# DIGITAL SYSTEM DESIGN APPLICATION

**EHB436E CRN: 11280**

**Salih Ömer Ongün**
**040220780**

**Experiment 4**

Salih Ömer Ongün
040220780

# HALF ADDER

## Design Sources

Half Adder design source code

```
module HA
(
    input x,
    input y,
    output cout,
    output sum
);
    assign sum = x ^ y ;
    assign cout = x & y;

endmodule
```

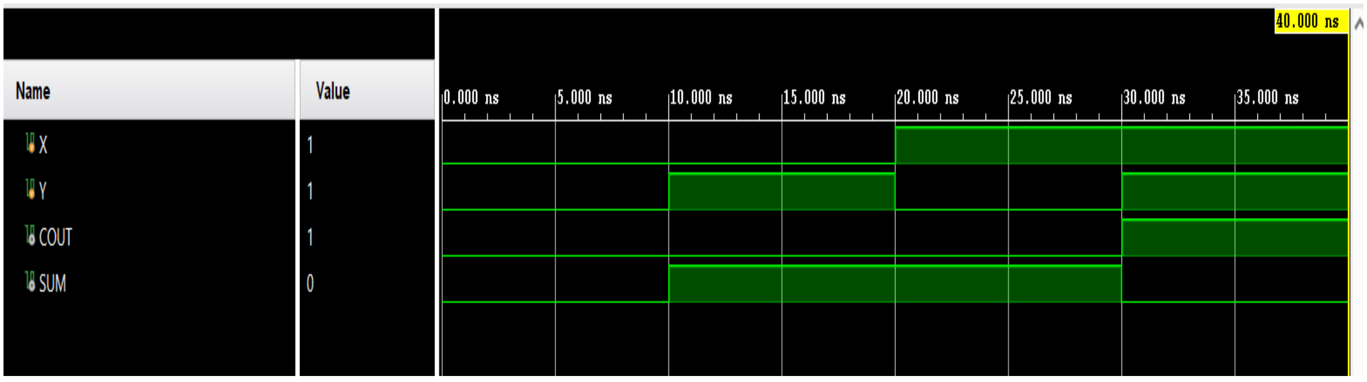
## Simulation Sources

Half Adder simulation source codes

```
module HA_tb();

    reg X= 1'b0;
    reg Y= 1'b0;
    wire COUT;
    wire SUM;

    HA uut
    (
        .x(X),
        .y(Y),
        .cout(COUT),
        .sum(SUM)
    );

    initial
    begin
        X = 0;
        Y = 0;
        #10;

        X = 0;
        Y = 1;
        #10;

        X = 1;
        Y = 0;
        #10;

        X = 1;
        Y = 1;
        #10;
        $finish;
    end
endmodule
```

## Simulation Wave

Behavioral simulation wave screenshot



As seen in the simulation, the module works as it should. Sum bit has logic 1 value if one of two inputs has logic 1 value. Cout has logic 1 value if two bit have logic 1 value.

# FULL ADDER

## Design Sources

Full Adder design source code

```
module FA
(
    input x,
    input y,
    input cin,
    output cout,
    output sum
);
    wire ha1_sum, ha1_cout,
ha2_cout;

    HA half1
    (
      .x(x),
      .y(y),
      .sum(ha1_sum),
      .cout(ha1_cout)
    );

    HA half2
    (
        .x(ha1_sum),
        .y(cin),
        .sum(sum),
        .cout(ha2_cout)
    );

    OR or1
    (
    .I1(ha1_cout),
    .I2(ha2_cout),
    .O(cout)
    );

endmodule
```

