DIGITAL SYSTEM DESIGN APPLICATION

HOMEWORK 7

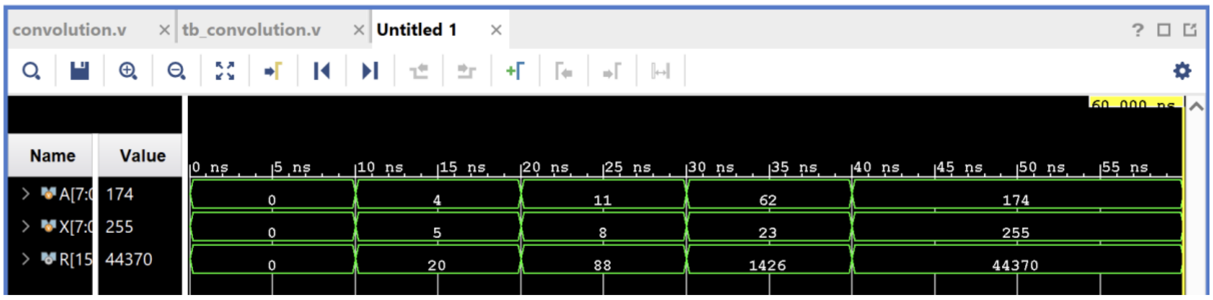
STRUCTURAL MULTIPLIER-UNSIGNED

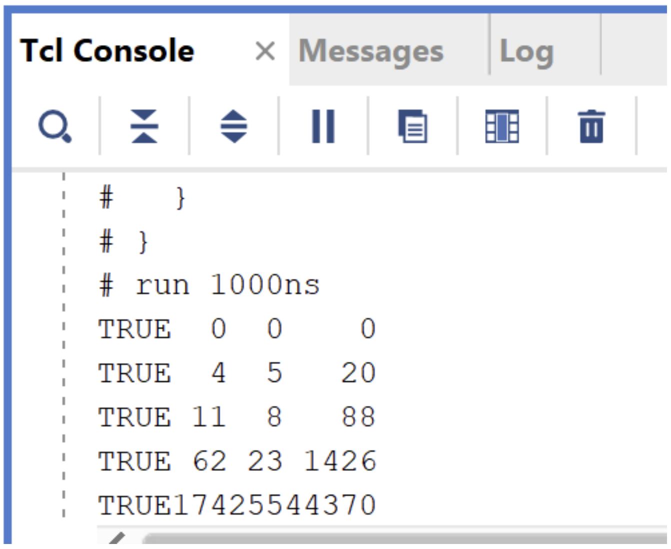
VERILOG CODE AND TESTBENCH

```
module MULTS
(
   input [7:0]A,
   input [7:0]X,
   output [15:0]R
   );
   wire [15:0] PP[0:7];

   genvar i;
   generate
      for (i=0; i< 8; i=i+1)begin
         assign PP[i] = ({15{X[i]}}&A)<<i;
      end
   endgenerate

  wire [15:0]sum1,sum2,sum3,sum4,sum5,sum6;
  wire [6:0]carry=0;
  cla16bit cla1(PP[0],PP[1],0, sum1,carry[0]);
  cla16bit cla2(PP[2],PP[3], 0, sum2,carry[1]);
  cla16bit cla3(PP[4],PP[5], 0, sum3,carry[2]);
  cla16bit cla4(PP[6],PP[7], 0, sum4,carry[3]);
```

```
module tb_MULTS(

    );
    reg [7:0]A,X;
    wire [15:0]R;

    MULTS uut(.A(A),.X(X),.R(R));

    initial begin

        A=8'b0000_0000; X=8'b0000_0000; #10;
        if(R==16'd0)
            $display("TRUE",A,X,R);
            else
            $display("FALSE",A,X,R);
        A=8'd4; X=8'd5; #10;
        if(R==16'd5)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd11; X=8'd8; #10;
        if(R==16'd88)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);

        A=8'd62; X=8'd23; #10;
        if(R==16'd1426)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd174; X=8'd255; #10;
        if(R==16'd44370)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        #10;
        $finish;
    end
endmodule
```

SIMULATION RESULT (WAVEFORM AND TCL CONSOLE)





```
#    }
# }
# run 1000ns
TRUE  0  0    0
TRUE  4  5   20
TRUE 11  8   88
TRUE 62 23 1426
TRUE17425544370
```

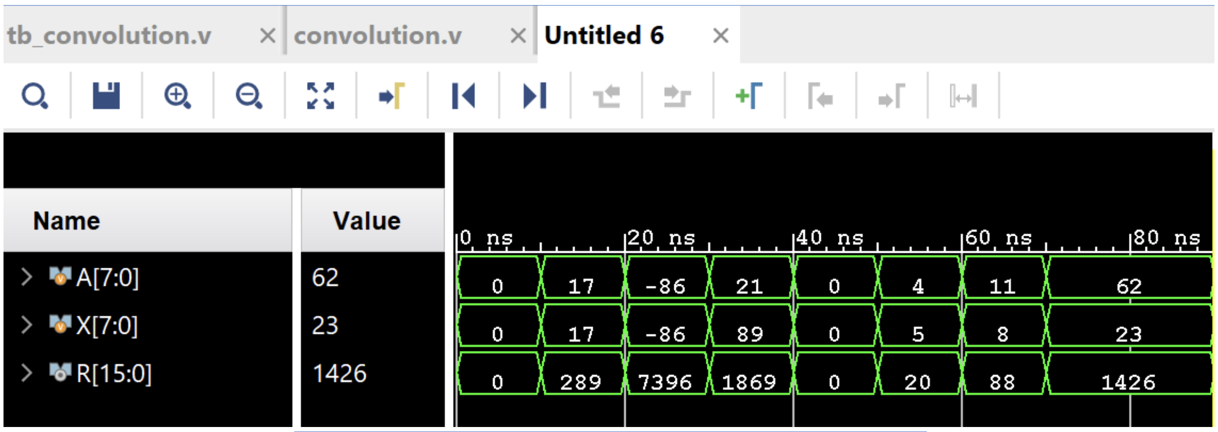
STRUCTURAL MULTIPLIER-SIGNED

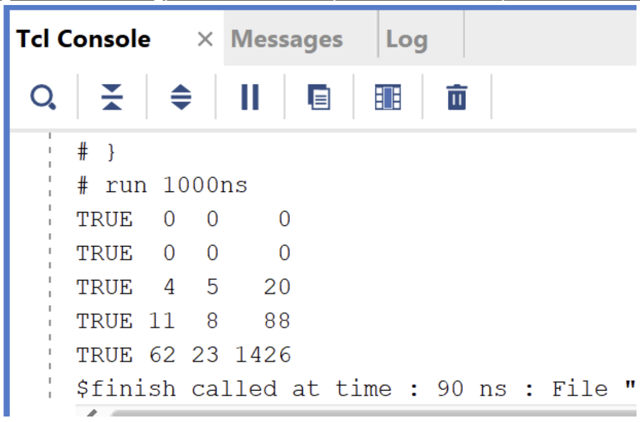
VERILOG CODE AND TESTBENCH

```
module MULTS_signed (
   input [7:0] A,X,
   output [15:0] R);
   wire [7:0] PP[7:0];
   wire [15:0] PP_shifted[7:0];
   wire [15:0]sum[6:0];

   genvar i;
   generate
      for (i=0; i< 7; i=i+1)begin
         assign PP[i][6:0]= (X[i]*A[6:0]);
         assign PP[i][7] = ~(X[i] * A[7]);
      end
      assign PP[7][6:0] = ~(X[7] * A[6:0]);
      assign PP[7][7] = X[7]*A[7];
   endgenerate

   generate
      for(i=0; i<=7; i=i+1)
      begin
      if(i==0)   begin
      assign PP_shifted[i][7:0] = PP[i][7:0];
      assign PP_shifted[i][8]=1'b1;
      assign PP_shifted[i][15:9] = 7'd0;
         end
      else
      assign PP_shifted[i] = PP[i] << i;
      end
   endgenerate
   wire cout[6:0];
   genvar j;
   generate
      for(j=0; j<4; j = j+1)
      begin
      cla16bit cla0(.a(PP_shifted[j]), .b(PP_shifted[j+4]), .cin(1'b0), .sum(sum[j]), .cout(cout[j]));
      end
   endgenerate
```

```
module tb_MULTB();
    reg [7:0]A,X;
    wire [15:0] R;
  // MULTB uut(.A(A),.X(X),.R(R));
    MULTS_signed uut(.A(A),.X(X),.R(R));
    initial begin
        A=8'b0000_0000; X=8'b0000_0000; #10;
        if(R==16'b0000_0000)
            $display("TRUE",A,X,R);
            else
            $display("FALSE",A,X,R);
        A=8'b0001_0001; X=8'b0001_0001; #10;
        A=8'b1010_1010; X=8'b1010_1010; #10;
        A=8'b0001_0101; X=8'b0101_1001; #10;
        A=8'b0000_0000; X=8'b0000_0000; #10;
        if(R==16'd0)
            $display("TRUE",A,X,R);
            else
            $display("FALSE",A,X,R);
        A=8'd4; X=8'd5; #10;
        if(R==16'd20)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd11; X=8'd8; #10;
        if(R==16'd88)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd62; X=8'd23; #10;
```

SIMULATION RESULT (WAVEFORM AND TCL CONSOLE)



