

Design and Implementation of Area Efficient Single Precision Floating Point Unit

Onkar Singh¹, Kanika Sharma²

¹(Department of ECE, Arni University, HP, India)

²(Department of ECE, NITTTR Chandigarh, India)

Abstract: A floating-point unit (FPU) is a part of a computer system specially designed to carry out operations on floating point numbers. Floating point representation can support a much wider range of values than fixed point representation. This paper presents high speed FPGA implementation of a floating point arithmetic unit which can perform addition, subtraction, multiplication, division functions on 32-bit operands that use the IEEE 754-2008 standard. The amount of hardware used in the single architecture is found to be less than the sum of hardware required in individual floating point units. All four individual units are implemented using VHDL and than one single unit is designed named floating point unit which can perform all the four operations. The code is dumped into vertex-5 FPGA.

Keywords: Exceptions, Exponent, Floating Point Number, Normalization, Precision, Simulation, Synthesis.

I. INTRODUCTION

An arithmetic unit (AU) is the part of a computer processor (CPU) that carries out arithmetic operations on the operands in computer instruction words. Generally arithmetic unit (AU) performs arithmetic operations like addition, subtraction, multiplication and division. Some processors contain more than one AU - for example, one for fixed-point operations and another for floating-point operations. To represent very large or small values, large range is required as the integer representation is no longer appropriate. These values can be represented using the IEEE-754 standard based floating point representation. Typical operations are addition, subtraction, multiplication and division. In most modern general purpose computer architectures, one or more FPUs are integrated with the CPU; however many embedded processors, especially older designs, do not have hardware support for floating-point operations. Almost every language has a floating-point data type; computers from PC's to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow.

The IEEE 754 is a floating point standard established by IEEE in 1985. It contains two representations for floating-point numbers, the IEEE single precision format and the IEEE double precision format.

1.1 IEEE Single Precision Format: The IEEE single precision format uses 32 bits for representing a floating point number, divided into three subfields, as illustrated in figure 1.1

S	Exponent	Fraction
1 bit	8 bits	23 bits

Figure 1.1: IEEE single precision floating point format

1.2 IEEE Double Precision Format: The IEEE double precision format uses 64 bits for representing a floating point number, as illustrated in figure 1.2

S	Exponent	Fraction
1 bit	11 bits	52 bits

Figure 1.2: IEEE double precision floating-point format

II. Implementation of Floating Point Arithmetic Unit:

The block diagram of the proposed floating point arithmetic unit is given in figure 2.1. The unit supports four arithmetic operations: Add, Subtract, Multiply and Divide.

All arithmetic operations have three stages:

Pre-normalize: The operands are transformed into formats that makes them easy and efficient to handle internally.

Arithmetic core: The basic arithmetic operations (addition, subtraction, multiplication and division) are done here.

Post-normalize: The result will be normalized if possible and then transformed into the format specified by the IEEE standard.

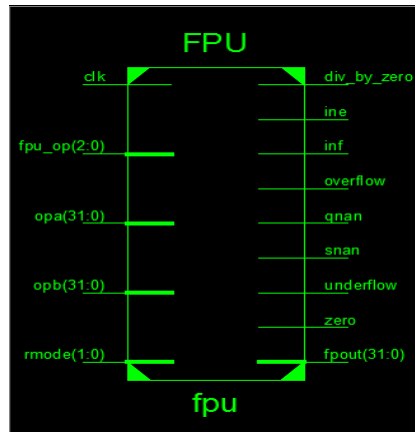


Figure 2.1:Block diagram of Floating Point Unit

The unit has following inputs:

1. Two 32-bit operands (opa, opb)
2. One 3-bit operation code (000 = add, 001 = subtract, 010 = multiply, 011 divide)
3. Rounding modes

00=Round to nearest even: This is the standard default rounding. The value is rounded up or down to the nearest infinitely precise result. If the value is exactly halfway between two infinitely precise results, then it should be rounded up to the nearest infinitely precise even.

01=Round-to-Zero: Basically in this mode the number will not be rounded. The excess bits will simply get truncated, e.g. 3.47 will be truncated to 3.5

10=Round-Up: In this mode the number will be rounded up towards $+\infty$, e.g. 5.2 will be rounded to 6, while -4.2 to -4

11=Round-Down: The opposite of round-up, the number will be rounded up towards $-\infty$, e.g. 5.2 will be rounded to 5, while -4.2 to -5)

4. Clock (Global)

The unit has following outputs:

1. 32-bit output (fpu_out)
2. Five Exceptions

Invalid: Operation are like square root of a negative number, returning of NaN by default, etc., output of which does not exist.

