An FPGA Based High Speed IEEE - 754 Double Precision Floating Point Adder/Subtractor and Multiplier Using Verilog

Purna Ramesh Addanki [1], Venkata Nagaratna Tilak Alapati[2]
and Mallikarjuna Prasad Avana[3]

[1]*Department of ECE, Sri Vasavi Engineering College,*
Pedatadepalli, Tadepalligudem, India
[2] *Professor of ECE, V.R.Siddhartha Engineering College, Kanuru, Vijayawada, India*
[3]*Professor of ECE, JNTUK, Kakinada, India*
purnarameshaddanki@gmail.com, avntilak@yahoo.com, a_malli65@yahoo.com

Abstract

Floating Point (FP) addition, subtraction and multiplication are widely used in large set of scientific and signal processing computation. A high speed floating point double precision adder/subtractor and multiplier are implemented on a Virtex-6 FPGA. In addition, the proposed designs are compliant with IEEE-754 format and handles over flow, under flow, rounding and various exception conditions. The adder/subtractor and multiplier designs achieved the operating frequencies of 363.76 MHz and 414.714 MHz with an area of 660 and 648 slices respectively.

Keywords**:** *Double precision, floating point, adder/subtractor, multiplier, FPGA, IEEE-754, Virtex-6*

1. Introduction

The real numbers represented in binary format are known as floating point numbers. Based on IEEE-754 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 1 shows the IEEE-754 double 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 (1) and (2).

Figure 1. IEEE-754 double precision floating point format

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

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

Floating point implementation on FPGAs has been the interest of many researchers. Oklobdzija implemented 32-bit and 64-bit leading zero detector (LZD) circuit using CMOS and ECL technology [1]. In [2], Pavle Belanovic and Miriam Leeser implemented reconfigurable floating point arithmetic unit using VHDL, which is mapped on to Xilinx XCV1000 FPGA. K. Scott Hemmert Keith and D. Underwood implemented open source library of highly optimized floating point units for Xilinx FPGAs. The units are fully IEEE compliant. The double precision add and multiply achieved the operating frequency of 230 MHz using a 10 stage adder pipeline and a 12 stage multiplier pipeline [3]. In [4], floating point adder was implemented using Leading One Predictor (LOP) algorithm instead of Leading One Detector (LOD) algorithm. The main function of the LOP is to predict the leading number of zeros in the addition result, working in parallel with the 2's complement adder. Dhiraj Sangwan and Mahesh K. Yadav implemented adder/subtractor and multiplication units for floating point arithmetic using VHDL. The floating point multiplication operation was implemented using sequential architecture based on Booth's Radix-4 recoding algorithm. For floating point addition, the sequential addition could have been complex so the combinational architecture has been implemented [5]. In [6], double precision floating point adder/subtractor was implemented using dynamic shifter, LOD, priority encoder. The design achieved the operating frequency of 353 MHz for a latency of 12 clock cycles.

In [7], an IEEE-754 single precision pipelined floating point multiplier is implemented on multiple FPGAs (4 Actel A1280). Nabeel Shirazi, Walters, and Peter Athanas implemented custom 16/18 bit three stage pipelined floating point multiplier, that doesn't support rounding modes [8]. L.Louca, T.A.Cook, W.H. Johnson [9] implemented a single precision floating point multiplier by using a digit-serial multiplier and Altera FLEX 8000. The design achieved 2.3 MFlops and doesn't support rounding modes. In [10], a parameterizable floating point multiplier is implemented using five stages pipeline, Handel-C software and Xilinx XCV1000 FPGA. The design achieved the operating frequency of 28MFlops.The floating point unit [11] is implemented using the primitives of Xilinx Virtex II FPGA. The design achieved the operating frequency of 100 MHz with a latency of 4 clock cycles. Mohamed Al-Ashrafy, Ashraf Salem, and Wagdy Anis [12] implemented an efficient IEEE-754 single precision floating point multiplier and targeted for Xilinx Virtex-5 FPGA. The multiplier handles the overflow and underflow cases but rounding is not implemented. The design achieves 301 MFLOPs with latency of three clock cycles. The multiplier was verified against Xilinx floating point multiplier core.