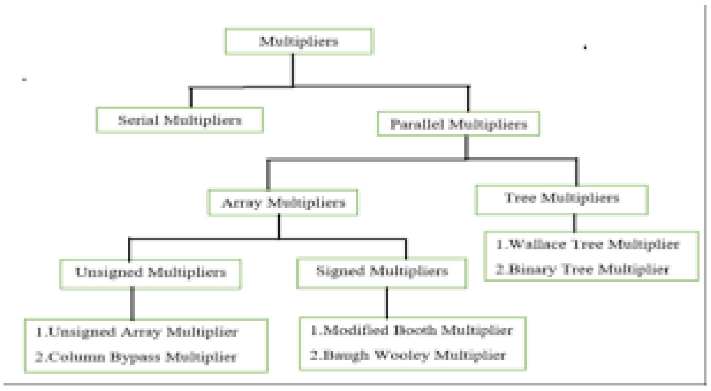


```
# }
# run 1000ns
TRUE   0   0     0
TRUE   0   0     0
TRUE   4   5    20
TRUE  11   8    88
TRUE  62  23  1426
$finish called at time : 90 ns : File "
```

HOW MANY NUMBER OF ADDER STAGES ARE NECESSARY

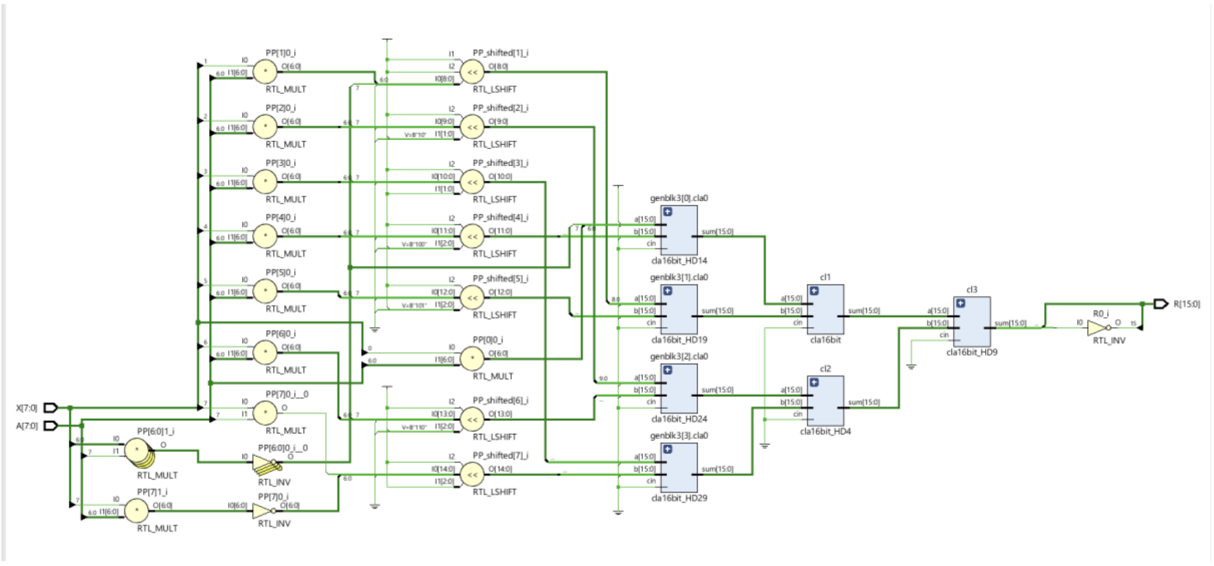
We have two stages and this is enough.

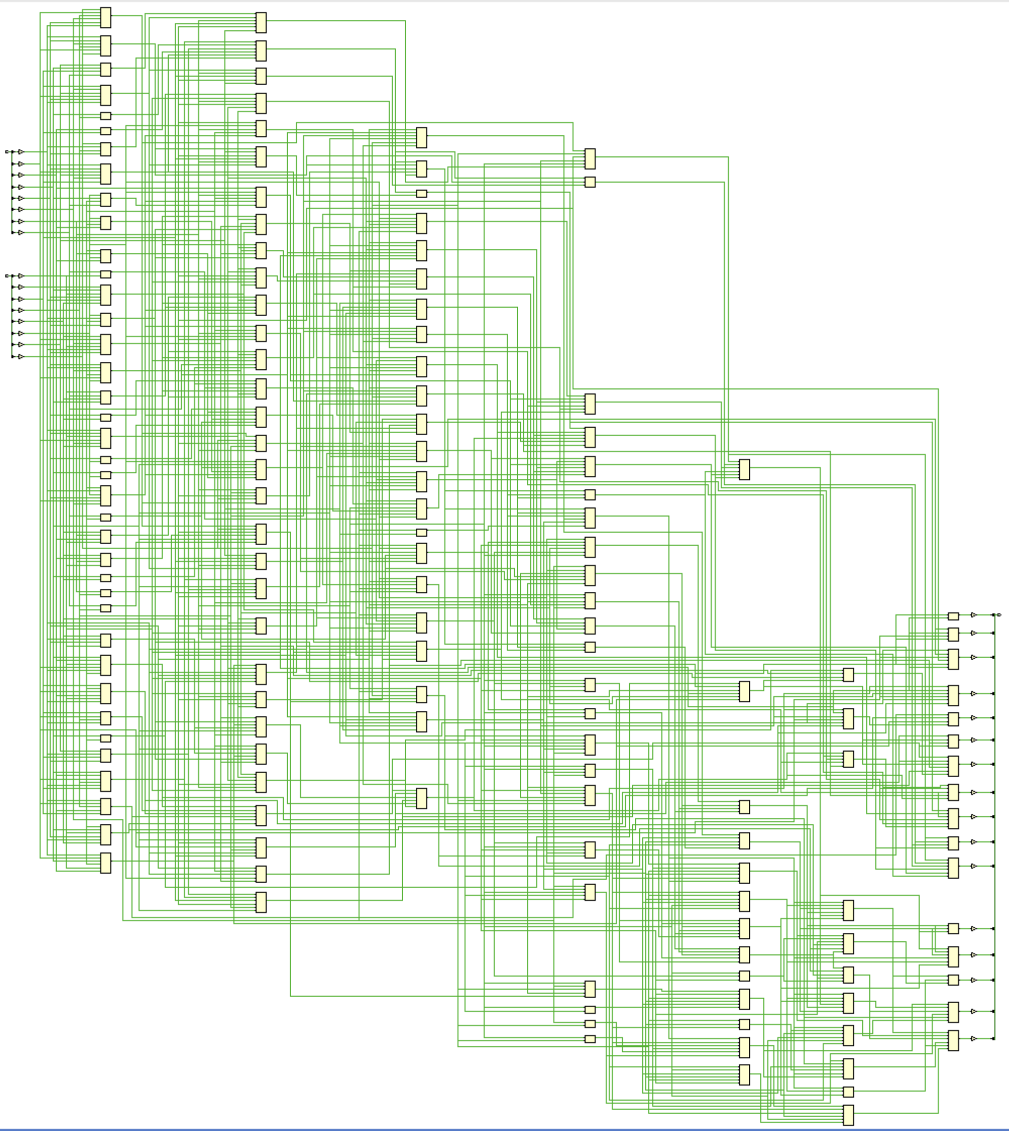
BAUGH-WOOLEY METHOD



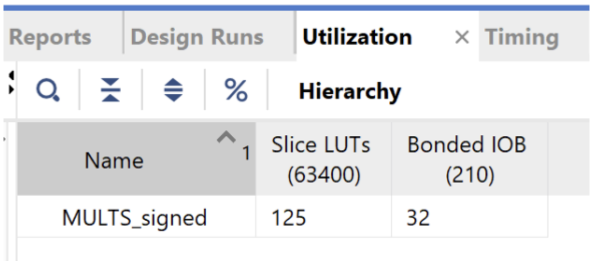
Application for 2s complement multiplication. The Baugh-Wooley multiplier uses a technique called carry-save addition to improve the speed of multiplication by reducing the number of full-adder operations that are needed. This makes it faster than other multiplication algorithms, such as the traditional long multiplication method. It is commonly used in high-performance computing applications, such as graphics processing and scientific computing.

