Self-Project

Topic: Implementation of 8x8 bit Dadda Multiplier

Name: Vipin Kumar

Roll No: 213070102

Mtech 2nd year - EE7(SSD)

Dadda Multipliers :

1. Generate all bits of the partial products in parallel.

2. Collect all partial products bits with the same place value in bunches of wires and reduce these in several layers of adders till each weight has no more than two wires.

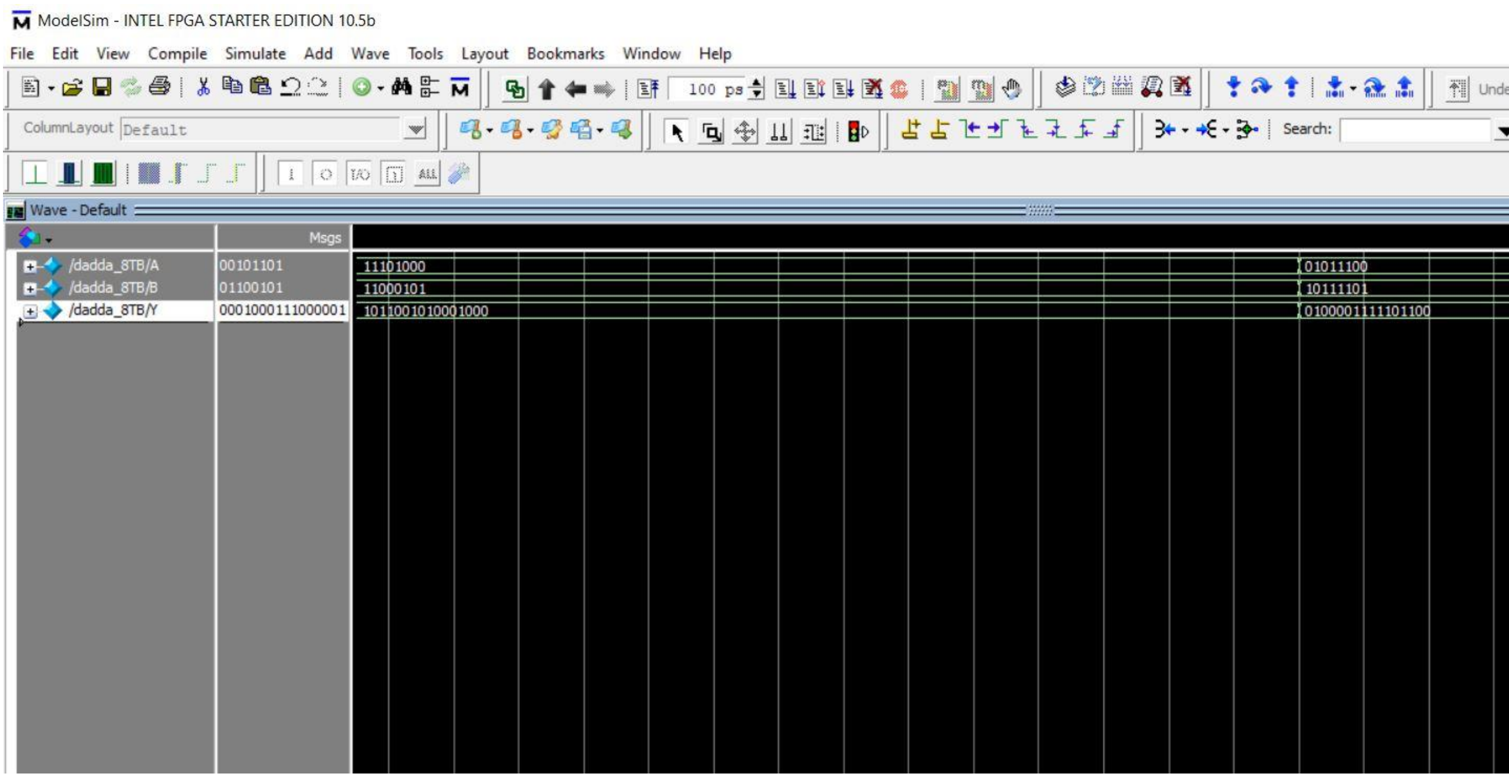
3. For all bit positions which have two wires, take one wire at corresponding place values to form one number, and the other wire to form another number.

