DIGITAL SYSTEM DESIGN APPLICATIONS

Experiment 7

CONVOLUTIONCIRCUITS

Berfin Duman
040190108

Contents

1 Structural Multiplier- Unsigned

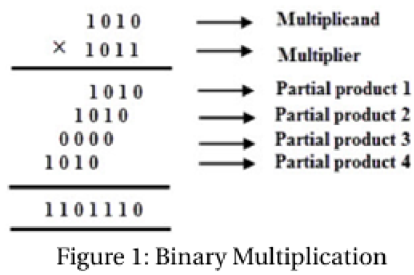


Figure 1: Binary Multiplication

$$
\begin{aligned}
\mathrm{PP0} = \quad & x_0 * \quad A * \quad 2^0 \\
\mathrm{PP1} = \quad & x_1 * \quad A * \quad 2^1 \\
\mathrm{PP2} = \quad & x_2 * \quad A * \quad 2^2 \\
\mathrm{PP3} = \quad & x_3 * \quad A * \quad 2^3 \\
\mathrm{PP4} = \quad & x_4 * \quad A * \quad 2^4 \\
\mathrm{PP5} = \quad & x_5 * \quad A * \quad 2^5 \\
\mathrm{PP6} = \quad & x_6 * \quad A * \quad 2^6 \\
\mathrm{PP7} = \quad & x_7 * \quad A * \quad 2^7
\end{aligned}
$$

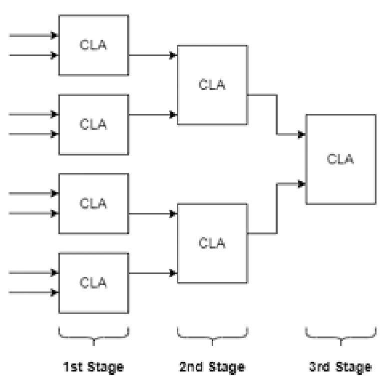


Figure 2: Three Adder Stages

Listing 1: mults module

```
`timescale 1ns / 1ps
module mults(
    input [7:0] X, A,
    output [15:0] result
```

```
    );
    wire [15:0] PP [7:0];
    wire number;
    wire [15:0] sum [6:0];
    wire [6:0] cout;
    initial
    begin
    end
    genvar i;
    generate
    for (i = 0; i < 8; i = i + 1) begin : PARTIAL_PRODUCTS
        assign PP[i][15:0]  = (X[i] == 1'b1) ? (A << i) :
           16'b0;
    end
    endgenerate
    // sum of partial product
    CLA CLA1(PP[0], PP[1], 1'b0, cout[0], sum[0]);
    CLA CLA2(PP[2], PP[3], 1'b0, cout[1], sum[1]);
    CLA CLA3(PP[4], PP[5], 1'b0, cout[2], sum[2]);
    CLA CLA4(PP[6], PP[7], 1'b0, cout[3], sum[3]);
    // sum of result of partial product
    CLA CLA5(sum[0], sum[1],1'b0, cout[4], sum[4]);
    CLA CLA6(sum[2], sum[3],1'b0, cout[5], sum[5]);
    CLA CLA7(sum[4], sum[5],1'b0, cout[6], sum[6]);
    assign result= sum[6];

endmodule
```

I used same testbench for behavioral or structural multiplier.

Listing 2: mults_tb.v

```
`timescale 1ns / 1ps

module mults_tb();
    reg [7:0] A;
    reg [7:0] X;
    wire [15:0] result;
    integer i;

    reg [7:0] A_s [7:0];
    reg [7:0] X_s [7:0];
    reg [15:0] true_ans [7:0];

    initial
    begin
        A_s[0]=0;  X_s[0]=0; true_ans[0]=0;
```

```
        A_s[1]=8;   X_s[1]=14;  true_ans[1]=112;
        A_s[2]=13;  X_s[2]=6;  true_ans[2]=78;
        A_s[3]=2;   X_s[3]=11;   true_ans[3]=22;
        A_s[4]=36;  X_s[4]=82;  true_ans[4]=2952;
        A_s[5]=4;   X_s[5]=75;true_ans[5]=300;
        A_s[6]=121;X_s[6]=139;   true_ans[6]=16819;
        A_s[7]=194;X_s[7]=237;true_ans[7]=45978;
    end
        /*
    multb UUT
    (
        .A(A),
        .B(X),
        .result(result)
    );
    */
    mults UUT
    //mults_signed UUT
    (
        .A(A),
        .X(X),
        .result(result)
    );

    initial
    begin
        for (i = 0; i < 8; i = i + 1) begin

            A = A_s[i];
            X = X_s[i];
             #15;
            $write("A * X = %d * %d => Result = %d, True
               Result= %d \n", A, X, result,true_ans[i] );
             if(result == true_ans[i])
                $display("TRUE");
             else
                $display("FALSE");


        end
        $finish;
    end
endmodule
```