RTL AND TECHNOLOGY SCHEAMTIC

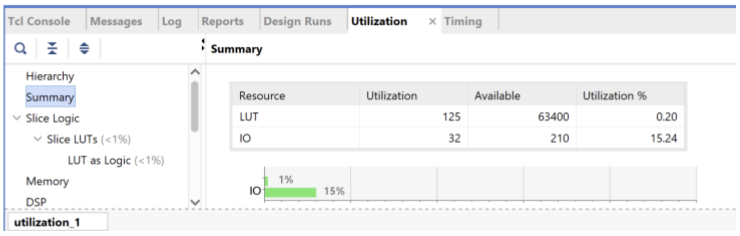




UTILIZATION RAPORT

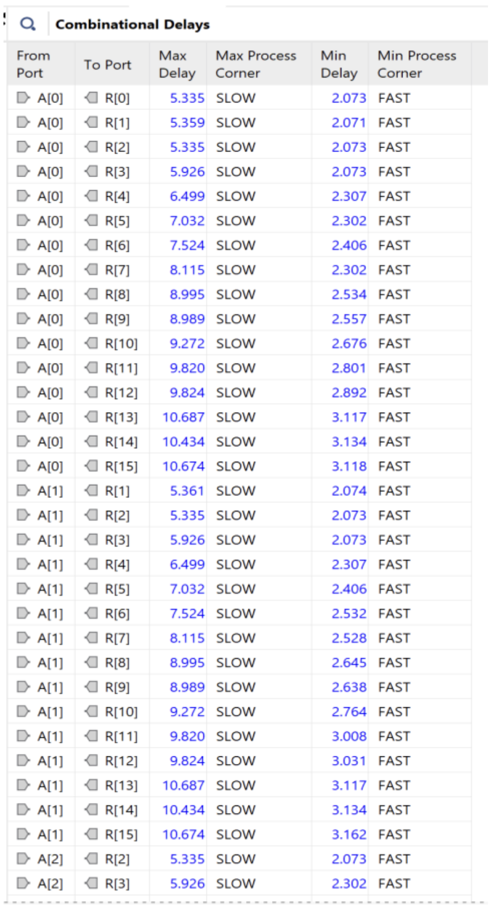


Name	Slice LUTs (63400)	Bonded IOB (210)
MULTS_signed	125	32

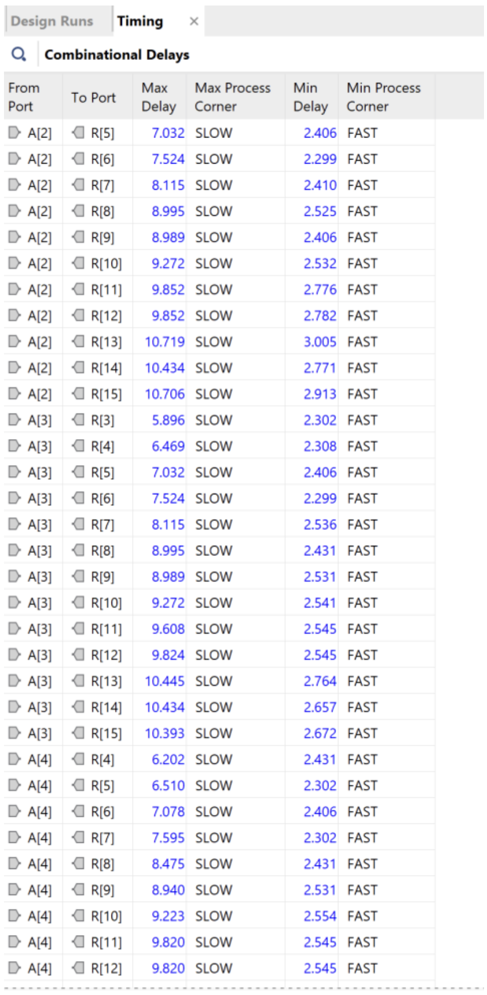


Resource	Utilization	Available	Utilization %
LUT	125	63400	0.20
IO	32	210	15.24

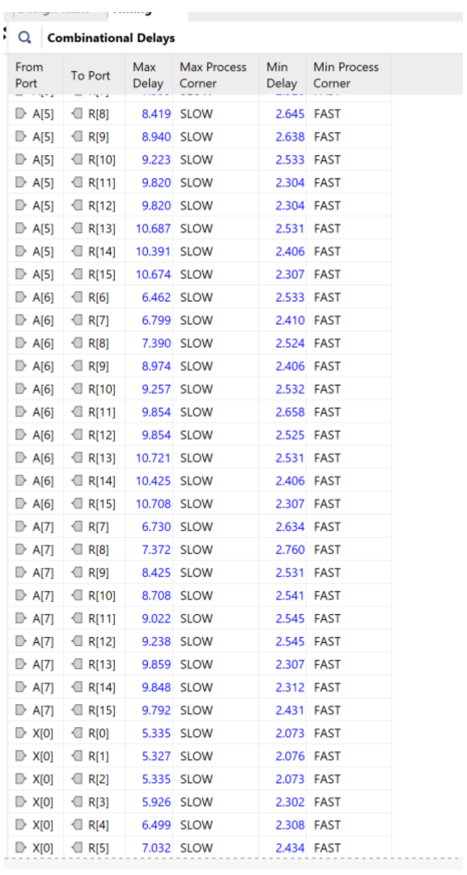
TIMING REPORT



From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	R[0]	5.335	SLOW	2.073	FAST
A[0]	R[1]	5.359	SLOW	2.071	FAST
A[0]	R[2]	5.335	SLOW	2.073	FAST
A[0]	R[3]	5.926	SLOW	2.073	FAST
A[0]	R[4]	6.499	SLOW	2.307	FAST
A[0]	R[5]	7.032	SLOW	2.302	FAST
A[0]	R[6]	7.524	SLOW	2.406	FAST
A[0]	R[7]	8.115	SLOW	2.302	FAST
A[0]	R[8]	8.995	SLOW	2.534	FAST
A[0]	R[9]	8.989	SLOW	2.557	FAST
A[0]	R[10]	9.272	SLOW	2.676	FAST
A[0]	R[11]	9.820	SLOW	2.801	FAST
A[0]	R[12]	9.824	SLOW	2.892	FAST
A[0]	R[13]	10.687	SLOW	3.117	FAST
A[0]	R[14]	10.434	SLOW	3.134	FAST
A[0]	R[15]	10.674	SLOW	3.118	FAST
A[1]	R[1]	5.361	SLOW	2.074	FAST
A[1]	R[2]	5.335	SLOW	2.073	FAST
A[1]	R[3]	5.926	SLOW	2.073	FAST
A[1]	R[4]	6.499	SLOW	2.307	FAST
A[1]	R[5]	7.032	SLOW	2.406	FAST
A[1]	R[6]	7.524	SLOW	2.532	FAST
A[1]	R[7]	8.115	SLOW	2.528	FAST
A[1]	R[8]	8.995	SLOW	2.645	FAST
A[1]	R[9]	8.989	SLOW	2.638	FAST
A[1]	R[10]	9.272	SLOW	2.764	FAST
A[1]	R[11]	9.820	SLOW	3.008	FAST
A[1]	R[12]	9.824	SLOW	3.031	FAST
A[1]	R[13]	10.687	SLOW	3.117	FAST
A[1]	R[14]	10.434	SLOW	3.134	FAST
A[1]	R[15]	10.674	SLOW	3.162	FAST
A[2]	R[2]	5.335	SLOW	2.073	FAST
A[2]	R[3]	5.926	SLOW	2.302	FAST

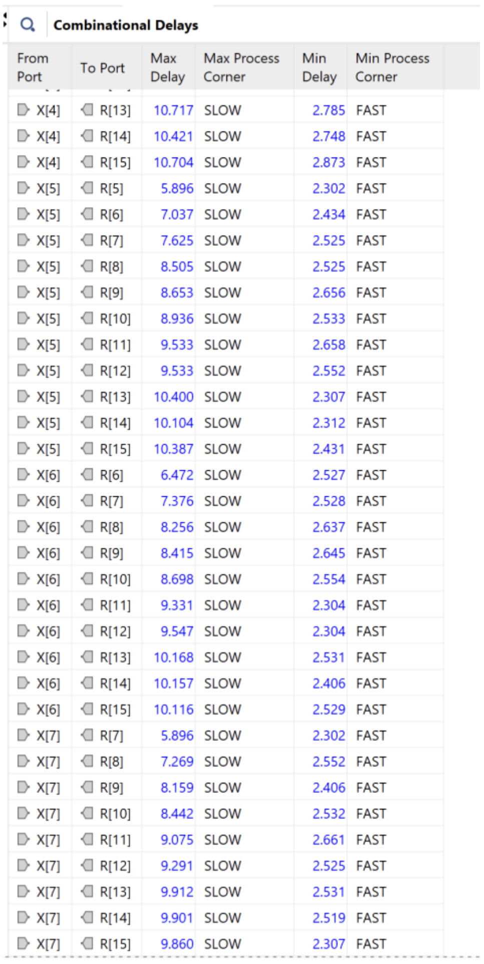


From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[2]	R[5]	7.032	SLOW	2.406	FAST
A[2]	R[6]	7.524	SLOW	2.299	FAST
A[2]	R[7]	8.115	SLOW	2.410	FAST
A[2]	R[8]	8.995	SLOW	2.525	FAST
A[2]	R[9]	8.989	SLOW	2.406	FAST
A[2]	R[10]	9.272	SLOW	2.532	FAST
A[2]	R[11]	9.852	SLOW	2.776	FAST
A[2]	R[12]	9.852	SLOW	2.782	FAST
A[2]	R[13]	10.719	SLOW	3.005	FAST
A[2]	R[14]	10.434	SLOW	2.771	FAST
A[2]	R[15]	10.706	SLOW	2.913	FAST
A[3]	R[3]	5.896	SLOW	2.302	FAST
A[3]	R[4]	6.469	SLOW	2.308	FAST
A[3]	R[5]	7.032	SLOW	2.406	FAST
A[3]	R[6]	7.524	SLOW	2.299	FAST
A[3]	R[7]	8.115	SLOW	2.536	FAST
A[3]	R[8]	8.995	SLOW	2.431	FAST
A[3]	R[9]	8.989	SLOW	2.531	FAST
A[3]	R[10]	9.272	SLOW	2.541	FAST
A[3]	R[11]	9.608	SLOW	2.545	FAST
A[3]	R[12]	9.824	SLOW	2.545	FAST
A[3]	R[13]	10.445	SLOW	2.764	FAST
A[3]	R[14]	10.434	SLOW	2.657	FAST
A[3]	R[15]	10.393	SLOW	2.672	FAST
A[4]	R[4]	6.202	SLOW	2.431	FAST
A[4]	R[5]	6.510	SLOW	2.302	FAST
A[4]	R[6]	7.078	SLOW	2.406	FAST
A[4]	R[7]	7.595	SLOW	2.302	FAST
A[4]	R[8]	8.475	SLOW	2.431	FAST
A[4]	R[9]	8.940	SLOW	2.531	FAST
A[4]	R[10]	9.223	SLOW	2.554	FAST
A[4]	R[11]	9.820	SLOW	2.545	FAST
A[4]	R[12]	9.820	SLOW	2.545	FAST



Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[5]	R[8]	8.419	SLOW	2.645	FAST
A[5]	R[9]	8.940	SLOW	2.638	FAST
A[5]	R[10]	9.223	SLOW	2.533	FAST
A[5]	R[11]	9.820	SLOW	2.304	FAST
A[5]	R[12]	9.820	SLOW	2.304	FAST
A[5]	R[13]	10.687	SLOW	2.531	FAST
A[5]	R[14]	10.391	SLOW	2.406	FAST
A[5]	R[15]	10.674	SLOW	2.307	FAST
A[6]	R[6]	6.462	SLOW	2.533	FAST
A[6]	R[7]	6.799	SLOW	2.410	FAST
A[6]	R[8]	7.390	SLOW	2.524	FAST
A[6]	R[9]	8.974	SLOW	2.406	FAST
A[6]	R[10]	9.257	SLOW	2.532	FAST
A[6]	R[11]	9.854	SLOW	2.658	FAST
A[6]	R[12]	9.854	SLOW	2.525	FAST
A[6]	R[13]	10.721	SLOW	2.531	FAST
A[6]	R[14]	10.425	SLOW	2.406	FAST
A[6]	R[15]	10.708	SLOW	2.307	FAST
A[7]	R[7]	6.730	SLOW	2.634	FAST
A[7]	R[8]	7.372	SLOW	2.760	FAST
A[7]	R[9]	8.425	SLOW	2.531	FAST
A[7]	R[10]	8.708	SLOW	2.541	FAST
A[7]	R[11]	9.022	SLOW	2.545	FAST
A[7]	R[12]	9.238	SLOW	2.545	FAST
A[7]	R[13]	9.859	SLOW	2.307	FAST
A[7]	R[14]	9.848	SLOW	2.312	FAST
A[7]	R[15]	9.792	SLOW	2.431	FAST
X[0]	R[0]	5.335	SLOW	2.073	FAST
X[0]	R[1]	5.327	SLOW	2.076	FAST
X[0]	R[2]	5.335	SLOW	2.073	FAST
X[0]	R[3]	5.926	SLOW	2.302	FAST
X[0]	R[4]	6.499	SLOW	2.308	FAST
X[0]	R[5]	7.032	SLOW	2.434	FAST



Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
X[4]	R[13]	10.717	SLOW	2.785	FAST
X[4]	R[14]	10.421	SLOW	2.748	FAST
X[4]	R[15]	10.704	SLOW	2.873	FAST
X[5]	R[5]	5.896	SLOW	2.302	FAST
X[5]	R[6]	7.037	SLOW	2.434	FAST
X[5]	R[7]	7.625	SLOW	2.525	FAST
X[5]	R[8]	8.505	SLOW	2.525	FAST
X[5]	R[9]	8.653	SLOW	2.656	FAST
X[5]	R[10]	8.936	SLOW	2.533	FAST
X[5]	R[11]	9.533	SLOW	2.658	FAST
X[5]	R[12]	9.533	SLOW	2.552	FAST
X[5]	R[13]	10.400	SLOW	2.307	FAST
X[5]	R[14]	10.104	SLOW	2.312	FAST
X[5]	R[15]	10.387	SLOW	2.431	FAST
X[6]	R[6]	6.472	SLOW	2.527	FAST
X[6]	R[7]	7.376	SLOW	2.528	FAST
X[6]	R[8]	8.256	SLOW	2.637	FAST
X[6]	R[9]	8.415	SLOW	2.645	FAST
X[6]	R[10]	8.698	SLOW	2.554	FAST
X[6]	R[11]	9.331	SLOW	2.304	FAST
X[6]	R[12]	9.547	SLOW	2.304	FAST
X[6]	R[13]	10.168	SLOW	2.531	FAST
X[6]	R[14]	10.157	SLOW	2.406	FAST
X[6]	R[15]	10.116	SLOW	2.529	FAST
X[7]	R[7]	5.896	SLOW	2.302	FAST
X[7]	R[8]	7.269	SLOW	2.552	FAST
X[7]	R[9]	8.159	SLOW	2.406	FAST
X[7]	R[10]	8.442	SLOW	2.532	FAST
X[7]	R[11]	9.075	SLOW	2.661	FAST
X[7]	R[12]	9.291	SLOW	2.525	FAST
X[7]	R[13]	9.912	SLOW	2.531	FAST
X[7]	R[14]	9.901	SLOW	2.519	FAST
X[7]	R[15]	9.860	SLOW	2.307	FAST

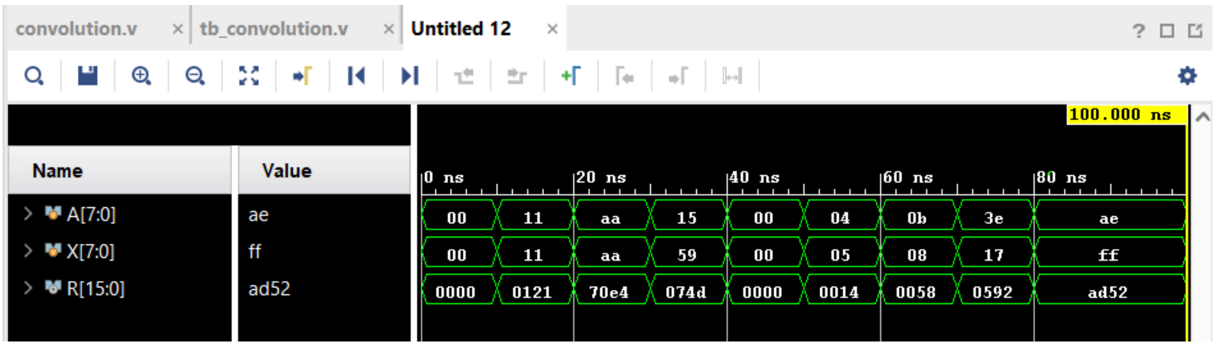
BEHAVIORAL MULTIPLIER

VERILOG CODE AND TESTBENCH

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

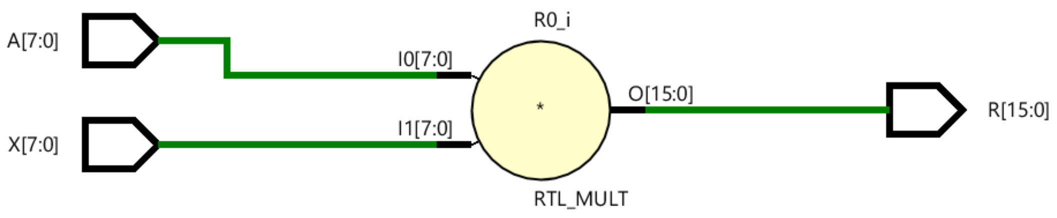
```
module tb_MULTB();
    reg[7:0]A,X;
    wire [15:0] R;
    MULTB uut(.A(A),.X(X),.R(R));
    initial begin
        A=8'b0000_0000; X=8'b0000_0000; #10;
        if(R==16'b0000_0000)
            $display("TRUE",A,X,R);
            else
            $display("FALSE",A,X,R);
        A=8'b0000_0000; X=8'b0000_0000; #10;
        if(R==16'd0)
            $display("TRUE",A,X,R);
            else
            $display("FALSE",A,X,R);
        A=8'd4; X=8'd5; #10;
        if(R==16'd20)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd11; X=8'd8; #10;
        if(R==16'd88)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd62; X=8'd23; #10;
        if(R==16'd1426)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        A=8'd174; X=8'd255; #10;
        if(R==16'd44370)
            $display("TRUE",A,X,R);
        else
            $display("FALSE",A,X,R);
        #10;
        $finish;
    end
endmodule
```

