**Mini-Project IV**

**Simple CNN Inference Engine for Character Recognition on DE-1 Board**

**At**

**San Jose State University**

**By**

**Suhas Sunil Raut**

**011432980**

**top.v**

`timescale 1 ns / 1 ps

module top;

// Inputs

reg clk,rst,enable;

// Outputs

wire [6:0]y;

wire res\_flag2;

//main(clk,rst,enable,y,res\_flag);

main m1(

.clk(clk),

.rst(rst),

.enable(enable),

.y(y),

.res\_flag2(res\_flag2)

);

always begin

#10 clk = ~clk;

end

initial begin

clk = 0;

enable=0;

rst=1;

#40;

rst=0;

#10;

enable = 1;

#10;

//enable=0;

#10;

enable=1;

#120;

//enable=0;

#10;

enable=1;

#300000 $finish;

end

initial begin

$dumpfile("feed\_nn.vcd");

$dumpvars(0,top);

end

endmodule

**main.v**

module main(clk,rst,enable,y,res\_flag2);

input clk,rst;

input enable;

output [6:0]y;

output reg res\_flag2;

reg res\_flag;

reg [6:0] Layer1\_op;

reg [8:0] count,count\_d;

reg [8:0] count2,count\_d2;

reg [8:0] count3,count\_d3;

reg [8:0] addr\_count1,addr\_count\_d1;

reg [13:0]address,address\_d;

reg [8:0]address\_X,address\_d\_X;

reg [8:0]address\_d\_T2,address\_T2;

wire [6:0]RAM\_X\_op,RAM\_T1\_op,RAM\_T2\_op;

wire [13:0] XT1;

reg [13:0]L1;

wire [6:0] L1\_op,y\_1,y\_2;

reg [6:0] y\_d,y\_d2;

reg [6:0] ip\_mult2,ip\_mult21,ip\_mult22;

wire [6:0] sig\_L1;

wire [6:0] sig\_L2;

wire [13:0] op\_mult2;

reg [13:0] reg\_op;

wire [6:0] op\_mult;

reg [6:0]inp\_l2[0:26];

reg [6:0]inp\_l2\_d[0:26];

assign L1\_op = L1[10:4];

assign op\_mult = reg\_op[10:4];

assign y = sig\_L2;

ram\_xx ram\_xx\_inst(

.address(address\_d\_X),

.clock(clk),

.data(7'h00),

.wren(1'b0),

.q(RAM\_X\_op));

ram\_t1\_proj t1\_read(

.address(address\_d),

.clock(clk),

.data(7'h00),

.wren(1'b0),

.q(RAM\_T1\_op));

ram\_t2 t2\_ram\_inst(

.address(address\_d\_T2),

.clock(clk),

.data(7'h00),

.wren(1'b0),

.q(RAM\_T2\_op));

mult1 m1(

.dataa(RAM\_X\_op),

.datab(RAM\_T1\_op),

.result(XT1));

mult2 m2(

.dataa(ip\_mult22),

.datab(RAM\_T2\_op),

.result(op\_mult2));

add1 a1(

.dataa(L1\_op),

.datab(y\_d),

.cout(carry\_out1),

.overflow(of1),

.result(y\_1));

add2 a2(

.dataa(op\_mult),

.datab(y\_d2),

.cout(carry\_out2),

.overflow(of2),

.result(y\_2));

sigmoid\_value s1(

.res\_flag(res\_flag),

.in\_sig(Layer1\_op),

.op\_sig(sig\_L1));

sigmoid\_value s2(

.res\_flag(res\_flag2),

.in\_sig(y\_d2),

.op\_sig(sig\_L2));

always @(\*) begin

address = address\_d;

address\_X = address\_d\_X;

res\_flag = 0;

res\_flag2 = 0;

Layer1\_op = 0;

addr\_count1 = addr\_count\_d1;

count = count\_d;

address\_T2 = address\_d\_T2;

ip\_mult2 = 0;

