EE224: Assignment-2 Report

Quarter Precision Multiplication
Of 8-Bit Signed Numbers

Jayesh Choudhary, 170070038

1 Introduction

Till now we were performing only multiplication of integers but now in this assignment we are performing the multiplication of floating numbers with precision upto 2 decimal places or quarter-precision. The representation of these number is different. The first bit represents the sign of the number. Next three digits (considering 8-bit number) represents the exponent (with bias added, in this case 3) of the number and final 4 bits represents the mantissa of the number.
In case of 16-bit number the split is 1-5-10 (bias being 15).

2 Algorithm

For the sign of the number we are using XOR of the sign of the 2 numbers.
For exponent we are adding the exponents of the two number and then adding the bias term. Here the bias term is 15 when the product of the mantissa term is a 9 bit number and we add 1 more to the bias term if the product of the mantissa is a 10-bit number.
To get the mantissa of the product we add 1 as the most significant bit to the mantissa to get 2 5-bit numbers.We then perform the multiplication of the two numbers and choose (the last 9 bits with '0' as the last bit) or (the last 8 bits with '00' as the last bits) depending on the 10th bit of the product of 5-bit multiplier.

3 VHDL files

- Gates
- Full_Adder
- mux
- tenbit
- multiplier
- precisionmultiplier
- Testbench
- DUT
- modified_precisionmultiplier

3.1 Gates, Full_Adder

These contains the entity description of basic logic gates and some other function that we are going to use as a component in our project.

3.2 mux

This is basically to perform the left shift operation.

3.3 tenbit

This file have an entity which performs the function of adding the partial sums.

3.4 multiplier

This unit contains the entity to perform the 5-bit multiplication to give out a 10-bit number.

3.5 precisionmultiplier

This is main driver of the program with the logic to assign the values to the 16-bit output taking in 8-bit input.

3.6 DUT

This file is for the wrapping-up of the whole code.

3.7 Testbench

This file is for the interaction of the code with the test cases so that we can apply various inputs and check the output.

3.8 modified_precisionmultiplier

This file have modified code for precision multiplier taking into account the representation of zero.

