CSE-306

Floating Point Adder Implementation

Submitted by:

Group: 1

Section: B-2

Members:

Nahian Salsabil - 1705091

Farhana Khan - 1705100

Rittik Basak Utsha - 1705105

Muhtasim Noor - 1705108

Introduction

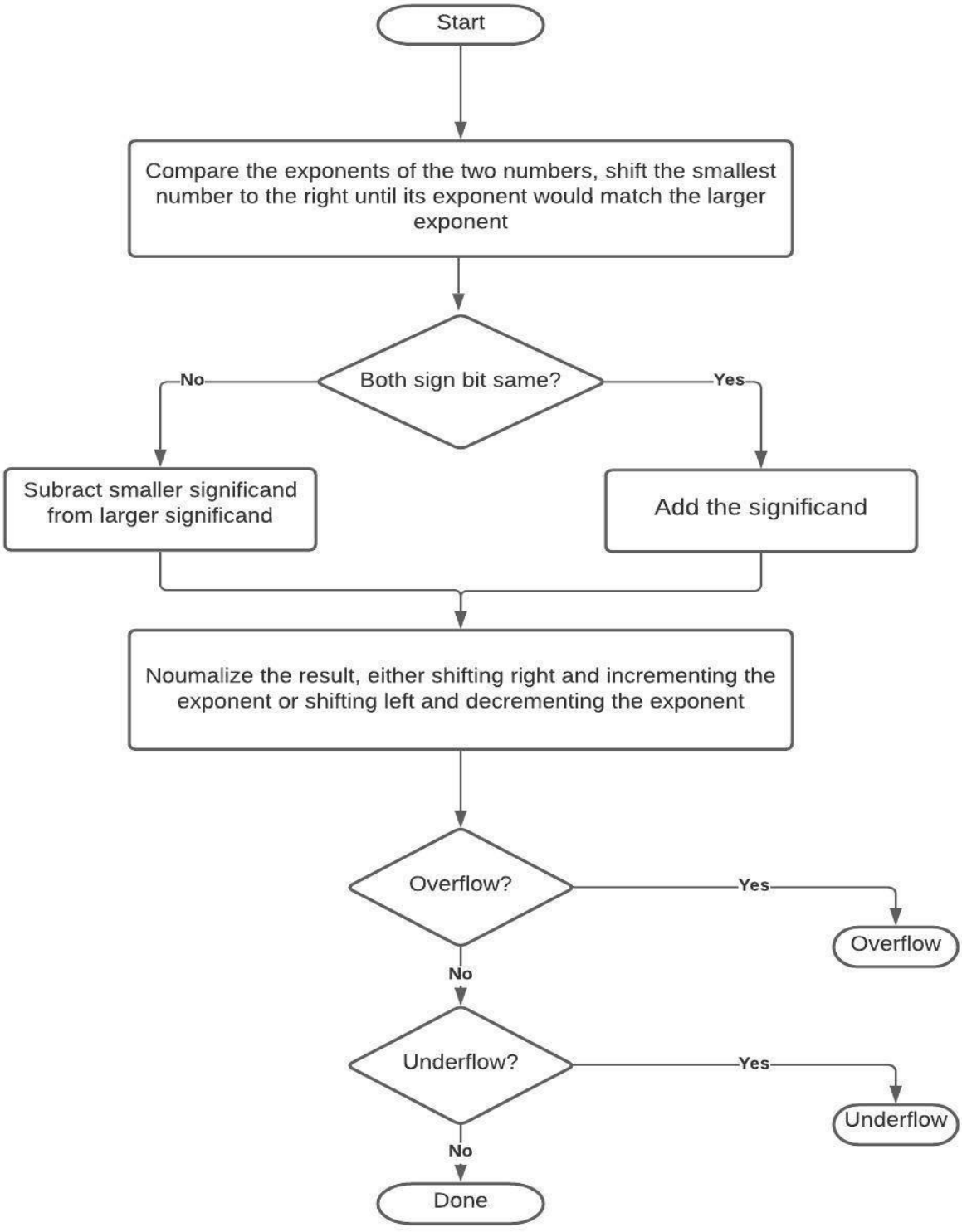
In this assignment, our goal is to design and implement a Floating-Point Adder. Floating points in computer memory are stored using binary representation following a certain convention. Adding floating points is different from adding integers.

