGESTONE ADDER

Each vertical stage produces a "propagate" and a "generate" bits as shown. In each step radix-2 gray cells are generated. By using the gray cells directly generate the carry values. The culminating generate bits (the carries) are produced in the last stage (vertically), and these bits are XOR with the initial propagate after the input to produce the sum bits. E.g., the first (least-significant) sum bit is calculated by XOR ing the propagate in the farthest-right black box (a "1") with the carry-in (a "0"), producing a "1". The second bit is calculated by XOR ing the propagate in second box from the right (a "0") with C0 (a "0"), producing a "0". It takes more area to implement than the Brent-Kung adder, but has a lower fan-out at each stage, which increases performance. Wiring congestion is often a problem for Kogge-Stone adders as well.

COMPRESSORS

In order to optimize this stage, compressors can be used for partial product accumulation. Compressors are used for addition operation and they contribute for reduced critical path delay, which is important in maintaining circuit's performance [8].In high speed multiplier, 4-2 compressors have been widely used to lower the latency of the partial product reduction stages [6].

The problems of this kind of conventional compressor are:

(i) The uneven delay profile of the outputs arriving from different input paths tends to generate a lot of glitches.

(ii) Compressors do the simple operation of addition that adds more number of bits at a time. But the conventional 4-2 compressors require one more half adder of which two inputs are 'COUT' and 'C', to produce the final addition result. Example: if X1=X2=X3=X4=1 and CIN =0 then the addition result be four (i.e) 100 but the conventional architecture produces COUT=1, C=1 and S=0. Now if COUT and C fed to a half adder then it produces the final result in exact form as shown in Figure 6.

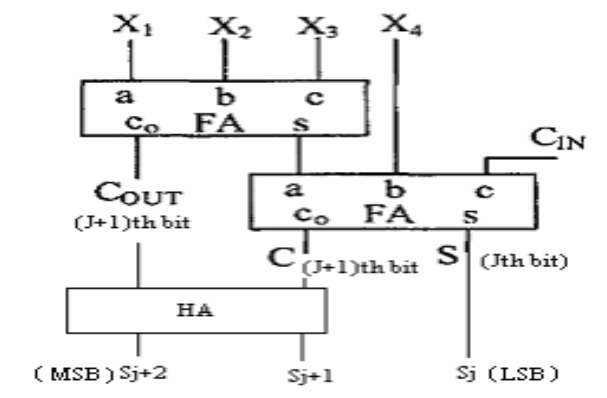

Fig.6 Modified 4:2 compressor [2]

C. HAN-CARLSON ADDER

It is a family of networks between kogge-Stone and Brent-kung adders. In this adder, the wiring congestion is reduced and area occupied will be less when compared to the previous adder.

D. PIPELINING

As the number of stages increases, the power consumption and area gets increased. Thus, pipeline technique can be introduced in Wallace Tree in order to improve the performance.

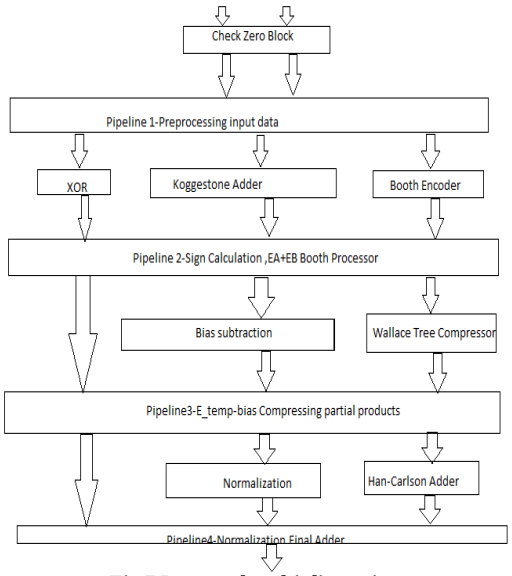

Fig.7 Proposed multiplier unit

V.CONCLUSION

The number of partial products gets reduced by using Radix-8 booth algorithm and the wiring congestion is minimized by using Han-Carlson adder. Finally the area gets reduced as well as the critical path gets reduced by using compressors.

REFERENCES

1.Sunesh N.V, Sathishkumar P,"**Design And Implementation Of Fast Floating Point Multiplier Unit**", International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI-SATA)-2015.

2. A. Dandapat, S. Ghosal, P. Sarkar, D. Mukhopadhyay, **"A 1.2-ns16×16-Bit Binary Multiplier Using High Speed Compressors"**, International Journal of Electrical and Electronics Engineering-2010.

3. Gary w. bewick **, "Fast multiplication:algorithms and implementation"** ,Thesis-1994.

4. P.V.Krishna Mohan Gupta,Ch.S.V.Mar uthi Rao, G.R. Padmini, **"An Efficient Implementation of High Speed Modified Booth Encoder for Floating Point Signed & Unsigned Numbers"** , International Journal of Engineering Research & Technology (IJERT)Vol. 2 Issue 8, August – 2013.

5. Ushasree G, R Dhanabal, Sarat Kumar Sahoo **" Implementation of a High Speed Single Precision Floating Point Unit using Verilog",** International Journal of Computer Applications ,National conference on VSLI and Embedded systems- 2013.

6. Marimuthu, R., Dhruv Bansal, S. Balamurugan and P.S. MALLICK **"Design of 8-4 and 9-4Compressors for high speed multiplication"**, American Journal of Applied Sciences 10 , 2013.

7.B. Jeevan, S. Narender, Dr C.V. Krishna Reddy, Dr K. Sivani **"A High Speed Binary Floating Point Multiplier Using Dadda Algorithm"** ,International journal-2012.

8. P.N.V.K. Hasini, T. Krishna Murthy, **"A Novel high-speed transistorized 8x8 Multiplier using 4-2 Compressors"** , International Journal of Engineering Research and General Science Volume 3, Issue 2, Part 2, March-April, 2015.

9. Preethi Sudha Gollamudi, M. Kamaraju, "**Design Of High Performance IEEE- 754 Single Precision (32 bit) Floating Point Adder Using VHDL",** International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 7, July – 2013.

10. Mohamed Al-Ashrafy, Ashraf Salem, Wagdy Anis, **"An Efficient Implementation of Floating Point Multiplier"**, IEEE-2011.

11. Minu Thomas**, "Design and simulation of Radix -8 Booth Encoder Multiplier for signed and Unsigned numbers"** , International Journal for Innovative Research in Science & Technology, Vol. 1, Issue 1, June 2014.

C.M.KALAISELVI received the B.E., degree in Electronics and Communication Engineering from oxford Engineering College ,Trichy under Anna University, Chennai in 2012, where she is currently pursuing the M.E. degree in VLSI Design.

V.P.DHIVYAPRIYA received the B.E., degree in Electronics and Communication Engineering from Velalar college of Engineering and Technology, Thindal. under Anna University, Chennai in 2014, where she is currently pursuing the M.E. degree in VLSI Design.

P.NITHYA received the B.E., degree in Electronics and Communication Engineering from Anna University, Chennai in 2015, where she is currently pursuing the M.E. degree in VLSI Design