L1 = XT1;

reg\_op = op\_mult2;

count2 = count\_d2;

count3 = count\_d3;

inp\_l2[0]= inp\_l2\_d[0];

inp\_l2[1]= inp\_l2\_d[1];

inp\_l2[2]= inp\_l2\_d[2];

inp\_l2[3]= inp\_l2\_d[3];

inp\_l2[4]= inp\_l2\_d[4];

inp\_l2[5]= inp\_l2\_d[5];

inp\_l2[6]= inp\_l2\_d[6];

inp\_l2[7]= inp\_l2\_d[7];

inp\_l2[8]= inp\_l2\_d[8];

inp\_l2[9]= inp\_l2\_d[9];

inp\_l2[10]= inp\_l2\_d[10];

inp\_l2[11]= inp\_l2\_d[11];

inp\_l2[12]= inp\_l2\_d[12];

inp\_l2[13]= inp\_l2\_d[13];

inp\_l2[14]= inp\_l2\_d[14];

inp\_l2[15]= inp\_l2\_d[15];

inp\_l2[16]= inp\_l2\_d[16];

inp\_l2[17]= inp\_l2\_d[17];

inp\_l2[18]= inp\_l2\_d[18];

inp\_l2[19]= inp\_l2\_d[19];

inp\_l2[20]= inp\_l2\_d[20];

inp\_l2[21]= inp\_l2\_d[21];

inp\_l2[22]= inp\_l2\_d[22];

inp\_l2[23]= inp\_l2\_d[23];

inp\_l2[24]= inp\_l2\_d[24];

inp\_l2[25]= inp\_l2\_d[25];

inp\_l2[26]= inp\_l2\_d[26];

//y=0;

if(enable) begin

address = address\_d + 14'd1;

addr\_count1 = addr\_count\_d1 + 9'd1;

if (address\_d\_X <= 9'd400) begin

address\_X = address\_d\_X + 9'd1;

end else begin

address\_X = 9'd1;

end

if(addr\_count\_d1 == 9'd403) begin

res\_flag = 1;

Layer1\_op = y\_d;

addr\_count1 = 9'd1;

//$display("%d",Layer1\_op);

end

//rounding logic

if (XT1[3]) begin

L1 = {XT1[13:3] + 11'b000\_0000\_0001,XT1[2:0]};

// L1 = XT1;

end

//rounding logic

if (op\_mult2[3]) begin

reg\_op = {op\_mult2[13:3] + 11'b000\_0000\_0001,op\_mult2[2:0]};

end

if(count <= 9'd25) begin

if(res\_flag) begin

count = count\_d + 1'b1;

inp\_l2[count] = sig\_L1;

//$display("inp\_l2=%b,count=%d",inp\_l2[count],count);

//ip\_mult2 = sig\_L1;

//address\_T2 = address\_d\_T2 + 1'b1;

//$display("%d",y\_d2);

end

end

if (count == 9'd26) begin

address\_T2 = address\_d\_T2 + 1'b1;

if (count2 <= 9'd25) begin

count2 = count\_d2 + 1'b1;

ip\_mult2 = inp\_l2[count2];

end else if (count2 == 9'd26) begin

count2 = 1;

end

end

if(count3 <= 9'd9) begin

if(count\_d2==9'd2 && address\_d\_T2 > 9'd2) begin

res\_flag2 = 1;

count3 = count\_d3 + 1'b1;

//$display("%d",y\_d2);

