International Journal of Computer System (ISSN: 2394-1065), Volume 02– Issue 04, April, 2015
Available at http://www.ijcsonline.com/

Implementation of 64-Bit Pipelined Floating Point ALU Using Verilog

Megha Sharma, Dr. Gurjit Kaur

School of ICT,
Gautam Buddha University,
Greater Noida, U.P., India

Abstract

A Pipelined double precision floating point arithmetic logic unit (ALU) using verilog design is introduced. The novelty of the ALU is it gives high performance through the pipelining concept. Pipelining is a technique where multiple instruction execution are overlapped. A double precision floating point unit is to provide five arithmetic operations: addition, subtraction, multiplication, division, and square root and in top-down design approach, four arithmetic modules: addition/subtraction, multiplication, division, and square root are combined to form a double precision floating point ALU. Each module is divided into smaller modules. Three bits selection determines which operation takes place at a particular time. The pipelined modules are independent of each other. The modules are realized and validated using verilog simulation in the Questasim SE 10.0b and synthesis using Xlinx ISE Design Suite 13.3.

Keywords: *Floating point, Pipelining, Double precision, ALU*

I. INTRODUCTION

Floating point describes a system for representing numbers that would be too large or too small to be represented as integers. Based on this standard, floating-point representation for digital systems should be platform-independent and data are interchanged freely among different digital systems. Pipeline is one of the popular method to realize high performance computing platform. A pipeline is a technique used in the design of computers and other digital electronic devices to increase their instruction throughput (number of instructions that can be executed in a unit of time).ALU is a block in microprocessor that handles arithmetic operations.

The double precision floating point unit implemented in this paper is a 64-bits processing unit which allows arithmetic operations on a floating point numbers. The floating point unit complies fully with IEEE 754 standard.

II. DESIGN AND METHODS

The main objective of this paper is to implement a double precision floating point unit capable of supporting the five basic operations i.e. addition, subtraction, multiplication, division, and square root. The sub-objectives are to design a 64-bit floating point arithmetic logic unit operating on the IEEE 754 standard. The second sub-objective is to model the behavior of the double precision floating point unit using VERILOG. The specifications for a 64-bit floating point unit design are:

i. Inputs op_a and op_b are of 64-bit binary floating point.
ii. Output op_o_add_sub and op_o_div is of 64-bit for addition, subtraction and division modules respectively.
iii. Output op_o_mul is of 55-bit for multiplication module.
iv. Output op_o_sr is of 21-bit for square root module.
v. Clock pulse is generated at each 100ns.
vi. Pos edge of clock edge and logic 1 of reset is used for initializing both the operands op_a and op_b.
vii. Select signal fpu_op_i of 3-bit is also used in the design to select the desired arithmetic operation as follows:-

TABLE I. Operation Table

FPU op i	Operation
000	Addition
001	Subtraction
010	Multiplication
011	Division
100	Square root

III. FLOATING POINT NUMBERS

The floating point representation is one way to represent real numbers. A floating point number „n" is represented with an exponent „e" and a mantissa „m" as follows [7] :

$$n = b^{e} * m$$

Here, n – Floating point number.

b – Base (i.e. 2).

e – Exponent/Power/Radix.

m – Mantissa/Fraction.

Example- If we choose a number n=17 and base b=2, the following floating point representation of 17 would be:

$$17 = 2^{4} * 1.0625$$

III. NEED TO GO BEYOND INTEGERS

For extremely large values such as distance between sun and Pluto and also for extremely small values such as mass of electron, these types of values cannot be represented using integer, rational, irrational, real, and complex numbers, hence floating point numbers are introduced.

IV. FPU REPRESENTATION

IEEE 754 standard has introduced the concept for writing any value in its floating point form. Basically, a floating point number consist of three parts: sign, exponent, and fraction part.

IEEE 754 standard defines two floating point representation, which is called as-

TABLE III. BIT DIVISION

Single Precision Number	Double Precision Number
SIGN (1-bit)	SIGN (1-bit)
EXPONENT (8-bit)	EXPONENT (11-bit)
FRACTION (23-bit)	FRACTION (52-bit)

Example- Suppose we had to represent a value „17" in the double precision floating point format. Then following is the required floating point representation:

0 00010000 00011001010100000000000

TABLE II. Range For Single & Double precision

	Single precision number	Double precision number
Range for Fraction	1< fraction < 2 – 2^ (-23)	1< fraction < 2 – 2^ (-52)
Range for Exponent	-126 < exponent < 127	-1022 < exponent < 1023

VALUES FOR SIGN

0 – For positive floating point number.

