ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Implementation of High Speed Floating Point Dot Product Unit Based on Vedic Mathematics for DSP Applications

Kanagala Thejaswi
PG Scholar,
Department of ECE(VLSI),
Sree Vahini Institute of Science & Technology.

Kota Venkanna
Assistant Professor,
Department of ECE
Sree Vahini Institute of Science & Technology.

ABSTRACT:

Floating Point (FP) multiplication is widely used in large set of scientific and signal processing computation. Multiplication is one of the common arithmetic operations in these computations. A high speed floating point dot product based on vedi mathmatics is implemented in HDL. This paper presents a high speed binary single precession floating point multiplier based on vedic Algorithm. Hence two term dot product unit is preferred approach for high performance of the processors. Dot product unit multiplies two set of floating point operands and add that products in single operation. To improve speed multiplication of mantissa is done using Vedic multiplier replacing Carry Save Multiplier. In addition, the proposed design is compliant with single precision floating format and handles over flow, under flow, rounding and various exception conditions. The design achieved the with the help of Xilinx and modelsim sim tools.

KEY WORDS:

Vedic Algorithm, Double precision, Floating point, Multiplier, SINGLE PRECISION FLOATING, Verilog HDL.

I.INTRODUCTION:

The real numbers represented in binary format are known as floating point numbers. Based on single precision floating standard, floating point formats are classified into binary and decimal interchange formats. Floating point multipliers are very important in DSP applications. This paper focuses on double precision normalized binary interchange format. Figure I shows the Single precision binary format representation. Sign (S) is represented with one bit, exponent (E) and fraction (M or Mantissa) are represented with eleven and fifty two bits respectively.

For a number is said to be a normalized number, it must consist of 'one' in the MSB of the significand and exponent is greater than zero and smaller than 1023. The real number is represented by equations (I) & (2).

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

$$Value = (-1^{\text{Sign bit}}) * 2^{(\text{Exponent} -1023)} * (1.\text{Mantissa}) \quad (2)$$

Floating point implementation has been the interest of many researchers. In an single precision floating single precision pipelined floating point multiplier is implemented with custom 16/18 bit three stage pipelined floating point multiplier, that doesn't support rounding modes [1]. L.Louca, T.A.Cook, W.H. Johnson [2] implemented a floating point point dot multiplier by using a digit-serial multiplier. The design achieved 2.3 MFlops and doesn't support rounding modes. The multiplier handles the overflow and underflow cases but rounding is not implemented. The design achieves 30 I MFLOPs with latency of three clock cycles. The multiplier was verified against Xilinx floating point multiplier core.

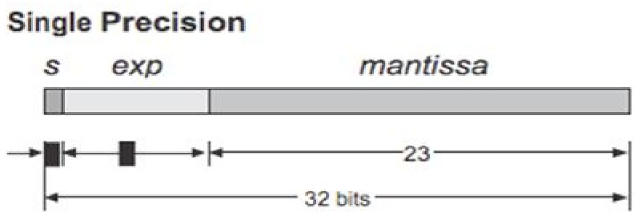

Figure1. Single Precision Floating Point Format.

The floating point point dot multiplier presented here is based on binary floating standard. We have designed a high speed floating point point dot multiplier using Verilog language. It operates at a very high frequency of 414.714 MFlops and occupies 648 slices. It handles the overflow, underflow cases and rounding mode.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

II.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
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 I in the MSB of the result (leading one).
The following steps are necessary to multiply two floating point numbers.
1. Multiplying the significand i.e. (I.MI * I.M2)
2. Placing the decimal point in the result
3. Adding the exponents i.e. (E I + E2 - Bias)
4. Obtaining the sign i.e. sl xor s2
5. Normalizing the result i.e. obtaining I at the MSB of the results “significand”
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence.

III. DESIGN:

In traditional floating-point hardware the dot product is performed with two multiplications and an addition. These operations may be performed in a serial fashion which limits the throughput.