Add these two numbers using a fast adder of appropriate size.

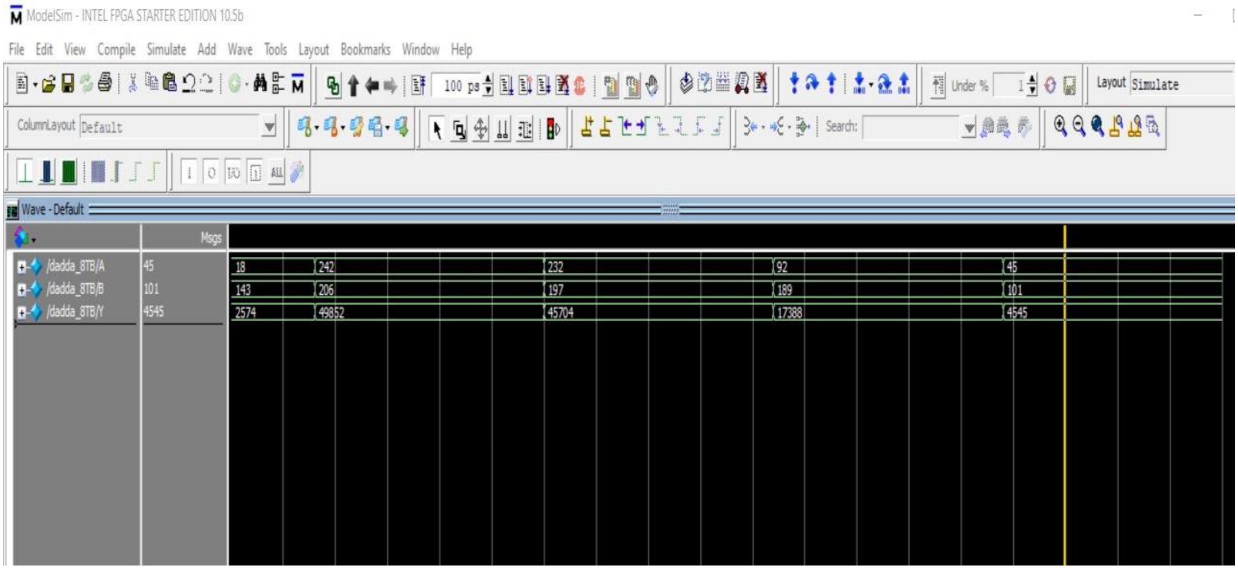
- Dadda multipliers are very similar to Wallace multipliers
    - Wallace multipliers reduce as soon as possible, while Dadda multipliers reduce as late as possible.
    - Dadda multipliers plan on reducing the final number of wires for any weight to 2 with as few and as small adders as possible.
    - We determine the number of layers required first, beginning from the last layer, where no more than 2 wires should be left.
    - The number of layers in Dadda multipliers is the same as in Wallace multipliers.
- We work back from the final adder to earlier layers till we find that we can manage all wires generated by the partial product generator.
- We know that the final adder can take no more than 2 wires for each weight
- Let dj represent the maximum number of wires for any weight in layer j, where j = 1 for the final adder. (Thus d1 = 2).
- The maximum number of wires which can be handled in layer j+1 (from the end) is the integral part of 3/2dj .

- We go up in j, till we reach a number which is just greater than or equal to the largest bunch of wires in any weight. The number of reduction layers required is this final j -1.