//y = y\_d2;

end

end

end

end

always @(posedge clk)begin

if(rst) begin

address\_d <= 14'd0;

address\_d\_X <= 9'd0;

address\_d\_T2 <= 9'd0;

y\_d <= 7'd0;

count\_d <= 9'd0;

addr\_count\_d1 <= 9'd0;

count\_d2 <= 9'd0;

count\_d3 <= 9'd0;

ip\_mult21 <= 7'd0;

ip\_mult22 <= 7'd0;

y\_d2 <= 7'd0;

inp\_l2\_d[0] <= 7'd0;

inp\_l2\_d[1] <= 7'd0;

inp\_l2\_d[2] <= 7'd0;

inp\_l2\_d[3] <= 7'd0;

inp\_l2\_d[4] <= 7'd0;

inp\_l2\_d[5] <= 7'd0;

inp\_l2\_d[6] <= 7'd0;

inp\_l2\_d[7] <= 7'd0;

inp\_l2\_d[8] <= 7'd0;

inp\_l2\_d[9] <= 7'd0;

inp\_l2\_d[10] <= 7'd0;

inp\_l2\_d[11] <= 7'd0;

inp\_l2\_d[12] <= 7'd0;

inp\_l2\_d[13] <= 7'd0;

inp\_l2\_d[14] <= 7'd0;

inp\_l2\_d[15] <= 7'd0;

inp\_l2\_d[16] <= 7'd0;

inp\_l2\_d[17] <= 7'd0;

inp\_l2\_d[18] <= 7'd0;

inp\_l2\_d[19] <= 7'd0;

inp\_l2\_d[20] <= 7'd0;

inp\_l2\_d[21] <= 7'd0;

inp\_l2\_d[22] <= 7'd0;

inp\_l2\_d[23] <= 7'd0;

inp\_l2\_d[24] <= 7'd0;

inp\_l2\_d[25] <= 7'd0;

inp\_l2\_d[26] <= 7'd0;

end else begin

address\_d <= address;

address\_d\_X <= address\_X;

address\_d\_T2 <= address\_T2;

if(res\_flag) begin

y\_d <= 0;

end else begin

if (carry\_out1 && of1) begin

y\_d <= 7'b0000001;

end else if(!carry\_out1 && of1) begin

y\_d <= 7'b0111111;

end else begin

y\_d <= y\_1;

end

end

if(res\_flag2) begin

y\_d2 <= 0;

end else begin

if (carry\_out2 && of2) begin

y\_d2 <= 7'b0000001;

end else if(!carry\_out1 && of1) begin

y\_d2 <= 7'b0111111;

end else begin

y\_d2 <= y\_2;

end

end

count\_d <= count;

addr\_count\_d1 <= addr\_count1;

count\_d2 <= count2;

count\_d3 <= count3;

ip\_mult21 <= ip\_mult2;

ip\_mult22 <= ip\_mult21;

inp\_l2\_d[0] <= inp\_l2[0];

inp\_l2\_d[1] <= inp\_l2[1];

inp\_l2\_d[2] <= inp\_l2[2];

inp\_l2\_d[3] <= inp\_l2[3];

inp\_l2\_d[4] <= inp\_l2[4];

inp\_l2\_d[5] <= inp\_l2[5];

inp\_l2\_d[6] <= inp\_l2[6];

inp\_l2\_d[7] <= inp\_l2[7];

inp\_l2\_d[8] <= inp\_l2[8];

inp\_l2\_d[9] <= inp\_l2[9];

inp\_l2\_d[10] <= inp\_l2[10];

inp\_l2\_d[11] <= inp\_l2[11];

inp\_l2\_d[12] <= inp\_l2[12];

inp\_l2\_d[13] <= inp\_l2[13];

inp\_l2\_d[14] <= inp\_l2[14];

inp\_l2\_d[15] <= inp\_l2[15];

inp\_l2\_d[16] <= inp\_l2[16];

inp\_l2\_d[17] <= inp\_l2[17];

inp\_l2\_d[18] <= inp\_l2[18];

inp\_l2\_d[19] <= inp\_l2[19];

inp\_l2\_d[20] <= inp\_l2[20];

inp\_l2\_d[21] <= inp\_l2[21];

inp\_l2\_d[22] <= inp\_l2[22];

