RESEARCH ARTICLE OPEN ACCESS

An Effective VHDL Implementation of IEEE 754 Floating Point Unit using CLA and Rad-4 Modified Booth Encoder Multiplier

Subitha. M. B.
Assistant Professor, SNGCET Payyanur,Kerala.

ABSTRACT
Most of the signal processing algorithms using floating point arithmetic, which requires millions of operations per second to be performed. For such stringent requirement design of fast, precise and efficient circuit is needed. This article present an IEEE 754 floating point unit using carry look ahead adder and radix-4 modified Booth encoder multiplier algorithm and the design is compared in terms of speed , area and power consumption. The adder used here will increase the speed and the multiplier is used to reduce power consumption, area and number of partial product get generated. The floating point unit design deals with the detection of exceptions and trapped overflow and underflow exceptions as an integral part of the rounding unit. This work is used to reduce area, power consumption and speed up the operations with more accurate results. The basic methodology and approach are implemented in VHDL (Very Large Scale Integration Hardware Description Language).

Keywords-FP multiplier, MBE.

I. INTRODUCTION

Multiplication is one of the basic functions used in digital signal processing (DSP). It requires more hardware resources and processing time than addition and subtraction. In fact, 8.72% of all instructions in a typical processing unit are multiplier. In computers, a typical central processing unit devotes a considerable amount of processing time in implementing arithmetic operations, particularly multiplication operations. Most high performance digital signal processing systems rely on hardware multiplication to achieve high data throughput. Multiplication is an important fundamental arithmetic operation. Multiplication-based operations such as Multiply and Accumulate (MAC) are currently implemented in many Digital Signal Processing (DSP) applications such as convolution, Fast Fourier Transform (FFT), filtering and in microprocessors in its arithmetic and logic unit. Since multiplication dominates the execution time of most DSP algorithms, so there is a need of high speed multiplier. Currently, multiplication time is still that dominant factor in determining the instruction cycle time of a DSP chip. The multiplier is a fairly large block of a computing system. The amount of circuitry involved is directly proportional to square of its resolution i.e., a multiplier of size of n bits has $O(n^2)$ gates. In the past, many novel ideas for multipliers have been proposed to achieve high performance. The demand for high speed processing has been increasing as a result of expanding computer and signal processing applications. Higher throughput arithmetic operations are important to achieve the desired performance in many real-time signal and image processing applications. One of the key arithmetic operations in such applications is multiplication and the development of a multiplier circuit has been a subject of interest over decades. Reducing the time delay and power consumption are very essential requirements for many applications.

This article presents a modified booth encoder multiplier architecture. Multiplier architectures fall generally into two categories i.e., "tree" multipliers and "array" multipliers. Tree multipliers add as many partial products in parallel as possible and therefore, are very high performance architectures. Unfortunately, tree multipliers are very irregular, hard to layout and hence large. Array multipliers, on the other hand, are very regular, small in size, but suffer in latency and propagation delay. Due to array organization, determining the propagation delay of array multiplier is not straight forward. Multiplier based on Modified Booth algorithm and Wallace addition is one of the fast and low power multiplier. The speed of modified Booth encoder multiplier can further be increased by pipelining.

II. THE IEEE-754 STANDARD FORMATS

IEEE 754 standard is a technical standard established by IEEE and the most widely used standard for floating-point computation, followed by many hardware (CPU and FPU) and software implementations. Single-precision floating-point format is a computer number format that occupies 32

bits in a computer memory and represents a wide dynamic range of values by using a floating point. In IEEE 754-2008, the 32-bit with base 2 format is officially referred to as single precision or binary 32. It was called single in IEEE 754-1985. The IEEE 754 standard specifies a single precision number as having sign bit which is of 1 bit length, an exponent of width 8 bits and a significant precision of 24 bits out of which 23 bits are explicitly stored and 1 bit is implicit 1.Sign bit determines the sign of the number where 0 denotes a positive number and 1 denotes a negative number. It is the sign of the mantissa as well. Exponent is an 8 bit signed integer from 128 to 127 (2's Complement) or can be an 8 bit unsigned integer from 0 to 255 which is the accepted biased form in IEEE 754 single precision definition. In this case an exponent with value 127 represents actual zero. The true mantissa includes 23 fraction bits to the right of the binary point and an implicit leading bit (to the left of the binary point) with value 1 unless the exponent is stored with all zeros. Thus only 23 fraction bits of the mantissa appear in the memory format but the total precision is 24 bits.

The IEEE Standard for Binary Floating Point Arithmetic ANSI/IEEE Std754-2008 is used. Numbers in this format are composed of the following three fields:

1-bit sign, S: A value of '1' indicates that the number is negative, and a '0' indicates a positive number.

Bias-127 exponent, e = E + bias: This gives us an exponent range from Emin = -126 to Emax =127.

