# Speed Optimization of 32 Bit Single Precision Floating Point Multiplier through Pipelining in VHDL

Tajinder Pal Singh*

* Electrical and Electronics Engineering, Faculty of Electrical Engineering, Chitkara University, India

(tajinderpal.singh@chitkarauniversity.edu.in)

‡ Corresponding Author Tajinderpal Singh, Chitkara University, Tel: +91 8872446979,tajinderpal.singh@chitkarauniversity.edu.in

*Received: 03.08.2018 Accepted:28.12.2018*

**Abstract-** The paper is presenting the architectural method for speed optimization of floating point multiplier involves increasing the frequency by implementing pipelines in the design using VHDL language. The whole algorithm of IEEE 754 standard 32 bit single precision floating point multiplier have two pipelining stages, which improve the frequency rate of clock to 422.556 MHz and result the output in 2.367ns. The design also handles the overflow/ underflow cases with normalization for the better accuracy of the result. Xilinx vertex 5 FPGA is targeted for the design and the simulation is done on Xilinx ISE simulator.

**Keywords** Floating Point multiplier; VHDL; Pipelining; Xilinx ISE tool.

## 1. Introduction

This paper explores the effect of pipelining on the performance of the proposed design of Floating point multiplier. Mostly latency in the clock affects highly the design performance. So the branch prediction and fast branch recovery will continue to increase in importance, which is called pipelining. In VHDL pipelining the multiple instructions are runs in parallel. The pipeline is divided in stages. Each stage completes a part of an instruction in parallel. The pipe stages are hooked together in which instructions enter at one end, progress through the stages, and exit at the other end. Hence increase the instruction throughput.

In floating point multiplier, multiplication required the following steps; first to multiply the mantissa or significand of two multiplicand, second is to add the exponent of both numbers and subtract the bais and third is to XORing the sign bit of both numbers. In these steps most of the calculation time is spent in signifcand multiplication, moderate in addition and very less in sign calculation. The pipelining helps to divide the critical path of floating multiplier into small stages which improve the overall performance of the design.

In our design of single precision floating point multiplier we introduce two pipelining stages in the architecture which achieves the frequency rate of 422.556MHz.The two stage pipelining increase the instruction throughput of the proposed design and result the output in 2.367ns. The complete algorithm & pipelining in floating point multiplier is discussed in next sections.

## 2. Operation of Floating point multiplication

Let A & B are two floating point numbers to be multiplied. E is representing Bias Exponent value of A or B. M is Mantissa or fraction value of the numbers and S is representing sign bit. The whole process takes following steps for the operation of multiplication as shown in the flow chart given below in figure 1a.

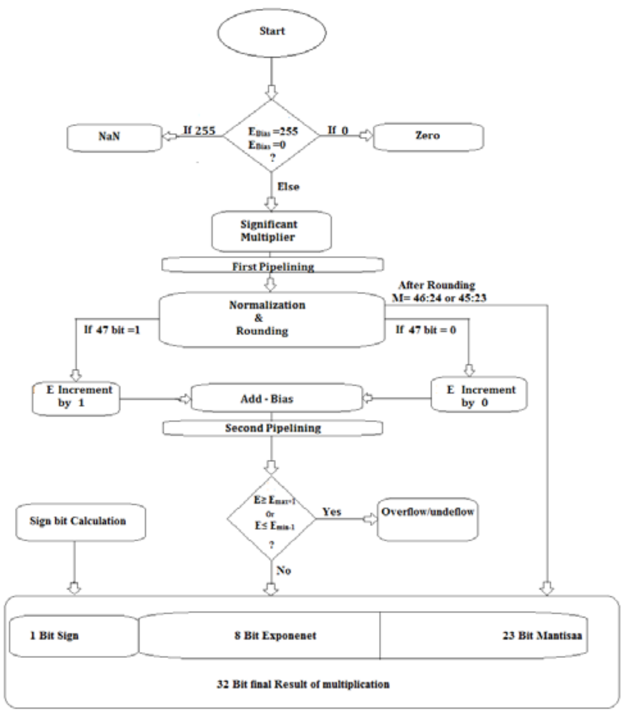


**Fig. 1.** Flow chart for Floating Point Multiplier with pipelining

1. First step in multiply the two floating point numbers is to check whether the both multiplicand is in the range of the format according to IEEE754 standard or not as in figure 2 given below.

