LAB # 06

OBJECTIVE : Develop, implement & simulate Dadda wallance Tree.

Lab Task 1

Implement & simulate 4X4 Dadda Wallance Tree.

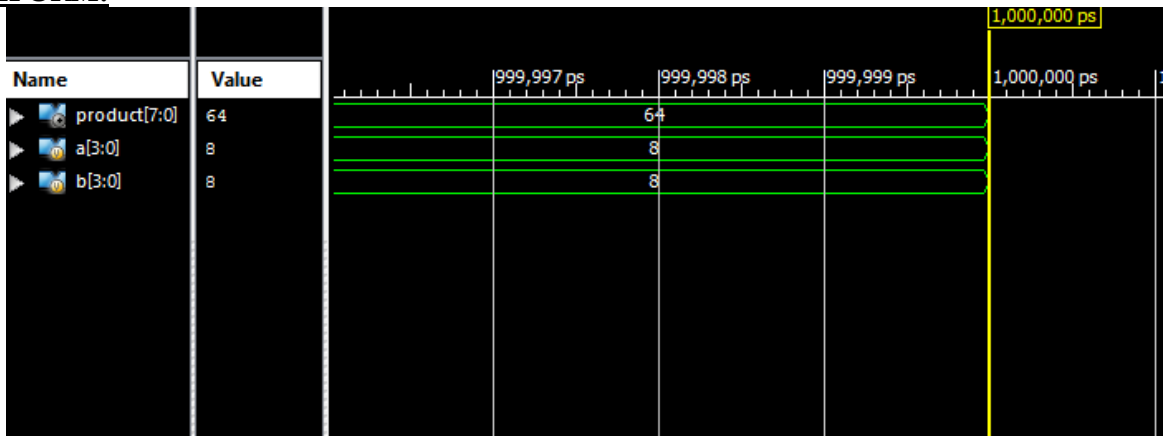
Verilog Module code:

```
module WallaceMul(
    input [3:0] a,
    input [3:0] b,
    output [7:0] product
    );

      wire s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,c0,c1,
      c2,c3,c4,c5,c6,c7,c8,c9,c10,c11;
      // sums & carries generated by FA

      reg p[3:0][3:0];  // double array for partial products
      integer i,j;


      always@(a or b)
      begin
            for(i=0;i<=3;i=i+1)        // nested for loops.
               for(j=0;j<=3;j=j+1)
                  p[i][j]<= a[j] & b[i]; // to generate partial products.

      end

      Halfadder HA0(p[1][0],p[0][1],s0,c0);
      Fulladder FA0(p[0][2],p[1][1],p[2][0],s1,c1);
      Fulladder FA1(p[0][3],p[1][2],p[2][1],s2,c2);
      Halfadder HA1(p[1][3],p[2][2],s3,c3);

      Halfadder HA2(s1,c0,s4,c4);
      Fulladder FA2(p[3][0],s2,c1,s5,c5);
      Fulladder FA3(p[3][1],s3,c2,s6,c6);
      Fulladder FA4(p[3][2],p[2][3],c3,s7,c7);

      Halfadder HA3(s5,c4,s8,c8);
      Fulladder FA5(s6,c5,c8,s9,c9);
      Fulladder FA6(s7,c6,c9,s10,c10);
      Fulladder FA7(p[3][3],c7,c10,s11,c11);


   assign product[0]=p[0][0];
   assign product[1]=s0;
   assign product[2]=s4;
   assign product[3]=s8;
   assign product[4]=s9;
   assign product[5]=s10;
   assign product[6]=s11;
   assign product[7]=c11;


endmodule
```

Verilog test fixture code:

```
26  module WallaceMulTF;
27
28      // Inputs
29      reg [3:0] a;
30      reg [3:0] b;
31
32      // Outputs
33      wire [7:0] product;
34      // Instantiate the Unit Under Test (UUT)
35      WallaceMul uut (
36          .a(a),
37          .b(b),
38          .product(product)
39      );
40      initial begin
41          // Initialize Inputs
42          a = 0;
43          b = 0;
44          // Wait 100 ns for global reset to finish
45          #100;
46          // Add stimulus here
47          a=4'b1000;
48          b=4'b1000;
49      end
50  endmodule
```

WAVEFORM:



Lab Task 2

To Develop, Implement and Simulate 6x6 Dadda Wallance Tree.

