

Ömer Faruk SERT
040170244

1. Unsigned Structural Multiplier

1-) Verilog Code

Module Code

```
`timescale 1ns / 1ps

module MULTS(
    input [7:0]A,
    input [7:0]X,
    output [15:0]result
);
    wire [15:0] PP0,PP1,PP2,PP3,PP4,PP5,PP6,PP7;
    wire [15:0] sum1,sum2,sum3,sum4,sum5,sum6,sum7;

    assign PP0 = {X[0],X[0],X[0],X[0],X[0],X[0],X[0],X[0]} & A;
    assign PP1 = ({X[1],X[1],X[1],X[1],X[1],X[1],X[1],X[1]} & A)<<1;
    assign PP2 = ({X[2],X[2],X[2],X[2],X[2],X[2],X[2],X[2]} & A)<<2;
    assign PP3 = ({X[3],X[3],X[3],X[3],X[3],X[3],X[3],X[3]} & A)<<3;
    assign PP4 = ({X[4],X[4],X[4],X[4],X[4],X[4],X[4],X[4]} & A)<<4;
    assign PP5 = ({X[5],X[5],X[5],X[5],X[5],X[5],X[5],X[5]} & A)<<5;
    assign PP6 = ({X[6],X[6],X[6],X[6],X[6],X[6],X[6],X[6]} & A)<<6;
    assign PP7 = ({X[7],X[7],X[7],X[7],X[7],X[7],X[7],X[7]} & A)<<7;

    //1st stage
    CLA16 cla1 (.a(PP0),.b(PP1),.cin(1'b0),.sum(sum1));
    CLA16 cla2 (.a(PP2),.b(PP3),.cin(1'b0),.sum(sum2));

    CLA16 cla3(.a(PP4),.b(PP5),.cin(1'b0),.sum(sum3));
    CLA16 cla4(.a(PP6),.b(PP7),.cin(1'b0),.sum(sum4));

    //2nd stage
    CLA16 cla5(.a(sum1),.b(sum2),.cin(1'b0),.sum(sum5));
    CLA16 cla6(.a(sum3),.b(sum4),.cin(1'b0),.sum(sum6));

    //3rd stage
    CLA16 cla7(.a(sum6),.b(sum5),.cin(1'b0),.sum(sum7));

    assign result = sum7;
```

```

endmodule

module CLA16(a,b, cin, sum,cout);
input [15:0] a,b;
input cin;
output [15:0] sum;
output cout;
wire c1,c2,c3;

CLA cla1 (.X(a[3:0]), .Y(b[3:0]), .c0(cin), .S(sum[3:0]), .cout(c1));
CLA cla2 (.X(a[7:4]), .Y(b[7:4]), .c0(c1), .S(sum[7:4]), .cout(c2));
CLA cla3(.X(a[11:8]), .Y(b[11:8]), .c0(c2), .S(sum[11:8]), .cout(c3));
CLA cla4(.X(a[15:12]), .Y(b[15:12]), .c0(c3), .S(sum[15:12]), .cout(cout));

endmodule

```

Testbench Code

```

`timescale 1ns / 1ps

module testbench7;
    reg [7:0]A;
    reg [7:0]X;
    wire [15:0]P;

    MULTS uut(.A(A),.X(X),.result(P));

    initial
    begin
        A=8'b10101010;
        X=8'b10101010;
        #10
        A=8'b11111111;
        X=8'b11111111;
        #10
        A=8'b10101011;
        X=8'b10101111;
        #10
        A=8'b10101010;
        X=8'b11111010;
        #10
    end
endmodule

```

```

    $finish;
end

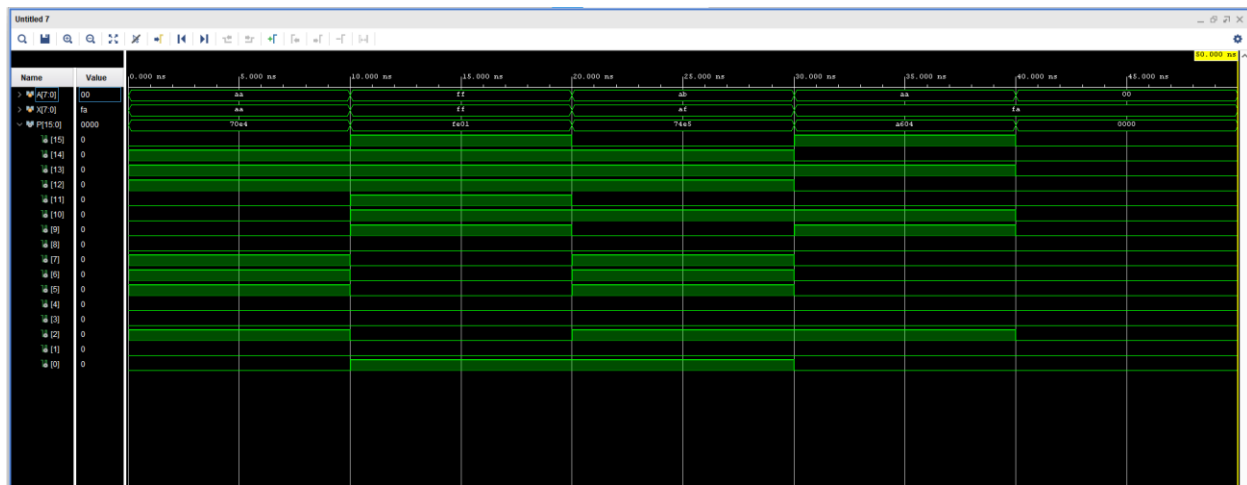
endmodule

```

2-) Simulation Results

Wave Form

Post implementation functional simulation



Tcl Console Output

2-) Signed Structural Multiplier

1- Verilog Code

Module Code

```

module MULTS_signed(
    input [7:0]A,
    input [7:0]X,
    output [15:0]result
);
    reg [15:0]Y=16'b0000000100000000;

