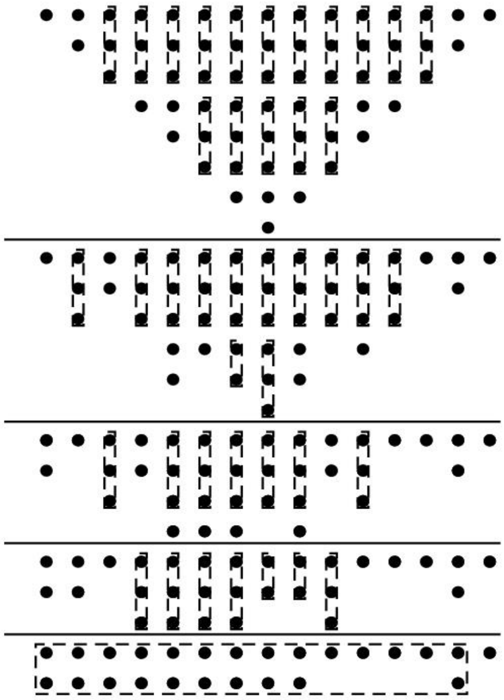


DIGITAL DESIGN PROJECT

VLSI Design Lab

TITLE OF THE PROJECT:

DESIGN OF 32-BIT WALLACE TREE ARRAY MULTIPLIER.

ANJALI BHARTI
1901030

TITLE: DESIGN OF 32-BIT WALLACE TREE ARRAY MULTIPLIER.

OBJECTIVE: To design of 32-bit Wallace tree array multiplier.

Theory and Algorithm:

Wallace Tree Algorithm Wallace tree is an efficient hardware implementation of a digital circuit that multiplies two integers, the idea of Wallace tree is given by Australian computer scientist Chris Wallace in 1964.

The Wallace tree has 3 steps:

Multiply the each bit of one of the arguments by each bit of the other, yielding n square results.

Reduce the number of partial products by using full adder and half adder.

Group the wires in two numbers and add them using CSA.

In last layer of algorithm we are using carry look ahead adder.

Algorithm:

The multiplication algorithm for an N bit multiplicand by N bit multiplier is shown below:

Y= Yn-1 Yn-2 ....Y2 Y1 Y0 Multiplicand

X= Xn-1 Xn-2 .... X2 X1 X0 Multiplier

```
                                  Y= Y_{n-1} Y_{n-2} ......................Y_2 Y_1 Y_0
                                  X= X_{n-1} X_{n-2}  ..................... X_2 X_1 X_0
=====================================================================================
                              Yn-1X0  Yn-2X0  Yn-3X0    ...... Y1X0  Y0X0
                         Yn-1X1  Yn-2X1  Yn-3X1    ...... Y1X1  Y0X1
                    Yn-1X2  Yn-2X2  Yn-3X2    ...... Y1X2  Y0X2
                     ...       ...       ...         ...
                   ....    ....     ....     ....      ....

          Yn-1Xn-2  Yn-2X0 n-2  Yn-3X n-2    ...... Y1Xn-2  Y0Xn-2
     Yn-1Xn-1  Yn-2X0n-1  Yn-3Xn-1    ...... Y1Xn-1  Y0Xn-1
-------------------------------------------------------------------------------------
```

Figure-2.1 Multiplication Algorithm

AND gates are used to generate the Partial Products, PP, If the multiplicand is N-bits and the Multiplier is M bits then there is N* M partial product. The way that the partial products are generated or summed up is the difference between the different architectures of various multipliers. Multiplication of binary numbers can be decomposed into additions. Consider the multiplication of two 8-bit numbers A and B to generate the 16 bit product P.

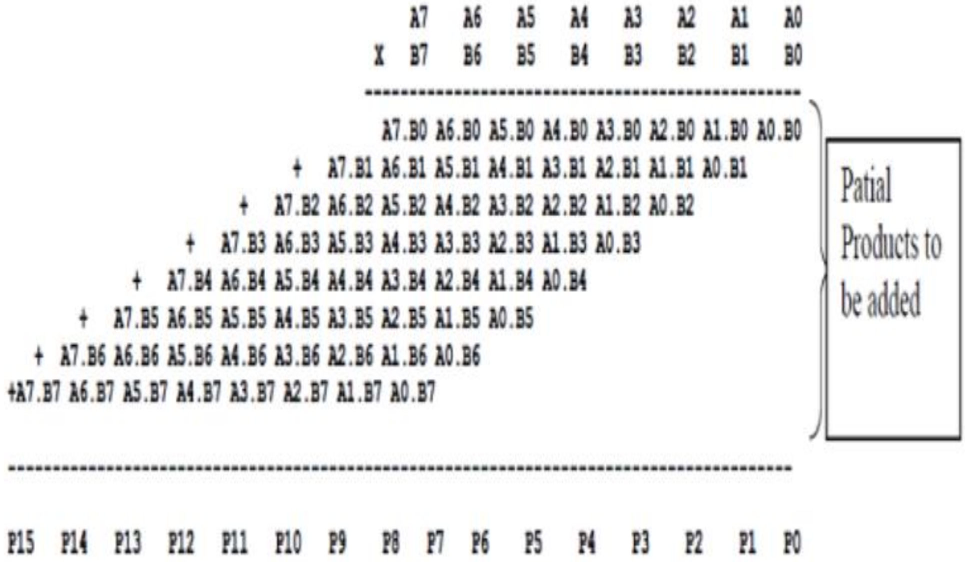


Figure-2.1.1 Generation of Partial Product

Block diagram:

