Precision Floating Multiplier

Syomantak Chaudhuri
170070004

April 2019

1 Algorithm

The algorithm used is rather simple as one would do with pen and paper. If we are to multiply two numbers A = $1.(a_3a_2a_1a_0) \cdot 2^{\alpha}$ and B = $1.(b_3b_2b_1b_0) \cdot 2^{\beta}$, then we simply compute

$$Z = 1a_3a_2a_1a_0 \ \cdot \ 1b_3b_2b_1b_0$$

Z is either a 9 bit number or a 10 bit number. For 10 bit number, lets say $Z = 1z_8....z_0$. Consider $K = 1.(k_8....k_0)$ then AxB = $K \cdot 2^{\alpha+\beta+1}$. We get an extra 1 in the exponent due to the fact that Z was 10 digit instead of 9. If Z was 9 digit then there would be no 1 in the exponent.
Now in lets say the exponent bit are equal to a and b (in decimal) respectively for the two input numbers. Then accounting for the bias, $\alpha = a - 3$ and $\beta = b - 3$. Final exponent is $a + b + n - 6$ where n is 1 if Z is 10 digit, else it is 0. Accounting for the bias in 16-bit representation, the value to be stored in decimal is $a + b + n - 6 + 15 = a + b + n + 9$. These additions are done using full adders and the Z computation is done using fast booth multiplier. Sign of output is just XOR of the two input number's sign bits. We find out whether Z is 10 digit or 9 digit by inspecting the value at output of booth multiplier at 9th position (0 indexed).
Finally, we get the output as following -
out(15) = XOR of sign bits on inputs
out(14 to 10) = a+b+n+9
if(n=1):

out(9 to 1) = $k_8....k_0$

out(0) = 0
else:

out(9 to 2) = $k_7....k_0$

out(1 to 0) = 00
Basically we just pad 0s at of mantissa.

