DIGITAL SYSTEM DESIGN APPLICATION – EHB 436E

Experiment VII

Yiğit Bektaş GÜRSOY

040180063

Class Lecturer: Sıddıka Berna Örs Yalçın

Class Assistant:
Serdar Duran
Yasin Fırat Kula
Mehmet Onur Demirtürk

Yiğit Bektaş GÜRSOY
040180063

1. STRUCTURAL MULTIPLIER - UNSIGNED

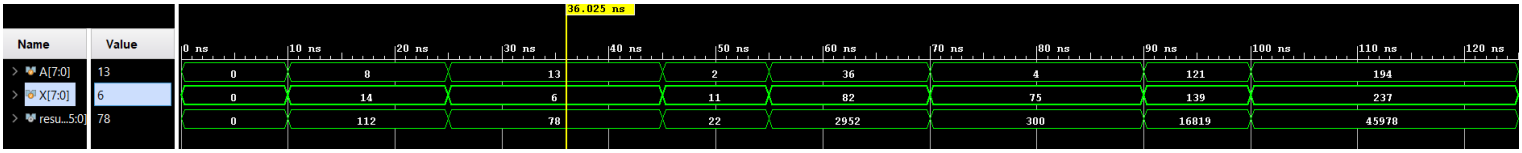
- Verilog Code

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

    wire [15:0]PP[7:0];

    genvar i;
    generate
        for(i = 0; i <= 7; i = i + 1)
        begin: gen_PP //PARTIAL PRODUCT
            assign PP[i][15:0] = (X[i] * A) << i;
        end
    endgenerate
    wire [15:0] sum [6:0];
    wire cout [6:0];

    // sum of partial product
    CLA CLA1(PP[0][15:0], PP[1][15:0], 0, cout[0], sum[0][15:0]);
    CLA CLA2(PP[2][15:0], PP[3][15:0], 0, cout[1], sum[1][15:0]);
    CLA CLA3(PP[4][15:0], PP[5][15:0], 0, cout[2], sum[2][15:0]);
    CLA CLA4(PP[6][15:0], PP[7][15:0], 0, cout[3], sum[3][15:0]);

    // sum of result of partial product
    CLA CLA5(sum[0][15:0], sum[1][15:0], 0, cout[4], sum[4][15:0]);
    CLA CLA6(sum[2][15:0], sum[3][15:0], 0, cout[5], sum[5][15:0]);
    CLA CLA7(sum[4][15:0], sum[5][15:0], 0, cout[6], sum[6][15:0]);
    assign result[15:0] = sum[6][15:0];
endmodule
```

- Test Bench Code

```
module MULTS_tb();
    reg [7:0] A;
    reg [7:0] X;
    wire [15:0] result;

    MULTS DUT
    (
    .A(A),
    .X(X),
    .result(result)
    );

    initial
    begin
        A=0; X=0;
        #10;
        $write("A * X = %d * %d => Result = %d\n",A, X, result);

        A=8; X=14;
        #15
        $write("A * X = %d * %d => Result = %d\n",A, X, result);

        A=13; X=6;
        #20
        $write("A * X = %d * %d => Result = %d\n",A, X, result);

        A=2; X=11;
        #10
        $write("A * X = %d * %d => Result = %d\n",A, X, result);

        A=36; X=82;
        #15
        $write("A * X = %d * %d => Result = %d\n",A, X, result);

        A=4; X=75;
        #20
        $write("A * X = %d * %d => Result = %d\n",A, X, result);

        A=121; X=139;
        #10
        $write("A * X = %d * %d => Result = %d\n",A, X, result);


        A=194; X=237;
        #15
        $write("A * X = %d * %d => Result = %d\n", A, X, result);
        #10
        $finish;
    end
endmodule
```

- Behavioral Simulation Result



- TCL Console Output

```
A * X =   0 *   0 => Result =     0
A * X =   8 *  14 => Result =   112
A * X =  13 *   6 => Result =    78
A * X =   2 *  11 => Result =    22
A * X =  36 *  82 => Result =  2952
A * X =   4 *  75 => Result =   300
A * X = 121 * 139 => Result = 16819
A * X = 194 * 237 => Result = 45978
$finish called at time : 125 ns : File "C:/Users/yigit/Desktop/Okul/2022guz/Sayisal Sistem Tasarim Uygulamalari/sstu_odevler/Experiment7/sstu_odev7/sstu_odev7.srcs/si
INFO: [USF-XSim-96] XSim completed. Design snapshot 'MULTS_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:05 . Memory (MB): peak = 949.609 ; gain = 0.000
```

2. STRUCTURAL MULTIPLIER – SIGNED

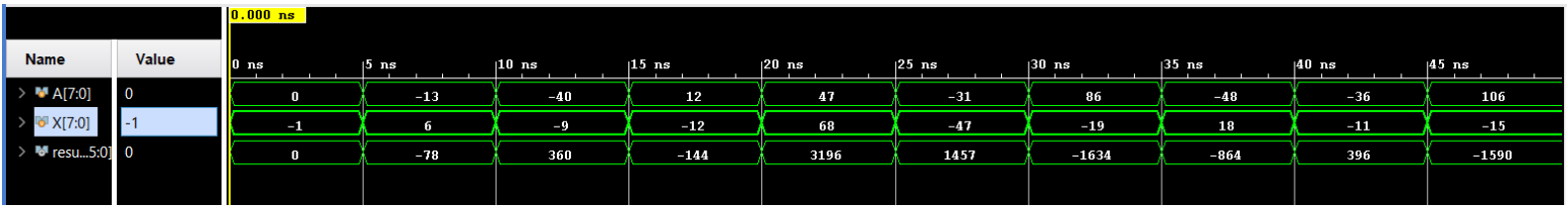
- Verilog Code

