Remadevi R / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 2, March -April 2013, pp.283-286

Design and Simulation of Floating Point Multiplier Based on VHDL

Remadevi R
(M-Tech Student,Department of ECE, Mangalam College of Engineering ,MG University Kerala,India)

ABSTRACT

Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. This paper focuses only on single precision normalized binary interchange format targeted for Xilinx Spartan-3 FPGA based on VHDL. The multiplier was verified against Xilinx floating point multiplier core. It handles the overflow and underflow cases. Rounding is not implemented to give more precision when using the multiplier in a Multiply and Accumulate (MAC) unit.

.

Keywords – Floating point multiplication, VHDL simulation

I. INTRODUCTION

Floating point numbers are one possible way of representing real numbers in binary format, the IEEE 754[1] standard presents two different floating point formats, Binary interchange format and Decimal interchange format. Multiplying floating point numbers is a critical requirement for DSP applications involving large dynamic range. This paper focuses only on single precision normalized binary interchange format. It consists of a one bit sign (S), an eight bit exponent (E), and a twenty three bit fraction (M or Mantissa). An extra 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. Multiplying two numbers in floating point format is done by adding the exponent of the two numbers then subtracting the bias from their result,and multiplying the significand of the two numbers, and calculating the sign by XORing the sign of the two numbers.

The multiplier was verified against Xilinx floating point multiplier core.In this paper represention of floating point multiplier in such a way that rounding support isn't implemented, thus accommodating more precision if the multiplier is connected directly to an adder in a MAC unit. Exponents addition, Significand multiplication, and Result's sign calculation are independent and are done in parallel.Xilinx ISE Design Suite 13.3 tool & VHDL programming is used. ISIM tool is used for Simulation process .Xilinx core generator tool is used to generate Xilinx floating point multiplier core The whole multiplier (top unit) was simulated against the Xilinx floating point multiplier core generated by Xilinx core generator.

In [2], an IEEE 754 single precision pipelined floating point multiplier was implemented on multiple FPGAs (4 Actel A1280). In [3], a custom 16/18 bit three stage pipelined floating point multiplier that doesn't support rounding modes was implemented. In [4], a single precision floating point multiplier that doesn't support rounding modes was implemented using a digit-serial multiplier: using the Altera FLEX 8000 it achieved 2.3 MFlops. In [5], a parameterizable floating point multiplier was implemented using the software-like language Handel-C, using the Xilinx XCV1000 FPGA,a five stages pipelined multiplier achieved 28MFlops. In [6], a latency optimized floating point unit using the primitives of Xilinx Virtex II FPGA was implemented with a latency of 4 clock cycles. In [7] an efficient implementation of floating point multiplier is targeted for Xilnx Virtex -5 FPGA.

II. FLOATING POINT MULTIPLICATION ALGORITHM

The normalized floating point numbers are of the form shown in Fig. (1)

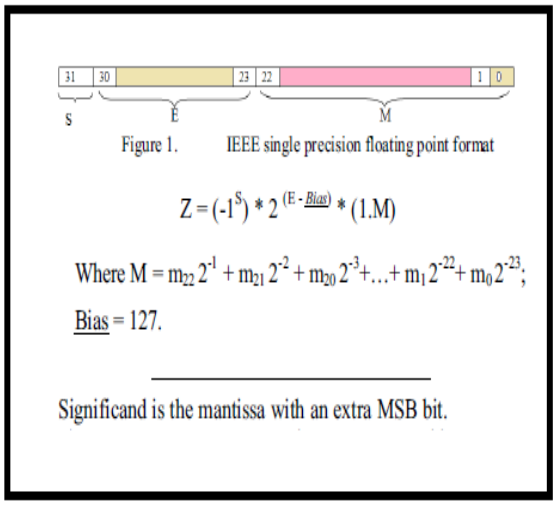


Figure 1: IEEE single precision floating point format

Floating point multiplication can be done by multiplying the significand of two floating point numbers and adding the exponents,then subtract the

Bias from added exponent result (E1 + E2 – Bias).Sign is obtained by xor-ing the MSB of two numbers,then normalize the result.Rounding of result is done to fit in the available bits and if desired finally check the underflow/overflow occurrence.The bias constant used is (127 = 001111111).