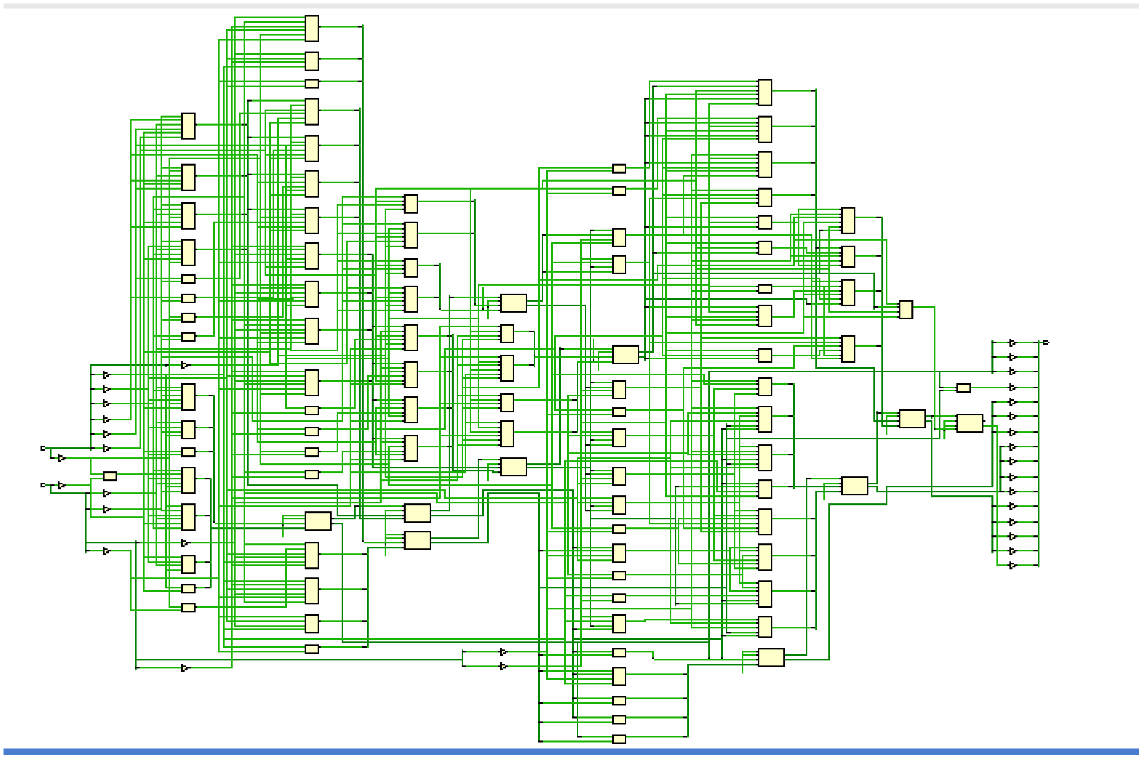
SIMULATION RESULT (WAVEFORM AND TCL CONSOLE)



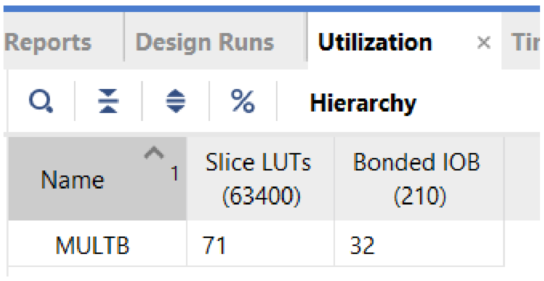
```
# run 1000ns
TRUE   0  0     0
TRUE   0  0     0
TRUE   4  5    20
TRUE  11  8    88
TRUE  62 23  1426
TRUE17425544370
$finish called at time : 100 ns :
```

RTL AND TECHNOLOGY SCHEAMTIC



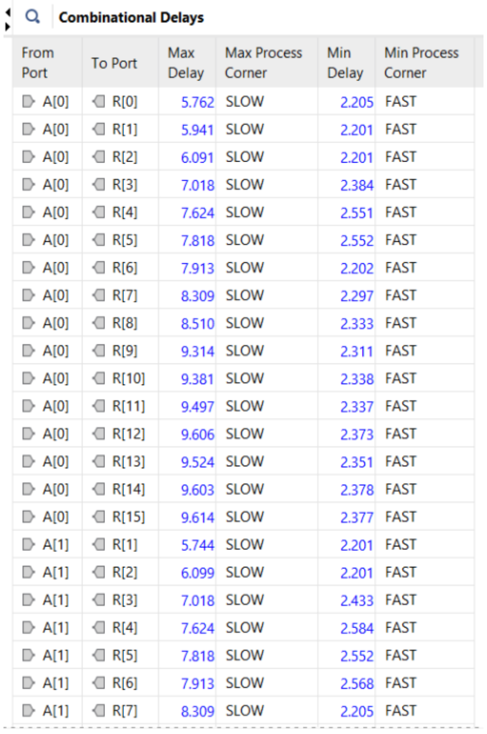


UTILIZATION RAPORT



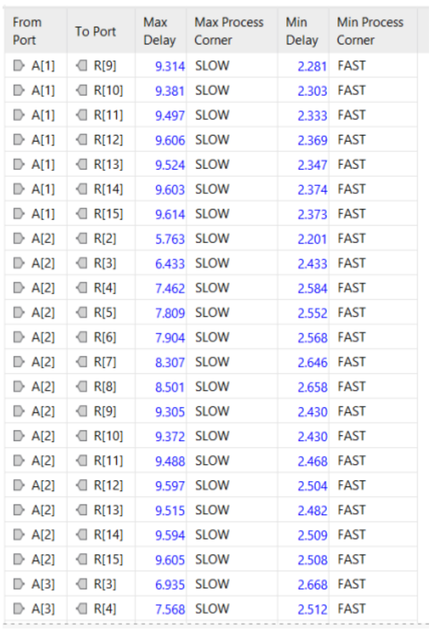
Name 1	Slice LUTs (63400)	Bonded IOB (210)
MULTB	71	32

TIMING REPORT

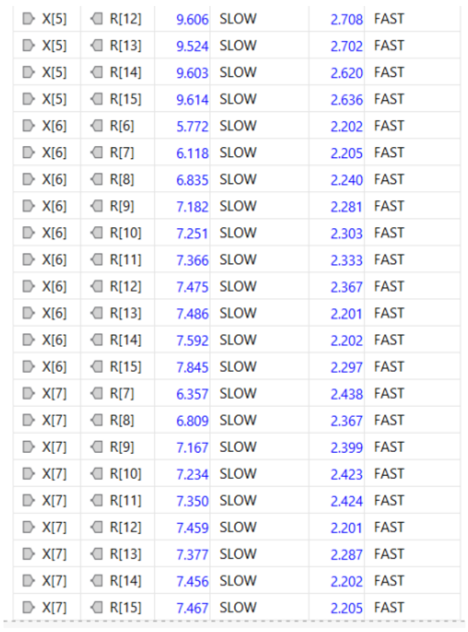


Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	R[0]	5.762	SLOW	2.205	FAST
A[0]	R[1]	5.941	SLOW	2.201	FAST
A[0]	R[2]	6.091	SLOW	2.201	FAST
A[0]	R[3]	7.018	SLOW	2.384	FAST
A[0]	R[4]	7.624	SLOW	2.551	FAST
A[0]	R[5]	7.818	SLOW	2.552	FAST
A[0]	R[6]	7.913	SLOW	2.202	FAST
A[0]	R[7]	8.309	SLOW	2.297	FAST
A[0]	R[8]	8.510	SLOW	2.333	FAST
A[0]	R[9]	9.314	SLOW	2.311	FAST
A[0]	R[10]	9.381	SLOW	2.338	FAST
A[0]	R[11]	9.497	SLOW	2.337	FAST
A[0]	R[12]	9.606	SLOW	2.373	FAST
A[0]	R[13]	9.524	SLOW	2.351	FAST
A[0]	R[14]	9.603	SLOW	2.378	FAST
A[0]	R[15]	9.614	SLOW	2.377	FAST
A[1]	R[1]	5.744	SLOW	2.201	FAST
A[1]	R[2]	6.099	SLOW	2.201	FAST
A[1]	R[3]	7.018	SLOW	2.433	FAST
A[1]	R[4]	7.624	SLOW	2.584	FAST
A[1]	R[5]	7.818	SLOW	2.552	FAST
A[1]	R[6]	7.913	SLOW	2.568	FAST
A[1]	R[7]	8.309	SLOW	2.205	FAST



From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[1]	R[9]	9.314	SLOW	2.281	FAST
A[1]	R[10]	9.381	SLOW	2.303	FAST
A[1]	R[11]	9.497	SLOW	2.333	FAST
A[1]	R[12]	9.606	SLOW	2.369	FAST
A[1]	R[13]	9.524	SLOW	2.347	FAST
A[1]	R[14]	9.603	SLOW	2.374	FAST
A[1]	R[15]	9.614	SLOW	2.373	FAST
A[2]	R[2]	5.763	SLOW	2.201	FAST
A[2]	R[3]	6.433	SLOW	2.433	FAST
A[2]	R[4]	7.462	SLOW	2.584	FAST
A[2]	R[5]	7.809	SLOW	2.552	FAST
A[2]	R[6]	7.904	SLOW	2.568	FAST
A[2]	R[7]	8.307	SLOW	2.646	FAST
A[2]	R[8]	8.501	SLOW	2.658	FAST
A[2]	R[9]	9.305	SLOW	2.430	FAST
A[2]	R[10]	9.372	SLOW	2.430	FAST
A[2]	R[11]	9.488	SLOW	2.468	FAST
A[2]	R[12]	9.597	SLOW	2.504	FAST
A[2]	R[13]	9.515	SLOW	2.482	FAST
A[2]	R[14]	9.594	SLOW	2.509	FAST
A[2]	R[15]	9.605	SLOW	2.508	FAST
A[3]	R[3]	6.935	SLOW	2.668	FAST
A[3]	R[4]	7.568	SLOW	2.512	FAST