```
module MULTS_signed
    (
    input [7:0] A,
    input [7:0] X,
    output [15:0] result
    );
    wire [7:0] PP [7:0];
    wire [7:0] PP_BW [7:0];
    wire [15:0] PP_shift [7:0];
    genvar i;
    generate
        for(i = 0; i <= 7; i = i + 1)
        begin: PP_LOOP
            assign PP[i][7:0] = (X[i] * A);
        end
    endgenerate
    genvar z, t;
    generate
        for(z = 0; z <= 7; z = z + 1)
        begin: PP_BW_LOOP
            for(t = 0; t <= 7; t = t + 1)
            begin
                if(z != 7)
                begin
                    if(t != 7)
                        assign PP_BW[z][t] = PP[z][t];
                    else
                        assign PP_BW[z][t] = ~PP[z][t];
                end
            else
            begin
                if(t == 7)
                    assign PP_BW[z][t] = PP[z][t];
                else
                    assign PP_BW[z][t] = ~PP[z][t];
                end
            end
        end
    endgenerate

    genvar j;
    generate
    for(j = 0; j <= 7; j = j + 1)
    begin: PP_SHIFT_LOOP
        assign PP_shift[j][15:0] = PP_BW[j] << j;
    end
    endgenerate
    wire [15:0] sum [7:0];
    wire cout [7:0];

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

➢ Test Bench Code

```
module MULTS_signed_tb();
    reg signed [7:0] A;
    reg signed [7:0] X;
    wire signed [15:0] result;

    MULTS_signed DUT
    (
    .A(A),
    .X(X),
    .result(result)
    );

    initial
    begin
    A=0; X=-1;
    #5;
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=-13; X=6;
    #50
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=-40; X=-9;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=12; X=-12;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=47; X=68;
    #50
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=-31; X=-47;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=342; X=-19;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=-48; X=18;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=-36; X=-11;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);

    A=-150; X=-15;
    #5
    $write("A * X = %d * %d => Result = %d\n",A, X, result);
    $finish;
    end

endmodule
```

