International Journal of Advanced Technology in Engineering and Science
Vol. No.5, Issue No. 03, March 2017
www.ijates.com
ijates
ISSN 2348 - 7550

LOW POWER AND AREA EFFICIENT MULTIPLIERS FOR DIGITAL SIGNAL PROCESSING

Md.Zakir Hussain[1], Firdous[2], Samreen Begum[3]

[1]Asst.Professor, [2,3]B.E. Student ECED Muffakham Jah college of Engineering and Ttechnology , Hyderabad, (India)

ABSTRACT

All digital devices which deal with audio, speech, image, signal processing, multimedia content consists of one or more multiplier circuits. The multiplier is the basic and the most crucial unit of any Digital Signal Processor (DSPs). There are numerous algorithms and techniques used to perform multiplication of two numbers in any processor. In this paper we present a detailed study on three multipliers i.e. Booths Multiplier, Floating point Multiplier and Q Point Multiplier. Speed, area and precision play a major role in any multiplier. We have studied various multipliers in order to design the most efficient multiplier in terms of power consumption, speed, area and precision which can be used in several Signal processing applications.

Keywords: Multiplier, Booths Multiplier, Floating point multiplier, Q point multiplier.

I. INTRODUCTION

Multiplication is a series of periodic additions. The number which is to be added is called as the multiplicand, and the number of times it is added is called as the multiplier, and the result is named as the multiplication result or product [1]. Partial products are formed after every step of addition. If the operands are integers, then the product is twice the length of the operands. Multiplication through repetitive addition method is very slow and consumes more power, hence there arises the need to implement a faster multiplier which consumes less power, area and can give high accuracy with high speed.

II. TYPES OF MULTIPLIERS

A. Booths Multiplier:

Andrew Donald Booth invented the Booth's algorithm in the year 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. Booth had used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed [2]. Booth algorithm is deployed in the multiplier block hence it is called Booths multiplier.

Booth algorithm is a method that will reduce the number of multiplicand multiples. Booth's Algorithm gives a method to multiply binary integers in signed 2's complement representation. It has the ability to both add and subtract in order to compute a product. Booth's algorithm preserves the sign of the end result. While doing multiplication, strings of 0s in the multiplier are directly shifted without performing any arithmetic operation. Whereas the strings of 1s in the multiplier require an arithmetic operation at every end. The string of zeroes and ones in the multiplier are skipped without any addition or multiplication being performed, thereby making it a faster multiplier. We need to merely add or subtract when there is a switch from 0 to 1 or from 1 to 0.

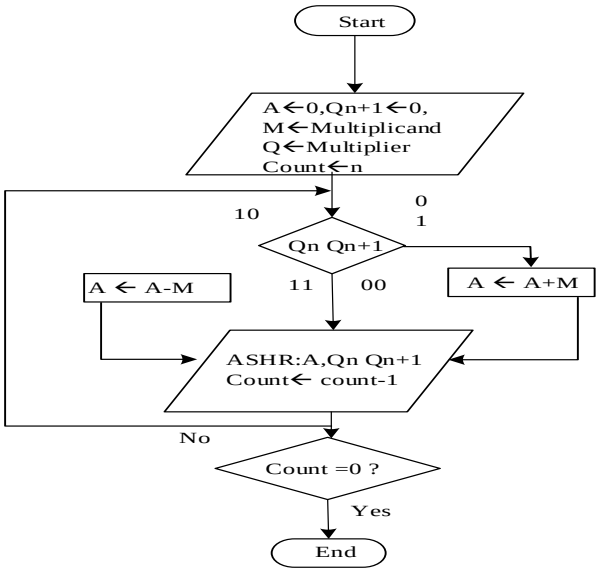

Fig. 1. Flowchart of Booths Algorithm

The above flow chart depicts the booth flow chart and its working is as shown below:

1. Booths Algorithm requires examination of multiplier and shifting of partial products.

2. Prior to shifting, the multiplicand can be added to or subtracted from the partial product or kept unchanged according to the following rules: -

a) The Multiplicand is subtracted from the partial product upon encountering the first L.S.B '1' in a string of ones in the multiplier.

b) The Multiplicand is added to the partial product upon encountering the first '0' (provided there was a previous '1') in a string of zeroes in the multiplier.

c) The partial product remains unchanged when the multiplier bit is identical to the previous multiplier bit [3].