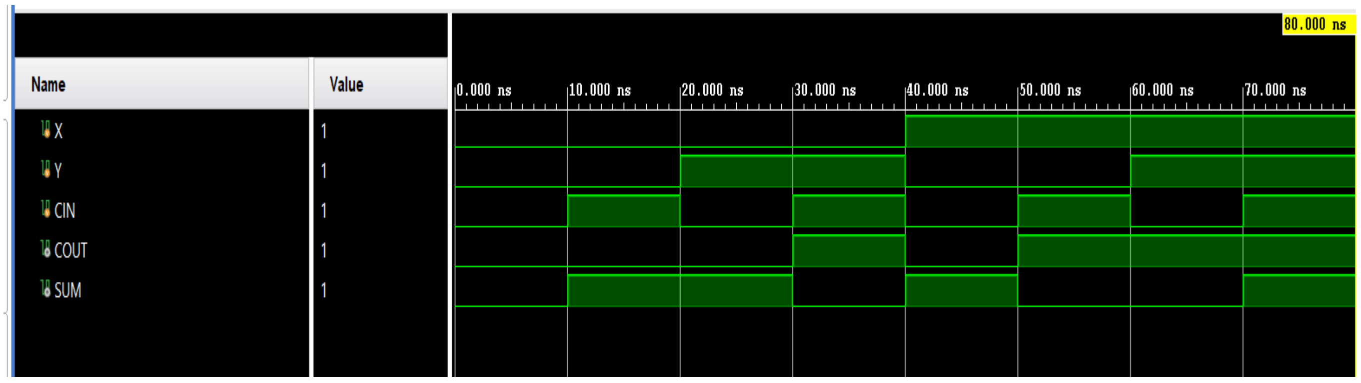
## Simulation Sources

Full Adder simulation source codes

```
module FA_tb();

    reg X= 1'b0;
    reg Y= 1'b0;
    reg CIN= 1'b0;
    wire COUT;
    wire SUM;

    FA uut
    (
        .x(X),
        .y(Y),
        .cin(CIN),
        .cout(COUT),
        .sum(SUM)
    );

    initial
    begin
        X = 0;
        Y = 0;
        CIN = 0;
        #10;

        X = 0;
        Y = 0;
        CIN = 1;
        #10;

        X = 0;
        Y = 1;
        CIN = 0;
        #10;

        X = 0;
        Y = 1;
        CIN = 1;
        #10;

        X = 1;
        Y = 0;
        CIN = 0;
        #10;

        X = 1;
        Y = 0;
        CIN = 1;
        #10;

        X = 1;
        Y = 1;
        CIN = 0;
        #10;

        X = 1;
        Y = 1;
        CIN = 1;
        #10;
        $finish;
    end
endmodule
```

## Simulation Wave

Behavioral simulation wave screenshot



As seen in the simulation, the module works as it should. Sum bit has logic 1 value if one of three (X, Y, CIN) inputs has logic 1 value. Cout has logic 1 if two of three (X, Y, CIN) inputs has logic 1 value. If all inputs have logic 1 values, then both outputs have logic 1 values.

Salih Ömer Ongün
040220780

# RIPPLE CARRY ADDER

## Design Sources

Ripple Carry Adder design source code

```
module RCA
(
    input [3:0] x,
    input [3:0] y,
    input cin,
    output cout,
    output [3:0] sum
);
    wire cout1,cout2,cout3;

    FA fa1
    (
        .x(x[0]),
        .y(y[0]),
        .cin(cin),
        .sum(sum[0]),
        .cout(cout1)
    );

    FA fa2
    (
        .x(x[1]),
        .y(y[1]),
        .cin(cout1),
        .sum(sum[1]),
        .cout(cout2)
    );

    FA fa3
    (
        .x(x[2]),
        .y(y[2]),
        .cin(cout2),
        .sum(sum[2]),
        .cout(cout3)
    );

    FA fa4
    (
        .x(x[3]),
        .y(y[3]),
        .cin(cout3),
        .sum(sum[3]),
        .cout(cout)
    );
endmodule
```