```

```

wire [15:0]PP00;
wire [7:0] PP0,PP1,PP2,PP3,PP4,PP5,PP6,PP7;
wire [15:0] sum1,sum2,sum3,sum4,sum5,sum6,sum7;
wire [15:0]
PPshift0,PPshift1,PPshift2,PPshift3,PPshift4,PPshift5,PPshift6,PPshift7;

assign PP0[6:0]= {X[0],X[0],X[0],X[0],X[0],X[0],X[0]} & A[6:0];
assign PP0[7]= ~(X[0]&A[7]);

assign PP1[6:0]= {X[1],X[1],X[1],X[1],X[1],X[1],X[1]} & A[6:0];
assign PP1[7]= ~(X[1]&A[7]);

assign PP2[6:0] = {X[2],X[2],X[2],X[2],X[2],X[2],X[2]} & A[6:0];
assign PP2[7] = ~(X[2]&A[7]);

assign PP3[6:0] = {X[3],X[3],X[3],X[3],X[3],X[3],X[3]} & A[6:0];
assign PP3[7] = ~(X[3]&A[7]);

assign PP4[6:0] = {X[4],X[4],X[4],X[4],X[4],X[4],X[4]} & A[6:0];
assign PP4[7] = ~(X[4]&A[7]);

assign PP5[6:0] = {X[5],X[5],X[5],X[5],X[5],X[5],X[5]} & A[6:0];
assign PP5[7] = ~(X[5]&A[7]);

assign PP6[6:0] = ({X[6],X[6],X[6],X[6],X[6],X[6],X[6]} & A[6:0]);
assign PP6[7] = ~(X[6]&A[7]);

assign PP7[6:0] = ~({X[7],X[7],X[7],X[7],X[7],X[7],X[7]} & A[6:0]);
assign PP7[7] = (X[7]&A[7]);

assign PPshift0[15:0] = {8'b00000000,PP0[7:0]};
assign PPshift1[15:0] = {7'b00000000,PP1[7:0],1'b0};
assign PPshift2[15:0] = {6'b00000000 ,PP2[7:0],2'b00};
assign PPshift3[15:0] = {5'b000000,PP3[7:0],3'b000};
assign PPshift4[15:0] = {4'b0000,PP4[7:0],4'b0000};
assign PPshift5[15:0] = {3'b000,PP5[7:0],5'b00000};
assign PPshift6[15:0] = {2'b00,PP6[7:0],6'b000000};
assign PPshift7[15:0] = {1'b0,PP7[7:0],7'b0000000};

//additional step

```

```

CLA16 clai(.a(PPshift0),.b(Y),.cin(1'b0),.sum(PP00));

//1st stage
CLA16 cla1 (.a(PP00),.b(PPshift1),.cin(1'b0),.sum(sum1));
CLA16 cla2 (.a(PPshift2),.b(PPshift3),.cin(1'b0),.sum(sum2));

CLA16 cla3(.a(PPshift4),.b(PPshift5),.cin(1'b0),.sum(sum3));
CLA16 cla4(.a(PPshift6),.b(PPshift7),.cin(1'b0),.sum(sum4));

//2nd stage
CLA16 cla5(.a(sum1),.b(sum2),.cin(1'b0),.sum(sum5));
CLA16 cla6(.a(sum3),.b(sum4),.cin(1'b0),.sum(sum6));

//3rd stage
CLA16 cla7(.a(sum6),.b(sum5),.cin(1'b0),.sum(sum7));

assign result = {~sum7[15],sum7[14:0]};

endmodule

```

Testbench Code

```

module tbMULTSigned;
    reg [7:0]A;
    reg [7:0]X;
    wire [15:0]P;

    MULTS_signed uut(.A(A),.X(X),.result(P));

    initial
    begin
        A=8'b10101010;
        X=8'b00101010;
        #10
        A=8'b11111111;
        X=8'b11111111;
        #10
        A=8'b00101011;
        X=8'b00101111;
        #10
        A=8'b10101010;
        X=8'b11111010;
    end
endmodule

```

```

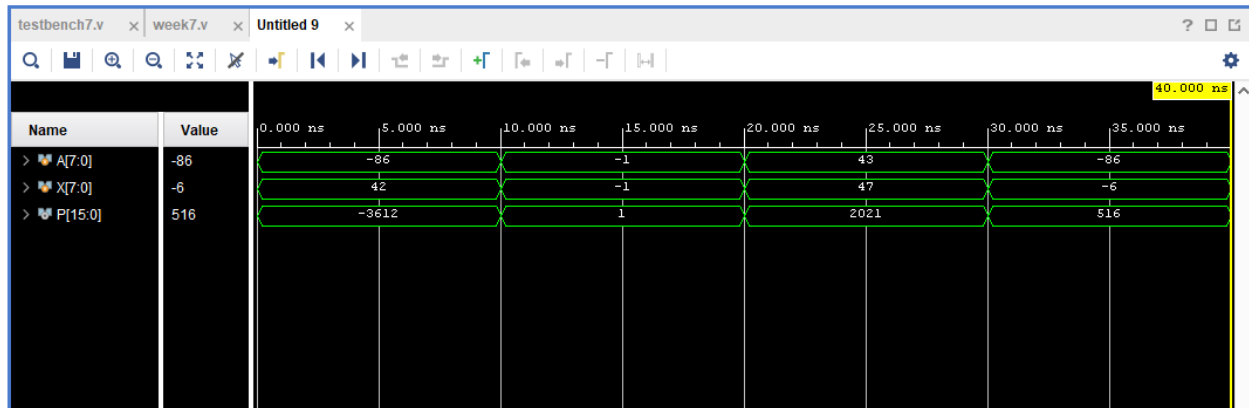
#10
$finish;
end

endmodule

```

2-)Simulation Results

Wave form



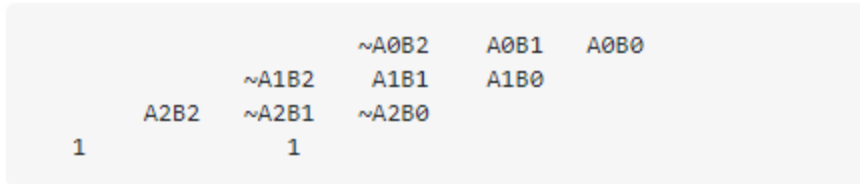
3-)Baugh Wooley Method

Explanation of Stages

4 stages are needed because as shown in the figure in the experiment file, there is additional ones in specific bits so we need to add them in a stage.