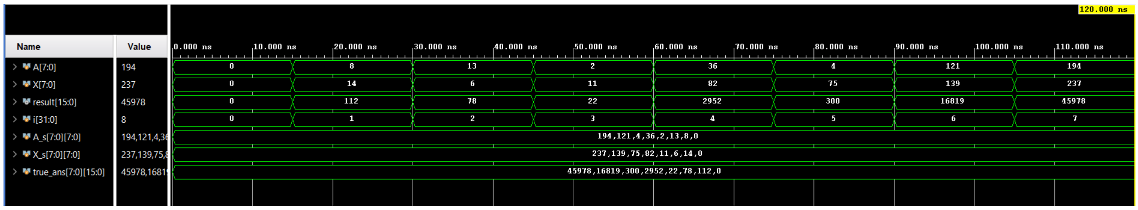


Figure 3: Binary Multiplication Simulation Result

```
A * X =   0 *   0 => Result =     0, True Result=     0
TRUE
A * X =   8 *  14 => Result =   112, True Result=   112
TRUE
A * X =  13 *   6 => Result =    78, True Result=    78
TRUE
A * X =   2 *  11 => Result =    22, True Result=    22
TRUE
A * X =  36 *  82 => Result =  2952, True Result=  2952
TRUE
A * X =   4 *  75 => Result =   300, True Result=   300
TRUE
A * X = 121 * 139 => Result = 16819, True Result= 16819
TRUE
A * X = 194 * 237 => Result = 45978, True Result= 45978
TRUE
$finish called at time : 120 ns : File
```

2 Structual Multiplier- Signed

2.1 Baugh-Wooley Method

One notable multiplication technique is the Baugh-Wooley method, designed for the multiplication of both positive and negative numbers. In this approach, the multiplication is executed by considering all the significant bits in the partial products, excluding the last partial product. In the final partial multiplication, only the most significant bit is considered. The illustration below depicts this procedure using a specific example, where each box represents a multiplication operation. The blue segments denote identical products, while the black segments represent the complemented products. The resulting numbers are then summed, proceeding from the most significant bit to the least significant bit.

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \quad (1)$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \quad (2)$$

Figure 4: Baugh Walley Method-1

The product, $P = A \times B$, is then given by the following equation:

$$\begin{aligned} P &= A \times B \\ &= \left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i\right) \times \left(-b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j\right) \\ &= a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} \\ &\quad -2^{n-1}\sum_{i=0}^{n-2} a_i b_{n-1} 2^i - 2^{n-1}\sum_{j=0}^{n-2} a_{n-1} b_j 2^j \end{aligned}$$

Figure 5: Baugh Walley Method-2

As we are performing the addition of two 8-bit numbers, it necessitates the computation of 8 partial products, leading to the requirement of utilizing 7 distinct Carry Look-Ahead (CLA) units. Specifically, 4 CLAs are employed for the summation of the partial products, 2 for combining the results of the initial 4 CLAs, and a final CLA for consolidating the last 2 sums. However, due to the application of the Baugh-Wooley method, an additional stage is introduced to accommodate the final number addition. Consequently, a total of 4 stages and 8 CLAs are essential for the overall operation.

					$\overline{a_4x_0}$	a_3x_0	a_2x_0	a_1x_0	a_0x_0
				$\overline{a_4x_1}$	a_3x_1	a_2x_1	a_1x_1	a_0x_1	
			$\overline{a_4x_2}$	a_3x_2	a_2x_2	a_1x_1	a_0x_2		
		$\overline{a_4x_3}$	a_3x_3	a_2x_3	a_1x_3	a_0x_3			
	a_4x_4	$\overline{a_3x_4}$	$\overline{a_2x_4}$	$\overline{a_1x_4}$	$\overline{a_0x_4}$				
1				1					
p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

Figure 6: Baugh-Wooley Method for 4x4 Multiplier

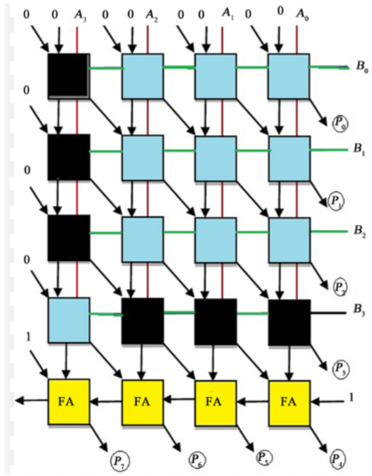


Figure 7: Block Diagram of Baugh-Wooley for 4x4 Multiplier

Listing 3: mults_signed module

```
`timescale 1ns / 1ps

module mults_signed(
     input [7:0] X, A,
    output [15:0] result);
    wire [7:0] PP [7:0];
    wire [15:0] PP_2 [7:0];
    wire [7:0] PP_1 [7:0];
    wire [15:0] sum [7:0];
    wire cout [7:0];
    genvar i;
    generate
    for (i = 0; i < 8; i = i + 1)
    begin : PP_start
        assign PP[i][7:0]  = X[i] & A[i];

    end
    endgenerate
    genvar j,k;
    generate
    for (j = 0; j < 8; j = j + 1)
    begin : PP_f1
        for (k = 0; k < 8; k = k + 1)
        begin
        if (j!=7)
            begin
             if (k!=7)
                begin
```

```
                assign PP_1[j][k]  = PP[j][k];
                end
             else
                begin
                assign PP_1[j][k]  = ~PP[j][k];
                end
            end
        else
            begin
             if (k!=7)
                begin
                assign PP_1[j][k]  =~PP[j][k];
                end
             else
                begin
                assign PP_1[j][k]  = PP[j][k];
                end
            end
        end
    end
    endgenerate

    genvar t;
    generate
    for (t = 0; t < 8; t = t + 1)
    begin
        assign PP_2[t][15:0]  = PP_1[t]<<t;
    end
    endgenerate


    CLA CLA1(PP_2[0][15:0], PP_2[1][15:0], 1'b0, cout[0],
       sum[0][15:0]);
    CLA CLA2(PP_2[2][15:0], PP_2[3][15:0], 1'b0, cout[1],
       sum[1][15:0]);
    CLA CLA3(PP_2[4][15:0], PP_2[5][15:0], 1'b0, cout[2],
       sum[2][15:0]);
    CLA CLA4(PP_2[6][15:0], PP_2[7][15:0], 1'b0, cout[3],
       sum[3][15:0]);
    CLA CLA5(sum[0][15:0], sum[1][15:0], 1'b0, cout[4],
       sum[4][15:0]);
    CLA CLA6(sum[2][15:0], sum[3][15:0], 1'b0, cout[5],
       sum[5][15:0]);
    CLA CLA7(sum[4][15:0], sum[5][15:0], 1'b0, cout[6],
       sum[6][15:0]);
```

```
    CLA CLA8(sum[6][15:0], 16'b1000000100000000, 0, cout[7],
       sum[7][15:0]);
    assign result[15:0] = sum[7][15:0];

endmodule
```