The double precision floating point adder/subtractor and multiplier presented here is based on IEEE-754 binary floating standard. We have designed a high speed double precision floating point adder/subtractor and multiplier using Verilog language and ported on Xilinx Vertex-6 FPGA. Adder/subtractor and multiplier operates at very high frequencies of 363.76 and 414.714 MFlops and occupies 660 and 648 slices respectively. It handles the overflow, underflow cases and rounding mode.

2. Implementation of Double Precision Floating Point Adder/Subtractor

The black box view and block diagram of double precision floating point adder/subtractor is shown in Figures 2 and 3 respectively. The input operands are separated into their sign, mantissa and exponent components. This module has inputs opa and opb of 64-bit width and clk, enable, rst are of 1-bit width. One of the operands is applied at opa and other operand at opb. Larger operand goes into 'mantissa_large' and 'exponent_large', similarly the smaller operand goes into 'mantissa_small' and 'exponent_small'. To determine which operand is larger, compare only the exponents of the two operands, so in fact, if the exponents are equal, the smaller operand might populate the mantissa_large and exponent_large registers. This is not an issue because the reason the operands are compared is to find the operand with the larger exponent, so that the mantissa of the operand with the smaller exponent can be right shifted before performing the addition. If the exponents are equal, the mantissas are added without shifting. The inter-connection of sub-modules of double precession floating point adder/subtractor is shown in Figure 4. Subtraction is similar to addition in that you need to calculate the difference in the exponents between the two operands, and then shift the mantissa of the smaller exponent to the right before subtracting. The flow chart of double precision floating point adder /subtractor is shown in Figure 5.

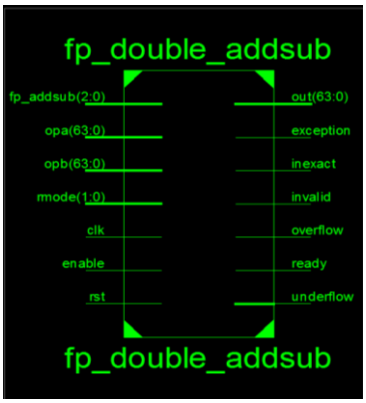


Figure 2. Black box view of double precision floating point adder/subtractor

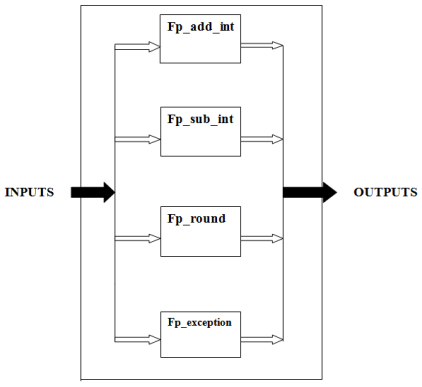


Figure 3. Block diagram of double precision floating point adder/subtractor

2.1 Algorithm

1. Compare exponent and mantissa of both numbers. Find the large exponent and mantissa and small exponent and mantissa.
2. If both exponents are equal then put leading 1 and 0 in front of each mantissa for overflow, *i.e.*, 52+2 bits.
3. If overflow occurs the exponent must be increased by one. 54^{th} bit will be shifted out of mantissa and this will be saved for rounding purpose.
4. The leftmost 1 in the result becomes leading 1 of mantissa and next 52 bits are actual mantissa.
5. If there is overflow 1 in result, the exponent of larger operand is increased by one. Then add both mantissas.
6. If one exponent is larger than other then subtract smaller exponent from larger exponent, and difference is saved.
7. Now shift smaller mantissa by that subtracted exponent difference.
8. If exponents are equal, *i.e.*, larger exponent is equal to smaller exponent after shifting then put leading 1 in both mantissas and perform addition for mantissas.
9. For subtraction if both exponents are equal then put implied 1 in front of mantissa and perform subtraction.
10. Store that result in "diff" register. Count the number of zeros in "diff" before the left most 1.
11. Reduce the large operand exponent by number of zeros in front of least 1.
12. The left most 1 will be leading 1 in front of mantissa.
13. Finally rounding and normalization is done.

