ISSN: 2278 – 909X
International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE)
Volume 4, Issue 9, September 2015

A Speed and Accurate binary multiplier for floating point values

Mudireddy Bala Subba Reddy [1]

T.Suneel Kumar [2]

Abstract:

This paper exhibits a speed and accurate binary multiplier for floating point values in view of Dadda Algorithm. Multiplication of mantissa is carried out utilizing Dadda multiplier supplanting Carry Save Multiplier to enhance the speed of operation. The configuration attains to high speed with maximum recurrence of 526 MHz contrasted with existing floating point multipliers. The floating point multiplier is developed to handle the underflow and overflow occurrences. Adjusting is not brought to give more accuracy for mantissa multiplication. The multiplier is implemented utilizing Verilog HDL furthermore it is focused for Xilinx Virtex-5 FPGA. The multiplier is contrasted and Xilinx floating point multiplier core.

I. Introduction:

The vast majority of the DSP applications need multiplication of numbers of floating point type. The conceivable approaches to stand for real numbers as binary arrangement floating point numbers are; Binary interchange and Decimal interchange format according to the IEEE 754 standard [1]. Single precision normalized binary interchange arrangement is actualized in this outline. Such arrangement is indicated in Figure 1; beginning from MSB it has an one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction called Mantissa (M). Including an additional bit to the fraction to structure and is characterized as significand. If exponent is more noteworthy than 0 and littler than 255, and there is 1 in the MSB of the significand then the number is said to be a normalized number; for this present case the real number is represented by equation (1).

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

Fig 1: Representation of IEEE single precision floating point format

II. Floating Point Multiplier Algorithm

Generally there are four stages to carry out the floating point multiplication of two numbers:

Step 1: Addition of two number's exponents and from this result subtracting the extra bias.

Step 2: Multiplication of the significands of the two numbers by the Dadda algorithm.

Step 3: Performing the XOR operation between sign bits of two numbers to identify the sign of the result.

Step 4: At last, the outcome is normalized such a manner that there must be 1 in the MSB of the outcome.

Most intriguing zone of numerous specialists is to actualize Floating-point multipliers on FPGAs. In [6], the configuration of an effective implementation of single precision floating point multiplier was examined against virtex-5 FPGA in which IEEE 754 single precision pipelined floating point multiplier was executed on multiple FPGAs (4 Actel A1280). In [8], an alternate custom 16/18 bit three stage pipelined floating point multiplier was implemented that has no adjusting modes. In [3], a solitary precision floating point multiplier that doesn't bolster adjusting modes was executed utilizing a digit-serial multiplier: utilizing the Altera FLEX 8000 it accomplished 2.3 MFlops. In [5], a parameterizable floating point multiplier was actualized utilizing the Handel-C, utilizing the Xilinx XCV1000 FPGA; a five stages pipelined multiplier accomplished 28MFlops. In [4], an idleness improved floating point unit utilizing the primitives of Xilinx Virtex II FPGA was implemented with an inactivity of 4 clock cycles. The multiplier arrived at a greatest clock recurrence of 100MHz.

The general algorithm for multiplication of two floating point values is as follows and is illustrated along with an example here itself considering two numbers.

Let x=4.75 =0 10000001 0011000000 , and y= -21.101 = 1 10000011 0101000110

1. Multiplication of significands and normalizing the result:

1.0011000000×1.0101000110= 1.10010000110010000000

2. Addition of exponents of two numbers to get exponent of the result:

10000001 + 10000011 = 100000100

The outcome in the wake of summing two exponents is not true exponent and is acquired by subtracting one bias value i.e 127 because the bias is added twice while the addition of that two exponents.

100000100 – 001111111 = 100001010

3. The sign of the result can be obtained by performing XOR operation between sign bits of two numbers.

0 xor 1 = 1 (gives negative sign for the result)

4. Now, normalizing the outcome so that there is a 1 fair before the radix point (decimal point).

1
10000101.10010000110010000000

5. On the off chance that the mantissa bits are more than 10 bits (mantissa accessible bits); rounding is required. On the off chance that we brought the truncation adjusting mode then the put away esteem is

1 10000101 1001000011

Rounding support is not actualized in this floating point multiplier which we are presenting. By this we get more exactness in MAC unit and this will be gotten to by the multiplier. Figure 2 shows the brief diagrammatic representation of the multiplier; Exponent calculator, Mantissa multiplier and sign bit calculator, utilizing the pipelining idea here all methods are completed in parallel.