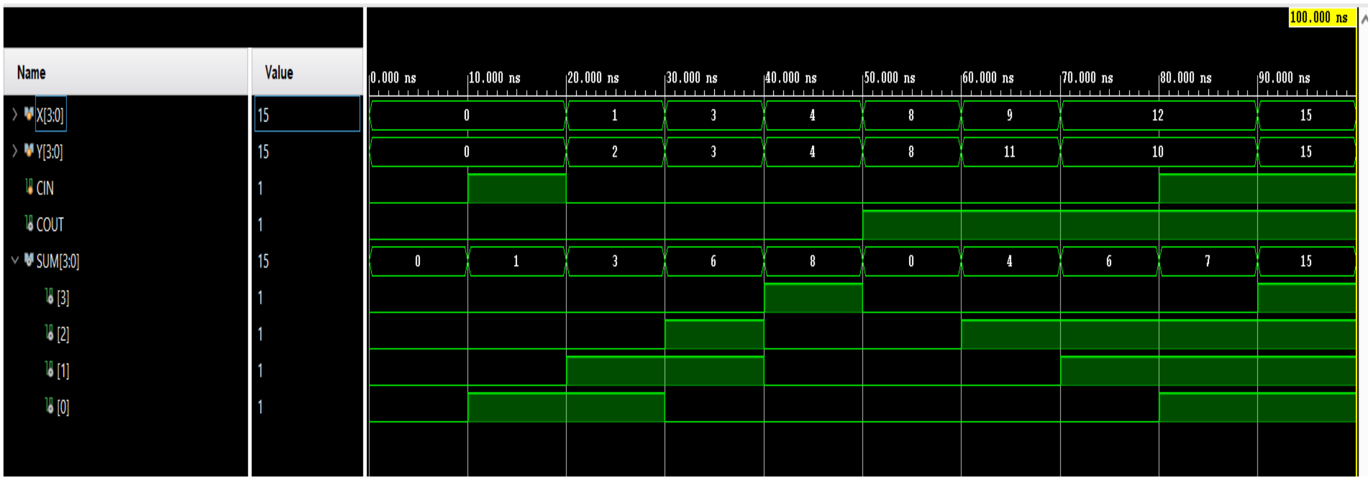
## Simulation Sources

Ripple Carry Adder simulation source codes

```
module RCA_tb();

    reg [3:0] X= 4'b0;
    reg [3:0] Y= 4'b0;
    reg CIN= 1'b0;
    wire COUT;
    wire [3:0] SUM;

    RCA uut
    (
        .x(X),
        .y(Y),
        .cin(CIN),
        .cout(COUT),
        .sum(SUM)
    );

    initial
    begin
        X = 4'b0000;
        Y = 4'b0000;
        CIN = 0;
        #10;

        X = 4'b0000;
        Y = 4'b0000;
        CIN = 1;
        #10;


        X = 4'b0001;
        Y = 4'b0010;
        CIN = 0;
        #10;


        X = 4'b0011;
        Y = 4'b0011;
        CIN = 0;
        #10;


        X = 4'b0100;
        Y = 4'b0100;
        CIN = 0;
        #10;


        X = 4'b1000;
        Y = 4'b1000;
        CIN = 0;
        #10;


        X = 4'b1001;
        Y = 4'b1011;
        CIN = 0;
        #10;


        X = 4'b1100;
        Y = 4'b1010;
        CIN = 0;
        #10;

        X = 4'b1100;
        Y = 4'b1010;
        CIN = 1;
        #10;

        X = 4'b1111;
        Y = 4'b1111;
        CIN = 1;
        #10;



        $finish;
    end
endmodule
```

## Simulation Wave

Behavioral simulation wave screenshot



| Name | Value | 0 ns | 10 ns | 20 ns | 30 ns | 40 ns | 50 ns | 60 ns | 70 ns | 80 ns | 90 ns |
|---|---|---|---|---|---|---|---|---|---|---|---|
| X[3:0] | 15 | 0 | 0 | 1 | 3 | 4 | 8 | 9 | 12 | 12 | 15 |
| Y[3:0] | 15 | 0 | 0 | 2 | 3 | 4 | 8 | 11 | 10 | 10 | 15 |
| CIN | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| COUT | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| SUM[3:0] | 15 | 0 | 1 | 3 | 6 | 8 | 0 | 4 | 6 | 7 | 15 |

Ripple Carry Adder has the same working principle as Full Adder. It have three inputs (X, Y, CIN) and two outputs (COUT, SUM). The main difference between the two modules is that Ripple Carry Adder has multiple bit inputs and outputs. Therefore, it uses 4 Full Adder for the module. As seen in the simulation, the module works as it should. If sum of X, Y and CIN have more than 15, COUT has logic 1 values. Because SUM has a 4- bit representation.