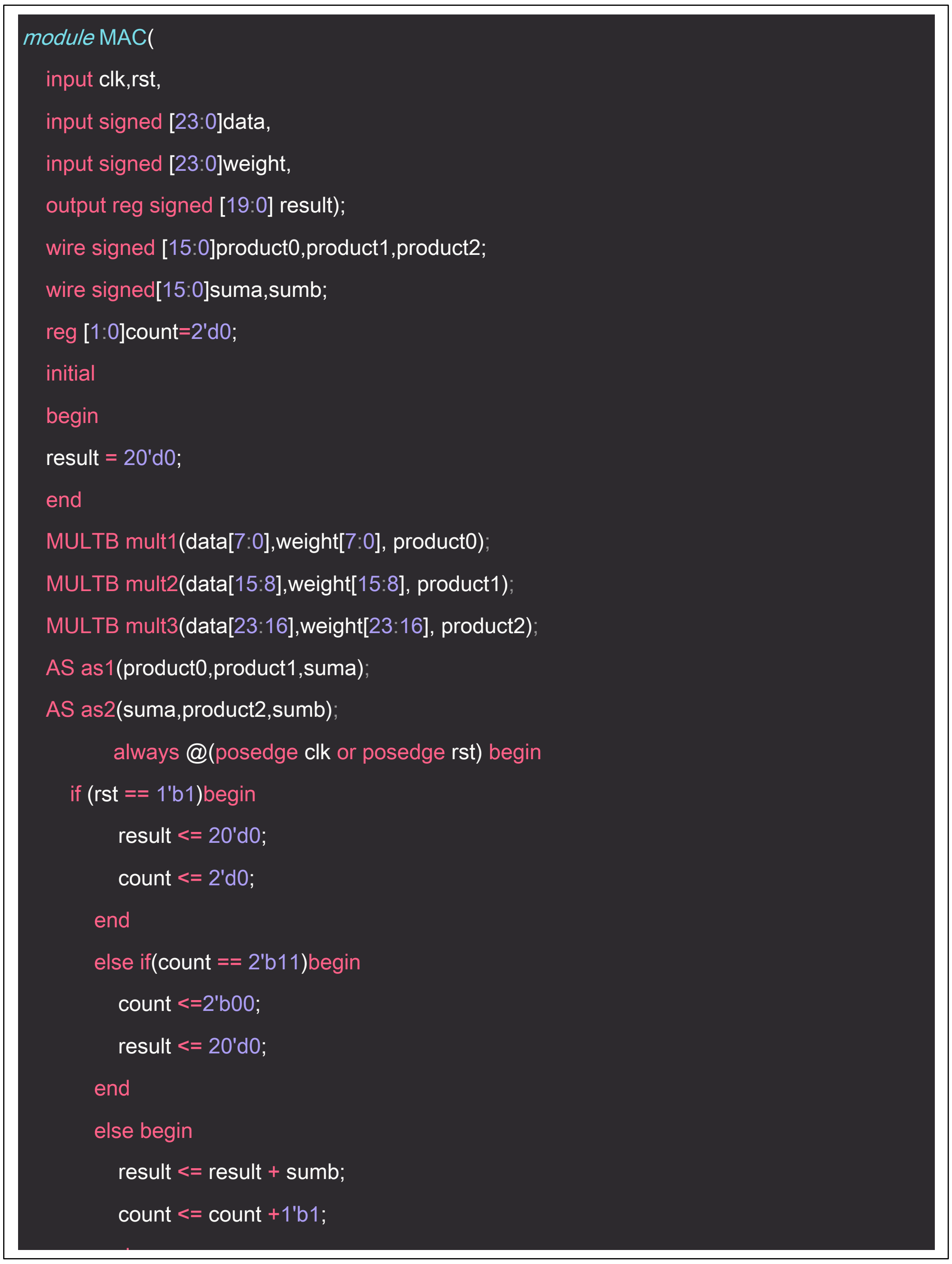
X[5]	R[12]	9.606	SLOW	2.708	FAST
X[5]	R[13]	9.524	SLOW	2.702	FAST
X[5]	R[14]	9.603	SLOW	2.620	FAST
X[5]	R[15]	9.614	SLOW	2.636	FAST
X[6]	R[6]	5.772	SLOW	2.202	FAST
X[6]	R[7]	6.118	SLOW	2.205	FAST
X[6]	R[8]	6.835	SLOW	2.240	FAST
X[6]	R[9]	7.182	SLOW	2.281	FAST
X[6]	R[10]	7.251	SLOW	2.303	FAST
X[6]	R[11]	7.366	SLOW	2.333	FAST
X[6]	R[12]	7.475	SLOW	2.367	FAST
X[6]	R[13]	7.486	SLOW	2.201	FAST
X[6]	R[14]	7.592	SLOW	2.202	FAST
X[6]	R[15]	7.845	SLOW	2.297	FAST
X[7]	R[7]	6.357	SLOW	2.438	FAST
X[7]	R[8]	6.809	SLOW	2.367	FAST
X[7]	R[9]	7.167	SLOW	2.399	FAST
X[7]	R[10]	7.234	SLOW	2.423	FAST
X[7]	R[11]	7.350	SLOW	2.424	FAST
X[7]	R[12]	7.459	SLOW	2.201	FAST
X[7]	R[13]	7.377	SLOW	2.287	FAST
X[7]	R[14]	7.456	SLOW	2.202	FAST
X[7]	R[15]	7.467	SLOW	2.205	FAST

COMPARISON BETWEEN STRUCTURAL AND MEHAVIORAL DESIGN

Structural design focuses on the physical arrangement and interconnection of the individual components, such as transistors and gates, that make up a circuit. Behavioral design, on the other hand, focuses on the function or behavior of the circuit, without considering its physical implementation. One key difference between structural and behavioral designs is that structural designs are more focused on the implementation details, while behavioral designs are more abstract and focused on the overall functionality of the circuit. This means that structural designs can be more complex and time-consuming to create, but they can also provide more control over the specific behavior of the circuit. Behavioral designs, on the other hand, are generally easier and faster to create, but they may not be as precise or customizable as structural designs. Another difference between the two approaches is that structural designs are typically easier to verify and validate, because they are based on the physical implementation of the circuit.

MULTIPLY AND ACCUMULATE (MAC)

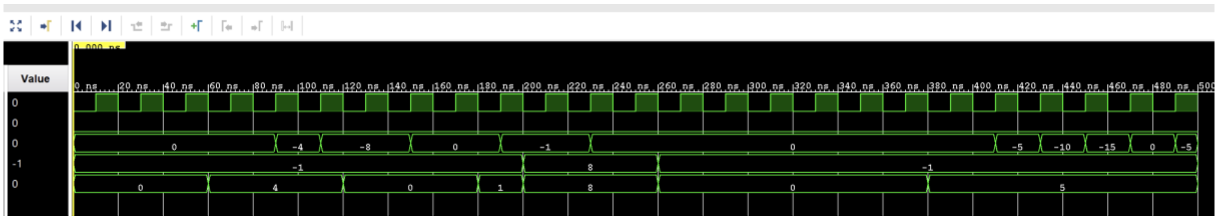
VERILOG CODE AND TESTBENCH



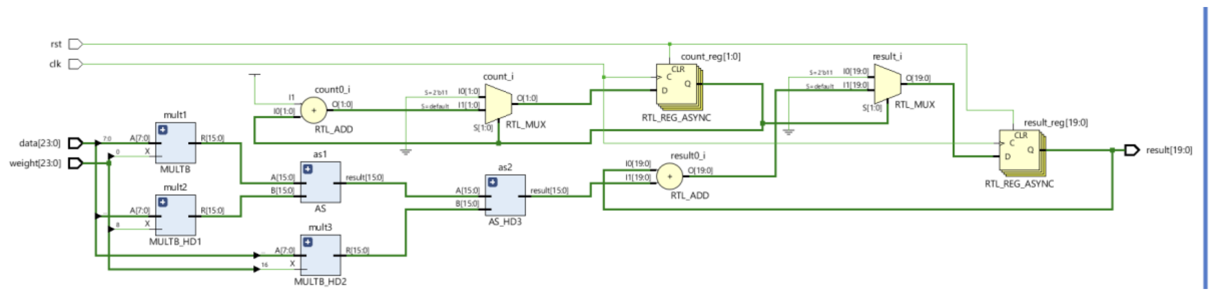
```
module MAC(
  input clk,rst,
  input signed [23:0]data,
  input signed [23:0]weight,
  output reg signed [19:0] result);
  wire signed [15:0]product0,product1,product2;
  wire signed[15:0]suma,sumb;
  reg [1:0]count=2'd0;
  initial
  begin
  result = 20'd0;
  end
  MULTB mult1(data[7:0],weight[7:0], product0);
  MULTB mult2(data[15:8],weight[15:8], product1);
  MULTB mult3(data[23:16],weight[23:16], product2);
  AS as1(product0,product1,suma);
  AS as2(suma,product2,sumb);
        always @(posedge clk or posedge rst) begin
    if (rst == 1'b1)begin
        result <= 20'd0;
        count <= 2'd0;
      end
      else if(count == 2'b11)begin
        count <=2'b00;
        result <= 20'd0;
      end
      else begin
        result <= result + sumb;
        count <= count +1'b1;
```