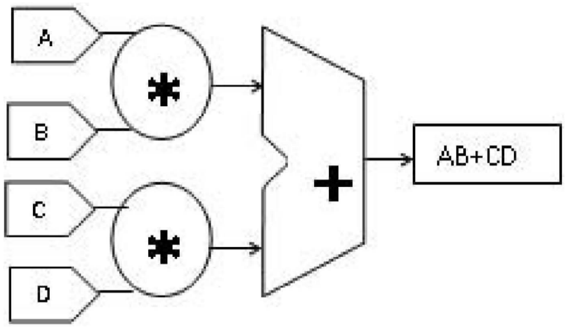

Figure 2.Conventional parallel dot product unit

The dot product operation performs
Y = A* B+ C * D

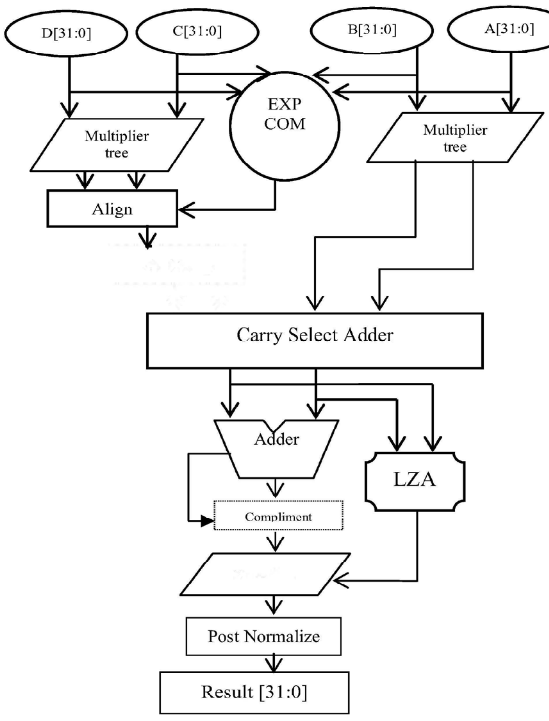

Figure.3. the Fused floating point dot product unit

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, the carry bits are passed diagonally downwards (i.e. the carry bit is propagated to the next stage). Partial products are generated by ANDing the inputs of two numbers and passing them to the appropriate adder. Carry save multiplier has three main stages:

1. The first stage is an array of half adders.
2. The middle stages are arrays of full adders. The number of middle stages is equal to the significand size minus two.
3. The last stage is an array of ripple carry adders.

This stage is called the vector merging stage.
The count of adders (Half adders and Full adders) in each stage is equal to the significand size minus one. For example,

a 4x4 carry save multiplier is shown in Figure 4 and it has the following stages:

1. The first stage consists of three half adders.
2. Two middle stages; each consists of three full adders.
3. The vector merging stage consists of one half adder and two full adders.

The decimal point is placed between bits 45 and 46 in the significand multiplier result. The multiplication time taken by the carry save multiplier is determined by its critical path. The critical path starts at the AND gate of the first partial products (i.e. a1b0 and a0b1), passes through the carry logic of the first half adder and the carry logic of the first full adder of the middle stages, then passes through all the vector merging adders. The critical path is marked in bold in Figure 4.

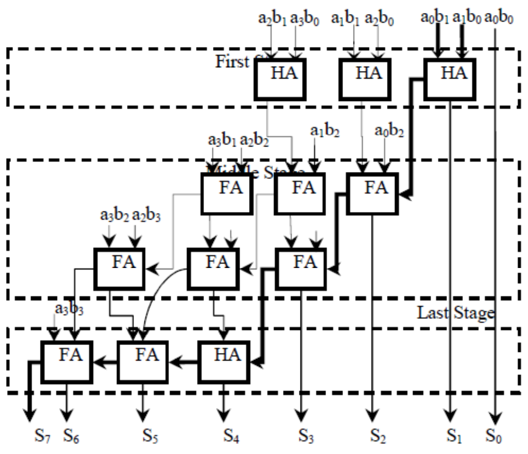

Fig. 4. 4x4 bit Carry Save multiplier

In Figure 4
1. Partial product: aibj ai and bj
2. HA: half adder.
3. FA: full adder.

IV.Proposed multiplier
Vedic Multiplier:

Vedic 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, vedic 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 1.5 times the height of its successor.

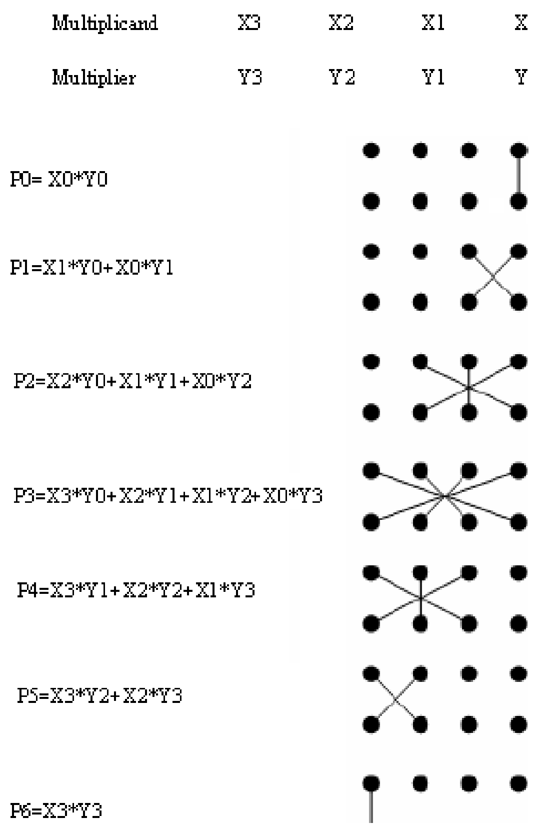

Fig 5: 4 by 4 Vedic Multiplier

The meaning of this sutra is “Vertically and crosswise” and it is applicable to all the multiplication operations. represents the general multiplication procedure of the 4x4 multiplication. This procedure is simply known as array multiplication technique. It is an efficient multiplication technique when the multiplier and multiplicand lengths are small, but for the larger length multiplication this technique is not suitable because a large amount of carry propagation delays are involved in these cases. To overcome this problem we are describing Nikhilam sutra for calculating the multiplication of two larger numbers.