III. Important Blocks of Floating Point Multiplier

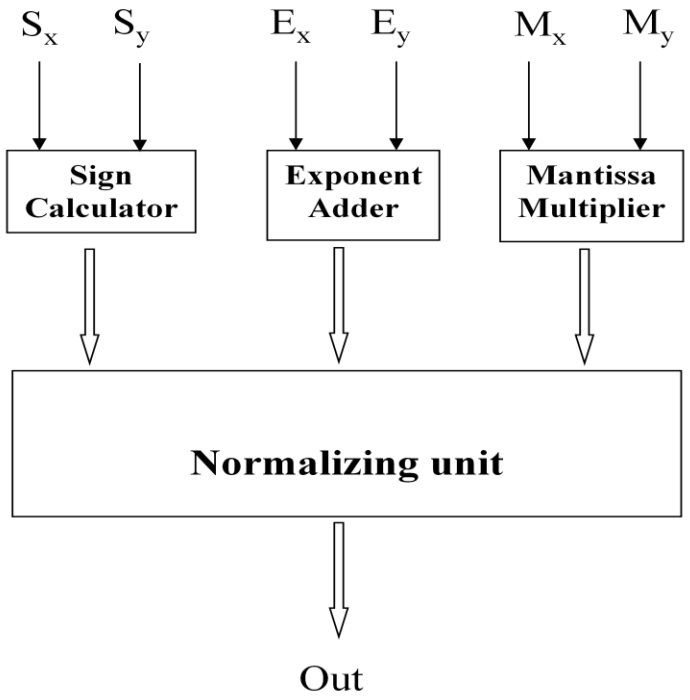

Fig 2: Block schematic of floating point multiplier

A. Sign calculator :

The principle segment of Sign calculation is XOR gate. On the off chance that any one of the numbers is negative then result will be negative. The result will be sure if two numbers are of same sign.

B. Exponent Adder :

The two floating point number's exponents are added and bias is subtracted in this block i.e. Ex + Ey – bias.

In this outline the above is carried out on two 8 bit exponents. In the designs the vast majority of the computational time is spending in the significand multiplication procedure (multiplying 24 bits by 24 bits); so brisk aftereffect of the addition is a not indispensable. Consequently we require a quick significand multiplier and a moderate exponent adder.

An 8-bit ripple carry adder is used for adding 8-exponents and a ripple borrow subtractor is used to subtract the bias from result of addition of two exponents.

Multiplication of two unsigned signifcands numbers and putting the decimal point in the result is done by the multiplier. The unsigned significand multiplication is carried out on 24 bit. This result is so called intermediate result. Multiplication is to be completed so as not to influence the entire multiplier's prophecy.

The carry bits are propagated slantingly downwards to the succeeding stages. The partial products are produced by ANDing the inputs of two numbers what's more passing them to their allocated adder.

Carry save multiplier has three primary stages: first stage is an array of half adders, middle stage is the arrays of full adders, in which the number of arrays of full adders are equal to two less in the size of significand and final stage also called vector merging stage consists of an array of ripple carry adders.

C. Significand Multiplication:

i. Existing Multiplier (Carry save multiplier):