Problem Specification

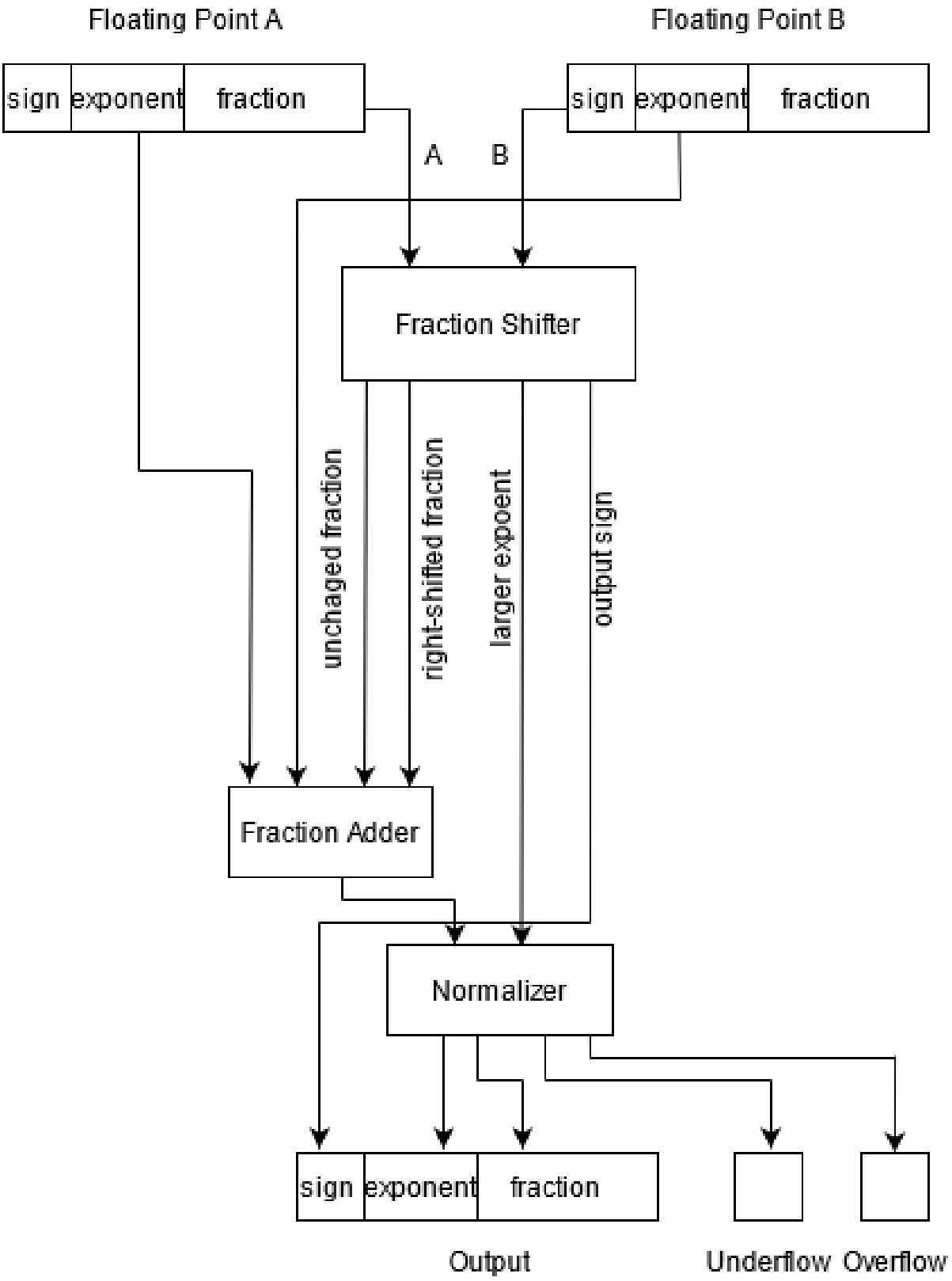
Given two floating points we have to design a circuit that adds them and gives another floating point as result. We also have to implement the circuit using simulation software. Floating points in this task will be represented by the following format:

Sign **1 bit**	*Exponent* **4 bit**	*Fraction* **11 bit**

Flow Chart

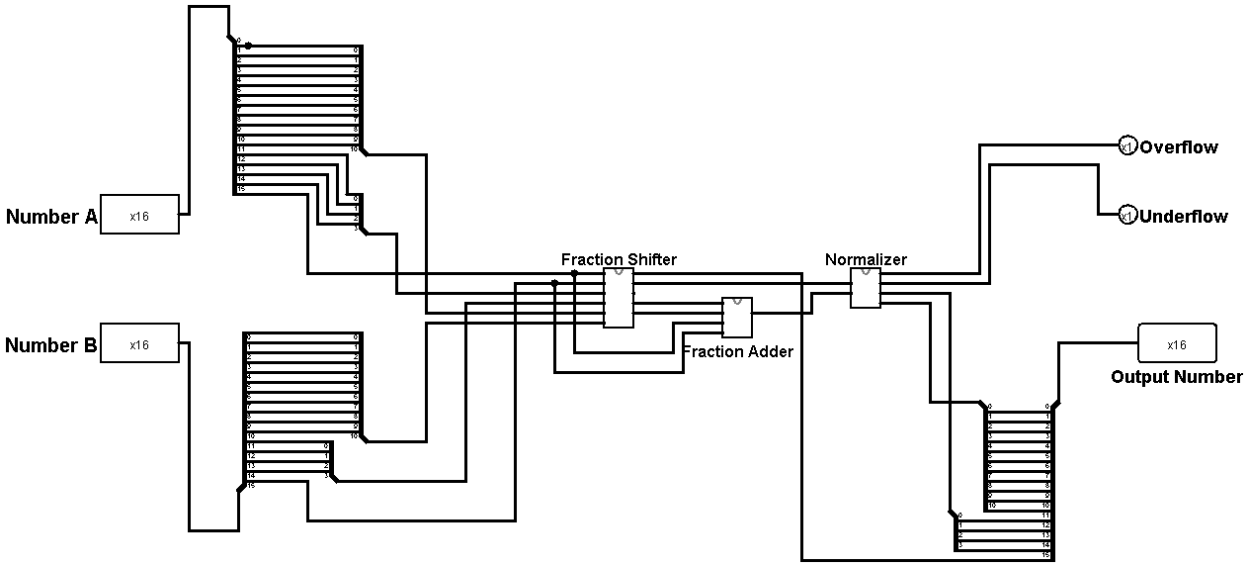

Start
Compare the exponents of the two numbers, shift the smallest number to the right until its exponent would match the larger exponent
Both sign bit same?
No
Yes
Subract smaller significand from larger significand
Add the significand
Noumalize the result, either shifting right and incrementing the exponent or shifting left and decrementing the exponent
Overflow?
Yes
Overflow
No
Underflow?
Yes
Underflow
No
Done

