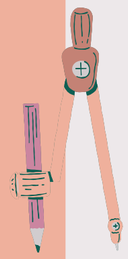


π =  5

32-bit Single Precision Floating Point Multiplier

×

Goutham Kumar Reddy Palem

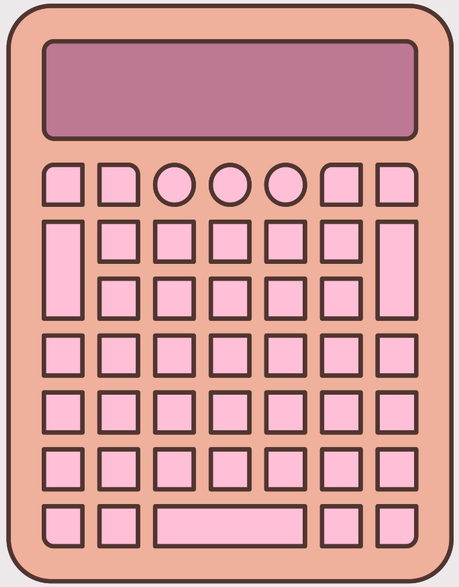
Namratha Bathula

Mahalsa Sai Dontha

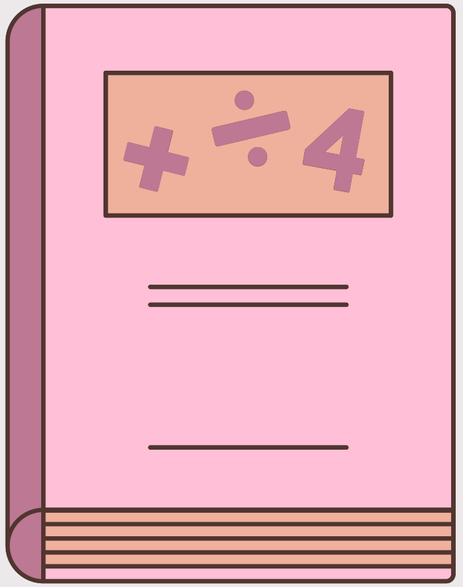
Prudvish Korrapati

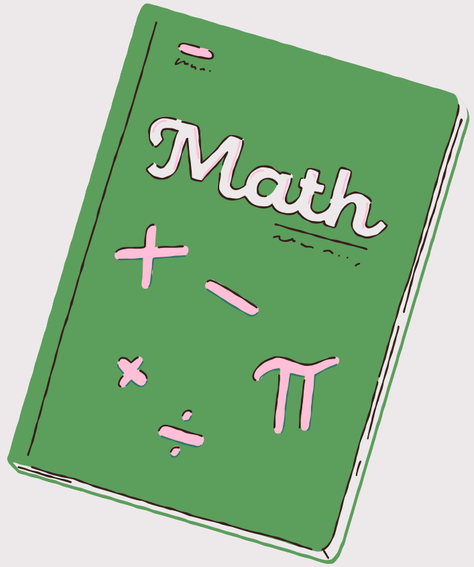
Github link: https://github.com/nbathulal6/SV-Project

TODAY WE'LL DISCUSS



- References
- Aim
- Introduction
- Flow Chart
- Design
- Verification Plan
- Bugs
- Applications