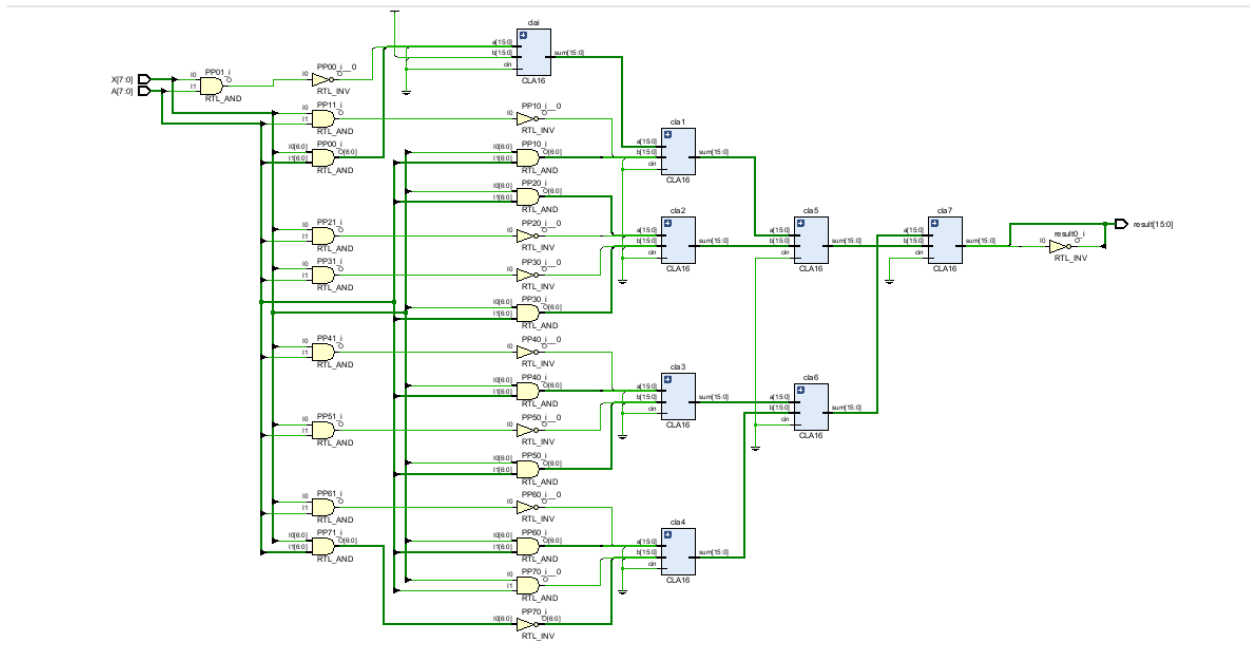
Explanation of Baugh Wooley Method for 3-bit inputs

We can extend the numbers we multiply and when we extend a signed number we extend it with replicating MSB. When we do that first partial products MSB is repeated for 4 times for 3-bit inputs. And it corresponds to MSB of first partial product *15, and also 15 is same as -1, so first two lines of partial products MSBs are multiplied with -1. And the last partial product's all bits except MSB is multiplied with -1. This also means that we can subtract -2 for every negation column it corresponds figure.

