CSE 306
ASSIGNMENT 2
FLOATING POINT ADDER SIMULATION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

LAB SUBSECTION: A2

LAB GROUP: 5

MEMBERS:

1905037

1905039

1905040

1905041

1905058

SUBMISSION DATE: 08/01/2023

Introduction

Floating point numbers are numbers that contain floating decimal points. They are used to represent real numbers in computing. For us to perform arithmetic addition/subtraction operation on floating point numbers, we need a circuit known as a Floating Point Adder (FPA). A Floating Point Adder performs addition/subtractions on floating point numbers in their binary representation. The IEEE-754 standard describes a standard binary format representation way of floating point numbers. The binary bits are stored as below:

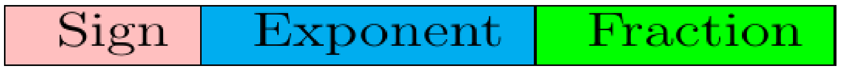
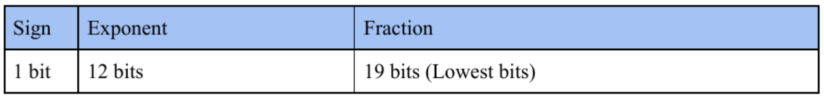


Figure 1: Floating Point Number Representation

Problem Specification

To design a floating point adder circuit which takes two signed floating point numbers as inputs and provides their sum, another floating point as output. The floating points are of 32-bit and thus represented as follows:



Sign	Exponent	Fraction
1 bit	12 bits	19 bits (Lowest bits)

Figure 2: Assigned Floating Point Number Representation

Algorithm Flowchart

The algorithm flowchart for Floating Point Addition is shown in Figure 3:

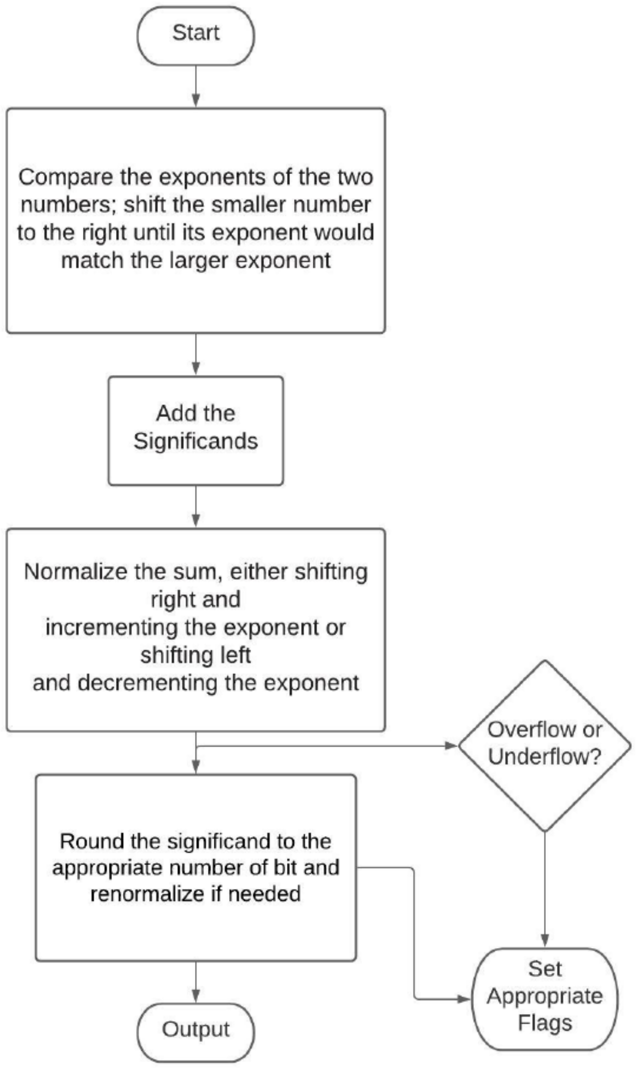

Start
Compare the exponents of the two numbers; shift the smaller number to the right until its exponent would match the larger exponent
Add the Significands
Normalize the sum, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow or Underflow?
Round the significand to the appropriate number of bit and renormalize if needed
Set Appropriate Flags
Output

