ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 9, November 2012

High Speed Double Precision Floating Point Multiplier

Addanki Purna Ramesh[1], Rajesh Pattimi[2]

Department of ECE, Sri Vasavi Engineering College, Pedatadepalli, Tadepalligudem, India[1, 2]

ABSTRACT: **In this paper we describe an implementation of high speed IEEE 754 double precision floating point multiplier targeted for Xilinx Virtex-6 FPGA. Verilog is used to implement the design. The multiplier implement area optimized design, high speed operation with latency of seven clock cycles, it handles the overflow, underflow cases, and the multiplier support truncation rounding mode was implemented. The multiplier was verified against Xilinx floating point multiplier core.**

Keywords: binary floating point, multiplication, FPGA.

I INTRODUCTION

Floating point numbers are one possible way of representing real numbers in binary format; the IEEE 754 [3] 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 on double precision floating point binary interchange format. Figure 1 shows the IEEE 754 double precision floating point binary format representation; it consists of a one bit sign (S), an eleven bits exponent (E), and a fifty two bits fraction (M or Mantissa). An extra bit is added to the fraction to form what is called the significand1. If the exponent is greater than 0 and smaller than 2047, and there is 1 in the MSB of the significand then the number is said to be a normalized number, Significand is the mantissa with an extra MSB bit.

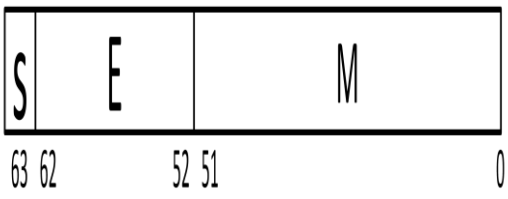

Figure . 1 IEEE double precision floating point format

$Z = (-1^S) * 2^{(E - Bias)} * (1.M)$
Where $M = m_{51}\, 2^{-1} + m_{50}\, 2^{-2} + m_{49}\, 2^{-3} + \ldots + m_1\, 2^{-51} + m_0\, 2^{-52}$
Bias = 1023.

Multiplying two numbers in floating point format is done by 1- calculating the sign by XORing the sign of the two numbers, 2- adding the exponent of the two numbers then subtracting the bias from their result, and 3- multiplying the significand 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).

II. FLOATING POINT MULTIPLICATION ALGORITHM

As stated in the introduction, normalized floating point numbers have the form of $Z = (-1^S) * 2^{(E - Bias)} * (1.M)$. To multiply two floating point numbers the following is done:

1. Obtaining the sign; i.e. S_a xor S_b
2. Adding the exponents; i.e. (E1 + E2 – *Bias*)
3. Multiplying the significand; i.e. (1.M1*1.M2)
4. Placing the decimal point in the significant result
5. Normalizing the result; i.e. obtaining 1 at the MSB of the results significand
6. Rounding the result to fit in the available bits
7. Checking for underflow/overflow occurrence

III. HARDWARE OF FLOATING POINT MULTIPLIER

The black box view of floating point multiplier is shown in figure 2.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 9, November 2012

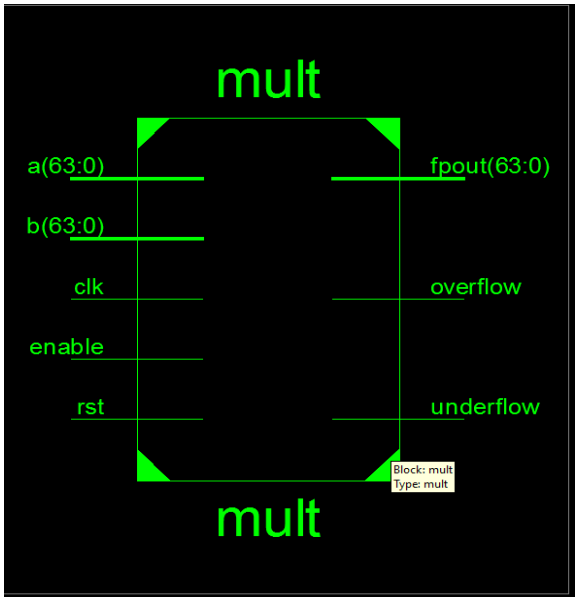

Figure .2 Black box view of floating point multiplier

A. 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 we find that this can be obtained by XORing the sign of two inputs.