4 RTL View

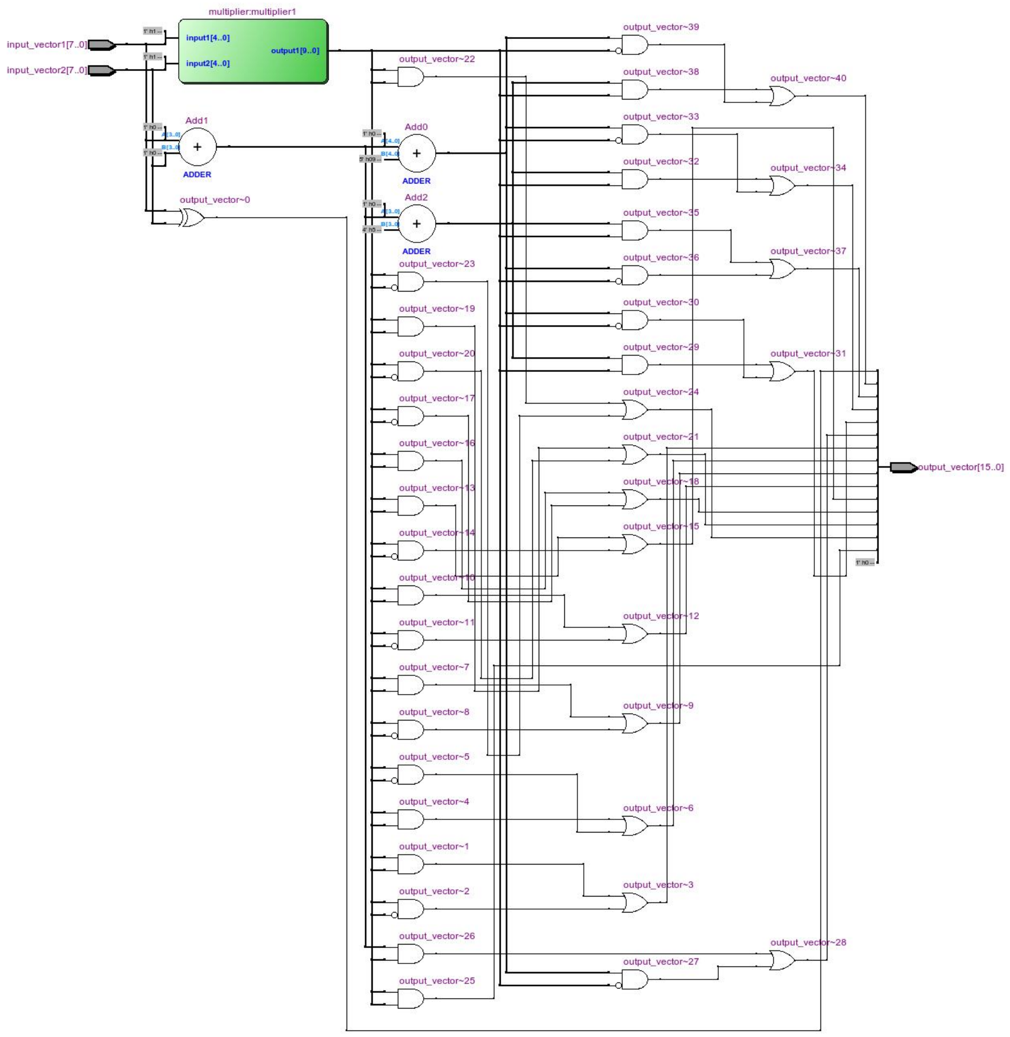


Figure 1: Design

5 Compilation and Success full Test-passing

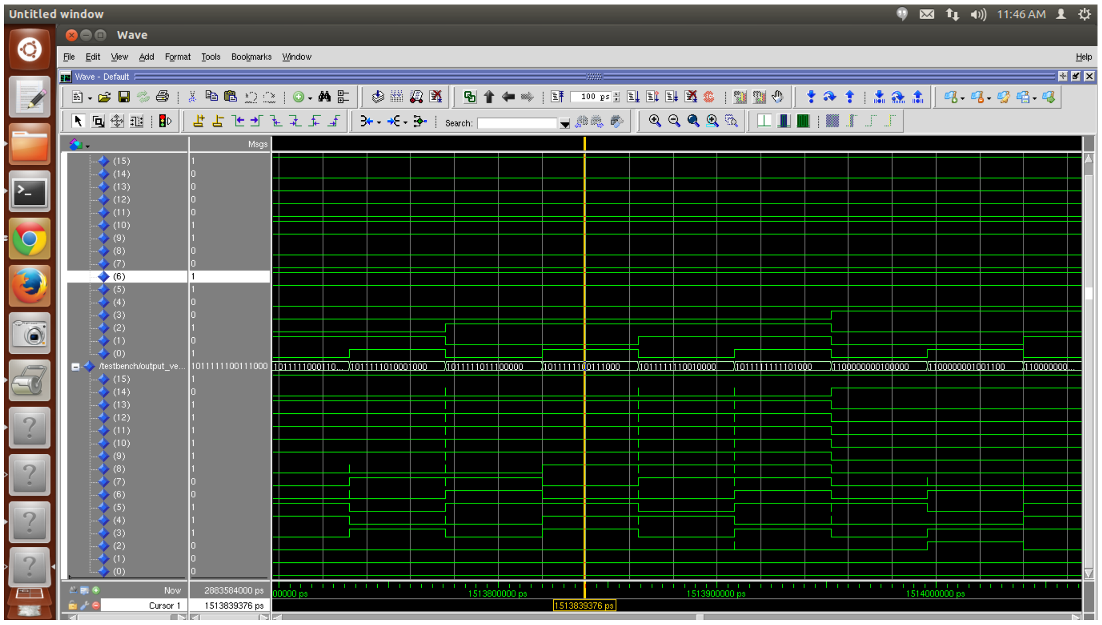
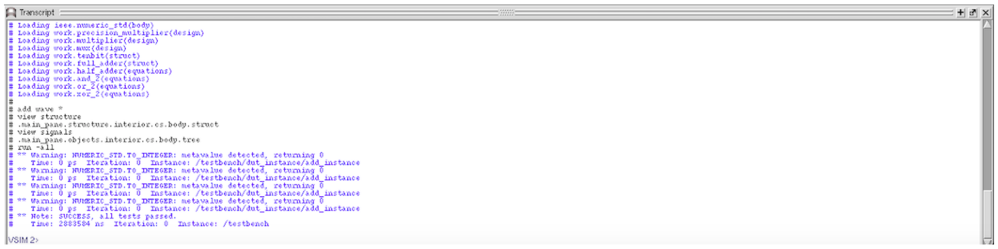


Figure 2: Output waveform for the given testcases



```
# Loading ieee.numeric_std(body)
# Loading work.precision_multiplier(design)
# Loading work.multiplier(design)
# Loading work.mux(design)
# Loading work.tenbit(struct)
# Loading work.full_adder(struct)
# Loading work.half_adder(equations)
# Loading work.and_2(equations)
# Loading work.or_2(equations)
# Loading work.xor_2(equations)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run -all
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /testbench/dut_instance/add_instance
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /testbench/dut_instance/add_instance
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /testbench/dut_instance/add_instance
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ps  Iteration: 0  Instance: /testbench/dut_instance/add_instance
# ** Note: SUCCESS, all tests passed.
#    Time: 2883584 ns  Iteration: 0  Instance: /testbench

VSIM 2>
```

Figure 3: Transcript for passing all the test cases