Design and Implementation of Double Precision Floating Point Unit

Chaitanya A. Kshirsagar
Department of Electronics
Y.C.C.E, Nagpur, Maharashtra, India

Prof.P.M. Palsodkar
Department of Electronics
Y.C.C.E, Nagpur, Maharashtra, India

Abstract- Arithmetic circuits form an important class of circuits in digital systems. With the remarkable progress in the very large scale integration (VLSI) circuit technology, many complex circuits, unthinkable yesterday have become easily realizable today. Algorithms that seemed impossible to implement now have attractive implementation possibilities for the future. In this paper an arithmetic unit based on IEEE standard for floating point numbers has been implemented on FPGA Board. Here FPU follows IEEE double precision format. The arithmetic unit implemented has a 64-bit processing unit which allows various arithmetic operations such as, Addition, Subtraction, Multiplication and Division on floating point numbers. Each operation can be selected by a particular operation code. We see that the overhead for double precision is less than that for single precision. The unit comprises of rounding and exception unit as specified in format. The FPU design achieved the operating frequency of 293.53MHz.

Keywords: Double precision, floating point, FPGA, IEEE-754 Standard, rounding unit.

I. INTRODUCTION

Digital arithmetic operations are very important in the design of digital processors and application-specific systems. Arithmetic circuits form an important class of circuits in digital systems. With the remarkable progress in the very large scale integration (VLSI) circuit technology, many complex circuits, unthinkable yesterday have become easily realizable today. Algorithms that seemed impossible to implement now have attractive implementation possibilities for the future. This means that not only the conventional computer arithmetic methods, but also the unconventional ones are worth investigation in new designs. In the digital signal processing, applications involving more accuracy and range floating point representation is used. One of the applications is audio processing.

A floating-point arithmetic and logic unit, generally termed a Floating Point Unit is a part of a computer system specially designed to carry out operations on floating point numbers. Floating point describes a system for representing numbers that would be too long or too short to be represented as integer. Floating point representation retained its resolution and accuracy compared to the fixed point number system representation.

The standard provides for many closely related formats, differing in only a few details, like single precision, double precision, double extended. The double precision format includes 64 bits, contains 1 sign bit, 11 bits for exponential and 52 bits wide mantissa.[1] Generally represented as follows:-

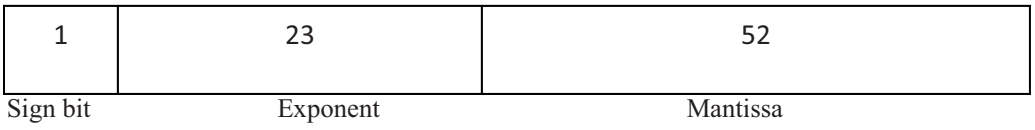

Figure 1. IEEE 754 Floating point standard

The value of representation is obtained from the equation;

$$value = (-1)^{s} \times (2)^{E-1023} \times (1.Mantissa) \quad \ldots (1)$$

The range of the magnitude of floating point number is from 1×2^{-1022} to $(2-2^{52})\times2^{1023}$ which can be represented in decimal format in range 2.23×10^{-308} to 1.8×10^{308} .

In this paper the two operands i.e. operand a and operand b are 64 bit wide, to choose which operation to be performed fpuop register gives information, similarly for rounding modes rmode is used to provide information that which mode among the four to be used. Standard rounding mode is round to nearest. Here we use status signals to represent the operation they are overflow, underflow, inexact, exception and invalid.

II. IMPLEMENTATION OF FLOATING POINT UNIT

The two operands are already in double precision format.

All arithmetic operations have these 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 are done here.
- **Post-normalize:** the result will be normalized if possible (leading bit before decimal point will be 1, if normalized) and then transformed into the format specified by the IEEE standard.

A. Addition and Subtraction

The conventional floating-point addition algorithm consists of five stages - exponent difference, pre-alignment, addition, normalization and rounding. Given floating-point numbers X1 = (s1, e1, f1) and X2 = (s2, e2, f2), the stages for computing X1 + X2 are described as follows:

- Find exponent difference d = e1-e2. If e1 < e2, swap position of mantissas. Set larger exponent as tentative exponent of result.
- Pre-align mantissas by shifting smaller mantissa right by d bits.
- Add or subtract mantissas to get tentative result for mantissa.
- Normalization. If there are leading-zeros in the tentative result, shift result left and decrement exponent by the number of leading zeros. If tentative result overflows, shift right and increment exponent by 1-bit.
- Round mantissa result. If it overflows due to rounding, shift right and increment exponent by 1-bit.