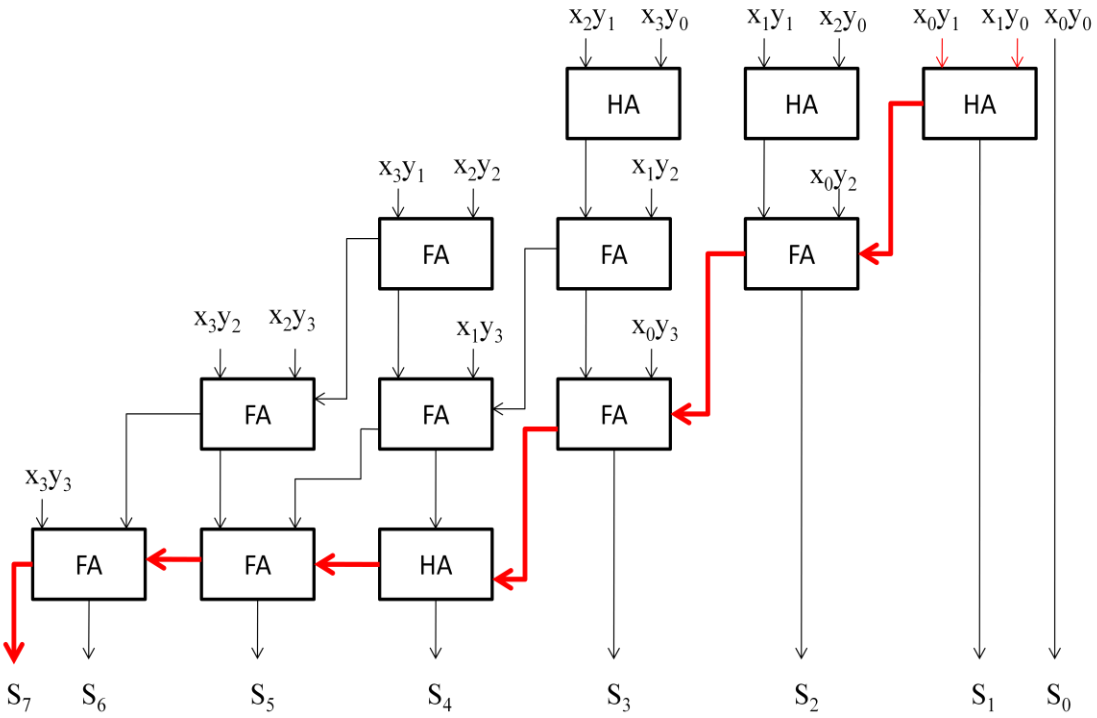

Fig 3: Carry Save Multiplier of 4 × 4 for example

The time taken to perform multiplication by the carry save multiplier is resolved by its critical path. The critical path begins at the AND gate of the first partial products (i.e. x_1y_0 and x_0y_0), goes through the carry logic of the first half adder and the carry logic of the first full adder of the middle stages, then goes through all the vector consolidating adders. Nevertheless the critical path is indicated with a bold line in the figure above.

ii. Proposed multiplier :

Dadda Multiplier:

Dadda projected an arrangement of matrix heights that are foreordained to give the minimum number of reduction stages. To reduce the N × N partial product matrix, Dadda multiplier build up an arrangement of matrix heights that are found by meeting expectations back from the final two-row matrix.

The height of every intermediate matrix is restricted to the least integer that is not greater than 1.5 times of its successor height to realize the number of reduction stages.

The process of reduction as cited in the following recursive algorithm.

1. Let $d_1=2$ and $d_j + 1 = [1.5 \times d_j]$, where d_j is the height of the matrix for the j^{th} stage from the end. Find the smallest j such that no less than one column of the original partial product matrix consists more than dj bits.

2. In the j^{th} stage from the end, utilize (3, 2) and (2, 2) counter to get a decreased matrix with close to d_j bits in any column.

3. Let j = j-1 and rehash step 2 until a matrix with just two rows is produced.

This technique for reduction, in light of the fact that it endeavors to constrict every column, is known as a column compression technique. One more pro of using Dadda multiplier is that it uses the minimum number of (3, 2) counters. {Therefore, the quantity of intermediate stages is situated regarding lower limits: 2, 3, 4, 6, 9 . . . there are N2 bits in the original partial product matrix and 4.N-3 bits in the last two row matrix of the Dadda multiplier. Since every (3, 2) counter takes three inputs and generates two yields, the quantity of bits in the matrix is decreased by one with every connected (3, 2) counter therefore}, the aggregate number of (3,2) counters is #(3, 2) = N2 – 4.N+3. The length of the CPA (carry propagation adder) = 2.N-2.

The number of (2, 2) counters utilized as a part of Dadda's reduction technique measures up to N-1.The calculation diagram for a 8X8 Dadda multiplier is demonstrated in figure 4.

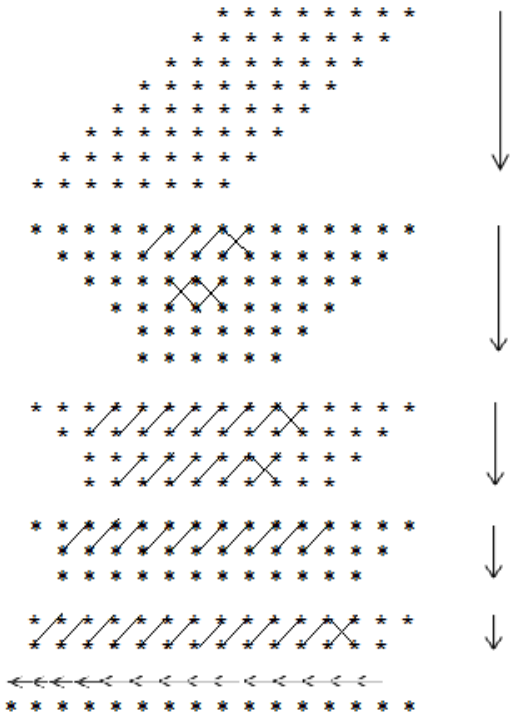

Fig 4: Star Diagram representing recursive algorithm for an 8 × 8 Dadda multiplier (In the figure pentagonal and hexagonal stars are used just to differ one stage from another. The vertical arrows indicates reduction process of stages and the horizontal arrows indicates carry bit propagation from MSB to LSB while addition)

This type of diagrams are useful for anticipating the position of (3, 2) and (2, 2) counter in parallel multipliers. Every star indicates an IR bit.

