CADD JOB ASSIGNMENT

NAME:ARAVIND AKSHIT NITTALA

SRN:PES2UG23EC026

SECTION:3RD SEM A SECTION

1. Floating Point Multiplier

(a) Steps for 32-bit Floating Point Multiplication

Extract Fields: Extract the sign, exponent, and mantissa from both 32-bit floating-point inputs.

Sign Calculation: XOR the sign bits of the two inputs to get the sign of the result.

Exponent Calculation: Add the exponents of the two inputs and subtract the bias (127 for single precision).

Mantissa Calculation: Multiply the mantissas of the two inputs. Normalize the result if necessary.

Normalization: Adjust the exponent and mantissa to ensure the mantissa is in the correct range.

Rounding: Apply round toward zero (truncate).

Assemble Result: Combine the sign, exponent, and mantissa to form the final 32-bit floating-point result.

System Verilog design code for floating point multiplier

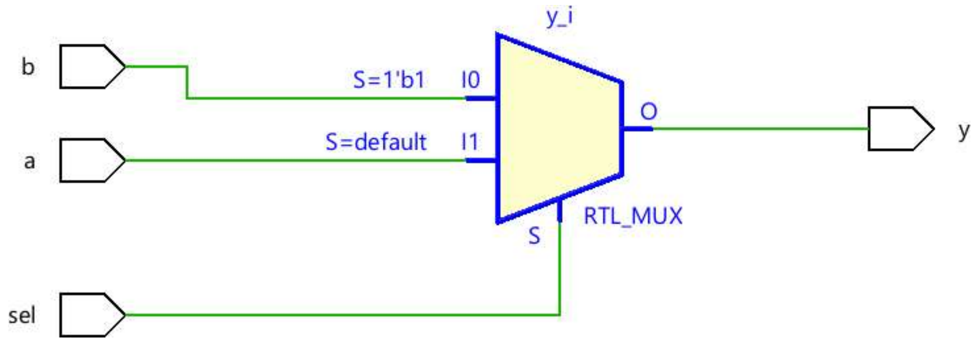
```
 module fp_multiplier (
input [31:0] a,
   input [31:0] b,
 output [31:0] result
 );
   wire sign_a, sign_b, sign_result;
   wire [7:0] exp_a, exp_b, exp_result;
   wire [23:0] mant_a, mant_b, mant_result;
   wire [47:0] mant_mult;
   wire [7:0] exp_sum;

   assign sign_a = a[31];
   assign sign_b = b[31];
   assign exp_a = a[30:23];
   assign exp_b = b[30:23];
   assign mant_a = {1'b1, a[22:0]};
   assign mant_b = {1'b1, b[22:0]};

   assign sign_result = sign_a ^ sign_b;
   assign exp_sum = exp_a + exp_b - 8'd127;
   assign mant_mult = mant_a * mant_b;

   assign mant_result = mant_mult[47] ? mant_mult[46:24] : mant_mult[45:23];
   assign exp_result = mant_mult[47] ? exp_sum + 1 : exp_sum;

   assign result = {sign_result, exp_result, mant_result[22:0]};
 endmodule
```

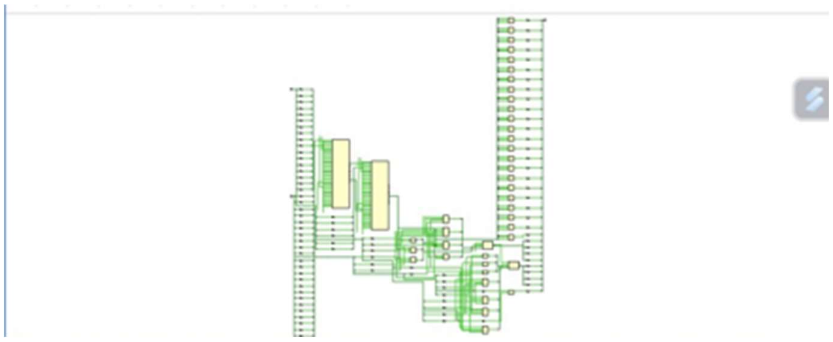
```
input [31:0] a,
   input [31:0] b,
 output [31:0] result
 );
   wire sign_a, sign_b, sign_result;
   wire [7:0] exp_a, exp_b, exp_result;
   wire [23:0] mant_a, mant_b, mant_result;
   wire [47:0] mant_mult;
   wire [7:0] exp_sum;
```

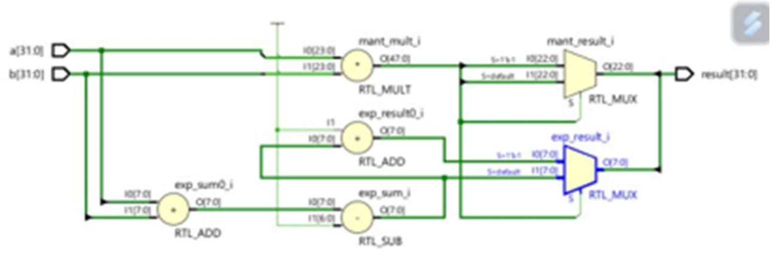
```
  assign sign_a = a[31];
  assign sign_b = b[31];
  assign exp_a = a[30:23];
  assign exp_b = b[30:23];
  assign mant_a = {1'b1, a[22:0]};
  assign mant_b = {1'b1, b[22:0]};

  assign sign_result = sign_a ^ sign_b;
  assign exp_sum = exp_a + exp_b - 8'd127;
  assign mant_mult = mant_a * mant_b;

  assign mant_result = mant_mult[47] ? mant_mult[46:24] : mant_mult[45:23];
  assign exp_result = mant_mult[47] ? exp_sum + 1 : exp_sum;

  assign result = {sign_result, exp_result, mant_result[22:0]};
endmodule
```