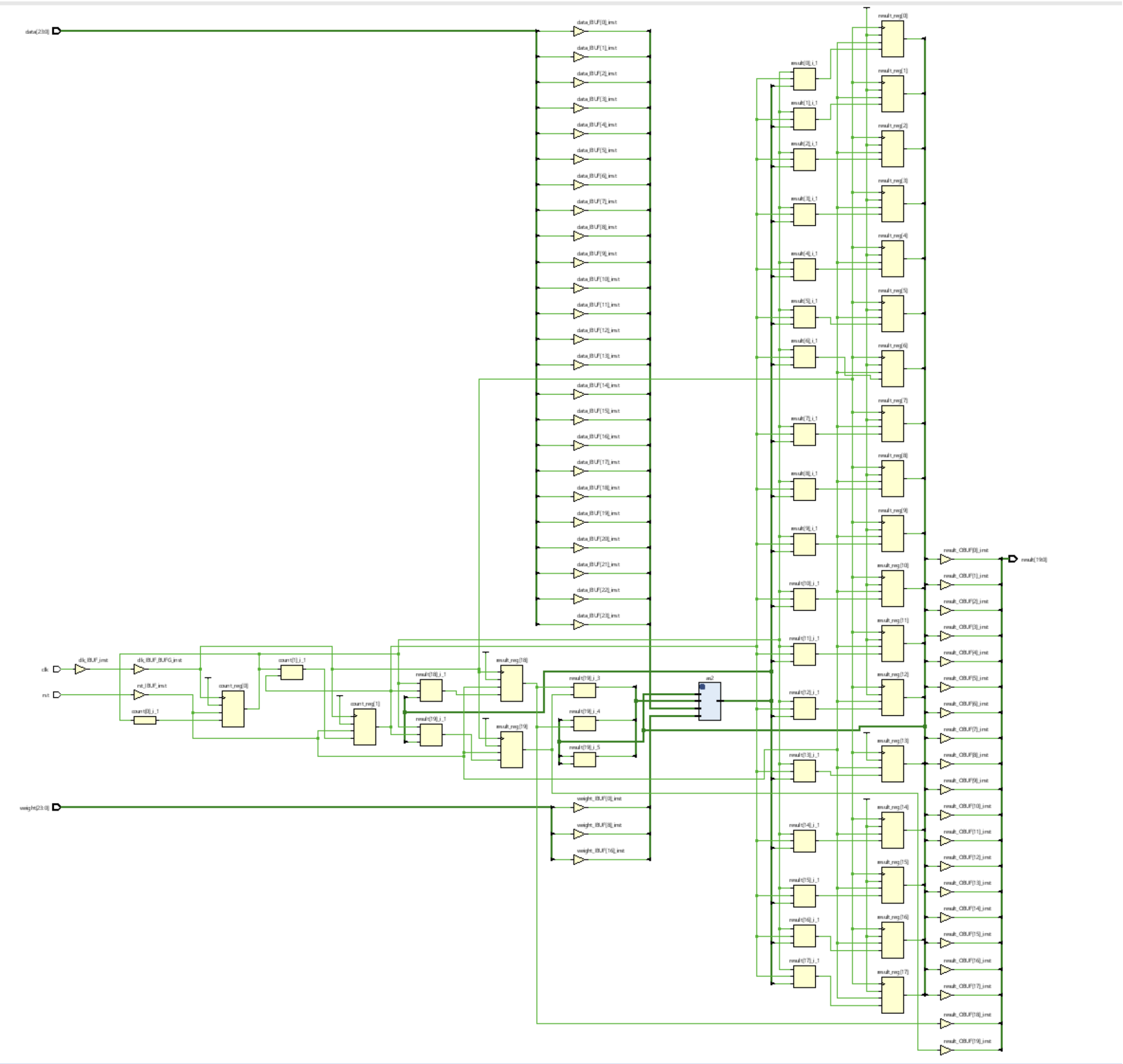
```
module tb_MAC();
   reg clk,rst;
   wire  [19:0] result;
   reg [23:0] weight;
   reg [23:0] data;
   MAC uut(.clk(clk),.rst(rst),.data(data),.weight(weight),.result(result));

   initial
   begin
  rst=1'b0;
  clk=1'b0;
  data=24'd0;
  weight=-24'd1; #10 clk=1'b1;  #10 clk=1'b0; #10  clk=1'b1;  #10  clk=1'b0;  #10 clk=1'b1;  #10
data=24'd4; 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'd0; 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'd1; clk=1'b0;
  #10 clk=1'b1;  #10  data=24'd8;  weight=24'd8;  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'd0;  weight=-24'd1;  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'd0;  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'd5;  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'd5;clk=1'b0;  #10  clk=1'b1;  #10  clk=1'b0;  #10  clk=1'b1;  #10  clk=1'b0;  #10  clk=1'b1;  #10
      $finish;
   end
```

SIMULATION RESULT (WAVEFORM AND TCL CONSOLE)

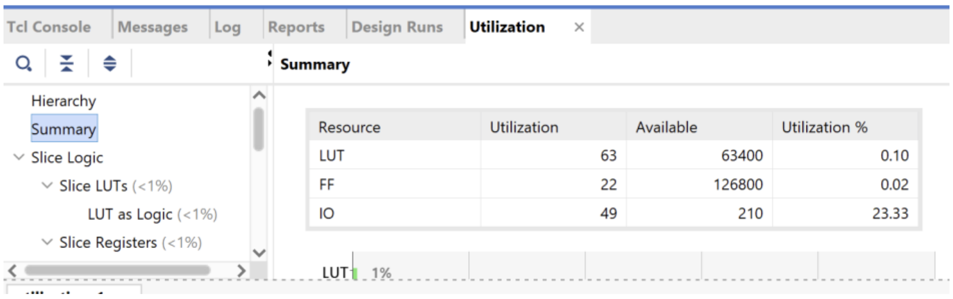


RTL AND TECHNOLOGY SCHEAMTIC





UTILIZATION RAPORT



Resource	Utilization	Available	Utilization %
LUT	63	63400	0.10
FF	22	126800	0.02
IO	49	210	23.33

TIMING REPORT

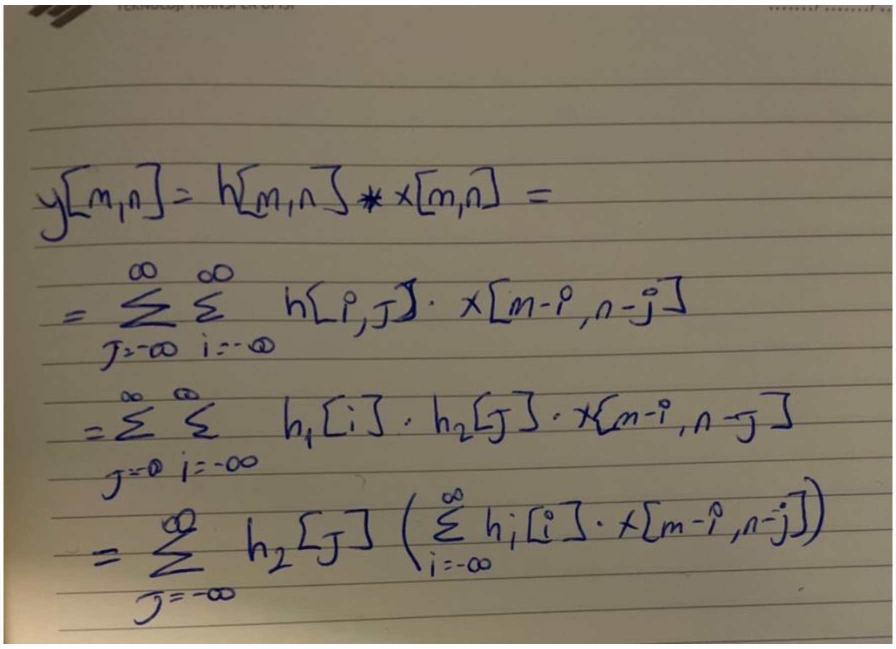
I cant get timing summary . in my opinion it coused by vivado. Sometime vivado give error.

2D CONVOLUTION

EXPLAIN 2-D CONVOLUTION BRIEFLY

Convolution of two functions involves moving one function and adding the results while multiplying the outputs of the two functions. In 2D convolution, a single function is moved by two dimensional inputs and accumulates across all, much like in 1D convolution.

MATHEMATICAL CALCULATION IF 2-D CONVOLUTION



$$y[m,n] = h[m,n] * x[m,n] =$$

$$= \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} h[i,j] \cdot x[m-i, n-j]$$

$$= \sum_{j=\infty}^{\infty} \sum_{i=-\infty}^{\infty} h_1[i] \cdot h_2[j] \cdot x[m-i, n-j]$$

$$= \sum_{j=-\infty}^{\infty} h_2[j] \left(\sum_{i=-\infty}^{\infty} h_1[i] \cdot x[m-i, n-j] \right)$$

EXPLAIN HOW 2-D CONVOLUTION (CAN BE CALCULATED BY SLİDİNG THE KERNEL WİNDOW OVER THE İMAGE)

In 5*5 and 3*3 multiplication, the first 3 rows and 3 columns of the 5*5 matrix are multiplied by the entire 3*3 matrix. Then the w matrix is multiplied with the 3*3 part to the right of the first 3*3 part of the 5*5 matrix. After doing this 3 times, it is done again by skipping the first row of the 5*5 matrix. Thus, a 3*3 matrix is obtained as a result.