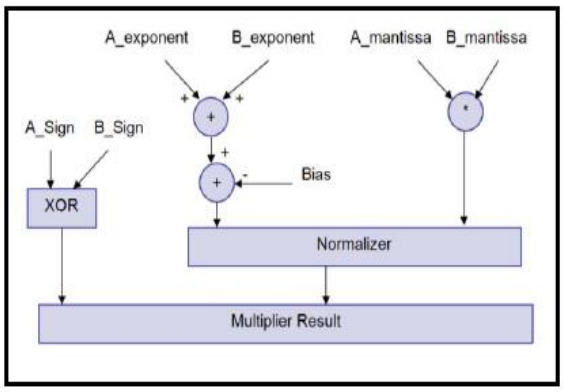
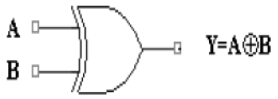


Figure 2: Block diagram of floating point multiplier

III.DESIGN OF FLOATING POINT MULTIPLIER

3.1 Sign Bit Calculation

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



A	B	Y=A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3: Sign bit calculator-XOR gate

3.2 Unsigned Adder (for Exponent Addition)

This unsigned adder is responsible for adding the exponent of the first input to the exponent of the second input and after that subtract the Bias (127) from the addition result (i.e. A_exponent + B_exponent - Bias). The result of this stage is called the intermediate exponent. The addition operation is done on 8 bits, and there is no need for a quick result because most of the calculation time is spent in the significand multiplication process (multiplying 24 bits by 24 bits); thus we need a moderate exponent adder and a fast significand multiplier.An 8-bit ripple carry adder is used to add the two input exponents. A ripple carry adder is a chain of cascaded full adders and one half adder; each full adder has three inputs (A, B, Ci) and two outputs (S, Co). The carry out(Co) of each adder is fed to the next full adder (i.e each carrybit "ripples" to the next full adder).The addition process produces an 8 bit sum (S7 to S0) and a carry bit (Co,7). These bits are concatenated to form a 9 bit addition result (S8 to S0) from which the Bias is subtracted. The Bias is subtracted using an array of ripple borrow subtractors. The addition process produces an 8 bit sum (S7 to S0) and a carry bit (Co,7). These bits are concatenated to form a 9 bit addition result (S8 to S0) from which the Bias is subtracted. The Bias is subtracted using an array of ripple borrow subtractors.

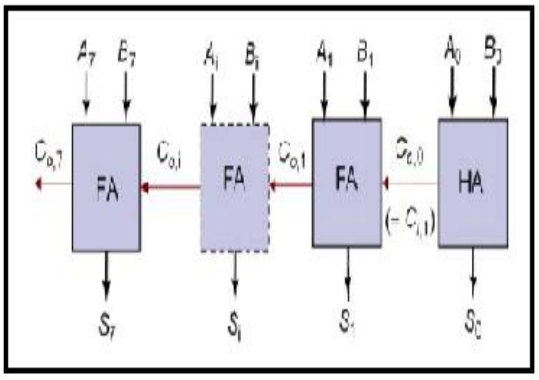


Figure 4:Unsigned Adder

3.3 Bias Subtraction

Subtract the bias constant (127 = 001111111) from unsigned exponent adder result for this, two zero subtractors (ZS) and seven one subtractors (OS) are used . S0…..S8 is the unsigned adder result (9 bit) .T=001111111 is the Bias constant. Bias subtractor result is R =S-T.

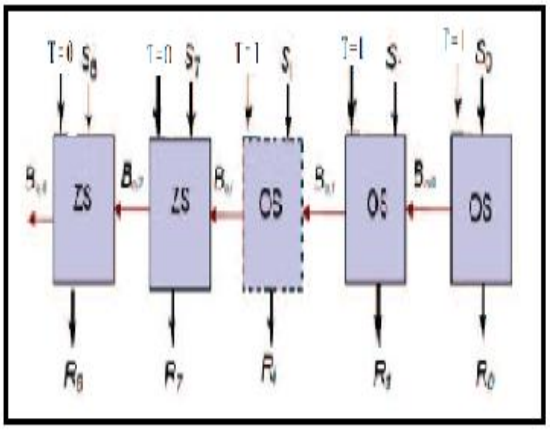


Figure 5: Bias Subtractor

3.4 Unsigned Multiplier (for significand multiplication)

This unit is responsible for multiplying the unsigned significand and placing the decimal point in the multiplication product. The result of significand multiplication will be called the intermediate product (IP). The unsigned significand

multiplication is done on 24 bit. Multiplier performance should be taken into consideration so as not to affect the whole multiplier's performance. A 24x24 bit carry save multiplier architecture is used 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 made by ANDing the inputs together and passing them to the appropriate adder.This is done in significand multiplication process which is one of the important steps in floating point multiplication

3.5 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. 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. If the leading one is at bit 47 then the intermediate product is shifted to the right and the exponent is incremented by 1.

3.6 Overflow/underflow 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 maybe 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 (sign determined according to the sign of the floating point multiplier inputs). When an underflow occurs an underflow flag signal goes high and the result turns to ±Zero (sign determined according to the sign of the floating point multiplier inputs). Denormalized numbers are signaled to zero with the appropriate sign calculated from the inputs and an underflow flag is raised.