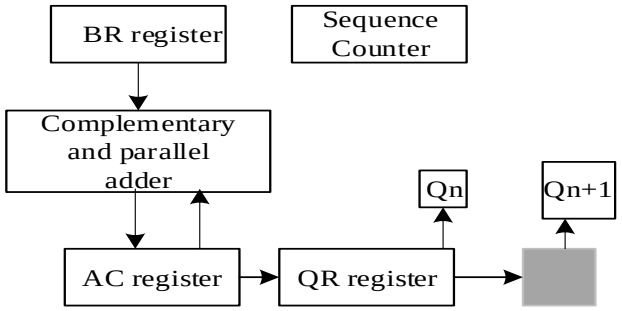

Fig. 2. Hardware for Booth Multiplier

The above figure 2 depicts the hardware architecture of Booths Multiplier. It consists of several registers, adder and flip flop.

AC register is used for storing the Accumulator value.BR and QR registers store Multiplicand and Multiplier values respectively. Qn designates the least significant bit of the multiplier in the register QR. flip flop Qn+1 is appended to QR to facilitate a double bit inspection of the multiplier.

Example:

Multiply: $(0100)_2$ and $(0110)_2$

Here we take the multiplicand as M = $(0100)_2$ and the multiplier is taken as Q = $(0110)_2$, the following Booths table is constructed based on the above steps.

TABLE I. MULTIPLICATION USING BOOTH ALGORITHM

M	A	Q	Qn+1	Count
0100	0000	0110	0	100
	0000-0110= 1100	0011	0	011
	1110	0001	1	010
	1111+0100=0011	0000	1	001
	0001	1000	0	000

B. Floating point multiplier:

Many processing applications command the computations of real numbers that aren't integers [4]. The computer processor can understand only binary language, so the real numbers, negative numbers, non-integer numbers should be represented in Binary format. Floating point representation gives us a notation for representing real numbers in binary format. In this representation the radix point of the number (the point which separates the integer part from the fractional part) can be placed anywhere relative to the significant digits of the number i.e. the radix point can "float" [5] which means that the number of bits are not reserved for integer and fractional part. A floating point number is represented by making use of a significand which is scaled using an exponent [6].

The base for scaling generally is the base of the number system which we are dealing with, in binary number system the base is taken as 2 , for decimal number system the base is taken as 10.

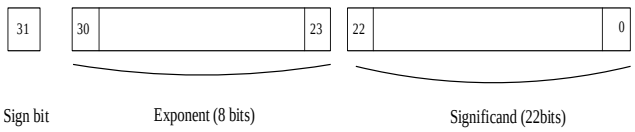

Fig. 3. IEEE single precision floating point format

The above figure 3 depicts IEEE 32bit single precision binary format representation which consists of a sign bit, exponent and a significand or mantissa.The MSB (31st bit) of the number represents the sign bit(S), if the sign bit is 1 then it represents a negative number or wise if sign bit is 0 then it represents a positive number. The bits numbered from 0 to 22 are stored to represent the Mantissa and the bits numbered from 23 to 30 are stored to represent the exponent. A floating point number can be written in the following notation

$$significand * base^{exponent}$$

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

Where S: sign bit

E: exponent

(1.M): mantissa along with the hidden bit.

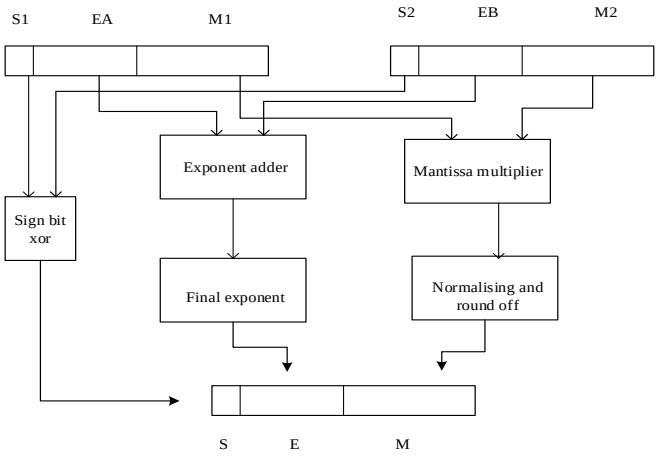

Fig. 4. Block Diagram of Floating point Multipliers

The figure 4 depicts the procedure of performing multiplication of two floating point numbers

1. Determine the sign of the number; i.e. s1 xor s2
2. Multiply both the mantissa's; i.e. (1.M1*1.M2)
3. Place the decimal point at the mantissa's product.
4. Add the exponents of the two numbers; i.e. (E1 + E2)
5. Normalize the result; i.e. obtaining 1 at the MSB of the result's mantissa.
6. Round off the result value within the available bits.
7. Check for underflow/overflow possibility.