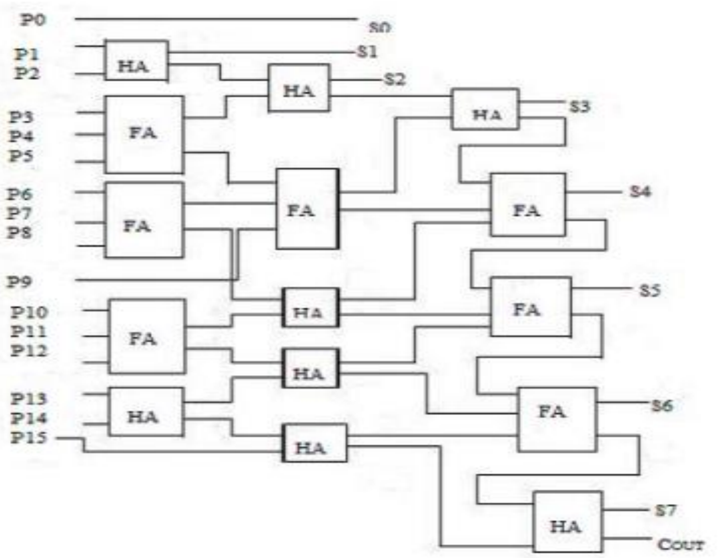


Figure 3.1.2-Block diagram of 4*4 Wallace tree multiplication

APPENDIX:

- ## CODE FOR HALFADDER:

```
module halfadder(
   input a,
   input b,
   output sum,
   output carry
   );
   assign #1 sum=a^b;
   assign #1 carry=a&b;
endmodule
```

CODE FOR FULLADDER:

```
module fulladd(
   input a,
   input b,
   input cin,
   output sum,
   output carry
   );
   assign #2 sum=a^b^cin;
   assign #5 carry=(a&b)|(a&cin)|(b&cin);
endmodule
```

- ## CODE FOR ADDER64:

```
module adder64(
   input [63:0] x,
   input [63:0] y,
   output [63:0] z
```

```
    );
    wire [63:0] cin;
    wire [64:0] carry;
    halfadder H(x[0],y[0],z[0],carry[0]);
    genvar i;
    generate for(i=0;i<63;i=i+1)
    begin
    fulladd f(x[1+i],y[1+i],carry[0+i],z[1+i],carry[1+i]);
    end
    endgenerate
endmodule
```