Figure 3: Algorithm

High Level Block Diagram

A very high-level block diagram of the floating point adder is shown in Figure 4:

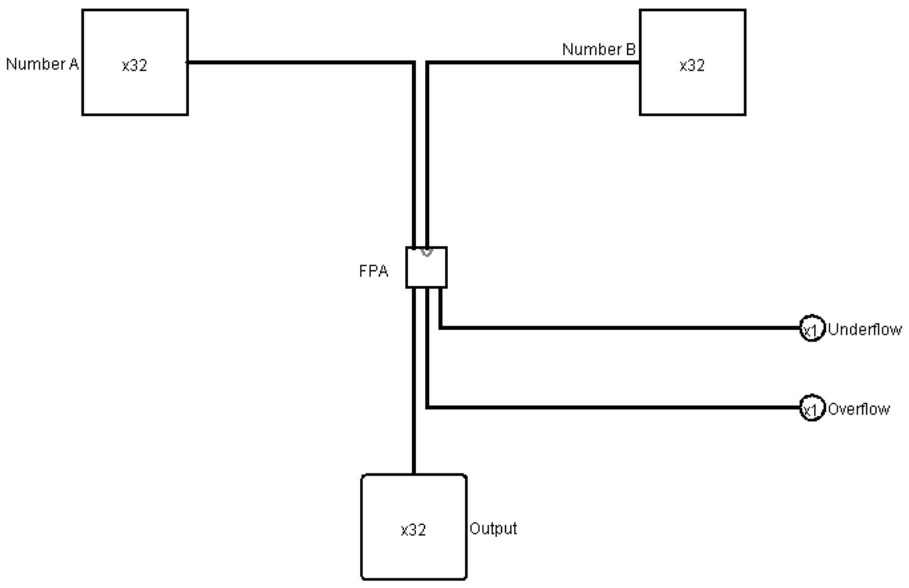


Figure 4: High Level Block Diagram

Floating Point Adder Block Diagram

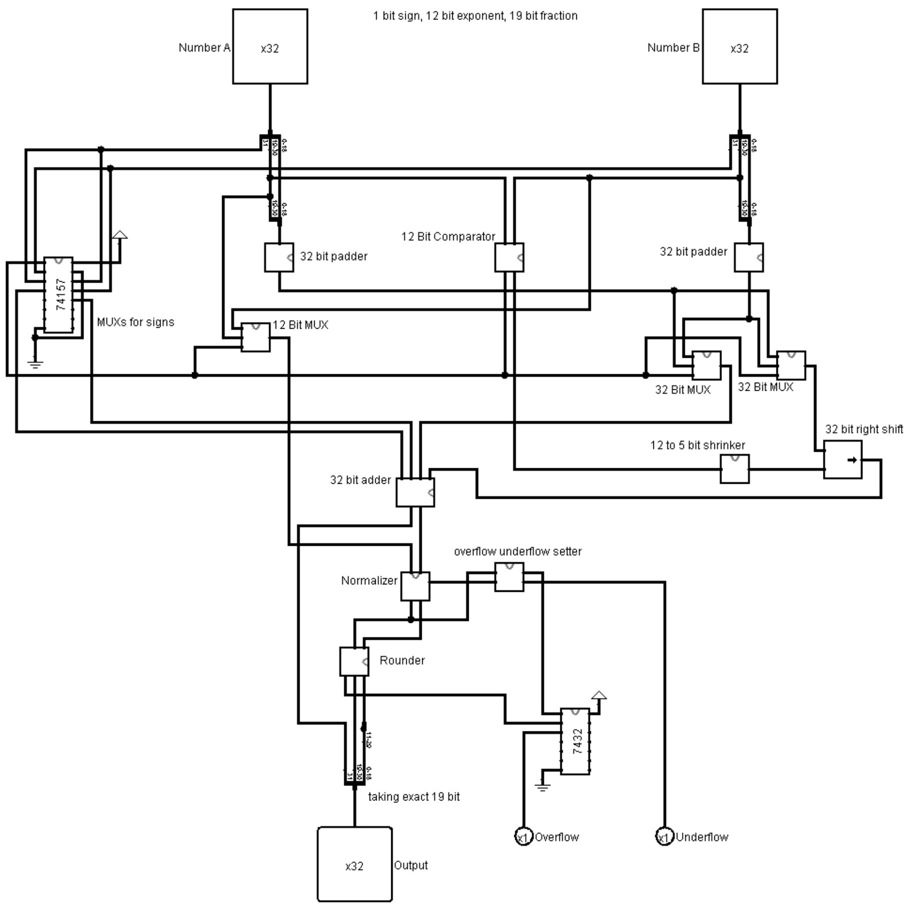


Figure 5: Floating Point Adder

Overview of the Components

12-bit Comparator

The Compare block compares the exponents and outputs the absolute value of their difference. It contains a 12-bit subtractor whose output B minus A,is sent to a MUX both directly and in negated form. The 12-bit negator performs 2's complement on its input. If $A > B$, the borrow out is 1 and the MUX outputs the negated value of subtractor to get the absolute difference. Otherwise, if B greater than A, the direct value is output.

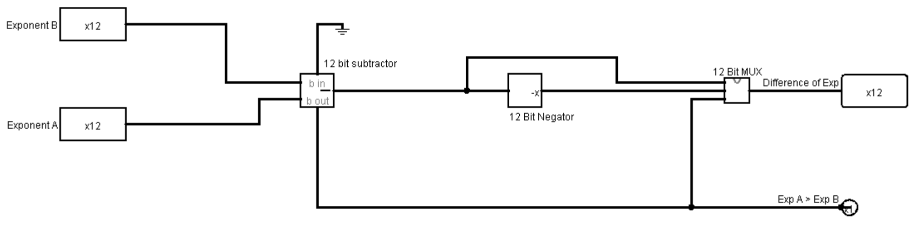


Figure 6: 12-bit Comparator

Padders

Here we used two types of padders. The 32-bit padder converts the 19-bit fraction to 32-bit significand. On the other hand the 12-bit padder converts the 5-bit output of set bit finder to 12-bit to be the input of a 12-bit MUX in the normalizer block.