The value of bias exponent (E) should lie in between 0 to 254(8 bit only). If not the bias exponent is represented by a special values. For example If the value of bias exponent (Ebias) is more than 254 than its turn to not a number (NaN) on the other hand if the value of bias exponent is less than 1 its turn to zero as shown in table 1.

2. Significand multiplication is the second step of the whole operation, if only both the multiplicands come in the range of IEEE754 standard. This operation is performed on the 23 bit mantissa (or fraction) part of the numbers which gives the 48 bit result in output. The 48 bit result of the mantissa multiplication is than round off to 23 bit so that it will fit in the range of IEEE754 32 bit floating point number format.

With that rounding, normalization is also done to represent the number in scientific notation, which also increases the accuracy of the result.

3. The result of significand multiplication is normalized so that the product will have leading bit one on the left side of decimal point. If the both multiplicands are the normalized numbers so the result will have leading one either on 47 bit or on 46 bit.

• If the leading one is at 46 bit or on the left side of decimal point the product don't require any shift. In the other words we can say there will not any increment in exponent value.

For example 1 10000111 1.001011000 (normalized form)

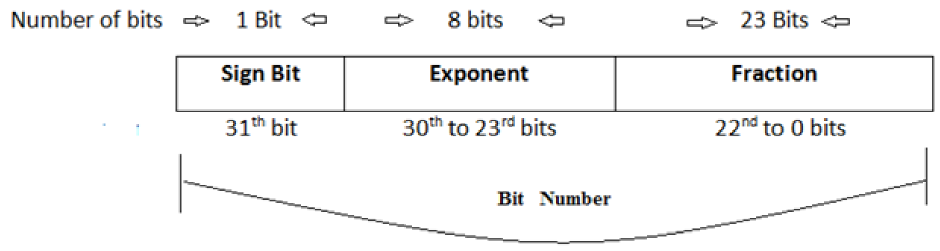


**Fig. 2.** IEEE 754 standard for 32 bit

**Table 1.** Appearance properties of accepted manuscripts

|  | Bias Exponent | Special Values |
|---|---|---|
| Bias Exponent Range (0 to 254) | If $E_{Bias} > 254$ | Not a number(NaN) |
|  | If $E_{Bias} < 1$ | Zero |

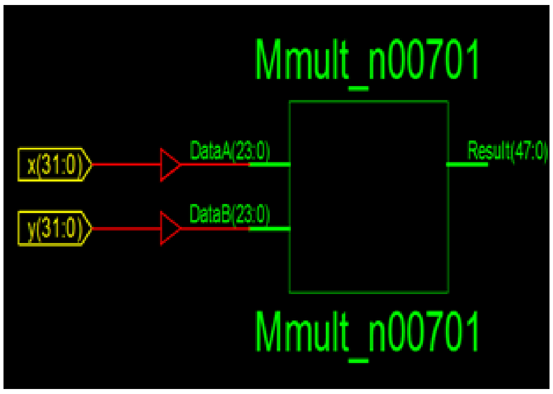


**Fig. 3.** Schematic diagram for significand multiplier.

For example

If the leading one is at 47 bit the product require shift on right which will increment the exponent by one.

For Example 1 10000110 1001011000 (Non normalized form)

If the both multiplicand are not a normalized numbers than for the requirement of normalize the result so that there is a 1 just before the radix point (decimal point) the radix point move one place to the left increments the exponent by 1; moving one place to the right decrements the exponent by 1.

4. Next step is to round off the result obtained from significand multiplication. In rounding the intermediate product obtained from significand multiplication become of 23 bit from 46 bits.

5. Adding the exponent of two numbers is done on 8 bit part of floating point number. After addition the constant value of 127 (bias) is subtracted from the addition result.

6. The result obtained from exponent addition is checked for overflow and underflow. If overflow occurs the result turned to infinity and when underflow occurs result turned to zero. The overflow and underflow can be compensated only under following situations

**Table 2.** Overflow and underflow detection with effect of normalization

| Exponent Value ($E_{result}$) | Over flow | Under flow | Compensated or not in normalization |
|---|---|---|---|
| $E_{result} \geq 255$ | Yes | No | No |
| $E_{result} = 0$ | No | Yes | Yes |
| $E_{result} < 0$ | No | Yes | No |

## 3. Pipelining in Floating point multiplier

Time performance enhancement is the requirement of the design. It has been achieved through the VHDL pipelining. Pipelining in the proposed design is accomplished at the two different stages. These two stages pipelining involving parallel processing of the data improve the frequency rate of the clock. Hence decrease the overall time consumption of floating point multiplier and gives better speed.