V.ROUNDING AND EXCEPTIONS:

The IEEE standard specifies four rounding modes round to nearest, round to zero, round to positive infinity, and round to negative infmity. Table 1 shows the rounding modes selected for various bit combinations of rmode. Based on the rounding changes to the mantissa corresponding changes has to be made in the exponent part also.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Bit combination	Rounding Mode
00	round_nearest_even
01	round_to_zero
10	round_up
11	round_down

Table1: Rounding modes selected for various bit combinations of rmode

In the exceptions module, all of the special cases are checked for, and if they are found, the appropriate output is created, and the individual output signals of underflow, overflow, inexact, exception, and invalid will be asserted if the conditions for each case exist.

VI.RESULTS:

The floating point point dot multiplier design was simulated in Modelsim 6.6c and synthesized using Xilinx ISE 12.1.

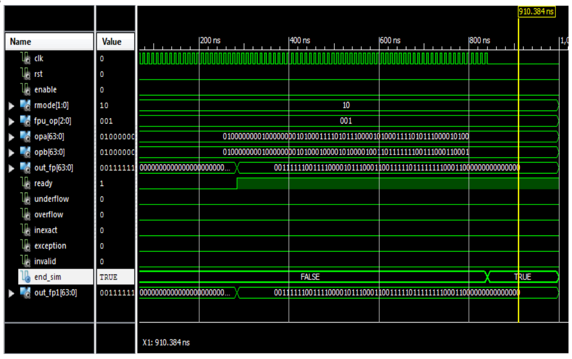

Fig. 6. Simulation Result of floating Point dot product Multiplier

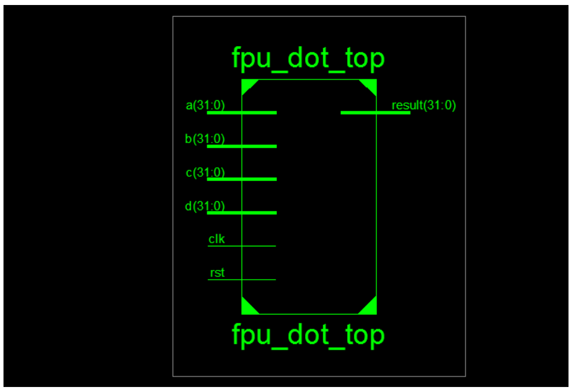

Fig.7. RTL Schematic for top level module

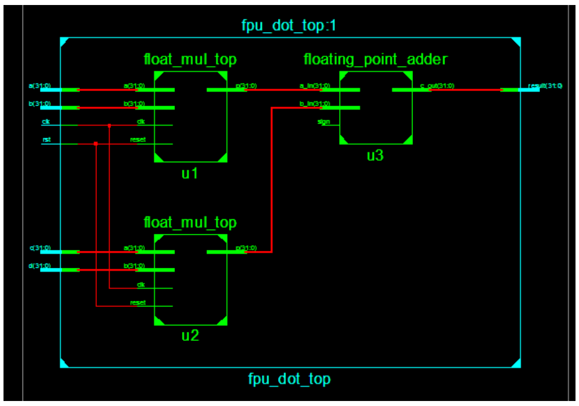

Fig.8. RTL Schematic for sub level module.

VII.CONCLUSION:

The floating point point dot multiplier supports the single precision floating binary interchange format. The implemented design is verified with floating point point dot multiplier [4] and Xilinx core, it provides high speed and supports double precision, which gives more accuracy compared to single precession. This design handles the overflow, underflow, and truncation rounding mode.

REFERENCES:

[1] 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.

[2] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs," Proceedings of 83rd IEEE Symposium on FPGAs for Custom Computing Machines (FCCM"96), pp. 107-116,1996.

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

[4] Mohamed AI-Ashraf)', Ashraf Salem, Wagdy Anis., "An Efficient Implementation of Floating Point Multiplier ", Saudi International Electronics, Communications and Photonics Conference (SIECPC), pp. 1-5,24-26 April 2011.

[5] B. Lee and N. Burgess, “Parameterisable Floating-point Operations on FPG A,” Conference Record of the ThirtySixth Asilomar Conference on Signals, Systems, and Computers, 2002.

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

[7] 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.

[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.