The yield of every (3, 2) and (2, 2) counter are indicated to as two stars associated by a plain inclining line. The yields of every (2, 2) counter are indicated to as two stars joined by a crossed askew line.

The 8 x 8 multiplier takes 4 reduction stages, with matrix heights 6, 4, 3 and 2. Here in this reduction it utilizes 35 (3,2) counters, 7 (2,2) counters, and a 14-bit carry propagate adder.

The sum of an AND gate delay, a (3,2) counter delay for each of four reduction stages and a 14-bit carry propagate adder delay is the total delay to the begot of final product value. This 14-bit carry propagate adder that adequately lessens the most detrimental possibility deferral of carry propagate adder. In the significand IR the decimal point is between 45 and 46 bits.

Critical path is utilized to resolve the time taken by the Dadda multiplier. The critical path begins at the AND gate of the first partial product goes through the full adder of the each stage, then goes through all the vector merging stages. These stages are less in this multiplier when compared with the carry save multiplier and in this way it has faster than that.

D. Normalizing Unit:

It must be normalize the intermediate product i.e. the result of significand multiplication. Having a main "1" recently quick to left side of the decimal point i.e. in the bit 46 in the intermediate result is known as a normalized number. Then the intermediate product has the leading one at bit 46 or 47 since the inputs are normalized.

1. No shift is required the intermediate product is known to be a normalized number when the one is at bit 46 (i.e. to the left side of the decimal point).

2. If the leading one is at bit 47 then the intermediate result is shifted to the right, the exponent is incremented by 1.

Here to perform the combinational shift logic the multiplexers are utilized for the shift operation.

IV. Underflow and Overflow Prophecy

Underflow/Overflow implies that the outcome's exponent is as well little/vast to be embodied in the field of exponent. The outcome of exponent must be 8 bits in size, and must be between 1 and 254, if not the worth is not a normalized one. While adding the two exponents or in the time of normalization overflow may happen.

Overflow because of addition of exponents may be repaid amid subtraction of the bias; coming about underflow that can never be compensated; if the intermediate exponent = 0 then amid normalization it's underflow that may be repaid by adding 1 to it.

An overflow signal is made high in the event that an overflow happens and the outcome turns to ±Infinity (indication of the floating point multiplier inputs decides this sign). On the off chance an underflow sign goes high in the event that an underflow happens and the outcome turns to ±Zero (this sign also decided by the indication of the floating point multiplier inputs).

After denormalized numbers are made to zero, An underflow flag is raised with the proper sign figured from the inputs. Let E_x and E_y are the exponents of the two numbers x and y separately; the outcome's exponent is ascertained by (2).

$$Ex + Ey - 127 \qquad (2)$$

E_x and E_y can have the values from 1 to 254; subsequent in E_{result} having values from -125 (2-127) to 381 (508-127); however for the normalized numbers, E_{result} can only have the values from 1 to 254. Table 1 abridges the E_{result} diverse qualities and the impact of normalization on it.

Table 1: Underflow and Overflow revelation and effect of normalization on exponent's result

E_{result}	Category	Remarks
$-125 \leq E_{result} \leq 0$	Underflow	Compensation is not possible
$E_{result} = 0$	Zero	During normalization it may turn to normalized number
$1 < E_{result} < 254$	Normalized number	During normalization it may tends to overflow
$255 \leq E_{result}$	Overflow	Compensation is not possible

V. Multiplier Pipelining

With a specific end goal to improve the performance of the multiplier, three pipelining stages are utilized to divide the critical path therefore increasing the maximum operating frequency of the multiplier.

The pipelining stages are imbedded at the accompanying areas:

1. Before the bias subtraction; amidst the significand multiplier and amidst the exponent adder.

2. After the significand multiplier and the exponent adder.

3. Sign, exponents and mantissa bits; at the floating point multiplier yields.

It was used the synthesis tool retiming so that to better place the pipelining registers across the critical path the synthesizer uses its optimization logic.

