ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

An Efficient Field Programmable Gate Array Implementation of Double Precision Floating Point Multiplier using VHDL

Sukhvir Kaur[1] and Parminder Singh Jassal [2]

M.Tech Student, ECE, Yadvindra College of Engineering, Talwandi Sabo (Pb)-India[1]

Assistant Professor, ECE, Yadvindra College of Engineering, Talwandi Sabo (Pb)-India[2]

Abstract: Floating point arithmetic is widely used in many areas, especially scientific computation and signal processing. The main applications of floating points today are in the field of medical imaging, biometrics, motion capture and audio applications. Multipliers play an important role in today's digital signal processing and various other applications. A system's performance is generally determined by the performance of the multiplier, because the multiplier is generally the slowest element in the system. The way floating point operations are executed depends on the data format of the operands. IEEE standards specify a set of floating point formats single precision and double precision. This paper presents an efficient FPGA implementation of double precision floating point multiplier using VHDL.

Key Words: Single Precision, Double Precision, Field Programmable Gate Array, Multiplier.

I. INTRODUCTION

Floating point number system is a common choice for many scientific computations due to its wide dynamic range feature. For instance, floating point arithmetic is widely used in many areas, especially in scientific computation, numerical processing, image processing and signal processing. The term floating point is derived from the fact that there is no fixed number of digits before and after the decimal point, that is, the decimal point or binary point can float. There are also representations in which the number of digits before and after the decimal or binary point is fixed; called fixed-point representations. The advantage of floating-point representation over fixed point representation is that it can support a much wider range of values. The floating point numbers is based on scientific notation. A scientific notation is just another way to represent very large or very small numbers in a compact form such that they can be easily used for computations. The floating point multiplication operations are greatly affected by how the floating point multiplier is designed.

Floating point number consists of three fields:
1. Sign (S): It used to denote the sign of the number i.e. 0 represent positive number and 1 represent negative number.
2. Significand or Mantissa (M): Mantissa is part of a floating point number which represents the magnitude of the number.
3. Exponent (E): Exponent is part of the floating point number that represents the number of places that the decimal point (binary point) is to be moved.

Number system is completely specified by specifying a suitable base β, significand (mantissa) M, and exponent E. A floating point number F has the value

$$F=(-1)^{S}\ M\ \beta^{E}$$

The way floating point operations are executed depends on the data format of the operands. IEEE standards specify a set of floating point data formats, single precision and double precision. The Single precision consists of 32 bits and the Double precision consists of 64 bits. Figure 1 shows the IEEE single and double precision data formats.

Sign S	8 bits –Biased Exponent	23 bits – Unsigned Fraction (f)

(a) IEEE single precision data format

Sign S	11 bits –Biased Exponent	52 bits – Unsigned Fraction (f)

(b) IEEE double precision data format

Figure 1: IEEE Single and Double Precision data Format

The value of the floating point number represented in single precision format is

$$F=(-1)^{S}\ 1.f\ 2^{E-127}$$

where 127 is the value of bias in single precision data format and exponent E ranges between 1 to 254, and E = 0 and E = 255 are reserved for special values.
The value of the floating point number represented in double precision data format is

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

$$F=(-1)^{S}\,1.f\;2^{E-1023}$$

where 1023 is the value of bias in double precision data format. Exponent E ranges between 1 to 2046, the values of E = 0 and E = 2047 are reserved for special values.

II. DOUBLE PRECISION FLOATING POINT MULTIPLIER

Multipliers are key components of many high performance systems such as FIR filters, Microprocessors, Digital Signal Processors etc. A system's performance is generally determined by the performance of the multiplier, because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a critical issue for an effective system design.

A. Floating point multiplication algorithm

Multiplication of floating point numbers F_1 and F_2 is a five step process. To multiply two floating point numbers the following is done:

1. Obtaining the sign; i.e. S1 xor S2.
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 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 [2].

B. Floating point Multiplier design

The multiplier for the floating point numbers represented in IEEE 754 format can be divided into three different units: Mantissa Calculation unit, Exponent Calculation unit and Sign Calculation unit [3].The Multiplier receives two 64-bit floating point numbers. First these numbers are unpacked by separating the numbers into sign, exponent, and mantissa bits. The floating point multiplication is carried out in following three parts.
Sign calculation unit:-
In this unit, we determine the sign of the product by performing a XOR operation on the sign bits of the two operands.
Exponent calculation unit:-
This unsigned adder is used for adding the exponent of the first input to the exponent of the second input and subtracting the Bias (1023) from the addition result.
Eresult = A_exponent + B_exponent – Bias
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. Overflow/underflow means that the result's exponent is too large/small.
Mantissa Calculation unit:-
The multiplication is done in two steps, partial product generation and partial product addition. For double precision operands (53-bit fraction fields), a total of 53*53-bit multiplier is required. Mantissa multiplier unit performs multiplication operation. After this the output of mantissa is normalized, i.e. if the MSB of the result obtained is not 1, then it is left shifted to make the MSB 1. If changes are made by shifting then corresponding changes has to be made in exponent also. The mantissa of operand A and the leading '1' (for normalized numbers) are stored in the 53-bit register (Ma). The mantissa of operand B and the leading '1' (for normalized numbers) are stored in the 53-bit register (Mb). Multiplying all 53 bits of Ma by 53 bits of Mb would result in a 106-bit product. Rounding is used to fit the result in available bit then output of mantissa multiplication is 56 bits.