- CODE FOR WALLACETREEMULTIPLIER:

```
module wallace(
    input [31:0] a,
    input [31:0] b,
    output [63:0] z
    );
    wire [1023:0]w;
    wire[319:0] s,c;
    wire[229:0] d,v;
   wire[169:0] g,n;
    wire[341:0]j,m;
     wire[63:0]x,y;
    genvar i;
    generate for(i=0;i<32;i=i+1)
    begin
    assign #1 w[i]=a[i]&b[0];
    assign #1 w[i+32]=a[i]&b[1];
    assign #1 w[i+64]=a[i]&b[2];
    assign #1 w[i+96]=a[i]&b[3];
```

```
assign #1 w[i+128]=a[i]&b[4];
assign #1 w[i+160]=a[i]&b[5];
assign #1 w[i+192]=a[i]&b[6];
assign #1 w[i+224]=a[i]&b[7];
assign #1 w[i+256]=a[i]&b[8];
assign #1 w[i+288]=a[i]&b[9];
assign #1 w[i+320]=a[i]&b[10];
assign #1 w[i+352]=a[i]&b[11];
assign #1 w[i+384]=a[i]&b[12];
assign #1 w[i+416]=a[i]&b[13];
assign #1 w[i+448]=a[i]&b[14];
assign #1 w[i+480]=a[i]&b[15];
assign #1 w[i+512]=a[i]&b[16];
assign #1 w[i+544]=a[i]&b[17];
assign #1 w[i+576]=a[i]&b[18];
assign #1 w[i+608]=a[i]&b[19];
assign #1 w[i+640]=a[i]&b[20];
assign #1 w[i+672]=a[i]&b[21];
assign #1 w[i+704]=a[i]&b[22];
assign #1 w[i+736]=a[i]&b[23];
assign #1 w[i+768]=a[i]&b[24];
assign #1 w[i+800]=a[i]&b[25];
assign #1 w[i+832]=a[i]&b[26];
assign #1 w[i+864]=a[i]&b[27];
assign #1 w[i+896]=a[i]&b[28];
assign #1 w[i+928]=a[i]&b[29];
assign #1 w[i+960]=a[i]&b[30];
assign #1 w[i+992]=a[i]&b[31];

end
endgenerate
always @*
$monitor(w[31:0]);
```

```
 halfadder h1(w[1],w[32],s[0],c[0]);
generate for(i=1;i<31;i=i+1)
begin
fulladd f1(w[1+i],w[32+i],w[63+i],s[i],c[i]);
end
endgenerate
halfadder h2(w[63],w[94],s[31],c[31]);
halfadder h3(w[97],w[128],s[32],c[32]);
generate for(i=0;i<30;i=i+1)
begin
fulladd f2(w[98+i],w[129+i],w[160+i],s[33+i],c[33+i]);
end
endgenerate
halfadder h4(w[159],w[190],s[63],c[63]);
halfadder h5(w[193],w[224],s[64],c[64]);
 generate for(i=0;i<30;i=i+1)
begin
fulladd f3(w[194+i],w[225+i],w[256+i],s[65+i],c[65+i]);
end
endgenerate
halfadder h6(w[225],w[286],s[95],c[95]);
halfadder h7(w[289],w[320],s[96],c[96]);
generate for(i=0;i<30;i=i+1)
begin
fulladd f4(w[290+i],w[321+i],w[352+i],s[97+i],c[97+i]);
end
endgenerate
halfadder h8(w[351],w[382],s[127],c[127]);
halfadder h9(w[385],w[416],s[128],c[128]);
 generate for(i=0;i<30;i=i+1)
begin
fulladd f5(w[386+i],w[417+i],w[448+i],s[129+i],c[129+i]);
end
```

```
endgenerate
halfadder h10(w[447],w[478],s[159],c[159]);
 halfadder h11(w[481],w[512],s[160],c[160]);
  generate for(i=0;i<30;i=i+1)
 begin
 fulladd f6(w[482+i],w[513+i],w[544+i],s[161+i],c[161+i]);
end
endgenerate
 halfadder h12(w[543],w[574],s[191],c[191]);
 halfadder h13(w[577],w[608],s[192],c[192]);
  generate for(i=0;i<30;i=i+1)
 begin
 fulladd f7(w[578+i],w[609+i],w[640+i],s[193+i],c[193+i]);
end
endgenerate
 halfadder h14(w[639],w[670],s[223],c[223]);
  halfadder h15(w[673],w[704],s[224],c[224]);
  generate for(i=0;i<30;i=i+1)
 begin
 fulladd f8(w[674+i],w[705+i],w[736+i],s[225+i],c[225+i]);
end
endgenerate
halfadder h16(w[735],w[766],s[255],c[255]);
 halfadder h17(w[769],w[800],s[256],c[256]);
  generate for(i=0;i<30;i=i+1)
 begin
 fulladd f9(w[770+i],w[801+i],w[832+i],s[257+i],c[257+i]);
end
endgenerate
 halfadder h18(w[831],w[862],s[287],c[287]);
 halfadder h19(w[865],w[896],s[288],c[288]);
  generate for(i=0;i<30;i=i+1)
 begin
```

```
 fulladd f10(w[866+i],w[897+i],w[928+i],s[289+i],c[289+i]);
end
endgenerate
halfadder h20(w[927],w[958],s[319],c[319]);
 halfadder h21(c[0],s[1],d[0],v[0]);
fulladd f11(c[1],s[2],w[96],d[1],v[1]);

generate for(i=0;i<29;i=i+1)
 begin
 fulladd f13(s[3+i],c[2+i],s[32+i],d[2+i],v[2+i]);
end
endgenerate
fulladd f14(w[95],c[31],s[61],d[31],v[31]);
halfadder h22(c[33],w[192],d[32],v[32]);
 halfadder h23(c[34],s[64],d[33],v[33]);
 generate for(i=0;i<29;i=i+1)
 begin
 fulladd f15(c[35+i],s[65+i],c[64+i],d[34+i],v[34+i]);
end
endgenerate

  halfadder h24(c[93],s[94],d[63],v[63]);
  halfadder h25(c[94],s[95],d[64],v[64]);
  halfadder h26(c[95],w[287],d[65],v[65]);
  halfadder h27(c[96],s[97],d[66],v[66]);
fulladd f17(c[97],s[98],w[384],d[67],v[67]);

generate for(i=0;i<29;i=i+1)
 begin
 fulladd f19(s[99+i],c[98+i],s[128+i],d[68+i],v[68+i]);
end
endgenerate
fulladd f20(c[127],w[283],s[157],d[97],v[97]);
```

```
 halfadder h28(c[129],w[480],d[98],v[98]);
  halfadder h29(c[130],s[160],d[99],v[99]);
 generate for(i=0;i<29;i=i+1)
 begin
 fulladd f21(c[131+i],s[161+i],c[160+i],d[100+i],v[100+i]);
end
endgenerate

  halfadder h30(c[189],s[190],d[129],v[129]);
  halfadder h31(c[190],s[191],d[130],v[130]);
  halfadder h32(c[191],w[575],d[131],v[131]);
  halfadder h33(c[192],s[193],d[132],v[132]);
fulladd f23(c[193],s[194],w[672],d[133],v[133]);

generate for(i=0;i<29;i=i+1)
 begin
 fulladd f19(s[195+i],c[194+i],s[224+i],d[134+i],v[134+i]);
end
endgenerate
fulladd f25(c[223],w[671],s[253],d[163],v[163]);
halfadder h34(c[225],w[768],d[164],v[164]);
  halfadder h35(c[226],s[256],d[165],v[165]);
 generate for(i=0;i<29;i=i+1)
 begin
 fulladd f26(c[227+i],s[257+i],c[256+i],d[166+i],v[166+i]);
end
endgenerate

  halfadder h36(c[285],s[286],d[195],v[195]);
  halfadder h37(c[286],s[287],d[196],v[196]);
  halfadder h38(c[287],w[863],d[197],v[197]);
   halfadder h39(c[288],s[289],d[198],v[198]);
 generate for(i=0;i<30;i=i+1)
```

```
 begin
 fulladd f28(c[289+i],s[290+i],w[960+i],d[199+i],v[199+i]);
end
endgenerate
fulladd f29(c[319],w[959],w[990],d[229],v[229]);
 halfadder h40(d[1],v[0],g[0],n[0]);
halfadder h41(d[2],v[1],g[1],n[1]);
fulladd f30(d[3],v[2],c[32],g[2],n[2]);
generate for(i=0;i<28;i=i+1)
 begin
  fulladd f31(d[4+i],v[3+i],d[32+i],g[3+i],n[3+i]);
end
endgenerate
  fulladd f32(s[62],v[31],d[60],g[31],n[31]);
 halfadder h42(s[63],d[61],g[32],n[32]);
 halfadder h43(w[191],d[62],g[33],n[33]);
  halfadder h44(v[34],w[288],g[34],n[34]);
  halfadder h45(v[35],s[96],g[35],n[35]);
  halfadder h46(v[36],d[66],g[36],n[36]);
  generate for(i=0;i<29;i=i+1)
 begin
  fulladd f33(v[37+i],d[67+i],v[66+i],g[37+i],n[37+i]);
end
endgenerate
halfadder h47(d[96],v[95],g[66],n[66]);
halfadder h48(d[97],v[96],g[67],n[67]);
halfadder h49(s[158],v[97],g[68],n[68]);
halfadder h50(d[99],v[98],g[69],n[69]);
 halfadder h51(d[100],v[99],g[70],n[70]);
fulladd f33(d[101],v[100],w[576],g[71],n[71]);
fulladd f34(d[102],v[101],s[192],g[72],n[72]);
generate for(i=0;i<29;i=i+1)
 begin
```

```
  fulladd f35(d[103+i],v[102+i],d[132+i],g[73+i],n[73+i]);
end
endgenerate
halfadder h52(v[131],d[161],g[102],n[102]);
halfadder h53(v[134],c[224],g[103],n[103]);
halfadder h54(v[135],d[164],g[104],n[104]);
 generate for(i=0;i<28;i=i+1)
 begin
  fulladd f35(v[136+i],d[165+i],v[164+i],g[105+i],n[105+i]);
end
endgenerate
 fulladd f36(s[255],d[193],v[192],g[133],n[133]);
  fulladd f37(w[767],d[194],v[193],g[134],n[134]);
  halfadder h55(d[195],v[194],g[135],n[135]);
  halfadder h56(d[196],v[195],g[136],n[136]);
  halfadder h57(d[197],v[196],g[137],n[137]);
  halfadder h58(d[199],v[198],g[138],n[138]);
  generate for(i=0;i<30;i=i+1)
 begin
  fulladd f38(d[200+i],v[199+i],w[992+i],g[139+i],n[139+i]);
end
endgenerate
fulladd f39(w[991],v[229],w[1022],g[169],n[169]);
halfadder h59(g[1],n[0],j[0],m[0]);
halfadder h60(g[2],n[1],j[1],m[1]);
halfadder h61(g[3],n[2],j[2],m[2]);
fulladd f40(g[4],n[3],v[32],j[3],m[3]);
fulladd f41(g[5],n[4],v[33],j[4],m[4]);
generate for(i=0;i<28;i=i+1)
 begin
  fulladd f42(g[6+i],n[5+i],g[34+i],j[5+i],m[5+i]);
  end
endgenerate
```

```
fulladd f43(d[63],n[33],g[62],j[33],m[33]);
halfadder h62(d[64],g[63],j[34],m[34]);
halfadder h63(d[65],g[64],j[35],m[35]);
halfadder h64(n[38],c[128],j[36],m[36]);
halfadder h65(n[39],d[98],j[37],m[37]);
halfadder h66(n[40],g[69],j[38],m[38]);
 generate for(i=0;i<28;i=i+1)
 begin
  fulladd f44(n[41+i],g[70+i],n[69+i],j[39+i],m[39+i]);


end
endgenerate
fulladd f45(w[479],g[98],n[97],j[67],m[67]);
generate for(i=0;i<4;i=i+1)
 begin
  halfadder h67(g[99+i],n[98+i],j[68+i],m[68+i]);
  end
endgenerate
halfadder h68(d[162],n[102],j[72],m[72]);
halfadder h69(g[104],n[103],j[73],m[73]);
halfadder h70(g[105],n[104],j[74],m[74]);
halfadder h71(g[106],n[105],j[75],m[75]);
fulladd f46(g[107],n[106],w[864],j[76],m[76]);
fulladd f47(g[108],n[107],s[288],j[77],m[77]);
fulladd f48(g[109],n[108],d[198],j[78],m[78]);
 generate for(i=0;i<28;i=i+1)
 begin
  fulladd f49(g[110+i],n[109+i],g[138+i],j[79+i],m[79+i]);
  end
endgenerate
fulladd f50(v[197],n[137],g[166],j[107],m[107]);
//reduction
 generate for(i=0;i<5;i=i+1)
```

```
  begin
   halfadder h72(j[1+i],m[0+i],j[108+i],m[108+i])
end
 endgenerate
 generate for(i=0;i<4;i=i+1)
  begin
   fulladd f51(j[6+i],m[5+i],n[34+i],j[113+i],m[113+i]);
    end
 endgenerate
  generate for(i=0;i<26;i=i+1)
  begin
   fulladd f51(j[10+i],m[9+i],j[36+i],j[117+i],m[117+i]);
  end
 endgenerate
    fulladd f52(g[65],m[35],j[62],j[143],m[143]);
    generate for(i=0;i<3;i=i+1)
  begin
   halfadder h73(g[66+i],j[63+i],j[144+i],m[144+i]);
  end
 endgenerate
 halfadder h74(s[159],j[66],j[147],m[147]);
 halfadder h75(m[42],v[132],j[148],m[148]);
 halfadder h76(m[43],v[133],j[149],m[149]);
 halfadder h77(m[44],g[103],j[150],m[150]);
 halfadder h78(m[45],j[73],j[151],m[151]);
  generate for(i=0;i<27;i=i+1)
  begin
   fulladd f53(m[46+i],j[74+i],m[73+i],j[152+i],m[152+i]);
   end
 endgenerate
  fulladd f54(s[254],j[101],m[100],j[179],m[179]);
  generate for(i=0;i<6;i=i+1)
  begin
```

```
  halfadder h79(j[102+i],m[101+i],j[180+i],m[180+i]);
end
endgenerate
 halfadder h80(g[167],m[107],j[186],m[186]);
 //reduction yahan tak dekh liya h
  generate for(i=0;i<9;i=i+1)
 begin
  halfadder h81(j[109+i],m[108+i],j[187+i],m[187+i]);
end
endgenerate
generate for(i=0;i<6;i=i+1)
 begin
  fulladd f55(j[118+i],m[117+i],m[36+i],j[196+i],m[196+i]);
end
endgenerate
generate for(i=0;i<24;i=i+1)
 begin
  fulladd f56(j[124+i],m[123+i],j[148+i],j[202+i],m[202+i]);
end
endgenerate
fulladd f57(j[67],m[147],j[172],j[226],m[226]);
 generate for(i=0;i<5;i=i+1)
 begin
  halfadder h82(j[68+i],j[173+i],j[227+i],m[227+i]);
end
endgenerate
halfadder h83(d[163],j[178],j[232],m[232]);
//reduction
generate for(i=0;i<15;i=i+1)
 begin
  halfadder h84(j[188+i],m[187+i],j[233+i],m[233+i]);
end
endgenerate
```

```
generate for(i=0;i<30;i=i+1)
 begin
  fulladd f58(j[203+i],m[202+i],m[148+i],j[248+i],m[248+i]);
end
endgenerate
fulladd f59(j[179],m[232],m[178],j[278],m[278]);
generate for(i=0;i<7;i=i+1)
 begin
  halfadder h85(j[180+i],m[179+i],j[279+i],m[279+i]);
end
endgenerate
halfadder h86(g[168],m[186],j[286],m[286]);
//reduction
generate for(i=0;i<23;i=i+1)
 begin
  halfadder h87(j[234+i],m[233+i],j[287+i],m[287+i]);
end
endgenerate
generate for(i=0;i<30;i=i+1)
 begin
  fulladd f60(j[257+i],m[256+i],n[138+i],j[310+i],m[310+i]);
end
endgenerate
fulladd f61(g[169],m[286],n[168],j[340],m[340]);
  halfadder h88(w[1023],n[169],j[341],m[341]);
   assign x[0]=w[0];
  assign x[1]=s[0];
  assign x[2]=d[0];
  assign x[3]=g[0];
  assign x[4]=j[0];
  assign x[5]=j[108];
  assign x[6]=j[187];
  assign x[7]=j[233];
```