# PARAMETRIC RIPPLE CARRY ADDER

## Design Sources

Parametric Ripple Carry Adder design source code

```
module parametric_RCA #(parameter SIZE = 8)
(
    input [SIZE-1:0] x,
    input [SIZE-1:0] y,
    input cin,
    output cout,
    output [SIZE-1:0] sum
);

    wire [SIZE:0] cout_gen;

    assign cout_gen[0] = cin;

    genvar i;
    generate
        for(i = 0; i<SIZE; i = i+1) begin :
gen_full_adder
            FA gen_full
            (
                x[i],
                y[i],
                cout_gen[i],
                cout_gen[i+1],
                sum[i]
            );
        end
    endgenerate
    assign cout = cout_gen[SIZE];

endmodule
```

## Simulation Sources

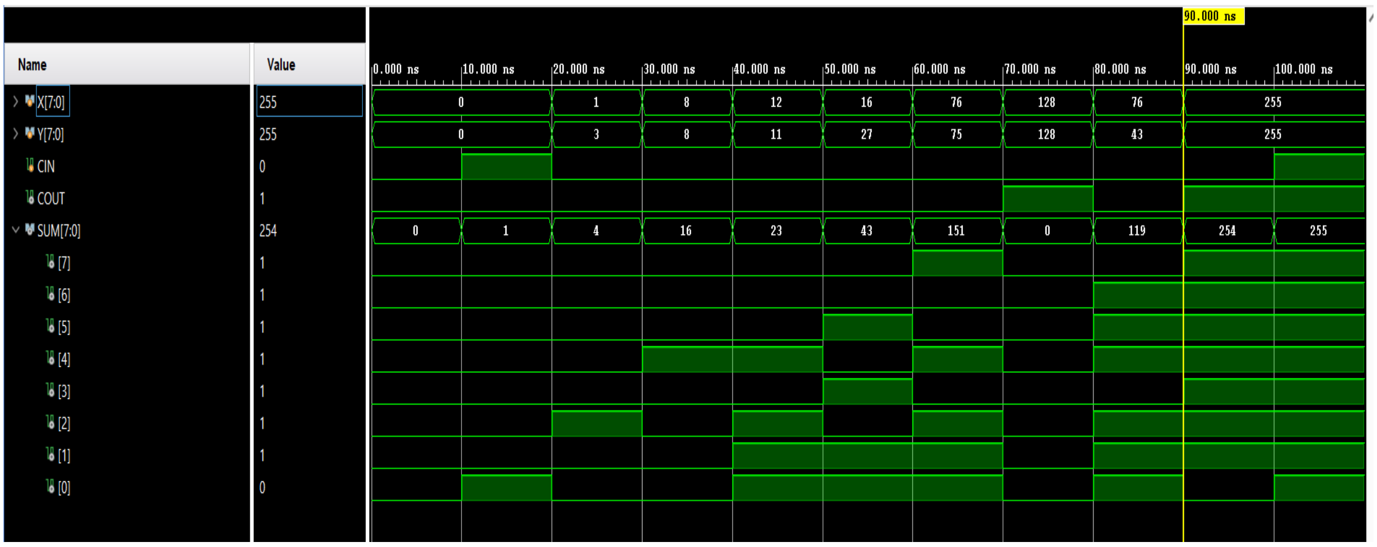
Parametric Ripple Carry Adder simulation source codes

```
module parametric_RCA_tb();

parameter SIZE = 8;

    reg [SIZE-1:0] X;
    reg [SIZE-1:0] Y;
    reg CIN;
    wire COUT;
    wire [SIZE-1:0] SUM;

    parametric_RCA #(.SIZE(SIZE))
uut
    (
        .x(X),
        .y(Y),
        .cin(CIN),
        .cout(COUT),
        .sum(SUM)
    );

    initial
    begin
        X = 8'b00000000;
        Y = 8'b00000000;
        CIN = 0;
        #10;

        X = 8'b00000000;
        Y = 8'b00000000;
        CIN = 1;
        #10;


        X = 8'b00000001;
        Y = 8'b00000011;
        CIN = 0;
        #10;


        X = 8'b00001000;
        Y = 8'b00001000;
        CIN = 0;
        #10;


        X = 8'b00001100;
        Y = 8'b00001011;
        CIN = 0;
        #10;


        X = 8'b00010000;
        Y = 8'b00011011;
        CIN = 0;
        #10;
```

```

        X = 8'b001001100;
        Y = 8'b01001011;
        CIN = 0;
        #10;


        X = 8'b10000000;
        Y = 8'b10000000;
        CIN = 0;
        #10;

        X = 8'b01001100;
        Y = 8'b00101011;
        CIN = 0;
        #10;

        X = 8'b11111111;
        Y = 8'b11111111;
        CIN = 0;
        #10;

        X = 8'b11111111;
        Y = 8'b11111111;
        CIN = 1;
        #10;

        $finish;
    end
endmodule
```

