Quarter precision floating multiplier

Pranjal Jain

1 Algorithim

To get the sign of the product, we take the XOR of the sign bits of the two inputs.
From the exponent bits of the input, to get the the actual exponent of the input being represented, 3 is subtracted.
From the 16 bit representation of the product, if we want to obtain the product's exponent, we subtract 15 from the exponent bits. So to calculate the exponent bits of the product, we add the exponent bits of the inputs, and add 9 to this(since -3-3+15=9), and add carry generated from significand multiplication.
The two significands are multiplied. If the significand is 0101 then the number to multiplied is 1.0101. If the result is obtained is 1 followed by 9 bits, the least significant 8 bits is made the the first 8 bits of the significand of the product and 2 zeros are appended to the least significant positions of the siginficand F of the product. However if the product of significands is 10 bits, the exponent E of the product is increased by 1, and the 9 least significant bits of the product of significands is taken to be the first 9 bits of the products significand F, followed by a 0 appended at the least significant position.
Exceptional cases of inputs:

- +inf = 0 111 0000
- -inf= 1 111 0000
- NaN (Not a number): A number with exponent 111, any sign bit, and any of the four F bits non-zero. eg. 0 111 0010
- +0: 0 000 0000 and -0: 1 000 0000

I have added flags to the code which indicate whether an input is +-inf, NaN or 0.

- Any one of the inputs is NaN, output is NaN
- +-inf multiplied by +-0 gives NaN output
- +-inf multiplied by non-zero, non-NaN value gives output +-inf depending on XOR of sign bits
- +-0 multiplied by non-NaN, non-inf value gives +0 output

By implementing the above logic, I calculate an exception value, ie. the output when exceptional case of input arise. The exception value and normal output value are always calculated, however the output is decided on the basis of whether an exceptional input has been detected or not.
NOTE: The circuit gives following EXCEPTION VALUES(OUTPUTS)

- **0000000000000000 when zero should be the output(irrespective of +-0)**
- **0111110000000001 when the output is NaN(any NaN number can be the output, so I selected this one)**
- **0111110000000000 represents output +inf**
- **1111110000000000 represents output -inf**

NOTE: The TRACEFILE I have made gives expected outputs according to the above convention. The TRACEFILE does NOT contain testcases for Subnormal numbers, the circuit is not designed to handle these cases. However, the circuit can handle the rest of the bonus corner cases.

2 Circuit

RTL viewer in quartus gives us the following circuit.