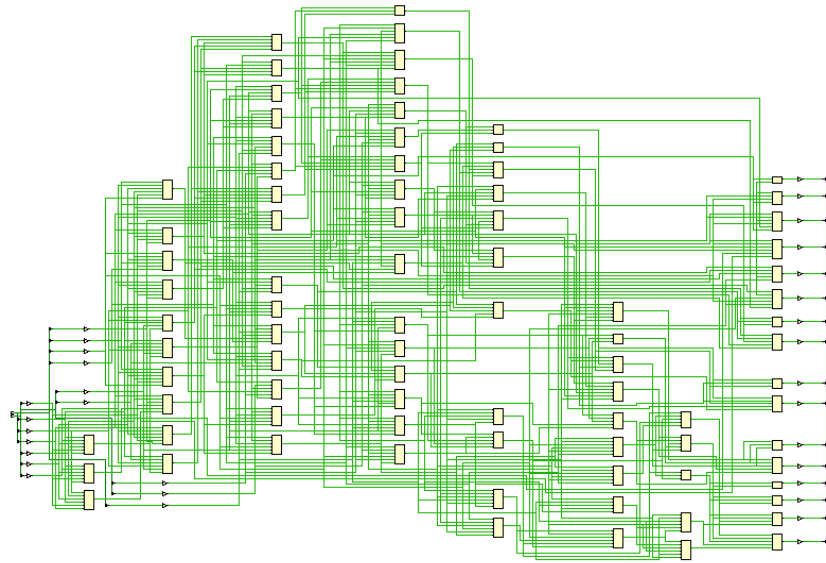


4-)RTL and Technology Schematics

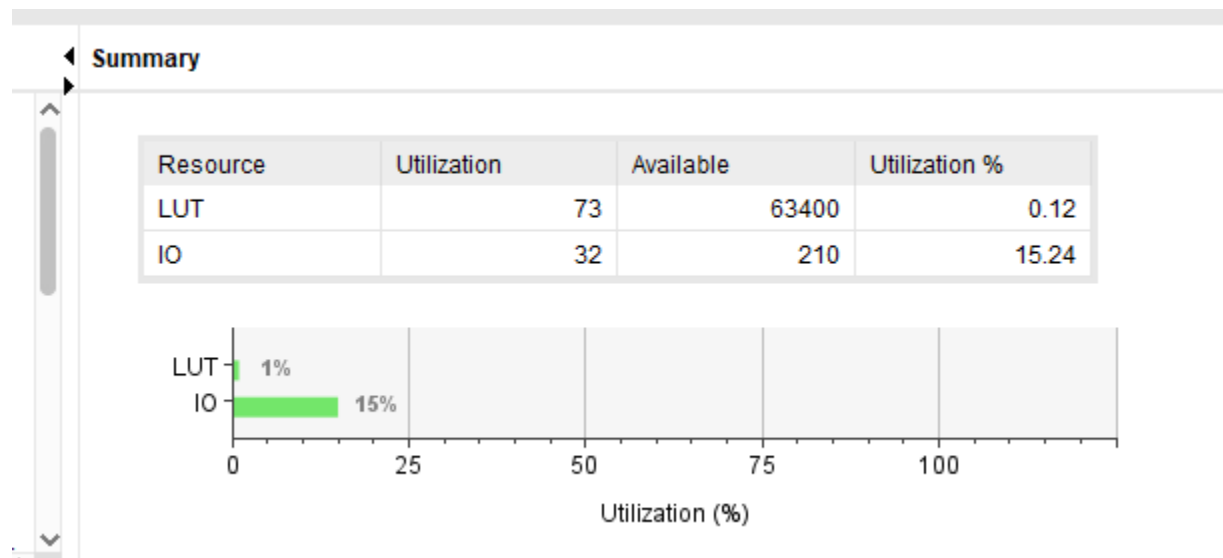
RTL Schematic



Technology Schematic



Utilization Report



Timing Report

Those are the maximum delays

From Port	To Port	Max Process Corner	Min Delay	Min Process Corner
A[1]	result[15]	16.428	SLOW	3.835
A[1]	result[14]	16.374	SLOW	3.785
A[1]	result[13]	16.360	SLOW	3.794
A[1]	result[12]	16.187	SLOW	3.725
A[0]	result[15]	15.962	SLOW	3.658
A[0]	result[14]	15.908	SLOW	3.608
A[0]	result[13]	15.894	SLOW	3.617
A[0]	result[12]	15.722	SLOW	3.548
A[1]	result[11]	15.492	SLOW	3.451
A[2]	result[15]	15.457	SLOW	3.739
A[2]	result[14]	15.403	SLOW	3.738
A[2]	result[13]	15.389	SLOW	3.802
X[1]	result[15]	15.224	SLOW	3.766
A[2]	result[12]	15.216	SLOW	3.732
X[1]	result[14]	15.170	SLOW	3.716
X[1]	result[13]	15.157	SLOW	3.726
A[0]	result[11]	15.026	SLOW	3.276
X[1]	result[12]	14.984	SLOW	3.656
A[3]	result[15]	14.928	SLOW	3.754
X[0]	result[15]	14.888	SLOW	3.799
A[3]	result[14]	14.874	SLOW	3.714
A[3]	result[13]	14.860	SLOW	3.723
X[0]	result[14]	14.834	SLOW	3.750
X[0]	result[13]	14.821	SLOW	3.759
A[3]	result[12]	14.687	SLOW	3.653
X[0]	result[12]	14.648	SLOW	3.689
A[1]	result[10]	14.525	SLOW	3.338
A[2]	result[11]	14.521	SLOW	3.455
A[1]	result[9]	14.330	SLOW	3.420
X[2]	result[15]	14.297	SLOW	3.350
X[1]	result[11]	14.288	SLOW	3.383
X[2]	result[14]	14.244	SLOW	3.300
X[2]	result[13]	14.230	SLOW	3.309

3-)Behavioral Multiplier

1-)Verilog Code

Module Code

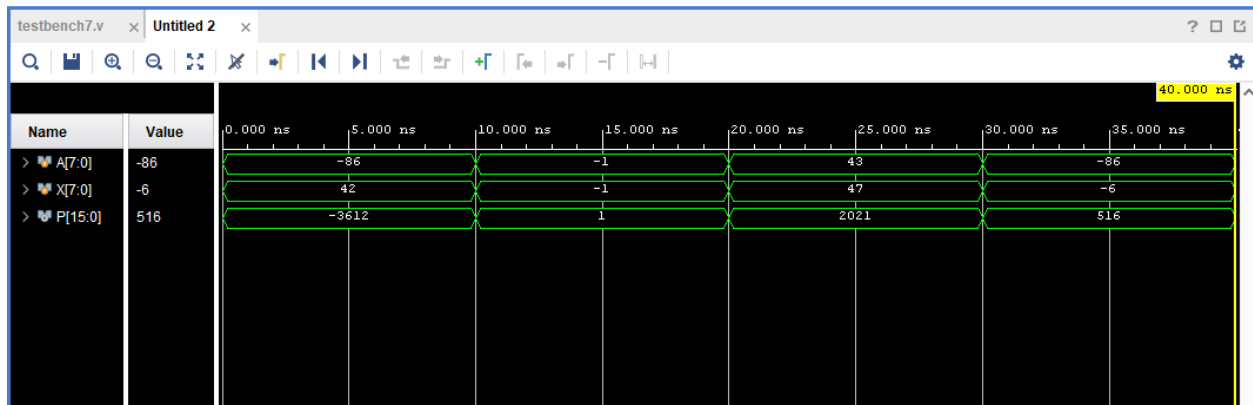
```
module MULTB(  
    input signed [7:0]A,  
    input signed [7:0]B,  
    output reg signed [15:0]result  
);  
    always @(*) begin  
        result=A*B;  
    end  
endmodule
```

Testbench Code

```
module tb;  
    reg [7:0]A;  
    reg [7:0]X;  
    wire [15:0]P;  
  
    MULTB uut(.A(A),.B(X),.result(P));  
  
    initial  
    begin  
        A=8'b10101010;  
        X=8'b00101010;  
        #10  
        A=8'b11111111;  
        X=8'b11111111;  
        #10  
        A=8'b00101011;  
        X=8'b00101111;  
        #10  
        A=8'b10101010;  
        X=8'b11111010;  
        #10  
        $finish;  
    end  
endmodule
```