1 – For negative floating point number

V. FLOATING POINT OPERATIONS AND FLOWCHARTS

A. Addition/Subtraction:

[(-1) ^S1 * F1 * 2^E1] + [(-1) ^S2 * F2 * 2^E2]
Suppose E1>E2, then we can write as,
[(-1)^S1 * F1 * 2^E1] + [(-1) ^S2 * F3 * 2^E1].
(Where, F3=F2/2^ (E1-E2))

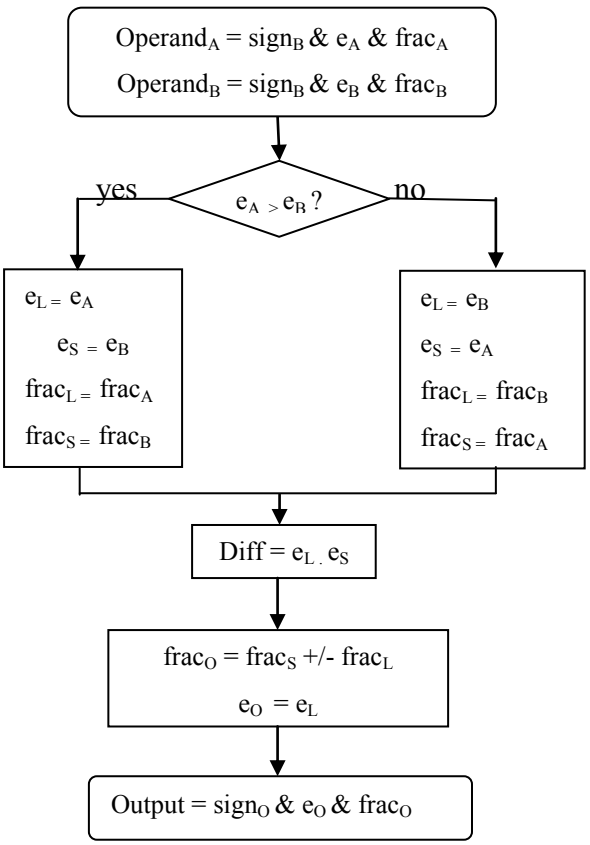

Fig 1. Flow Chart- Add/Sub

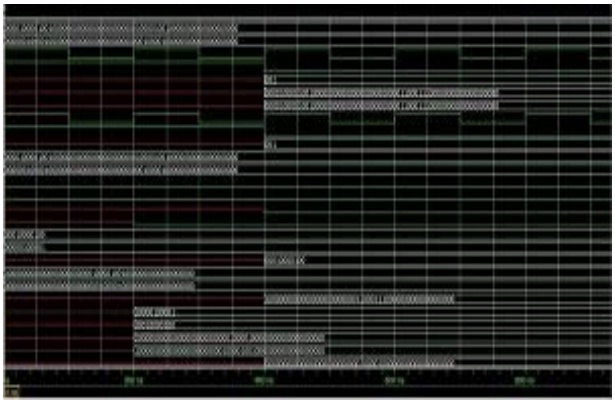

Fig 2. Simulation window- Add/Sub

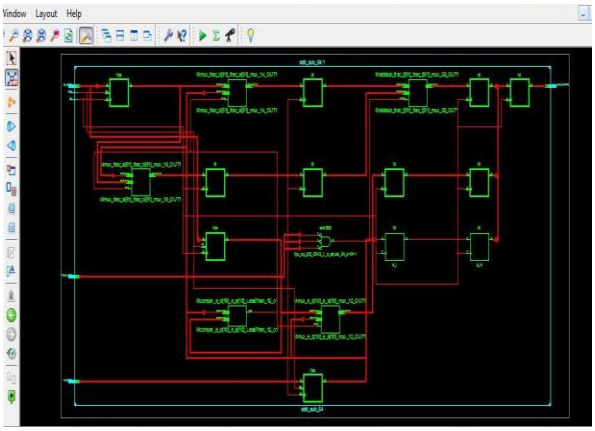

Fig 3. RTL window- Add/Sub Module

In this window Green boxes represents registers in which values are stored and Red lines are interconnects that are used to connect registers.