VI. Realization and Testing

The entire multiplier (top unit) was tried against the Xilinx floating point multiplier core. Xilinx core was modified to have two flags to show overflow and underflow, and to have a most extreme inertness of three cycles. Xilinx core employs the "round to nearest" adjusting mode.

A stimulus project is composed and is applied it to the actualized floating point multiplier and to the Xilinx core at that results are analyzed. The floating point multiplier code was additionally checked utilizing Xilinx13.4 [7]. The outline was synthesized utilizing Xilinx synthesis XST [7] and it is focused on Virtex-5 FPGA (xc5vlx20t-2ff323). Table 2 demonstrates the change in the actualized floating point multiplier contrasted with Xilinx core.

Simulation Results

Considering any two random floating point numbers

Inputs:

x = 4.75= 01000001110011000000000000000000;

y = -21.101= 10111110110000000000000000000000

then the result obtained is

Output :

Out= -100.22975

= 11000001000110010000000000000000

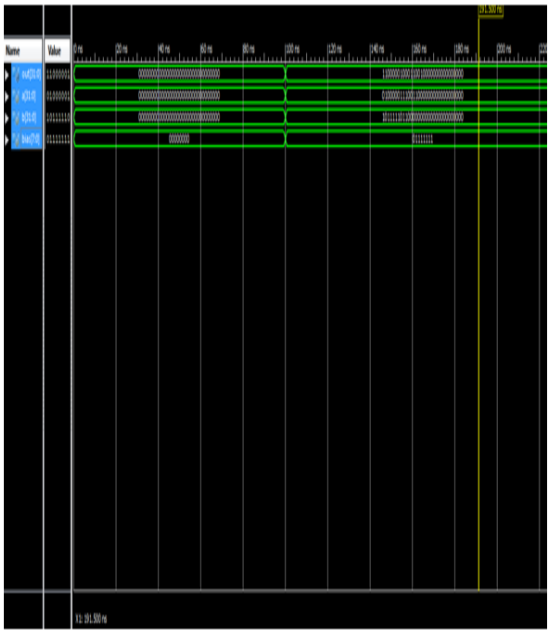

Figure 5: Floating point multiplier simulation results

Table 2: A comparison chronology between proposed and existing multiplier

	Proposed system	**Existing System**	**Xilinx Core**
Look-up Tables and Flip-flop pairs	1146	1110	235
Configurable Logic Blocks	1149	1112	732
Maximum frequency in MHz	526.86	401.71	206.30

VII. Conclusion and Future scope

This paper depicts an accomplishment of a floating point multiplier utilizing Dadda multiplier that backings the IEEE 754- 2008 binary interchange format; the multiplier is more accurate since it doesn't put into practice rounding and simply introduces the significand multiplication consequence as may be (48 bits). The significand multiplication time is lessened by utilizing Dadda Algorithm. The outline has been actualized on a Xilinx Virtex5 FPGA and accomplished the rate of 526MHz.

VIII. References

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008.

[2] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95), pp.155-162, 1995.

[3] Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, 2001, vol. 2, pp.897-900.

[4] B. Lee and N. Burgess, "Parameterisable Floating-point Operations on FPGA," Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems, and Computers, 2002.

[5] Whytney J. Townsend, Earl E. Swartz, "A Comparison of Dadda and Wallace multiplier delays". Computer Engineering Research Center, The University of Texas.

[6] Mohamed Al-Ashrfy, Ashraf Salem and Wagdy Anis "An Efficient implementation of Floating Point Multiplier" IEEE Transaction on VLSI 978-1-4577-0069-9/11@2011 IEEE, Mentor Graphics.

[7] Xilinx13.4, Synthesis and Simulation Design Guide", UG626 (v13.4) January 19, 2012.

[8] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," Proceedings of 83 the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'96), pp. 107-116, 1996.

[9] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365- 367, 1994.

[1] **M.Bala Subba Reddy,** P.G.scholor of Electronics and communications in VLSI Design Specification, PBR Viswodaya Institute of Technology and Science, Kavali,Nellore, India. He had completed his bachelor degree in Electronics and Communication Engineering from Sri Sai Institute of Technology and Science, Rayachoti, Kadapa, India.

[2] **T.Suneel Kumar,** M.Tech, Assistant Professor in the Dept. of ECE, PBR Viswodaya Institute of Technology and Science, Kavali,Nellore, India. He has completed his Master Degree at SRM University, Chennai, India. He published several research papers in national and international journals and conferences. His research area is VLSI design and Low Power VLSI.