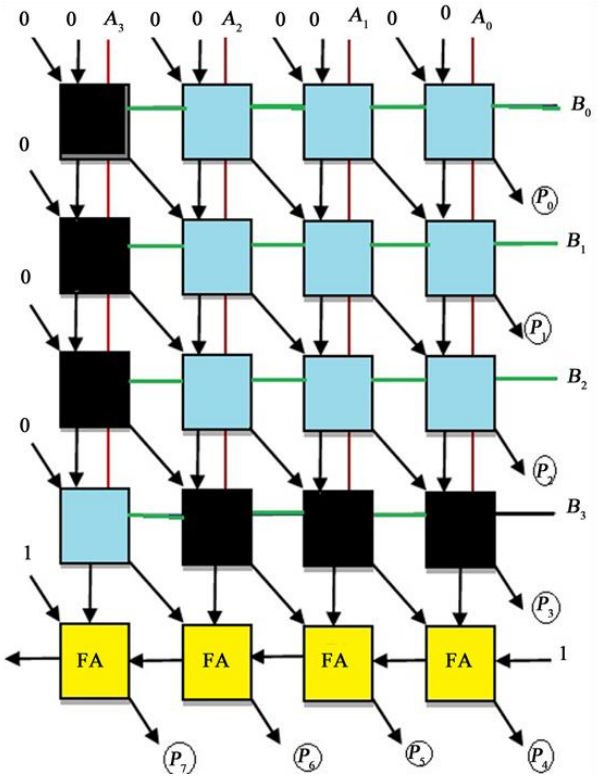
- Behavioral Simulation Result



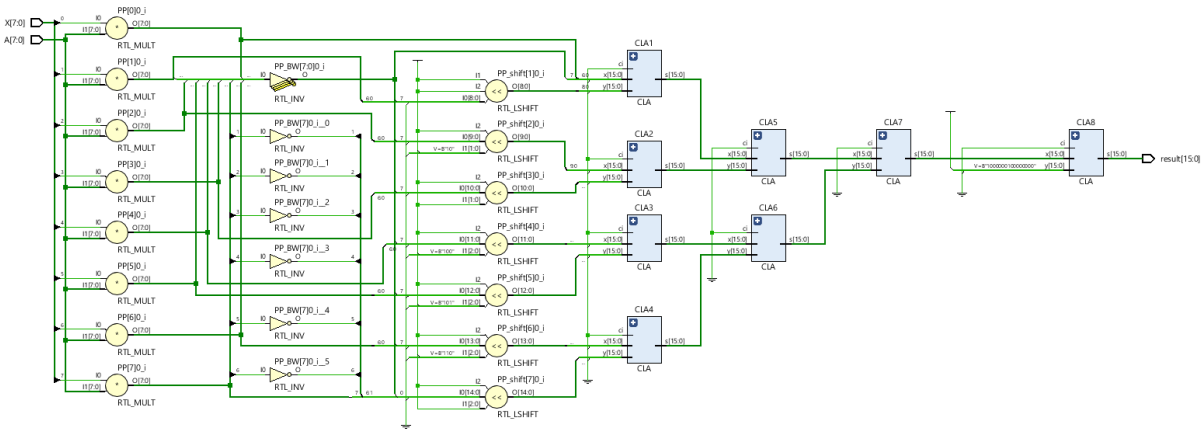
- TCL Console Output

```
A * X =    0 *   -1 => Result =      0
A * X =  -13 *    6 => Result =    -78
A * X =  -40 *   -9 => Result =    360
A * X =   12 *  -12 => Result =   -144
A * X =   47 *   68 => Result =   3196
A * X =  -31 *  -47 => Result =   1457
A * X =   86 *  -19 => Result =  -1634
A * X =  -48 *   18 => Result =   -864
A * X =  -36 *  -11 => Result =    396
A * X =  106 *  -15 => Result =  -1590
$finish called at time : 50 ns : File "C:/Users/yigit/Desktop/Okul/2022guz/Sayisal Sistem Tasarim Uygulamalari/sstu_odevler/Experiment7/sstu_odev7/sstu_odev7.srcs/sim
INFO: [USF-XSim-96] XSim completed. Design snapshot 'MULTS_signed_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

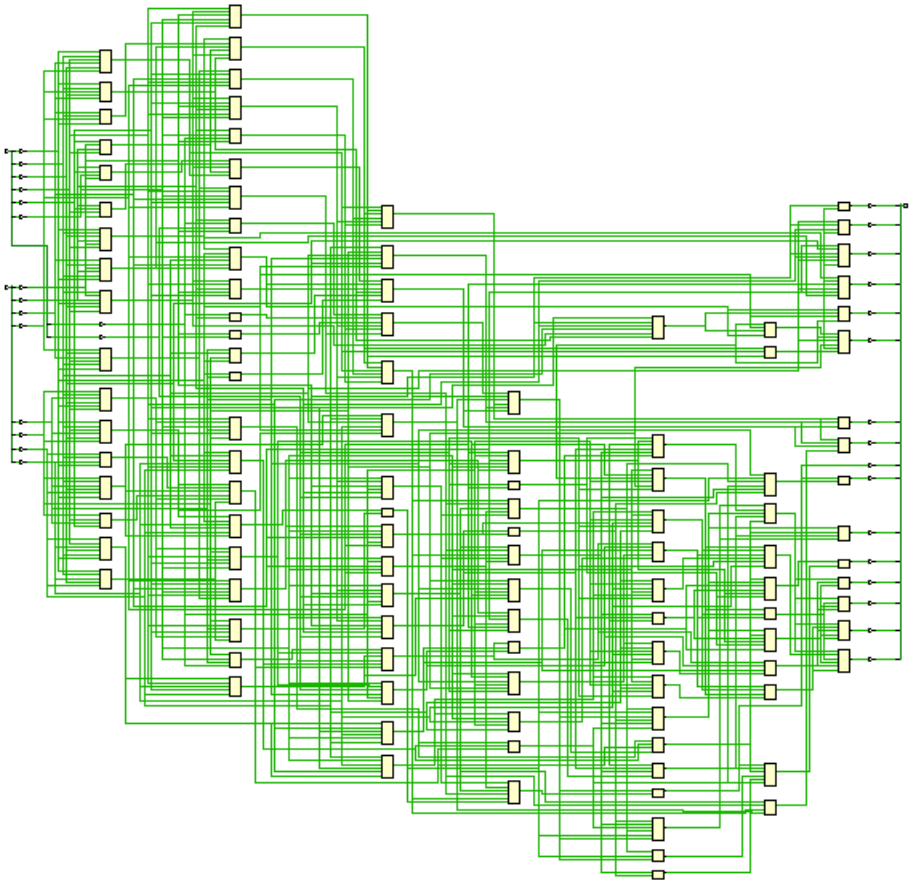
- Number of Stages
  - As a result of two 8-bit operations, a 16-bit number and 8 partial multiplication stages are obtained in partial multiplication in the last stage. As a result of these data, 4 collection sections and 8 CLAs are needed.
- Baugh – Wooley Method
  - The Baugh-Wooley method is an algorithm used to multiply 2 numbers, these numbers can be negative or positive. Multiplication is performed by taking not the most valuable bit of the partial products other than the last partial product. In the last partial multiplication, all other bits except the most significant bit are taken. In the photo below, this process is explained with an example. Multiplication is done in each of the boxes. The blue parts are the part where the products are the same, the black parts are the parts where the products become its compliment. Then these found numbers are summed as follows and written in order from the most significant bit to the least significant bit.



- Technology Schematic



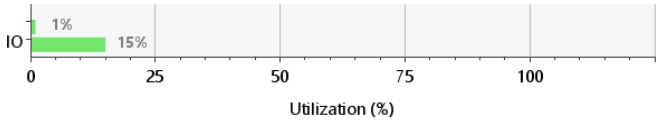
- Technology Schematic



- Utilization Report

Summary

Resource	Utilization	Available	Utilization %
LUT	91	32600	0.28
IO	32	210	15.24



- Timing Summary

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	result[14]	14.483	SLOW	3.763	FAST
A[3]	result[14]	14.310	SLOW	3.521	FAST
A[0]	result[13]	14.268	SLOW	3.714	FAST
A[0]	result[15]	14.261	SLOW	3.417	FAST
A[2]	result[14]	14.253	SLOW	3.571	FAST
A[0]	result[11]	14.252	SLOW	3.556	FAST
A[1]	result[14]	14.226	SLOW	3.633	FAST
A[4]	result[14]	14.221	SLOW	3.538	FAST
A[0]	result[12]	14.193	SLOW	3.688	FAST
X[0]	result[14]	14.191	SLOW	3.664	FAST
A[3]	result[13]	14.096	SLOW	3.517	FAST
A[3]	result[15]	14.088	SLOW	3.336	FAST
A[3]	result[11]	14.080	SLOW	3.381	FAST
A[2]	result[13]	14.038	SLOW	3.526	FAST
A[2]	result[15]	14.031	SLOW	3.275	FAST
A[2]	result[11]	14.022	SLOW	3.458	FAST
A[3]	result[12]	14.020	SLOW	3.505	FAST
A[1]	result[13]	14.011	SLOW	3.629	FAST
X[1]	result[14]	14.009	SLOW	3.518	FAST
A[4]	result[13]	14.006	SLOW	3.490	FAST
A[1]	result[15]	14.004	SLOW	3.447	FAST
X[3]	result[14]	14.003	SLOW	3.190	FAST
A[4]	result[15]	13.999	SLOW	3.186	FAST

➢ Combinational delays are indicated in the table above. The maximum delay of the designed circuit was 14.48 ns.

3. BEHAVIORAL MULTIPLIER

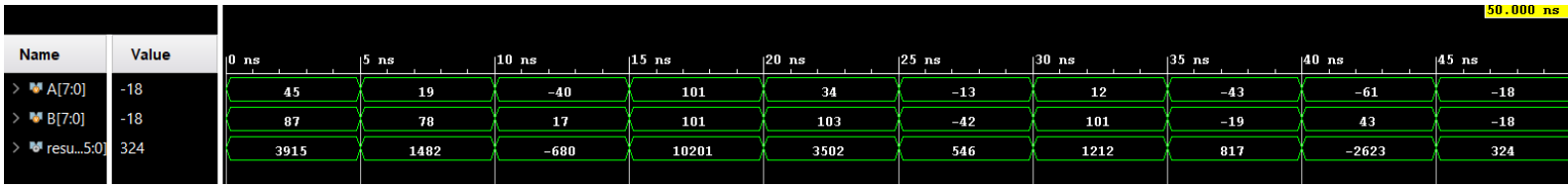
- Verilog Code