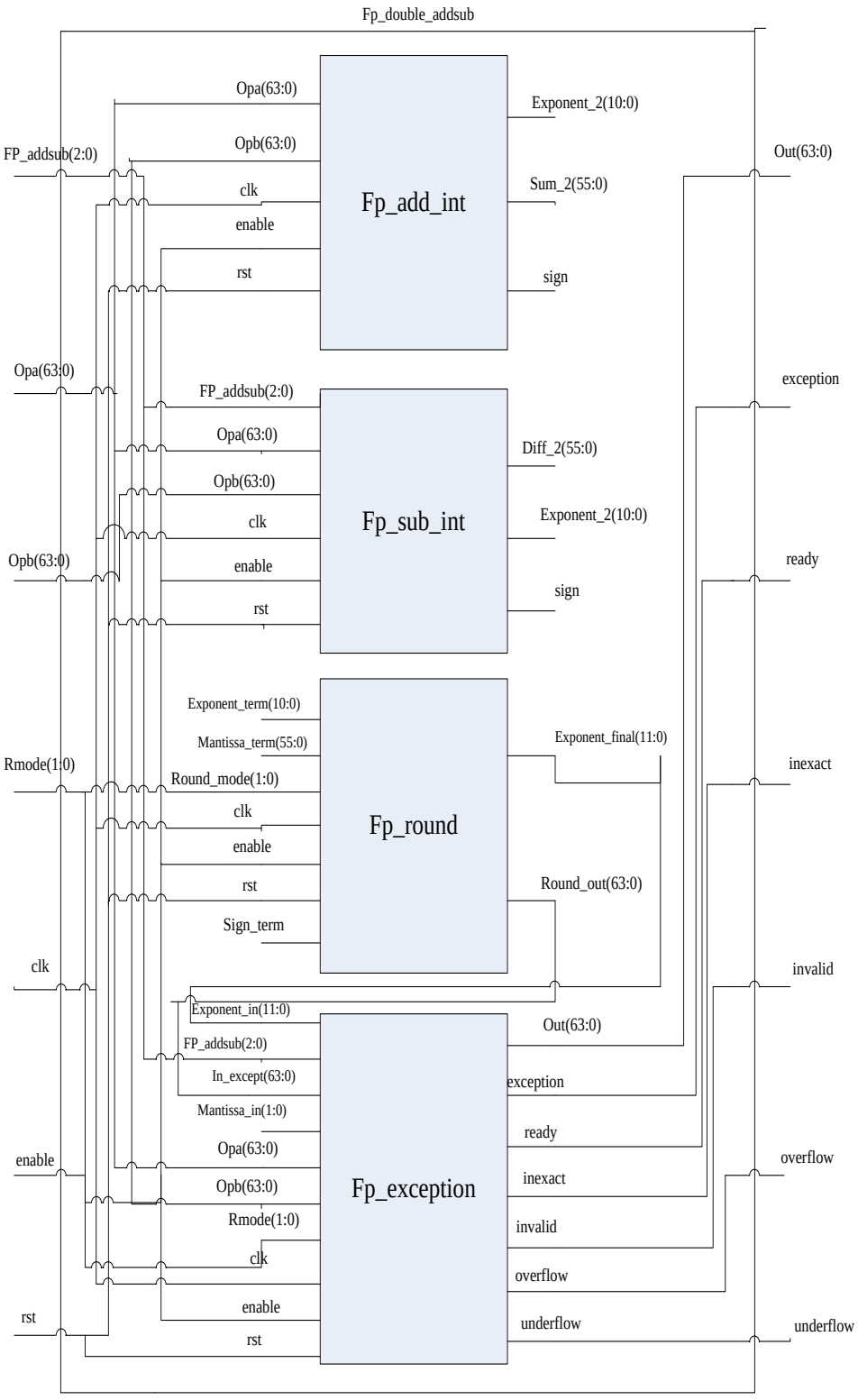


Figure 4. Inter-connection of sub-modules of double precision floating point adder/subtractor