Verilog Module code:

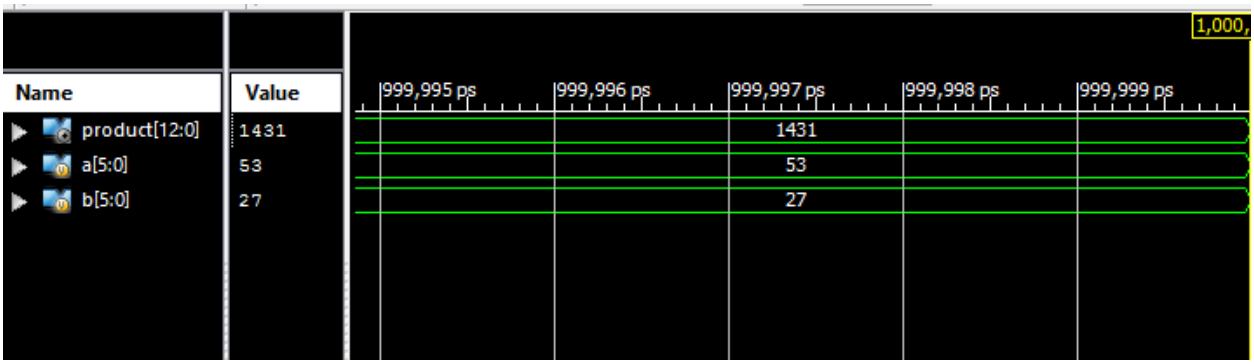
```
23  module WallaceMul(
24      input [7:0] a,
25      input [7:0] b,
26      output [12:0] product
27      );
28
29        wire s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,
30        s16,s17,s18,s19,s20,s21,s22,s23,s24,s25,s26,s27,s28,s29,
31        s30,s31,s32,s33,s34,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,
32        c12,c13,c14,c15,c16,c17,c18,c19,c20,c21,c22,c23,c24,c25,
33        c26,c27,c28,c29,c30,c31,c32,c33,c34;
34        // sums & carries generated by FA
35
36        reg p[7:0][7:0];  // double array for partial products
37        integer i,j;
38
39
40        always@(a or b)
41        begin
42              for(i=0;i<=5;i=i+1)         // nested for loops.
43                 for(j=0;j<=5;j=j+1)
44                    p[i][j]<= a[j] & b[i]; // to generate partial products.
45
46        end
```

```
47
48          Halfadder HA0 (p[1][0],p[0][1],s1,c1);
49          Fulladder FA0 (p[0][2],p[1][1],p[2][0],s2,c2);
50          Fulladder FA1 (p[0][3],p[1][2],p[2][1],s3,c3);
51          Fulladder FA2 (p[0][4],p[1][3],p[2][2],s4,c4);
52          Fulladder FA3 (p[0][5],p[1][4],p[2][3],s5,c5);
53          Halfadder HA1 (p[1][5],p[2][4],s6,c6);
54
55          Halfadder HA2 (p[3][1],p[4][0],s7,c7);
56          Fulladder FA4 (p[3][2],p[4][1],p[5][0],s8,c8);
57          Fulladder FA5 (p[3][3],p[4][2],p[5][1],s9,c9);
58          Fulladder FA6 (p[3][4],p[4][3],p[5][2],s10,c10);
59          Fulladder FA7 (p[3][5],p[4][4],p[5][3],s11,c11);
60          Halfadder HA3 (p[4][5],p[5][4],s12,c12);
61
62          Halfadder HA4 (s2,c1,s13,c13);
63          Fulladder FA8 (p[3][0],s3,c2,s14,c14);
64          Fulladder FA9 (s4,c3,s7,s15,c15);
65          Fulladder FA10 (s5,c4,s8,s16,c16);
66          Fulladder FA11 (s6,c5,s9,s17,c17);
67          Fulladder FA12 (p[2][5],c6,s10,s18,c18);
68
69          Halfadder HA5 (s14,c13,s19,c19);
70          Halfadder HA6 (s15,c14,s20,c20);
71          Fulladder FA13 (s16,c15,c7,s21,c21);
72          Fulladder FA14 (s17,c16,c8,s22,c22);
73          Fulladder FA15 (s18,c17,c9,s23,c23);
74          Fulladder FA16 (s11,c18,c10,s24,c24);
75          Halfadder HA7 (s12,c11,s25,c25);
76          Halfadder HA8 (p[5][5],c12,s26,c26);
77
78          Halfadder HA9 (s20,c19,s27,c27);
79          Fulladder FA17 (s21,c20,c27,s28,c28);
80          Fulladder FA18 (s22,c21,c28,s29,c29);
81          Fulladder FA19 (s23,c22,c29,s30,c30);
82          Fulladder FA20 (s24,c23,c30,s31,c31);
83          Fulladder FA21 (s25,c24,c31,s32,c32);
84          Fulladder FA22 (s26,c25,c32,s33,c33);
85          Halfadder HA10 (c26,c33,s34,c34);
86
87          assign product[0]=p[0][0];
88          assign product[1]=s1;
89          assign product[2]=s13;
90          assign product[3]=s19;
91          assign product[4]=s27;
92          assign product[5]=s28;
93          assign product[6]=s29;
94          assign product[7]=s30;
95          assign product[8]=s31;
96          assign product[9]=s32;
97          assign product[10]=s33;
98          assign product[11]=s34;
99          assign product[12]=c34;
100
101     endmodule
```