Mantissa, M: A twenty three bit fraction, a bit is added to the fraction to form what is called the significand. 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.

III. FLOATING POINT MULTIPLICATION ALGORITHM

Multiplying two numbers in floating point format is done by

1-adding the exponent of the two numbers then subtracting the bias from their result,

2-multiplying the significand of the two numbers, and

3-calculating the sign by XORing the sign of the two numbers. In order to represent the multiplication result as a normalized number there should be 1 in the MSB of the result (leading one).

To multiply two floating point numbers the following is done:

1. Multiplying the significand($1.M_1 * 1.M_2$)
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 result's significand.
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence

Figure 1 shows the multiplier structure; exponent's addition, significand multiplication, and result's sign calculation are independent and are done in parallel. The significand multiplication is done on two 24 bit numbers and results in a 48 bit product, which we will call the intermediate product (IP). The IP is represented as (47 down to 0) and the decimal point is located between bits 46 and 45 in the IP. The following sections detail each block of the floating point multiplier.

IV. HARDWARE OF FP MULTIPLIER

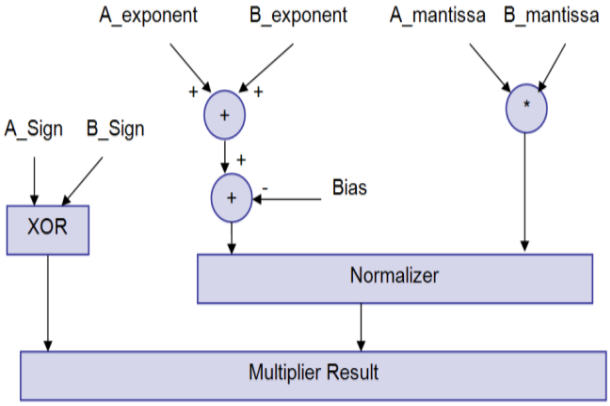

Fig1: Block diagram of floating point multiplier

1 Sign bit calculation

Multiplying two numbers results in a negative sign number if one of the multiplied numbers is of a negative value. By the aid of a truth table it find that this can be obtained by XORing the sign of two inputs.

2 Unsigned adder / subtractor (for exponent addition)

To reduce the delay caused by the effect of carry propagation in the ripple carry adder, we attempt to evaluate the carry-out for each stage (same as carry-in to next stage) concurrently with the computation of the sum bit. The two Boolean functions for the sum and carry are as follows:

Sum = Ai⊕ Bi⊕Ci

Cout =Ai. Bi +(Ai ⊕ Bi)Ci

Let Gi = Ai · Bi be the carry generate function and Pi = (Ai ⊕ Bi) be the carry propagate function, Then we can rewrite the carry function as follows:

Ci+1 = Gi + Pi · Ci

Thus, for 4-bit adder, we can compute the carry for all the stages as shown below:

C1 = G0 + P0 · C0

C2 =G1+ P1·C1 = G1 + P1 · G0 + P1 · P0 · C0

C3= G2+P2·G1 + P2·P1·G0 +P2 · P1 · P0 · C0

C4=G3+P3·G2+P3·P2·G1+P3·P2·P1·G0+P3·P2.P1. P. C0

In general, we can write:

The sum function: $SUM_i = A_i B_i \oplus Ci = P_i \oplus C_i$

The carry function: Carry=Gi+Pi.Ci
Carry Look Ahead Adder can produce carries faster due to parallel generation of the carry bits by using additional circuitry.

3 Multiplier for unsigned data

Multiplication involves the generation of partial products, one for each digit in the multiplier, as in figure 2. These partial products are then summed to produce the final product. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length.

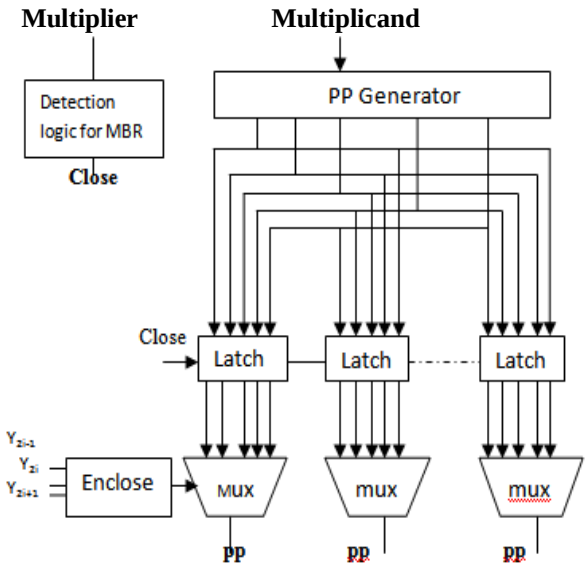

Fig2:Block diagram of low power modified booth multiplier

The Modified booth algorithm (MBA) or Modified booth encoding (MBE) was proposed by O.L Macsorley. The recording method is widely used to generate partial products for implementation of large parallel multipliers, which adopts the parallel encoding scheme. One of the solutions of realizing high speed multipliers is to enhance parallelism, which helps to reduce subsequent stages. The original booth algorithm (radix 2) had 2 drawbacks:

- The number of add subtract operations and the number of shift operations become variable and become inconvenient for designing parallel multipliers.
- The algorithm becomes inefficient when there are isolated 1's.

