INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 4, Issue 5 AUG 2015

AN FPGA BASED HIGH SPEED DOUBLE PRECISION FLOATING POINT MULTIPLIER USING VERILOG

N.GIRIPRASAD [(1)] , K.MADHAVA RAO[(2)]

VLSI System Design,Tudi Ramireddy Institute of Technology & Sciences[(1)]

Asst.Prof., Tudi Ramireddy Institute of Technology & Sciences [(2)]

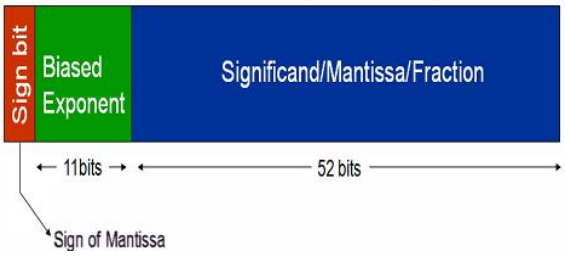
Abstract- Floating-point numbers are widely adopted in many applications due to their dynamic representation capabilities. Basically floating point numbers are one possible way of representing real numbers in binary format. Multiplying floating point numbers is also a critical requirement for DSP applications involving large dynamic range. The IEEE 754 standard presents two different floating point formats, Binary interchange format and Decimal interchange format. Multiplication is one of the common arithmetic operations in these computations. A high speed floating point double precision multiplier is implemented on a Virtex-6 FPGA. In addition, the proposed design is compliant with IEEE-754 format and handles over flow, under flow, rounding and various exception conditions. The design achieved the operating frequency of 414.714 MHz with an area of 648 slices.

Keywords- Double precision, Floating point, Multiplier,FPGA,IEEE-754.

I. 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 I 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.



Figurel. TEEE-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. In [I] , an IEEE-754 single precision pipelined floating point multiplier is implemented on multiple FPGAs (4 Actel AI280).

Nabeel Shirazi, Walters, and Peter Athanas implemented custom 16/18 bit three stage pipelined floating point multiplier, that doesn't support rounding modes [2] . L.Louca, T.A.Cook, W.H. Johnson [3] 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 [4], a parameterizable floating point

multiplier is implemented using five stages pipeline, Handel-C software and Xilinx XCYIOOO FPGA.The design achieved the operating frequency of 28MFlops. The floating point unit [5] is implemented using the primitives of Xilinx Yirtex IT FPGA. The design achieved the operating frequency of 100 MHz with a latency of 4 clock cycles. Mohamed AI-Ashraf}', Ashraf Salem, and Wagdy Anis [6] implemented an efficient TEEE-754 single precision floating point multiplier and targeted for Xilinx Yirtex-5 FPGA. The multiplier handles the overflow and underflow cases but rounding is not implemented. The design achieves 30I MFLOPs with latency of three clock cycles. The multiplier was verified against Xilinx floating point multiplier core.

The double precision floating point multiplier presented here is based on TEEE-754 binary floating standard. We have designed a high speed double precision floating point multiplier using Verilog language and ported on Xilinx Yertex-6 FPGA. It operates at a very high frequency of 414.714 MFlops and occupies 648 slices. It handles the overflow, underflow cases and rounding mode.

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.

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. IMPLEMENTATION OF DOUBLE PRECISION FLOATING POINT MUL TIPLTER

In this paper we implemented a double precision floating point multiplier with exceptions and rounding.

Figure 2 shows the multiplier structure that includes exponents addition, significand multiplication, and sign calculation. Figure 3 shows the multiplier, exceptions and rounding that are independent and are done in parallel.

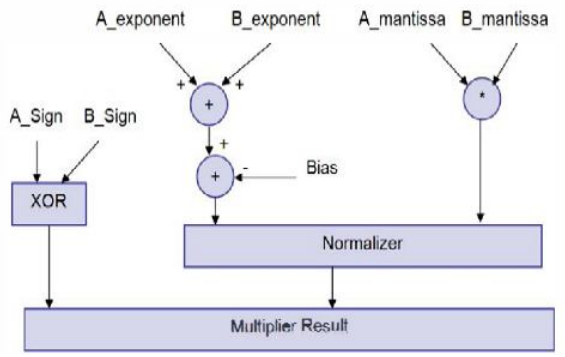
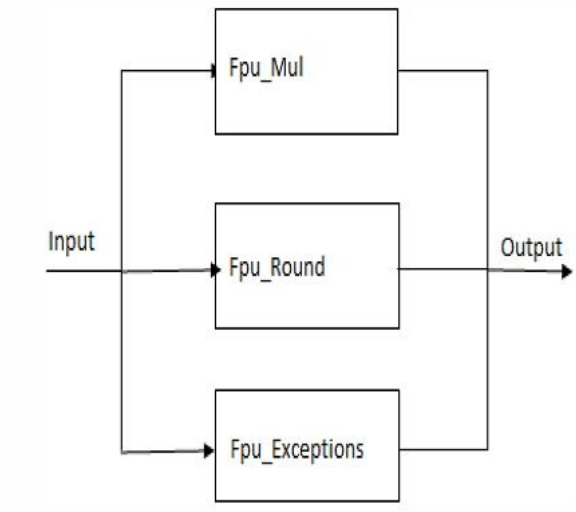


Figure 2. Multiplier structure.



A. Multiplier

The black box view of the double precision floating point multiplier is shown in figure 4.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 I, then it is left shifted to make the MSB I. If changes are made by shifting then corresponding changes has to be made in exponent also. The multiplication operation is performed in the module (fJ:lU_mul). The mantissa of operand A and the leading 'I' (for normalized numbers) are stored in the 53-bit register (mul_ a). The mantissa of operand B and the leading' I' (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 (fJ:lU_mul) breaks up the multiply into smaller 24-bit by 17-bit multiplies. The Xilinx Virtex-6 device contains DSP48E I slices with 25 by 18 twos complement multipliers, which can perform a 24-bit by 17-bit unsigned multiply.

The breakdown of the multiply in module (fJ:lU_mul) is broken up 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] * mutb[52:51]

product_e = mul_a[40:24] * mul_b[16:0]

product_f= mul_a[40:24] * mutb[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 'I' 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 (fJ:lU_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 'I' 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.

B. 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.

Table!: 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

IV. RESULTS

The double precision floating point multiplier design was simulated in Modelsim 6.6c and synthesized using Xilinx ISE 12.2i which was mapped on to Virtex-6 FPGA. The simulation results of 64-bit floating point double precision multiplier are shown in figure 5. The 'opa' and 'opb' are the inputs and 'out' is the output. Table 2 shows the device utilization for implementing the circuit on Virtex-6 FPGA. Table 3 shows the timing summary of double precision floating point multiplier. Table 4 shows the area and operating frequency of double precision floating point multiplier, Single precision floating point multiplier [6] and Xilinx core respectively. M.AI-AshrafY, A.Salem and W.Anis [6] implemented single

precision floating point multiplier and it occupies an area of 604 slices and it' s operating frequency is 301.114 MHz. Where as in case of XiI in x core, it occupies an area of 266 slices and it' s operating frequency is 221.484 MHz. So the implemented design provides high operating frequency with more accuracy

V. CONCLUSION

The double precision floating point multiplier supports the LEEE-754 binary interchange format, targeted on a Xilinx Virtex-6 xc6vlx75t-3ff484 FPGA. The design achieved the operating frequency of 414.714 MFLOPs with area of 648 slices. The implemented design is verified with single precision floating point multiplier [6] 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]B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLS1, vol. 2, no. 3, pp. 365-367, 1994.

[2]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.

[3]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.

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

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

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