ECE 571- Winter 2023

FINAL PROJECT

32 BIT SINGLE PRECISION FLOATING POINT MULTIPLIER

Portland State UNIVERSITY

GROUP 14

Goutham Kumar Reddy Palem goutham2@pdx.edu
Namratha Bathula nbathula@pdx.edu
Prudvish Korrapati prudvish@pdx.edu
Mahalsa Sai Donthamahalsa@pdx.edu

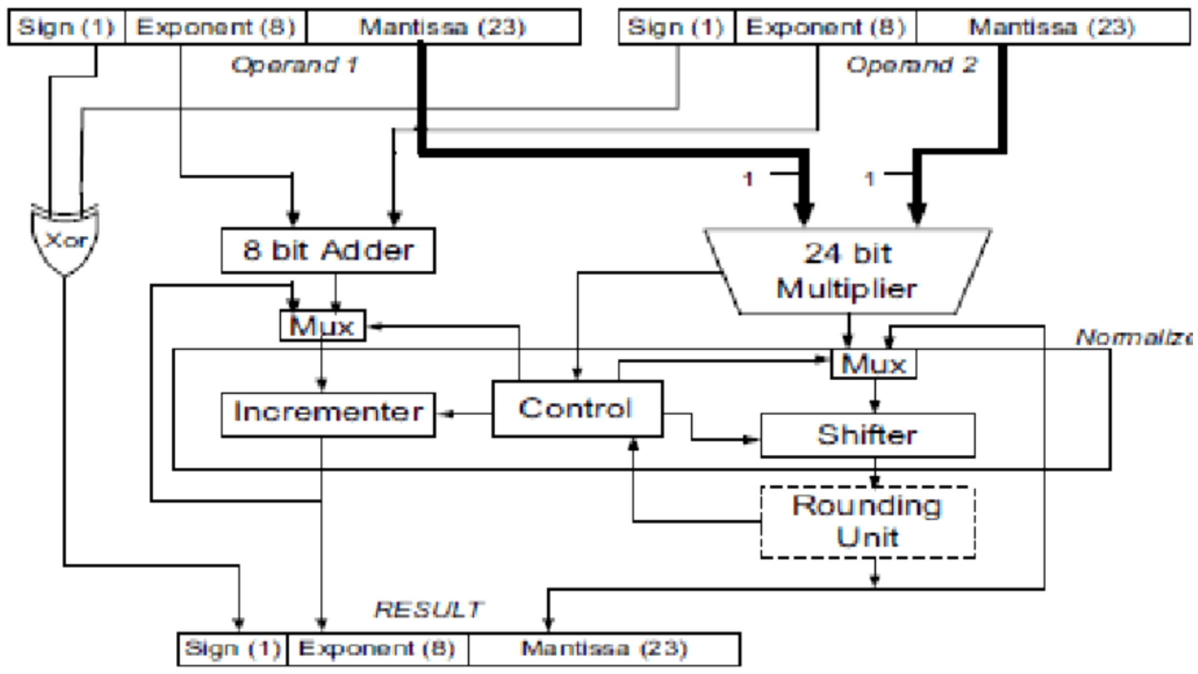
Github Link: https://github.com/nbathula16/SV-Project

FINAL REPORT

Table of contents

1.Starting point of the Project: Designing the 32-bit floating point multiplier by referring to the 24-bit multiplier design using Verilog HDL from the book Computer Architecture Design Using Verilog HDL by Joseph Cavanagh.

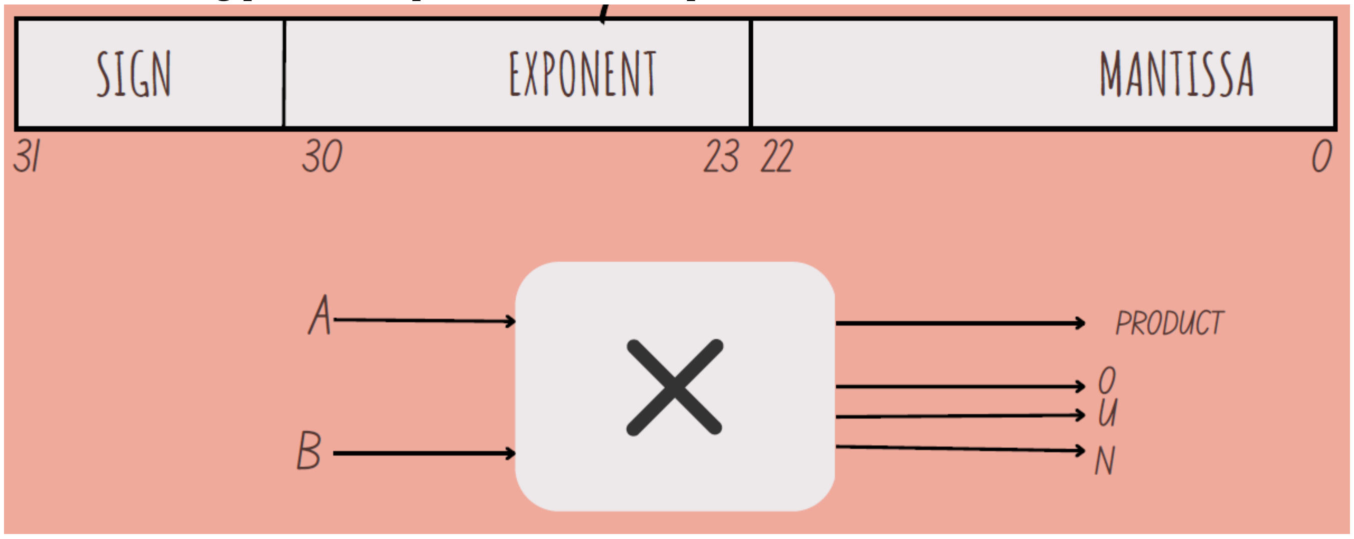
2.Block Diagram:



Architecture of 32-bit Single Precision Floating-Point Multiplier

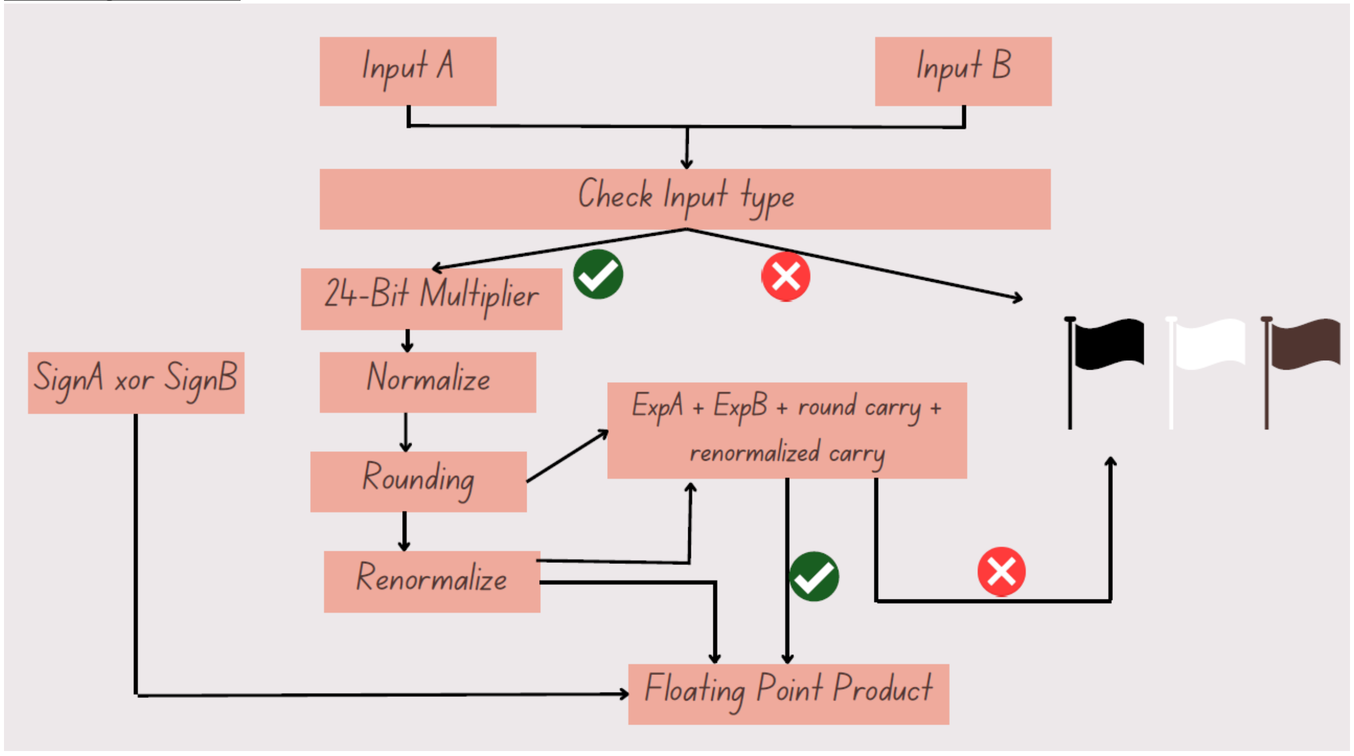
3.AIM:

32-Bit Floating point Multiplication of two inputs:



- Supporting denormalized numbers
- Raising Overflow, Underflow, NaN flags when applicable
- Implementing Assertions
- Performing Self-Check in the testbench
- Checking the design using directed test cases (conditional compilation)
- Randomizing the test cases using classes
- Performing Functional Coverage

4.Design Flow:

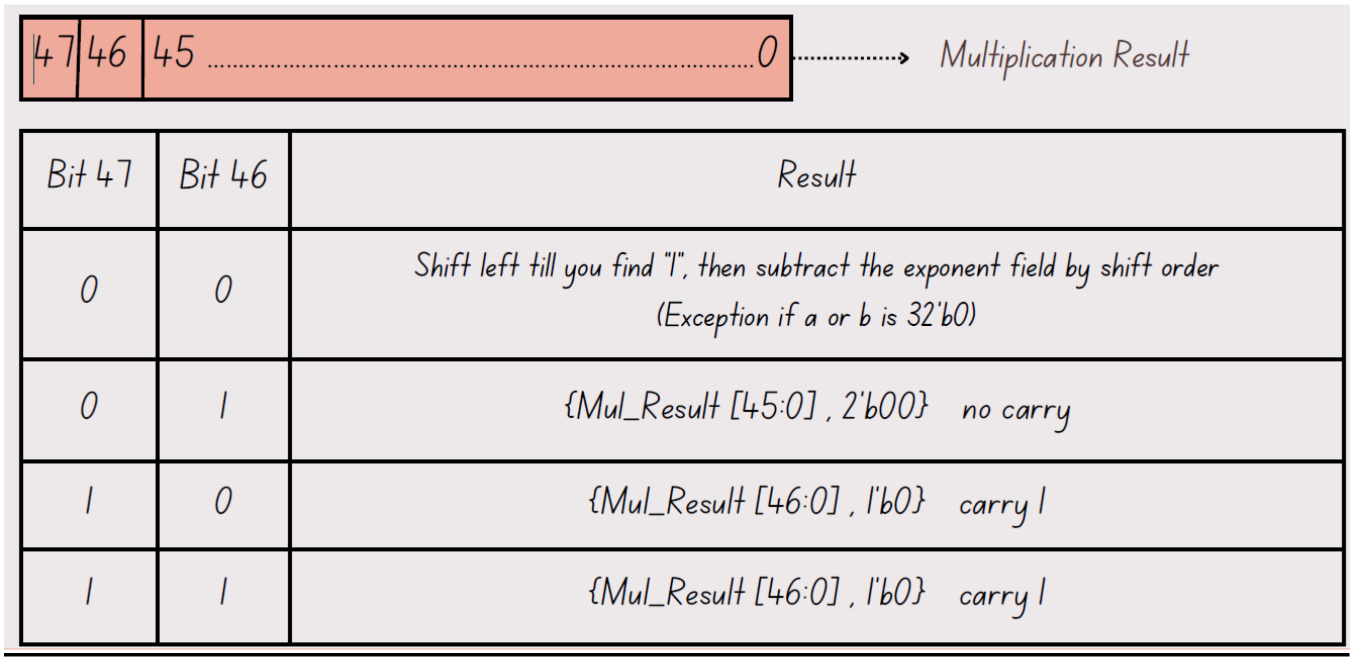


5.Design :

1)24-Bit Multiplier:

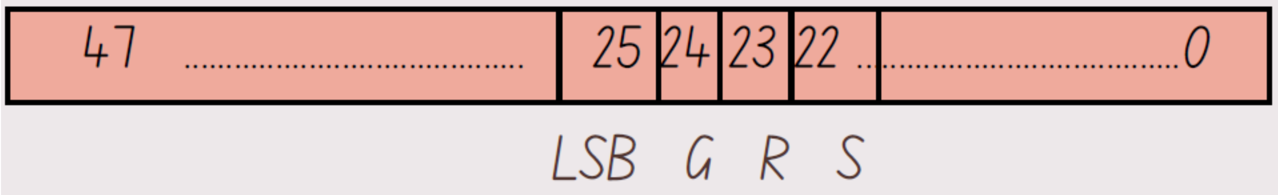
- 48-Bit Product
- Store the product in a row of a 2-D array
- consequently.
- Shift left by 1 bit for every new stage of
- multiplication.
- Sum the coloumn elements with the same indices
- by propagating the carry.