Simulated Result in Binary :



Simulated result in Unsigned format :




```
# run -all
#  A= 36,B=129,AxB= 4644
#  A=  9,B= 99,AxB=  891
#  A= 13,B=141,AxB= 1833
#  A=101,B= 18,AxB= 1818
#  A=  1,B= 13,AxB=   13
#  A=118,B= 61,AxB= 7198
#  A=237,B=140,AxB=33180
#  A=249,B=198,AxB=49302
#  A=197,B=170,AxB=33490
#  A=229,B=119,AxB=27251
#  A= 18,B=143,AxB= 2574
#  A=242,B=206,AxB=49852
#  A=232,B=197,AxB=45704
#  A= 92,B=189,AxB=17388
#  A= 45,B=101,AxB= 4545
```

Verilog code :

```
// Designing in Half Adder
// Sum = a XOR b, Cout = a AND b



module HA(a, b, Sum, Cout);
```

```
input a, b; // a and b are inputs with size 1-bit
output Sum, Cout; // Sum and Cout are outputs with size 1-bit

assign Sum = a ^ b;
assign Cout = a & b;

endmodule



//carry save adder -- for implementing dadda multiplier
//csa for use of half adder and full adder.

module csa_dadda(A,B,Cin,Y,Cout);
input A,B,Cin;
output Y,Cout;

assign Y = A^B^Cin;
assign Cout = (A&B)|(A&Cin)|(B&Cin);

endmodule



// dadda multiplier
```

```
// A - 8 bits , B - 8bits, y(output) - 16bits

module dadda(A,B,y);

   input [7:0] A;
   input [7:0] B;
   output wire [15:0] y;
   wire  gen_pp [0:7][7:0];
// stage-1 sum and carry
   wire [0:5]s1,c1;
// stage-2 sum and carry
   wire [0:13]s2,c2;
// stage-3 sum and carry
   wire [0:9]s3,c3;
// stage-4 sum and carry
   wire [0:11]s4,c4;
// stage-5 sum and carry
   wire [0:13]s5,c5;



// generating partial products
```

```
////////////////////////////////

//// j=0
assign gen_pp[0][0]  = A[0]*B[0];
assign gen_pp[1][0]  = A[0]*B[1];
assign gen_pp[2][0]  = A[0]*B[2];
assign gen_pp[3][0]  = A[0]*B[3];
assign gen_pp[4][0]  = A[0]*B[4];
assign gen_pp[5][0]  = A[0]*B[5];
assign gen_pp[6][0]  = A[0]*B[6];
assign gen_pp[7][0]  = A[0]*B[7];


//// j=1
assign gen_pp[0][1]  = A[1]*B[0];
assign gen_pp[1][1]  = A[1]*B[1];
assign gen_pp[2][1]  = A[1]*B[2];
assign gen_pp[3][1]  = A[1]*B[3];
assign gen_pp[4][1]  = A[1]*B[4];
assign gen_pp[5][1]  = A[1]*B[5];
assign gen_pp[6][1]  = A[1]*B[6];
assign gen_pp[7][1]  = A[1]*B[7];
```

```
//// j=2
assign gen_pp[0][2]  = A[2]*B[0];
assign gen_pp[1][2]  = A[2]*B[1];
assign gen_pp[2][2]  = A[2]*B[2];
