International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 10, October - 2013

An FPGA Based Double Precision Floating Point Arithmetic Unit using Verilog

[1] KavithaSravanthi, [2] Addula Saikumar

[1]Assistant Professor, MRITS Dundigal Hyderabad, JNTUH affiliated, [2]M.Tech student, MRITS Dundigal Hyderabad, JNTUH affiliated

Abstract

Floating point unit (FPU) addition, subtraction, multiplication and division are widely used in large set of scientific, commerce, financial and in signal processing computation. A high speed floating point double precision adder/subtractor, multiplier and divider are implemented on a Virtex-7 Fpga. In addition /subtractor unit, the proposed designs are compliant with IEEE-754 format and handles overflow, underflow, rounding and various exception conditions. The proposed FPU designs have achieved the operating frequencies of 371.858 MHz while sequential execution of all the operations with a selected inputs given through a test bench.All the modules are realized and validated using Verilog simulation in the Model sim and synthesized using Xilinx 14.1 ISE software.

Keywords- Double Precision, Floating point, IEEE-754, adder/subtractor, multiplier, divider, FPGA, Virtex-7

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).

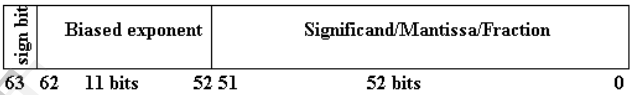

Figure 1. IEEE-754 double precision floating point format

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

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

Floating point implementation on FPGAs has been the interest of many researchers. In [2], an IEEE-754 single precision pipelined floating point multiplier is implemented on multiple FPGAs (4 Actel A1280). Nabeelshirazi, Walters, and peter Athanas implemented custom 16/18 bit three stage pipelined floating point multiplier, that doesn't support rounding modes [3]. L.Louca, T.A. Cook, W.H. Johnson [4] 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 [5], 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 [6] is implemented using the primitives of Xilinx Virtex 7 FPGA. The design achieved the operating frequency of

100MHz with a latency of 4 clock cycles. Mohamed Al-Ashrafy, Ashraf Salem, and WagdyAnis [7] implemented an efficient IEEE-754 single precision floating point multiplier and targeted for Xilinx Virtex-7 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 multiplier presented here is based on IEEE-754 binary floating point standard. We have designed a high speed Arithmetic Floating point unit (FPU) which achieves the rounding modes even for division operation also using Verilog language and ported on Xilinx Virtex-7 FPGA. It operates at a very high frequencies of 371.858 MHz for all operations in a sequence and occupies 4205 slice registers. It handles the overflow, underflow cases and rounding mode.

2. Implementation of Double precision FP Arithmetic unit

A) Adder/Subtractor

The black box view of double precision floating point Adder/subtractor is shown in figure (2) and (3) respectively. The input operands are separated into their sign, mantissa and exponent components. This module has input 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_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 with the smaller exponent can be right shifted before performing the addition. If the exponents are equal, the mantissa are added without shifting. The inter-connection of sub-modules of double precision floating point adder/subtractor are connected such a way to improvise the FPU speed and reduce latency at rounding and exceptionstages.