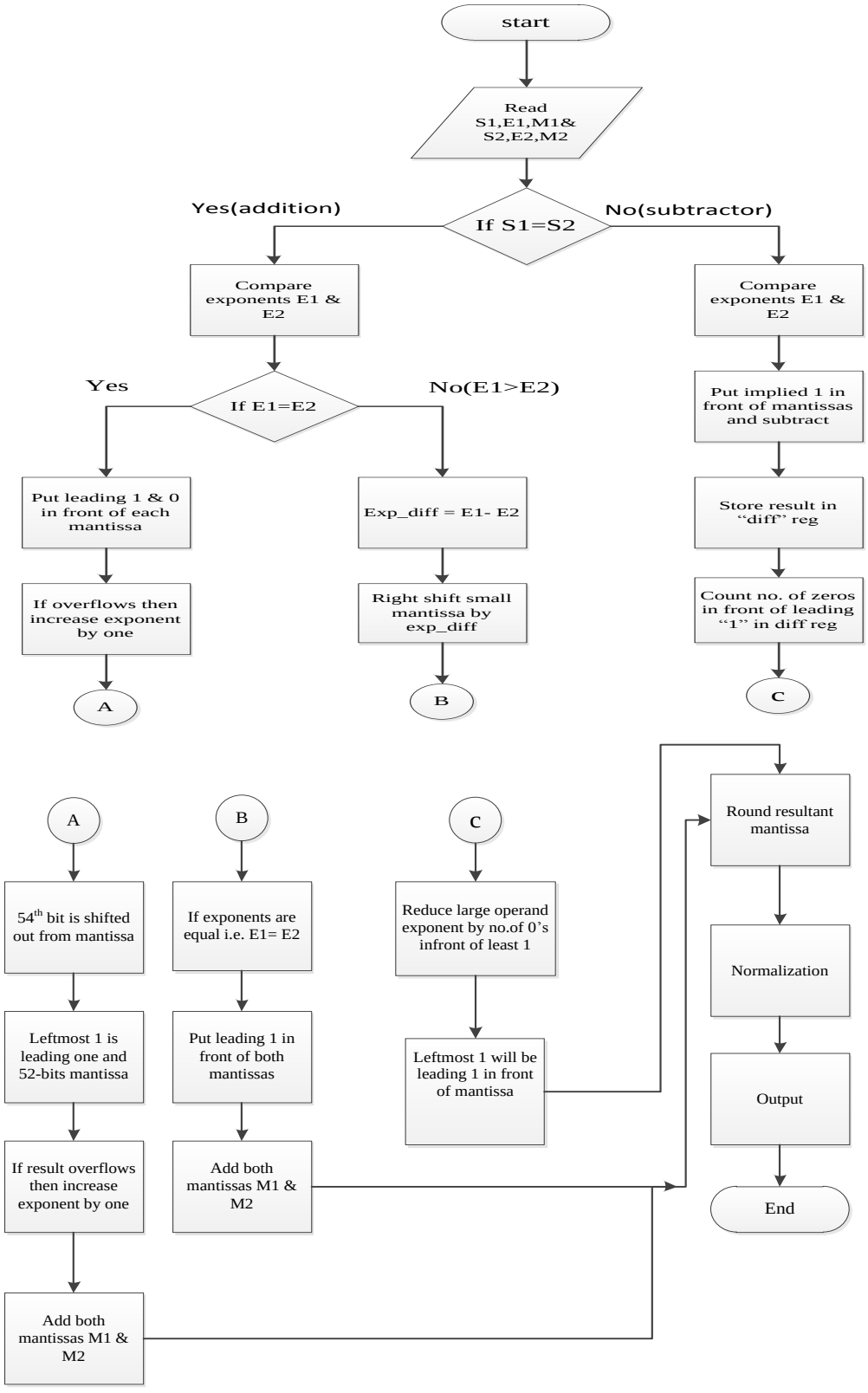


Figure 5. Flow chart of double precision floating point adder/subtractor

3. Implementation of Double Precision Floating Point Multiplier

3.1 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 1 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.*, (1.M1*1.M2)
2. Placing the decimal point in the result
3. Adding the exponents, *i.e.*, (E1 + E2 – *Bias*)
4. Obtaining the sign i.e. s1 xor s2
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

3.2 Implementation

In this paper we implemented a double precision floating point multiplier with exceptions and rounding. Figure 6 shows the multiplier structure that includes exponents addition, significand multiplication, and sign calculation. Figure 7 shows the multiplier, exceptions and rounding that are independent and are done in parallel.