32-bit Padder

The padder converts the 19-bit fraction to a 32-bit significand as follows:

$$Significand = 0X\ 19\ bit\,fraction\ 00000000000$$

The 0X padded to the left represents two binary digits to the left of the binary point. Here X is 1 if the exponent is non-zero, otherwise X is 0. These two extra binary bits make the normalization hardware much simpler. The 11 zeros padded to the right allows us to perform rounding later on. While padding with three extra bit to the right would have been enough, we store 11 extra bits because these reduce the loss of precision during shifting and makes calculation easier.

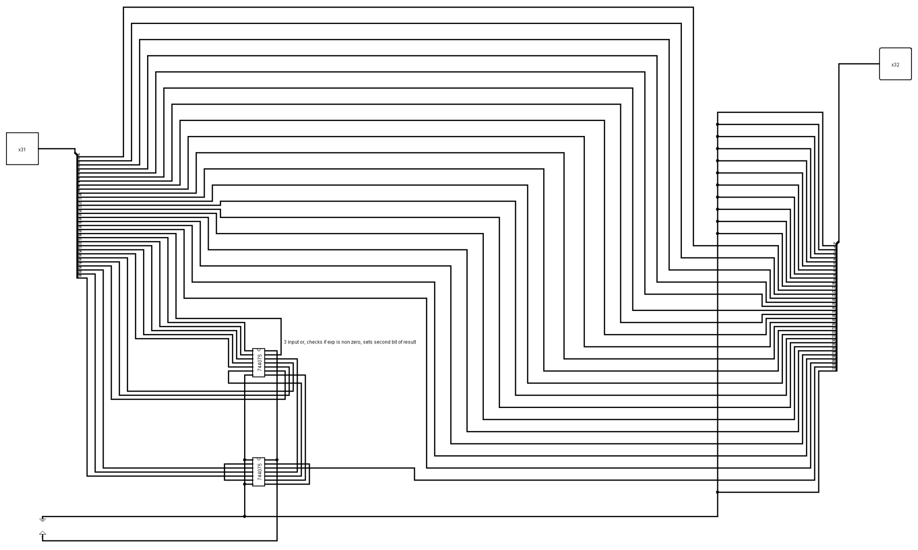

x31
x32
3 input or, checks if exp is non zero, sets second bit of result
744075
744075

Figure 7: 32-bit Padder

12-bit Padder

This is used in the normalizing hardware. It converts the 5-bit output from the set bit finder to 12-bit for further usage as an input of a 12-bit MUX.

