INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume3, Issue2)

Analyzing Two-Term Dot Product of Multiplier Using Floating Point and Booth Multiplier

[1]Mukesh Krishna
Department Electrical and Electronics Engineering
Students, Dr. Mahalingam College of Engineering and Technology, Udumalai Road, Pollachi.

[2]Mohana Priya .S
Department Electrical and Electronics Engineering
Students, Dr. Mahalingam College of Engineering and Technology, Udumalai Road, Pollachi.

[3]ManickaVasagam .P
Department Electrical and Electronics Engineering
Students, Dr. Mahalingam College of Engineering and Technology, Udumalai Road, Pollachi.

Dr. B. Vinoth Kumar
Assistant professor
Department Electrical and Electronics Engineering,
Students, Dr. Mahalingam College of Engineering and Technology, Udumalai Road, Pollachi.

Abstract: The Floating Point in two-term Dot-Product of multiplier referred as discrete design. Floating Point is a wide variety for increasing accuracy, high speed, high performance and reducing delay, area and power consumption. This application of floating point is used for algorithms of Digital Signal Processing and Graphics. Many floating point application is to reduce area, from the survey the fused floating point gives better performance using both the single precision and the double precision in multiplication, addition, and subtraction. The scientific notations sign bit, mantissa and exponent are used. The real numbers are divided into two, fixed component of significant range (lack of dynamic range) and exponential component in floating point (largest dynamic range). Converting decimal to fused floating point and normalize the exponent part and rounding operation for reducing latency. The other operation is compared by booth multiplication. It is also used for reduction of area and power. That multiplies two signed binary numbers in two's complement notation, it used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed, then both the results are verified in verilog hardware description Language.

Keywords: Dot-Product Unit-Fused Floating-Point Operations, Booth Multiplier Normalization, Rounding Operations, Latency, and VHDL.

I.INTRODUCTION

The demand of floating point multiplier is more in Three-Dimensional (3D) array and also used in graphics and image processing. Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT) and Butterfly operations are needed floating point numbers [1]. Due to output data size is twice larger than the input data size so complexity, area and time are consumed by the multipliers. The best design challenge to get high speed working is in Field Programmable Gate Array (FPGA). The floating point shows the base, the location, the precision and it normalized or not. There are many models for multiplication floating point. Precision is the main role in floating point. We deal with both single and double precision floating point. The main significant of floating point number are (Sign bit * Mantissa * Base $^{\text{Exponent}}$). The single precision has 24-bits which contain 0 to 31, left to right and double precision has 64-bits which contain 0 to 63, left to right [2]. The difference of this two precision is data, the double precision has twice the data of RAM, Cache and Band Width and reduces the performance. The result of sign bit by XOR and carry save adder used for two exponent components.

1.1 Data formats for single and double precision

Sign bit S	**Biased Exponent 8 bit – E**	**Unsigned fraction 23 – bits P**

a) Single Precision Data Format

Sign bit S	**Biased Exponent 11 bit – E**	**Unsigned fraction 53– bits P**

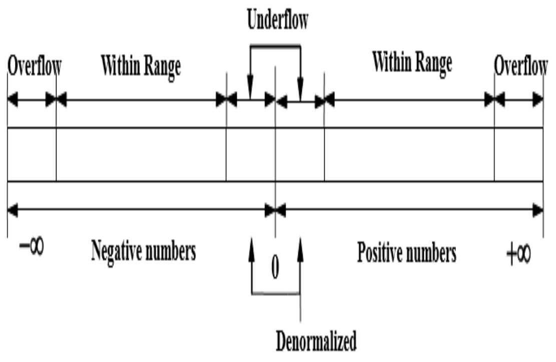
b) Double Precision Data Format

1.2 Format Parameters

The implementation of hardware and software has basic IEEE format. In the standard IEEE format, the floating points are in binary number. The binary floating point numbers are single precision and double precision. The single precision contains 32 bits and the precision which add the fraction and hidden bits 23+1, exponent bit 8 is used. The maximum and the minimum values from +127 to -126.