Verilog test fixture code:

```
26  module WallaceMulTF;
27      // Inputs
28      reg [7:0] a;
29      reg [7:0] b;
30      // Outputs
31      wire [12:0] product;
32      // Instantiate the Unit Under Test (UUT)
33      WallaceMul uut (
34          .a(a),
35          .b(b),
36          .product(product)
37      );
38      initial begin
39          // Initialize Inputs
40          a = 0;
41          b = 0;
42          // Wait 100 ns for global reset to finish
43          #100;
44          // Add stimulus here
45          a=6'b110101;
46          b=6'b011011;
47      end
48  endmodule
```

WAVEFORM:



HOME ASSIGNMENT

TASK:

To Develop, Implement and Simulate 8x8 Dadda Wallance Tree.

Verilog Module code:

```
module WallaceMul(
    input [7:0] a,
    input [7:0] b,
    output [16:0] product
    );

      wire
      s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,s21,
      s22,s23,s24,s25,s26,s27,s28,s29,s30,s31,s32,s33,s34,s35,s36,s37,s38,s39,s40,s41,
      s42,s43,s44,s45,s46,s47,s48,s49,s50,s51,s52,s53,s54,s55,s56,s57,s58,s59,s60,s61,
      s62,s63,s64,c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16,c17,c18,c19,c20,c21,
      c22,c23,c24,c25,c26,c27,c28,c29,c30,c31,c32,c33,c34,c35,c36,c37,c38,c39,c40,c41,
      c42,c43,c44,c45,c46,c47,c48,c49,c50,c51,c52,c53,c54,c55,c56,c57,c58,c59,c60,c61,c62,c63,c6
      // sums & carries generated by FA

      reg p[7:0][7:0];  // double array for partial products
      integer i,j;


      always@(a or b)
      begin
            for(i=0;i<=8;i=i+1)        // nested for loops.
               for(j=0;j<=8;j=j+1)
                  p[i][j]<= a[j] & b[i]; // to generate partial products.
```