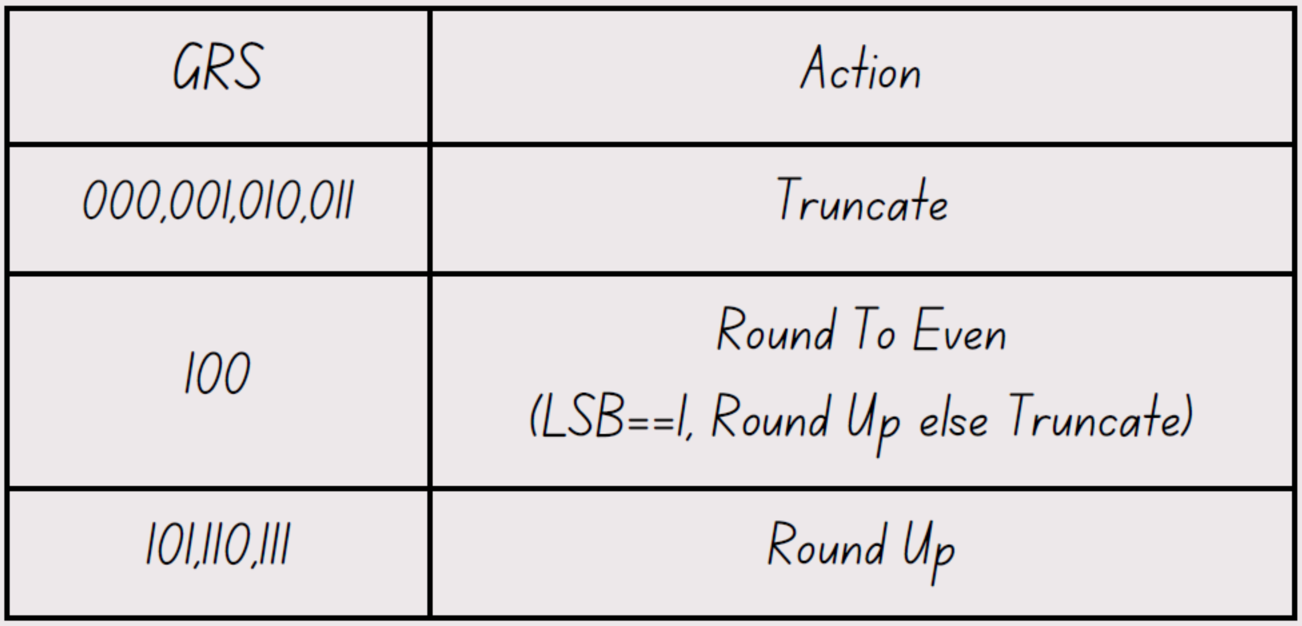
2)NormalizationLogic:


47 | 46 | 45 0
Multiplication Result

Bit 47	Bit 46	Result
0	0	Shift left till you find "1", then subtract the exponent field by shift order (Exception if a or b is 32'b0)
0	1	{Mul_Result [45:0] , 2'b00} no carry
1	0	{Mul_Result [46:0] , 1'b0} carry 1
1	1	{Mul_Result [46:0] , 1'b0} carry 1

3)Rounding Logic:





GRS	Action
000,001,010,011	Truncate
100	Round To Even (LSB==1, Round Up else Truncate)
101,110,111	Round Up

4)Exponent field:

- Add A exponent , B exponent and other propagated carries.
- If the sum equals to -126???
  Exponent field would become 8'b0, a denormalized product would generate!
- If the sum with bias exceeds 254???
  Either NaN(255) or Overflow
- If the sum is less than -126???
  Underflow

6.Verification Plan: (TestBench)

Step 1: Listed the Corner cases: (Direct test cases):

- Zero x Any Number = Zero
- Overflow
- Underflow
- NaN as Product
- Denormalized x Denormalized
- Normalized x Denormalized
- Normalized x Normalized
- Input x Reciprocal = 1
- Invalid Inputs

Step 2: Self Checking Test Bench:

- Used shortreal values to multiply.
- Used real values to find out Overflow or Underflow!

Step 3: Randomization:

- Created classes for both A and B to generate constraint random stimulus.
- Built constraints and sampled the values to inputs.
- Exercised the design by running 100000..... many many test cases ;)
- Found and Fixed Unexpected Bugs.

Step 4: Coverage:

- Created covergroup and declared the final floating point result to be the coverpoint.
- Created bins according to the constraints given in randomization.
- Ran the design until 100% Coverage is obtained.
- Generated Coverage Report in Questasim.

7.Bugs:

- Zero x any number failed, this is due to shifting operation to find '1' in the multiplication result(48-bit) which the program will never find.
- NaN in shortreal(self checking) had anonymous behavior as the range is exceed (We instead checked with real to detect overflow and underflow flags)
- While randomization, we found out that our rounding module didn't function as expected.

8.Applications:

- To improve the speed and efficiency of the real-number computations in computers(64-bits
- for real).
- To represent non-integer fractional numbers in engineering and technical calculations.
- To cover a large range with less number of bits.
- In calculating cosmological distances.
- Used in Digital Signal Processing.
- To study molecular Dynamics