For the double precision, contain 64 bits and the precision which has 52+1, exponent bit is 11 is used. The maximum and the minimum values from -126 to -1022. For quadruple precision, hidden bits are 112+ 1 and the maximum and the minimum value from -16382 to + 16383 [3].

1.3 Representation of the floating point



Denormalized

Exponent part contains zeros and fraction or significant contain non-zeros denormalized is taken. Denormalized occur in zeros and lower normalized range [3]. Zero is a special value for exponent field all zeros and fraction zeroes.

Overflow

Overflow occur limited range in smallest value and higher range in highest value. It indicates the range when reaching extreme value. It doesn't show the indication when one operand is infinity. It must have the exact range [4]. When the result reaches extreme range, bias should adjust and a NaN is delivered instead.

Underflow

Underflow takes place when floating point is smaller than the smallest value. It may be negative or positive exponent from -128 to 127, when lesser than -128 underflows occur. The result may be zero or denormal [4]. There is a loss of accuracy after the denormalized numbers. Underflow adjust the result from overflow delivery.

Infinity

The value of -infinity and +infinity used in exponent 0s and 1s. Sign bit for positive 0 and negative 1 are used. It denotes infinity as a special value for operations to continue past overflow situations. It used undefined operations [5].

Not a Number

It is an invalid value when does not show the real number representation. The exponent has 1s and the fraction has non-zeroes are taken in NaNs [6].

II. APPROACH

According to IEEE 754 2008 supports the floating point multiplier which has efficient carry saver. For the high performance of multiplier, pipelining stages are used to increase operating frequency multiplier [5]. Here two approaches are seen for dot product unit, floating point adder, and multiplier. This greatly improves the performance. By rounding operation the addition cycle eliminated and directly performs the multiplication [6]. For the exact result of floating point operation rounding is needed.

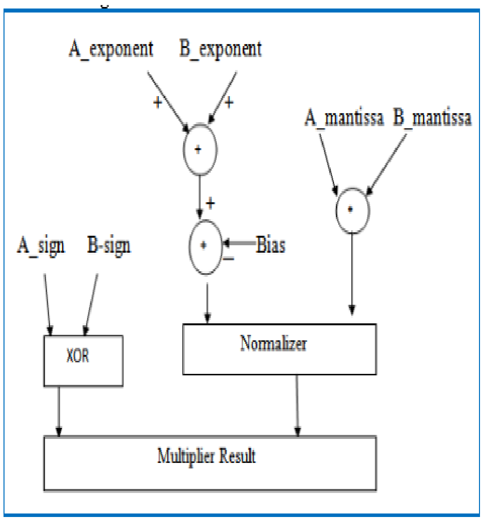
Booth multiplier is another multiplication to reduce latency and power consumption. Its algorithm serves 2 processes. The output product of the multiplier can decrease power by sacrificing a bit of output precision. To combine the addition, sign bit and multiplier a modified output result is developed.

The Different Modes Used Are

Rounding Mode	**Encoding**	**Unrounded**	**Rounded**
Nearest Even	00	3.4	3
Zero	01	5.6	6
Positive Infinity	10	3.5	4
Negative Infinity	11	2.5	2

III. FLOATING POINT MULTIPLICATION OPERATION

The fused dot product derived from floating point add subunit. They were done separately and multiplexers choose to add and sub with XOR process [7]. Converting decimal number to the floating point number, block diagram for floating point multiplier.



c) Block diagram of floating point multiplier

3.1 FLOATING POINT ALGORITHM

1. Converting the value to binary, take fractional part for separating the integral value and fractional value. This fractional part is converting by multiplication [8]. Multiply by 2 repeatedly and harvest each one bit. It shows the decimal value to floating point.
For Example:
Convert 2.625 to floating Point format:
The integral part is $2_{10} = \mathbf{10_2}$
The fractional parts are
0.625*2= 1.25 – **1**
0.25 *2= 0.5 -- **0**
0.5 *2= 1.0 -- **1**
For $0.625_{10} = 0.101_2$ and $2.625_{10} = \mathbf{10.101_2}$

2. Adding an exponent part to a binary number: One possibility for handling numbers with fractional parts is to add bits after the decimal point. The first bit after the decimal point is the halves place, the next bit the quarter's place, the next bit the eighth place and to split the bits of the representation between the places to the left of the decimal point and places to the right of the decimal point. For example, a 32-bit fixed-point representation might allocate 24 bits for the integer part and 8 bits for the fractional part. The Product of Append and 2 power Exponent in the end of binary numbers.