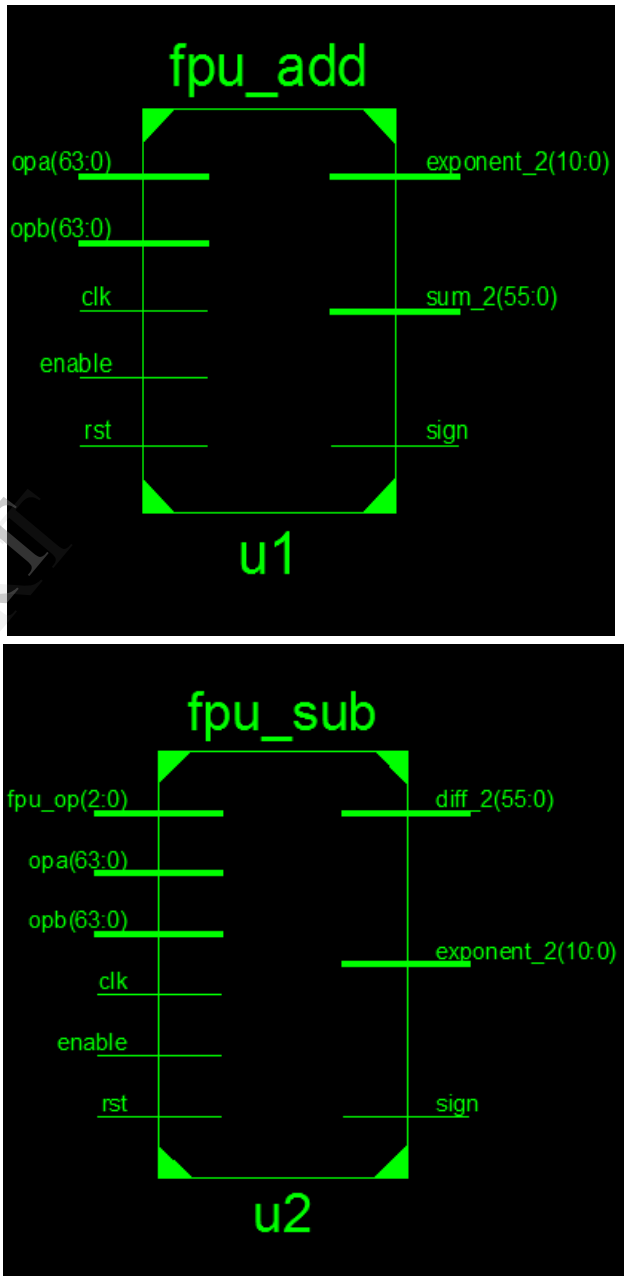

Figure 2, 3. Black box view of FPU's Adder/subtractor

B) *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 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 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 number) are storedin 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 multiplier's 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 bit by 17-bit multiplies. The Xilinx Virtex-7 device contains DSP48E1 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 (fpu_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]*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-7 DSP48E1 slices have been used that follow each 24 bit by 17 bit multiply block. The final 106-bit product is stored in the register (product). The output will be shifted if there is not a '1' in the MSB of product. The number of leading zeros in the 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 bit 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.

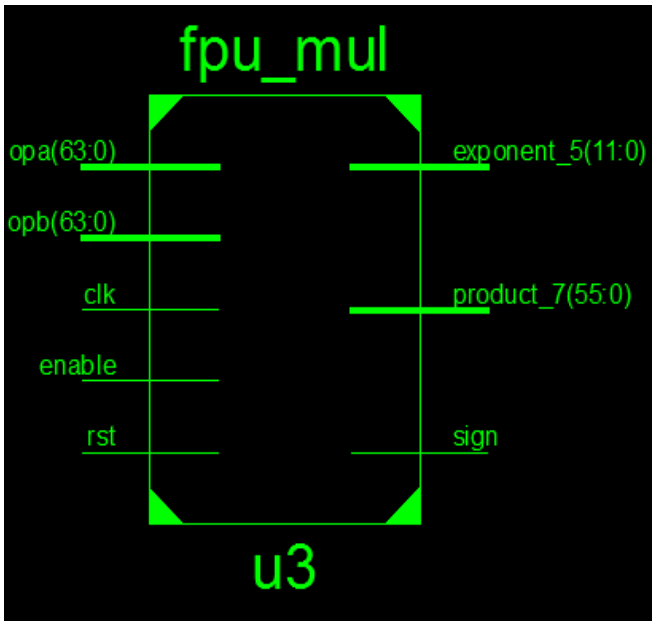

Figure 4. The black box view of FPU's multiplier

C) Divider

The divide operation is performed in the module (fpu-div) and the black box view is shown in the figure (5). The leading '1'(if normalized) and mantissa of operand A is the dividend, and the leading '1'(if normalized) and mantissa of operand B is the divisor. The divide is executed long hand style, with one bit of the quotient calculated each block cycle based on a comparison between the dividend register (dividend_reg) and the divisor register (divisor_reg). If the dividend is greater than the divisor, the quotient bit is '1', and then the divisor is subtracted from the dividend, this difference is shifted one bit to the left, and it becomes the dividend for the next clock cycle. If the dividend is less than the divisor, the dividend is shifted one bit to the left, and then this shifted value becomes the dividend for the next clock cycle.

The exponent for the divide operation is calculated from the exponent fields of operands A and B. The exponent of operand A is added to 1023, and then the exponent of operand B is subtracted from this sum. The result is the exponent value of the output of the divide operation. If the result is less than 0, the quotient will be right shifted by the amount.

The divide operation takes 54 clock cycles to complete, as it takes 1 clock cycle to calculate each of the 54 bits of the quotient. The register (count_out) counts down from 53 to 0, and when it reaches 0, the 54-bit quotient register has its final value. The value that is passed on to the rounding module is stored in the 56-bit register (mantissa_7). The first most significant bit is a '0' to hold a value in case of overflow in the rounding stage, the next bit is the leading '1' for normalized numbers, and the next 52 bits are the mantissas bits. The remaining 2 bits are extra bits rounding purposes. The first extra bit is the last bit that was calculated in the quotient. The quotient has 54 bits, while the mantissa and leading '1' are only 53 bits, so the extra bit is saved and passed on to the rounding stage. The second extra bit is calculated by performing an OR on all of the remainder bits that were left over after the last compare between the dividend and divisor registers.