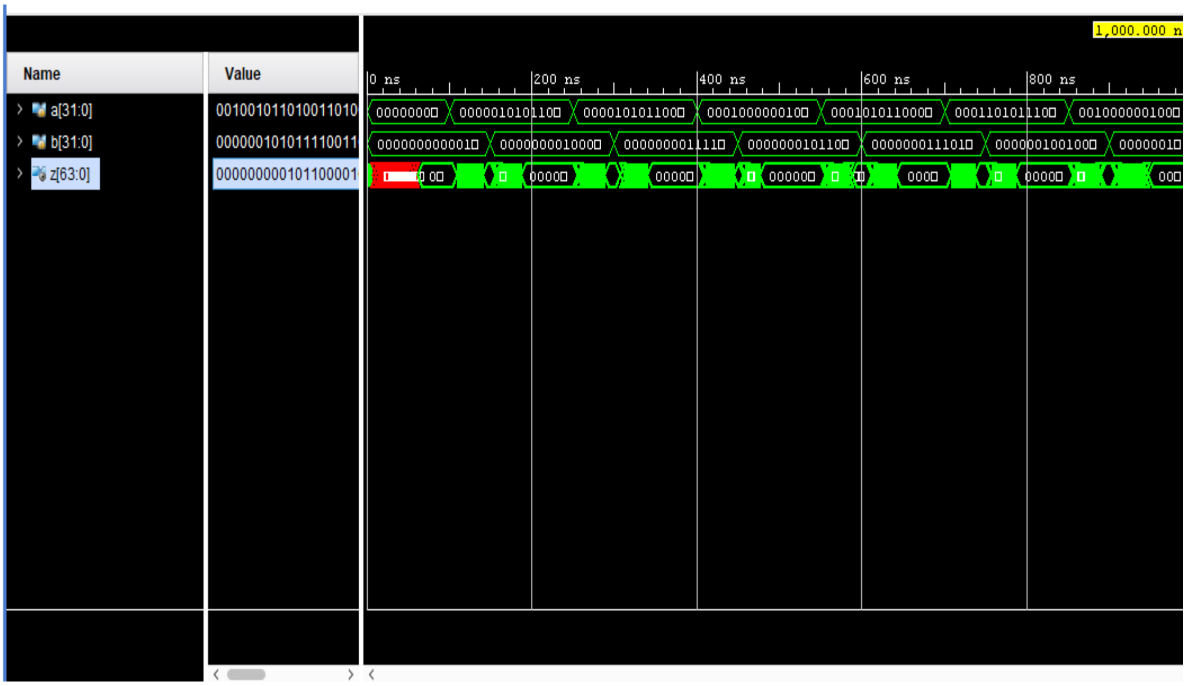
```
    generate for(i=0;i<55;i=i+1)
   begin
   assign x[8+i]=j[287+i];
    end
  endgenerate
  assign x[63]=0;
    generate for(i=0;i<9;i=i+1)
   begin
   assign y[0+i]=0;
  end
  endgenerate
   generate for(i=0;i<55;i=i+1)
   begin
   assign y[9+i]=m[287+i];
   end
   endgenerate
   adder64 ad(x,y,z);
endmodule
```

