# CA_Lab2

**K. Ram Mohan**

**COE19B055**

**Verilog Code:**

```
`include "rdca.v"

// Module for carry save adder
module FA1 (input [105:0] x,input [105:0] y,input [105:0] z,output [105:0] u,output [105:0] v);
   assign u = x^y^z;
   assign v[0] = 0;
   assign v[105:1] = (x&y) | (y&z) | (z&x);
endmodule

// Module for multyplying mantessas
module MUL_MANT (ma, mb, pro);
   input[52:0]   ma, mb;
   output[105:0]  pro;

   reg[105:0] pp[0:52];

   wire[127:0] temp_pro;

   wire c, carry;

   integer i;
   always @(*) begin
```

```
    for (i = 0; i<53; i++) begin
      // pp[i] = ma&{53{mb[i]}};
      if(mb[i]==1) begin
        pp[i] <= ma << i;
      end
      else begin
        pp[i] =
106'b00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000;
      end
    end
  end

  // Level 1
  wire[105:0] u11, v11, u12, v12, u13, v13, u14, v14, u15, v15, u16, v16, u17, v17, u18, v18, u19, v19, u110, v110, u111, v111, u112, v112, u113, v113, u114, v114, u115, v115, u116, v116, u117, v117;

  FA1 fa11(pp[0], pp[1], pp[2], u11, v11);
  FA1 fa12(pp[3], pp[4], pp[5], u12, v12);
  FA1 fa13(pp[6], pp[7], pp[8], u13, v13);
  FA1 fa14(pp[9], pp[10], pp[11], u14, v14);
  FA1 fa15(pp[12], pp[13], pp[14], u15, v15);
  FA1 fa16(pp[15], pp[16], pp[17], u16, v16);
  FA1 fa17(pp[18], pp[19], pp[20], u17, v17);
  FA1 fa18(pp[21], pp[22], pp[23], u18, v18);
  FA1 fa19(pp[24], pp[25], pp[26], u19, v19);
  FA1 fa110(pp[27], pp[28], pp[29], u110, v110);
  FA1 fa111(pp[30], pp[31], pp[32], u111, v111);
  FA1 fa112(pp[33], pp[34], pp[35], u112, v112);
  FA1 fa113(pp[36], pp[37], pp[38], u113, v113);
```

```
    FA1 fa114(pp[39], pp[40], pp[41], u114, v114);
    FA1 fa115(pp[42], pp[43], pp[44], u115, v115);
    FA1 fa116(pp[45], pp[46], pp[47], u116, v116);
    FA1 fa117(pp[48], pp[49], pp[50], u117, v117);

    // Level 2
    wire[105:0] u21, v21, u22, v22, u23, v23, u24, v24, u25, v25, u26, v26, u27, v27, u28, v28,
u29, v29, u210, v210, u211, v211, u212, v212;

    FA1 fa21(u11, v11, u12, u21, v21);
    FA1 fa22(v12, u13, v13, u22, v22);
    FA1 fa23(u14, v14, u15, u23, v23);
    FA1 fa24(v15, u16, v16, u24, v24);
    FA1 fa25(u17, v17, u18, u25, v25);
    FA1 fa26(v18, u19, v19, u26, v26);
    FA1 fa27(u110, v110, u111, u27, v27);
    FA1 fa28(v111, u112, v112, u28, v28);
    FA1 fa29(u113, v113, u114, u29, v29);
    FA1 fa210(v114, u115, v115, u210, v210);
    FA1 fa211(u116, v116, u117, u211, v211);
    FA1 fa212(v117, pp[51], pp[52], u212, v212);

    // Level 3
    wire[105:0] u31, v31, u32, v32, u33, v33, u34, v34, u35, v35, u36, v36, u37, v37, u38, v38;

    FA1 fa31(u21, v21, u22, u31, v31);
    FA1 fa32(v22, u23, v23, u32, v32);
    FA1 fa33(u24, v24, u25, u33, v33);
    FA1 fa34(v25, u26, v26, u34, v34);
    FA1 fa35(u27, v27, u28, u35, v35);
```

```
FA1 fa36(v28, u29, v29, u36, v36);
FA1 fa37(u210, v210, u211, u37, v37);
FA1 fa38(v211, u212, v212, u38, v38);

// Level 4
wire[105:0] u41, v41, u42, v42, u43, v43, u44, v44, u45, v45;

FA1 fa41(u31, v31, u32, u41, v41);
FA1 fa42(v32, u33, v33, u42, v42);
FA1 fa43(u34, v34, u35, u43, v43);
FA1 fa44(v35, u36, v36, u44, v44);
FA1 fa45(u37, v37, u38, u45, v45);

// Level 5
wire[105:0] u51, v51, u52, v52, u53, v53;

FA1 fa51(u41, v41, u42, u51, v51);
FA1 fa52(v42, u43, v43, u52, v52);
FA1 fa53(u44, v44, u45, u53, v53);

// Level 6
wire [105:0] u61, v61, u62, v62;

FA1 fa61(u51, v51, u52, u61, v61);
FA1 fa62(v52, u53, v53, u62, v62);

// Level 7
wire[105:0] u71, v71, u72, v72;
```

```
    FA1 fa71(u61, v61, u62, u71, v71);
    FA1 fa72(v62, v45, v38, u72, v72);

    // Level 8
    wire[105:0] u81, v81;

    FA1 fa81(u71, v71, u72, u81, v81);

    // Level 9
    wire[105:0] u91, v91;

    FA1 fa91(u81, v81, v72, u91, v91);



    // Final Level
    RCDA rcda1(u91[63:0], v91[63:0], 1'b0, temp_pro[63:0], c);
    RCDA rcda2({22'b0000000000000000000000,u91[105:64]},
{22'b0000000000000000000000,v91[105:64]},  c, temp_pro[127:64], carry);



    assign pro = temp_pro[105:0]; //pp[51] gives ma&53{mb[51]}; pp[51][51] gives 51st bit in
pp[51]
endmodule



// Module to normalize the result
module NORM_RES (pro1, carry, er, sr, mr, er_new);
    input[52:0]  pro1;
    input carry, sr;
    input[10:0]  er;
```

```
output reg[51:0] mr;
output reg[10:0] er_new;

reg[52:0] tmp;

always @(pro1) begin
   if(carry==1) begin
      tmp = pro1 >> 1;
      tmp[52] = carry;
      er_new = er + 1;

      mr = tmp[51:0];
   end
   else begin
      if(pro1[52]==1) begin
         mr = pro1[51:0];
         er_new = er;
      end
      else begin
         tmp = pro1;
         er_new = er;
         while(tmp[52]!=1'b1) begin
            tmp = tmp << 1;
            er_new = er_new - 1;
         end

         mr = tmp[51:0];
      end
   end
```

```
    end
endmodule


// Main module
module FP_MUL (a, b, out);
    input[63:0]  a, b;