Testbench

```
module test_fp_multiplier;
  logic [31:0] a, b;
  logic [31:0] result;

  fp_multiplier uut (
    .a(a),
    .b(b),
    .result(result)

  );

  initial begin
    $display("Time\t a\t\t\t b\t\t\t result");
    $monitor("%0t\t %h\t %h\t %h", $time, a, b, result);

    // Test cases
    a = 32'h3f800000; // 1.0
    b = 32'h40000000; // 2.0
    #10;

    a = 32'h40400000; // 3.0
    b = 32'h40800000; // 4.0
    #10;

    a = 32'h3f800000; // 1.0
    b = 32'h3f800000; // 1.0
    #10;

    a = 32'h3f800000; // 1.0
    b = 32'h00000000; // 0.0
    #10;

    a = 32'h3f800000; // 1.0
    b = 32'hbf800000; // -1.0
    #10;

    $finish;
 end
 endmodule
```


y_i
b
S=1'b1
I0
a
S=default
I1
O
y
S
RTL_MUX
sel




a[31:0]
b[31:0]
mant_mult_i
I0[23:0]
I1[23:0]
O[47:0]
RTL_MULT
mant_result_i
S=1'b1
I0[22:0]
S=default
I1[22:0]
O[22:0]
S
RTL_MUX
result[31:0]
exp_result0_i
I1
I0[7:0]
O[7:0]
RTL_ADD
exp_result_i
S=1'b1
I0[7:0]
S=default
I1[7:0]
O[7:0]
S
RTL_MUX
exp_sum0_i
I0[7:0]
I1[7:0]
O[7:0]
RTL_ADD
exp_sum_i
I0[7:0]
I1[6:0]
O[7:0]
RTL_SUB

32-bit prefix adder

Input A -> Input B -> Generate & Propagate -> Prefix Tree -> Sum Calculation -> Output Sum.

System Verilog design code for 32-prefix adder

```
module prefix_adder (
  input [31:0] a,
  input [31:0] b,
  output [31:0] sum
);
  wire [31:0] g, p, c;

  assign g = a & b;
  assign p = a ^ b;

  assign c[0] = 0;
  genvar i;
  generate
    for (i = 1; i < 32; i = i + 1) begin
    assign c[i] = g[i-1] | (p[i-1] & c[i-1]);
    end
  endgenerate

  assign sum = p ^ c;
endmodule
```

testbench

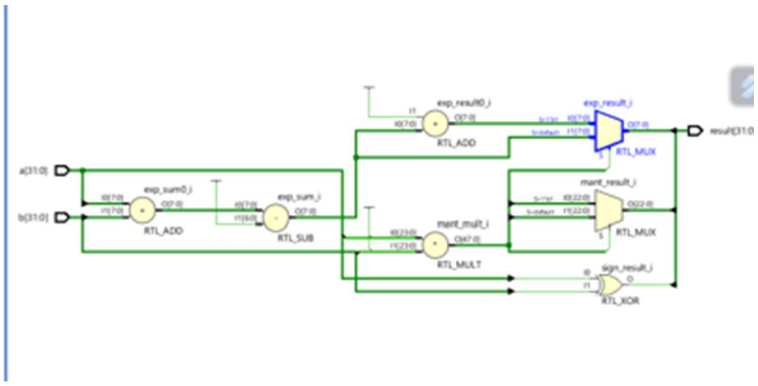
```
module test_prefix_adder;
  logic [31:0] a, b;
  logic [31:0] sum;

  prefix_adder uut (
    .a(a),
    .b(b),
    .sum(sum)
  );

  initial begin
    $display("Time\t a\t\t\t b\t\t\t sum");
    $monitor("%0t\t %h\t %h\t %h", $time, a, b, sum);

    // Test cases
    a = 32'h00000000; b = 32'h00000000; #10;
    a = 32'h00000001; b = 32'h00000001; #10;
    a = 32'hFFFFFFFF; b = 32'h00000001; #10;
    a = 32'h12345678; b = 32'h87654321; #10;
    a = 32'hAAAAAAAA; b = 32'h55555555; #10;

    $finish;
end
endmodule
```

Delay Calculation

The delay of the 32-bit prefix adder can be calculated based on the number of stages in the prefix tree. Assuming each two-input gate delay is 100 ps, the delay can be estimated by counting the number of stages

pipelined_prefix_adder

```
module pipelined_prefix_adder (
  input clk,
  input [31:0] a,
  input [31:0] b,
  output reg [31:0] sum
);
  reg [31:0] g, p, c;

  always @(posedge clk) begin
    g <= a & b;
    p <= a ^ b;
  end

  always @(posedge clk) begin
    c[0] <= 0;
    for (int i = 1; i < 32; i = i + 1) begin
    c[i] <= g[i-1] | (p[i-1] & c[i-1]);
    end
  end

  always @(posedge clk) begin
    sum <= p ^ c;
  end
endmodule
```


a[31:0]
b[31:0]
exp_sum0_i
I0[7:0]
I1[7:0]
O[7:0]
RTL_ADD
exp_sum_i
I0[7:0]
I1[6:0]
O[7:0]
RTL_SUB
exp_result0_i
I0[7:0]
I1
O[7:0]
RTL_ADD
exp_result_i
I0[7:0]
I1[7:0]
O[7:0]
S
RTL_MUX
mant_mult_i
I0[23:0]
I1[23:0]
O[47:0]
RTL_MULT
mant_result_i
I0[22:0]
I1[22:0]
O[22:0]
S
RTL_MUX
sign_result_i
I0
I1
O
RTL_XOR
result[31:0]