assign gen_pp[3][2]  = A[2]*B[3];
assign gen_pp[4][2]  = A[2]*B[4];
assign gen_pp[5][2]  = A[2]*B[5];
assign gen_pp[6][2]  = A[2]*B[6];
assign gen_pp[7][2]  = A[2]*B[7];

//// j=3
assign gen_pp[0][3]  = A[3]*B[0];
assign gen_pp[1][3]  = A[3]*B[1];
assign gen_pp[2][3]  = A[3]*B[2];
assign gen_pp[3][3]  = A[3]*B[3];
assign gen_pp[4][3]  = A[3]*B[4];
assign gen_pp[5][3]  = A[3]*B[5];
assign gen_pp[6][3]  = A[3]*B[6];
assign gen_pp[7][3]  = A[3]*B[7];


//// j=4
assign gen_pp[0][4]  = A[4]*B[0];
assign gen_pp[1][4]  = A[4]*B[1];
```

```
assign gen_pp[2][4]  = A[4]*B[2];
assign gen_pp[3][4]  = A[4]*B[3];
assign gen_pp[4][4]  = A[4]*B[4];
assign gen_pp[5][4]  = A[4]*B[5];
assign gen_pp[6][4]  = A[4]*B[6];
assign gen_pp[7][4]  = A[4]*B[7];


//// j=5
assign gen_pp[0][5]  = A[5]*B[0];
assign gen_pp[1][5]  = A[5]*B[1];
assign gen_pp[2][5]  = A[5]*B[2];
assign gen_pp[3][5]  = A[5]*B[3];
assign gen_pp[4][5]  = A[5]*B[4];
assign gen_pp[5][5]  = A[5]*B[5];
assign gen_pp[6][5]  = A[5]*B[6];
assign gen_pp[7][5]  = A[5]*B[7];


//// j=6
assign gen_pp[0][6]  = A[6]*B[0];
assign gen_pp[1][6]  = A[6]*B[1];
assign gen_pp[2][6]  = A[6]*B[2];
assign gen_pp[3][6]  = A[6]*B[3];
```

```
assign gen_pp[4][6]  = A[6]*B[4];
assign gen_pp[5][6]  = A[6]*B[5];
assign gen_pp[6][6]  = A[6]*B[6];
assign gen_pp[7][6]  = A[6]*B[7];



//// j=7
assign gen_pp[0][7]  = A[7]*B[0];
assign gen_pp[1][7]  = A[7]*B[1];
assign gen_pp[2][7]  = A[7]*B[2];
assign gen_pp[3][7]  = A[7]*B[3];
assign gen_pp[4][7]  = A[7]*B[4];
assign gen_pp[5][7]  = A[7]*B[5];
assign gen_pp[6][7]  = A[7]*B[6];
assign gen_pp[7][7]  = A[7]*B[7];



/////////////////////////////////




//Reduction by stages.
// di_values = 2,3,4,6,8,13...
```

```
//Stage 1 - reducing fom 8 to 6

   HA h1(.a(gen_pp[6][0]),.b(gen_pp[5][1]),.Sum(s1[0]),.Cout(c1[0]));
   HA h2(.a(gen_pp[4][3]),.b(gen_pp[3][4]),.Sum(s1[2]),.Cout(c1[2]));
   HA h3(.a(gen_pp[4][4]),.b(gen_pp[3][5]),.Sum(s1[4]),.Cout(c1[4]));

   csa_dadda
c11(.A(gen_pp[7][0]),.B(gen_pp[6][1]),.Cin(gen_pp[5][2]),.Y(s1[1]),.C
out(c1[1]));
   csa_dadda
c12(.A(gen_pp[7][1]),.B(gen_pp[6][2]),.Cin(gen_pp[5][3]),.Y(s1[3]),.C
out(c1[3]));
   csa_dadda