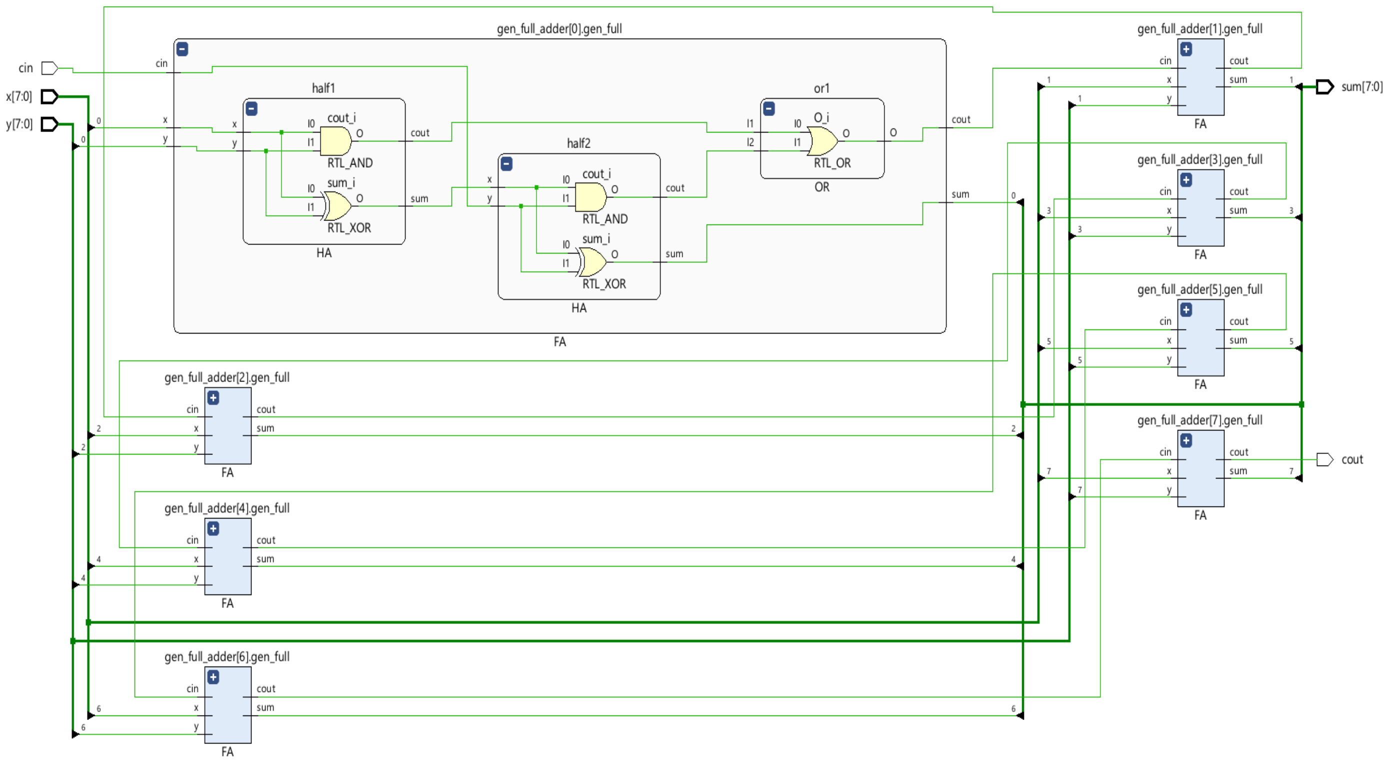
## Simulation Wave

Behavioral simulation wave screenshot

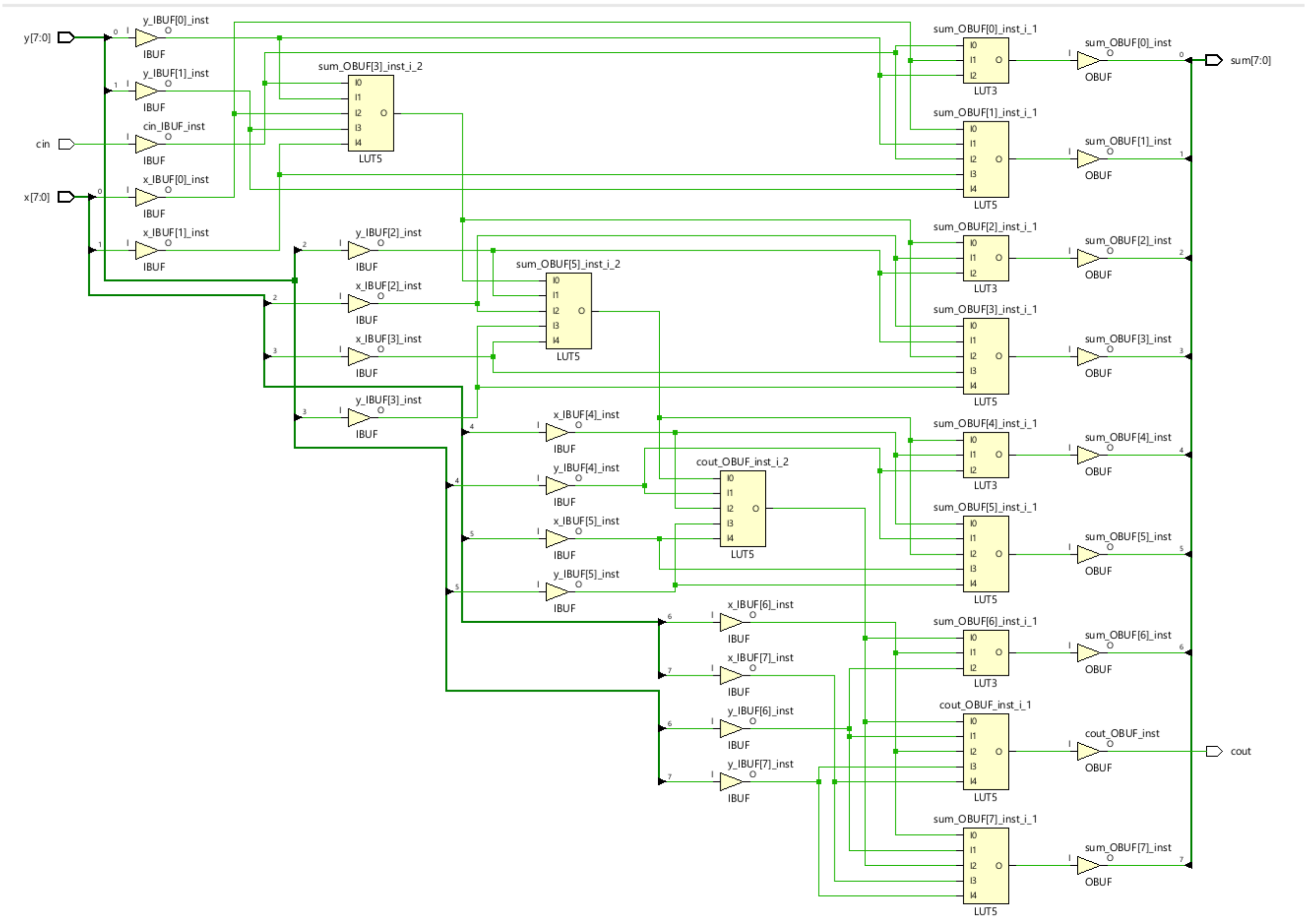

90.000 ns
Name
Value
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
X[7:0] 255 | 0 | 1 | 8 | 12 | 16 | 76 | 128 | 76 | 255
Y[7:0] 255 | 0 | 3 | 8 | 11 | 27 | 75 | 128 | 43 | 255
CIN 0
COUT 1
SUM[7:0] 254 | 0 | 1 | 4 | 16 | 23 | 43 | 151 | 0 | 119 | 254 | 255
[7] 1
[6] 1
[5] 1
[4] 1
[3] 1
[2] 1
[1] 1
[0] 0

As seen in the simulation, the module works as it should. The main difference between Ripple Carry Adder and Parametric Ripple Carry Adder is their use of parameters. Therefore, users design any number of bits adder design. SIZE parameter is 8 for the design. If sum of X, Y and CIN have more than 255, COUT has logic 1 values.