III. FPGA IMPLEMENTATION OF MULTIPLIER

IV. The FPGA-based implementations of the double precision floating point multiplier have been presented in this section. For implementation purpose, Xilinx Integrated Software Environment ISE 10.1i software tool has been used. The double precision floating point multiplier component has been coded in VHDL .Code has been synthesized and simulated using Xilinx ISE 10.1i which are mapped on to Spartan 3E FPGA. The RTL view, design summary and simulation result of the floating point multiplier are shown in following section. The 'clk', 'rst', 'enable', 'opa' and 'opb' are the inputs of double precision floating point multiplier. The 'exponent_5', 'product_7'and 'sign' is the outputs.

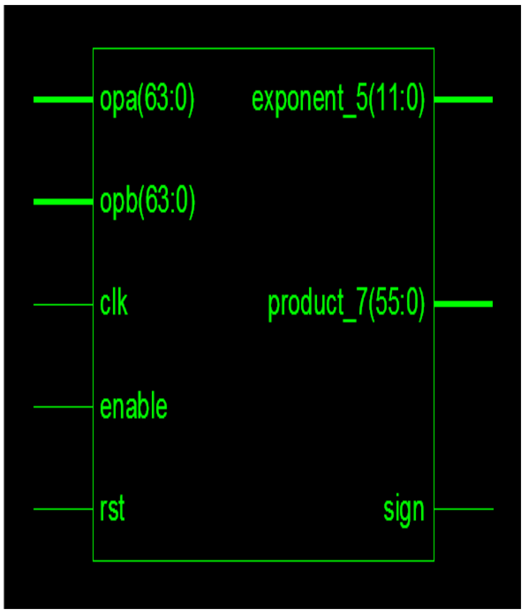
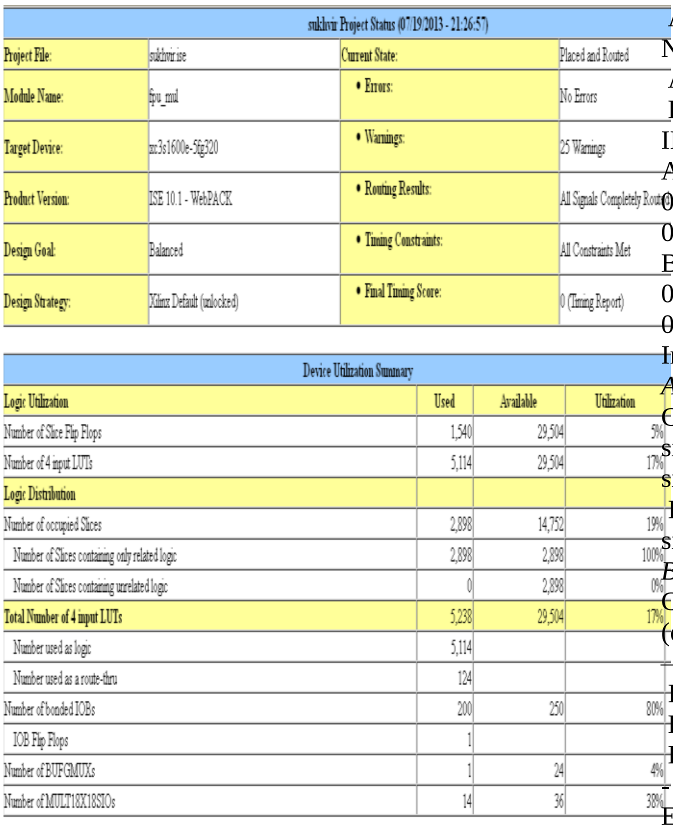


Figure 2: RTL Diagram of the Floating point multiplier

Table 1 show the summary of resources utilized by the double precision Floating point multiplier for a Spartan 3E device.