These problems can be overcome by modified booth algorithm. MBA process three bits at a time during recoding. Recoding the multiplier in higher radix is a powerful way to speed up standard booth multiplication algorithm. In each cycle a greater number of bits can be inspected and eliminated therefore, total number of cycles required to obtain products get reduced. Number of bits inspected in radix r is given by $n= 1+\log_2 r$. Algorithm for Modified booth is given below. In each cycle of radix-4 algorithm, 3 bits are inspected and two are eliminated. Procedure for implementing radix-4 algorithm is as follows,

- Append a zero to the right of LSB.
- Extend the sign bit 1 position if necessary to ensure that n is even.
- According to the value of each vector, find each partial product.

Y_{2i+1}	Y_{2i}	Y_{2i-1}	Recoded Digit	Operand Multiplication
0	0	0	0	0*Multiplicand
0	0	1	+1	+1*Multiplicand
0	1	0	+1	+1*Multiplicand
0	1	1	+2	+2*Multiplicand
1	0	0	-2	-2*Multiplicand
1	0	1	-1	-1*Multiplicand
1	1	0	-1	-1*Multiplicand
1	1	1	0	0*Multiplicand

Table 1: Modified Booth Algorithm

Radix-4 encoding reduces the total number of multiplier digits by a factor of two, which means in this case the number of multiplier digits will reduce from 16 to 8. Booth's recoding method does not propagate the carry into subsequent stages. This algorithm groups the original multiplier into groups of 3consecutive digits where the outer most digit in each group is shared with the outer most digit of the adjacent group. Each of these group of three binary digits then corresponds to one of the numbers from the set {+2,+1,0,-1,-2}. Each recoder produces a 3-bit output where the 1st bit represents the number 1 and the 2nd bit represent number 2. The 3rd and final bit indicates whether the number in the 1st or 2nd bit is negative.

4 Normalizer

The result of the significand multiplication (intermediate product) must be normalized to have a leading '1' just to the left of the decimal point (i.e., in the bit 46 in the intermediate product). Since the inputs are normalized numbers then the intermediate product has the leading one at bit 46 or 47.
1-If the leading one is at bit 46 (i.e., to the left of the decimal point) then the intermediate product is already a normalized number and no shift is needed.
2- If the leading one is at bit 47 then the intermediate product is shifted to the right and the exponent is incremented by 1.The shift operation is done using combinational shift logic made by multiplexers.

5 Underflow/overflow detection

Overflow/underflow means that the result's exponent is too large/small to be represented in the exponent field. The exponent of the result must be 8 bits in size, and must be between 1 and 254 otherwise the value is not a normalized one. An overflow may occur while adding the two exponents or during

normalization. Overflow due to exponent addition may be compensated during subtraction of the bias; resulting in a normal output value (normal operation). An underflow may occur while subtracting the bias to form the intermediate exponent. If the intermediate exponent < 0 then it's an underflow that can never be compensated; if the intermediate exponent= 0 then it's an underflow that may be compensated during normalization by adding 1 to it. When an overflow occurs an overflow flag signal goes high and the result turns to ±Infinity .When an underflow occurs an underflow flag signal goes high and the result turns to ±Zero.

6 Pipelining the multiplier

In order to enhance the performance of the multiplier, three pipelining stages are used to divide the critical path thus increasing the maximum operating frequency of the multiplier. The pipelining stages are imbedded at the following locations:

1.In the middle of the significand multiplier, and in the middle of the exponent adder (before the bias subtraction).

2.After the significand multiplier, and after the exponent adder.

3.At the floating point multiplier outputs (sign, exponent and mantissa bits).Figure 3 shows the pipelining stages as dotted lines.

Three pipelining stages mean that there is latency in the output by three clocks.

