EE224: Digital System

ASSIGNMENT 2

Design a signed 8-bit (8 bit inputs and 16-bit output) quarter precision floating multiplier.

Components Used

1. Booth_multiplier(**booth2bit_v1)**: (Made in the previous assignment) used for multiplication of mantissa.
2. Full_adder 5 bit (**fa5bit)**: For addition of exponents.
3. Multiplexer(**MUX_4):** Return 5 bit exponent adder of the two possible input combinations based on the control pin

   (**MUX_2):** Return 5 bit mantissa output of the two possible input combinations based on the control pin

Algorithm:

Two 8 bit inputs, with representation of numbers in quarter precision floating point format as: S: 1 bit sign bit, E: 3 bit exponent (-3, 4)and F: 4 bit significant (fraction).

The output is a 16 bit with representation as : S: 1 bit sign bit, E: 5 bit exponent(-15,16) and F: 10 bit significant (fraction).

(i) The output signed bit is computed using XOR of input signed bits.

(ii) The mantissa of each input is first normalised by adding 1 to the MSB. Now, these 5 bits are multiplied using booth multiplier (by prefixing with '000' to make the inputs as 8 bit: compatible with the multiplier logic created). The resultant product is 16 bit, so the last significant bits are taken as the mantissa of the product. The output mantissa product may be 10 or 9 bits. Thus, these are extracted from 16 bit output computed by multiplier using a multiplexer (**MUX_4)** with a control pin as 10th bit of the output which when one results in

last 10 bits as required product else 9 bit which is then right shifted once. The normalised constant 1 at the beginning is removed then as represented in standard mantissa form in output.

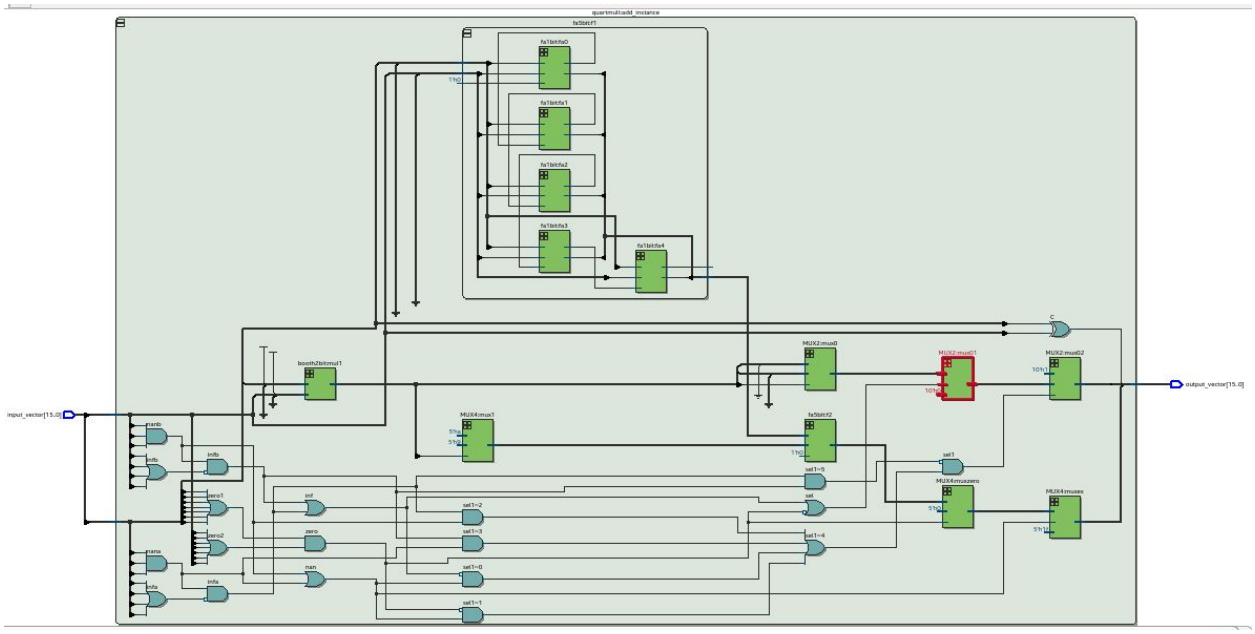
(iii) The 3 bit input exponent in binary represents 3 more than the actual exponent in decimal(eg 3 in decimal represented as 110.). Thus, the resultant exponent is computed by adding the two input exponents and adding 9 to the result using full adder (15 - 2*3). In case, there is a carry from mantissa, then 10 needs to be added instead of 9. Hence, a multiplexer (**MUX_4)** is used here.