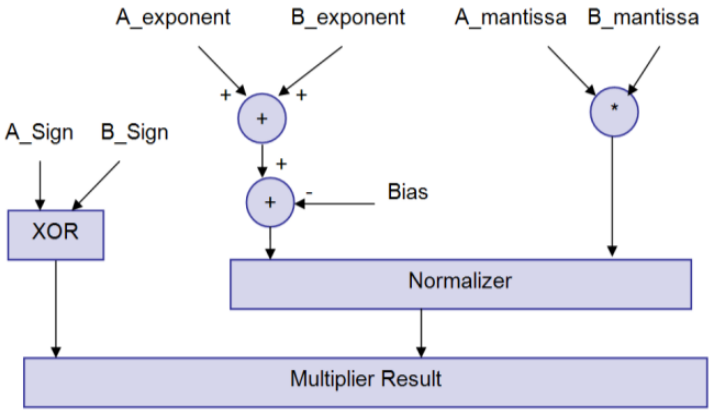


Figure 6. Multiplier Structure

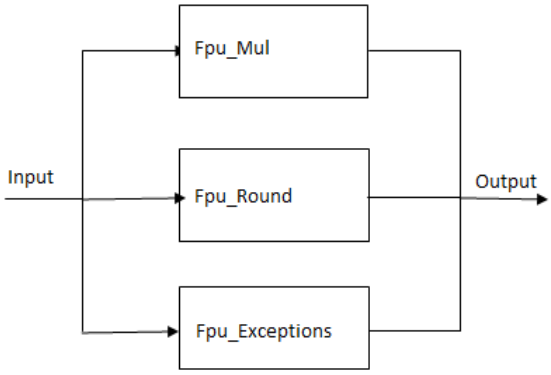


Figure 7. Multiplier structure with rounding and exceptions

3.2.1 Multiplier

The black box view of the double precision floating point multiplier is shown in figure 8.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 sign logic is a simple XOR. The exponents of the two numbers are added and then subtracted with a bias number i.e., 1023. Mantissa multiplier block performs multiplication operation. After this the output of mantissa division 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.

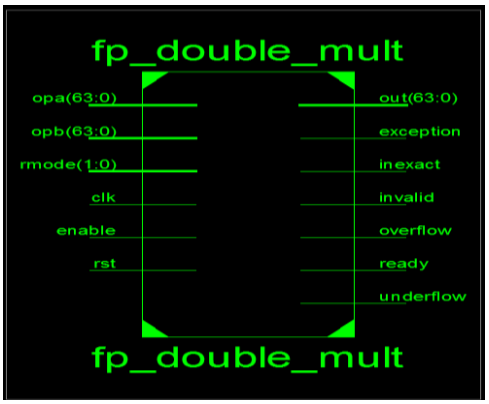


Figure 8. Black box view of floating point double precision multiplier

The multiplication operation is performed in the module (fpu_mul). The mantissa of operand A and the leading '1' (for normalized numbers) are stored in the 53-bit register (mul_a). The mantissa of operand B and the leading '1' (for normalized numbers) are stored in the 53-bit register (mul_b). Multiplying all 53 bits of mul_a by 53 bits of mul_b would result in a 106-bit product. 53 bit by 53 bit multipliers are not available in the most popular Xilinx and Altera FPGAs, so the multiply would be broken down into smaller multiplies and the results would be added together to give the final 106-bit product. The module (fpu_mul) breaks up the multiply into smaller 24-bit by 17-bit multiplies. The Xilinx Virtex-6 device contains DSP48E1 slices with 25 by 18 two's complement multipliers, which can perform a 24-bit by 17-bit unsigned multiply.