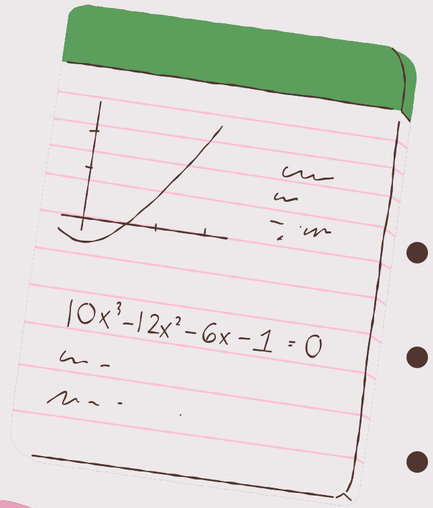


Computer Architecture Design Using Verilog HDL

By Joseph Cavanagh

REFERENCES

PROJECT AIM



32-Bit Floating point Multiplication of two inputs

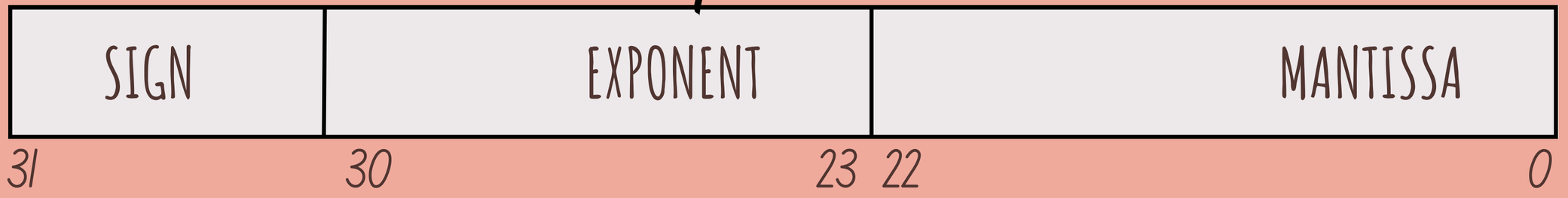
- Supporting denormalized numbers
- Raising Overflow, Underflow, NaN flags when applicable
- Implementing Assertions
- Performing Self-Check in the testbench
- Checking the design using directed test cases (conditional compilation)
- Randomizing the test cases using classes
- Performing Functional Coverage