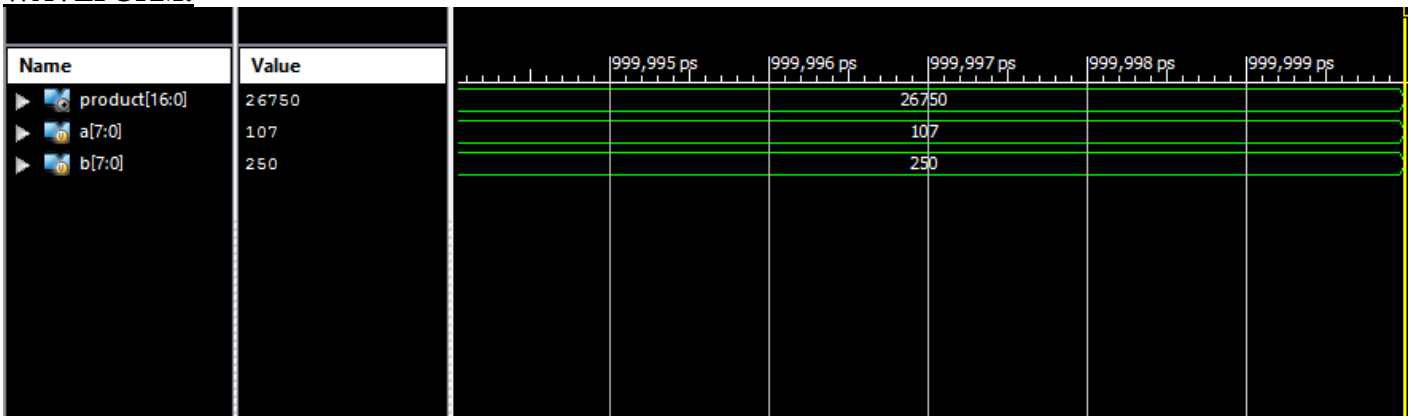
```
      end
      Halfadder HA0  (p[0][1],p[1][0],s1,c1);
      Fulladder FA0  (p[0][2],p[1][1],p[2][0],s2,c2);
      Fulladder FA1  (p[0][3],p[1][2],p[2][1],s3,c3);
      Fulladder FA2  (p[0][4],p[1][3],p[2][2],s4,c4);
      Fulladder FA3  (p[0][5],p[1][4],p[2][3],s5,c5);
      Fulladder FA4  (p[0][6],p[1][5],p[2][4],s6,c6);
      Fulladder FA5  (p[0][7],p[1][6],p[2][5],s7,c7);
      Halfadder HA1  (p[1][7],p[2][6],s8,c8);

      Halfadder HA2   (p[3][1],p[4][0],s9,c9);
      Fulladder FA6   (p[3][2],p[4][1],p[5][0],s10,c10);
      Fulladder FA7   (p[3][3],p[4][2],p[5][1],s11,c11);
      Fulladder FA8   (p[3][4],p[4][3],p[5][2],s12,c12);
      Fulladder FA9   (p[3][5],p[4][4],p[5][3],s13,c13);
      Fulladder FA10  (p[3][6],p[4][5],p[5][4],s14,c14);
      Fulladder FA11  (p[3][7],p[4][6],p[5][5],s15,c15);
      Halfadder HA3   (p[4][7],p[5][6],s16,c16);

      Halfadder HA4   (s2,c1,s17,c17);
      Fulladder FA12  (p[3][0],s3,c2,s18,c18);
      Fulladder FA13  (s4,c3,s9,s19,c19);
      Fulladder FA14  (s5,c4,s10,s20,c20);
      Fulladder FA15  (s6,c5,s11,s21,c21);
      Fulladder FA16  (s7,c6,s12,s22,c22);
      Fulladder FA18  (s8,c7,s13,s23,c23);
      Fulladder FA19  (p[2][7],c8,s14,s24,c24);
```

```
76          Halfadder HA5   (p[6][0],c10,s25,c25);
77          Fulladder FA20  (p[6][1],p[7][0],c11,s26,c26);
78          Fulladder FA21  (p[6][2],p[7][1],c12,s27,c27);
79          Fulladder FA22  (p[6][3],p[7][2],c13,s28,c28);
80          Fulladder FA23  (p[6][4],p[7][3],c14,s29,c29);
81          Fulladder FA24  (p[6][5],p[7][4],c15,s30,c30);
82          Fulladder FA25  (p[6][6],p[7][5],c16,s31,c31);
83          Halfadder HA6   (p[6][7],p[7][6],s32,c32);
84
85          Halfadder HA7   (s18,c17,s33,c33);
86          Halfadder HA8   (s19,c18,s34,c34);
87          Fulladder FA26  (s20,c19,c9,s35,c35);
88          Fulladder FA27  (s21,c20,s25,s36,c36);
89          Fulladder FA28  (s22,c21,s26,s37,c37);
90          Fulladder FA29  (s23,c22,s27,s38,c38);
91          Fulladder FA30  (s24,c23,s28,s39,c39);
92          Fulladder FA31  (s15,c24,s29,s40,c40);
93          Halfadder HA9   (s16,s30,s41,c41);
94          Halfadder HA10  (p[5][7],s31,s42,c42);
95
96           Halfadder HA11  (s34,c33,s43,c43);
97           Halfadder HA12  (s35,c34,s44,c44);
98           Halfadder HA13  (s36,c35,s45,c45);
99           Fulladder FA32  (s37,c36,c25,s46,c46);
100          Fulladder FA33  (s38,c37,c26,s47,c47);
101          Fulladder FA34  (s39,c38,c27,s48,c48);
102          Fulladder FA35  (s40,c39,c28,s49,c49);
103          Fulladder FA36  (s41,c40,c29,s50,c50);
104          Fulladder FA37  (s42,c41,c30,s51,c51);
105          Fulladder FA38  (s32,c42,c31,s52,c52);
106          Halfadder HA14  (p[7][7],c32,s53,c53);
107
108          Halfadder HA15  (s44,c43,s54,c54);
109          Fulladder FA39  (s45,c44,c54,s55,c55);
110          Fulladder FA40  (s46,c45,c55,s56,c56);
111          Fulladder FA41  (s47,c46,c56,s57,c57);
112          Fulladder FA42  (s48,c47,c57,s58,c58);
113          Fulladder FA43  (s49,c48,c58,s59,c59);
114          Fulladder FA44  (s50,c49,c59,s60,c60);
115          Fulladder FA45  (s51,c50,c60,s61,c61);
116          Fulladder FA46  (s52,c51,c61,s62,c62);
117          Fulladder FA47  (s53,c52,c62,s63,c63);
118          Halfadder HA16  (c53,c63,s64,c64);
119
120
121          assign product[0]=p[0][0];
122          assign product[1]=s1;
123          assign product[2]=s17;
124          assign product[3]=s33;
125          assign product[4]=s43;
126          assign product[5]=s54;
127          assign product[6]=s55;
128          assign product[7]=s56;
129          assign product[8]=s57;
130          assign product[9]=s58;
131          assign product[10]=s59;
132          assign product[11]=s60;
133          assign product[12]=s61;
134          assign product[13]=s62;
135          assign product[14]=s63;
136          assign product[15]=s64;
137          assign product[16]=c64;
138
139
140     endmodule
```