B. exponent addition

This unsigned adder is responsible for adding the exponent of the first input to the exponent of the second input and subtracting the Bias (1023) from the addition result (i.e. A_exponent + B_exponent - Bias). The result of this stage is called the intermediate exponent. The add 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 53 bits by 53 bits); thus we need a moderate exponent adder and a fast significand multiplier.

An 11-bit ripple carry adder is used to add the two input exponents. As shown in Figure 3 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, C). The carry out (C) of each adder is fed to the next full adder (i.e each carry bit "ripples" to the next full adder). The addition process produces an 11 bit sum (S_{10} to S_0) and a carry bit (C_{11}). These bits are concatenated to form a 12 bit addition result (S_{12} to S_0) from which the Bias is subtracted.

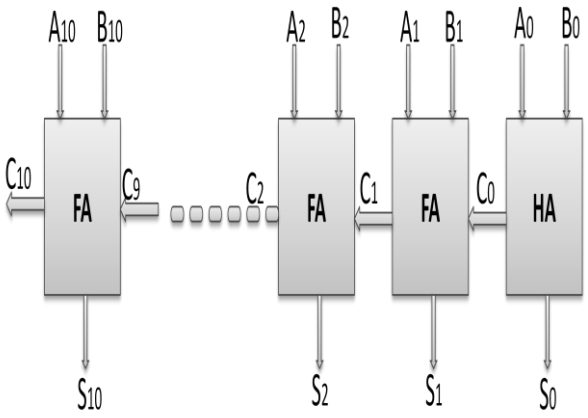

Figure. 3 Ripple Carry Adder

The Bias is subtracted using an array of ripple borrow subtractors. A normal subtractor has three inputs (minuend (S), subtrahend (T), Borrow in (Bi)) and two outputs (Difference (R), Borrow out (B)). The subtractor logic can be optimized if one of its inputs is a constant value which is our case, where the Bias is constant ($1023|_{10}$ = $001111111111|_2$).Table I shows the truth table for a 1-bit subtractor with the input T equal to 1 which we will call "one subtractor (OS)". Table II shows the truth table for a 1-bit subtractor with the input T equal to 0 which we will call "zero subtractor (ZS)".

Table I.
1-Bit Subtractor with the input T = 1

S	T	B_i	Difference(R)	B
0	1	0	1	1
1	1	0	0	0
0	1	1	0	1
1	1	1	1	1

Table II
1-Bit Subtractor with the input T = 0

S	T	B_i	Difference(R)	B
0	0	0	0	0
1	0	0	1	0
0	0	1	1	1
1	0	1	0	0

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 9, November 2012

Figure 4 shows the Bias subtractor which is a chain of 10 one subtractors (OS) followed by 2 zero subtractors (ZS); the borrow output of each subtractor is fed to the next subtractor. If an underflow occurs then $E_{result} < 0$ and the number is out of the IEEE 754 single precision normalized numbers range; in this case the output is signaled to 0 and an underflow flag is asserted.

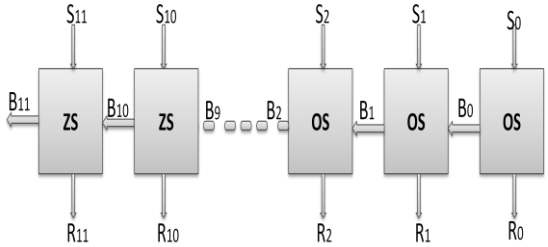

Figure .4 Ripple Borrow Subtractor

IV. 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 11 bits in size, and must be between 1 and 2046 otherwise the value is not a normalized one. An overflow may occur while adding the double precision floating point multiplier code was a checked using Design Xilinx targeting on Virtex-6 xc5vlx110-3ff1760. Figure 5 shows the simulation results of high speed double precision floating point multiplier 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. Table III shows the normalization effect on result's exponent and overflow/underflow detection.

Table III
Normalization effect on result's exponent and overflow/underflow detection

E_{result}	Category	Comments
$-1021 \leq E_{result} < 0$	Underflow	Can't be compensated during normalization
$E_{result} = 0$	Zero	May turn to normalized number during normalization (by adding 1 to it)
$1 \leq E_{result} < 2046$	Normalized number	May result in overflow during normalization
$2047 \leq E_{result}$	Overflow	Can't be compensated

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. Assume that E1 and E2 are the exponents of the two numbers A and B respectively; the results exponent is calculated by (1).

$$E_{result} = E1 + E2 - 1023 \text{ ---------- } (1)$$

E1 and E2 can have the values from 1 to 2046; resulting in E_{result} having values from -1021 (2-1023) to 3069 (4092-1023); but for normalized numbers, E_{result} can only have the values from 1 to 2046.