Table 1: Resource utilization by the Floating point multiplier



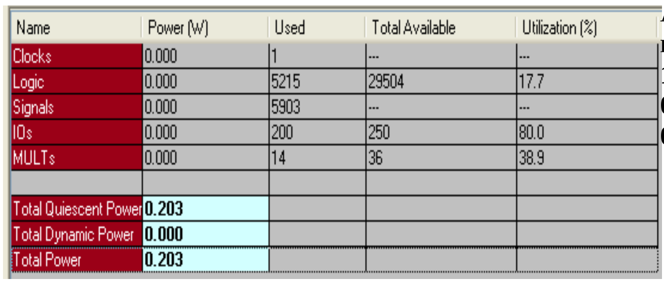
sukhvir Project Status (07/19/2013 - 21:26:57)			
Project File:	sukhvir.ise	Current State:	Placed and Routed
Module Name:	fpu_mul	• Errors:	No Errors
Target Device:	xc3s1600e-5fg320	• Warnings:	25 Warnings
Product Version:	ISE 10.1 - WebPACK	• Routing Results:	All Signals Completely Routed
Design Goal:	Balanced	• Timing Constraints:	All Constraints Met
Design Strategy:	Xilinx Default (unlocked)	• Final Timing Score:	0 (Timing Report)

Device Utilization Summary			
Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	1,540	29,504	5%
Number of 4 input LUTs	5,114	29,504	17%
Logic Distribution			
Number of occupied Slices	2,898	14,752	19%
Number of Slices containing only related logic	2,898	2,898	100%
Number of Slices containing unrelated logic	0	2,898	0%
Total Number of 4 input LUTs	5,238	29,504	17%
Number used as logic	5,114		
Number used as a route-thru	124		
Number of bonded IOBs	200	250	80%
IOB Flip Flops	1		
Number of BUFGMUXs	1	24	4%
Number of MULT18X18SIOs	14	36	38%

Table 2: Device utilization summary (xc3s1600e-5fg320)

Logic Utilization	Used	Available	Utilization in %age
No. of slices	2898	14752	19%
No. of 4 input LUTs	5114	29504	17%
No of slice Flip Flops	1540	29504	5%
Number of bonded IOBs	200	250	80%
Max Delay	0.477ns		

Table 3: Summary of Power Consumption by Floating point multiplier



Name	Power (W)	Used	Total Available	Utilization (%)
Clocks	0.000	1	---	---
Logic	0.000	5215	29504	17.7
Signals	0.000	5903	---	---
IOs	0.000	200	250	80.0
MULTs	0.000	14	36	38.9
Total Quiescent Power	0.203			
Total Dynamic Power	0.000			
Total Power	0.203			

Let's suppose a multiplication of two floating-point numbers
A and B, where A= -18.0 and B = 9.5 Binary representation
of the operands:
A = -10010.0   B = +1001.1
Normalized representation of the operands:
A = $-1.001x2^{4}$
B = $+1.0011x2^{3}$
IEEE representation of the operands:
A= 1 10000000011
0010000000000000000000000000000000000000000000
000000   (hex) 64'hC032000000000000
B= 0 10000000010
0011000000000000000000000000000000000000000000
000000   (hex) 64'h4023000000000000
In our case, we get:

A) Sign output:

Calculation of the sign of the result: Sign = Sa ⊕ Sb .The sign of the result is given by the XOR of the operands signs (Sa and Sb).

In our example, we get: Sign = 1 ⊕ 0 = **1** i.e. a negative sign.

B) Exponent output:

Calculation of the exponent field of the result: Er (exponent_5) = (Ea-1023) + (Eb-1023) + 1023 =Ea + Eb – 1023

In our example:
Ea   10000000011
Eb   10000000010
- 1023 10000000001
Er (exponent_5) 10000000110, the exponent of the result in hexadecimal is **12'h406**.

C) Mantissa output:

Multiplication of the mantissas: we must extract the mantissas, adding a 1 as most significant bit, for normalization. Ma and Mb is 53 bit i.e. Mr = Ma*Mb (53*53)

Ma=1001000000000000000000000000000000000000000
0000000000
Mb=100110000000000000000000000000000000000000
00000000000
The Mr (106-bit) result of the multiplication is: 0x5580000000000   only the most significant bits are useful: after normalization we get the 56-bit mantissa of the result is

56'h55800000000000

A result (sign, exponent and mantissas) gives us the final result of our multiplication:
1 10000000110
010101100000000000000000000000000000000000
0000000000

= $-18.0x9.5 = -1.0101011 \times 2^{1030-1023}$
= - 10101011.0 = -171.010

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 7, July 2013

The output is also verified by the Xilinx ISE simulator that is same. Figure 3 shows Simulation results of the double precision floating point multiplier for given input.