Listing 4: mults$_t b.v$

```
`timescale 1ns / 1ps

module mults_tb();
    reg [7:0] A;
    reg [7:0] X;
    wire [15:0] result;
    integer i;

    reg [7:0] A_s [7:0];
    reg [7:0] X_s [7:0];
    reg [15:0] true_ans [7:0];

    initial
    begin
        A_s[0]=0;  X_s[0]=0; true_ans[0]=0;
        A_s[1]=8;  X_s[1]=14; true_ans[1]=112;
        A_s[2]=13; X_s[2]=6; true_ans[2]=78;
        A_s[3]=2;  X_s[3]=11;  true_ans[3]=22;
        A_s[4]=36; X_s[4]=82; true_ans[4]=2952;
        A_s[5]=4;  X_s[5]=75;true_ans[5]=300;
        A_s[6]=121;X_s[6]=139;  true_ans[6]=16819;
        A_s[7]=194;X_s[7]=237;true_ans[7]=45978;
    end

    mults UUT
    //mults_signed UUT
    (
        .A(A),
        .X(X),
        .result(result)
    );

    initial
    begin
        for (i = 0; i < 8; i = i + 1) begin

            A = A_s[i];
            X = X_s[i];
```

```
                #15;
                $write("A * X = %d * %d => Result = %d, True
                   Result= %d \n", A, X, result,true_ans[i] );
                if(result == true_ans[i])
                    $display("TRUE");
                else
                    $display("FALSE");


            end
            $finish;
        end
endmodule
```

```
Time resolution is 1 ps
A * X =    0 *   -1 => Result =     0, True Result=     0
TRUE
A * X =  -13 *    6 => Result =   -78, True Result=   -78
TRUE
A * X =  -40 *   -9 => Result =   360, True Result=   360
TRUE
A * X =   12 *  -12 => Result =  -144, True Result=  -144
TRUE
A * X =   47 *   68 => Result =  3196, True Result=  3196
TRUE
A * X =  -31 *  -47 => Result =  1457, True Result=  1457
TRUE
A * X =   86 *  -19 => Result = -1634, True Result= -1634
TRUE
A * X =  -48 *   18 => Result =  -864, True Result=  -864
TRUE
$finish called at time : 120 ns : File
```

