DEPARTMENT OF ELECTRICAL ENGINEERING, IIT BOMBAY

Master of Technology, Electronic Systems

PROJECT REPORT

ON

Design of 32-bit Floating Point Adder & Multiplier

July, 2021

AUTHOR NAME

Kanak Vijay (203070050)

Deepak Chand (203070060)

Theory:

Floating Point Representation: IEEE 754 specifies 32-bit floating point representation as

1-bit (Sign)	8-bit(exponent)	23-bit(mantissa)

With 8-bit exponent we can represent in the range of 0-255.

The exponent value has a bias of 127. It means the exponent value will be between -126 (00000000(2) and +127 (11111110(2) being zero at the value (01111111(2). The mantissa value is 23 bits long but it contains an implicit bit.

Design of 32-bit Floating Point adder:

We have defined registers of size 23-bit(mantissa) + 126-bit (right shift) + 1-bit (implicit bit) + 127-bit (left shift).

276 150	149	148 126	125 0

32-bit Floating Point adder is designed using FSM. There are 7 stages in FSM.

1) Ready: In this stage defined registers is initialized to 0.

2) Start: In this stage, mantissa of the two numbers are stored in two registers at location bit 148-126 depending on greater and smaller number & exponent of both numbers is calculated by subtracting bias 127.

3) Shift stage: in this data stored in registers are shifted left or right depending on exponent.

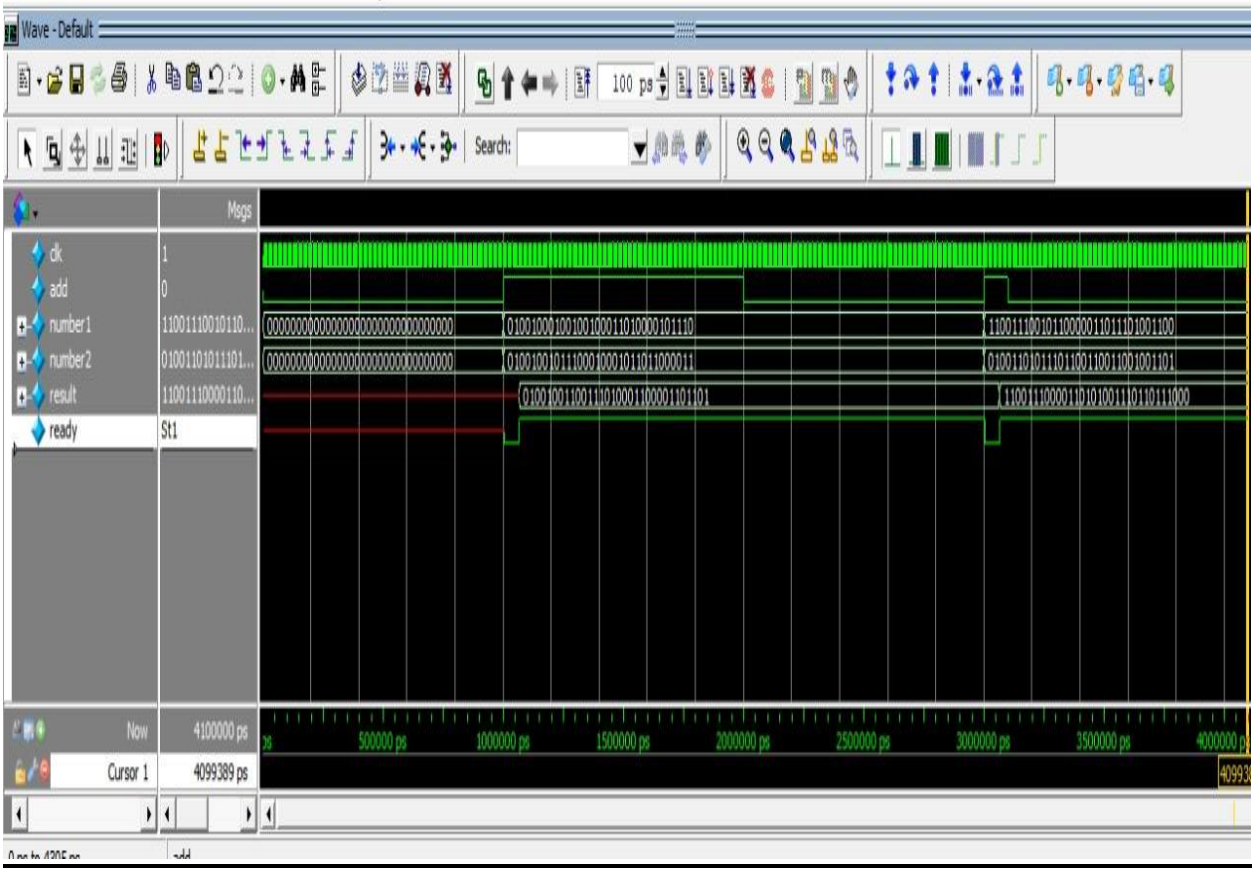
4) Add/sub stage: If both the no are of a similar sign, then numbers are added otherwise subtracted.