The pipelining comes at two different locations is discussed below in detail in figure 3.

- ➢ One after the significand multiplier and before the exponent adder & rounding.
- ➢ Second after the exponent addition and before the overflow/underflow detection & final result of multiplication

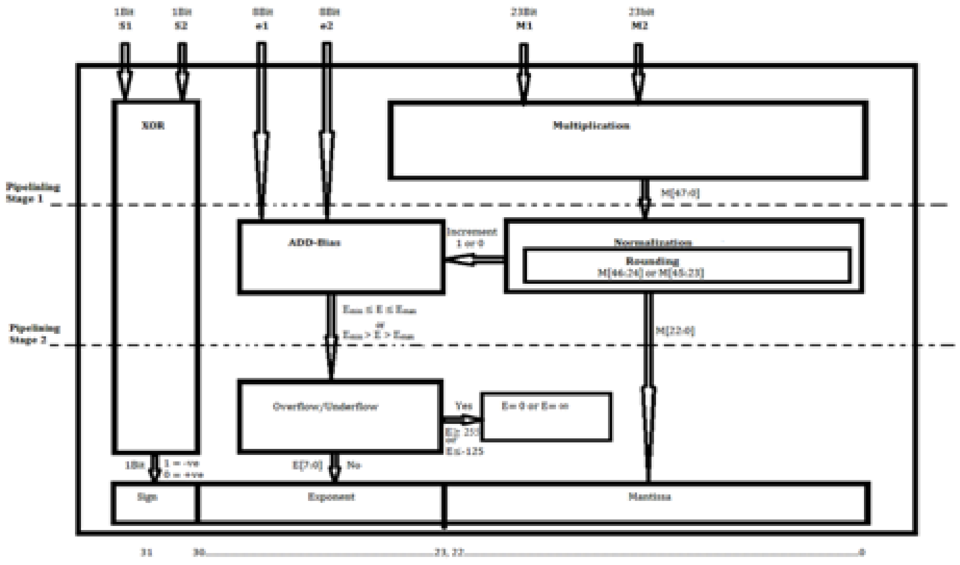
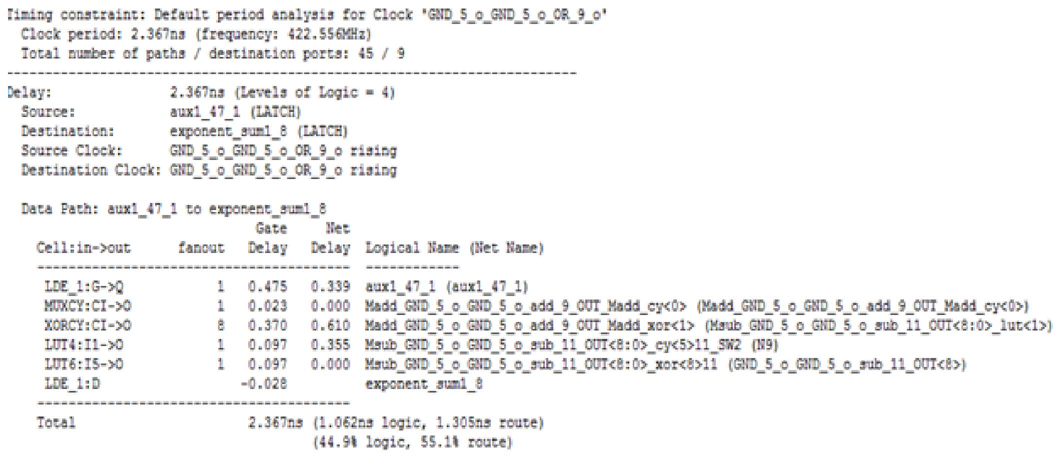


**Fig. 4.** Proposed architecture with pipelining

Significand multiplier is one of the critical parts of floating point multiplier. In this unit most of the calculation takes place which decrease the overall speed of multiplier. A parallel branch in between the significand multiplier and before the exponent adder & rounding helps to overcome the problem of speed optimization.

After that the other parallel branch in between the exponent addition and before the overflow/underflow detection add on in improving the clock rate ( up to 422.556MHz) and output the result in just 2.367ns. The figure 04 given below represents the maximum path delay in the design in between two parallel stages.