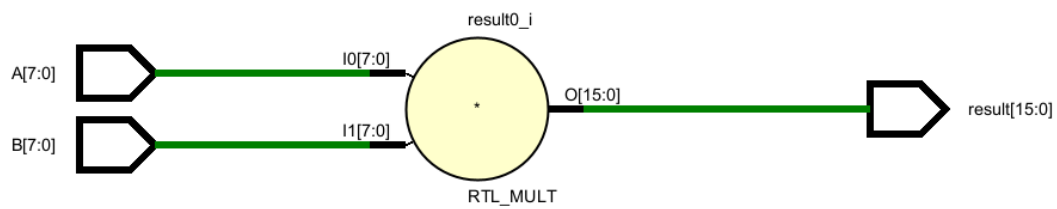
2-)Simulation Results

Wave Form

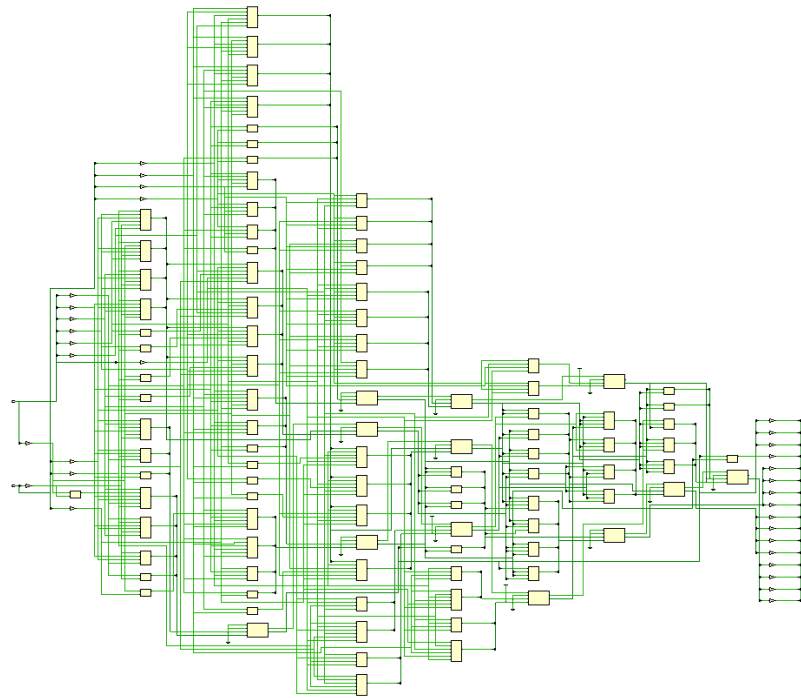


3-)RTL and Technology Schematics

RTL Schematic

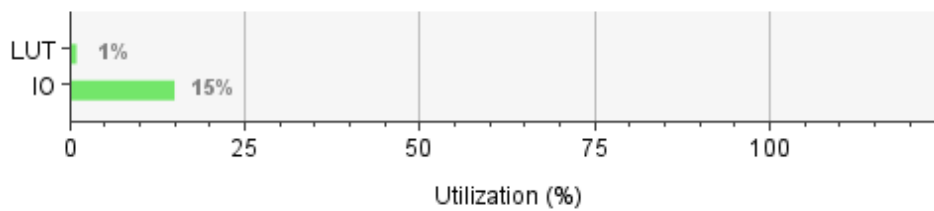


Technology Schematic






































































Utilization Report

Resource	Utilization	Available	Utilization %
LUT	61	63400	0.10
IO	32	210	15.24



Timing Report

From Port	To Port	Max Delay 	Max Process Corner	Min Delay	Min Process Corner	
 A[4]	 result[13]	12.889	SLOW	3.123	FAST	
 A[2]	 result[13]	12.869	SLOW	3.280	FAST	
 B[1]	 result[13]	12.844	SLOW	3.281	FAST	
 A[4]	 result[14]	12.796	SLOW	3.090	FAST	
 A[2]	 result[14]	12.777	SLOW	3.251	FAST	
 A[4]	 result[15]	12.771	SLOW	3.108	FAST	
 B[1]	 result[14]	12.752	SLOW	3.248	FAST	
 A[2]	 result[15]	12.751	SLOW	3.224	FAST	
 B[2]	 result[13]	12.726	SLOW	3.253	FAST	
 B[1]	 result[15]	12.726	SLOW	3.266	FAST	
 A[6]	 result[13]	12.690	SLOW	3.055	FAST	
 A[3]	 result[13]	12.658	SLOW	3.343	FAST	
 B[2]	 result[14]	12.634	SLOW	3.220	FAST	
 A[4]	 result[12]	12.628	SLOW	3.023	FAST	
 B[2]	 result[15]	12.608	SLOW	3.239	FAST	
 A[2]	 result[12]	12.608	SLOW	3.184	FAST	
 A[6]	 result[14]	12.597	SLOW	3.119	FAST	
 B[1]	 result[12]	12.583	SLOW	3.181	FAST	
 A[6]	 result[15]	12.572	SLOW	3.137	FAST	
 A[3]	 result[14]	12.565	SLOW	3.398	FAST	
 A[3]	 result[15]	12.540	SLOW	3.416	FAST	
 A[4]	 result[11]	12.509	SLOW	3.011	FAST	
 A[2]	 result[11]	12.489	SLOW	3.100	FAST	
 B[2]	 result[12]	12.465	SLOW	3.154	FAST	
 B[1]	 result[11]	12.464	SLOW	3.086	FAST	
 A[4]	 result[10]	12.456	SLOW	3.022	FAST	
 A[2]	 result[10]	12.436	SLOW	3.147	FAST	
 A[6]	 result[12]	12.429	SLOW	3.052	FAST	
 B[1]	 result[10]	12.411	SLOW	3.133	FAST	
 A[1]	 result[13]	12.400	SLOW	3.285	FAST	
 A[3]	 result[12]	12.397	SLOW	3.331	FAST	
 B[3]	 result[13]	12.389	SLOW	3.084	FAST	
 B[4]	 result[13]	12.368	SLOW	3.154	FAST	

➤ A[3]	➤ result[11]	12.356	SLOW	3.231	FAST
➤ B[2]	➤ result[11]	12.346	SLOW	3.059	FAST
➤ A[5]	➤ result[13]	12.336	SLOW	3.162	FAST
➤ A[2]	➤ result[9]	12.332	SLOW	2.989	FAST
➤ A[6]	➤ result[11]	12.310	SLOW	2.943	FAST
➤ A[1]	➤ result[14]	12.307	SLOW	3.340	FAST
➤ B[3]	➤ result[14]	12.297	SLOW	3.102	FAST
➤ B[2]	➤ result[10]	12.293	SLOW	3.106	FAST
➤ A[3]	➤ result[10]	12.292	SLOW	3.304	FAST
➤ A[1]	➤ result[15]	12.282	SLOW	3.358	FAST
➤ B[4]	➤ result[14]	12.276	SLOW	3.131	FAST
➤ B[3]	➤ result[15]	12.271	SLOW	3.120	FAST
➤ B[5]	➤ result[13]	12.262	SLOW	2.946	FAST
➤ A[6]	➤ result[10]	12.257	SLOW	3.008	FAST
➤ B[4]	➤ result[15]	12.250	SLOW	3.149	FAST
➤ A[5]	➤ result[14]	12.244	SLOW	3.214	FAST
➤ A[3]	➤ result[9]	12.230	SLOW	3.220	FAST
➤ A[5]	➤ result[15]	12.218	SLOW	3.232	FAST
➤ A[4]	➤ result[9]	12.186	SLOW	2.963	FAST
➤ B[0]	➤ result[13]	12.182	SLOW	3.209	FAST
➤ B[5]	➤ result[14]	12.170	SLOW	2.963	FAST
➤ A[2]	➤ result[8]	12.160	SLOW	3.021	FAST
➤ B[5]	➤ result[15]	12.144	SLOW	2.982	FAST
➤ A[1]	➤ result[12]	12.139	SLOW	3.273	FAST
➤ B[3]	➤ result[12]	12.128	SLOW	3.035	FAST
➤ A[0]	➤ result[13]	12.125	SLOW	3.253	FAST
➤ B[4]	➤ result[12]	12.107	SLOW	3.065	FAST
➤ B[0]	➤ result[14]	12.089	SLOW	3.177	FAST
➤ A[5]	➤ result[12]	12.075	SLOW	3.147	FAST
➤ B[0]	➤ result[15]	12.064	SLOW	3.195	FAST