V. RESULTS

The whole multiplier (top unit) was tested against the Xilinx floating point multiplier core generated by Xilinx core and an efficient implementation of floating point multiplier in [1]. Xilinx core and multiplier in [1] 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 but multiplier doesn't support rounding modes.

A test bench is used to generate the stimulus and applies it to the high speed double precision floating point

VI. CONCLUSIONS

This paper presents an implementation of a floating point multiplier that supports the IEEE 754-2008 binary interchange format. The design implemented on a Xilinx Virtex6 xc6vlx110-3ff1760 FPGA it achieves with a latency of seven clock cycles, handles the overflow, underflow cases, and this multiplier support truncation rounding mode was implemented.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021
International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 9, November 2012

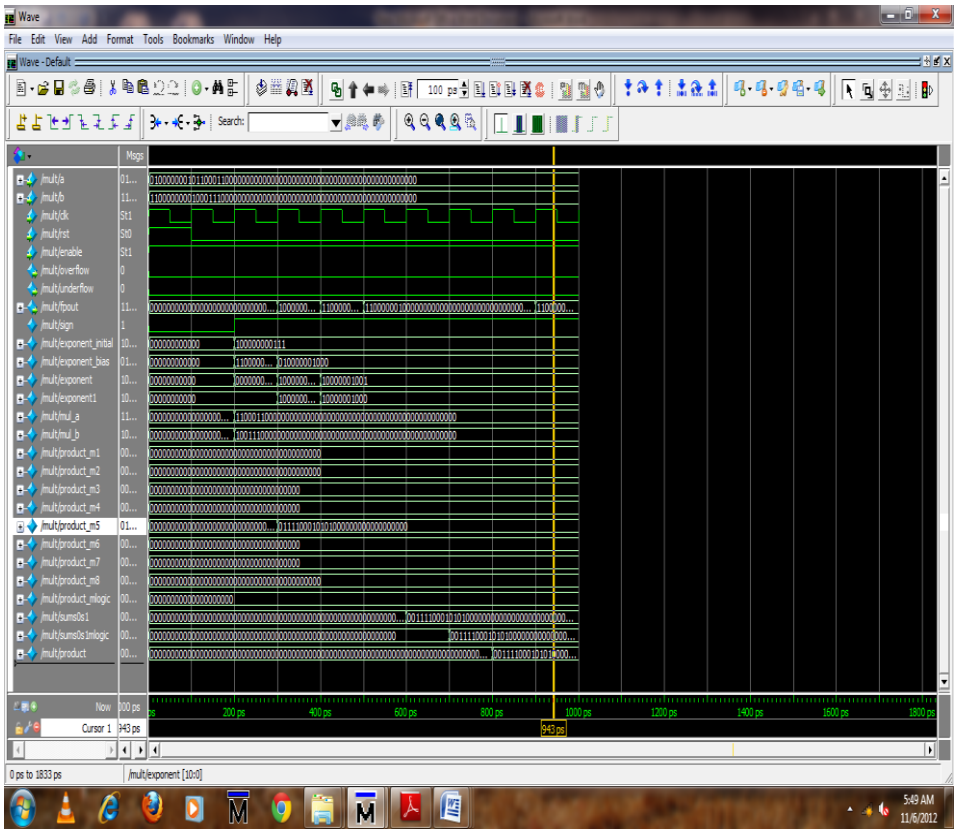

Figure 5 Simulation results of high speed double precision floating point multiplier

REFERENCES

[1] M.Al-Ashrafy, A.Salem and W.Anis,"An Efficient Implementation of Floating Point Multiplier " Electronics Communications and Photonics Conference(SIECPC) 2011 Saudi International, pp.1-5,2011.

[2] F.de Dinechin and B.Pasca. Large multipliers with fewer DSP blocks.In Field Pro- grammable Logic and Applications. IEEE, Aug. 2009.

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

[4] Patterson, D. & Hennessy, J. (2005), computer Organization and Design : The Hardware/software Interface , Morgan Kaufmann.

[5] 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

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

[7]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.

[8] John G. Proakis and Dimitris G. Manolakis (1996), "Digital Signal Processing: Principles,.Algorithms and Applications", Third Edition.

[9] 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.

[10] 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.