$10.101_2 = 10.101_2 * \mathbf{2^0}$

3. Normalization

The mantissa of a floating point number represents an implicit fraction whose denominator is the base raised to the power of the precision. Since the largest represent mantissa is one less than this denominator (base raise to the power of the precision), the value of the fraction is always strictly less than 1. The mathematical value of a floating point number is then the product of this fraction, the sign, and the base raised to the exponent. If the number is not normalized, then you can subtract 1 from the exponent while multiplying the mantissa by the base, and get another floating point number with the same value. Normalization consists of doing this repeatedly until the number is normalized. Two distinct normalized floating point numbers cannot be equal in value.

4. The value doesn't change when exponent adjust to one bit left in binary number. $10.101_2*2^0 = 1.0101_2*$ **2^1**

5. Mantissa: Mantissa or significant is next to leading number which filled with zeroes on the right. When working in binary, the significant is characterized by its width in bits. Because the most significant bit is always 1 for a normalized number, this bit is not typically stored.
Depending on the context, the hidden bit may or may not be counted towards the width of the significant. For example, the same double precision format is commonly described as having either a 53-bit significant, including the hidden bit or a 52-bit significant, not including the hidden bit. The notion of a hidden bit only applies to binary representations.
Mantissa – **0101**

6. Now add the bias to the exponent of 2 in exponent field, for all biasing $2^{k-1} – 1$, k=number of bits in exponent field.
For example
8 – bit format $2^{3-1} -1 = 3$
32 –bit format $2^{8-1} -1 = 127$
Here the exponent power is 1, so
$1 + 3 = 4 = 100_2$

7. The sign bit of negative is 1 and positive is 0 of given number.
For 8 bit and 32 bit

SIGN BIT	MANTISSA	EXPONENT
0	100	0101
0	10001001	01010000000000000000000

3.2 MULTIPLICATION OPERATION

Multiplying the two input values after the normalization. The sign, exponent, and mantissa are taken separately. The multiplication must take account of the integer part, implicit in normalization. The number of bits of the result is twice the size of the operands (48 bits). Biasing is done because of exponent have to be signed values in order to represent both tiny and huge values. The exponent is biased before stored, by adjusting its value to put it within an unsigned range suitable range for comparison.

Take 2 floating point number A= -18.0 and B= 9.5

Their binary values are A= -10010.0

B= +1001.1 and [9] the normalization

$A= -1.001*2^4$ $B= +1.0011*2^3$

1	10000011	00100000000000000000000
0	10000010	00110000000000000000000

3.3 Multiplication of Mantissa

We must extract the mantissa, adding 1 as a most significant bit, for normalization the result of the multiplication only the most significant bits are useful: after normalization (elimination of the most significant 1), we get the 23-bit mantissa of the result. This normalization can lead to a correction of the result's exponent. The multiplication is more complex for floating point and integer. The multiplication of mantissa is given in the table. For normalization [10] adding 1 to the most significant bit is useful,

A	100100000000000000000000
B	100110000000000000000000

01	01010110000000000000000	00000000000000000000000

The result is in 48 bit: 0*558000000000

3.4 Adding the exponents

The exponent of the result is equal to the sum of the operands exponents. A 1 can be added if needed by the normalization of the mantissa multiplication. For mantissa multiplication remove bias in two operands and add again the bias [10].