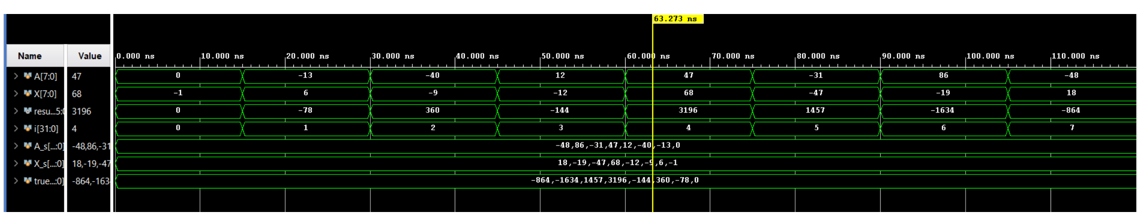


Figure 8: Multiple Signed Module Simulation Results

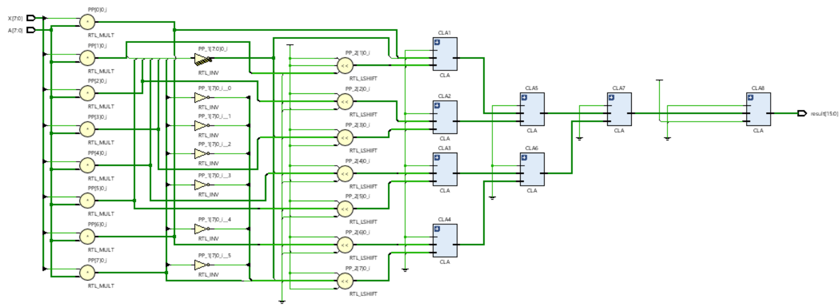


Figure 9: Multiple Signed Module Rtl Schematic

2.2 Synthesis and Implementation Part

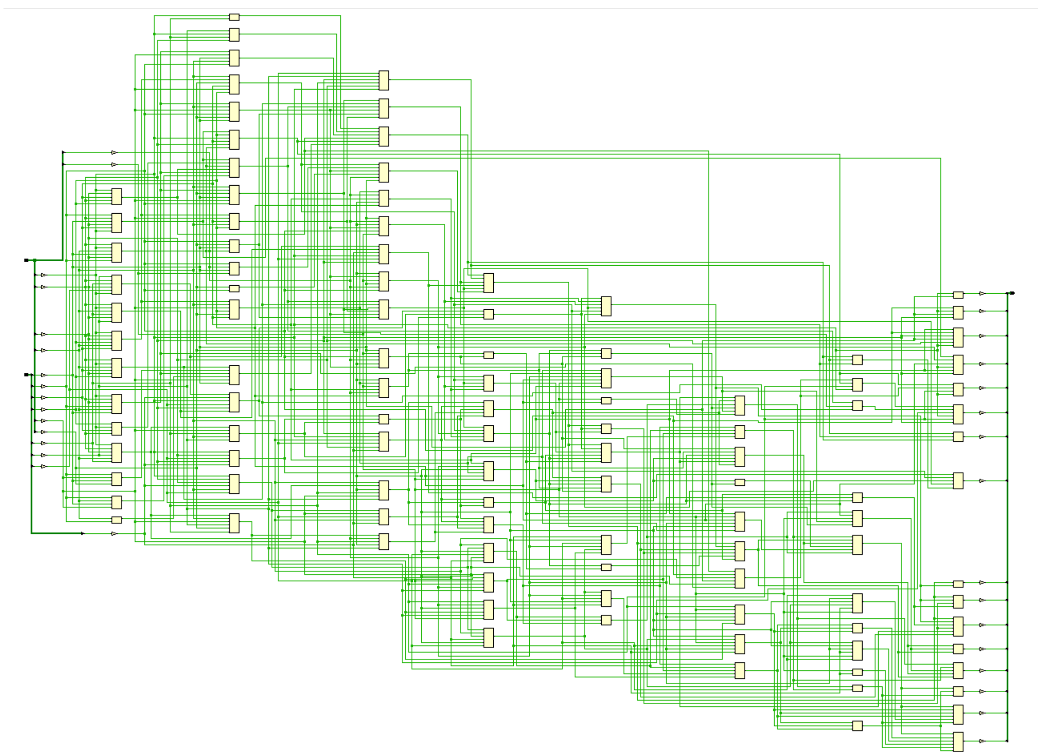
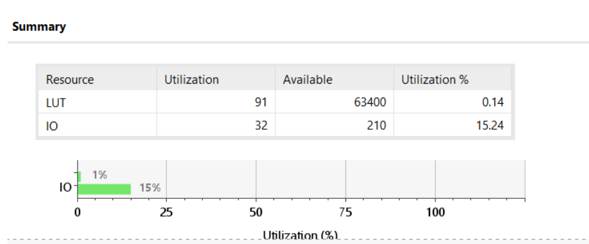


Figure 10: Multiple Signed Module Technolgy Schematic



Summary

Resource	Utilization	Available	Utilization %
LUT	91	63400	0.14
IO	32	210	15.24

Figure 11: Multiple Signed Module Utilization Report

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[4]	result[11]	14.374	SLOW	3.396	FAST
X[5]	result[15]	14.372	SLOW	3.010	FAST
X[5]	result[14]	14.345	SLOW	2.830	FAST
X[4]	result[15]	14.328	SLOW	3.118	FAST
X[0]	result[14]	14.322	SLOW	3.421	FAST
X[1]	result[14]	14.314	SLOW	3.128	FAST
A[5]	result[14]	14.308	SLOW	2.918	FAST
X[4]	result[14]	14.300	SLOW	2.939	FAST
A[4]	result[12]	14.252	SLOW	3.341	FAST
A[5]	result[13]	14.244	SLOW	3.214	FAST
A[2]	result[11]	14.228	SLOW	3.590	FAST
X[6]	result[15]	14.217	SLOW	3.000	FAST
X[5]	result[13]	14.204	SLOW	3.126	FAST
X[6]	result[14]	14.189	SLOW	2.820	FAST
X[3]	result[15]	14.179	SLOW	3.224	FAST
X[4]	result[13]	14.159	SLOW	3.234	FAST
A[0]	result[11]	14.117	SLOW	3.418	FAST
A[2]	result[12]	14.109	SLOW	3.535	FAST
X[6]	result[13]	14.048	SLOW	3.119	FAST
X[3]	result[14]	14.010	SLOW	2.951	FAST
A[1]	result[11]	14.007	SLOW	3.505	FAST
A[0]	result[12]	13.996	SLOW	3.363	FAST
X[7]	result[15]	13.958	SLOW	2.654	FAST

Figure 12: Multiple Signed Module Timing Summary

3 Behavioral Multipler

Listing 5: mults$_t b.v$

```
`timescale 1ns / 1ps
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
```

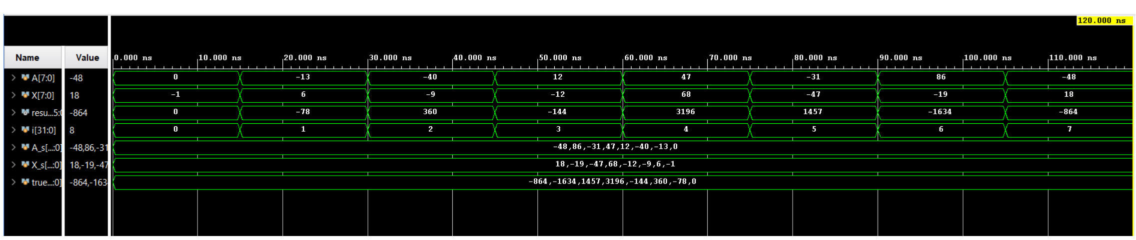


Figure 13: Caption

Using the same bench as the previous signed multiplier, I obtained identical, correct results.

```
A * X =    0 *   -1 => Result =      0, True Result=      0
TRUE
A * X =  -13 *    6 => Result =    -78, True Result=    -78
TRUE
A * X =  -40 *   -9 => Result =    360, True Result=    360
TRUE
A * X =   12 *  -12 => Result =   -144, True Result=   -144
TRUE
A * X =   47 *   68 => Result =   3196, True Result=   3196
TRUE
A * X =  -31 *  -47 => Result =   1457, True Result=   1457
TRUE
A * X =   86 *  -19 => Result =  -1634, True Result=  -1634
TRUE
A * X =  -48 *   18 => Result =   -864, True Result=   -864
TRUE
```