The breakdown of the multiply in module (fpu_mul) is done as follows:

product_a = mul_a[23:0] * mul_b[16:0]
product_b = mul_a[23:0] * mul_b[33:17]
product_c = mul_a[23:0] * mul_b[50:34]
product_d = mul_a[23:0] * mul_b[52:51]
product_e = mul_a[40:24] * mul_b[16:0]
product_f = mul_a[40:24] * mul_b[33:17]
product_g = mul_a[40:24] * mul_b[52:34]
product_h = mul_a[52:41] * mul_b[16:0]
product_i = mul_a[52:41] * mul_b[33:17]
product_j = mul_a[52:41] * mul_b[52:34]

The products (a-j) are added together, with the appropriate offsets based on which part of the mul_a and mul_b arrays they are multiplying. In this work the adders in the Virtex-6 DSP48E slices have been used that follow each 24 by 17 multiply block. The final 106-bit product is stored in the register 'product'. The output will be left-shifted if there is not a '1' in the MSB of product. The number of leading zeros in register 'product' is counted by signal 'product_shift'. The output exponent will also be reduced by 'product_shift'. The exponent fields of operands A and B are added together and then the value (1023) is subtracted from the sum of A and B. If the resultant exponent is less than 0, then the 'product' register needs to be right shifted by the amount. This value is stored in register 'exponent_under'. The final exponent of the output operand will be 0 in this case, and the result will be a denormalized number. If exponent_under is greater than 52, then the mantissa will be shifted out of the product register, and the output will be 0, and the 'underflow' signal will be asserted. The mantissa output from the fpu_mul module is in 56-bit register 'product_7'. The MSB is a leading '0' to allow for a potential overflow in the rounding module. The first bit '0' is followed by the leading '1' for normalized numbers, or '0' for denormalized numbers. Then the 52 bits of the mantissa follow. Two extra bits follow the mantissa and are used for rounding purposes. The first extra bit is taken from the next bit after the mantissa in the 106-bit product result of the multiply. The second extra bit is an OR of the 52 LSB's of the 106-bit product.

4. Rounding and Exceptions

The IEEE standard specifies four rounding modes round to nearest, round to zero, round to positive infinity, and round to negative infinity. 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.

Table1. Rounding modes selected for various bit combinations of rmode

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

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.

5. Results

The double precision floating point adder/subtractor and multiplier designs were simulated in Modelsim 6.6c and synthesized using Xilinx ISE 12.2i which are mapped on to Virtex-6 FPGA. The simulation results of 64-bit floating point double precision adder /subtractor and multiplier are shown in Figure 9 and 10 respectively. The 'opa' and 'opb' are the inputs and 'out' is the output. Table 2 gives the device utilization for implementing the circuits on Virtex-6 FPGA. Table 3 shows the timing summary of double precision floating point adder/subtractor and multiplier.

The area and operating frequency of double precision floating point adder/subtractor, [6] and Xilinx core are given in Table 4. Manish Kumar Jaiswal and Ray C.C. Cheung [6] implemented double precision floating point adder /subtractor and it occupies an area of 1397 slices and its operating frequency is 353 MHz, where as in case of Xilinx core, these are 1266 slices and 284 MHz respectively. Hence the present design provides high operating frequency.

The area and operating frequency of double precision floating point multiplier, single precision floating point multiplier [12] and Xilinx core are given in Table 5. M. Al-Ashrafy, A. Salem and W. Anis [12] implemented single precision floating point multiplier and it occupies an area of 604 slices and its operating frequency is 301.114 MHz, where as in case of Xilinx core, these are 266 slices and 221.484 MHz respectively. Hence the present design provides high operating frequency with more accuracy.