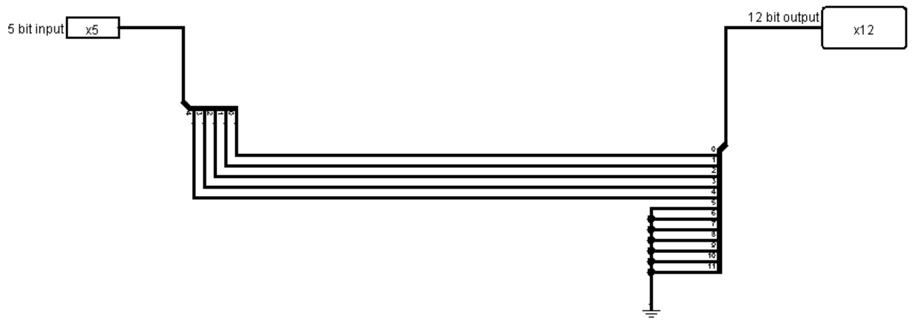


Figure 8: 12-bit Padder

Shrinker

This component shrinks the 12-bit exponent to 5-bit to use as the shift amount of the Logisim provided 32-bit logical shifter. We take the least significant 5 bit as is, if higher bits are 0, otherwise set 11111 as output.

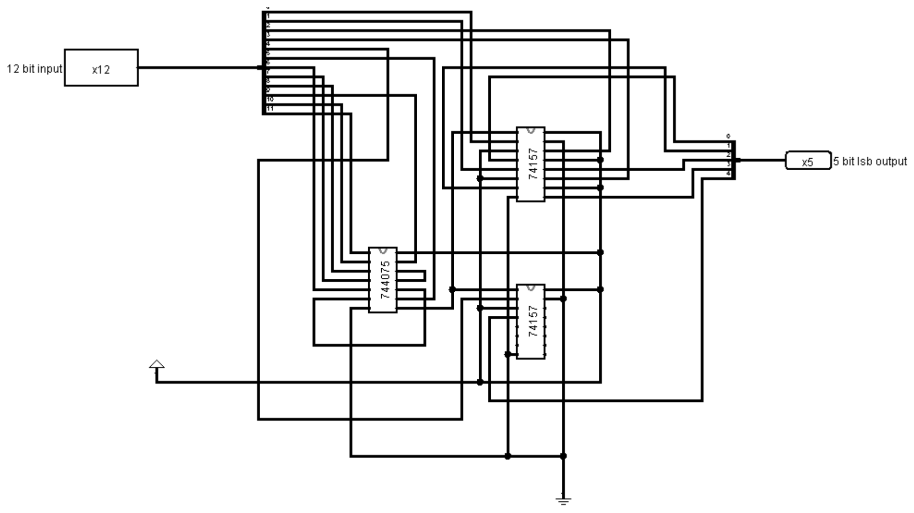


Figure 9: 5 Bit Shrinker

Series Of MUXs

We have a series of MUXs(12 bit and 32 bit) each of which use the (Exponent A > Exponent B) output from the Compare Block as their selection bit. These MUXs are used as follows:

- One 12-bit MUX to select the larger Exponent
- One 32-bit MUX to select the Significand with the larger Exponent and another such MUX to select its corresponding Sign
- One 1-bit MUX to select the Significand with the smaller Exponent and another to select its corresponding Sign