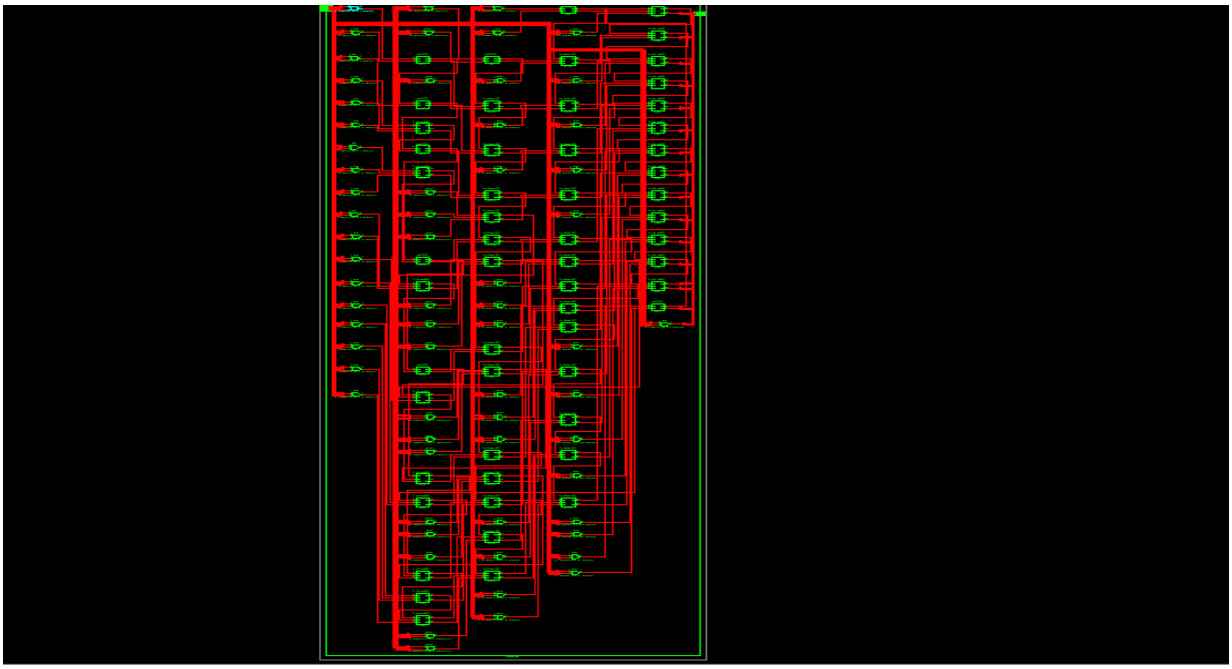
Verilog test fixture code:

```
26      module WallaceMulTF;
27
28          // Inputs
29          reg [7:0] a;
30          reg [7:0] b;
31          // Outputs
32          wire [16:0] product;
33          // Instantiate the Unit Under Test (UUT)
34          WallaceMul uut (
35              .a(a),
36              .b(b),
37              .product(product)
38          );
39          initial begin
40              // Initialize Inputs
41              a = 0;
42              b = 0;
43              // Wait 100 ns for global reset to finish
44              #100;
45              // Add stimulus here
46              a=8'b01101011;
47              b=8'b11111010;
48
49          end
50      endmodule
```

WAVEFORM:



Schematic Diagram:



CONCLUSION:

In this lab, We have developed 4x4 and 6x6 wallace tree in our lab task. Developing 4x4 wallace tree takes 4 bit values as input and generates 8 bits output. Similiarly developing 6x6 wallace tree takes 6 bit values as input and generates 12 bits output. We Developed 8x8 wallace tree in our home assignment which takes 8 bit values as input and generates 16 bits output.

So in general we learn how to multiply integers as binary bits using Dadda Wallance Tree. Using the ISE Design Suite, we also utilized data flow modelling to simulate multiplying. We also produced a wave output in which we can see the characteristics of the output after performing multiplication.