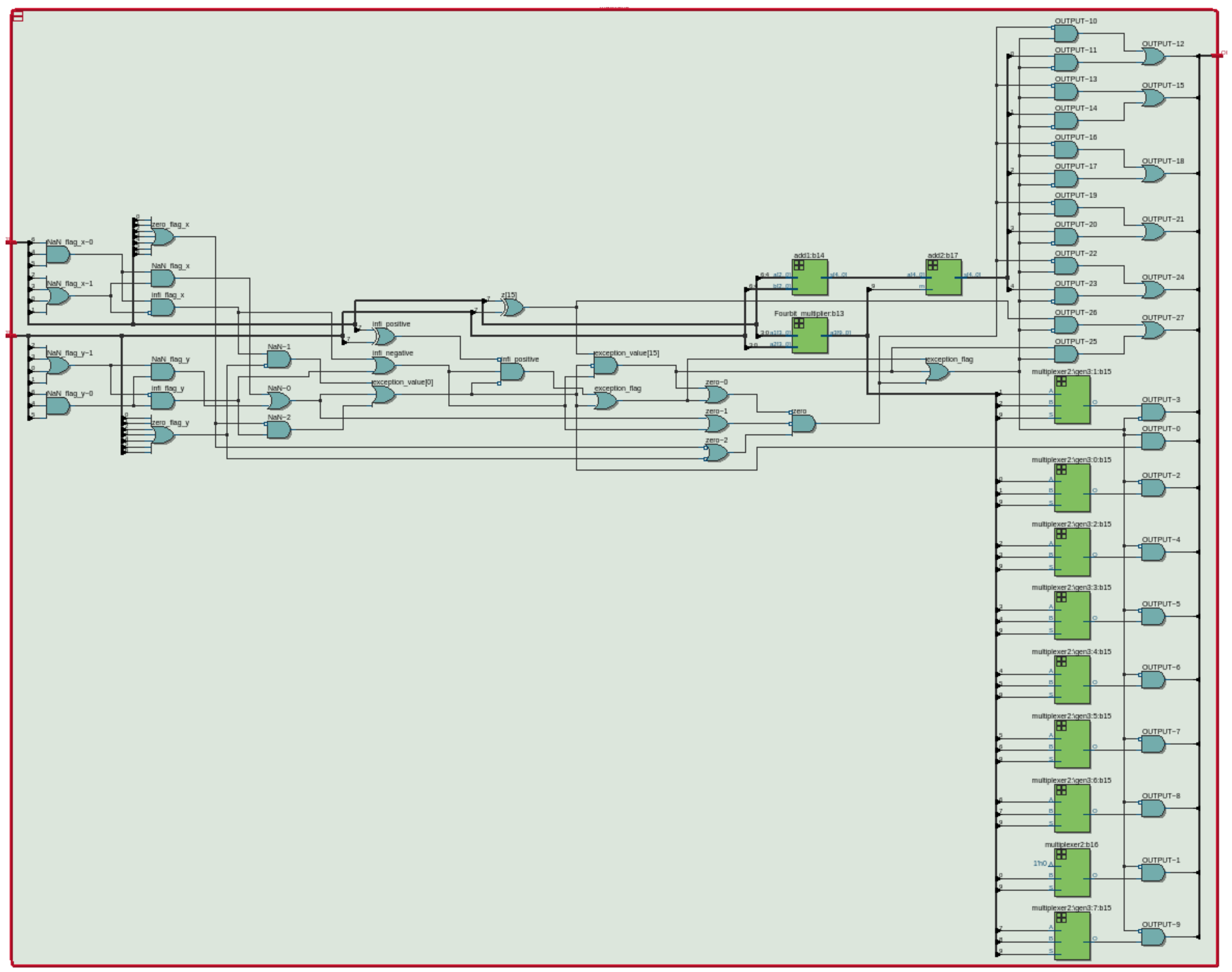


Figure 1: Hardware Circuit

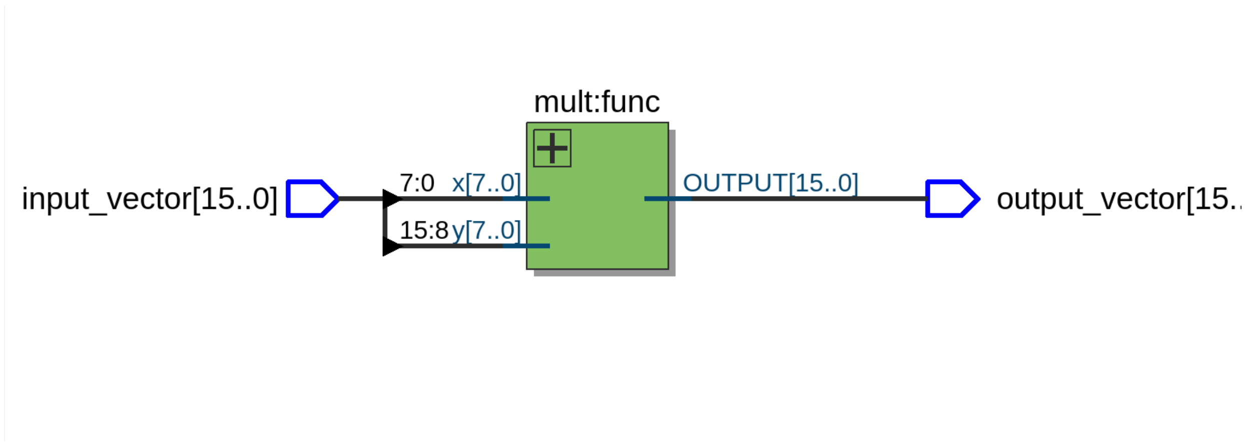


Figure 2: The functional unit

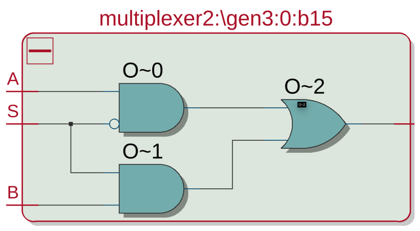


Figure 3: 2-to-1 Multiplexer

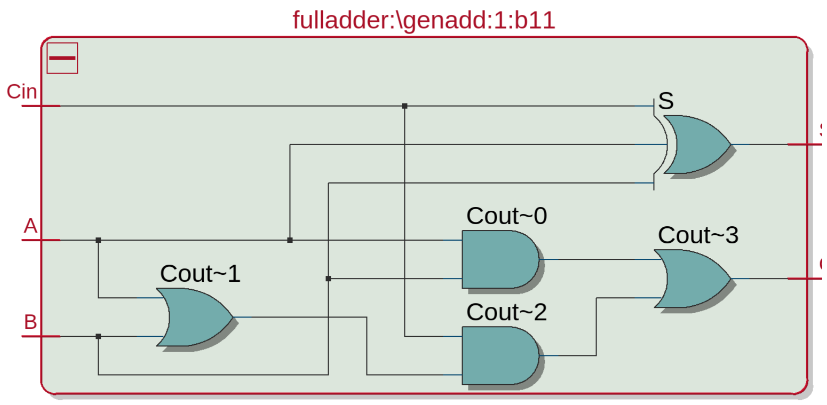
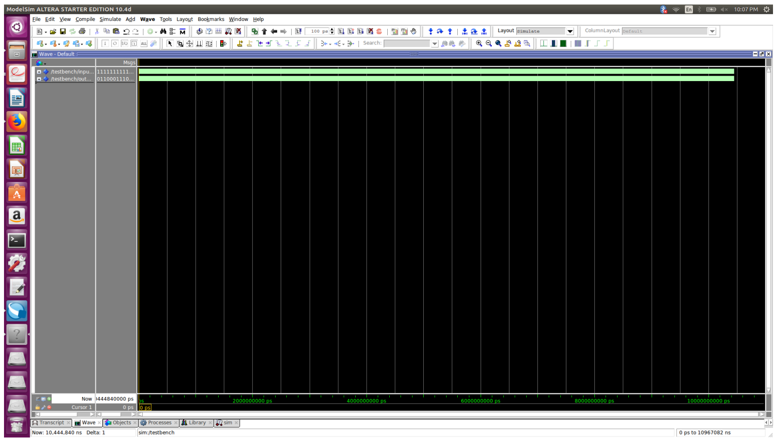


Figure 4: Full adder

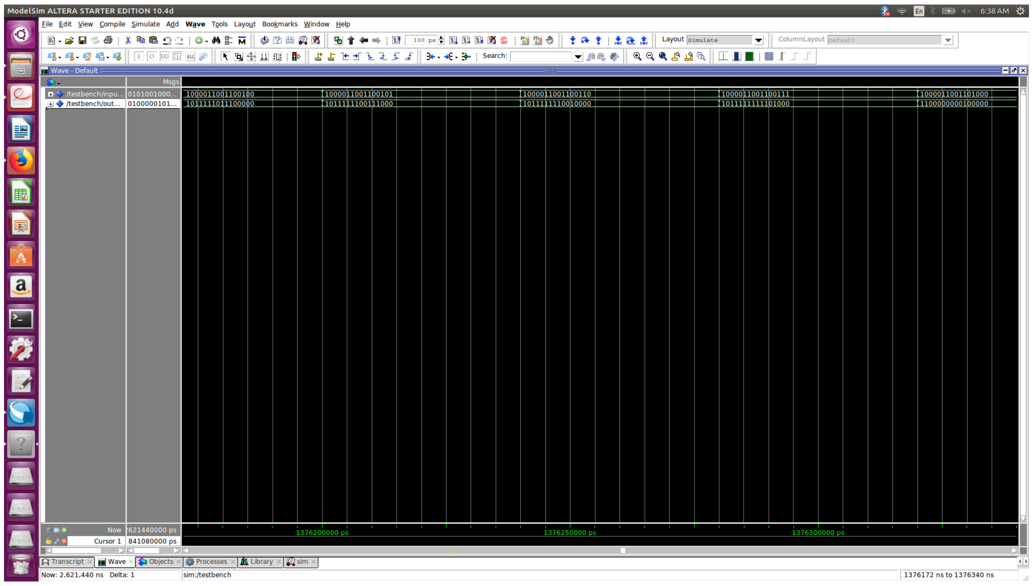
3 Waveforms



```
/testbench/inpu...   1111111111...
/testbench/out...    0110001110...

Now       0444840000 ps
Cursor 1  0 ps

0 ps   2000000000 ps   4000000000 ps   6000000000 ps   8000000000 ps   10000000000 ps

Now: 10,444,840 ns  Delta: 1    sim:/testbench    0 ps to 10967082 ns
```

ModelSim ALTERA STARTER EDITION 10.4d
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Wave - Default
Msgs
Transcript Wave Objects Processes Library sim