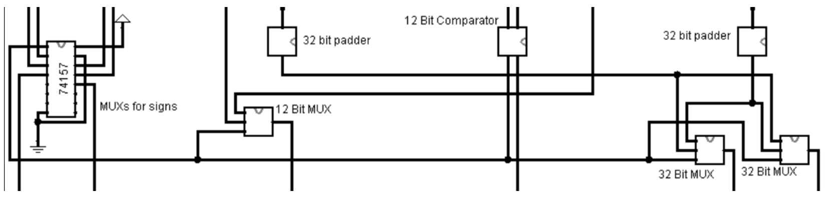


Figure 10: Series of MUXs

12 Bit MUX

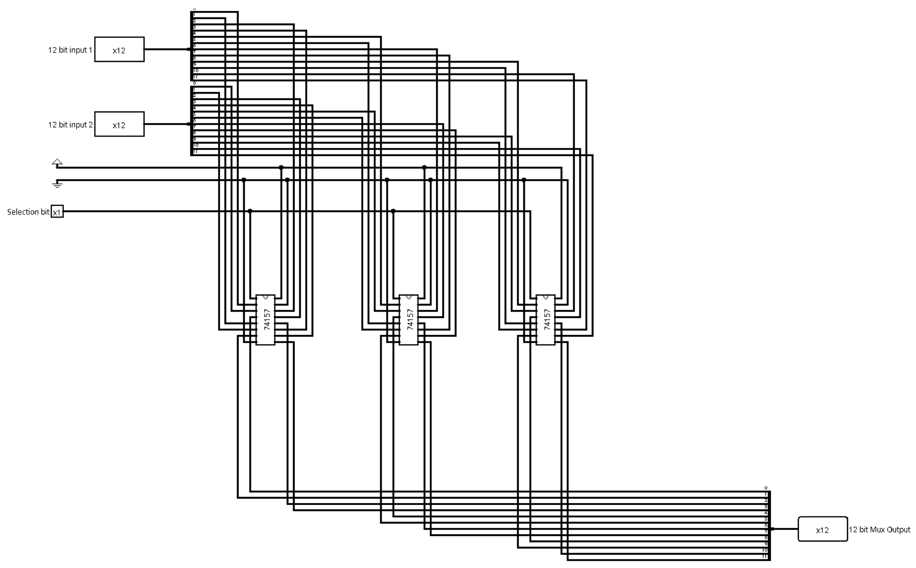


Figure 11: 12 bit MUX

32 Bit MUX

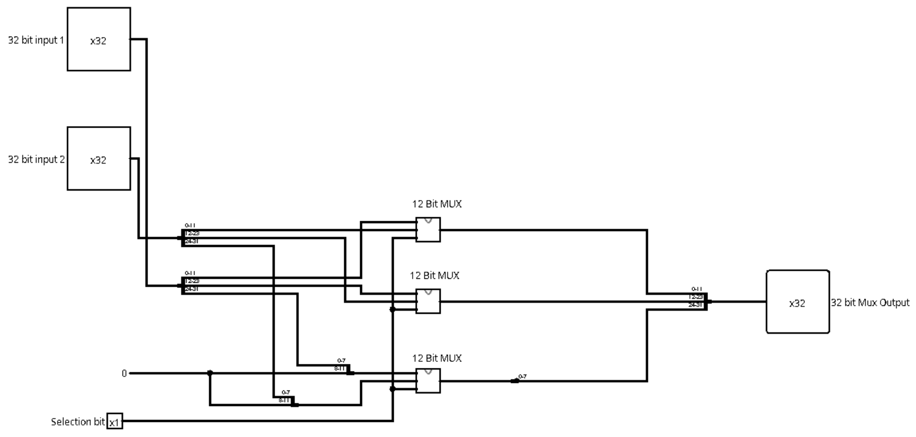


Figure 12: 32 bit MUX

Shifter:

We want the exponent part of both the numbers to be the same before they are added. For this purpose the shifter is used which right-shifts the significand of the number with the smaller exponent. The shift amount is determined by the absolute value of the difference between the exponents. We are not providing any diagram showing the internal implementation of the shifter because we used the shifter circuit readily provided by Logisim.