(iv) Cases of Zero, +/- Inf and Nan are also considered.

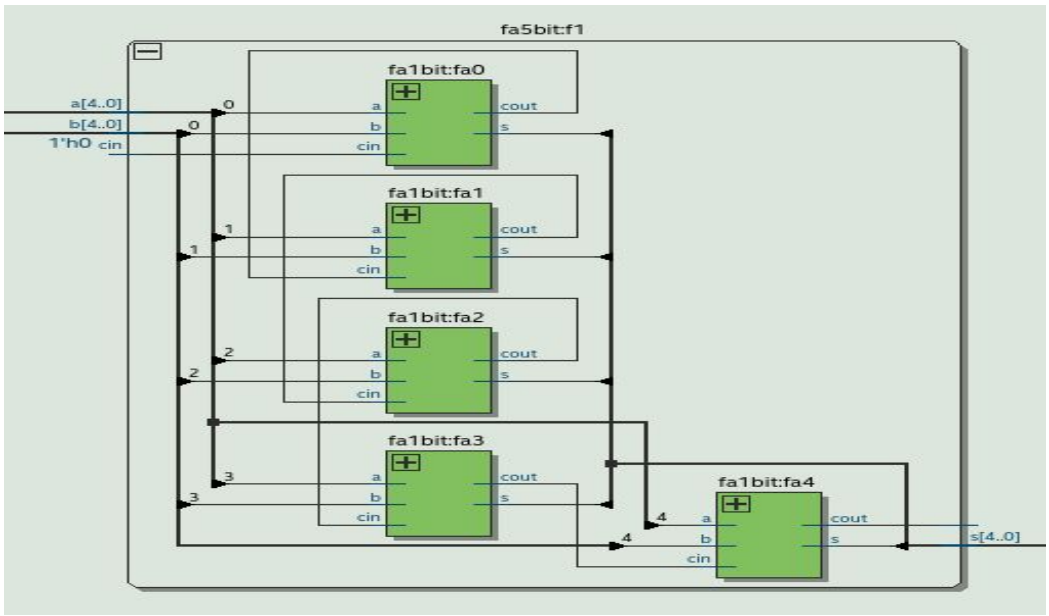
- Zero is represented by *"00000000"* (8 bit input) which when multiplied with any other combination results in zero represented in 16 bit as all zeros.
- Inf is represented by exponent being *"111"* and mantissa as *"0000"* (similarly for 16 bit), which when multiplied by any other combination or by itself results in Inf with appropriate signed bit.
- With exponent being *"111"* and any mantissa other than *"0000"* represents Nan. Product of any number with Nan results in Nan. Thus, for uniformity in all cases, represented in this case as *"<signed-bit>_11111_0000000001".* Also product of zero and Inf is also Nan as ideally their product is indeterministic.

NOTE: Denormalised form of representation is not considered in this project.

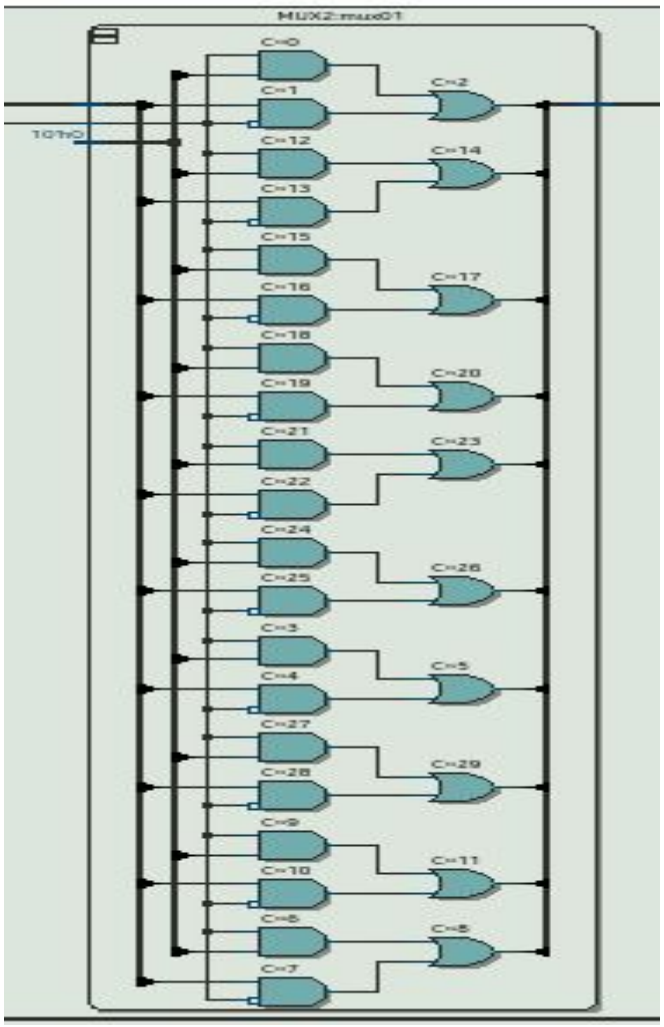
Hardware:



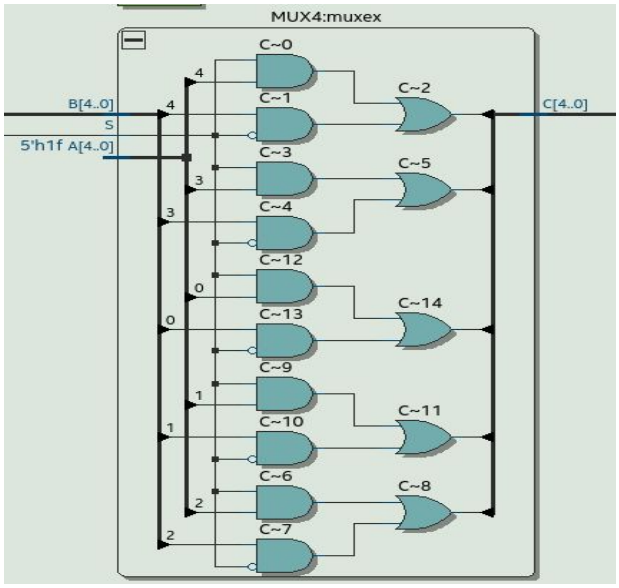
The left-most block is the booth encoded multiplier logic developed previously, the middle block is the 5 bit full adder implemented as 5 1-bit full adder and the right blocks are the multiplexers.



5 BIT FULL ADDER


MUX2:muxx01
C~0
C~1
C~2
C~12
C~14
C~13
C~15
C~17
C~16
C~18
C~20
C~19
C~21
C~23
C~22
C~24
C~26
C~25
C~3
C~5
C~4
C~27
C~29
C~28
C~9
C~11
C~10
C~6
C~8
C~7

MUX-2


MUX4:muxex
B[4..0]
S
5'h1f A[4..0]
C[4..0]
C~0
C~1
C~2
C~3
C~5
C~4
C~12
C~14
C~13
C~9
C~11
C~10
C~6
C~8
C~7

MUX-4

Note: The image may not be so clear. So an image file of the above hardware logic named as '**hardware.png'** is also attached along with codes.

FILES:

1. **Quartmult.vhd:** Precision multiplier architecture
2. **Fa5bit.vhd:** 5 bit full adder
3. **MUX2.vhd**
4. **MUX4.vhd**
5. **DUT.vhd:** Device under test file for compilation
6. **Testbench.vhdl:** Testbench
7. **TRACEFILE(1).txt:** All test cases written here
8. **OUTPUTS.txt:** The simulated output with errors if any is written here
9. **qpm8bit.qpf:** Project file
10. **Hardware.png:** hardware implementation
11. **Compilereport.png**
12. **Simulation.png:** simulated report

    Note: The below VHDL codes are ones required for booth multiplier.

13. **Booth2bit_v1.vhd:** booth multiplier architecture
14. **MUX.vhd:** multiplexer component
15. **MUX3.vhd:** decoder component
16. **Left_shift.vhd:** left-shift component
17. **Fa16bit.vhd:** 16 bit full adder
18. **Fa8bit.vhd:** 8 bit full adder
19. **Fa1bit:** 1 bit full adder

RESULTS:

The precision multiplier architecture was first compiled successfully. Then the testbench with all possible input combinations is added and simulated. The compilation and simulation reports are also attached with the codes.

REFERENCE:

http://lslwww.epfl.ch/pages/teaching/cours_lsl/sl_info/FPMultiplier.pdf

http://www.toves.org/books/float/