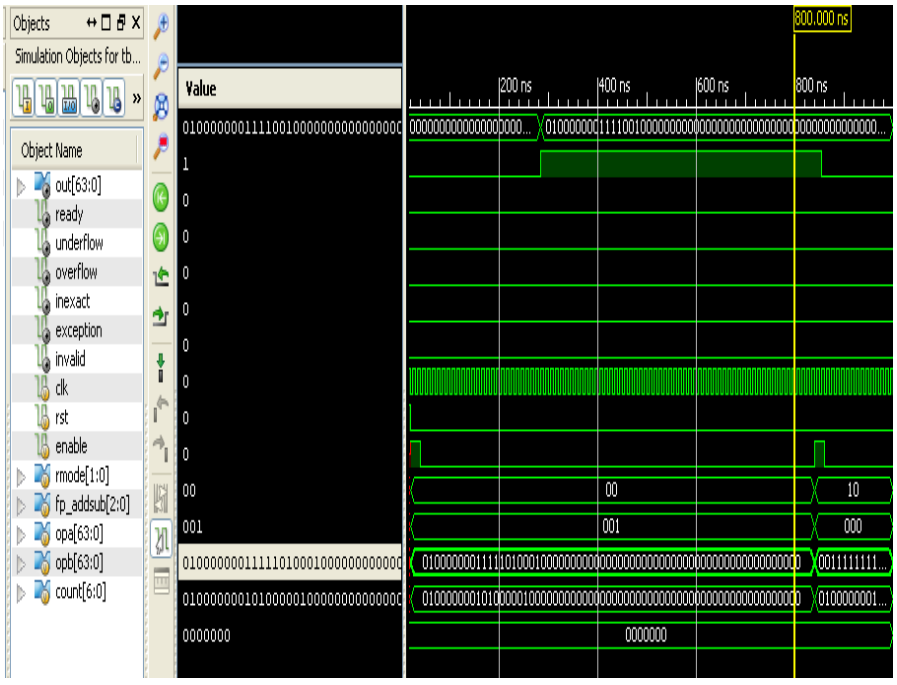


Figure 9. Simulation results of double precision floating point adder/subtractor

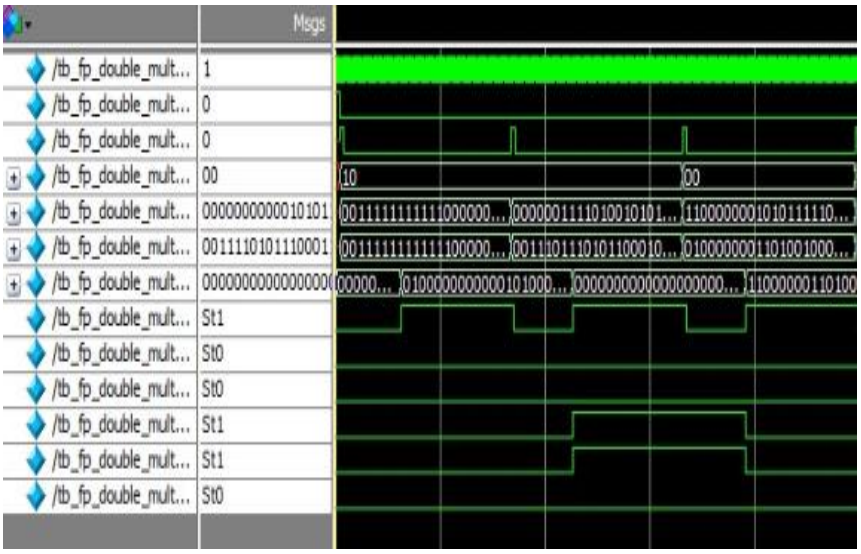


Figure 10. Simulation results of double precision floating point multiplier

Table 2. Device utilization summary (6vlx75tff484-3) of double precision floating point adder/subtractor and multiplier

Slice Logic Utilization	Adder/subtractor Used	Multiplier Used
Number of Slice Registers (Flip-Flops)	2423	1,998
Number of Slice LUTs	2049	2,181
Number of Occupied Slices	660	648

Table 3.Timing summary of double precision floating point adder/subtractor and multiplier

Parameter	Adder/subtractor	Multiplier
Minimum Period (ns)	2.749	2.411
Maximum Frequency (MHz)	363.769	414.714

Table 4. Area and operating frequency of double precision floating point adder/subtractor, [6] and Xilinx core [6]

Parameter	Present Work	Manish Kumar Jaiswal and Ray C.C. Cheung [6]	Xilinx Core [6]
No. of Slices Required	660	1397	1266
Frequency (MHz)	363.76	353	284

Table 5. Area and operating frequency of double precision floating point multiplier, single precision floating point multiplier [12] and Xilinx core [12]