Adder

The input to the Adder are the two Significands and their corresponding Signs.The Adder block adds the 32-bit Significands. It contains a 32 bit Comparator, MUXs, 32 bit Negator and a 32 bit Adder.

If Significand 1 >= Significand 2 : A MUX sends Significand 1 directly to the Adder; another MUX sends Significand 2 to a third MUX in both original and negated form. This third MUX sends the negated form of Significand 2 to the adder if the signs are different (XOR of sign bits). Otherwise the original form of Significand 2 is sent to the adder.

If Significand 1 < Significand 2 : A MUX sends Significand 2 directly to the Adder; another MUX sends Significand 1 to a third MUX in both original and negated form. This third MUX sends the negated form of Significand 1 to the adder if the signs are different (XOR of sign bits). Otherwise the original form of Significand 1 is sent to the adder.

Since in both cases the larger value is sent directly to the adder and the smaller one is negated (if signs are different), subtraction produces absolute difference. This is important as we do not need to deal with complements in the answer. Finally, we have a MUX for the two signs. The output of the Comparator is used by a MUX to select the sign of the larger Significand as the output sign.

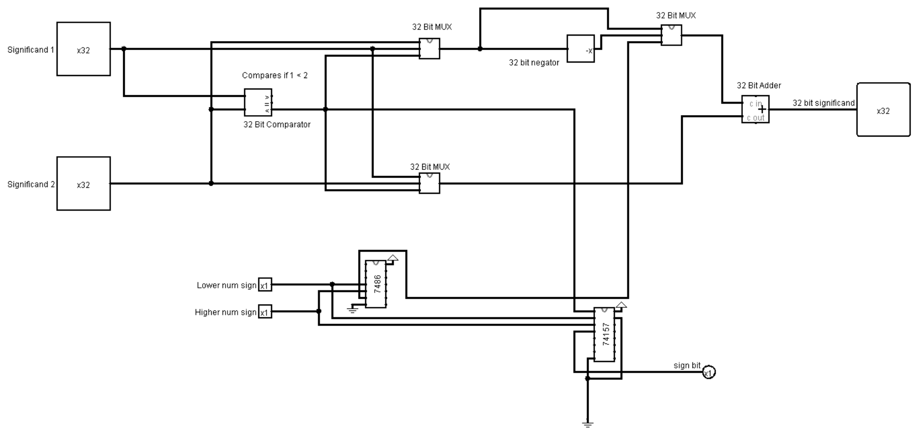


Figure 13: 32-bit Adder

Normalizing Block

In this component the input is a 12-bit Exponent and 32-bit Significand; the output Exponent and Significand are in normalized form if normalization is possible. Otherwise the output maybe a Denormal number, or 0, or infinity. Additionally, another output tells if the Significand had 0-bits only.

The Normalization block makes use of a Set Bit Finder component which outputs the following:

- If any 1 is present in the Significand
- The number of positions the highest set bit needs to be shifted for the output to be in normalized form.

For Normalization, left shift by 0 to 31 bits might be needed, or right shift by 1 bit maybe needed. To simplify the situation we used Left Rotator instead because left rotation by 31 bits is the same as right shift by 1 bit in this case.

If the amount of left shift needed is greater than or equal to the value of the exponent, then the output will be a Denormal number. The Significand is shifted $Exponent - 1$ times to the left and the output Exponent is 000000000000.

If the value of the output Exponent is 0xFFF, then it is a case of overflow. And according to IEEE-754's representation of infinity, the Fraction part of the Significand is made all 0s.