Figure 5: Waveform for all testcases



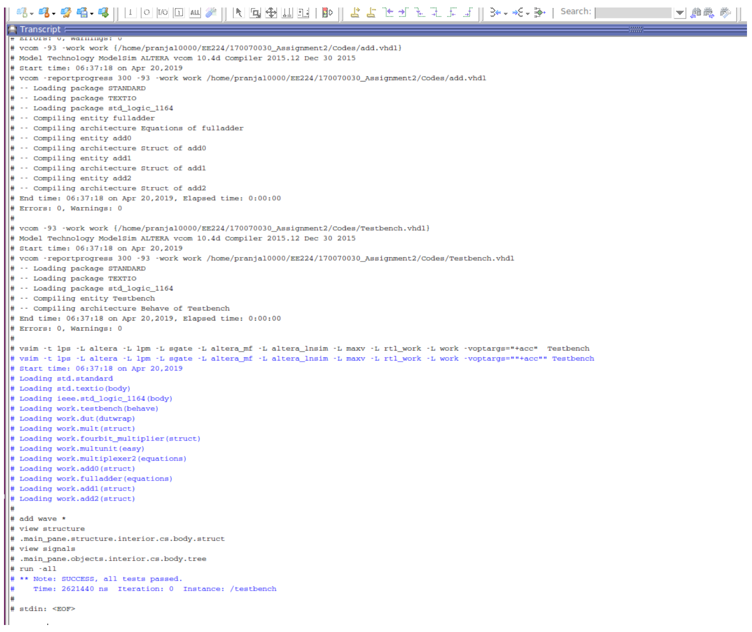
```
/testbench/inpu...   0101001000...   1000011001100100   1000011001100101   1000011001100110   1000011001100111   1000011001101000
/testbench/out...    0100000101...   1011111011100000   1011111110011000   1011111110010000   1011111111101000   1100000000100000

Now       2621440000 ps
Cursor 1  841080000 ps

1376200000 ps   1376250000 ps   1376300000 ps

Now: 2,621,440 ns  Delta: 1    sim:/testbench    1376172 ns to 1376340 ns
```

ModelSim ALTERA STARTER EDITION 10.4d
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
Search:
Wave - Default
Msgs
Transcript Wave Objects Processes Library sim

Figure 6: Waveform for certain testcases

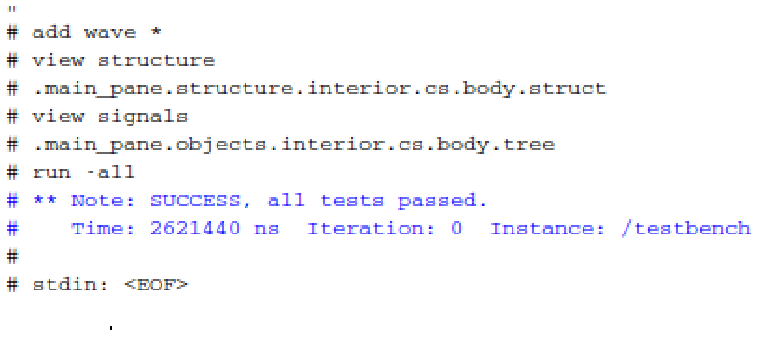
4 Success

The transcript of RTL simulation is shown below



```
# vcom -93 -work work {/home/pranjal0000/EE224/170070030_Assignment2/Codes/add.vhdl}
# Model Technology ModelSim ALTERA vcom 10.4d Compiler 2015.12 Dec 30 2015
# Start time: 06:37:18 on Apr 20,2019
# vcom -reportprogress 300 -93 -work work /home/pranjal0000/EE224/170070030_Assignment2/Codes/add.vhdl
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Compiling entity fulladder
# -- Compiling architecture Equations of fulladder
# -- Compiling entity add0
# -- Compiling architecture Struct of add0
# -- Compiling entity add1
# -- Compiling architecture Struct of add1
# -- Compiling entity add2
# -- Compiling architecture Struct of add2
# End time: 06:37:18 on Apr 20,2019, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
#
# vcom -93 -work work {/home/pranjal0000/EE224/170070030_Assignment2/Codes/Testbench.vhdl}
# Model Technology ModelSim ALTERA vcom 10.4d Compiler 2015.12 Dec 30 2015
# Start time: 06:37:18 on Apr 20,2019
# vcom -reportprogress 300 -93 -work work /home/pranjal0000/EE224/170070030_Assignment2/Codes/Testbench.vhdl
# -- Loading package STANDARD
# -- Loading package TEXTIO
# -- Loading package std_logic_1164
# -- Compiling entity Testbench
# -- Compiling architecture Behave of Testbench
# End time: 06:37:18 on Apr 20,2019, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
#
# vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L maxv -L rtl_work -L work -voptargs="+acc"  Testbench
# vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L maxv -L rtl_work -L work -voptargs=""+acc"" Testbench
# Start time: 06:37:18 on Apr 20,2019
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.testbench(behave)
# Loading work.dut(dutwrap)
# Loading work.mult(struct)
# Loading work.fourbit_multiplier(struct)
# Loading work.multunit(easy)
# Loading work.multiplexer2(equations)
# Loading work.add0(struct)
# Loading work.fulladder(equations)
# Loading work.add1(struct)
# Loading work.add2(struct)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 2621440 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>
```

Figure 7: The transcript of the RTL



```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 2621440 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>
```

Figure 8: Success of the RTL simulation