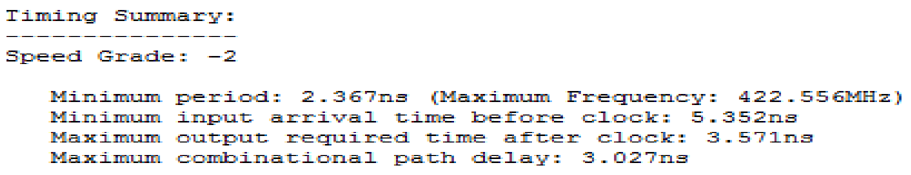
```
Timing constraint: Default period analysis for Clock 'GND_5_o_GND_5_o_OR_9_o'
  Clock period: 2.367ns (frequency: 422.556MHz)
  Total number of paths / destination ports: 45 / 9
-------------------------------------------------------------------------
Delay:               2.367ns (Levels of Logic = 4)
  Source:            aux1_47_1 (LATCH)
  Destination:       exponent_sum1_8 (LATCH)
  Source Clock:      GND_5_o_GND_5_o_OR_9_o rising
  Destination Clock: GND_5_o_GND_5_o_OR_9_o rising

  Data Path: aux1_47_1 to exponent_sum1_8
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     LDE_1:G->Q            1   0.475   0.339  aux1_47_1 (aux1_47_1)
     MUXCY:CI->O           1   0.023   0.000  Madd_GND_5_o_GND_5_o_add_9_OUT_Madd_cy<0> (Madd_GND_5_o_GND_5_o_add_9_OUT_Madd_cy<0>)
     XORCY:CI->O           8   0.370   0.610  Madd_GND_5_o_GND_5_o_add_9_OUT_Madd_xor<1> (Msub_GND_5_o_GND_5_o_sub_11_OUT<8:0>_lut<1>)
     LUT4:I1->O            1   0.097   0.355  Msub_GND_5_o_GND_5_o_sub_11_OUT<8:0>_cy<5>11_SW2 (N9)
     LUT6:I5->O            1   0.097   0.000  Msub_GND_5_o_GND_5_o_sub_11_OUT<8:0>_xor<8>11 (GND_5_o_GND_5_o_sub_11_OUT<8>)
     LDE_1:D                      -0.028          exponent_sum1_8
    ----------------------------------------
    Total                      2.367ns (1.062ns logic, 1.305ns route)
                                       (44.9% logic, 55.1% route)
```

**Fig. 5.** Path delay in between two pipelining stages

## 4. Result

The resultant section of the paper is representing the different figures of results which justifying the design, performance and accuracy of the proposed design. The figure 05 given below representing the Frequency rate, clock timing and the combinational path delay of the floating point multiplier. This result shows that the proposed design is 40% faster response than previous deigns & 90% faster than conventional design.



```
Timing Summary:
---------------
Speed Grade: -2

   Minimum period: 2.367ns (Maximum Frequency: 422.556MHz)
   Minimum input arrival time before clock: 5.352ns
   Maximum output required time after clock: 3.571ns
   Maximum combinational path delay: 3.027ns
```