Division by zero: It is an operation on a finite operand which gives an exact infinite result for e.g., $1/0$ or $\log(0)$ that returns positive or negative infinity by default.

Overflow: It occurs when an operation results a very large number that can't be represented correctly i.e. which returns \pm infinity by default (for round-to-nearest mode).

Underflow: It occurs when an operation results very small i.e. outside the normal range and inexact by default.

Inexact: It occurs whenever the result of an arithmetic operation is not exact due to the restricted exponent or precision range.

III. Blocks of Floating Point Arithmetic Unit are:

3.1 Floating Point Adder/Subtractor:

Instead of designing two identical floating point adders, the fused floating point add-subtract unit share the common logic to generate the sum and difference simultaneously. Therefore it saves more area and power consumption compared to a discrete floating point add-subtract unit. The floating point adder/subtractor is a unit which perform both addition and subtraction function.

Two floating point numbers are added as shown.

$$(F_1 * 2^{E_1}) + (F_2 * 2^{E_2}) = F * 2^E$$

Two floating point numbers are subtracted as shown.

$$(F_1 * 2^{E_1}) - (F_2 * 2^{E_2}) = F * 2^E$$

In order to add/Subtract two fractions, the associated exponents must be equal. Thus, if the exponents E_1 and E_2 are different, we must unnormalize one of the fractions and adjust the exponents accordingly. The smaller number is the one that should adjusted so that if significant digits are lost, the effect is not significant

The steps required to carry out floating point addition/Subtraction are as follows

1. Compare exponents. If the exponents are not equal, shift the fraction with the smaller exponent right and add

- 1 to its exponent; repeat until the exponents are equal.
2. Add/Subtract the fractions.
3. If the result is 0, set the exponents to the appropriate representation for 0 and exit.
4. If fraction overflow occurs, shift right and add 1 to the exponent to correct the overflow.
5. If the fraction is unnormalized, shift left and subtracts 1 from the exponent until the fraction is normalized.
6. Check for exponent overflow. Set overflow indicator, if necessary.
7. Round to the appropriate number of bits.

3.2 Floating Point Multiplier:

In this section, the design of multiplier for floating point numbers is proposed.

Given two floating point numbers, the product is
 $(F1 * 2^{E1}) * (F2 * 2^{E2}) = (F1 * F2) * 2^{(E1+E2)} = F * 2^E$

The fraction part of the product is the product of the fractions, and exponent part of the product is the sum of the exponents.

The general procedure for performing floating point multiplication is the following:

1. Add the exponents
2. Multiply the two fractions
 - 2.1 If the product is zero, adjust the representation to the proper representation for zero.
 - a) If the product fraction is too big, normalize by shifting it right and incrementing the exponent.
 - b) If the product fraction is too small, normalize by shifting left and decrementing the exponent.
3. If an exponent underflow or overflow occurs, generate an exception or error indicator.
4. Round to the appropriate number of bits. If rounding resulted in loss of normalization, go to step 3 again.

3.3 Floating Point Division:

In this section, divider for floating point numbers is designed.

Given two floating point numbers, the division is
 $(F1 * 2^{E1}) / (F2 * 2^{E2}) = (F1 / F2) * 2^{(E1-E2)} = F * 2^E$

The general procedure for performing floating point division is the following:

1. Left shift divisor by the no. of bits and right shift dividend by no. of bits.
2. Compare the divisor with the dividend.
3. If divisor is greater than dividend set the corresponding quotient bit to zero.
4. If divisor is less than dividend subtract the divisor from the dividend and place the result in the divisor place, and put one in quotient position.
5. After each comparison right shift divisor by one position.
6. Repeat the above steps by the number of bits time.
7. The number in the dividend place gives remainder and quotient place gives quotient.

IV. Synthesis Report

These are the final results which are obtained in the synthesis report when we are going to synthesis VHDL code of floating point unit on Virtex 5. Table 1 shows the device utilization summary for Floating point unit. The parameters such as number of slices registers, number of slice flip flop, GCLKs etc are outline in the synthesis report are as follows.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	400	19,200	2%
Number used as Flip Flops	400		
Number of Slice LUTs	2,235	19,200	11%
Number used as logic	2,226	19,200	11%
Number used as Memory	8	5,120	1%
Number used as Shift Register	8		
Number of occupied Slices	870	4,800	18%
Number of LUT Flip Flop pairs used	2,336		
Number of fully used LUT-FF pairs	299	2,336	12%
Number of unique control sets	35		
Number of bonded IOBs	110	220	50%
Number of BUFG/BUFGCTRLs	1	32	3%
Number of DSP48Es	2	32	6%
Average Fan-out of Non-Clock Nets	4.53		

Table 1 Device utilization summary for single precision floating point unit

V. Simulation Result

The simulation results of single precision floating point unit (Addition, Subtraction, Multiplication and Division) is shown in figures 5.1, 5.2, 5.3 and 5.4 respectively.