E result= (Ea-127) + (Eb-127) +127   E r= Ea+ Eb-127 then E r= 10000110

3.5 Calculation

The setting of these intermediate results sign, exponent and mantissa gives us the final result of our multiplication. The Sign result Sr by EXOR of two operands Sa= 1 and Sb= 0 is

$Sr = Sa \oplus Sb$ $Sr = 1 \oplus 0 = 1$

The final result of sign, exponent and mantissa are

1	10000110	01010110000000000000000000

A*B = -18.0* 9.5

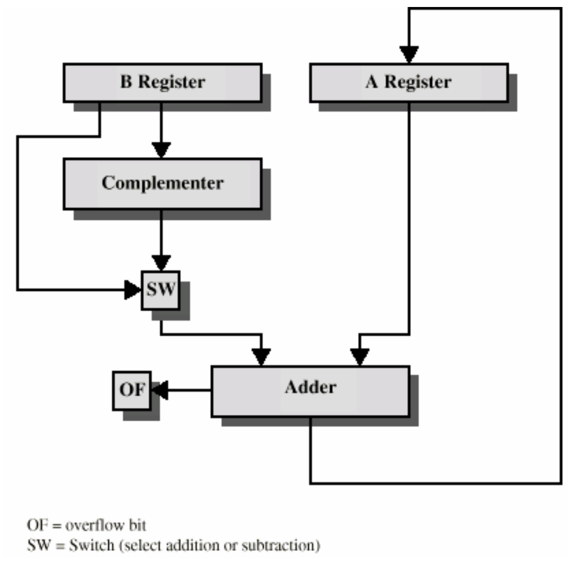
$= -1.0101011 * 2^{134-127}$

= -10101011.0

$= -171.0_{10}$

IV. BOOTH MULTIPLICATION OPERATION

When using Booth's Algorithm twice as many bits product as having in original two operands. The left the most bit of your operands (both your multiplicand and multiplier) is a sign bit, and cannot be used as part of the value.



Block diagram of Booth multiplier

Decide which operand will be the multiplier and which will be the multiplicand. Convert both operands to two's complement representation using that bits must be at least one more bit than is required for the binary representation of the numerically larger operand [11]. Begin with a product that consists of the multiplier with an additional X leading zero bits.

4.1 ALGORITHM FOR BOOTH MULTIPLICATION

The procedure for booth multiplication is if x is the count of bits of the multiplicand and y is the count of bits of the multiplier. A- Add, S- Subtract and P- Product.

In two's complement notation, fill x bit of each line as A is multiplicand, S is negative of multiplicand and P is zeros. Then fill y bits of each line with A is zeros, S is zeros and P are multiplier and fill last bit of each line with zeros [12].

If the last two bits of product which possible arithmetic actions are:

00 → no arithmetic operation

01 → add multiplicand to left half of product (P=P+A)

10 → subtract multiplicand from left half of product (P+P+S)

11 → no arithmetic operation

Then arithmetic operations are continued for y times, which is a multiplier. Add the leading zeros in the left half of the product, when possible arithmetic operations occur.

4.2 MULTIPLICATION OPERATION

Let us take 2 values for multiplying

(-11) * 4

-11 is (multiplier)

4 is (multiplicand)

And we added 5 leading zeros to the **multiplier** to get the **beginning product**.

Initial Product and previous LSB

A=1 0101 0000 0

S=0 1011 0000 0

P=0 0000 1100 0

Perform the loop for 4 times because the multiplier or the y term is 4.

1. **P=0 0000 1100 0**

The last 2 bits are **00**

Now right shift arithmetic takes place,

P=0 0000 0110 0

2. **P=0 0000 0110 0**

The last 2 bits are **00**

Now right shift arithmetic takes place,

P=0 0000 0011 0

3. **P=0 0000 0011 0**

The last 2 bits are **10**

Now add the subtract value to product,

P=P+S

SYMBOL	ADDITION
P	**0 0000 0011 0**
S	**0 1011 0000 0**

P=0 1011 0011 0

If the last two bits 01 product should be added to add value and takes right shift arithmetic.

4. **P=0 1011 0011 0**

After adding take right shift arithmetic,

P=0 0101 1001 1

The last 2 bits are **11**

Now right shift arithmetic takes place,

P=0 0010 1100 1

The Product is **0010 1100** now neglect the first and last bit. The result is **44** [13].