Block Diagram

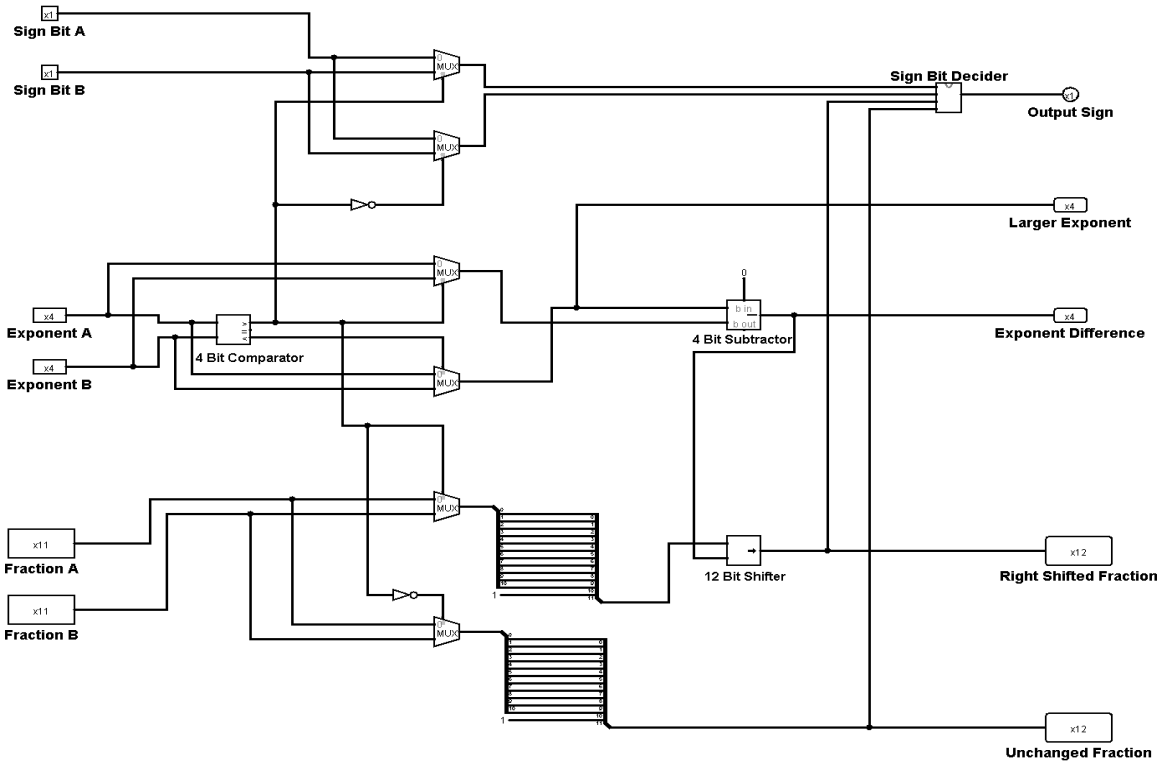


Circuit Diagrams:

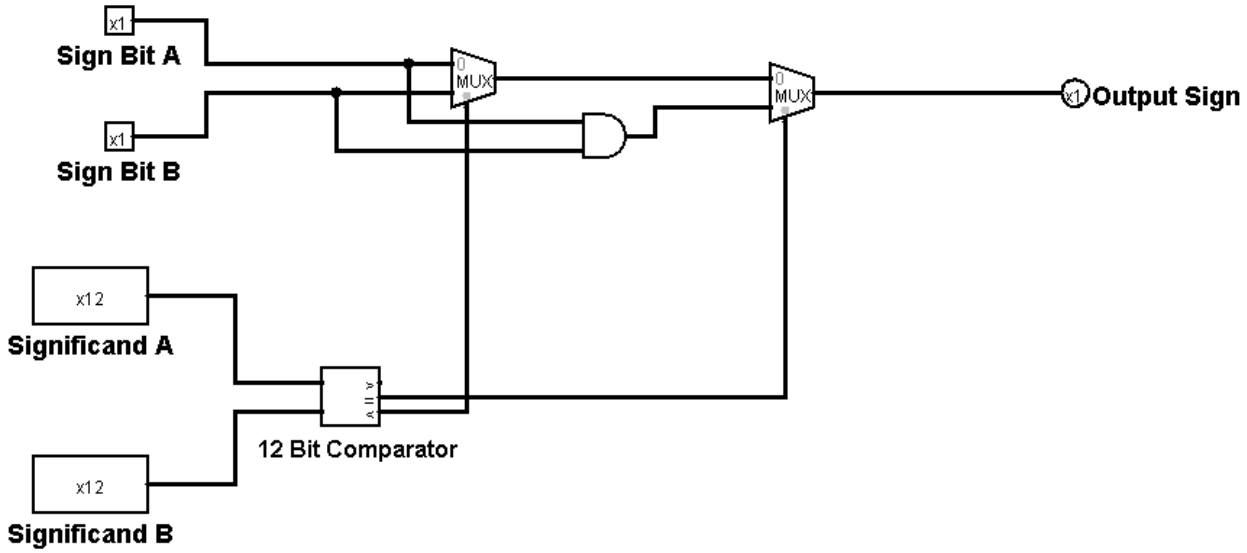
Fp Adder:



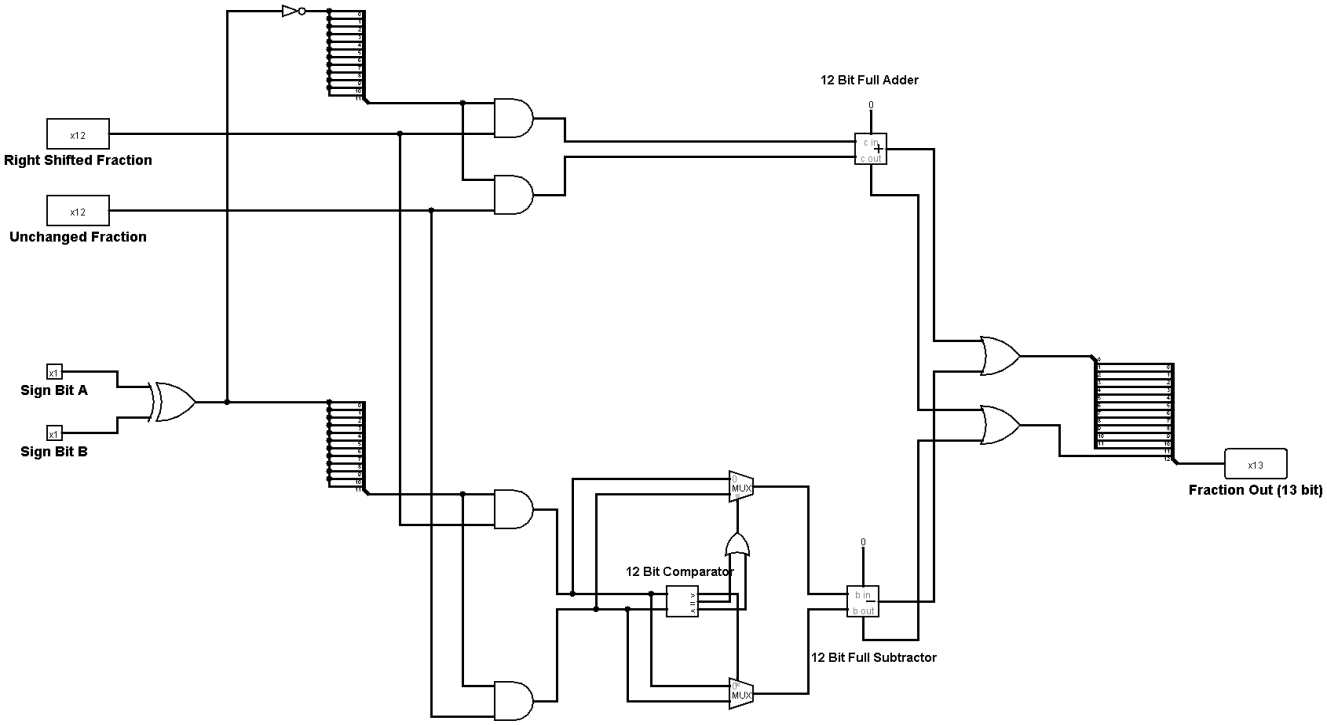
Fraction Shifter:



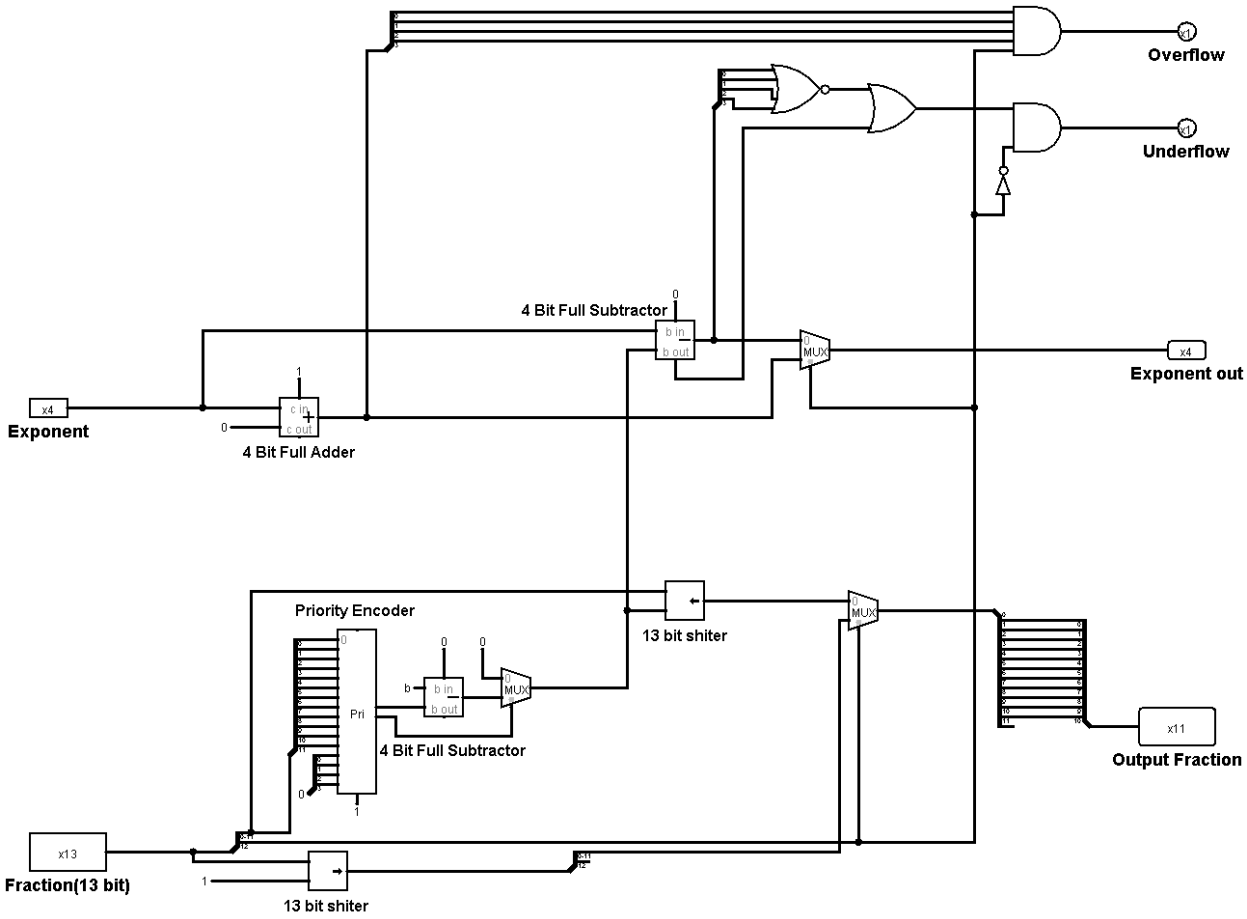
Sign Bit Decider:



Fraction Adder:



Normalizer:


Overflow
Underflow
4 Bit Full Subtractor
b in
b out
MUX
x4
Exponent out
x4
Exponent
c in
c out
4 Bit Full Adder
Priority Encoder
13 bit shiter
Pri
4 Bit Full Subtractor
x11
Output Fraction
x13
Fraction(13 bit)
13 bit shiter

IC Count

IC	Name	Count
7432	OR Gate	4
7408	AND Gate	7
7486	XOR Gate	1
7402	NOR Gate	1
7404	NOT Gate	3
74157	2x1 Mux	13
7485	Magnetude Comparator	3
7483	4-bit Full Adder	2
74148	Priority Encoder	1
	Shifter	3
	4-bit Full Subtractor	2

Simulator Used

We have used “Logisim 2.7.1” to implement and simulate our design of Floating Point Adder

Discussion

We tried to minimize the use of ICs in this design and it was designed to be as simple as we could think.

We used a priority-encoder to find the most significant 1.

The normalized result was truncated to 11 bit fraction. After normalization overflow (exponent value above 14) and underflow (exponent value below 1) was detected.