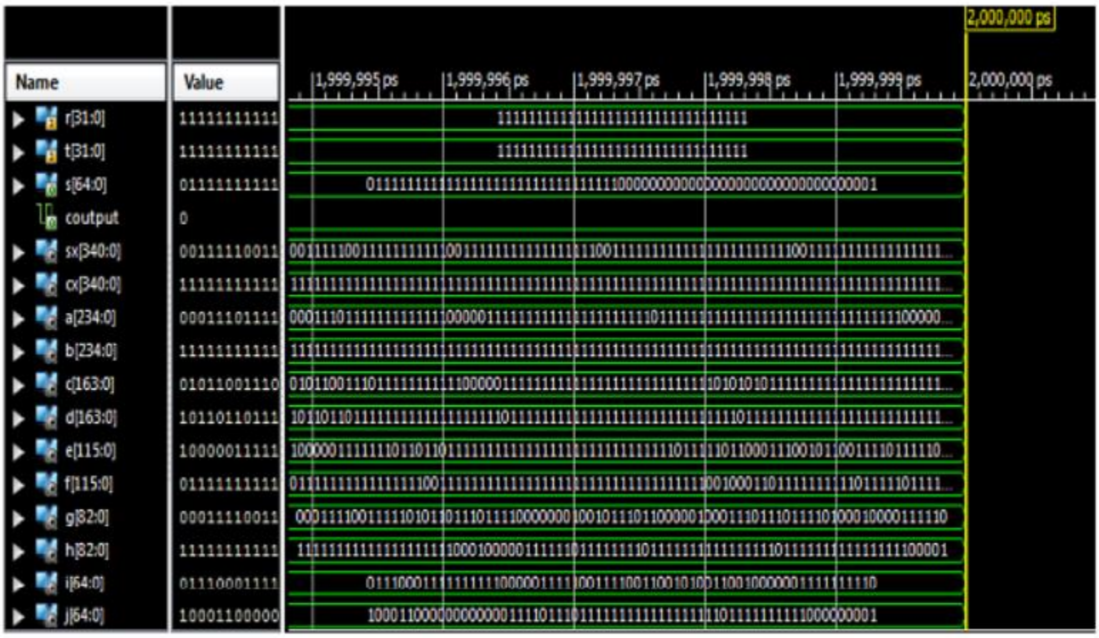
- CODE FOR TESTBENCH:

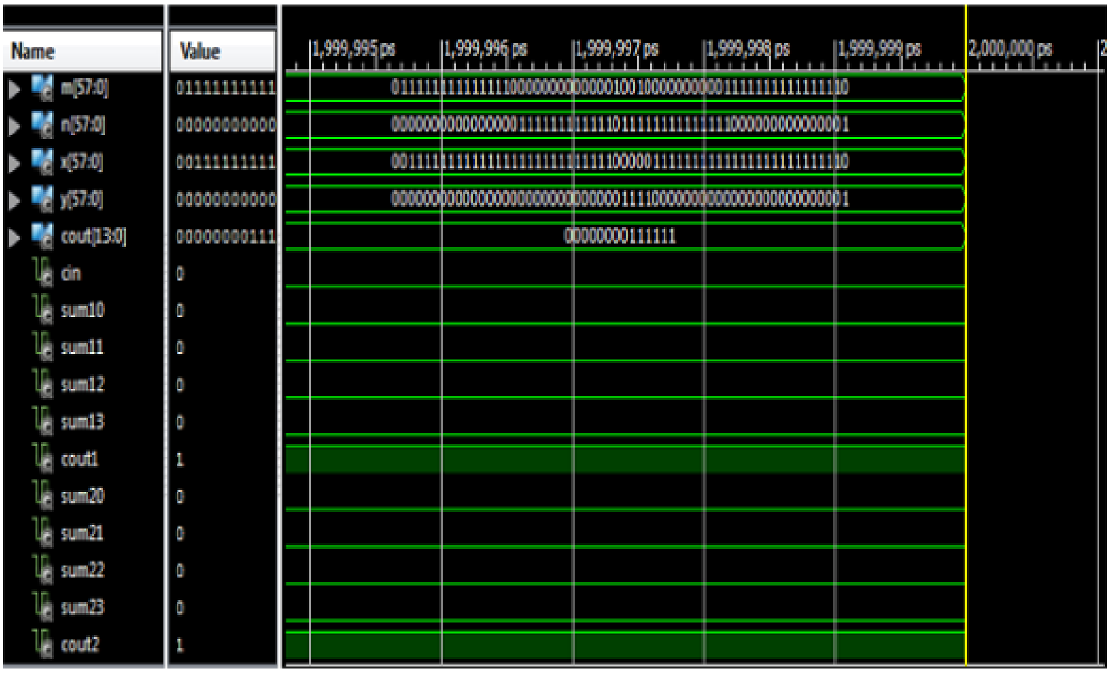
```
module testbench();
reg[31:0] a,b;
wire[63:0] z;
wallace i1(a,b,z);
initial begin
a=145556;
b=1200000;
end
always
begin
#100 assign a=a+90217771;
#50 assign b=b+7455000;
```