Example:

Multiply 40 and -7.5 [6].

Let A=40 be the multiplicand and B= -7.5 be the multiplier.

A= 0 10000100 0100 = 40

B= 1 10000001 1110 = -7.5

Y= 1. 0100 * 1. 1110 = 1001011000 (Multiplying the mantissa's)

Place the decimal point: 10. 01011000

Add the exponents: 10000100 + 10000001 = 10000101

EA+EB = EA-true + EB-true + 2 bias

Obtain the sign bit and assemble the result:

1 10000110 10.01011000

Normalize the result:

110000110 10. 01011000 (before normalizing)

1 10000111 1.001011000 (normalizing)

The final product is

1 10000111 00101100 (without the hidden bit)

C. Q Format Multipliers:

Floating point multipliers consume more silicon area and are relatively slower than the fixed point (Q-format) multipliers. An N-bit fixed point number can be represented as either an integer or a fractional number. Integer fixed point is not suitable in processors due to possible overflow. For e.g. In a 16-bit processor for signed integers the dynamic range is from -2∧ 15 to -2∧ 15-1 i.e. 32768 to 32767. If 300 is multiplied by 400 the result is 120000 which is an overflow. In order to overcome such situations fractional fixed point representation also known as Q-format is used [7].The labelling convention is as follows:

Q[QI][QF]

where QI= is of integer bits (also includes sign bit for integer values) & QF= of fractional bits.

1) Q-format Representation

Q-format representation is denoted by Q*m. n*, where *m* is the number of bits to represent the integer, *n* is the number of bits to represent the fractional part and the total number of bits is given by N = *m*+*n*+1 for signed numbers.

For e.g. Q4.11 format specifies that a total of 16 bits are required to represent the fractional number in which 4 bits are reserved for the integer part and 11 bits are for the fractional part and 1 bit indicates the sign bit. Some cases of Q-format consist of zero bits to represent the integer part. Q0.15 (Q15) and Q0.31 (Q31) are two such formats for 16 bit and 32 bit representations respectively. As there is no integer part the fractional number has a range between 1 and -1. Hence the products of such numbers also lie between 1 and -1. This property is best suited for implementing multipliers since the bit length of the product is same as the input bit length and thus Q-format finds its application in digital signal processing hardware.

2) Q-format Multiplication

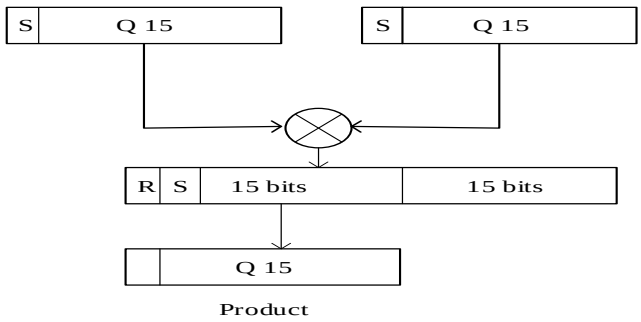

Fig. 5. Multiplication of two Q15 format numbers

Fig.5. demonstrates multiplication of two Q15 format numbers. If two Q15 numbers are multiplied and we want to have the product in Q15 form. The required operation is to take the Q30 product, and shift it right by 15 bits. The result can then be stored in 16 bits. There is also the option of rounding the product before shifting out the lower 15 bits. The product consists of a redundant sign bit.

In Q-format, multiplications of two fractional numbers can be carried out by using integer multiplications. Integer multiplications consume less area and are faster compared to floating point multipliers which is the major advantage of Q format representation [1]. The resultant value is truncated or rounded off to the nearest integer. Therefore, a small amount of precision loss is involved which reduces as the number of bits representing the fractional part. Therefore, the rounded value will be more precise.