inp\_l2\_d[23] <= inp\_l2[23];

inp\_l2\_d[24] <= inp\_l2[24];

inp\_l2\_d[25] <= inp\_l2[25];

inp\_l2\_d[26] <= inp\_l2[26];

end

end

endmodule

**sigmoid.v**

module sigmoid\_value(res\_flag,in\_sig,op\_sig);

input res\_flag;

input [6:0] in\_sig;

output [6:0] op\_sig;

reg [6:0] sig\_value;

assign op\_sig = sig\_value;

always @(\*) begin

sig\_value = 7'd0;

if(res\_flag) begin

case(in\_sig)

7'b1101100: sig\_value = 7'b0000100;

7'b0000111: sig\_value = 7'b0001010;

7'b0011010: sig\_value = 7'b0001101;

7'b0110010: sig\_value = 7'b0001111;

7'b0010101: sig\_value = 7'b0001101;

7'b1101011: sig\_value = 7'b0000011;

7'b1100101: sig\_value = 7'b0000010;

7'b0011110: sig\_value = 7'b0001110;

7'b1101000: sig\_value = 7'b0000011;

7'b0110000: sig\_value = 7'b0001111;

7'b1111110: sig\_value = 7'b0001000;

7'b0010100: sig\_value = 7'b0001100;

7'b0000100: sig\_value = 7'b0001001;

7'b1010100: sig\_value = 7'b0000001;

7'b0001100: sig\_value = 7'b0001011;

7'b0111010: sig\_value = 7'b0010000;

7'b0101001: sig\_value = 7'b0001111;

7'b0011101: sig\_value = 7'b0001110;

//7'b0000111: sig\_value = 7'b0001010;

7'b0010110: sig\_value = 7'b0001101;

7'b1011100: sig\_value = 7'b0000010;

7'b1010110: sig\_value = 7'b0000001;

//7'b0011010: sig\_value = 7'b0001101;

7'b1100111: sig\_value = 7'b0000011;

7'b1010000: sig\_value = 7'b0000001;

7'b0100111: sig\_value = 7'b0001111;

7'b1111011: sig\_value = 7'b0000111;

7'b1011010: sig\_value = 7'b0000001;

//7'b1101100: sig\_value = 7'b0000100;

//7'b0000111: sig\_value = 7'b0001010;

//7'b0011010: sig\_value = 7'b0001101;

//7'b0110010: sig\_value = 7'b0001111;

//7'b0010101: sig\_value = 7'b0001101;

//7'b1101011: sig\_value = 7'b0000011;

//7'b1100101: sig\_value = 7'b0000010;

//7'b0011110: sig\_value = 7'b0001110;

//7'b1101000: sig\_value = 7'b0000011;

//7'b0110000: sig\_value = 7'b0001111;

//7'b1111110: sig\_value = 7'b0001000;

//7'b0010100: sig\_value = 7'b0001100;

//7'b0000100: sig\_value = 7'b0001001;

7'b0000000: sig\_value = 7'b0001000;

//7'b1010100: sig\_value = 7'b0000001;

//7'b0001100: sig\_value = 7'b0001011;

//7'b0111010: sig\_value = 7'b0010000;

//7'b0101001: sig\_value = 7'b0001111;

//7'b0011101: sig\_value = 7'b0001110;

7'b1011001: sig\_value = 7'b0000001;

7'b0111100: sig\_value = 7'b0010000;

7'b1000111: sig\_value = 7'b0000000;

//7'b0010110: sig\_value = 7'b0001101;

7'b1000011: sig\_value = 7'b0000000;

7'b1000100: sig\_value = 7'b0000000;

//7'b1111110: sig\_value = 7'b0001000;

7'b0011000: sig\_value = 7'b0001101;

7'b1101110: sig\_value = 7'b0000100;

//7'b0111010: sig\_value = 7'b0010000;

7'b0100101: sig\_value = 7'b0001111;