2 Components

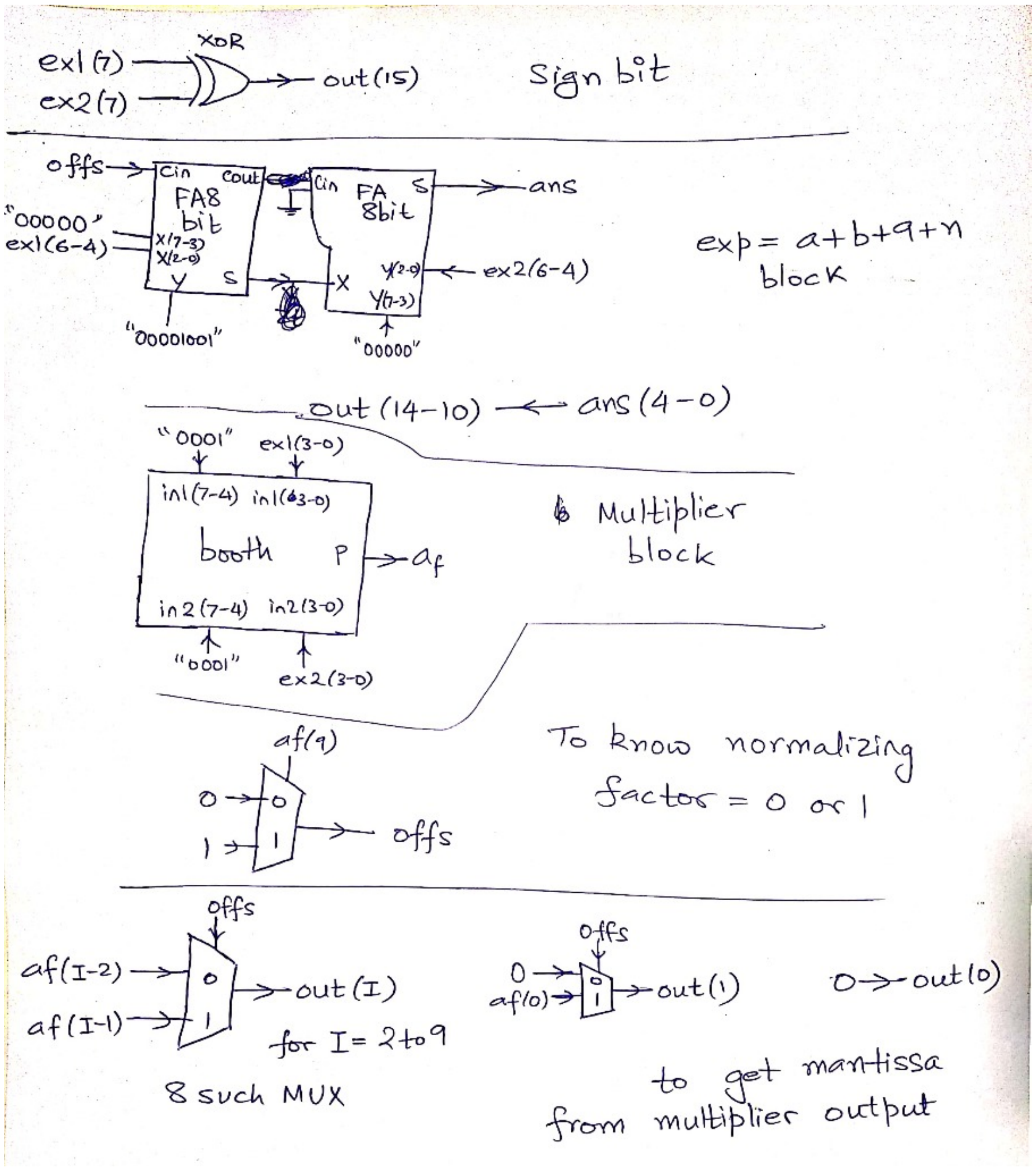
- 1 Booth multiplier
- 1 XOR gate
- 10 MUX-2 input
- 2 Full Adder 8-bit

Each MUX-2 input has 2 AND gate, 1 OR gate and 1 NOT gate. Each booth multiplier has the following inside it:

- 8 MUX-2 input
- 32 NOT gates
- 10 8bit Full Adders
- 16 MUX-4 input
- 48 MUX-8 input

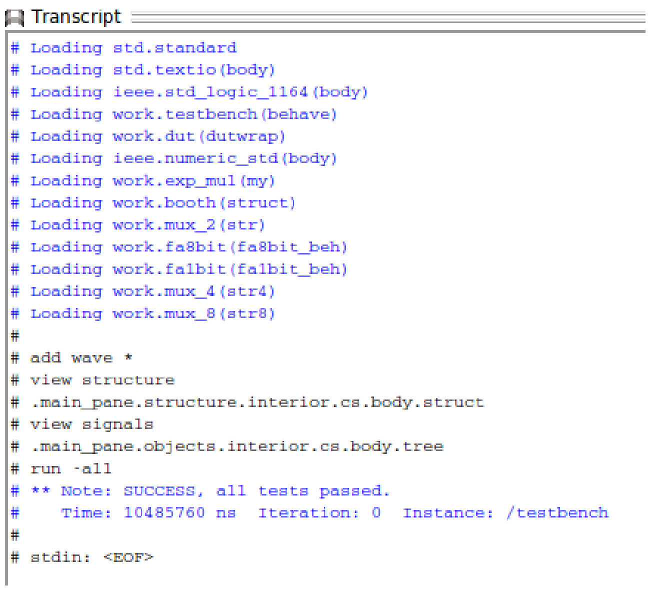
3 Hardware

The circuit designed is shown below



4 Results

I generated a tracefile that covers all possible inputs and did RTL simulation. It passed all cases and output is shown.