In the waveforms clock defines the applied frequency to the signals. Fpu_op defines the operation to be performed that is 0=addition,1=subtraction, 2=multiplication and 3=division. Opa1 and Opa2 defines the input operand one and input operand two respectively. The r_mode signal defines the various rounding modes (00=Round to nearest even, 01=Round-to-Zero, 10=Round-Up, 11=Round-Down. Fpu_out defines the final output of the signals.

5.1 Simulation Result of single precision floating point addition- The single precision addition operation has been performed for the inputs, Opa1=245.0 and Opa2=40.0 shown in figure 5.1 for which result has been obtained as fpout=285.0

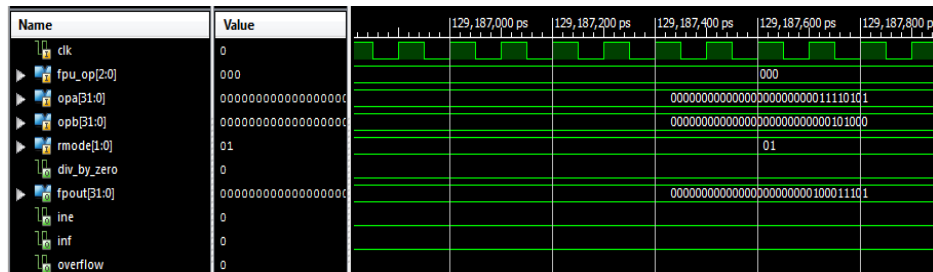


Figure 5.1: Single precision floating point addition

5.2 Simulation result of single precision floating point subtraction- The single precision subtraction operation has been performed for the inputs, Opa1=245.0 and Opa2=40.0 shown in figure 5.2 for which result has been obtained as fpout=205.0

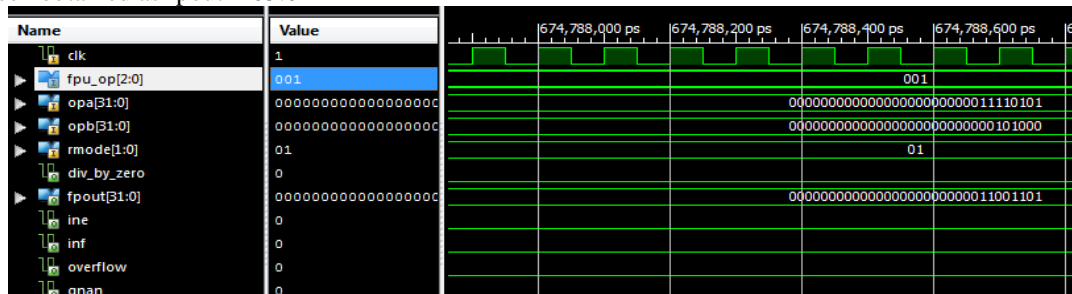


Figure 5.2: Single precision floating point subtraction

5.3 Simulation result of single precision floating point multiplication- The single precision multiplication operation has been performed for the inputs, Opa1=05.0 and Opa2=02.0 shown in figure 5.3 for which result has been obtained as fpout=10.0

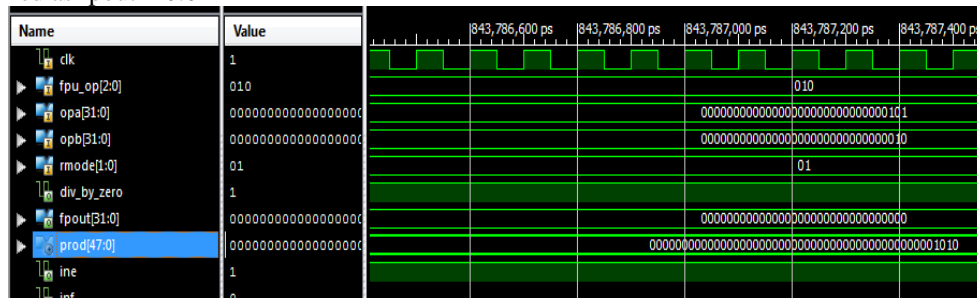


Figure 5.3: Single precision floating point multiplication

5.4 Simulation result of single precision floating point division- The single precision division operation has been performed for the inputs, Opa1=245.0 and Opa2=40.0 shown in figure 5.4 for which result has been obtained as fpout=6.125