//7'b1011010: sig\_value = 7'b0000001;

7'b1100110: sig\_value = 7'b0000011;

7'b1111100: sig\_value = 7'b0000111;

7'b1101001: sig\_value = 7'b0000011;

7'b0110101: sig\_value = 7'b0001111;

//7'b0010101: sig\_value = 7'b0001101;

7'b0101110: sig\_value = 7'b0001111;

//7'b0111100: sig\_value = 7'b0010000;

7'b1100000: sig\_value = 7'b0000010;

7'b1111001: sig\_value = 7'b0000110;

7'b0100000: sig\_value = 7'b0001110;

7'b0010111: sig\_value = 7'b0001101;

//7'b0110101: sig\_value = 7'b0001111;

7'b0111111: sig\_value = 7'b0010000;

//7'b1010110: sig\_value = 7'b0000001;

7'b1101101: sig\_value = 7'b0000100;

7'b0000001: sig\_value = 7'b0001000;

7'b0101010: sig\_value = 7'b0001111;

7'b0111011: sig\_value = 7'b0010000;

//7'b0101001: sig\_value = 7'b0001111;

//7'b0111111: sig\_value = 7'b0010000;

7'b0111101: sig\_value = 7'b0010000;

7'b0001000: sig\_value = 7'b0001010;

//7'b0000100: sig\_value = 7'b0001001;

7'b1100100: sig\_value = 7'b0000010;

7'b1101010: sig\_value = 7'b0000011;

//7'b0110010: sig\_value = 7'b0001111;

//7'b1101101: sig\_value = 7'b0000100;

//7'b0100111: sig\_value = 7'b0001111;

7'b1110010: sig\_value = 7'b0000101;

7'b0001101: sig\_value = 7'b0001011;

7'b0011100: sig\_value = 7'b0001110;

7'b0010010: sig\_value = 7'b0001100;

7'b0001110: sig\_value = 7'b0001011;

7'b0011111: sig\_value = 7'b0001110;

//7'b1101101: sig\_value = 7'b0000100;

7'b1110100: sig\_value = 7'b0000101;

7'b0000011: sig\_value = 7'b0001001;

7'b1111000: sig\_value = 7'b0000110;

//7'b1111011: sig\_value = 7'b0000111;

7'b0100010: sig\_value = 7'b0001110;

7'b1110110: sig\_value = 7'b0000110;

7'b1010010: sig\_value = 7'b0000001;

//7'b0011100: sig\_value = 7'b0001110;

//7'b1100101: sig\_value = 7'b0000010;

7'b1110001: sig\_value = 7'b0000101;

7'b0000010: sig\_value = 7'b0001000;

//7'b0000000: sig\_value = 7'b0001000;

7'b1011000: sig\_value = 7'b0000001;

default: sig\_value = 7'd0;

endcase

end

end

endmodule

clear all;

clc;

filename = "sig2.txt"

fid = fopen(filename,"r")

filename1 = "Test\_data.txt";

fid1 = fopen (filename1, "w");

while(!feof(fid))

a = fgetl(fid);

a = str2num(a);

a = dec2bin(a,7)

value = 0;

if (a(1)=='1' )

value = value - 4;

endif

if(a(2)=='1')

value = 2 + value;

endif

if(a(3)=='1')

value = 1 + value;

endif

if(a(4)=='1')

value = 0.5 + value;

endif

if(a(5)=='1')

value = 0.25 + value;

endif

if(a(6)=='1')

value = 0.125 + value;

endif

if(a(7)=='1')

value = 0.0625 + value;

endif

value

g = 0;

g = 1.0 / (1.0 + exp(-value))

number = g;

n=4;

if number >= 0

%number = floor(number\*(2^n))/(2^n);

number = number;

else

%number = -number;