B. *Multiplication*

$$[(-1)^{S1} * F1 * 2^{E1}] * [(-1)^{S2} * F2 *$$

$$= (-1)^{(S1 \text{ xor } S2)} * (F1 * F2) * 2^{(E1+E2)}.$$

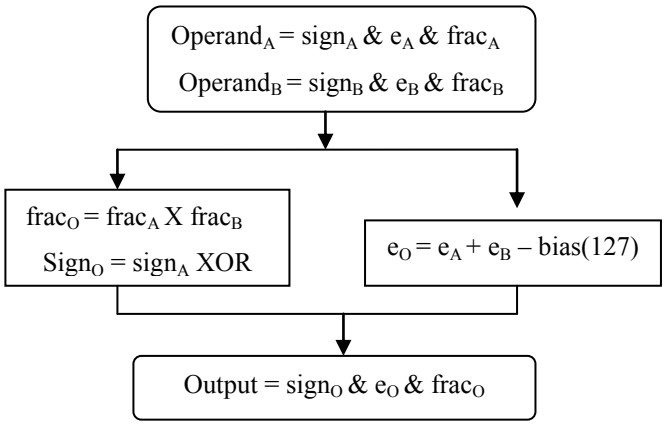

Fig 4. Flow Chart- Multiplication

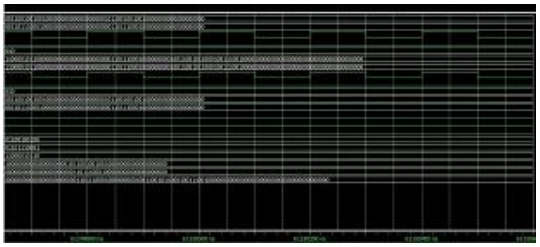

Fig 5. Simulation Window-Multiplication

Verilog works at four values i.e. 0,1,X,Z.In simulation window, red line shows that value can be 0 or 1(simulator dosen"t know whether it is 0 or 1), Green line shows value is either 0 or 1, Blue line shows high-impedance state, Yellow line shows what is the value at particular time interval.

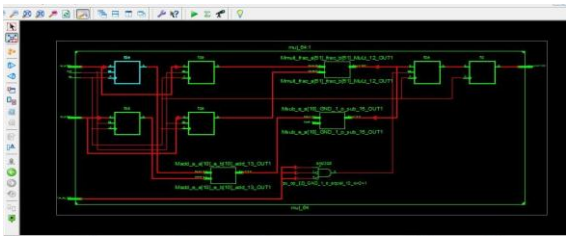

Fig 6. RTL Window-Multiplication Module

In this window Green boxes represents registers in which values are stored and Red lines are interconnects that are used to connect registers.