If the exponents differ by more than 53, the smaller number will be shifted right entirely out of the mantissa field, producing a zero mantissa. The sum will then equal the larger number.

When adding numbers of opposite sign, cancellation may occur, resulting in a sum which is arbitrarily small, or even zero if the numbers are equal in magnitude. Normalization in this case may require shifting by the total number of bits in the mantissa, resulting in a loss of accuracy.

Floating point subtraction is achieved simply by inverting the sign bit and performing addition of signed mantissas as outlined above.

B. Multiplication

In discussing floating-point multiplication, by complies with the IEEE 754 Standard, the two mantissas are to be multiplied, and the two exponents are to be added. The sign logic is a simple XOR. In order to perform floating-point multiplication, a simple algorithm is realized:

- Add the exponents and subtract 1023 (bias).
- Multiply the mantissas and determine the sign of the result.
- Normalize the resulting value, if necessary.

Overflow occurs when the sum of the exponents exceeds 1023, the largest value which is defined in bias -1023 exponent representation. When this occurs, the exponent is set to 1024 (E = 2047) and the mantissa is set to zero indicating + or - infinity.

Underflow occurs when the sum of the exponents is more negative than -1022, the most negative value which is defined in bias -1022 exponent representation.

Rounding occurs in floating point multiplication when the mantissa of the product is reduced from 104 bits to 52 bits.

C. Division

The division was done serially using the basic algorithm which is division through multiple subtractions. Since divisions are not needed as often as multiplications. Divisions can be done also through multiplication. It was implemented as serial and in the process saving some hardware area. The conventional floating-point division algorithm consists of five stages – counting leading zeroes in both numbers, shifting left, division, rounding and. Normalization.

Count leading zeroes in both floating point numbers.
- Shift left fractional bits of both floating point numbers according to number of zeroes.
- Divide the fractional bits. Sign of result is calculated from xoring sign of two operand.
- Exponent of result is calculated by equation:

e0 = eA – eB + bias (127) -zA + zB

- Round the fraction and normalize it if required.

D. Rounding

In floating point arithmetic, rounding errors occur as a result of the limited precision of the mantissa. For example, consider the average of two floating point numbers with identical exponents, but mantissas which differ by 1. Although the mathematical operation is well-defined and the result is within the range of represent able numbers, the average of two adjacent floating point values cannot be represented exactly. In this paper we implemented all four rounding modes.

The IEEE FPS defines four rounding rules for choosing the closest floating point when a rounding error occurs:
– RN - Round to Nearest. Break ties by choosing the least significant bit = 0.
– RZ - Round toward Zero. Same as truncation in sign-magnitude.
– RP - Round toward Positive infinity.
– RM - Round toward Minus infinity. Same as truncation in integer 2's complement arithmetic.

III. SIMULATION RESULTS

The double precision floating point unit designs were simulated in ISim and synthesized using Xilinx ISE 13.2i which are mapped on to Vertex 6 FPGA. The simulation results of 64-bit floating point unit shown in following figures. The ‘opa’ and ‘opb’ are the inputs and ‘out’ is the output.