c13(.A(gen_pp[7][2]),.B(gen_pp[6][3]),.Cin(gen_pp[5][4]),.Y(s1[5]),.C
out(c1[5]));

//Stage 2 - reducing fom 6 to 4

   HA h4(.a(gen_pp[4][0]),.b(gen_pp[3][1]),.Sum(s2[0]),.Cout(c2[0]));
   HA h5(.a(gen_pp[2][3]),.b(gen_pp[1][4]),.Sum(s2[2]),.Cout(c2[2]));
```

```
csa_dadda c21(.A(gen_pp[5][0]),.B(gen_pp[4][1]),.Cin(gen_pp[3][2]),.Y(s2[1]),.Cout(c2[1]));
csa_dadda c22(.A(s1[0]),.B(gen_pp[4][2]),.Cin(gen_pp[3][3]),.Y(s2[3]),.Cout(c2[3]));
csa_dadda c23(.A(gen_pp[2][4]),.B(gen_pp[1][5]),.Cin(gen_pp[0][6]),.Y(s2[4]),.Cout(c2[4]));
csa_dadda c24(.A(s1[1]),.B(s1[2]),.Cin(c1[0]),.Y(s2[5]),.Cout(c2[5]));
csa_dadda c25(.A(gen_pp[2][5]),.B(gen_pp[1][6]),.Cin(gen_pp[0][7]),.Y(s2[6]),.Cout(c2[6]));
csa_dadda c26(.A(s1[3]),.B(s1[4]),.Cin(c1[1]),.Y(s2[7]),.Cout(c2[7]));
csa_dadda c27(.A(c1[2]),.B(gen_pp[2][6]),.Cin(gen_pp[1][7]),.Y(s2[8]),.Cout(c2[8]));
csa_dadda c28(.A(s1[5]),.B(c1[3]),.Cin(c1[4]),.Y(s2[9]),.Cout(c2[9]));
csa_dadda c29(.A(gen_pp[4][5]),.B(gen_pp[3][6]),.Cin(gen_pp[2][7]),.Y(s2[10]),.Cout(c2[10]));
csa_dadda c210(.A(gen_pp[7][3]),.B(c1[5]),.Cin(gen_pp[6][4]),.Y(s2[11]),.Cout(c2[11]));
csa_dadda c211(.A(gen_pp[5][5]),.B(gen_pp[4][6]),.Cin(gen_pp[3][7]),.Y(s2[12]),.Cout(c2[12]));
```

```
    csa_dadda
c212(.A(gen_pp[7][4]),.B(gen_pp[6][5]),.Cin(gen_pp[5][6]),.Y(s2[13]),.
Cout(c2[13]));


//Stage 3 - reducing fom 4 to 3


    HA h6(.a(gen_pp[3][0]),.b(gen_pp[2][1]),.Sum(s3[0]),.Cout(c3[0]));


    csa_dadda
c31(.A(s2[0]),.B(gen_pp[2][2]),.Cin(gen_pp[1][3]),.Y(s3[1]),.Cout(c3[1]
));
    csa_dadda c32(.A(s2[1]),.B(s2[2]),.Cin(c2[0]),.Y(s3[2]),.Cout(c3[2]));
    csa_dadda c33(.A(c2[1]),.B(c2[2]),.Cin(s2[3]),.Y(s3[3]),.Cout(c3[3]));
    csa_dadda c34(.A(c2[3]),.B(c2[4]),.Cin(s2[5]),.Y(s3[4]),.Cout(c3[4]));
    csa_dadda c35(.A(c2[5]),.B(c2[6]),.Cin(s2[7]),.Y(s3[5]),.Cout(c3[5]));
    csa_dadda c36(.A(c2[7]),.B(c2[8]),.Cin(s2[9]),.Y(s3[6]),.Cout(c3[6]));
    csa_dadda
c37(.A(c2[9]),.B(c2[10]),.Cin(s2[11]),.Y(s3[7]),.Cout(c3[7]));
    csa_dadda
c38(.A(c2[11]),.B(c2[12]),.Cin(s2[13]),.Y(s3[8]),.Cout(c3[8]));
    csa_dadda
c39(.A(gen_pp[7][5]),.B(gen_pp[6][6]),.Cin(gen_pp[5][7]),.Y(s3[9]),.C
out(c3[9]));


//Stage 4 - reducing fom 3 to 2
