RESEARCH ARTICLE OPEN ACCESS

Efficient Hybrid Method for Binary Floating Point Multiplication

S. Praveenkumar Reddy, S. Parvathi Nair
Dept. of Electronics and Communication Engineering SRM University Chennai, India
Dept. of Electronics and Communication Engineering SRM University Chennai, India

Abstract
This paper presents a high speed binary floating point multiplier based on Hybrid Method. To improve speed multiplication of mantissa is done using Hybrid method replacing existing multipliers like Carry Save Multiplier, Dadda Multiplier and Modified Booth Multiplier. Hybrid method is a combination of Dadda Multiplier and Modified Radix-8 Booth Multiplier. The design achieves high speed with maximum frequency of 555 MHz compared to existing floating point multipliers. The multiplier implemented in Verilog HDL and analyzed in Quartus II 10.0 version. Hybrid Multiplier is compared with existing multipliers.
Keywords— Hybrid method, Dadda Multiplier, Booth Multiplier, Floating point multiplication, Verilog HDL;

I. INTRODUCTION

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 Fig.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).

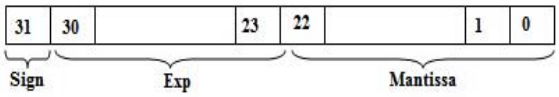

Fig. 1. IEEE single precision floating point format

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

Where , $M = n_{22}\, 2^{-1} + n_{21}\, 2^{-2} + n_{20}\, 2^{-3} + \ldots + n_1\, 2^{-22} + n_0\, 2^{-23}$, Bias = 127.

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

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

Step b. Significands multiplication of the two numbers using Dadda algorithm.

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

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

II. FLOATING POINT MULTIPLIER ALGORITHM

The normalized floating point numbers have the form of $Z= (-1^S) * 2^{(E -Bias)} * (1.M)$. The following algorithm is used to multiply two floating point numbers.

1. Significand multiplication i.e. $(1.M_1 * 1.M_2)$.
2. Placing the decimal point in the result.
3. Exponent's addition i.e. $(E_1 + E_2 - Bias)$.
4. Getting the sign i.e. s_1 xor s_2.
5. Normalizing the result i.e. obtaining 1 at the MSB of the significand.
6. Rounding implementation.

Consider the following IEEE 754 single precision floating point numbers to perform the multiplication, but the number of mantissa bits is reduced for simplification. Here only 5 bits are considered while still considering one bit for normalized numbers.

A = 0 10000001 01100 = 5.5, B = 1 10000100 00011 = -35

By following the algorithm the multiplication of A and B is

1. Significand Multiplication: 1.01100

```
          ×1.00011
           101100
          101100
         000000
        000000
       000000
      101100
  011000000100
```

2. Normalizing the result: 1.1000000100

3. Adding two exponents: 10000001
+10000100
100000101

The result after adding two exponents is not true exponent and is obtained by subtracting bias value i.e 127. The same is shown in following equations.

$E_1 = E_{1\text{-true}} + \text{bias}$

$E_2 = E_{2\text{-true}} + \text{bias}$

$E_1 + E_2 = E_{1\text{-true}} + E_{2\text{-true}} + 2 \times \text{bias}$

Therefore

$E_{true} = E_1 + E_2 - \text{bias}.$

From the above analysis bias is added twice. Hence bias has to be subtracted once from the result.

100000101
-001111111
10000110

4. Sign bit of result is extracted by doing XOR operation of
sign bit of multiplier and multiplicand:

1 10000110 01.1000000100

5. Then normalize the result so that there is a 1 just before the radix point (decimal point). Moving the radix point one place
to the left increments the exponent by 1; moving one place to the right decrement the exponent by 1.

6. If the mantissa bits are more than 5 bits (mantissa available bits), rounding is needed. If we applied the truncation rounding mode then the stored value is:

1 10000110 10000.

In this paper, we are presenting a floating point multiplier in which rounding support is not implemented. By this, more precision can be attained in MAC unit and this will be accessed by the multiplier or by a floating point adder unit. Figure 2 shows the block diagram of the multiplier structure: Exponents calculator, Mantissa multiplier and sign bit calculator, using the pipelining concept.

Two 24 bit significands are multiplied and the result is a 48 bit product, denoting this as Intermediate Result (IR). The IR width is 48-bit i.e. 47 down to 0 and the decimal point is located between bits 46 and 45 in the IR. Each block is elaborated in the following sections. In [3], the design of an efficient implementation of single precision floating point multiplier was done.