4-)Multiply and Accumulate

1-)Verilog Code

Module Code

```

module MAC(
    input clk,
    input reset,
    input signed [23:0]data,weight,
    output reg signed [19:0]result
);
    initial begin
        result = 20'b00000000000000000000;
    end
    wire signed [15:0]product0,product1,product2;
    wire signed [15:0] sum0,sum1;
    reg [1:0]count=2'b00;
    MULTB u1(.A(data[7:0]),.B(weight[7:0]),.result(product0));
    MULTB u2(.A(data[15:8]),.B(weight[15:8]),.result(product1));
    MULTB u3(.A(data[23:16]),.B(weight[23:16]),.result(product2));
    BA u4(.A(product0),.B(product1),.result(sum0));
    BA u5(.A(product2),.B(sum0),.result(sum1));
    always @(posedge clk )begin
        if (reset == 1'b1)begin
            result <= 20'b00000000000000000000;
            count <= 2'b00;
        end
        else if(count == 2'b00)begin
            result <= sum1;
            count <= count + 1'b1;
        end
        else if(count == 2'b10)begin
            count <=2'b00;
            result <= result + sum1;
        end
        else begin
            result <= result + sum1;
            count <= count +1'b1;
        end
    end
end

endmodule

```


Testbench Code

```
module testbench8;
    reg [23:0]data;
    reg [23:0]weight;
    reg clk;
    reg reset;
    wire [19:0]result;

    MAC uut(.clk(clk),.reset(reset),.data(data),.weight(weight),.result(result));

    initial
    begin
        reset=1'b0;
        clk=1'b0;
        data=24'b0000000000000000000000100;
        weight=24'b111111111111111111111111;
        #10
        clk=1'b1;
        #10
        clk=1'b0;
        #10
        clk=1'b1;
        #10
        clk=1'b0;
        #10
        clk=1'b1;
        #10
        data=24'b0000000000000000000000100;
        clk=1'b0;
        #10
        clk=1'b1;
        #10
        clk=1'b0;
        #10
        clk=1'b1;
        #10
        clk=1'b0;
        #10
        clk=1'b1;
        #10
        data=24'b000000000000000000000010;
        clk=1'b0;
```

```
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
data=24'b000000000000000000000000;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
data=24'b000000000000000000000011;
weight=24'b111111111111111111111000;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
data=24'b000000000000000000000001;
weight=24'b111111111111111111111111;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
```

```
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
data=24'b000000000000000000000000;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
data=24'b0000000000000000000000100;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
data=24'b000000000000000000000000;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
```

```

#10
data=24'b000000100000000000000000;
clk=1'b0;
#10
clk=1'b1;
#10
clk=1'b0;
#10
clk=1'b1;
reset=1'b1;
#10

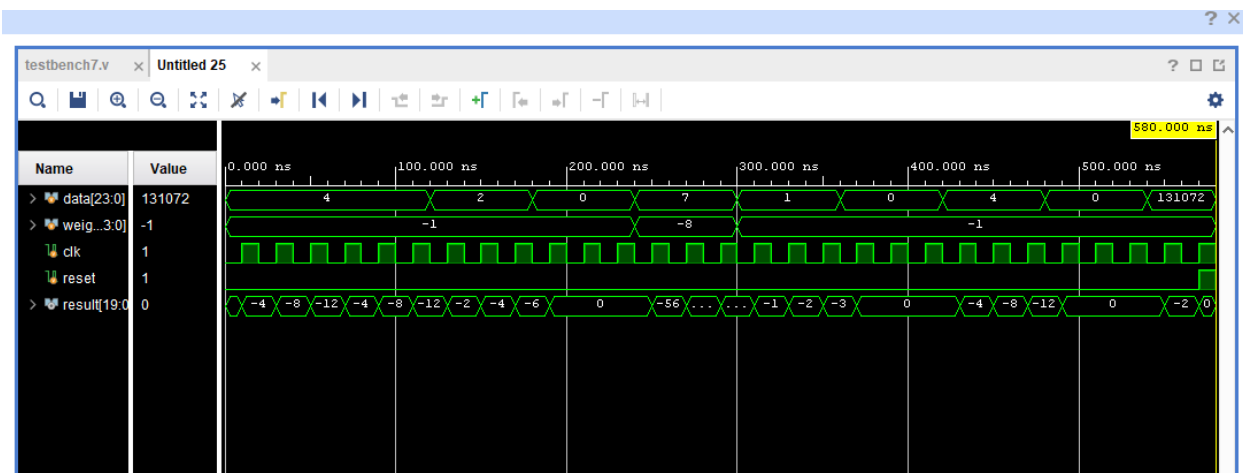
$finish;
end

endmodule