C. *Division*

$$[(-1)^{S1} * F1 * 2^{E1}] / [(-1)^{S2} * F2 * 2^{E2}]$$

$$= (-1)^{(S1 \text{ xor } S2)} * (F1 / F2) * 2^{(E1-E2)}.$$

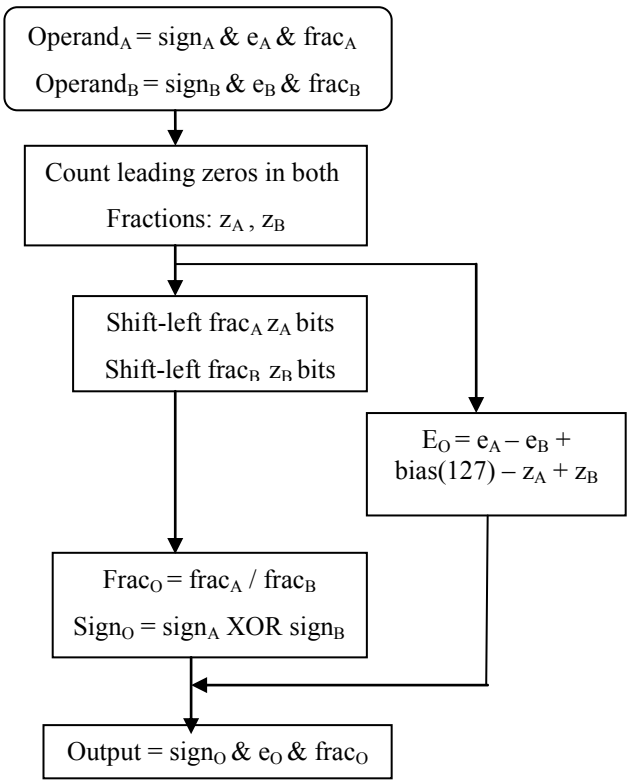

Fig7. Flow Chart-Division

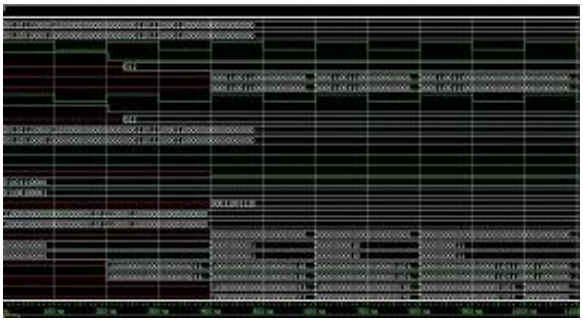

Fig 8. Simulation window-Division

Verilog works at four values i.e. 0,1,X,Z.In simulation window, red line shows that value can be 0 or 1(simulator dosen"t know whether it is 0 or 1), Green line shows value is either 0 or 1, Blue line shows high-impedance state, Yellow line shows what is the value at particular time interval.

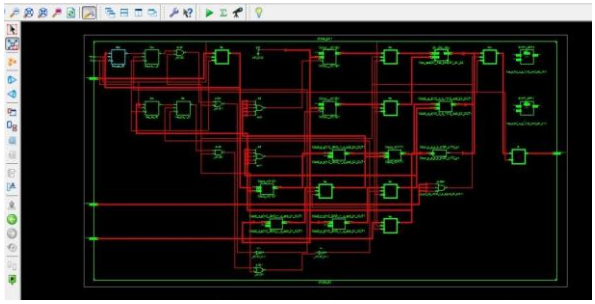

Fig 9. RTL window-Division Module

In this window Green boxes represents registers in which values are stored and Red lines are interconnects that are used to connect registers.

D. Square Root

The non-restoring square root algorithm using non-redundant binary representation is given below.

1. Set q16 = 0, r16 = 0 and then iterate from k = 15 to 0.

2. If rk+1 _ 0, rk = rk+1D2k+1D2k − qk+101, else rk = rk+1D2k+1D2k + qk+111,

3. If rk _ 0, qk = qk+11 (i.e., Qk = 1), else qk = qk+10 (i.e., Qk = 0),

4. Repeat steps 2 and 3, until k = 0. If r0 < 0, r0 = r0 + q01.

Where qk = Q15Q14...Qk has (16 − k) bits, e.g., q0 = Q

= Q15Q14Q13...Q1Q0, and rk has (17 − k) bits, e.g., r0

= R = R16R15R14...R1R0. Notice that rk+1D2k+1D2k means rk+1 × 4 + D2k+1 × 2 + D2k. Similarly, qk+11 means qk+1 × 2 + 1.

The multiplications and additions are not needed. Instead, we use shifts and concatenations by suitable wiring.

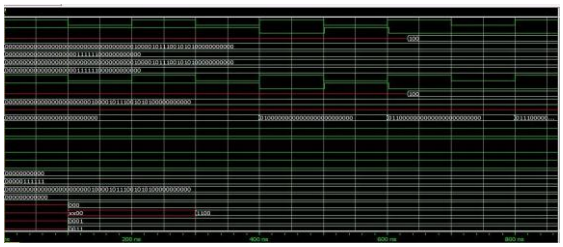

Fig 10. Simulation window-Square root

Verilog works at four values i.e. 0,1,X,Z.In simulation window, red line shows that value can be 0 or 1(simulator dosen"t know whether it is 0 or 1), Green line shows value is either 0 or 1, Blue line shows high-impedance state, Yellow line shows what is the value at particular time interval.