V. SIMULATION

Floating point calculation and booth multiplication design have been simulated in VIVADO, create a design in work library as 45nm CMOS study cell and compile the design by going to simulation and start to simulate to run, before that source code in VHDL. VHDL is a high-level language and used for digital circuits and systems. This representation allows easy description and direct functionality through simulation. By giving the signal value, two 8 bits are multiplied and resulted in 16-bit conversion. The schematic diagram also elaborated. The Spartan-3E High Volume Starter Kit gives designers instant access to the complete platform capabilities of the Spartan-3E family. Using this result is verified. Complete kit includes board, power supply, evaluation software, resource CD. A few system-level design trade-offs were required in order to provide the Spartan-3E Starter Kit board with the most functionality.

5.1 Power Synthesis

The output power of the floating point multiplier is executed in VIVADO, the resulted power is 42.134 W synthesised. This power may be a change in on-chip power. The floating point multiplier results derived from the combination of the sign, mantissa and exponent values and mainly from constraints files, simulation files or vector less analysis. Device static power is the power from transistor leakage on all connected voltage rails and a function of process, voltage, and temperature. This represents the steady state, intrinsic leakage in the device. Design power is the power of the user design, due to the input data pattern and the design internal activity. This power is instantaneous. This also includes static current from I/O terminations, clock managers.

5.2 Comparison of array multiplier and booth multiplier

Synthesis	**Floating point multiplier**	**Booth Multiplier**
Area Used	**7338.636 (μm^2)**	**9493.916 (μm^2)**
Delay	**119-51**	**132-68**
Leakage Power	**156.179 (nw)**	**172.532 (nw)**
Dynamic Power	**389775.698 (nw)**	**579122.171 (nw)**
Total Power	**398931.878 (nw)**	**579294.703 (nw)**

Thus the comparison of both floating point multiplier and the booth multiplier are executed in VHDL and verified in Cadence digital design which is an automation machine tool. The verified results are showing that the area, power and delay timing are reduced than the booth multiplication. Thus, the floating point array multiplier having the greater efficiency and more accuracy than booth multiplier.

CONCLUSION

The floating point multiplier is varied by 32-bit inputs. The simulation for both array multiplier and booth multiplier design and implementation are analyzed. The array multiplier floating point is faster speed than the booth multiplier. Comparing the floating point is higher performance than discrete values. The delay and silicon area are reduced and gives high speed.

Sometimes more performance in dual path reduction and pipe lining are used. The more accurate results are performed by eliminating the rounding modes and normalization. The results are executed in VHDL simulation VIVADO. As a result, the area, latency and power consumption are reduced by about 40% compared to the array multiplication [14]. The more accurate results are performed by eliminating the rounding modes and normalization. The fused floating points are used for processors, system controllers, and hardware.

REFERENCE

[1] 32-bit Single Precision floating point Multiplier, Ms.Radhika Jumde, AVBIT, Pawnar, Wardha.
[2] IEEE Standard for floating point arithmetic, IEEE Standard 754-2008, New York, Inc., Aug.29, 2008.
[3] Saleh and E.Swartzlander, Jr., "A floating point fused dot product unit," in Proc. IEEE Int. Conf. Computer Design, 2008, pp. 427—431.
[4] Design and Simulation of Binary floating point multiplier using VHDL, U.V. Chaudhari and Prof.A.P.Dhande, Feb-2015.
[5] Simulation and Synthesis for multipliers using VHDL, Raj Kumar Singh, Shivanada Reddy, 2008.

[6] IEEE 754 floating point fused add sub unit, Sharmila Hemanandha, Siva Subramanian, Aug-2015.
[7] Floating point fused dot –Product Unit, Kishore, Prakash, May-2015.
[8] Floating Point Adder and Multiplier, Eduardo Sanchez EPFL- HEIG- VD. Aug-2013.
[9] Normalization on floating point Multiplication using Verilog HDL, V.Narasimha, V.Swathi,
[10] Multiplication of floating point numbers using VHDL, Ms. Sobin Daniel, Sep-2014.
[11] A floating point multiplier, Concordia University.
[12] Design and Implementation of different multiplier using VHDL by Moumita Ghosh, Rourkela, 2007.
[13] Design of Booth Multiplier, Vamsi Krishna, Trivedi, Mar-2014.
[14] VHDL Modeling of floating point for VLSI designer Library, Wai-Leong Pang, Kah-Yoong Chan, Sew-Kin Wong, Choon-Siang Tan, July-2012.