Device Parameters	Present Work	M.Al-Ashrafy, A.Salem and W.Anis [12]	Xilinx Core[12]
Precision	Double	Single	Single
No. of Slices	648	604	266
Maximum Frequency (MHz)	414.714	301.114	221.484

6. Conclusion

The double precision floating point adder/subtractor and multiplier supports the IEEE-754 binary interchange format, targeted on a Xilinx Virtex-6 xc6vlx75t-3ff484 FPGA. The designs achieved the operating frequencies of 363.76 MHz and 414.714 MFLOPs with an area of 660 and 648 slices respectively. The adder/subtractor design operates at a frequency which is 3% and 28% more compared to [6] and Xilinx core respectively. As compared to the single precision floating point multiplier [12] and Xilinx core, the multiplier design supports double precision, provides high speed and gives more accuracy. These designs handles the overflow, underflow, rounding mode and various exception conditions.

References

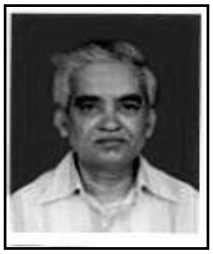
[1] V. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: comparison with logic synthesis", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 2, no. 1, **(1994)** March, pp. 124–128.

[2] P. Belanovic and M. Leeser, "A Library of Parameterized Floating-Point Modules and Their Use", in 12th International Conference on Field-Programmable Logic and Applications (FPL-02). London, UK: Springer-Verlag, **(2002)** September, pp. 657–666.

[3] K. Hemmert and K. Underwood, "Open Source High Performance Floating-Point Modules", in 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM-06), **(2006)** April, pp. 349–350.
[4] A. Malik and S. -B. Ko, "A Study on the Floating-Point Adder in FPGAs", in Canadian Conference on Electrical and Computer Engineering (CCECE-06), **(2006)** May, pp. 86–89.
[5] D. Sangwan and M. K. Yadav, "Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic", in International Journal of Electronics Engineering, **(2010)**, pp. 197-203.
[6] M. K. Jaiswal and R. C. C. Cheung, "High Performance FPGA Implementation of Double Precision Floating Point Adder/Subtractor", in International Journal of Hybrid Information Technology, vol. 4, no. 4, **(2011)** October.
[7] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic", IEEE Transactions on VLSI, vol. 2, no. 3, **(1994)**, pp. 365–367.
[8] 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), **(1995)**, pp. 155–162.
[9] 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), **(1996)**, pp. 107–116.
[10] A. Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, vol. 2, **(2001)**, pp. 897-900.
[11] 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)**.
[12] M. Al-Ashrafy, A. Salem, W. Anis, "An Efficient Implementation of Floating Point Multiplier", Saudi International Electronics, Communications and Photonics Conference (SIECPC), **(2011)** April 24-26, pp. 1-5.

Authors

Addanki Purna Ramesh has more than 14 years of experience in teaching. He is presently working as Associate professor of Electronics and Communication Engineering at Sri Vasavi Engineering College, Tadepalligudem. He is Life Member of MIETE, Associate Member in Institute of Engineers (India).



Alapati Venkata Nagaratna Tilak has more than 25 years of teaching and research experience. He obtained his Master's degree from Indian Institute of Technology, Kanpur and Ph.D. from Indian Institute of Technology, Madras during 1984 and 1997 respectively. He is presently working as a professor of Electronics and Communication Engineering at V.R.Siddhartha Engineering College, Vijayawada. He is a Member of IEEE, Fellow of Institution of Electronics and Telecommunication Engineers (IETE), Fellow of Institution of Engineers (India). He is also life member of Indian Society for Technical Education (ISTE).

Avana Mallikarjuna Prasad has more than 22 years of experience in teaching. He is presently working as a professor of Electronics and Communication Engineering at JNTUK, Kakinada. He is Life Member of ISTE, IETE, ISI, and Society of EMC. He won best teacher award by student evaluation of 2008 batch outgoing students. He has guided about 40 students in M.Tech Instrumentation Engineering and presently guiding 8 research students for their PhD works. His areas of interest are Antennas and Process control Instrumentation. He has 25 publications in various International and National Journals and conferences. He has conducted a "National Workshop on Electromagnetic field applications" in the year 2004.