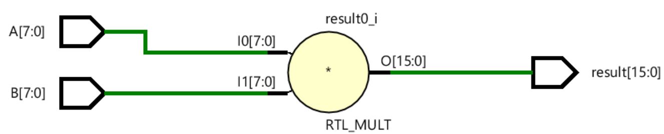


Figure 14: Behavioral Multiple Signed Module Rtl Schematic

3.1 Synthesis and Implementation Part

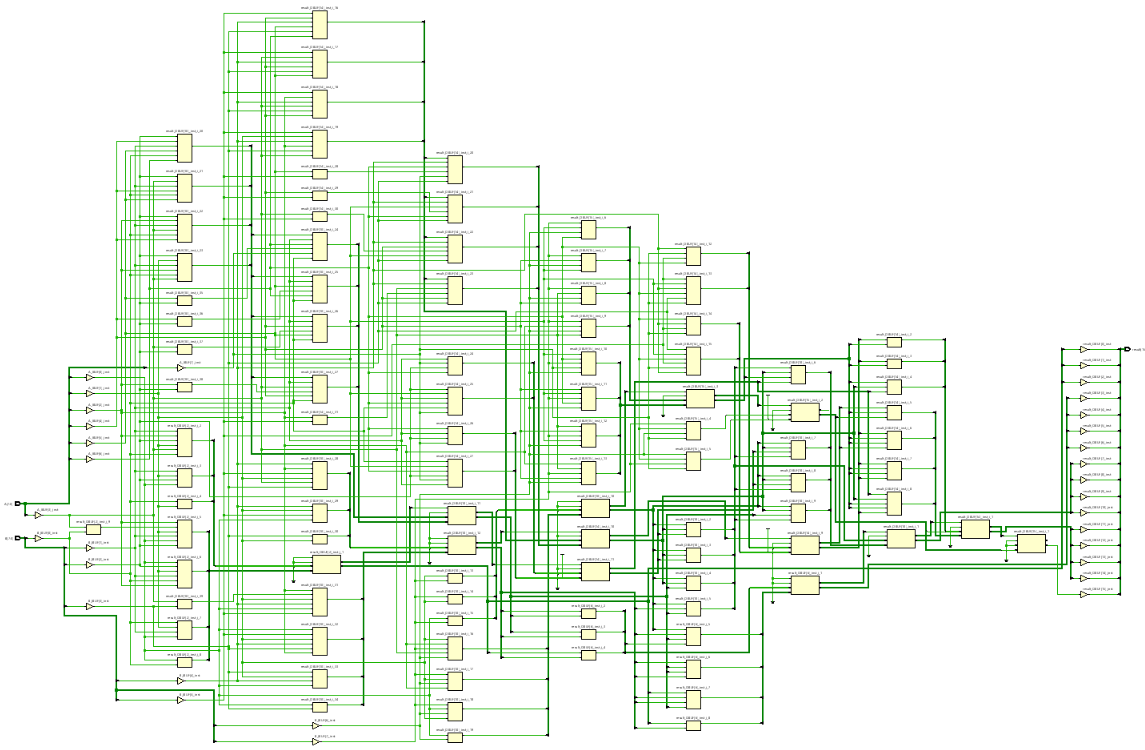
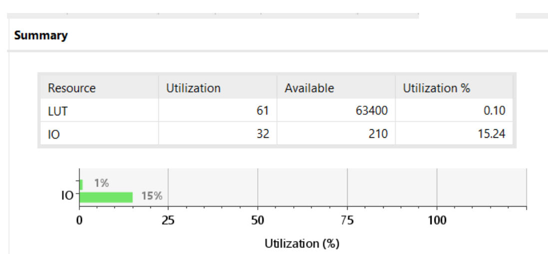


Figure 15: Behavioral Multiple Signed Module Technolgy Schematic



Summary

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

Figure 16: Behavioral Multiple Signed Module Utilization Report

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
B[2]	result[15]	12.966	SLOW	3.204	FAST
B[2]	result[14]	12.926	SLOW	3.192	FAST
B[2]	result[12]	12.902	SLOW	3.153	FAST
A[1]	result[15]	12.799	SLOW	3.403	FAST
A[1]	result[14]	12.759	SLOW	3.391	FAST
A[1]	result[12]	12.736	SLOW	3.329	FAST
B[2]	result[13]	12.698	SLOW	3.109	FAST
A[4]	result[15]	12.668	SLOW	3.271	FAST
B[2]	result[11]	12.647	SLOW	3.082	FAST
A[4]	result[14]	12.628	SLOW	3.186	FAST
A[2]	result[15]	12.612	SLOW	3.331	FAST
A[4]	result[12]	12.605	SLOW	3.166	FAST
A[2]	result[14]	12.572	SLOW	3.292	FAST
A[2]	result[12]	12.549	SLOW	3.276	FAST
B[2]	result[10]	12.541	SLOW	3.032	FAST
A[1]	result[13]	12.532	SLOW	3.309	FAST
A[3]	result[15]	12.493	SLOW	3.323	FAST
A[1]	result[11]	12.480	SLOW	3.270	FAST
B[4]	result[15]	12.476	SLOW	3.140	FAST
B[2]	result[9]	12.468	SLOW	3.065	FAST
A[3]	result[14]	12.453	SLOW	3.240	FAST
B[4]	result[14]	12.436	SLOW	3.093	FAST
A[3]	result[12]	12.429	SLOW	3.267	FAST

Figure 17: Behavioral Multiple Signed Module Timing Summary

3.2 Comparison of Multipliers

The behavioral design employs 61 LUTs for its operation, while the structural design uses 91 LUTs. Hence, the number of LUTs in behavioral design is less than that of the structural design. Additionally, for timing comparison, the maximum delay observed in structural design is 14.374 ns, while in behavioral design, it is 12.966 ns. Thus, the behavioral design has smaller delays than the structural design.