```
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
```

- Test Bench Code

```
module MULTB_tb();
    reg signed [7:0] A;
    reg signed [7:0] B;
    wire signed [15:0] result;

    MULTB DUT
    (
    .A(A),
    .B(B),
    .result(result)
    );

    initial
    begin
        A=45; B=87;
        #5;
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=19; B=78;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=-40; B=17;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=101; B=101;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=34; B=103;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=-13; B=-42;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=12; B=-155;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=-43; B=-19;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=-61; B=43;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);

        A=-18; B=-18;
        #5
        $write("A * B = %d * %d => Result = %d\n",A, B, result);
        $finish;
        end
endmodule
```

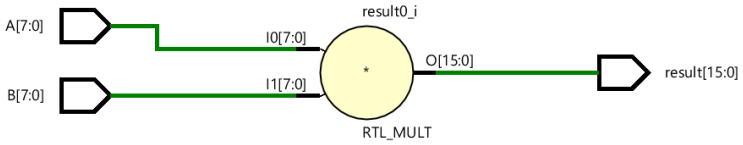
- Behavioral Simulation



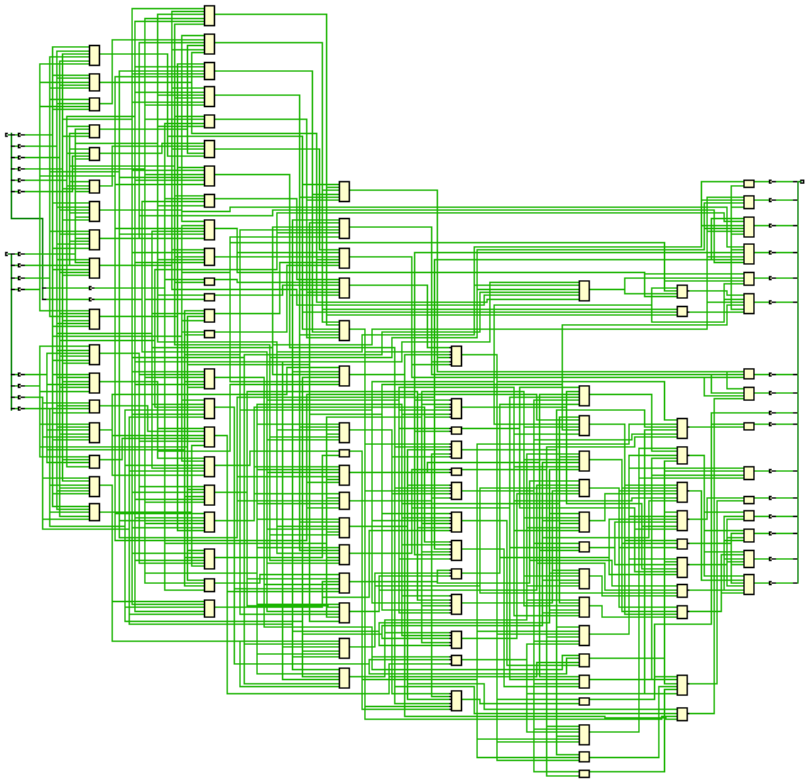
- TCL Console

```
A * B =   45 *   87 => Result =   3915
A * B =   19 *   78 => Result =   1482
A * B =  -40 *   17 => Result =   -680
A * B =  101 *  101 => Result =  10201
A * B =   34 *  103 => Result =   3502
A * B =  -13 *  -42 => Result =    546
A * B =   12 *  101 => Result =   1212
A * B =  -43 *  -19 => Result =    817
A * B =  -61 *   43 => Result =  -2623
A * B =  -18 *  -18 => Result =    324
$finish called at time : 50 ns : File "C:/Users/yigit/Desktop/Okul/2022guz/Sayisal Sistem Tasarim Uygulamalari/sstu_odevler/Experiment7/sstu_odev7/sstu_odev7.srcs/sim
INFO: [USF-XSim-96] XSim completed. Design snapshot 'MULTB_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