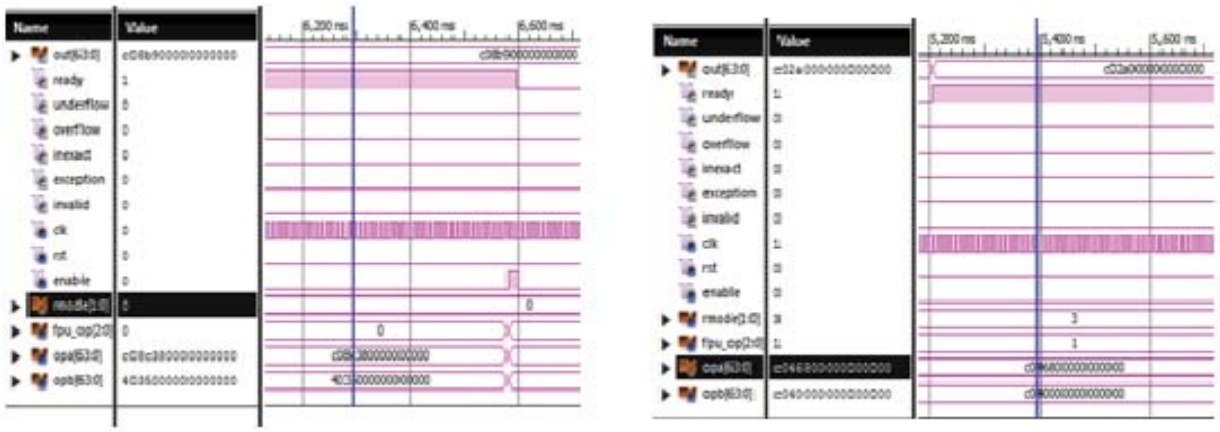

Figure 2. Simulation result of addition and subtraction.

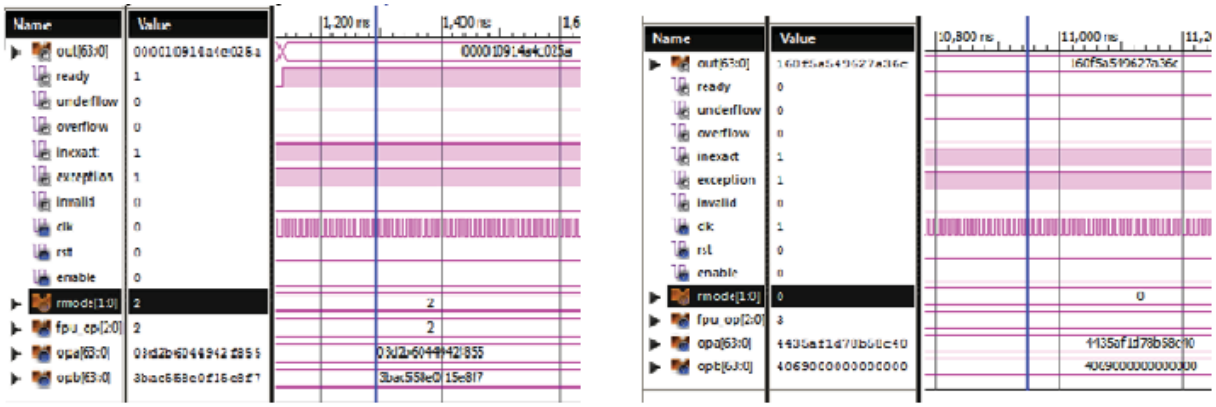

Figure 3. Simulation result of multiplication and division.

Table 1. Device utilization summary

Slice Logic Utilization	Used
Number of Slice Registers (flip flop)	4250
Number of Slice LUTs	6103

Table 2. Timing analysis

Parameter	value
Minimum period (nsec)	3.361
Maximum Frequency (MHz)	297.53

IV.CONCLUSION

Arithmetic unit has been designed to perform four arithmetic operations, addition, subtraction, multiplication, and division floating point numbers. IEEE 754 standard based floating point representation has been used. The unit has been coded in Verilog. Code has been synthesized for the Vertex FPGA board using XILINX ISE.. As compared to the single precision floating point multiplier and Xilinx core, the multiplier design supports double precision, provides high speed and gives more accuracy. The design achieved maximum frequency of 293.53 MHz.

REFERENCES

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, **2008**.
[2] Even,G.,Paul, W., **"On the design of IEEE compliant floating point units**," *Computer Arithnmetic, 1997, Proceedings., 13th IEEE Symposium on* , vol., no., pp.54,63, 6-9 Jul **1997**.
[3] D. Goldberg, "What every computer scientist should know about floating-point arithmetic" pp. 5-48 in ACM Computing Surveys vol. 23-1 **1991**.
[4] Steve Furber, "ARM System-on-Chip Architecture" second edition, seventeenth impression, PP. 170-174, **2013.**