4 Multiply and Accumulate (MAC)

Listing 6: MAC.v

```
module MAC(
input clk, rst,
input signed [23:0] data,weight,
output reg signed [19:0] result);
wire signed [15:0] product [2:0];
wire signed [15:0] sum[1:0];
reg signed [19:0] result_temp;
multb multb1(.A(data[7:0]), .B(weight[7:0]),
    .result(product[0]));
multb multb2(.A(data[15:8]), .B(weight[15:8]),
    .result(product[1]));
multb multb3(.A(data[23:16]), .B(weight[23:16]),
    .result(product[2]));
adderb adderb1(.A(product[0]),.B(product[1]),.result(sum[0]));
```

```
adderb adderb2(.A(product[2]),.B(sum[0]),.result(sum[1]));

reg  [1:0]count;
always @(posedge clk or posedge rst )
        if (rst) begin
            result_temp <= 20'd0;
            result <= 20'd0;
            count <= 2'b00;
        end

        else begin
            result_temp <= result_temp + sum[1];
            count <= count + 1;
            if (count == 2'b10) begin
                result <= result_temp + sum[1];
                result_temp <= 20'd0;
                count <= 2'b00;
            end
        end
endmodule
```

Listing 7: MAC_tb.v

```
`timescale 1ns / 1ps
module mac_tb();
reg clk, rst;
reg signed [23:0] data,weight;
wire signed [19:0] result;
MAC uut(.clk(clk), .rst(rst), .data(data), .weight(weight),
    .result(result));
initial
begin
        rst = 1;  #1
        clk=0;
        rst=0;
        data = 24'b00000000_0000100_00000000; // 0 4 0
        weight = 24'b11111111_11111111_11111111;
        #9
        data = 24'b00000001_00001001_00000000; // 1 9 0
        weight = 24'b11111111_00001000_11111111;
        #9
        data = 24'b00000001_00000000_00001000; // 108
        weight = 24'b11111111_11111111_11111111;
        #20;
        $finish;
end
```

```
always
begin
    clk<=~clk;
    #5;
end
endmodule
```

Maximum delay is in the setup delays and its 16.094 ns. Using this information and knowledge of T = 1/f, maximum achievable clock frequency is 62.13Mhz.

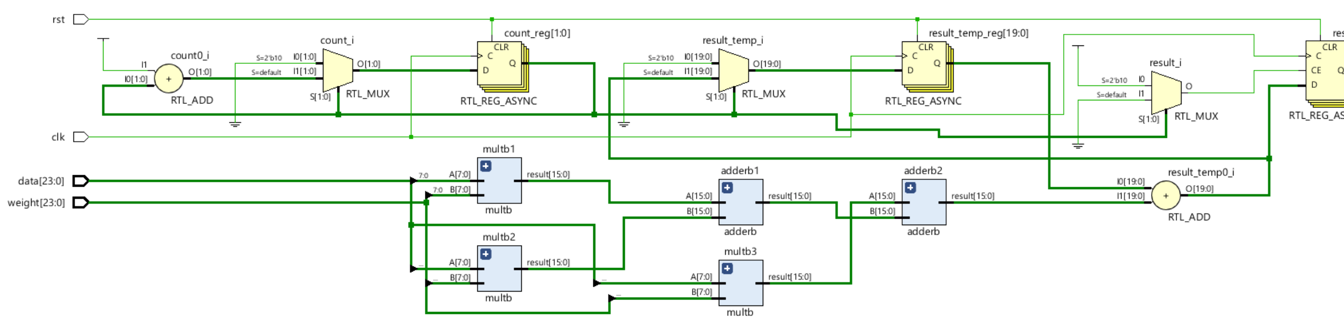


Figure 18: Multiply and Accumulate (MAC) Module Rtl Schematic

4.1 Synthesis and Implementation Part

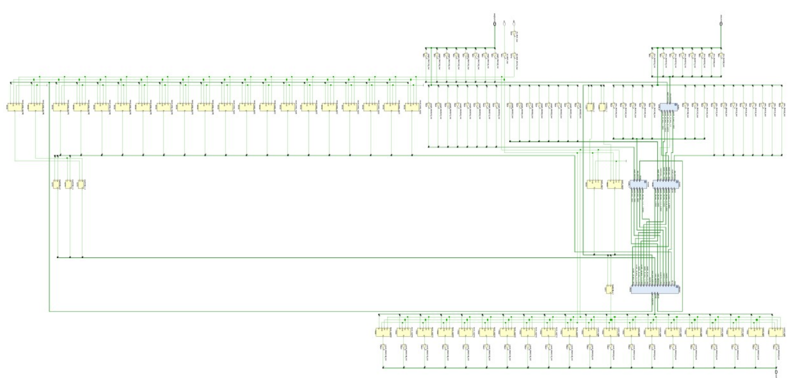


Figure 19: Multiply and Accumulate (MAC) Module Technolgy Schematic

Summary

Resource	Utilization	Available	Utilization %
LUT	247	63400	0.39
FF	42	126800	0.03
IO	70	210	33.33