- RTL Schematic



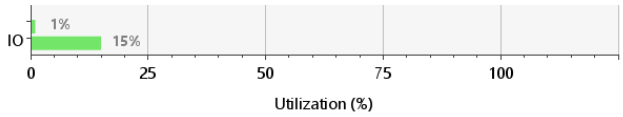
- Technology Schematic



- Utilization Report

Summary

Resource	Utilization	Available	Utilization %
LUT	61	32600	0.19
IO	32	210	15.24



- Timing Summary

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
B[5]	result[15]	13.172	SLOW	3.029	FAST
B[5]	result[14]	13.142	SLOW	2.921	FAST
A[2]	result[15]	13.107	SLOW	3.204	FAST
A[5]	result[15]	13.091	SLOW	3.183	FAST
A[2]	result[14]	13.078	SLOW	3.194	FAST
B[2]	result[15]	13.070	SLOW	3.206	FAST
A[5]	result[14]	13.061	SLOW	3.173	FAST
B[2]	result[14]	13.041	SLOW	3.196	FAST
B[5]	result[12]	13.028	SLOW	2.988	FAST
A[2]	result[12]	12.964	SLOW	3.155	FAST
A[5]	result[12]	12.948	SLOW	3.092	FAST
B[5]	result[13]	12.931	SLOW	2.876	FAST
B[5]	result[11]	12.929	SLOW	2.970	FAST
B[2]	result[12]	12.927	SLOW	3.157	FAST
A[3]	result[15]	12.874	SLOW	3.326	FAST
A[2]	result[13]	12.867	SLOW	3.113	FAST
A[2]	result[11]	12.865	SLOW	3.134	FAST
A[5]	result[13]	12.850	SLOW	3.092	FAST
A[5]	result[11]	12.848	SLOW	3.081	FAST
A[3]	result[14]	12.845	SLOW	3.284	FAST
B[2]	result[13]	12.830	SLOW	3.115	FAST
B[2]	result[11]	12.828	SLOW	3.137	FAST
A[3]	result[12]	12.731	SLOW	3.194	FAST

  - Combinational delays are indicated in the table above. The maximum delay of the designed circuit was 13.17ns.
  - Comparing structural and behavioral designs, behavioral design appears to be more efficient in terms of delay and space. 91 LUTs are used in the structural structure and 61 LUTs are used in behavioral design. When the delays are compared, it seems to have a maximum delay of 13.17 ns in the structural 14.48 ns behavioral design.

4. MULTIPLY AND ACCUMULATE (MAC)

- Verilog Code

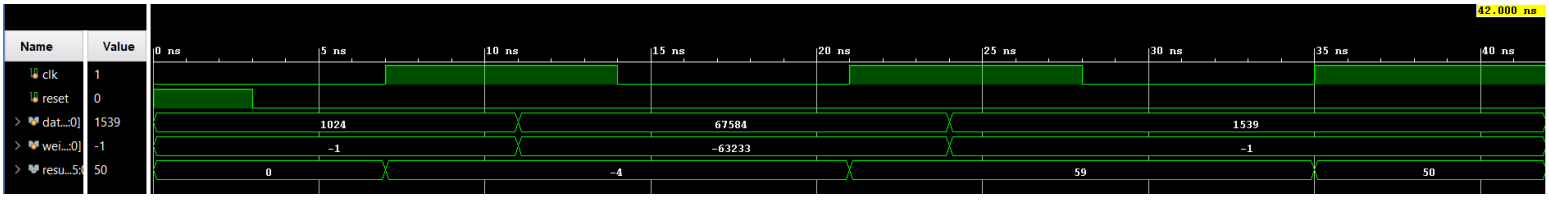
```
module MAC
    (
    input clk,
    input reset,
    input signed [23:0] data,
    input signed [23:0] weight,
    output reg signed [15:0] result
    );

    wire signed [15:0] product [2:0];
    wire signed [15:0] sum [1:0];
    reg [1:0] count;

    MULTB MULT0(data[7:0], weight[7:0], product[0][15:0]);

    MULTB MULT1(data[15:8], weight[15:8], product[1][15:0]);

    MULTB MULT2(data[23:16], weight[23:16], product[2][15:0]);

    Behav_Adder ADD1(product[0][15:0], product[1][15:0],
sum[0][15:0]);

    Behav_Adder ADD2(product[2][15:0], sum[0][15:0],
sum[1][15:0]);

    always @ (posedge clk or posedge reset)
    begin
        if(reset)
        begin
            count <= 0;
            result <= 0;
        end
        else
        begin
            result <= result + sum[1][15:0];
            count <= count + 1;
        end
    end
endmodule
```

- Testbench Code

```
module MAC_tb();
    reg clk, reset;
    reg signed [23:0] data;
    reg signed [23:0] weight;
    wire signed [15:0] result;

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

    initial
    begin
        clk = 0;
        reset = 1;
        data = 24'b00000000_0000100_00000000; // 0 4 0
        weight = 24'b11111111_11111111_11111111;
        #12;
        reset = 0;

        #9;
        data = 24'b00000001_00001000_00000000; // 1 8 0
        weight = 24'b11111111_00001000_11111111;

        #14;
        data = 24'b00000000_00000101_00000011; // 0 6 3
        weight = 24'b11111111_11111111_11111111;
        #5;

        $write("Dataset = [0 4 0; 1 8 0; 0 6 3], Weights = [-1 -1 -1; -1 8 -1; -1 -1 -1]\n");
        $write("R11 = D11 * W11, R12 = D12 * W12, R13 = D13 * W13\n");
        $write("R21 = D21 * W21, R22 = D22 * W22, R23 = D23 * W23\n");
        $write("R31 = D31 * W31, R32 = D32 * W32, R33 = D33 * W33\n");
        $write("Result Matrix = [0 -4 0; -1 64 0; 0 -6 -3]\n");
        $write("Result = R11 + R12 + R13 + R21 + R22 + R23 + R31 + R32 + R33\n");
        $write("Result = %d\n", result);
        $finish;
    end

    always
    begin
        #10; clk = ~clk;
    end
endmodule
```

- Behavioral Simulation

➢ According to the code written, when the posedge signal of the clock comes, the product of each line is calculated. Since we have a 3*3 matrix, the matrix multiplication will be calculated in the case where reset=0 and posedge comes 3 times in a row. The following is the description of the simulation.