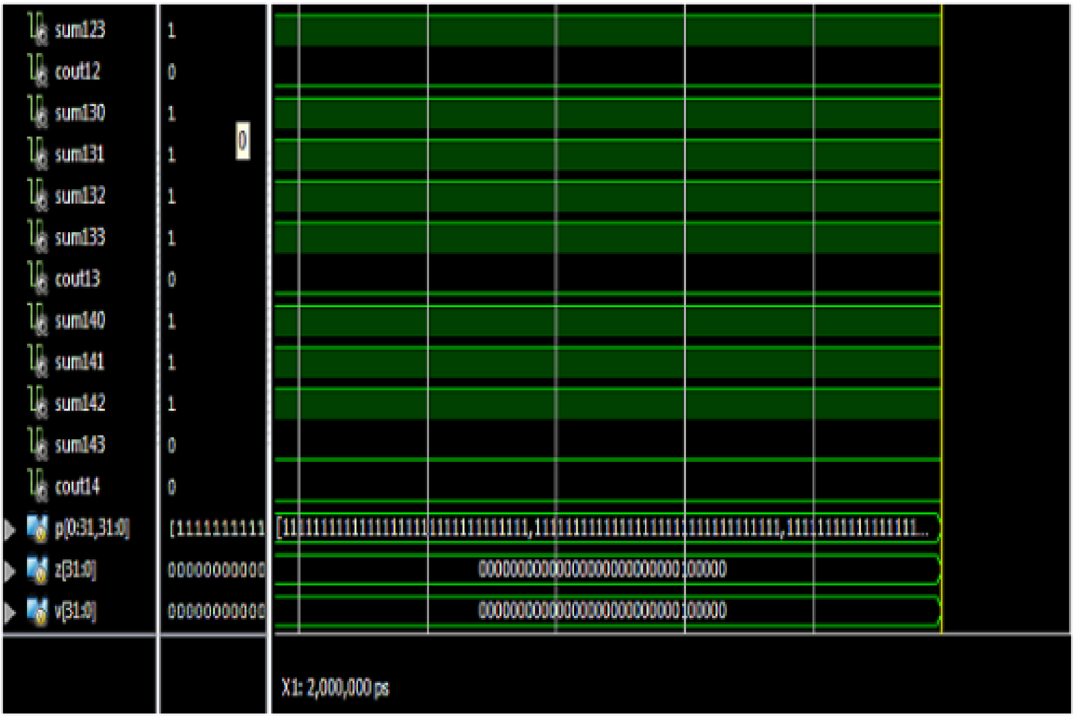
```
 $display("%b +%b =%b",a,b,z);
End
endmodule
```