%number = floor(number\*(2^n))/(2^n);

number = -number;

end

number

if number >= 0

sign=0;

integ = floor(number);

fract = number-integ;

else

sign = 1;

fract = -(number-ceil(number));

integ = (-number)-fract;

end

s=[];

%collect bits for numbers after decimal in a string

for x = 1:(n+1)

prod = fract \* 2;

if prod >= 1

fract = prod - 1;

bit = 1;

s = strcat(s,num2str(bit));

else

fract = prod;

bit = 0;

s = strcat(s,num2str(bit));

end

end

sign\_dumm=0;

final\_int = num2str(dec2bin(integ,2));

signed\_bin = strcat(num2str(sign\_dumm),final\_int,s);

if sign==0

%keep the number same and send for round

else

%take 2's complement

for i=1:length(signed\_bin)

if (signed\_bin(i)=='1')

signed\_bin(i)='0';

else

signed\_bin(i)='1';

end

end

end

x = length(signed\_bin);

temp=0;

if sign==1

if signed\_bin(length(signed\_bin))=='1'

for i=1:length(signed\_bin)

if signed\_bin(i)=='1'

temp = temp + (2^(x-1));

end

x = x-1;

end

temp = temp + 1;

signed\_bin= num2str(dec2bin(temp,n+1));

else

%keep it same no calculation needed for s(n+1)==0

end

end

%rounding logic

temp=0;

x=length(signed\_bin);

if signed\_bin(length(signed\_bin))=='1'

for i=1:length(signed\_bin)

if signed\_bin(i)=='1'

temp = temp + (2^(x-1));

end

x = x-1;

end

temp = temp + 1;

signed\_bin = num2str(dec2bin(temp,7));

else

%keep it same no calculation needed for s(n+1)==0

end

signed\_bin = signed\_bin(1:(length(signed\_bin)-1));

disp(signed\_bin);

fputs (fid1,"7'b");

fputs (fid1,a);

fputs (fid1,": sig\_value = ");

fputs (fid1,"7'b");

fputs (fid1,signed\_bin);

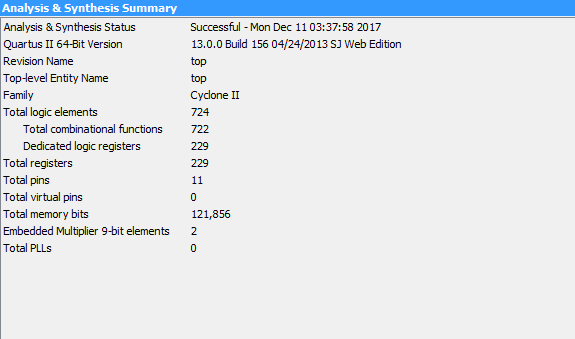
fputs (fid1,";");

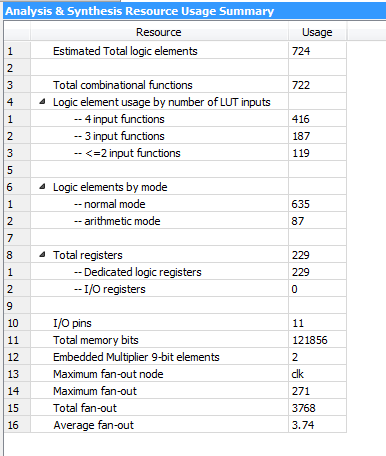
fputs (fid1,"\n");

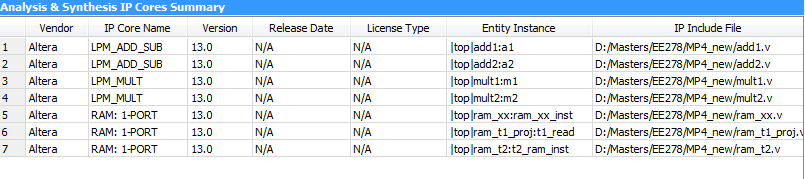
endwhile

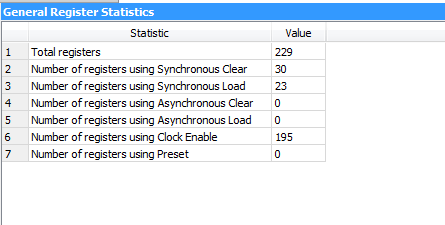
fclose (fid);