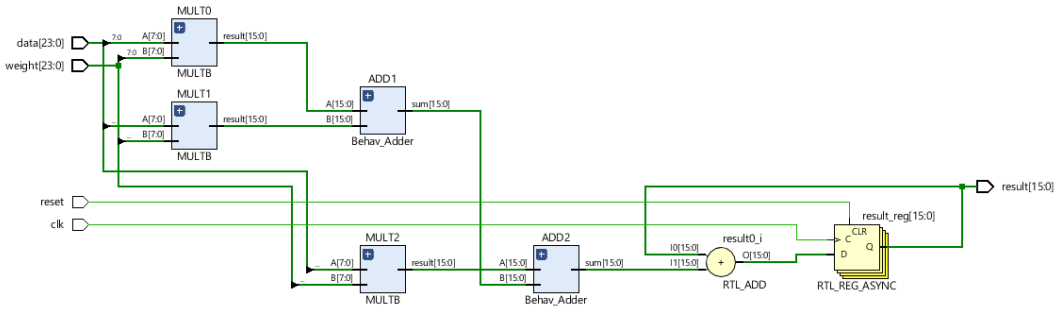


- TCL Console and MATLAB console output for verification

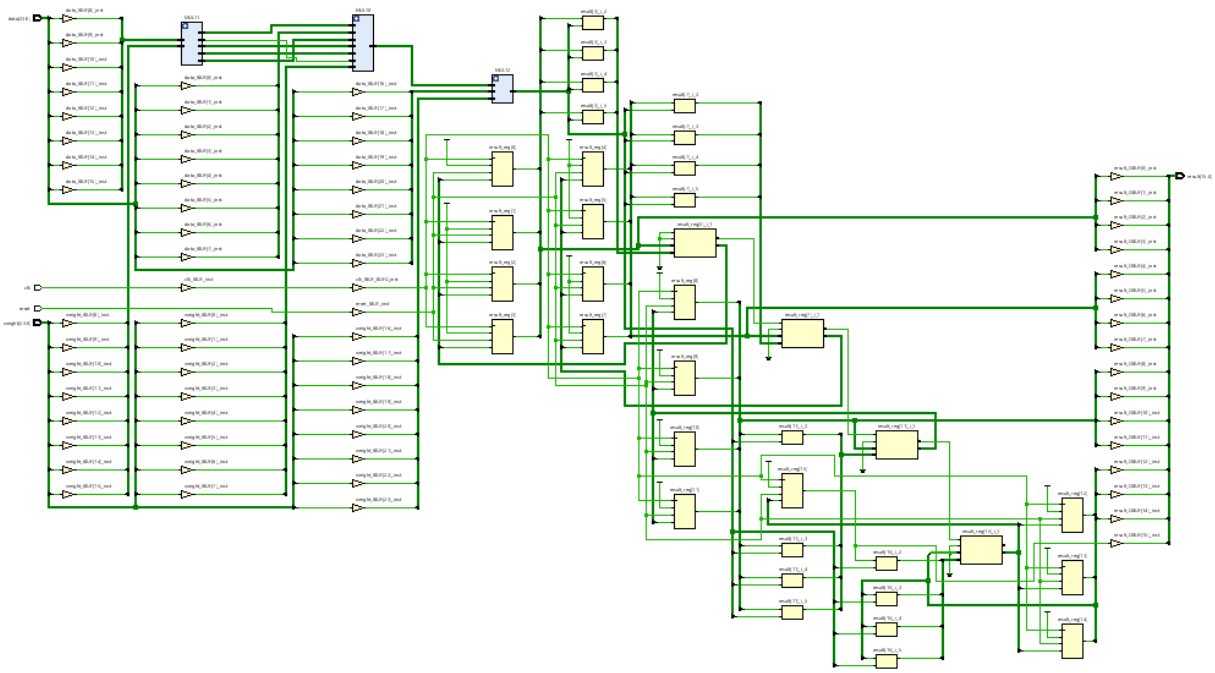
```
Dataset = [0 4 0; 1 8 0; 0 6 3], Weights = [-1 -1 -1; -1 8 -1; -1 -1 -1]
R11 = D11 * W11, R12 = D12 * W12, R13 = D13 * W13
R21 = D21 * W21, R22 = D22 * W22, R23 = D23 * W23
R31 = D31 * W31, R32 = D32 * W32, R33 = D33 * W33
Result Matrix = [0 -4 0; -1 64 0; 0 -6 -3]
Result = R11 + R12 + R13 + R21 + R22 + R23 + R31 + R32 + R33
Result =     50
$finish called at time : 42 ns : File "C:/Users/yigit/Desktop/Okul/2022guz/Sayisal Sistem Tasarim Uygulamalari/sstu_odevler/Experiment7/sstu_odev7/sstu_odev7.srcs/sim
INFO: [USF-XSim-96] XSim completed. Design snapshot 'MAC_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```