Transcript

```
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading work.testbench(behave)
# Loading work.dut(dutwrap)
# Loading ieee.numeric_std(body)
# Loading work.exp_mul(my)
# Loading work.booth(struct)
# Loading work.mux_2(str)
# Loading work.fa8bit(fa8bit_beh)
# Loading work.fa1bit(fa1bit_beh)
# Loading work.mux_4(str4)
# Loading work.mux_8(str8)
#
# 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: 10485760 ns  Iteration: 0  Instance: /testbench
#
# stdin: <EOF>
```

Figure 1: All Test Cases Passed

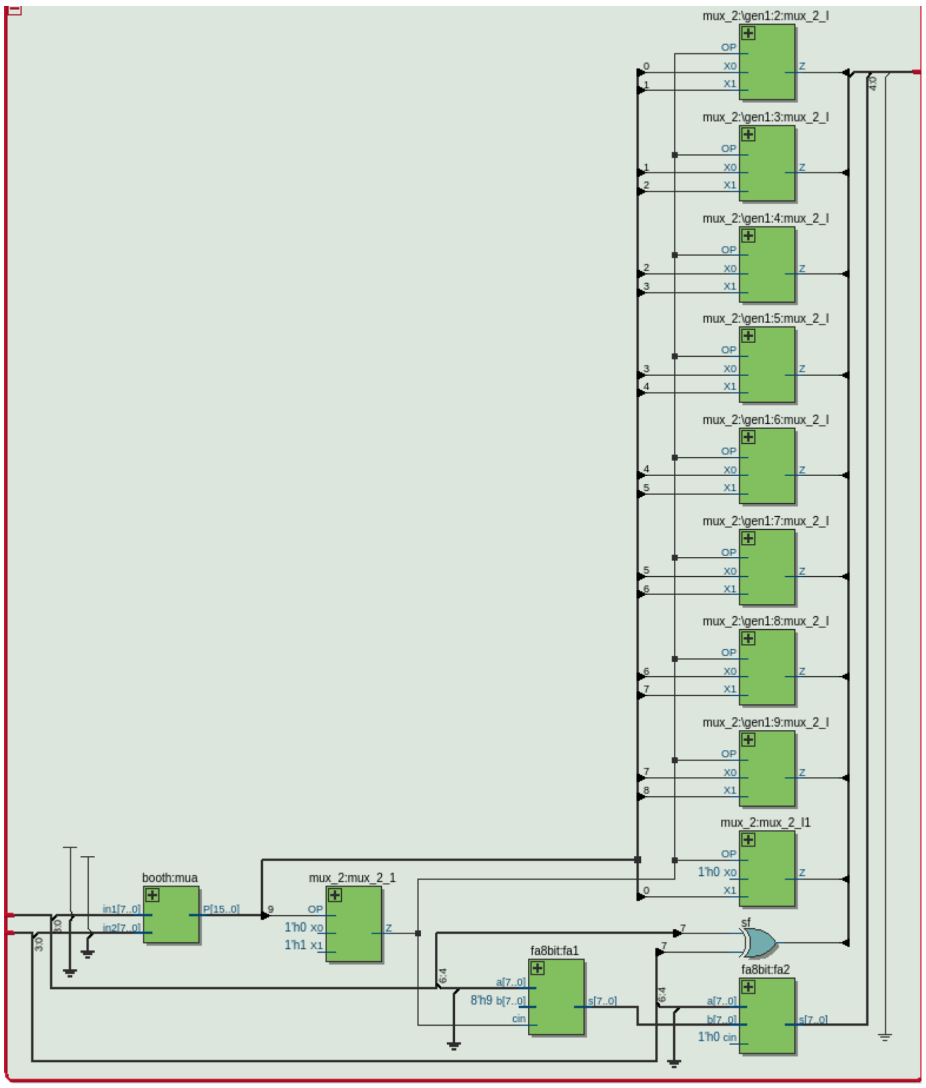


Figure 2: Netlist View