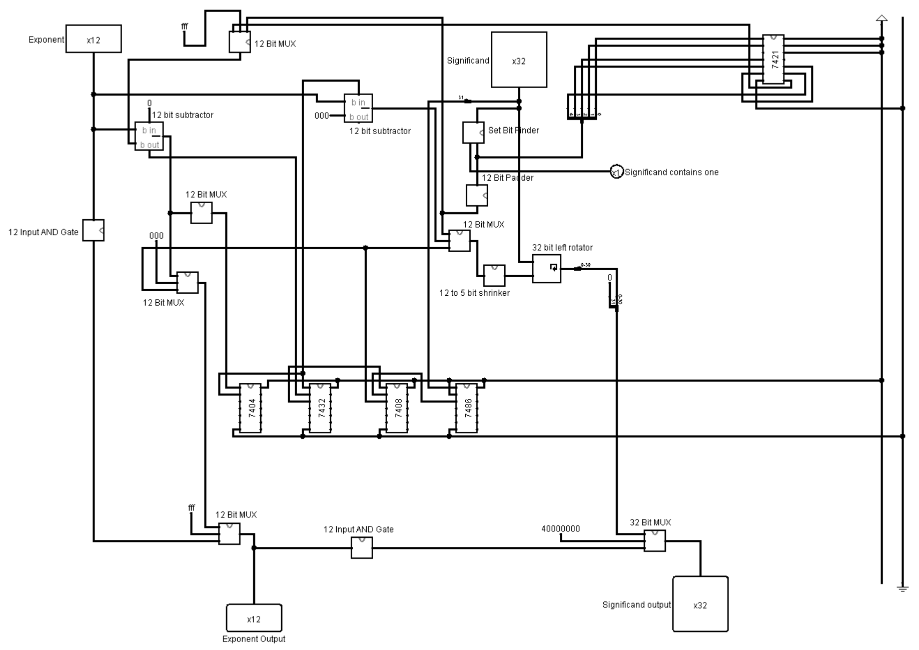


Figure 14: Normalizer

Set Bit Finder

It takes the 32-bit Significand as input and then reverses the input. The index of the lowest set bit is found and 1 is subtracted from it. The resulting value is equal to the number of left rotation needed for the Significand to be converted to Normalized form.

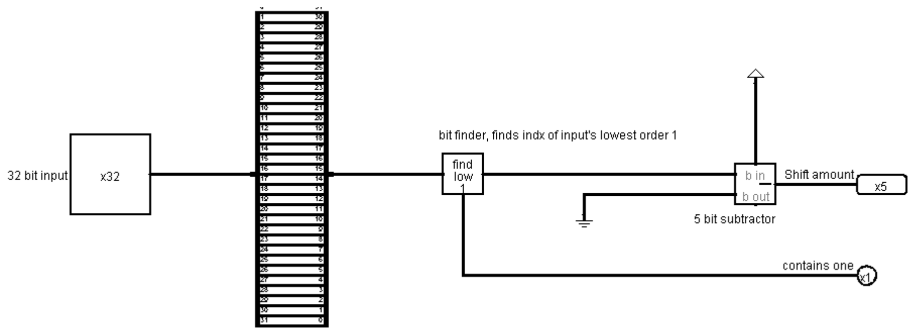


Figure 15: Set Bit Finder

Rounding Circuitry

The rounding block has one component named rounder mini to round the significand and the final rounder to work with the exponent and the significand.

Rounder Mini

Here, the least significand bit(LSB) of significand is used to calculate LSB of a 32-bit significand as $L = G(L + R + S)$ with all other bits set 0. Then it is added to the significand.Here L is the LSB before GR and S is the OR of all bits after GR.