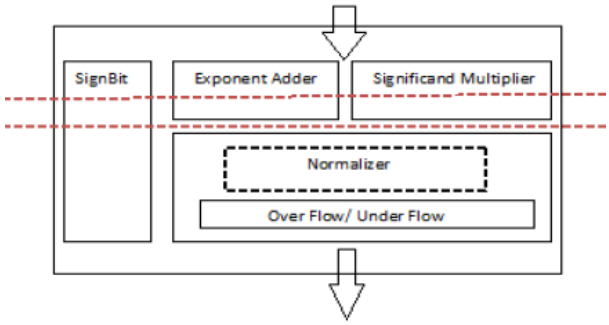

Fig3: Floating point multiplier with pipelined stages

V. COMPARISONS AND VHDL SIMULATION

First the VHDL simulation of two multipliers is considered. The VHDL code for both multipliers, using a fast carry look-ahead adder are generated. The multiplier uses 24-bit values. The worst case was applied to the two multipliers, where the gate delay is assumed to be 5ns. The array multiplier has a delay of 28.713 ns with a total of 23.58 mW power consumption. The FP multiplier using modified booth encoder multiplier has provided a delay of 26.990ns with 23.6 mW of power consumption. The whole multiplier was tested against the Xilinx floating point multiplier core generated by Xilinx coregen. Xilinx coregen was customized to have two flags to indicate overflow and underflow , and to have a maximum latency of three cycles. Xilinx core implements the "round to nearest " rounding mode. A test-bench is used to generate the stimulus and applies it to the implemented floating point multiplier and to the Xilinx core then compares the result .The floating point multiplier was also checked using precision synthesis tool targeted to XC3S1500-5FG456 Spartan-3 device.

VI. CONCLUSION AND FUTURE WORK

The Floating point unit is designed. The design is done in such a way that the floating point unit can be effectively interfaced with any processor that is either 32 bit or 64 bit. The architectures that are used in the design are selected based on careful analysis considering the three parameters, speed, area and power consumption. The FP unit may extended to quadruple precision format for more advanced and scientific computations. As an attempt to develop, IEEE 754 compatible floating point unit algorithm and architecture level optimization techniques for low power high-speed multiplier design, techniques presented in this article. There are several future research directions are possible as follows:

One possible direction is radix higher-than-4 recoding. Only radix-4 recoding is considered in this article as it is a simple and popular choice. Higher-radix recoding further reduces the number of PPs and thus has the potential of power saving. In order to enhance the performance, higher order compressors like 7:2, 9:2 can be used to accumulate the partial products. Deep level pipeline architecture can be used for speed improvements. Multiplication intensive applications, such as DSP or graphics, could benefit significantly from several high performance multipliers on the same chip. A single very high throughput multiplier, or several multipliers working in parallel on the same chip, could open up new possibilities such as single chip video signal processors.

REFERENCES

[1] Amine Bermark, Guixuan Liang and Qingzheng, "A High-speed 32-bit Signed/ Unsigned Pipelined Multiplier", Department of Electronics and Computer Engineering, Honkong University of Science and Technology, Hong Kong, China.

[2] Cang-YuanGuo, Jiun-Ping Wang and Shiann-Rong Kuang, Member IEEE "Modified Booth Multipliers with Regular Partial Product Array", IEEE Transactions on Circuits and Systems (2009).

[3] Carl Hamacher, Zvonko Vranesic and Safwat Zaky, "Computer Organization", Fifth edition.

[4] Chris Babb, Jeff Blank, Ivan Castellanos and John Moskal, "Floating Point Multiplier", ECE 587.

[5] Michael Doan and Mounir Bohsali, "Rectangular Styled Wallace Tree Multiplier".
[6] Pouya Asadi, and Keivan Navi, "A new low power 32×32- bit multiplier", World Applied Sciences Journal 2 (4): 341-347, 2007
[7] Purushottam D. Chidgupkar, and Mangesh T. Karad, "The implementation of algorithms in digital signal processing", Global J. of Engineering Education, vol.8, no.2 © 2004 UICEE Published in Australia.
[8] Michael Andrew Lia, "Arithmetic units for high performance processors", Thesis for degree of Master of Science, University of California, 2002.
[9] Soojin Kim, and Kyeongsoon Cho, "Design of high-speed modified Booth multipliers operating at GHz ranges", World Academy of Science, Engineering and Technology, 2010.
[10] Rabey, Nikolic, and Chandrasekhran, "Digital Integrated Circuits: A Design Perspective", 2nd Edition, Prentice Hall, pp. 586-594, 2003.
[11] Roy, Kaushik, Yeo, and Kiat-Seng, "Low voltage Low-power VLSI Subsystems", McGraw-Hill, pp.124-141.
[12] S S Manvi,L Rajesh /V Joshi and Prashant. "An FPGA based Implementation of Floating-point Multiplier "International Conference on Computing and Control Engineering (ICCCE 2012), 12 & 13 April, 2012 .

Biographies

SUBITHA M B received the B.TECH. degree in Electronics & Communication Engineering from the CUSAT University Cochin Kerala, in 2009, and the M.E. degree in Electronics & Communication Engineering from the ANNA University Chennai , Tamil Nadu, in 2013.Currently working as an assistant Professor of Electronics & Communication Engineering at Kannur University Kerala. Teaching and research areas include floating point unit, control systems, and embedded system design.