    output  reg[63:0] out;

    wire sa, sb;                // Variables for storing signs of original inputs
    wire[10:0] ea, eb;          // Variables for storing expo of original inputs
    wire[52:0] ma, mb;          // Variables for storing mantissa of original inputs

    wire[105:0] pro;
    wire[104:0] final_pro;
    wire[52:0] pro_man;
    wire pro_carry;

    wire[10:0] temp_er;

    // Declaring output indvivdual variables
    reg sr;
    reg[10:0] er;
    reg[51:0] mr;


    wire sr_temp;
    wire[10:0] er_temp;
    wire[51:0] mr_temp;
```

```
  reg[1:0] temp;

  wire[10:0] zeroes, ones;
  wire[52:0] zeroes1, ones1;

  assign zeroes = 11'b00000000000;
  assign zeroes1 = 52'b0000000000000000000000000000000000000000000000000000;
  assign ones = 11'b11111111111;
  assign ones1 = 52'b1111111111111111111111111111111111111111111111111111;

  // Assigning the components to inputs to individual variables. Added one before mantissa that is 1 before decimal
  assign sa = a[63];
  assign ea = a[62: 52];
  assign ma = {1'b1,a[51:0]};
  assign sb = b[63];
  assign eb = b[62: 52];
  assign mb = {1'b1,b[51:0]};

  always @(a, b) begin
    if ((ea[10:0]==ones & ma[51:0]==zeroes1) | (eb[10:0]==ones & mb[51:0]==zeroes1)) begin
      // Either number is INF
      temp = 2'b00;
    end
    else if ((ea[10:0]==zeroes & ma[51:0]==zeroes1) | (eb[10:0]==zeroes & mb[51:0]==zeroes1)) begin
      // Either number is zero
```

```
        temp = 2'b01;
      end
      else if ((ea[10:0]==ones & ma[51:0]==ones1) | (eb[10:0]==ones & mb[51:0]==ones1))
begin
        // Nan case
        temp = 2'b11;
      end
      else begin
        temp = 2'b10;
      end
    end

    MUL_MANT mul_mant1(ma, mb, pro);

    // Putting carry bit aside for further calculation
    assign final_pro = pro[104:0];
    assign pro_man = pro[104:52];
    assign pro_carry = pro[105];

    assign temp_er = ea+eb-1023;
    NORM_RES norm_res1(pro_man, pro_carry, temp_er, sr_temp, mr_temp, er_temp);



    always @(*) begin
      if (temp==2'b00) begin
        sr = 0;
        er = ones;
        mr = 0;
      end
      else if(temp==2'b01)begin
```

```
        sr = 0;
        er = zeroes;
        mr = 0;
      end
      else if(temp==2'b11) begin
        sr = 0;
        er = ones;
        mr = ones1;
      end
      else begin
        sr = sa^sb;
        er = er_temp;
        mr = mr_temp;
      end

      if(er[10:0]==ones & mr[51:0]==zeroes1) begin
        out = {1'b0, ones, zeroes1};
      end
      else if(er[10:0]==ones & mr[51:0]==ones1) begin
        out = {1'b0, ones, ones1};
      end
      else begin
        out = {sr, er, mr};
      end
  end
endmodule
```

```
module FP_MUL_TB;
  reg[63:0] a, b;

  wire[63:0] out;

  FP_MUL fp_mul1(a, b, out);

  initial begin
    // a=1.5; b=1.0000000000000002
    #0
a=64'b0011111111111000000000000000000000000000000000000000000000000000;
b=64'b0011111111111000000000000000000000000000000000000000000000000001;

    // a=1.5; b=-3
    #10
a=64'b0011111111111000000000000000000000000000000000000000000000000000;
b=64'b1100000000001000000000000000000000000000000000000000000000000000;

    // a=1; b=-2
    #10
a=64'b0011111111111000000000000000000000000000000000000000000000000000;
b=64'b1100000000000000000000000000000000000000000000000000000000000000;

    // a=-5.000499999999997228883330535 6E0  b=1.500499999999999944932937978 59
    // ans = 7.503250249999988999852471367 92E0
    #10
a=64'b1100000000010100000000001000001100010010011011101001011110001101;
b=64'b0011111111111100000000010000011000100100110111010010111100011 0101;

    // a=-5.0005(approx) b=0.5005(approax)
    // ans = -2.502750249999991769641383143 6E0
```

```
    #10
a=64'b1100000000010100000000001000001100010010011011101001011110001101;
b=64'b0011111111110000000000100000110001001001101110100101111000110101 0;

    // a=0
    #10
a=64'b0000000000000000000000000000000000000000000000000000000000000000;
b=64'b1100000000001000000000000000000000000000000000000000000000000000;

    // b=-0
    #10
b=64'b1000000000000000000000000000000000000000000000000000000000000000;
a=64'b0011111111111000000000000000000000000000000000000000000000000000;

    // a=INF
    #10
a=64'b0111111111110000000000000000000000000000000000000000000000000000;
b=64'b1100000000001000000000000000000000000000000000000000000000000000;

    // b=-INF
    #10
a=64'b0011111111111000000000000000000000000000000000000000000000000000;
b=64'b1111111111110000000000000000000000000000000000000000000000000000;
  end

  initial begin
    $monitor(" in1=%b in2=%b Pro=%b", a, b, out);
    // $monitor("%d: a=%b b=%b ma=%b mb=%b sa=%b sb=%b sum=%b sign=%b exp=%b
man=%b", $time, a, b, res, res1, res2, res3, res4, res5, res6, res7);
    $dumpfile("fp_multiplier.vcd");
    $dumpvars(0, fp_mul1);
  end
endmodule
```