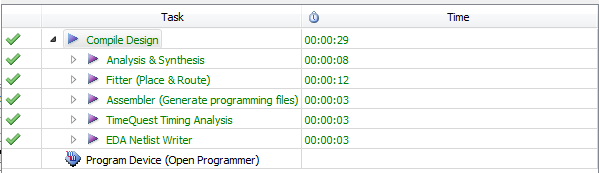
fclose (fid1);

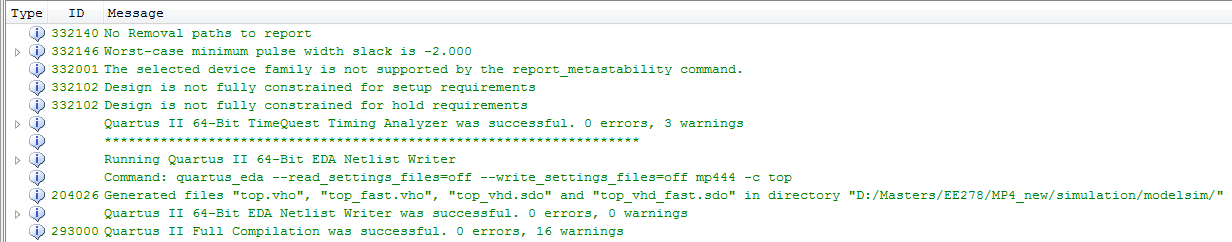
****

****

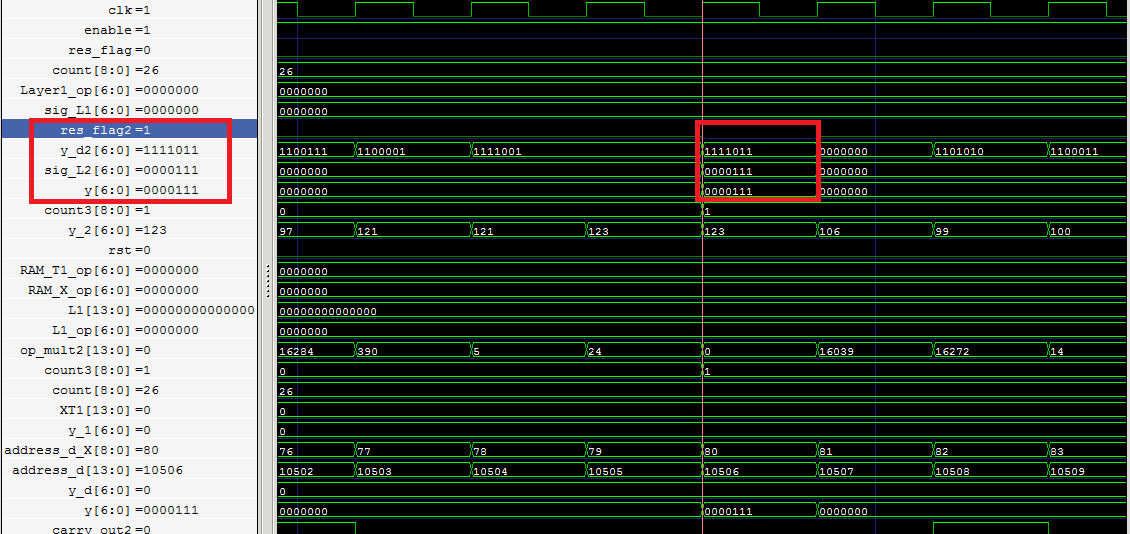
****

****

****

****

****

****

****