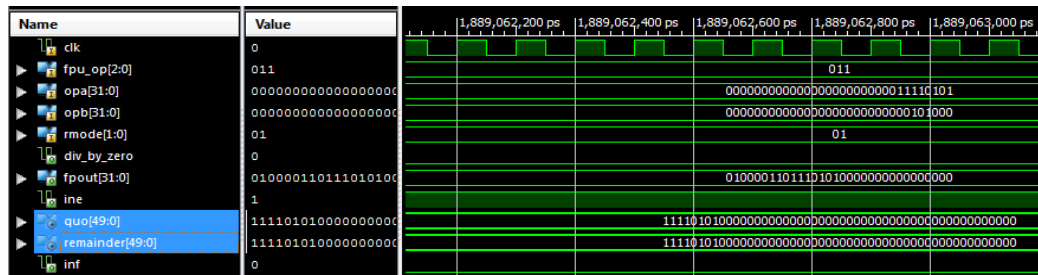


Figure 5.3: Single precision floating point division

VI. Conclusion And Future Work

This paper presents the implementation of single precision floating point unit which is able to perform the four mathematical operations (Addition, Subtraction, Multiplication and Division). The whole design is captured in VHDL and simulated, placed and routed on a Vertex 5 FPGA from Xilinx. The proposed design of the single precision floating point unit required less hardware than the previous designs. The amount of hardware used in the single architecture is found to be less than the sum of hardware required in individual floating point units. For future work the whole design can be implemented on vertex-6 FPGA and also other mathematical units such as exponential, logarithmic can be designed.

References

- [1] Deepa Saini , Bijender M'dia "Floating Point Unit Implementation on FPGA" International Journal Of Computational Engineering Research(IJCER), Vol. 2 , pp.972-976, Issue No.3, May-June 2012
- [2] Karan Ghmber, Sharmelle Thangjam "Performance Analysis of Floating Point Adder using VHDL on Reconfigurable Hardware" International Journal of Computer Application, Vol.46-No 9, May 2012
- [3] Ushasree G, R Dhanabal, Sarat Kumar Sahoo "Implementation of a High Speed Single Precision Floating Point Unit using Verilog" International Journal of Computer Applications National conference on VSLI and Embedded systems, pp.32-36, 2013
- [4] Dhiraj Sangwan , Mahesh K. Yadav "Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic" International Journal of Electronics Engineering, 2(1), pp. 197-203, 2010
- [5] Tarek Ould Bachir, Jean-Pierre David "Performing Floating-Point Accumulation on a modern FPGA in Single and Double Precision" 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, pp.105-108, 2010
- [6] Sateesh Reddy, Vinit T Kanojia "Unified Reconfigurable Floating-Point Pipelined Architecture" International Journal of Advanced Engineering Sciences and Technologies, Vol No. 7, Issue No. 2, pp. 271 – 275, 2011
- [7] Sameh Galal, Mark Horowitz "Energy-Efficient Floating-Point Unit Design" IEEE transactions on computers, Vol. 60, No. 7, pp. 913-922, July 2011
- [8] Rathindra Nath Giri, M.K.Pandit "Pipelined Floating-Point Arithmetic Unit (FPU) for Advanced Computing Systems using FPGA" International Journal of Engineering and Advanced Technology (IJEAT), Volume-1, Issue-4, pp. 168-174, April 2012
- [9] Shrivastava Purnima, Tiwari Mukesh, Singh Jaikaran and Rathore Sanjay "VHDL Environment for Floating point Arithmetic Logic Unit - ALU Design and Simulation" Research Journal of Engineering Sciences, Vol. 1(2), pp.1-6, August -2012
- [10] Hwa-Joon Oh, Silvia M. Mueller, Christian Jacobi, Kevin D. Tran, Scott R. Cottier "A Fully Pipelined Single-Precision Floating-Point Unit in the Synergistic Processor Element of a CELL Processor" IEEE Journal of Solid-State Circuits, Vol. 41, No. 4, pp. 759-771, April 2006
- [11] Jongwook Sohn, Earl E. Swartzlander "Improved Architectures for a Fused Floating Point Add-Subtract Unit" IEEE Transactions on Circuits and Systems—I: regular papers, Vol. 59, No. 10, pp. 2285-2291, October 2012
- [12] Per Karlstrom, Andreas Ehliar, Dake Liu "High Performance, Low Latency FPGA based Floating Point Adder and Multiplier Units in a Virtex 4", 24th Norchip Conference, pp. 31 – 34, Nov. 2006.