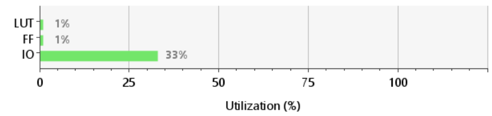


Figure 20: Multiply and Accumulate (MAC) Module Utilization Report

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination
Path 1	∞	17	11	19	data[3]	result_temp_reg[18]/D	16.094	6.471	9.623	∞	input port clock	
Path 2	∞	17	11	19	data[3]	result_temp_reg[16]/D	15.971	6.441	9.530	∞	input port clock	
Path 3	∞	17	11	19	data[3]	result_temp_reg[19]/D	15.911	6.580	9.331	∞	input port clock	
Path 4	∞	17	11	19	data[3]	result_temp_reg[17]/D	15.834	6.583	9.252	∞	input port clock	
Path 5	∞	16	10	19	data[3]	result_temp_reg[15]/D	15.527	6.373	9.154	∞	input port clock	
Path 6	∞	16	10	19	data[3]	result_temp_reg[12]/D	15.515	6.005	9.510	∞	input port clock	
Path 7	∞	16	10	19	data[3]	result_temp_reg[13]/D	15.379	6.147	9.232	∞	input port clock	
Path 8	∞	16	10	19	data[3]	result_temp_reg[14]/D	15.271	6.275	8.997	∞	input port clock	
Path 9	∞	15	9	19	data[3]	result_temp_reg[11]/D	15.004	5.692	9.312	∞	input port clock	
Path 10	∞	16	11	19	data[3]	result_reg[17]/D	14.923	6.251	8.673	∞	input port clock	

Figure 21: Multiply and Accumulate (MAC) Module Timing Report: Setup

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 11	∞	3	2	2	result_temp_reg[18]/C	result_reg[19]/D	0.313	0.250	0.063	-∞	
Path 12	∞	3	2	2	result_temp_reg[14]/C	result_reg[14]/D	0.316	0.251	0.065	-∞	
Path 13	∞	3	2	2	result_temp_reg[2]/C	result_reg[2]/D	0.316	0.251	0.065	-∞	
Path 14	∞	3	2	2	result_temp_reg[6]/C	result_reg[6]/D	0.316	0.251	0.065	-∞	
Path 15	∞	3	2	2	result_temp_reg[18]/C	result_reg[18]/D	0.316	0.251	0.065	-∞	
Path 16	∞	3	2	2	result_temp_reg[14]/C	result_reg[15]/D	0.352	0.287	0.065	-∞	
Path 17	∞	3	2	2	result_temp_reg[2]/C	result_reg[3]/D	0.352	0.287	0.065	-∞	
Path 18	∞	3	2	2	result_temp_reg[6]/C	result_reg[7]/D	0.352	0.287	0.065	-∞	
Path 19	∞	3	2	2	result_temp_reg[0]/C	result_reg[0]/D	0.354	0.256	0.098	-∞	
Path 20	∞	3	2	2	result_temp_reg[12]/C	result_reg[12]/D	0.354	0.256	0.098	-∞	

Figure 22: Multiply and Accumulate (MAC) Module Timing Report: Hold

5 2D Convolution

5.1 Explanation of 2D Convolution

Generally, 1D convolution is used in speech processing, 2D convolution is in image processing, and 3D convolution is commonly used in video processing. 2D convolution can be used to define images' edges or remove noise. The 2D convolution process is done as follows: The first element of the kernel matrix is placed in the first element of the image matrix. In other words, each element of the kernel matrix rests on an element on the image matrix. Next, each element of the kernel matrix is multiplied by its corresponding (i.e. overlapping) element in the image matrix. The values obtained as a result of the multiplications are collected and placed in the same location, which is the center of the kernel, in the image matrix in the output matrix.

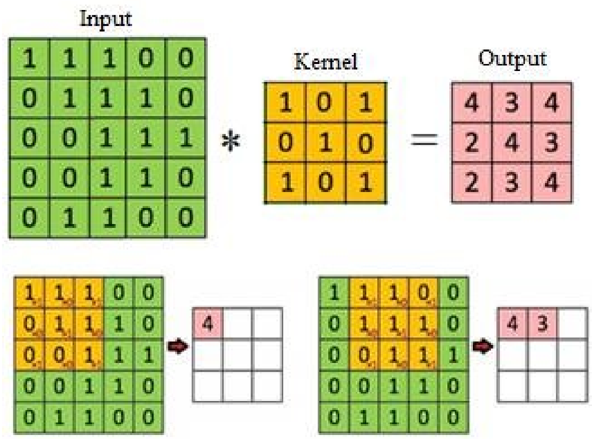


Figure 23: Explanation of 2D Convolution

Listing 8: conv.v

```
module conv(
input clk, rst,
  input signed [23:0] data,
  input signed [23:0] weight,
  output signed [19:0] result
);
MAC mac1 (.clk(clk), .rst(rst), .data(data), .weight(weight),
    .result(result));

endmodule
```