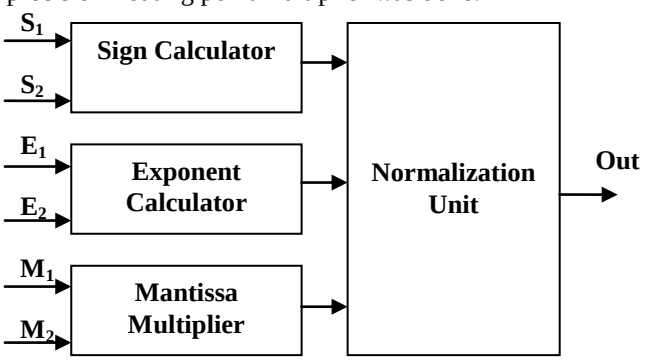

Fig. 2. Floating Point Multiplier Block Diagram

III. BLOCKS OF FLOATING POINT MULTIPLIER

A. Sign Calculator

The main component of Sign calculator is XOR gate. XOR operation is performed between the Sign bits of input binary floating point numbers. If any one of the numbers is negative then result will be negative. The result will be positive if two numbers are having same sign.

B. Exponent Adder

This sub-block adds the exponents of the two floating point numbers and the Bias (127) is subtracted from the result to get true result. For Single precision Floating point Multiplication addition is done on two 8 bit exponents.

$$\text{Exponent (E)} = E_1 + E_2 - \text{bias}$$

C. Significand multiplication using Unsigned Multiplier

Significand will be formed by adding an extra one bit to the Fraction (or) Mantissa part. Significand multiplication is done using Unsigned Multipliers.

i. Significand Multiplication using Carry Save Multiplier

This unit is used to multiply the two unsigned significand numbers and it places the decimal point in the multiplied product. The unsigned significand multiplication is done on 24 bit. The result of this significand multiplication will be called the IR. Multiplication is to be carried out so as not to affect the whole multiplier's performance. In this carry save multiplier architecture is used for 24X24 bit as it has a moderate speed with a simple architecture. In the carry save multiplier [2], the carry bits are passed diagonally downwards i.e. the carry bit is propagated to the next stage. It is a parallel

multiplier for unsigned operands. It is composed of 2-input AND gates for producing the partial products, a series of Carry save adders for adding them and a Ripple-carry adder for producing the final result (vector merging stage). Carry save multipliers consists of Full adders and Half adders.

The count of adders (Half adders and Full adders) in each stage is one less than the significand size. For example, an 8x8 carry save multiplier is shown in Figure 3 and it has the following stages:

1. The first stage consists of six full adders and a half adder.
2. Middle stages; each consists of six full adders and a half adder.
3. The vector merging stage consists of one half adder and six full adders.

The decimal point is placed between bits 45 and 46in the significand multiplier result.

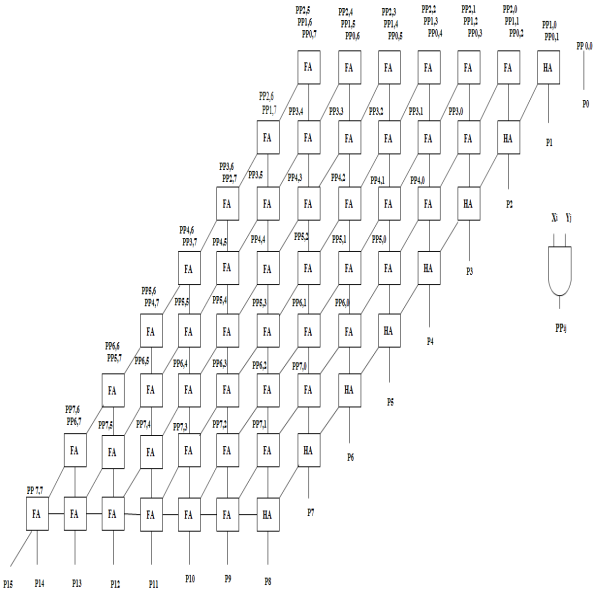

Fig. 3. 8x8 bit Carry Save Multiplier

In Figure 3

HA: Hall Adder.
FA: Full Adder.

ii. Significand Multiplication Using Dadda Multiplier

The Dadda multiplier is a hardware multiplier design, invented by computer scientist Luigi Dadda in 1965. It is slightly faster (for all operand sizes) and requires fewer gates (for all but the smallest operand sizes). Dadda proposed a sequence of matrix heights that are predetermined to give the minimum number of reduction stages. To reduce the N by N partial product matrix, dada multiplier develops a sequence of matrix heights that are found by working back from the final two-row matrix. In order to realize the minimum number of reduction stages, the height of each intermediate matrix is limited to the least integer that is no more than 3/2 times the height of its successor.