VERILOG CODE AND TESTBENCH

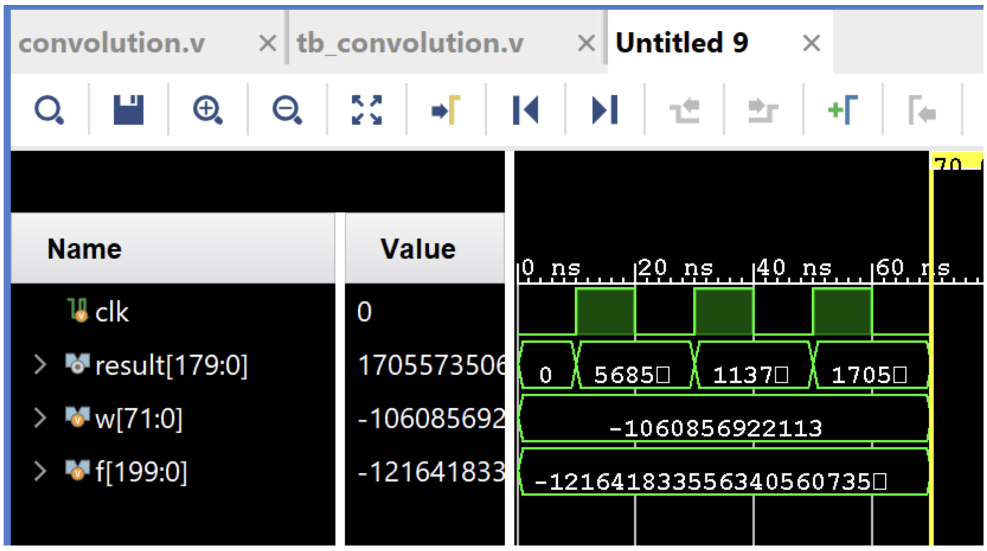
```
module D2C(
    input [199:0]f,
    input [71:0]w,
    input clk,
    output [179:0]result
);
    wire rst=1'b0;
    wire [19:0]
result99,result1,result2,result3,result4,result5,result6,result7,result8,result9,result10,result11,result12,result13,result14,result15,result16
,result17,result18,result19,result20,result21,result22,result23,result24,result25,resul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



    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 result[19:0]= result99+result1+result2;

    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[23:16],w[15:8],w[7:0]}),.result(result5));

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

    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 result[59:40]= result8+result6+result7;

    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 result[79:60]= result9+result10+result11;

    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 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 result[79:60]= result9+result10+result11;

  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 result[99:80]= result12+result13+result14;

  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[47:40],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 result[119:100]= result15+result16+result17;

  MAC mac111(.clk(clk),.reset(rst),.data({f[87:80],f[95:88],f[103:96]}),.weight({w[71:64],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 result[139:120]= result18+result19+result20;

  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 result[159:140]= result21+result22+result23;

  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 result[139:120]= result18+result19+result20;


   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 result[159:140]= result21+result22+result23;


   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 result[179:160]= result24+result25+result26;
```

```
endmodule
```

```
module tb_D2C();
  reg clk;
  wire  [179:0] result;
  reg [71:0] w;
  reg [199:0] f;
  D2C uut(.clk(clk),.f(f),.w(w),.result(result));
  initial
  begin
    clk=1'b0;
    f[7:0]=8'd128;f[15:8]=8'd128;f[23:16]=8'd128;f[31:24]=8'd128;f[39:32]=8'd128;
    f[47:40]=8'd255;f[55:48]=8'd255;f[63:56]=8'd128;f[71:64]=8'd255;f[79:72]=8'd255;
    f[87:80]=8'd255;f[95:88]=8'd255;f[103:96]=8'd128;f[111:104]=8'd255;f[119:112]=8'd255;
    f[127:120]=8'd255;f[135:128]=8'd255;f[143:136]=8'd128;f[151:144]=8'd255;f[159:152]=8'd255;
    f[167:160]=8'd255;f[175:168]=8'd255;f[183:176]=8'd128;f[191:184]=8'd255;f[199:192]=8'd255;
    w[7:0]=-8'd1;w[15:8]=-8'd1;w[23:16]=-8'd1;w[31:24]=-8'd1;w[39:32]=8'd8;
    w[47:40]=-8'd1;w[55:48]=-8'd1;w[63:56]=-8'd1;w[71:64]=-8'd1;
    #10clk=1'b1;#10; clk=1'b0;#10; clk=1'b1; #10; clk=1'b0;#10; clk=1'b1; #10 ;  clk=1'b0;
       #10  if(result[19:0]==20'd635)
      $display("TRUE",f,w,result);
       else
      $display("FALSE",f,w,result);
     if(result[39:20]==-20'd508)
      $display("TRUE",f,w,result);
    else
      $display("FALSE",f,w,result);
    if(result[59:40]==20'd635)
      $display("TRUE",f,w,result);
    else
      $display("FALSE",f,w,result);
    if(result[79:60]==20'd381)
      $display("TRUE",f,w,result);
    else
      $display("FALSE",f,w,result);
    if(result[99:80]==-20'd762)
      $display("TRUE",f,w,result);
    else
      $display("FALSE",f,w,result);
      if(result[119:100]==20'd381)
      $display("TRUE",f,w,result);
    else
      $display("FALSE",f,w,result);
      if(result[139:120]==20'd381)
```

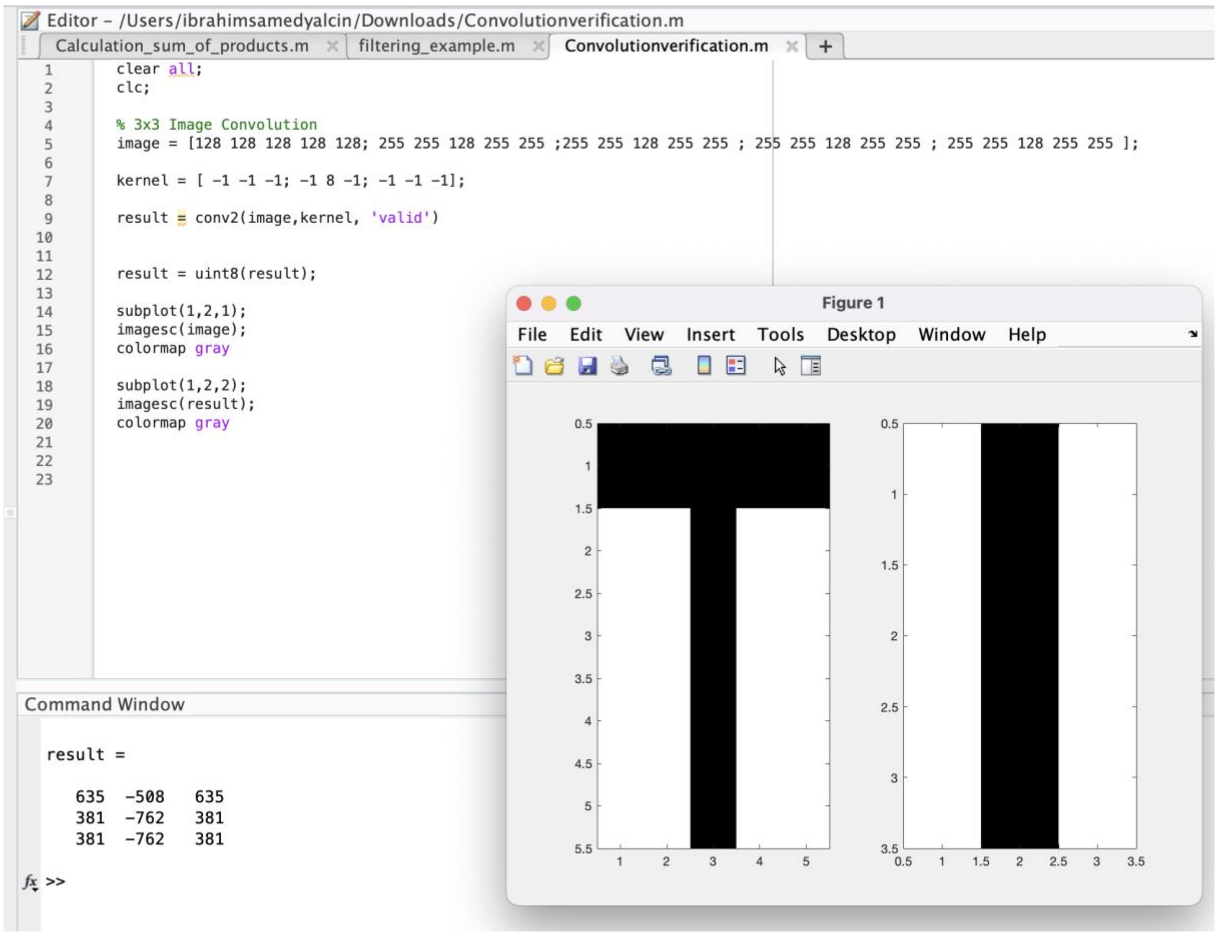
SIMULATION RESULT (WAVEFORM)



Name	Value
clk	0
result[179:0]	1705573506
w[71:0]	-106085692
f[199:0]	-121641833

```
result: 0 | 5685 | 1137 | 1705
w: -1060856922113
f: -12164183355634056073 5
```

MATLAB CODE AND THE RESULT (OBTAİNED FROM COMMAND WİNDOW)


Editor – /Users/ibrahimsamedyalcin/Downloads/Convolutionverification.m
Calculation_sum_of_products.m
filtering_example.m
Convolutionverification.m
Figure 1
File Edit View Insert Tools Desktop Window Help
Command Window

```
clear all;
clc;

% 3x3 Image Convolution
image = [128 128 128 128 128; 255 255 128 255 255 ;255 255 128 255 255 ; 255 255 128 255 255 ; 255 255 128 255 255 ];

kernel = [ -1 -1 -1; -1 8 -1; -1 -1 -1];

result = conv2(image,kernel, 'valid')


result = uint8(result);

subplot(1,2,1);
imagesc(image);
colormap gray

subplot(1,2,2);
imagesc(result);
colormap gray
```

```
result =

   635  -508   635
   381  -762   381
   381  -762   381

>>
```