```

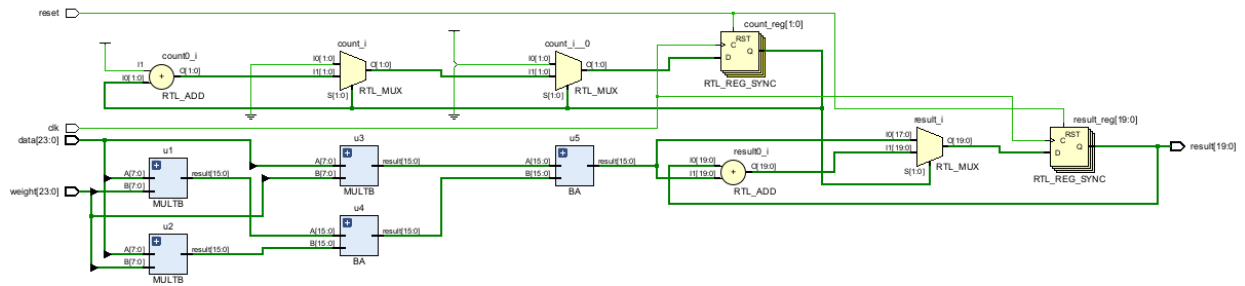
2-)Simulation Results

Wave Form

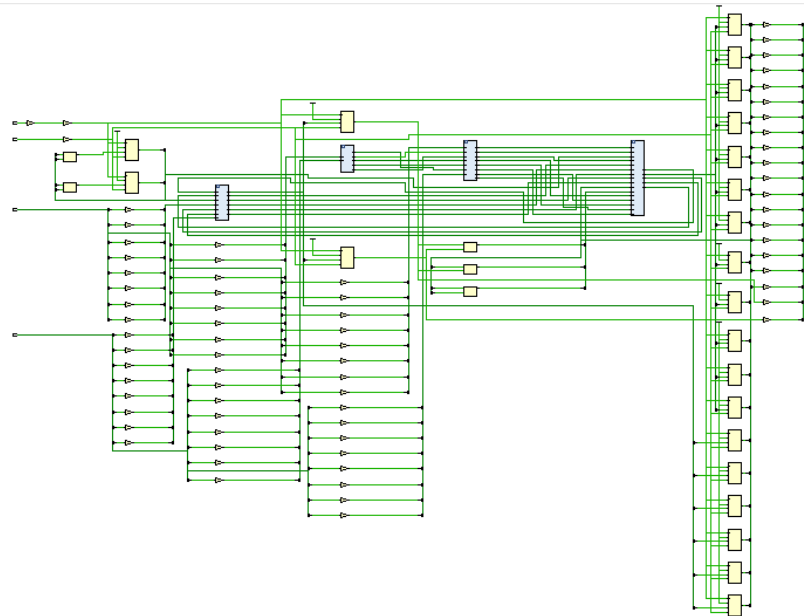


3-)RTL and Technology Schematics

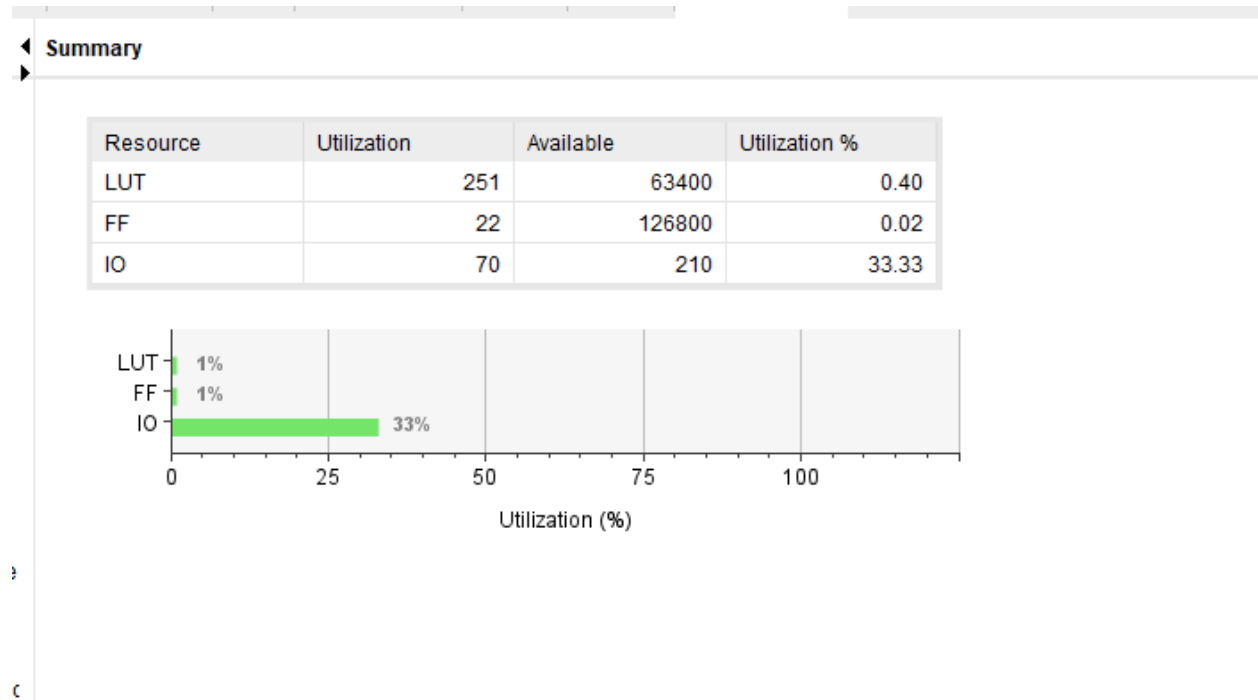
RTL Schematic



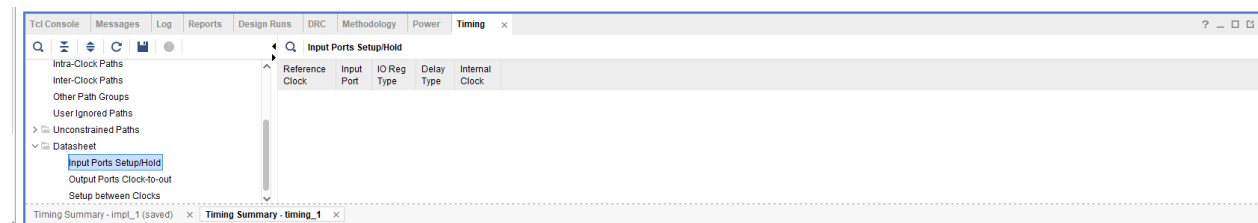
Technology Schematic



Utilization Report



Timing Report



Vivado did not give me the combinational delay results.

That's why also I could not calculate the maximum achievable frequency.

4-) 2D Convolution

2D convolution takes a kernel then reverses it, after that it pass through the reverse kernel in the matrix of image or something else.

Sliding The Kernel

First we reverse the kernel then we put it in left top side of the matrix the we multiply every corresponding square then we add them this is first result then we slide the kernel o square right and same procedure, when there is no more right square we go to left most square but this time it is one cell down.