```
>> image

image =

     0     4     0
     1     8     0
     0     6     3

>> kernel

kernel =

    -1    -1    -1
    -1     8    -1
    -1    -1    -1

>> result

result =

    50
```

- RTL Schematic



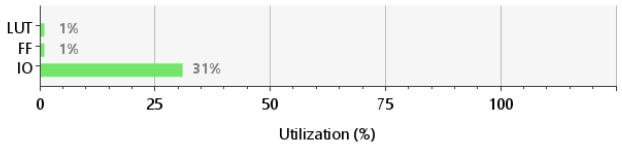
- Technology Schematic



- Utilization Report

Summary

Resource	Utilization	Available	Utilization %
LUT	231	32600	0.71
FF	16	65200	0.02
IO	66	210	31.43



- Timing Summary
  - Setup Delays

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 10	∞	13	7	19	data[11]	result_reg[6]/D	12.052	4.852	7.200	∞	input port clock
Path 9	∞	13	7	19	data[11]	result_reg[7]/D	12.112	4.912	7.200	∞	input port clock
Path 8	∞	16	10	19	data[11]	result_reg[8]/D	12.680	5.439	7.241	∞	input port clock
Path 7	∞	16	10	19	data[11]	result_reg[9]/D	12.857	5.616	7.241	∞	input port clock
Path 6	∞	16	10	19	data[11]	result_reg[10]/D	13.389	5.892	7.497	∞	input port clock
Path 5	∞	16	10	19	data[11]	result_reg[11]/D	13.493	5.996	7.497	∞	input port clock
Path 4	∞	17	11	19	data[11]	result_reg[12]/D	13.762	6.265	7.497	∞	input port clock
Path 3	∞	17	11	19	data[11]	result_reg[13]/D	13.873	6.376	7.497	∞	input port clock
Path 2	∞	14	8	17	data[12]	result_reg[14]/D	13.964	5.628	8.336	∞	input port clock
Path 1	∞	14	8	17	data[12]	result_reg[15]/D	14.068	5.732	8.336	∞	input port clock

  - Hold Delays

Unconstrained Paths - NONE - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement
Path 19	∞	2	1	3	result_reg[9]/C	result_reg[10]/D	0.368	0.287	0.081	-∞
Path 20	∞	2	1	3	result_reg[1]/C	result_reg[2]/D	0.368	0.287	0.081	-∞
Path 16	∞	2	1	3	result_reg[12]/C	result_reg[13]/D	0.356	0.265	0.091	-∞
Path 17	∞	2	1	3	result_reg[4]/C	result_reg[5]/D	0.356	0.265	0.091	-∞
Path 18	∞	2	1	3	result_reg[8]/C	result_reg[9]/D	0.356	0.265	0.091	-∞
Path 15	∞	2	1	3	result_reg[0]/C	result_reg[1]/D	0.354	0.265	0.089	-∞
Path 11	∞	2	1	3	result_reg[10]/C	result_reg[11]/D	0.352	0.268	0.084	-∞
Path 12	∞	2	1	3	result_reg[14]/C	result_reg[15]/D	0.352	0.268	0.084	-∞
Path 13	∞	2	1	3	result_reg[2]/C	result_reg[3]/D	0.352	0.268	0.084	-∞
Path 14	∞	2	1	3	result_reg[6]/C	result_reg[7]/D	0.352	0.268	0.084	-∞

  - Maximum delay is 14.06ns. If the maximum clock frequency is calculated from f=1/T, the approximation is 1/14.06 = 73MHz. While doing simulation, we should pay attention to this value and determine the frequency accordingly. This value shows the maximum frequency value it can receive.

5. 2-D 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 the edges of images or to 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.

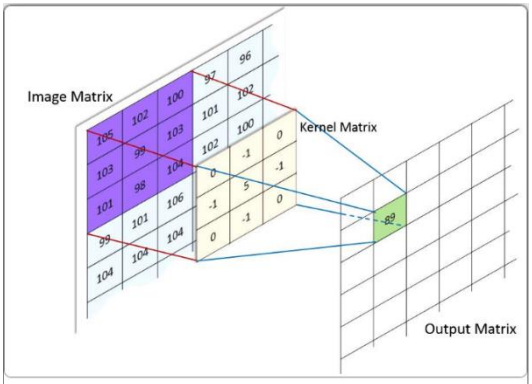


Figure 3: To calculate the value of convolution output at pixel (2,2), center the kernel at the same pixel position on the image matrix

In the picture above 105*0 + 102*(-1) + 100 * 0 + 103 *(-1) + 99 * 5 + 103 * (-1) + 101 * 0 + 98*(-1) + 104*0 = The value is 89 and placed in the center. Although the operation may seem complicated, it is essentially dot multiplication in vectors. That is,

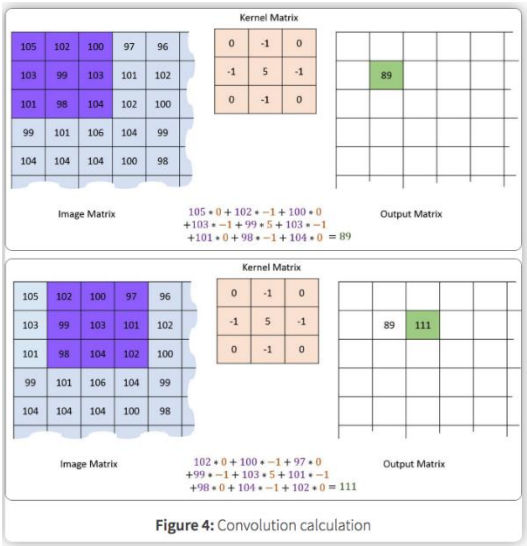


Figure 4: Convolution calculation

we say multiply the directly corresponding elements, add them together, assign that value to the central position. After this process is finished, the kernel matrix is shifted on the image matrix and the same process is repeated and the output matrix is filled. Note that no calculation can be made for the edges of the output matrix with this method. There are several suggested solutions to this situation. We can cancel the

edges and make the output matrix smaller. Depending on the kernel size we use, we can add padding to our input matrix, in the example above, we can apply 1-pixel padding (3x3 kernel size).

- Verilog Code

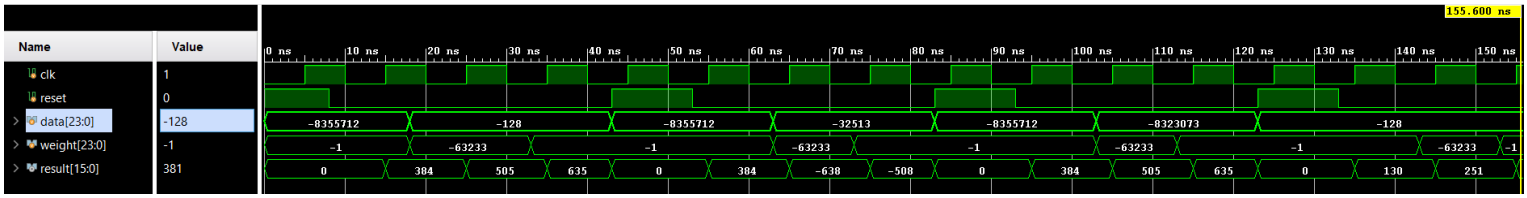
```
module Conv