## Output:

1. a=1.5; b=1.0000000000000002
2. a=1.5; b=-3
3. a=1; b=-2
4. a=-5.000499999999999722888833305356E0,
   b=1.500499999999999944493293797859

   ans = 7.503250249999998899852471367 92E0
5. a=-5.0005(approx), b=0.5005(approax)

   ans = -2.502750249999999176964138314 36E0
6. a=0
7. b=-0
8. a=INF
9. b=INF

```
C:\Users\rammo\OneDrive\Documents\CA\Lab\Lab2>vvp fp_mul
VCD info: dumpfile fp_multiplier.vcd opened for output.
 in1=00111111111110000000000000000000000000000000000000000000000000
 in2=00111111111110000000000000000000000000000000000000000000000001
 Pro=00111111111110000000000000000000000000000000000000000000000001

 in1=00111111111110000000000000000000000000000000000000000000000000
 in2=11000000000010000000000000000000000000000000000000000000000000
 Pro=11000000000100100000000000000000000000000000000000000000000000

 in1=00111111111110000000000000000000000000000000000000000000000000
 in2=11000000000000000000000000000000000000000000000000000000000000
 Pro=11000000000000000000000000000000000000000000000000000000000000

 in1=11000000000101000000000010000011000100100111011101001011110001101
 in2=00111111111110000000010000011000100100111011101001011110001101
 Pro=11000000000111100000011010101000000100010010101110100001011011

 in1=11000000000101000000000010000011000100100111011101001011110001101
 in2=00111111111100000000010000011000100100111011101001011110001101010
 Pro=11000000000001000000101101000011110110001001110011100100101001

 in1=00000000000000000000000000000000000000000000000000000000000000000
 in2=11000000000010000000000000000000000000000000000000000000000000000
 Pro=00000000000000000000000000000000000000000000000000000000000000000

 in1=00111111111110000000000000000000000000000000000000000000000000000
 in2=10000000000000000000000000000000000000000000000000000000000000000
 Pro=00000000000000000000000000000000000000000000000000000000000000000

 in1=01111111111100000000000000000000000000000000000000000000000000000
 in2=11000000000010000000000000000000000000000000000000000000000000000
 Pro=11111111111000000000000000000000000000000000000000000000000000000

 in1=00111111111111000000000000000000000000000000000000000000000000000
 in2=11111111111100000000000000000000000000000000000000000000000000000
 Pro=11111111111110000000000000000000000000000000000000000000000000000
```