5) Detect position: Detection of the leftmost position where the bit is 1, is evaluated in this stage.

6) Write stage: In this stage final mantissa & exponent is calculated.

7) Result: Ready status flag is high.

Modelsim Simulation Result :



Floating Point Multiplier

Sign (S)	Exponent (E)	Mantissa (F)

Representation of a floating point number : $(-1)^S$ X F X $2^{E'}$

Where E'=E-Bias

IEEE 754 Floating point standard

Single Precision Floating Point Number Representation

- It is a representation of 32 bit.
- It has 1- sign Bit, 8- Exponent, 23- Fraction bit or Mantissa.

Sign (1)	Exponent(8)	Mantissa(23)

Double Precision Floating Point Number Representation

- It is a representation of 64 bit.
- It has 1- sign bit, 11- Exponent, 52- fraction bit or mantissa.

Sign (1)	Exponent(11)	Mantissa(52)	

IEEE 754 uses biased representation for the exponent:

So Exponent (E) = E' + Bias

Here E = Biased exponent , E' = Unbiased exponent

Bias = 127 ; single precision FP number

Bias = 1023; double precison FP number

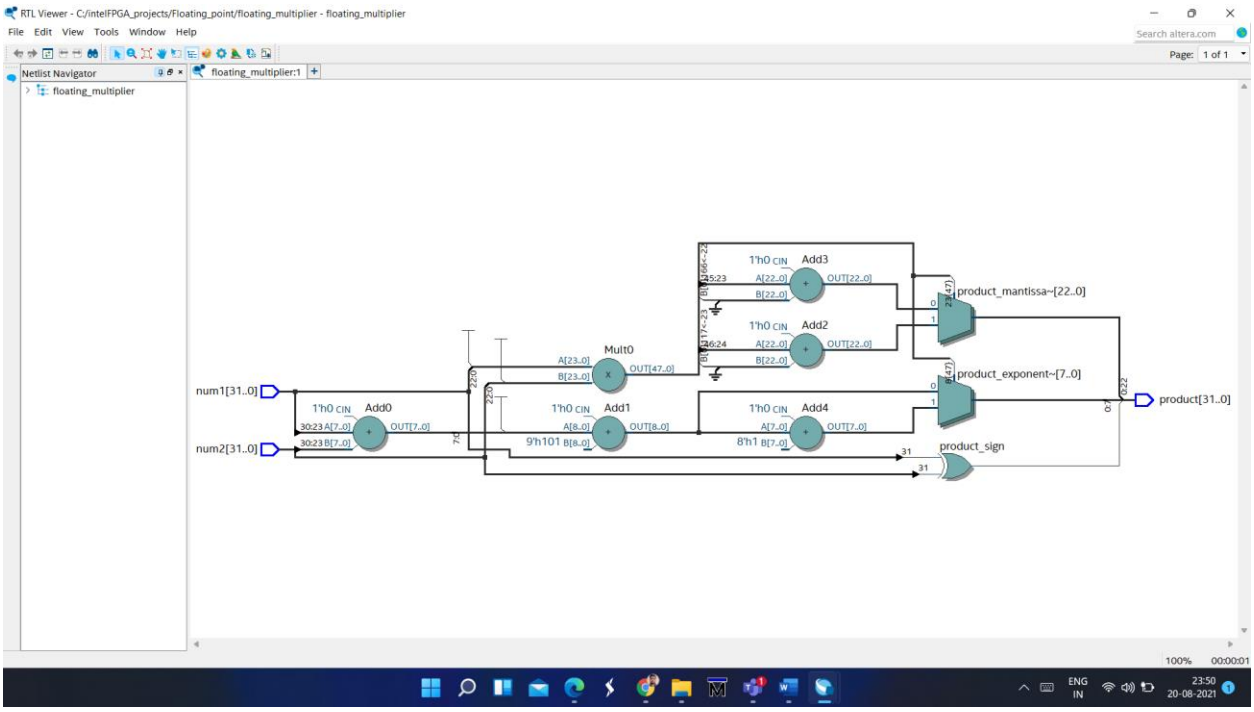
Example :