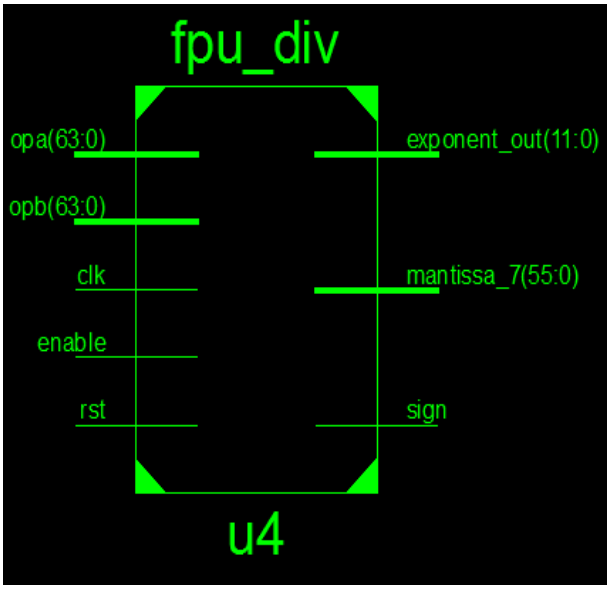

Figure 5. The black box view of FPU's divider

D) Rounding and Exceptions

The IEEE standard specifies four rounding modes such as round to nearest, round to zero, round to positive infinity, and round to negative infinity. Table I shows the rounding modes selected for various bit combinations of mode. Based on the rounding changes to the mantissa corresponding changes has to be made in the exponent part also.

Table I: Rounding modes selected for various bit combinations of mode

Bit combination	**Rounding mode**
00	Round to 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 the individual output signals of underflow, overflow, inexact, exception, and invalid will be asserted if the conditions for each case exist.

3. Results

The double precision floating point Arithmetic unit's design was simulated in modelsim and synthesized using Xilinx ISE 14.1 which was mapped on to the Virtex-7 FPGA. The simulation results of 64-bit floating point double precision Arithmetic unit (FPU) are shown in figure 6. The 'opa' and 'opb' are the inputs and 'out' is the output.

Table II shows the device utilization for implementing the circuit on Virtex-7 FPGA. Table III shows the timing summary of double precision floating point Arithmetic unit (FPU). Table IV shows the area and operating frequency of double precision floating point Arithmetic unit.

Table II: Device utilization summary (XC7vx330t-3ffg1157)

Logic utilization	**Used / Available**
Number of slice registers	4205 / 408k
Number of slice LUTs	6116 / 204k
Number of fully used LUT-FF pairs	2895 / 7426
Number of bonded IOBs	206 / 600
Number of BUFG / BUFGCTRLs	2 / 32
Number of DSP48E1s	9 / 1120

Table III:The Timing summary

parameter	**Adder / subtractor**	**Multiplier**	**Divider**
Minimum period(ns)	2.749	2.411	2.209
Maximum frequency(MHz)	363.769 (for this operation)	414.714 (for this operation)	452.694 (for this operation)

Table IV:Area and operating frequency of FPU

parameter	**value**
area	1628
Operating frequency	371.858 (for all operations in sequence)

4. Conclusion

The double precision floating point adder/subtractor, multiplier and divider supports the IEEE 754 binary interchange format, targeted on a Xilinx Virtex-7 XC7vx330t-3ffg1157 FPGA. The designs achieved the operating frequencies of 363.76MHz, 414.714MHz and 452.694MHz with an area of 660, 648 and 841 slices

respectively. These designs handles the overflow, underflow, rounding mode and various exception conditions.

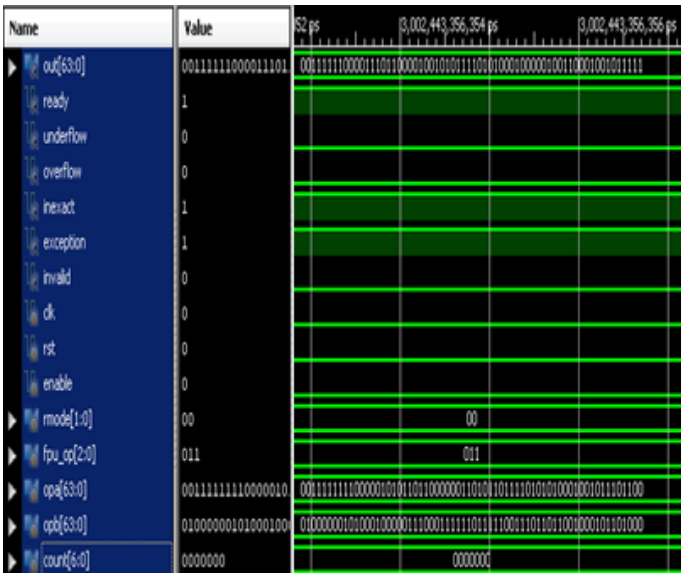

Figure 6. The simulation results of double precision floating point Arithmetic unit

5. References

[1]An FPGA Based High Speed IEEE-754 Double Precision Floating point multiplier using Verilog by A.P.Ramesh, AVN Tilak and AM Prasad.

[2] 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.

[3] 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.

[4] 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.

[5] A Jaaenicke and W.Luk,'Parameterized Floating point Arithmetic on FPGAs", proc of IEEE [CASSP, 2001, vol 2, pp.897-900.

[6] B. Lee and N Burgess,"Parameterisable Floating point operations on FPGA," conference on signals, systems, and computers, 2002.

[7] Mohamed Al-Asrafy, Asraf Salem, WagdyAnis, "An Efficient Implementation of Floating Point Multiplier', Saudi International Electronics, Communications and Photonics Conference (SIECPC), pp. 1-5, 24-26 April 2011.