The process of reduction for a Dadda multiplier [5] is developed using the following recursive algorithm.

1. Let d1 = 2 and dj+1 = [3.dj / 2], where dj is the matrix height for the j-th stage from the end. Find the largest j such that at least one column of the matrix has more than dj bits.
2. Employ (3, 2) and (2, 2) counters to obtain a reduced matrix with no more than d_j elements in any column.
3. Let j = j-1 and repeat step 2 until a matrix with only two rows is generated.

This method of reduction, because it attempts to compress each column, is called a column compression technique. Another advantage of utilizing Dadda multipliers is that it utilizes the minimum number of (3, 2) counters. Therefore, the number of intermediate stages is set in terms of lower bounds: 2, 3, 4, 6, 9 . . .

For Dadda multipliers there are N^2 bits in the original partial product matrix and 4.N-3 bits in the final two row matrix. Since each (3, 2) counter takes three inputs and produces two outputs, the number of bits in the matrix is reduced by one with each applied (3, 2) counter therefore, the total number of (3,2) counters is $N^2 – 4.N+3$, the length of the carry propagation adder is CPA length = 2.N–2.

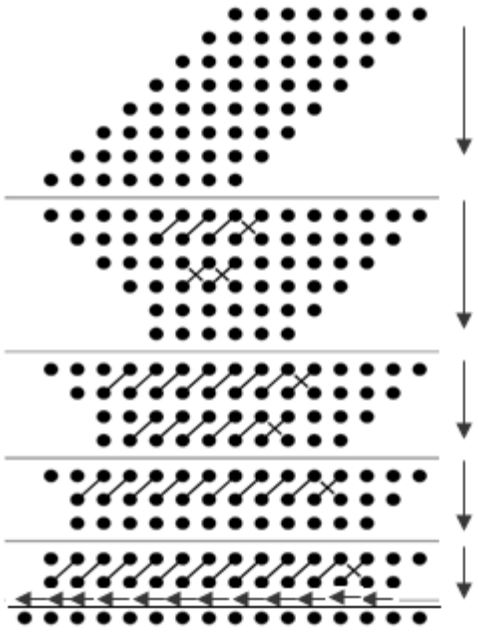

Fig. 4. Dot diagram for 8 by 8 Dadda Multiplier

The number of (2, 2) counters used in Dadda's reduction method equals N-1.The calculation diagram for an 8X8 Dadda multiplier is

shown in figure 4. Dot diagrams are useful tool for predicting the placement of (3, 2) and (2, 2) counter in parallel multipliers. Each IR bit is represented by a dot.The output of each (3, 2) and (2, 2) counter are represented as two dots connected by a plain diagonal line. The outputs of each (2, 2) counter are represented as two dots connected by a crossed diagonal line.

The 8 by 8 multiplier takes 4 reduction stages, with matrix height 6, 4, 3 and 2. The reduction uses 35 (3, 2) counters, 7 (2, 2) counters, reduction uses 35 (3, 2) counters, 7 (2, 2) counters, and a 14-bit carry propagate adder. The total delay for the generation of the final product is the sum of one AND gate delay, one (3, 2) counter delay for each of the four reduction stages, and the delay through the final 14-bit carry propagate adder arrive later, which effectively reduces the worst case delay of carry propagate adder. The decimal point is between bits 45 and 46 in the significand IR.

The below Table 1 shows the number of reduction stages required to implement Dadda architecture for various number of bits.

Table 1: Number of Reduction Stages For Dadda Multiplier

iii. Significand Multiplication using Modified Radix-8 Booth Multiplier

Bits in Multiplier(N)	Number of Stages
3	1
4	2
$5 \leq N \leq 6$	3
$7 \leq N \leq 9$	4
$10 \leq N \leq 13$	5
$14 \leq N \leq 19$	6
$20 \leq N \leq 28$	7
$29 \leq N \leq 42$	8
$43 \leq N \leq 63$	9
$63 \leq N \leq 94$	10

Modified Booth [4] is twice as fast as Booth algorithm. Modified Booth encoding algorithm is an efficient way to reduce the number of partial products by grouping consecutive bits in one of the two operands to form the signed multiples. The operand that is Booth encoded is called the multiplier and the other operand is called the multiplicand.