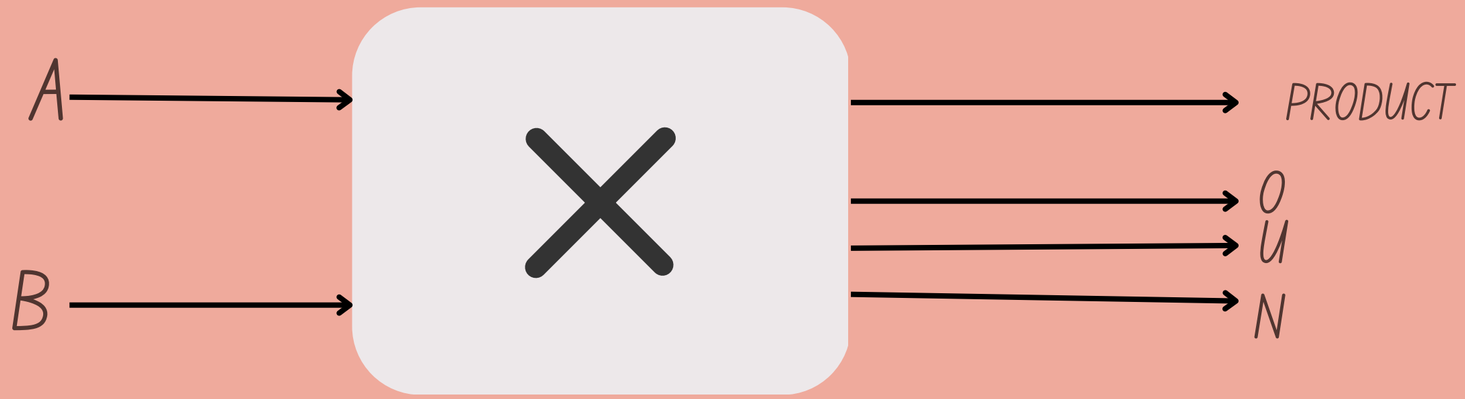
INTRODUCTION

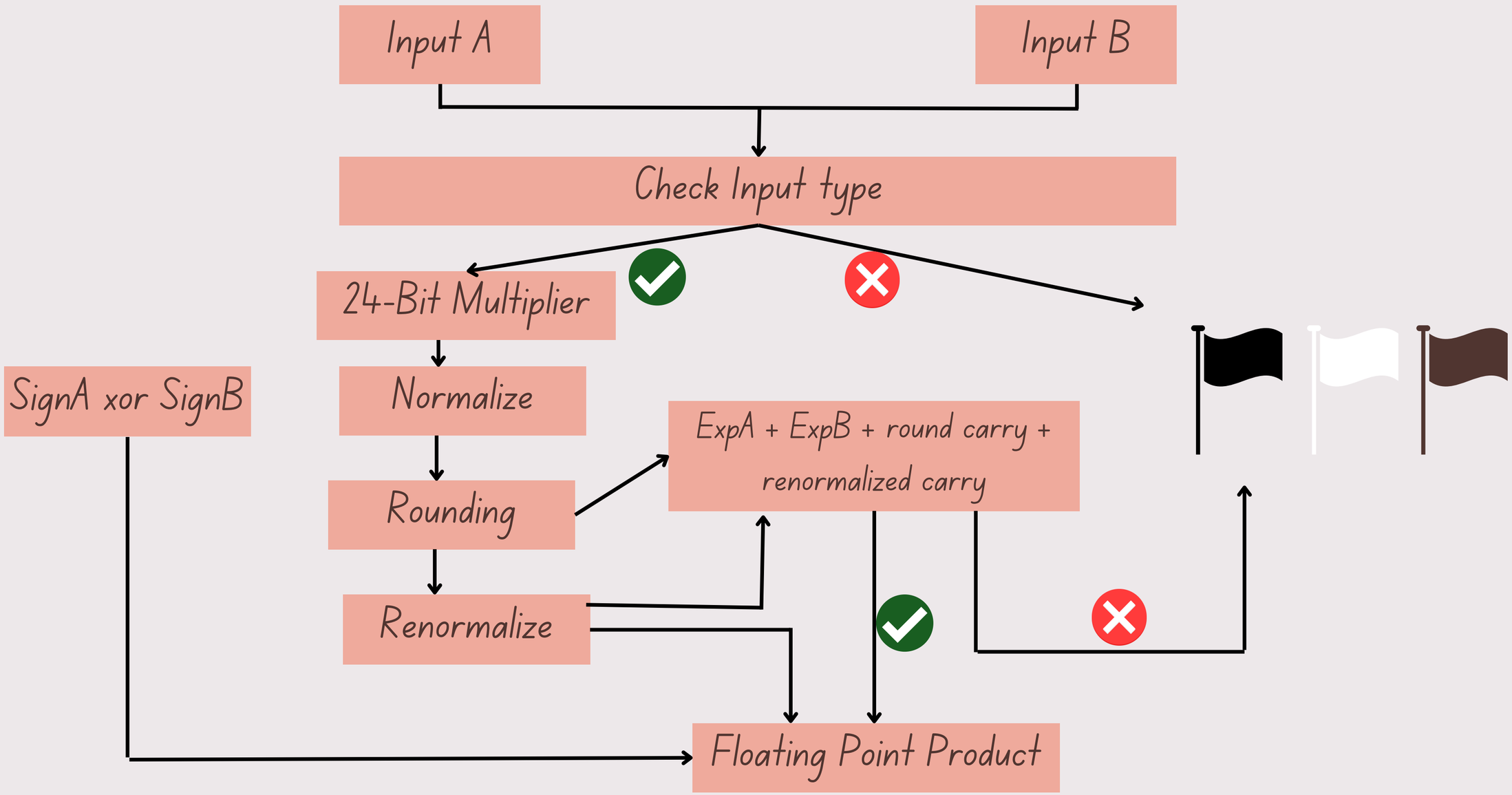
+127 biased

+∞ -∞

NAN






Input A
Input B
Check Input type
24-Bit Multiplier
Normalize
Rounding
Renormalize
SignA xor SignB
ExpA + ExpB + round carry + renormalized carry
Floating Point Product