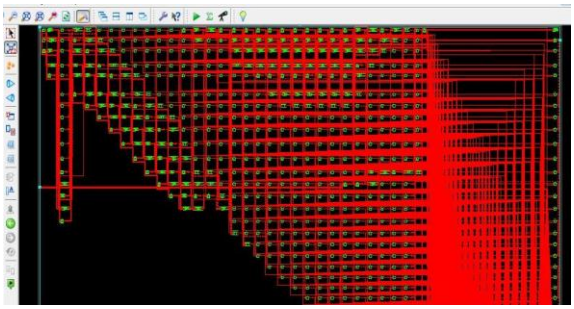

Fig 11. RTL window-Square root

E. Top-Down Model

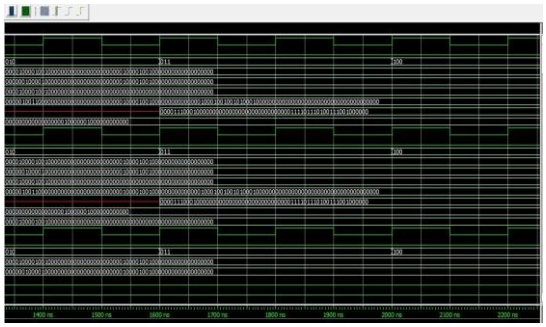

Fig 12. Simulation window-Top-Down model

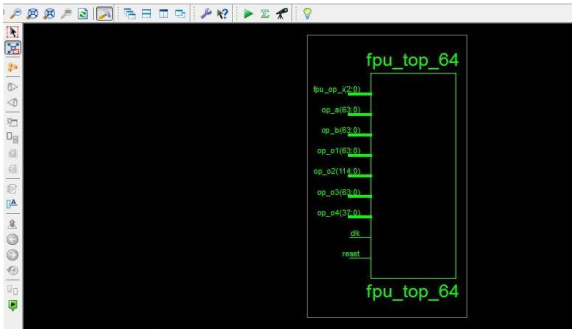

Fig 13. RTL window-Top-Down Module

In this model all the four operations i.e. add/sub, multiplication, division, square root are combined in a single unit or module.

VI. CONCLUSION

In this paper four functionalities: addition/subtraction, multiplication, division and square root are introduced for double precision numbers. And top-down model has also been introduced. Coding in verilog has been done according to the flow-charts shown above. More work can be done regarding power consumption. We can use the concept of clock gating for this. Such that required sub module will perform its operation, and all other sub module will remain off, on applying the values select pins.

REFERENCES

[1] Mamun Bin Ibne Reaz,, Md. Shabiul Islam, Mohd, and S.Sulaiman, "Pipeline Floating Point ALU Design using VHDL", IEEE International Conference on Semiconductor Electronics, Proceedings, ICSE, pp: 204 – 208, (2002).

[2] Rajit Ram Singh, Asish Tiwari, Singh, V.K. ; Tomar, G.S. "VHDL environment for floating point Arithmetic Logic Unit-ALU design and simulation", Proceedings of the International Conference on Communication Systems and Network Technologies, Pages 469-472, (2011).

[3] Shrivastava Purnima, Tiwari Mukesh, Singh Jaikaran, and Rathore Sanjay, "VHDL Environment for Floating Point Arithmatic Logic Unit- ALU Design and Simulation", July (2012).

[4] L.F.Rahman, Md. Mamun, and M.S.Amin, "VHDL Environment for Pipeline Floating Point Arithmatic Logic Unit Design and Simulation", Journal of Applied Sciences Research, 8(1): 611-619, (2012).

[5] Yamin Li, and Wanming Chu, "Implementation of Single Precision Floating Square Root on FPGAs", April (1997).

[6] M. Daumas, and C. Finot, "Division of Floating Point Expansions with an Application to the Computation of the Determinant", June (1999).

[7] ANSIWEE std 754-1985," IEEE standard for binary Floating-point arithmetic", IEEE New York (1985).

[8] Chen S., Mulgeew B. and Grant P.M., "A clustering technique for digital communications channel equalization using radial basis function networks", In IEEE Transactions on Neural Networks, Volume:4 , Issue: 4 , pp: 570 – 590, 1993, DOI: http://dx.doi.org/10.1109/72.238312.