III. RESULTS AND DISCUSSION

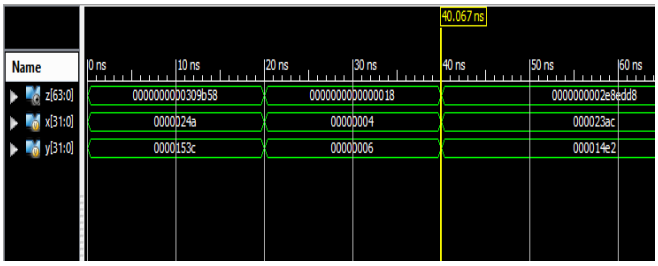

Fig. 6. Simulation result of Booth Multiplier

Figure 6 depicts the simulation results of Booths Multiplier where x, y are given as the inputs of 32 bits and z is output of 64 bits.

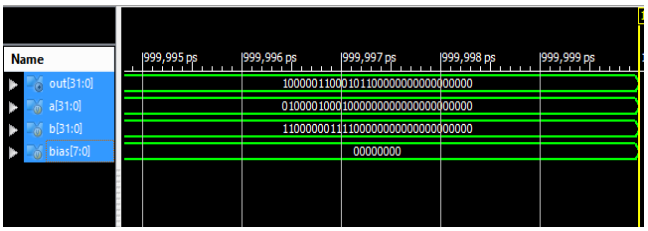

Fig. 7. Simulation result of Floating point Multiplier

The above figure 7 depicts the simulation results of Floating point multipliers using a and b as 32 bit inputs , 8 bit bias and out is output of 32 bit.

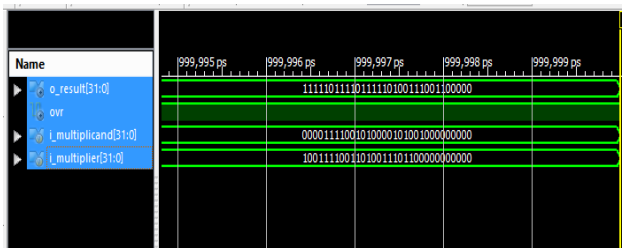

Fig. 8. Simulation result of Q Point Multipliers

The above figure 8 depicts the simulation results of Q point multiplication where the inputs are i-multiplicand and i-multiplier of 32 bits and we obtained the output as o-result of 32 bits.

TABLE II. COMPARISON OF MULTIPLIERS

Parameters	Booth Multiplier	Floating Point Multiplier	Q-point Multiplier
Device Used	Vertix5-Xc5vfx30t	Vertix5-Xc5vfx30t	Vertix5-Xc5vfx30t
No.Of LUTs Used	3174	28	4
No. Of Bonded Iobs	128	104	97
Path Delay (Ns)	48.34	5.537	8.727

In above comparison table, different multipliers like Booth multiplier, Floating point multiplier and Q Point multiplier are compared and by observing different parameters of each multiplier and comparing them, it can be concluded that Q-point multiplier is efficient and it produces less delay and consumes less power for processing its multiplication operation.

IV. CONCLUSION

Our main aim is to develop a processor where multiplier used, should be efficient enough so as to consume less power and make processor operations speed and accurate. By observing the above three multipliers, it is observed that booth requires more operation for partial products and floating point multiplier provides it rounding off large numbers, whereas in Q Point, number of bits can be represented exactly and it reduces area and as a result number of operations are reduced and produces less delay. Thus Q Point multiplier is efficient which can reduce delay and can be extensively used for low power processors applications.

REFERENCES

[1] International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com (ISSN 2250-2459, ISO 9001:2008 Certified Journal, Volume 3, Issue 3, March 2013) Booth Multiplier: Ease of multiplication.

[2] en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm.

[3] Computer system architecture by M.Morris Mano, California State University, Los Angeles.

[4] Efficient Floating Point 32-bit single Precision Multipliers Design using VHDL Dr. Raj Singh, CEERI, Pilani

[5] 2015 International Conference on Soft Computing Techniques and Implementations-(ICSCTI)Area and Time Optimized Realization of 16 Point FFT and IFFT Blocks by using IEEE 754 Single Precision Complex Floating Point Adder and Multiplier.

[6] International Journal of VLSI and Embedded Systems-IJVES Normalisation of floating point multiplication using VHDL.

[7] High Speed Signed Multiplier for Digital Signal Processing Applications ieeexplore.ieee.org/iel5/6186902/6224337/06224373.pdf.

[8] Sen-Maw Kuo and Woon-Seng Gan, "Digital Signal Processor, architectures, implementations and applications," Pearson Prentice Hall, 2005, pp. 253-3