IV. XILINX FLOATING POINT MULTIPLIER CORE

DSP48E is the IP core generated for simulation.Resource estimation & vhdl program can be obtained when a core is generated in xilinx. Using this vhdl program its simulation can be done So this simulation result can compare with proposed floating point multiplier

Figure 6: Xilinx floating point multiplier core generated by Xilinx core generator tool

The whole multiplier (top unit) was tested against the Xilinx floating point multiplier core generated by Xilinx core generator. Xilinx core was customized to have two flags to indicate overflow and underflow conditions. Xilinx core implements the "round to nearest" rounding mode .A testbench is used to generate the stimulus and applies it to the implemented floating point multiplier and to the Xilinx core then compares the results. The area of Xilinx core is less than the proposed floating point multiplier because the latter doesn't truncate/round the 48 bits result of the mantissa multiplier which is reflected in the amount of function generators and registers used to perform operations on the extra bits. But the proposed one indicates Normalized significand product of 48 bits separately give more precision when compared to Xilinx floating point core..So proposed floating point multiplier has greater importance when it utilized in another Unit-floating point adder to form a MAC unit involving large dynamic range

V. SOFTWARE TOOLS

Xilinx ISE Design Suite 13.3 tool & VHDL programming is used. ISIM tool is used for Simulation process .Xilinx core generator tool is used to generate Xilinx floating point multiplier core The whole multiplier (top unit) was tested against theXilinx floating point multiplier core. Xilinx core was customized to have two flags to indicate overflow and underflow conditions. Xilinx core implements the "round to nearest" rounding mode.A testbench is used to generate the stimulus and applies it to the proposed floating point multiplier and to the Xilinx core then compares the results.

VI.SIMULATION RESULTS

ISIM tool is used for Simulation process Xilinx core generator tool is used to generate Xilinx floating point multiplier core The whole multiplier (top unit) was tested against the Xilinx floating point multiplier core.

6.1 Simulation results of floating point multiplier core generated by Xilinx core generator tool

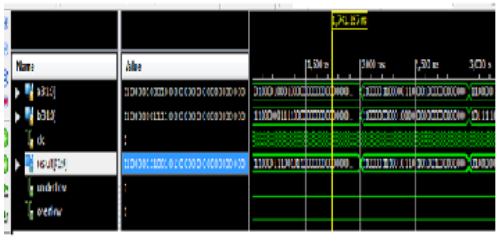


Figure 7: Simulation result of Xilinx floating point multiplier core generated by Xilinx core generator tool

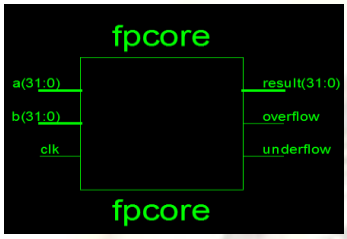


Figure 8: RTL Schematic of Xilinx floating point multiplier core generated by Xilinx core generator tool

6.2 Simulation results of proposed floating point multiplier

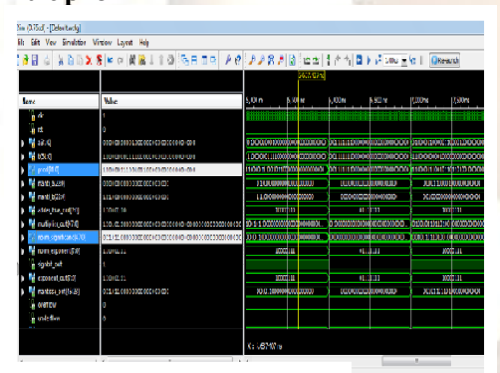


Figure 9: Simulation result of Proposed floating point multiplier

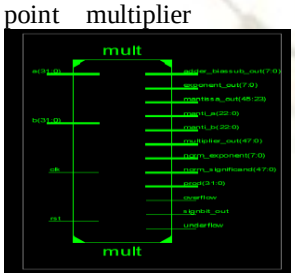


Figure 10: RTL Schematic of proposed Xilinx floating point multiplier

VII.CONCLUSION

This paper presents design and simulation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format, the proposed multiplier doesn't implement rounding and presents the significand multiplication result as is (48 bits), this gives better precision if the whole 48 bits are utilized in another unit; i.e.with a floating point adder to form a MAC unit. But the floating point multiplier core generated by Xilinx core generator does not indicates the entire 48 bits of mantissa due to rounding and is not beneficial in case of DSP application of large dynamic range especially when using it in another high precision floating point units like Multiply and Accumulate (MAC) unit.

REFERENCES

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

[2] B. Fagin and C. Renard, "Field Programmable Gate Arrays and FloatingPoint Arithmetic," *IEEE Transactions on VLSI, vol. 2, no. 3, pp. 365–367, 1994.*

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

[4] L. Louca, T. A. Cook, and W. H. Johnson, "Implementation of IEEESingle Precision Floating Point Addition and Multiplication on FPGAs,"*Proceedings of 83 the IEEE Symposium on FPGAs for CustomComputing Machines (FCCM'96), pp. 107–116, 1996.*

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

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

[7] Mohamed Al-Ashrafy, Ashraf Salem and Wagdy Anis" An Efficient Implementation of Floating PointMultiplier" *Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International*