24-Bit Multiplier

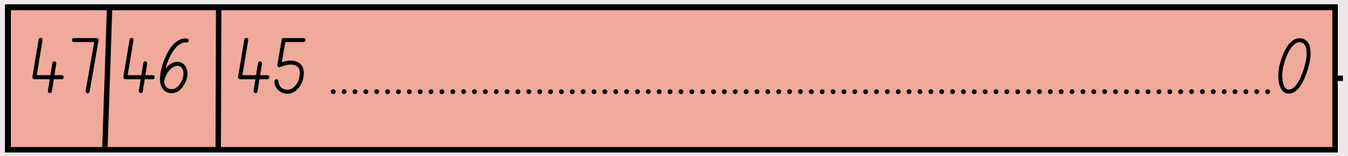
```
   1.1100110101
 x 1.1010001
-------------------
       1100110101
     0000000000x
    0000000000xx
   0000000000xxx
  1100110101xxxx
0000000000xxxxx
-------------------
       .......
       .......

.....00101
```

48-Bit Product

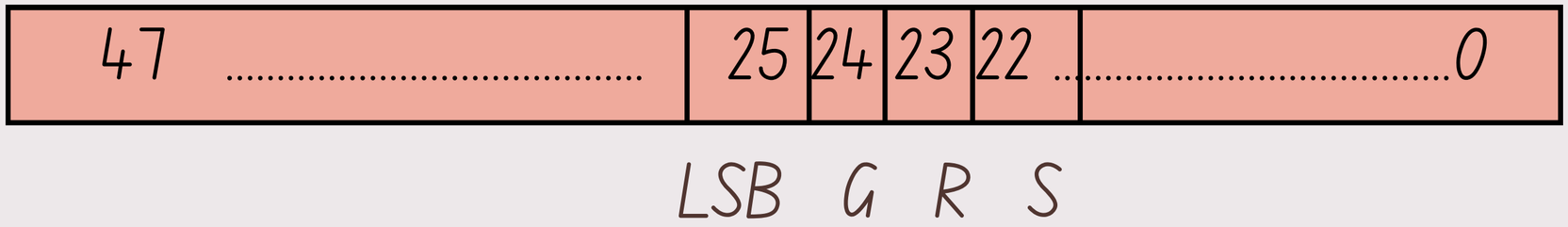
- Store the product in a row of a 2-D array consequently.
- Shift left by 1 bit for every new stage of multiplication.
- Sum the coloumn elements with the same indices by propagating the carry.

HOW TO NORMALIZE?



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

ROUNDING



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

In very few cases round carry is generated!

EXPONENT FIELD

1) Add A exponent , B exponent and other propagated carries.

2) If the sum equals to -126???
Exponent field would become 8'b0, a denormalized product would generate!

3) If the sum with bias exceeds 254???
Either NaN(255) or Overflow

4) If the sum is less than -126???
Underflow

VERIFICATION PLAN

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

VERIFICATION PLAN

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.

VERIFICATION PLAN

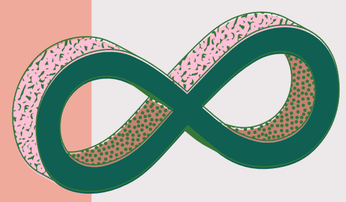
Step 4: Coverage

- 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.

.....Bugs on the way

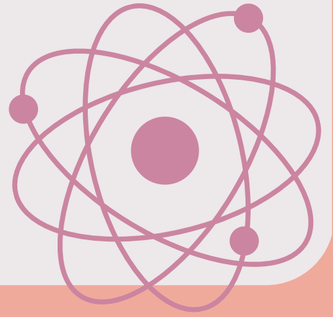
BUGS BUGS BUGS...

- 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.
- When negative powers are added together and are compared to a RHS value there is unexpected behavior (We added bias on both sides of comparison operator)



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.



THANK YOU!