    (
    input clk,
    input reset,
    input signed [23:0] data,
    input signed [23:0] weight,
    output signed [15:0] result
    );

    MAC MAC
    (
    .clk(clk),
    .reset(reset),
    .data(data),
    .weight(weight),
    .result(result)
    );
endmodule
```

- Test Bench Code

```
module Conv_tb();
 reg clk, reset;
 reg signed [23:0] data;
 reg signed [23:0] weight;
 wire signed [15:0] result;

     Conv DUT
     (
     .clk(clk),
     .reset(reset),
     .data(data),
     .weight(weight),
     .result(result)
     );
    initial
    begin
        clk = 0;
        reset = 1;
        data =24'b10000000_10000000_10000000; weight = 24'b11111111_11111111_11111111; #8;
        reset = 0; #10;
        data = 24'b11111111_11111111_10000000; weight = 24'b11111111_00001000_11111111; #15;
        data = 24'b11111111_11111111_10000000; weight= 24'b11111111_11111111_11111111; #10;
        $write("Result(11) = %d\n",result);

        reset = 1;
        data = 24'b10000000_10000000_10000000; weight =24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data = 24'b11111111_10000000_11111111; weight = 24'b11111111_00001000_11111111; #10
        data = 24'b11111111_10000000_11111111;weight = 24'b11111111_11111111_11111111; #10;
        $write("Result(12) = %d\n", result);

        reset = 1;
        data = 24'b10000000_10000000_10000000; weight =24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data =24'b10000000_11111111_11111111; weight =24'b11111111_00001000_11111111; #10;
        data =24'b10000000_11111111_11111111; weight =24'b11111111_11111111_11111111; #10;
        $write("Result(13) = %d\n",result);

        reset = 1;
        data = 24'b11111111_11111111_10000000; weight =24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data =24'b11111111_11111111_10000000; weight = 24'b11111111_00001000_11111111; #10;
        data = 24'b11111111_11111111_10000000; weight = 24'b11111111_11111111_11111111; #10;
        $write("Result(21) = %d\n",result);

        reset = 1;
        data = 24'b11111111_10000000_11111111; weight = 24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data = 24'b11111111_10000000_11111111; weight =24'b11111111_00001000_11111111; #10;
        data = 24'b11111111_10000000_11111111; weight = 24'b11111111_11111111_11111111; #10;
        $write("Result(22) = %d\n",result);
```

```
        reset = 1;
        data =24'b11111111_11111111_10000000; weight =24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data = 24'b11111111_11111111_10000000; weight = 24'b11111111_00001000_11111111; #10;
        data =24'b11111111_11111111_10000000; weight =24'b11111111_11111111_11111111; #10;
        $write("Result(31) = %d\n",result);

        reset = 1;
        data =24'b11111111_10000000_11111111; weight =24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data =24'b11111111_10000000_11111111; weight =24'b11111111_00001000_11111111; #10;
        data =24'b11111111_10000000_11111111; weight =24'b11111111_11111111_11111111; #10;
        $write("Result(32) = %d\n",result);

        reset = 1;
        data =24'b10000000_11111111_11111111; weight =24'b11111111_11111111_11111111; #10;
        reset = 0; #10
        data =24'b10000000_11111111_11111111; weight =24'b11111111_00001000_11111111; #10;
        data =24'b10000000_11111111_11111111; weight =24'b11111111_11111111_11111111; #7;
        $write("Result(33) = %d\n",result);

        $finish;
    end

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

- Behavioral Simulation
  - In cases where reset=0 and 3 consecutive posedge signals, the simulation gives results. Demonstrate that this simulation result and codes work correctly. Information about the dataset is available in the MATLAB code.



- TCL Console

```
Result(11) =    635
Result(12) =   -508
Result(13) =    635
Result(21) =    381
Result(22) =   -762
Result(23) =    381
Result(31) =    381
Result(32) =   -762
Result(33) =    381
$finish called at time : 360 ns : File "C:/Users/yigit/Desktop/Okul/2022guz/Sayisal Sistem Tasarim Uygulamalari/sstu_odevler/Experiment7/sstu_odev7/sstu_odev7.srcs/sim_1
INFO: [USF-XSim-96] XSim completed. Design snapshot 'Conv_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:07 . Memory (MB): peak = 866.074 ; gain = 66.816
```

- $\mathbf{R:}\begin{bmatrix} 635 & -508 & 635 \\ 381 & -762 & 381 \\ 381 & -762 & 381 \end{bmatrix}$

- MATLAB Code

```
% 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 ];
%image = [0 4 0; 1 9 0; 7 4 5];
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
```

- Output

```
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 =
   635  -508   635
   381  -762   381
   381  -762   381
```

- **REFERENCES FOR EXPLANATION OF 2D CONVOLUTION**

➢ ILERI, A. (2018, August 26). *2D convolution*. Medium. Retrieved December 13, 2022, from https://abdulsamet-ileri.medium.com/2d-convolution-ced5d339aa5