Verilog Code

Module Code

```
module D2C(
    input [199:0]f,
    input [71:0]w,
    input clk,
    output
    [19:0]result_33,result_32,result_31,result_23,result_22,result_21,result_13,result_12,result_11
);
    wire [179:0]resultp;
    wire rst=1'b0;
    wire [19:0]
    result99,result1,result2,result3,result4,result5,result6,result7,result8,result9,
    result10,result11,result12,result13,result14,result15,result16,result17,result18,
    result19,result20,result21,result22,result23,result24,result25,result26;

    MAC
    mac1(.clk(clk),.reset(rst),.data({f[7:0],f[15:8],f[23:16]}),.weight({w[71:64],w[63:56],w[55:48]}),.result(result99));
```

```

    MAC
mac2(.clk(clk),.reset(rst),.data({f[47:40],f[55:48],f[63:56]}),.weight({w[47:40],
w[39:32],w[31:24]}),.result(result1));

    MAC
maci(.clk(clk),.reset(rst),.data({f[87:80],f[95:88],f[103:96]}),.weight({w[23:16]
,w[15:8],w[7:0]}),.result(result2));

    assign resultp[19:0]= result99+result1+result2;
    assign result_11 = resultp[19:0];

    MAC
mac3(.clk(clk),.reset(rst),.data({f[15:8],f[23:16],f[31:24]}),.weight({w[71:64],w
[63:56],w[55:48]}),.result(result3));

    MAC
mac4(.clk(clk),.reset(rst),.data({f[55:48],f[63:56],f[71:64]}),.weight({w[47:40],
w[39:32],w[31:24]}),.result(result4));

    MAC
mac5(.clk(clk),.reset(rst),.data({f[95:88],f[103:96],f[111:104]}),.weight({w[23:1
6],w[15:8],w[7:0]}),.result(result5));

    assign resultp[39:20]= result3+result4+result5;
    assign result_12=resultp[39:20];

    MAC
mac6(.clk(clk),.reset(rst),.data({f[23:16],f[31:24],f[39:32]}),.weight({w[71:64],
w[63:56],w[55:48]}),.result(result6));

    MAC
mac7(.clk(clk),.reset(rst),.data({f[63:56],f[71:64],f[79:72]}),.weight({w[47:40],
w[39:32],w[31:24]}),.result(result7));

    MAC
mac8(.clk(clk),.reset(rst),.data({f[103:96],f[111:104],f[119:112]}),.weight({w[23
:16],w[15:8],w[7:0]}),.result(result8));

    assign resultp[59:40]= result8+result6+result7;
    assign result_13 = resultp[59:40];

    MAC
mac9(.clk(clk),.reset(rst),.data({f[47:40],f[55:48],f[63:56]}),.weight({w[71:64],
w[63:56],w[55:48]}),.result(result9));

    MAC
mac10(.clk(clk),.reset(rst),.data({f[87:80],f[95:88],f[103:96]}),.weight({w[47:40
],w[39:32],w[31:24]}),.result(result10));

    MAC
mac11(.clk(clk),.reset(rst),.data({f[127:120],f[135:128],f[143:136]}),.weight({w[
23:16],w[15:8],w[7:0]}),.result(result11));

```



```

    assign resultp[79:60]= result9+result10+result11;
    assign result_21 = resultp[79:60];

    MAC
mac12(.clk(clk),.reset(rst),.data({f[55:48],f[63:56],f[71:64]}),.weight({w[71:64]
,w[63:56],w[55:48]}),.result(result12));
    MAC
mac13(.clk(clk),.reset(rst),.data({f[95:88],f[103:96],f[111:104]}),.weight({w[47:
40],w[39:32],w[31:24]}),.result(result13));
    MAC
mac14(.clk(clk),.reset(rst),.data({f[135:128],f[143:136],f[151:144]}),.weight({w[
23:16],w[15:8],w[7:0]}),.result(result14));

    assign resultp[99:80]= result12+result13+result14;
    assign result_22 = resultp[99:80];

    MAC
mac15(.clk(clk),.reset(rst),.data({f[63:56],f[71:64],f[79:72]}),.weight({w[71:64]
,w[63:56],w[55:48]}),.result(result15));
    MAC
mac26(.clk(clk),.reset(rst),.data({f[103:96],f[111:104],f[119:112]}),.weight({w[4
7:40],w[39:32],w[31:24]}),.result(result16));
    MAC
maci7(.clk(clk),.reset(rst),.data({f[143:136],f[151:144],f[159:152]}),.weight({w[
23:16],w[15:8],w[7:0]}),.result(result17));

    assign resultp[119:100]= result15+result16+result17;
    assign result_23 = resultp[119:100];

    MAC
mac111(.clk(clk),.reset(rst),.data({f[87:80],f[95:88],f[103:96]}),.weight({w[71:6
4],w[63:56],w[55:48]}),.result(result18));
    MAC
mac2a(.clk(clk),.reset(rst),.data({f[127:120],f[135:128],f[143:136]}),.weight({w[
47:40],w[39:32],w[31:24]}),.result(result19));
    MAC
maciv(.clk(clk),.reset(rst),.data({f[167:160],f[175:168],f[183:176]}),.weight({w[
23:16],w[15:8],w[7:0]}),.result(result20));

    assign resultp[139:120]= result18+result19+result20;
    assign result_31 = resultp[139:120];

```

```

    MAC
mac1f(.clk(clk),.reset(rst),.data({f[95:88],f[103:96],f[111:104]}),.weight({w[71:64],w[63:56],w[55:48]}),.result(result21));
    MAC
mac2e(.clk(clk),.reset(rst),.data({f[135:128],f[143:136],f[151:144]}),.weight({w[47:40],w[39:32],w[31:24]}),.result(result22));
    MAC
mactt(.clk(clk),.reset(rst),.data({f[175:168],f[183:176],f[191:184]}),.weight({w[23:16],w[15:8],w[7:0]}),.result(result23));

    assign resultp[159:140]= result21+result22+result23;
    assign result_32 = resultp[159:140];

    MAC
mac1a(.clk(clk),.reset(rst),.data({f[103:96],f[111:104],f[119:112]}),.weight({w[71:64],w[63:56],w[55:48]}),.result(result24));
    MAC
mac2g(.clk(clk),.reset(rst),.data({f[143:136],f[151:144],f[159:152]}),.weight({w[47:40],w[39:32],w[31:24]}),.result(result25));
    MAC
macff(.clk(clk),.reset(rst),.data({f[183:176],f[191:184],f[199:192]}),.weight({w[23:16],w[15:8],w[7:0]}),.result(result26));

    assign resultp[179:160]= result24+result25+result26;
    assign result_33 = resultp[179:160];

endmodule

```

Testbench Code

```

module tbcon;
    reg signed [199:0]f;
    reg signed [71:0]w;
    reg clk;
    wire
[19:0]result_33,result_32,result_31,result_23,result_22,result_21,result_13,result_12,result_11;

    D2C
uut(.f(f),.w(w),.clk(clk),.result_33(result_33),.result_32(result_32),.result_31(result_31),.result_23(result_23),.result_22(result_22),.result_21(result_21),.result_13(result_13),.result_12(result_12),.result_11(result_11));

```

```

initial
begin
    f={8'd128,8'd128,8'd128,8'd128,8'd128,8'd255,8'd255,8'd128,8'd255,8'd255,
8'd255,8'd255,8'd128,8'd255,8'd255,8'd255,8'd255,8'd128,8'd255,8'd255,8'd255,8'd2
55,8'd128,8'd255,8'd255};
    w={-8'd1,-8'd1,-8'd1,-8'd1,8'd8,-8'd1,-8'd1,-8'd1,-8'd1};
    clk=1'b0;
    #10
    clk=1'b1;
    #10
    clk=1'b0;
    #10
    $finish;
end

```

Simulation

Wave Form