In the radix-8 booth multiplier we consider group of 4 bits.. Each group is coded as a signed-digit using the Table 2. Number of Partial products in radix-8 is N/3+1

Radix-8 Booth algorithm scans strings of 4 bits with the algorithm given below:

1. Extending the sign bit position if require, to ensure that n is even only.
2. Append a 0 to the right side of the least significant bit of the multiplier.
3. According to the value of each vector, Partial Product will be 0, +Y, –Y, +2Y, –2Y, +3Y, –3Y, 4Y, –4Y. The negative values of y are considered by taking the 2‘s complement to the Booth recode the multiplier term, we have to consider the bits in groups of four, in a way that each group overlaps with the previous group by one bit. Grouping starts from the LSB.

Table 2: Encoding Of Modified Radix-8 Booth Multiplier

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

Here we have an odd multiple of the multiplicand, 3Y, which is not immediately available. To generate it we need to perform this previous add: 2Y+Y=3Y. For finding 2Y multiplicand is left shifted once and Y is the multiplicand itself.

Let us take an example:
Multiplicand is (00101010)
Multiplier is (01010100)

Now we will consider the group of four bits for Multiplier. Now according to the Table 2 we get to know that

(0010) – (+1)
(0101) – (+3)
(1000) – (–4)

Thus Multiplicand is multiplied with the three encoded digits which are 1, 3 and –4.

(i) –4 * (00101010) = 01011000

And now 11111111 is added with the result because of the negative sign. So final result of multiplication of -4 is 1111111101011000, here negative term sign is extended.

(ii) 3* (00101010) = 01111110

Here 00000 is added with the result because of the positive sign. So final result is 0000001111110. Now the result is added to previous result with 3 bits shifted left.

(iii) 1* (00101010) = 00101010

Here 00000 is added with the result because of the positive sign. So final result is 0000101010. Now the result is added to previous result with 3 bits shifted left. Final result after adding all the partial products is 0000110111001000, here we have discarded the carried high bit.

By using this Modified Radix-8 Booth multiplier we will perform 24 bit significand multiplication.

iv. Proposed Multiplier

Hybrid Multiplier:

Hybrid Multiplier [6] is a combination of Modified Radix-8 Booth Multiplier and Dadda algorithm. By using Modified Radix-8 Booth Multiplier we will generate partial products and that partial products are reduced by using Dadda algorithm.

Modified Radix-8 Booth Multiplier generates N/3+1 partial products. These partial products are reduced by using Dadda algorithm. The number of stages in Dadda algorithm is shown in Table 1 in accordance to number of partial products.

From example, as explained in Modified radix-8 booth multiplier

Multiplicand is (00101010) -- 42
Multiplier is (01010100) -- 84

```
    00101010
  X 01010100
1111111101011000
0000001111110
0000101010
0000001101001000
0000101010
0000110111001000
```

By using this hybrid multiplier we will perform 24 bit significand multiplication. Here stages are less compared to previous multipliers and therefore it has high speed.

D. Normalizing Unit:

The result of the significand multiplication (intermediate product) must be normalizing. Having a leading '1' just immediate to the left of the decimal point (i.e. in the bit 46 in the intermediate product) is known as a normalized number. Since the inputs are normalized numbers then the intermediate product has the leading one at bit 46 or 47.

1. No shift is needed the intermediate product is known to be a normalized number when the one is at bit 46 (i.e. to the left of the decimal point) .
2. The exponent is incremented by 1 if the leading one is at bit 47 then the intermediate product is shifted to the right.

IV. MULTIPLIER PIPELINING

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 embedded at the following locations:

1. Before the bias subtraction; in the middle of the significand multiplier and in the middle of the exponent adder.
2. After the significand multiplier and the exponent adder.
3. Sign, exponent and mantissa bits; at the floating point multiplier outputs.