We have two Floating point number A and B So Multiplication of A and B will be

$A * B = (-1)^{(S1 \text{ xor } S2)} \times (F1 \times F2) \times 2^{(E1'+E2')}$

Process to calculate multiplication of A and B

1. Find Sign bit of the final product using xor of sign bit of the A and B.
2. Find Exponent value of according to precision representation.
3. Multiply fraction part of the number.



From this RTL viewer we can observe that it is a fully combinational circuit because in this circuit all components are combinational.

VERILOG CODE OF MULTIPLIER :

```
module floating_multiplier(num1,num2,product);

input [31:0] num1,num2;

output [31:0] product;

wire [7:0] num1_exponent,num2_exponent,exponent,product_exponent ;

wire [22:0] num1_mantissa,num2_mantissa,product_mantissa ;

wire num1_sign,num2_sign,product_sign ;

wire [47:0] prod_mantissa;

wire [7:0] prod_exponent;

assign num1_exponent = num1[30:23];

assign num1_mantissa = num1[22:0];

assign num1_sign = num1[31];

assign num2_exponent = num2[30:23];

assign num2_mantissa = num2[22:0];

assign num2_sign = num2[31];

assign product_sign = num1_sign ^ num2_sign;

assign exponent = num1_exponent + num2_exponent;

assign prod_exponent = exponent - 8'd127;

 assign prod_mantissa = {1'b1, num1_mantissa} * {1'b1, num2_mantissa};

assign product_mantissa = prod_mantissa[47] ? prod_mantissa[46:24] + prod_mantissa[23] :
prod_mantissa[45:23] + prod_mantissa[22];

// assign prod2_mantissa = prod_mantissa[47:24] + {23'b0, 1'b0};

// assign product_mantissa = prod2_mantissa[22:0];

assign product_exponent = prod_mantissa[47] ? prod_exponent + 8'd1 : prod_exponent;

assign product = {product_sign, product_exponent, product_mantissa};
```

```
initial begin
    $monitor("%b",prod_mantissa);
End
Endmodule
```

Testbench for Design :

```
module main;
reg [31:0] num1,num2;
wire [31:0] product;
floating_multiplier dut (num1,num2,product);
 initial
   begin
   //   num1 = 32'b10111110100110011001100110011010;
   //   num2 = 32'b01000011111111010001000000000000;
     num1 = 32'b10111110100110011001100110011010;
     num2 = 32'b01000011111111010001000000000000;
     #10;
     $display("%b, %b, %b", num1, num2, product);
     $finish ;
   end
endmodule
```

Modelsim Simulation Result :