**Fig. 6.** Timing summary of the proposed design

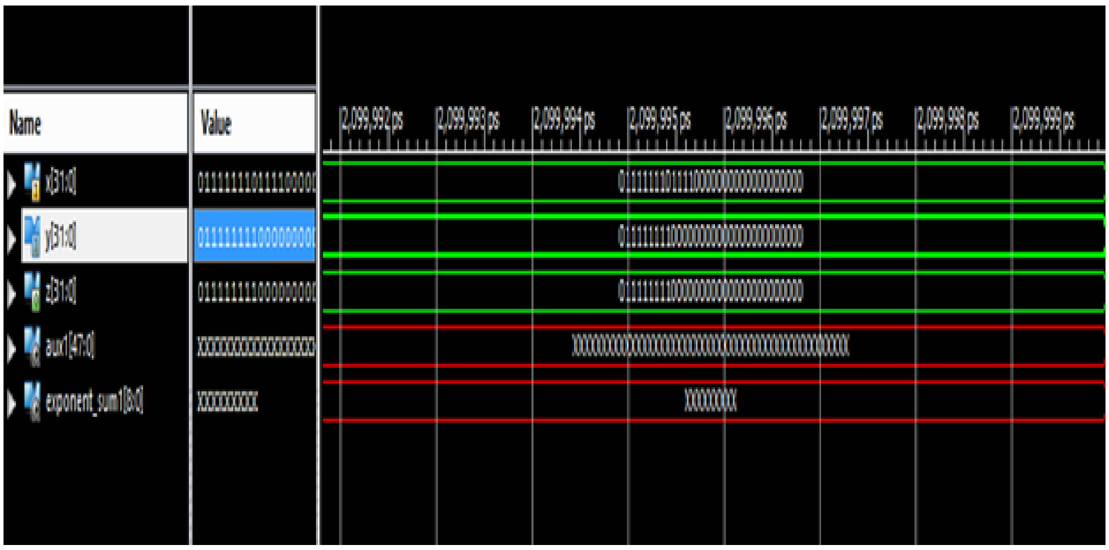


**Fig. 7.** Simulation result of multiplitication of two normalized numbers

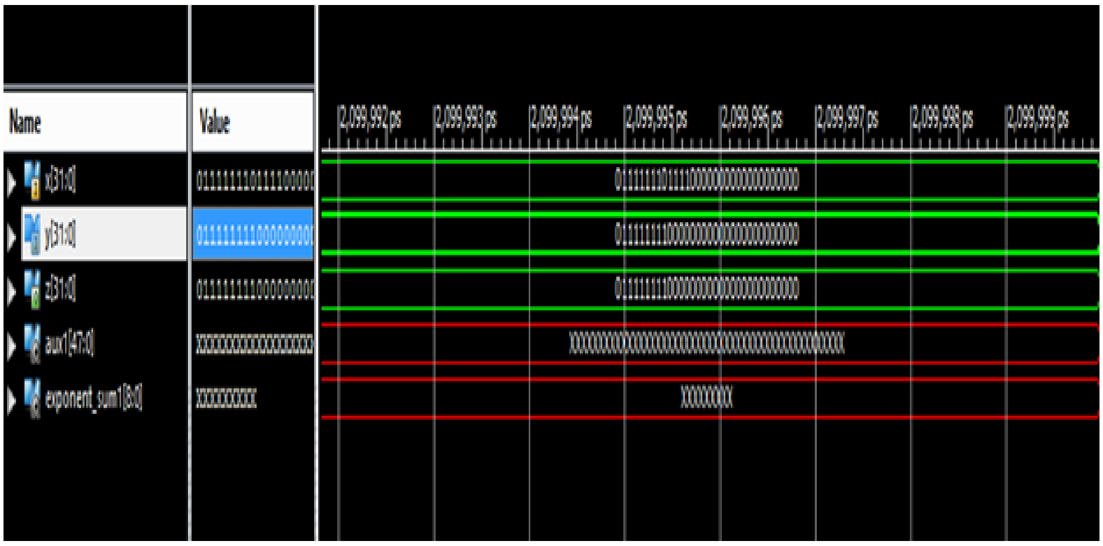


**Fig. 8.** Simulation result of multiplitication of two numbers results in infinity

## 5. Conclusion

The paper presents an implementation of floating point multiplier, which follow IEEE754 binary interchange format. The both normalization and rounding techniques are implemented in the design. The proposed design gives the output in latency of two clock cycles and the frequency rate of 422.556MHz is achieved. The whole designed is targeted to Xilinx vertex 5 FPGA.

## References

[1] IEEE 754-2008, IEEE Standard for Floating-Point Arithmetic, 2008

[2] Mohamed Al-Ashrafy, Ashraf Salem and Wagdy Anis, "An efficient implementation of floating point multiplier" 2011 IEEE

[3] H. Poor, An Introduction to Signal Detection and Estimation, New York: Springer-Verlag, 1985, ch. 4. (Book Chapter)

[4] B.Sreenivasa Ganesh, J.E.N. Abhilash and G.Rajesh Kumar, " *Design and implemntation of floating point multiplier for better response*" IJARCET,vol 1, issue 17, sep 2012.

[5] Naresh Grover, M.K Soni., " *Design of FPGA based 32 bit floating point airthmetic unit and verfication of its VHDL code using MATLAB*" I.J information engineering and electronics business ,feb 2014,MECS press

[6] Ameya deshmukh and Pooja hatwalne, " *Design and implemenatatio time efficient floating point multiplier using VHDL*, international journal of latestest trend in engineering and technology, vol 8 , issue 03, pp 084-090.