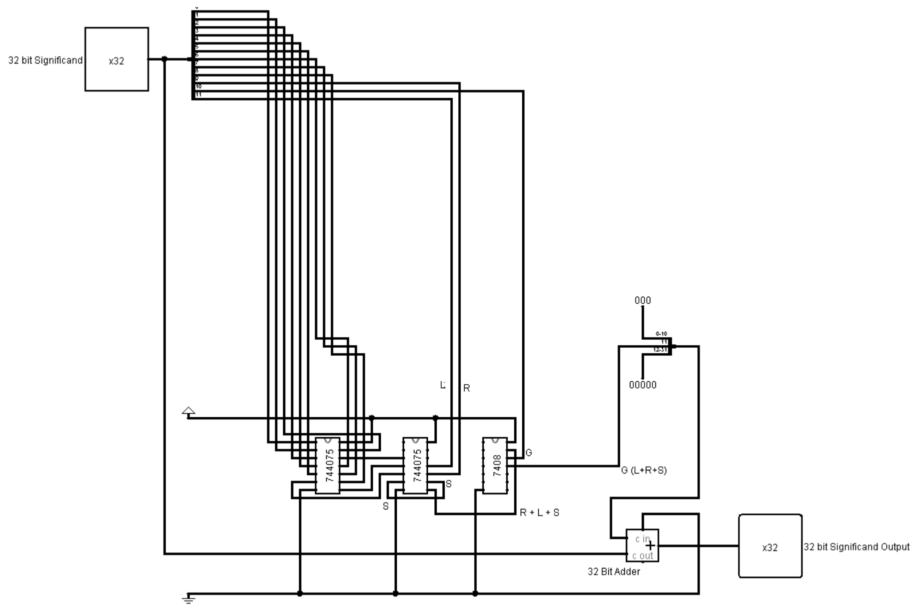


Figure 16: Rounder Mini

Rounder

We have taken the Significand and Exponent as input and given as output the the final Significand, Exponent and Overflow detection. If preliminary rounding causes number to become de-normalized, we right shift the Significand by 1 and add 1 to the Exponent.

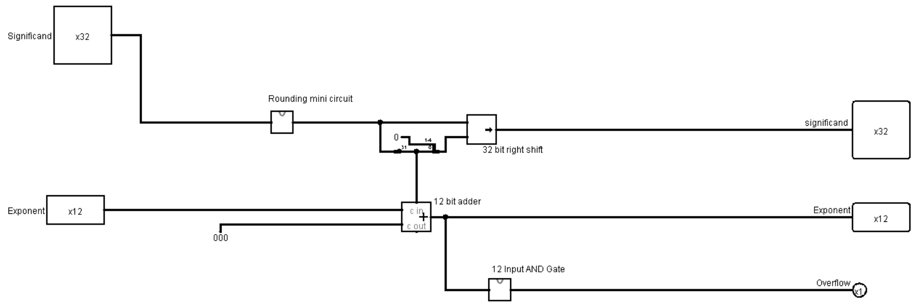


Figure 17: Rounder

Overflow and Underflow Detection

Overflow occurs when all bits of the Exponent are 1. Underflow occurs when all bits of the Exponent are 0 but all bits of the Significand are not 0.

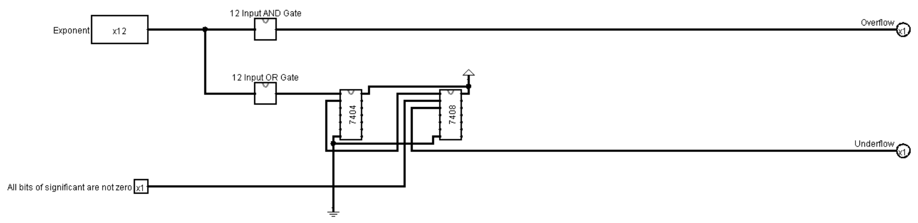


Figure 18: Overflow and Underflow Setter

IC Count

Operation	IC	Quantity
32-bit Right Shift	-	2
32-bit Left Rotate	-	1
32-bit Bit Finder	-	1
32-bit Adder	-	1
32-bit Comparator	-	1
12-bit Adder	-	1
12-bit Subtractor	-	3
12-bit Negator	-	1
5-bit Subtractor	-	1
QUAD 2:1 MUX	74157	79
QUAD 2-AND	7408	3
DUAL 4-AND	7421	9
QUAD 2-OR	7432	1
TRIPLE 3-OR	744075	7
QUAD 2-XOR	7486	2
HEX NOT	7404	2

Simulation Software

Logisim-2.7.1

Discussion

Moreover due to the allowed use of Logisim modules we have mostly used them in our design such as adder, subtractor, comparator etc in our design instead of using third party 32-bit ALUs.

We used some 12-bit AND and OR gates and 12-bit and 32-bit MUXs. These were simulated with ICs and used as needed in main FPA blocs, so the work was simplified.

After rounding the number can become de-normalized. So it must be fed back to the Normalization Hardware. However, this means the process would involve clock pulses. To avoid the use of clock pulses we identified the specific scenario where this problem would occur and handled it separately. This means our circuit remains a combinational one, however this approach does increase IC count slightly.