Listing 9: conv_tb.v

```
`timescale 1ns / 1ps
`timescale 1ns / 1ps

module conv_tb();
reg clk, rst;
reg signed [23:0]
    data1,data2,data3,data4,data5,data6,data7,data8,data9 ;
reg signed [23:0] weight ;
reg signed [23:0] weight1 ;
reg signed [23:0] weight2 ;
reg signed [23:0] weight3 ;
wire signed [19:0]
    result_1,result_2,result_3,result_4,result_5,result_6,result_7,result_8,resul
```

```
conv uut1(.clk(clk), .rst(rst), .data(data1),.weight(weight),
    .result(result_1));
conv uut2 (.clk(clk), .rst(rst), .data(data2),.weight(weight),
    .result(result_2));
conv uut3 (.clk(clk), .rst(rst), .data(data3),
    .weight(weight), .result(result_3));

conv uut4(.clk(clk), .rst(rst), .data(data4),.weight(weight),
    .result(result_4));
conv uut5 (.clk(clk), .rst(rst), .data(data5),.weight(weight),
    .result(result_5));
conv uut6 (.clk(clk), .rst(rst), .data(data6),
    .weight(weight), .result(result_6));


conv uut7(.clk(clk), .rst(rst), .data(data7),.weight(weight),
    .result(result_7));
conv uut8 (.clk(clk), .rst(rst), .data(data8),.weight(weight),
    .result(result_8));
conv uut9 (.clk(clk), .rst(rst), .data(data9),
    .weight(weight), .result(result_9));
initial
begin
        clk=0;
        rst = 1;  #1
        clk = 0;
        rst=0;
        weight1 = 24'b11111111_11111111_11111111;
        weight2 = 24'b11111111_00001000_11111111;
        weight3 = 24'b11111111_11111111_11111111;

        data1=24'b10000000_10000000_10000000;
        data2=24'b10000000_10000000_10000000;
        data3=24'b10000000_10000000_10000000;
        data4=24'b11111111_11111111_10000000;
        data5=24'b11111111_10000000_11111111;
        data6=24'b10000000_11111111_11111111;
        data7=24'b11111111_11111111_10000000;
        data8=24'b11111111_10000000_11111111;
        data9=24'b10000000_11111111_11111111;
        weight= weight1; #10;


        data1=24'b11111111_11111111_10000000;
        data2=24'b11111111_10000000_11111111;
```

```
        data3=24'b10000000_11111111_11111111;
        data4=24'b11111111_11111111_10000000;
        data5=24'b11111111_10000000_11111111;
        data6=24'b10000000_11111111_11111111;
        data7=24'b11111111_11111111_10000000;
        data8=24'b11111111_10000000_11111111;
        data9=24'b10000000_11111111_11111111;
        weight= weight2; #10;

        data1=24'b11111111_11111111_10000000;
        data2=24'b11111111_10000000_11111111;
        data3=24'b10000000_11111111_11111111;
        data4=24'b11111111_11111111_10000000;
        data5=24'b11111111_10000000_11111111;
        data6=24'b10000000_11111111_11111111;
        data7=24'b11111111_11111111_10000000;
        data8=24'b11111111_10000000_11111111;
        data9=24'b10000000_11111111_11111111;
        weight= weight3; #10;
        $write("Results \n%d %d %d \n%d %d %d \n%d %d %d
           ",result_1,result_2,result_3,result_4,result_5,
        #20;
        $finish;
end
always
begin
    clk<=~clk;
    #5;
end
endmodule
```

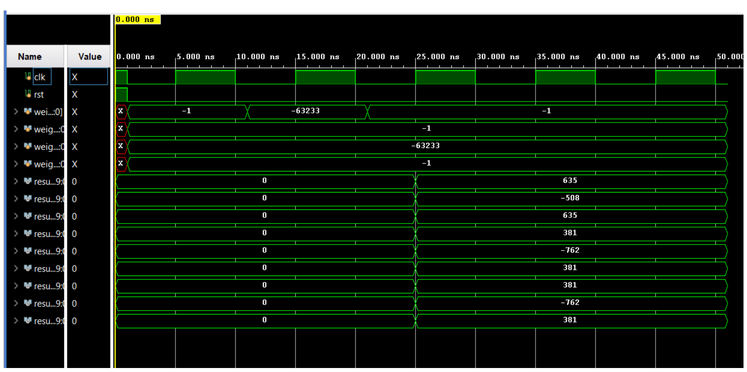


Figure 24: Conv Module Simulation Result

```
Results
    635    -508    635
```

```
     381     -762      381
     381     -762      381
$finish called at time : 51 ns : File
```

Compared to the given matlab codes, MATLAB result and my code's output are same. So, the design is implemented correctly.

6 Appendix

6.1 CLA and Behavioral Adder Modules

Listing 10: conv_tb.v

```
    module CLA (

  input [15:0] X,
  input [15:0] Y,
  input  C_i,
  output C_o,
  output [15:0] S
);
  wire [15:0] g_wires;
  wire [15:0] p_wires;
  wire [16:0] c_wires;

  assign c_wires[0] = C_i;
  assign C_o = c_wires[16];

  genvar j;
  generate
    for (j = 0; j < 16; j = j + 1) begin : GandP_loop
      assign g_wires[j] = X[j] & Y[j];
      assign p_wires[j] = X[j] ^ Y[j];
      assign S[j] = p_wires[j] ^ c_wires[j];
      assign c_wires[j + 1] = g_wires[j] | (p_wires[j] &
          c_wires[j]);
    end
  endgenerate
endmodule

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

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
```

6.2 References

References

[1] Ramaiyan, Abinaya; Dnvsls, Indira; Lanka, Dhanalakshmi. *Acoustic based Scene Event Identification Using Deep Learning CNN. Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12, 1398-1405 (May 2021).