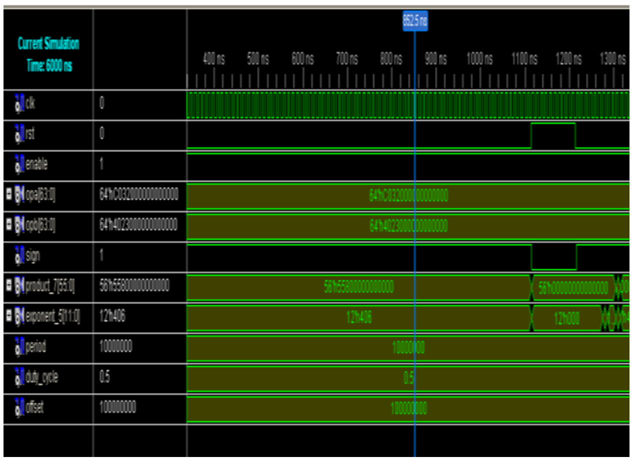


Figure 3: Simulation results of floating point multiplier for given input

IV. CONCLUSION

This paper presents FPGA implementation of double precision floating point multiplier that supports the IEEE 754-2008 binary interchange format. The multiplier is designed in VHDL. The design implemented on a Xilinx ISE 10.1i tool targeting the Spartan 3E device xc3s1600e-5fg320. The power consumption for the design is 203 mW.The implementation of Floating point multiplier shows the use of 5% Slice Flip Flops, 17% 4 input LUTs, 80% bonded IOBs and 19% number of slices. Although double precision floating point multiplier uses more resources and consumes more power, yet it has very high speed. The max delay of this design is 0.477ns

REFERENCES

[1] Purna Ramesh Addanki,Venkata Nagaratna Tilak Alapati and Mallikarjuna Prasad Avana, "An FPGA Based High Speed IEEE - 754 Double Precision Floating Point Adder/Subtractor and Multiplier Using Verilog", International Journal of Advanced Science and Technology, Vol. 52, pp.61-74, March 2013.
[2] Riya Saini and R.D.Daruwala,"Efficient Implementation of Pipelined Double Precision Floating Point Multiplier",International Journal of Engineering Research and Applications, Vol. 3, Issue 1, pp.1676-1679, January -February 2013.
[3] Anna Jain, Baisakhy Dash and Ajit Kumar Panda,"FPGA Design of a Fast 32-bit Floating Point Multiplier Unit", IEEE Conference Publications, pp. 545 – 547, 2012.
[4] Addanki Purna Ramesh and Rajesh Pattimi," High Speed Double Precision Floating Point Multiplier",International Journal of Advanced Research in Computer and Communication Engineering, Vol. 1, Issue 9, pp. 647 – 650, November 2012.
[5] Xia Hong and Jia Jinging,"Research and optimization on Rounding Alogrithms for Floating-Point Multiplier",IEEE Conference Publications,International Conference on Computer Science and Electronics Engineering, pp. 137 – 142, 2012.
[6] Tashfia.Afreen, Minhaz. Uddin Md Ikram, Aqib. Al Azad, and Iqbalur Rahman Rokon, "Efficient FPGA Implementation of Double Precision Floating Point Unit Using Verilog HDL" ,International Conference on Innovations in Electrical and Electronics Engineering,Oct. 6-7, 2012 .
[7] Puneet Paruthi, Tanvi Kumar and Himanshu Singh,"Simulation of IEEE 754 Standard Double Precision Multiplier using Booth Techniques", International Journal of Engineering Research and Applications, Vol. 2, Issue 5, pp. 1761-1766, September- October 2012.
[8] B.Sreenivasa Ganesh, J.E.N.Abhilash and G. Rajesh Kumar, "Design and Implementation of Floating Point Multiplier for Better Timing Performance", International Journal of Advanced Research in Computer Engineering & Technology, Vol. 1, Issue 7, September 2012.
[9] Mohamed Al-Ashrafy, Ashraf Salem and Wagdy Anis,"An Efficient Implementation of Floating Point Multiplier",IEEE Electronics,Communications and Photonics Conference (SIECPC), Saudi International , pp. 1 - 5, 2011.
[10] Soojin Kim And Kyeongsoon Cho, "Design Of High-Speed Modified Booth Multipliers Operating At Ghz Ranges", World Academy Of Science, Engineering and Technology, 2010.
[11] Gong Renxi, Zhang Shangjun, Zhang Hainan, Meng Xiaobi, Gong Wenying, Xie Lingling and Huang Yang, "Hardware Implementation of a High Speed Floating Point Multiplier Based on FPGA", IEEE Conference Publications, 4th International Conference on Computer Science & Education, pp.1902 – 1906, 2009.
[12] Saroja.V Siddamal, R.M Banakar and B.C. Jinaga, "Design of High-Speed Floating Point Multiplier", 4th IEEE International Symposium on Electronic Design, Test & Applications, pp. 285 – 289, 2008.
[13] Manish Kumar Jaiswal and Nitin Chandrachoodan,"Efficient Implementation of IEEE Double Precision Floating-Point Multiplier on FPGA", IEEE Region 10 Colloquium and the Third ICIIS, Kharagpur, India, December 2008.