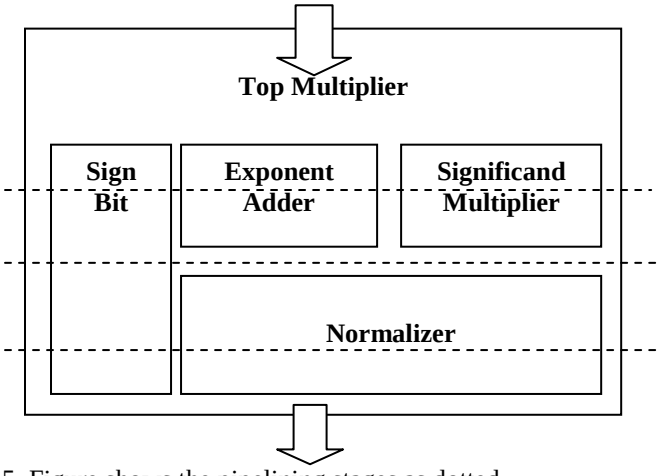

Fig. 5. Figure shows the pipelining stages as dotted lines.

V. SIMULATION RESULTS

The simulation results for corresponding inputs are shown in Fig. 6. The simulation is done using Altera Modelsim 6.5e. Considering the random floating point numbers,

Inputs: a = 32'H41F40000; →(30.5)
b = 32'HC2220000; →(- 40.5)
Output: result = 32'HC49A6800;

Here the input 'a' is positive, input 'b' is negative, so the output result will be negative whose sign bit is '1'. Output is obtained in latency of three clock cycles.

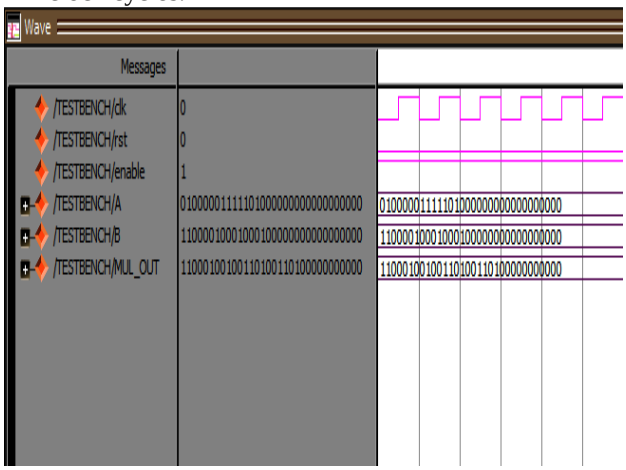

Fig. 6. Floating point multiplier Simulation

The synthesis results of Floating point multiplier using different multipliers are calculated using Quartus II 10.0 [7] is shown in following Table 3.

Table 3: Area, Power and Frequency comparision between different Floating Point Multipliers.

	Floating point multiplier using Hybrid method	Floating point multiplier using radix-8 booth multiplier	Floating point multiplier using Dadda multiplier	Floating point multiplier using Carry save multiplier
Logic Elements	2021	1851	1432	1543
Power (mW)	179.85	178.99	131.24	192.46
Freq (MHz)	555.15	542.79	526.86	483.09

VI. CONCLUSION AND FUTUREWORK

This paper describes an implementation of a floating point multiplier using Dadda Multiplier that supports the IEEE 754-2008 binary interchange format; the multiplier is more precise because it doesn't implement rounding and just presents the significand multiplication result as is (48 bits). The significand multiplication time is reduced by using Hybrid method. Speed achieved using hybrid method is 555 MHz. Double precision Floating Point Multiplication can also be done using same method.

REFERENCES

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.
[2] Jeevan. B, Narender. S, Reddy. C. V. K & Sivani K. "A high speed binary floating point multiplier using Dadda algorithm" IEEE Conference on computing, 2013 IEEE.
[3] Mohamed Al-Ashrfy, Ashraf Salem and Wagdy Anis "An Efficient implementation of Floating Point Multiplier" IEEE Transaction on VLSI, 2011 IEEE, Mentor Graphics.
[4] Deepali Chandel, Gagan Kumawat, Pranay Lahoty, Vidhi Vart Chandrodaya, Shailendra Sharma, "Modified Booth Multiplier: Ease of multiplication", International Journal of Emerging Technology and Advanced Engineering, Volume 3, Issue 3, March 2013.
[5] Whytney J. Townsend, Earl E. Swartz, "A Comparison of Dadda and Wallace multiplier delays". Computer science Engineering Research Center, The University of Texas.
[6] Brian Millar, Philip E. Madrid, Earl E. Swartzlander, Jr. "A fast hybrid multiplier combining of booth and wallace/dadda algorithms"
[7] Introduction to Altera Quartus II Software 10.0 Version.