```

```
    HA h7(.a(gen_pp[2][0]),.b(gen_pp[1][1]),.Sum(s4[0]),.Cout(c4[0]));



    csa_dadda
c41(.A(s3[0]),.B(gen_pp[1][2]),.Cin(gen_pp[0][3]),.Y(s4[1]),.Cout(c4[1]
));
    csa_dadda
c42(.A(c3[0]),.B(s3[1]),.Cin(gen_pp[0][4]),.Y(s4[2]),.Cout(c4[2]));
    csa_dadda
c43(.A(c3[1]),.B(s3[2]),.Cin(gen_pp[0][5]),.Y(s4[3]),.Cout(c4[3]));
    csa_dadda c44(.A(c3[2]),.B(s3[3]),.Cin(s2[4]),.Y(s4[4]),.Cout(c4[4]));
    csa_dadda c45(.A(c3[3]),.B(s3[4]),.Cin(s2[6]),.Y(s4[5]),.Cout(c4[5]));
    csa_dadda c46(.A(c3[4]),.B(s3[5]),.Cin(s2[8]),.Y(s4[6]),.Cout(c4[6]));
    csa_dadda
c47(.A(c3[5]),.B(s3[6]),.Cin(s2[10]),.Y(s4[7]),.Cout(c4[7]));
    csa_dadda
c48(.A(c3[6]),.B(s3[7]),.Cin(s2[12]),.Y(s4[8]),.Cout(c4[8]));
    csa_dadda
c49(.A(c3[7]),.B(s3[8]),.Cin(gen_pp[4][7]),.Y(s4[9]),.Cout(c4[9]));
    csa_dadda
c410(.A(c3[8]),.B(s3[9]),.Cin(c2[13]),.Y(s4[10]),.Cout(c4[10]));
    csa_dadda
c411(.A(c3[9]),.B(gen_pp[7][6]),.Cin(gen_pp[6][7]),.Y(s4[11]),.Cout(c4
[11]));


//Stage 5 - reducing fom 2 to 1
    // adding total sum and carry to get final output
```

```
HA h8(.a(gen_pp[1][0]),.b(gen_pp[0][1]),.Sum(y[1]),.Cout(c5[0]));



csa_dadda
c51(.A(s4[0]),.B(gen_pp[0][2]),.Cin(c5[0]),.Y(y[2]),.Cout(c5[1]));
csa_dadda c52(.A(c4[0]),.B(s4[1]),.Cin(c5[1]),.Y(y[3]),.Cout(c5[2]));
csa_dadda c54(.A(c4[1]),.B(s4[2]),.Cin(c5[2]),.Y(y[4]),.Cout(c5[3]));
csa_dadda c55(.A(c4[2]),.B(s4[3]),.Cin(c5[3]),.Y(y[5]),.Cout(c5[4]));
csa_dadda c56(.A(c4[3]),.B(s4[4]),.Cin(c5[4]),.Y(y[6]),.Cout(c5[5]));
csa_dadda c57(.A(c4[4]),.B(s4[5]),.Cin(c5[5]),.Y(y[7]),.Cout(c5[6]));
csa_dadda c58(.A(c4[5]),.B(s4[6]),.Cin(c5[6]),.Y(y[8]),.Cout(c5[7]));
csa_dadda c59(.A(c4[6]),.B(s4[7]),.Cin(c5[7]),.Y(y[9]),.Cout(c5[8]));
csa_dadda
c510(.A(c4[7]),.B(s4[8]),.Cin(c5[8]),.Y(y[10]),.Cout(c5[9]));
csa_dadda
c511(.A(c4[8]),.B(s4[9]),.Cin(c5[9]),.Y(y[11]),.Cout(c5[10]));
csa_dadda
c512(.A(c4[9]),.B(s4[10]),.Cin(c5[10]),.Y(y[12]),.Cout(c5[11]));
csa_dadda
c513(.A(c4[10]),.B(s4[11]),.Cin(c5[11]),.Y(y[13]),.Cout(c5[12]));
csa_dadda
c514(.A(c4[11]),.B(gen_pp[7][7]),.Cin(c5[12]),.Y(y[14]),.Cout(c5[13]));
```

```
    assign y[0] =  gen_pp[0][0];
    assign y[15] = c5[13];



endmodule
```