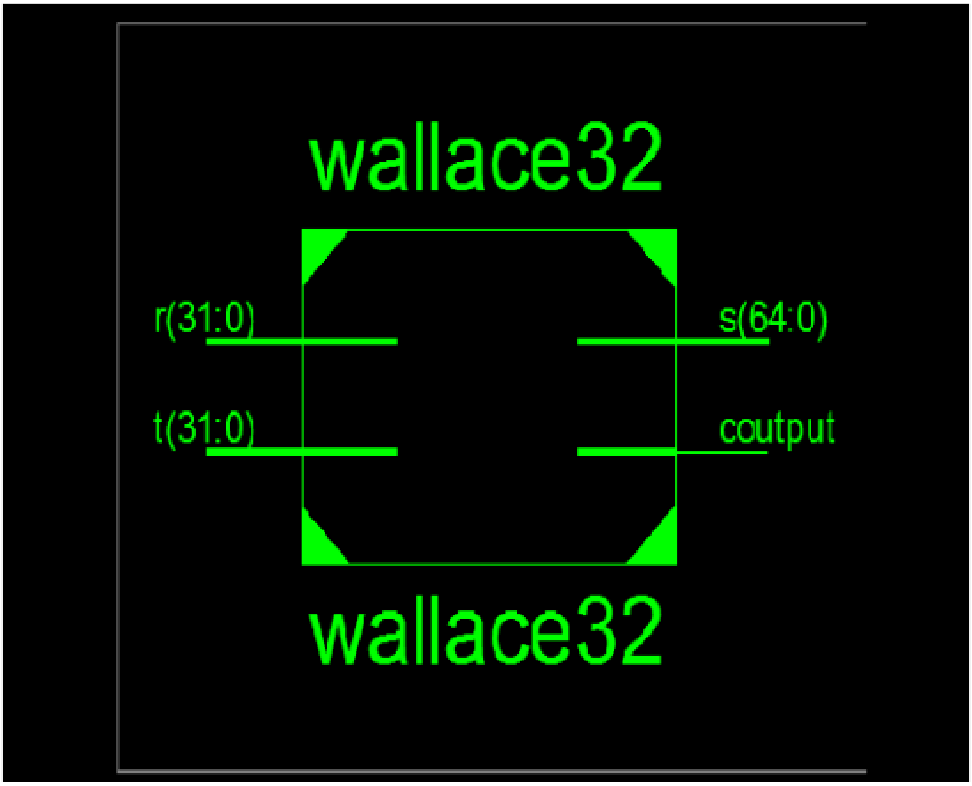
SIMULATION RESULT:






Name
Value
1,999,995 ps
1,999,996 ps
1,999,997 ps
1,999,998 ps
1,999,999 ps
2,000,000 ps
m[57:0]
n[57:0]
x[57:0]
y[57:0]
cout[13:0]
00000000111111
cin
0
sum10
0
sum11
0
sum12
0
sum13
0
cout1
1
sum20
0
sum21
0
sum22
0
sum23
0
cout2
1


sum123
1
cout12
0
sum130
1
sum131
1
sum132
1
sum133
1
cout13
0
sum140
1
sum141
1
sum142
1
sum143
0
cout14
0
p[0:31,31:0]
z[31:0]
v[31:0]
X1: 2,000,000 ps



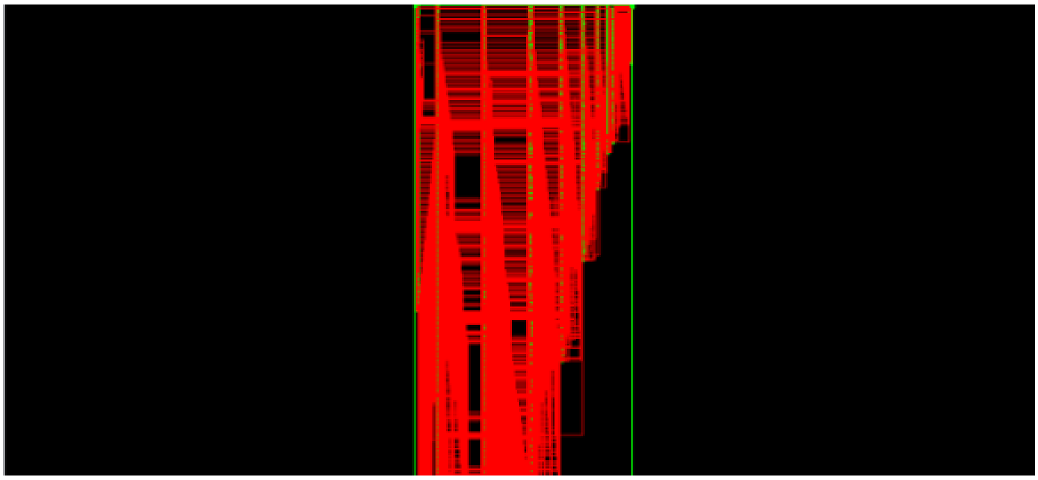


Figure-6.1 Technological View of Wallace tree multiplier.

CONCLUSION:

The entire design of a 32 bit Wallace tree multiplier is coded in verilog and implemented in Xilinx FPGA. The RTL schematic view of the design is presented in figure Simulation results and technological view are shown in figure. The delay is 16.56ns and speed has been increased and due to the reduced layer the area required is also less.