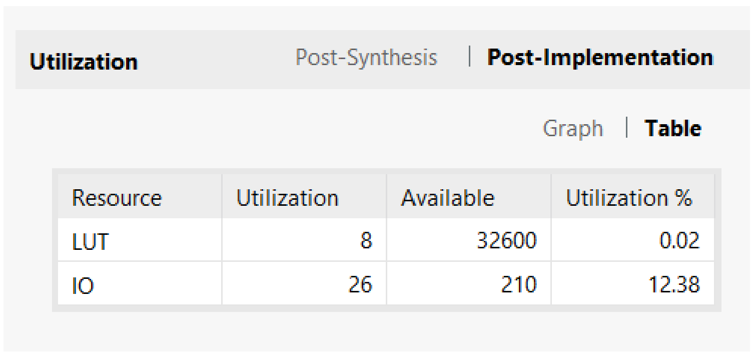
## RTL Schematic


cin
x[7:0]
y[7:0]
gen_full_adder[0].gen_full
half1
cout_i
RTL_AND
sum_i
RTL_XOR
HA
half2
or1
O_i
RTL_OR
OR
FA
gen_full_adder[1].gen_full
gen_full_adder[2].gen_full
gen_full_adder[3].gen_full
gen_full_adder[4].gen_full
gen_full_adder[5].gen_full
gen_full_adder[6].gen_full
gen_full_adder[7].gen_full
sum[7:0]
cout

## Technology Schematic



## Utilization Report



Utilization Post-Synthesis | Post-Implementation

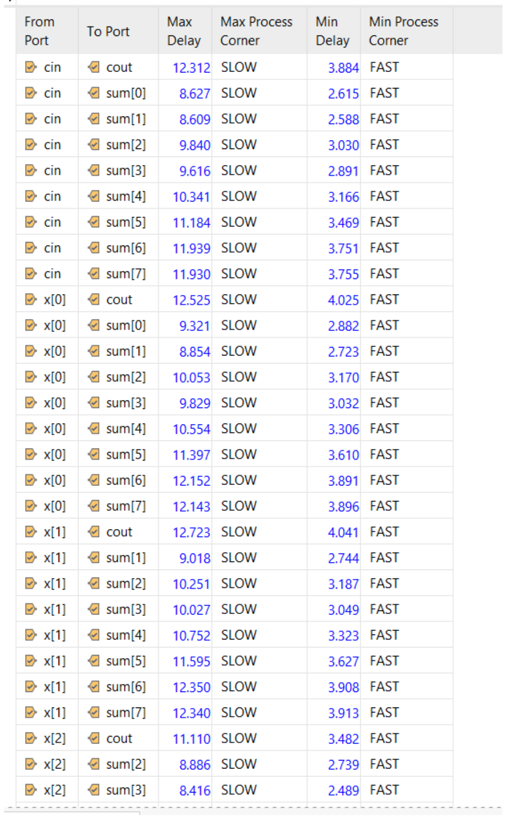
Graph | Table

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 8 | 32600 | 0.02 |
| IO | 26 | 210 | 12.38 |

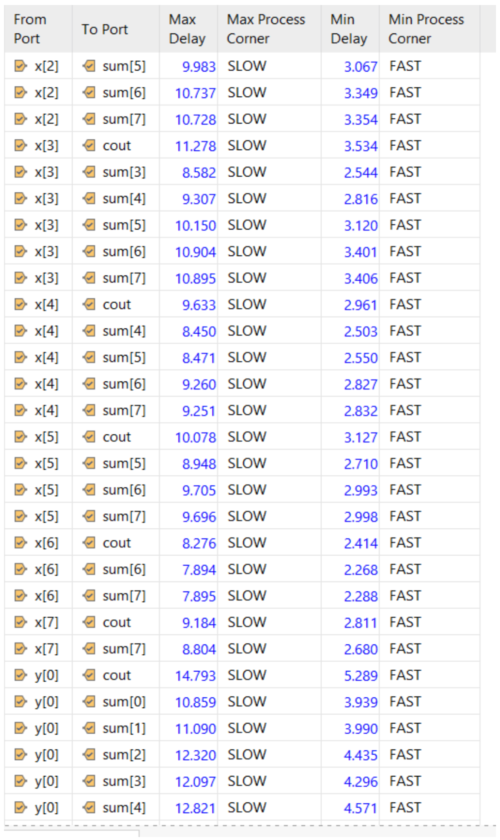
8 LUT’s is used in the design.

Salih Ömer Ongün
040220780

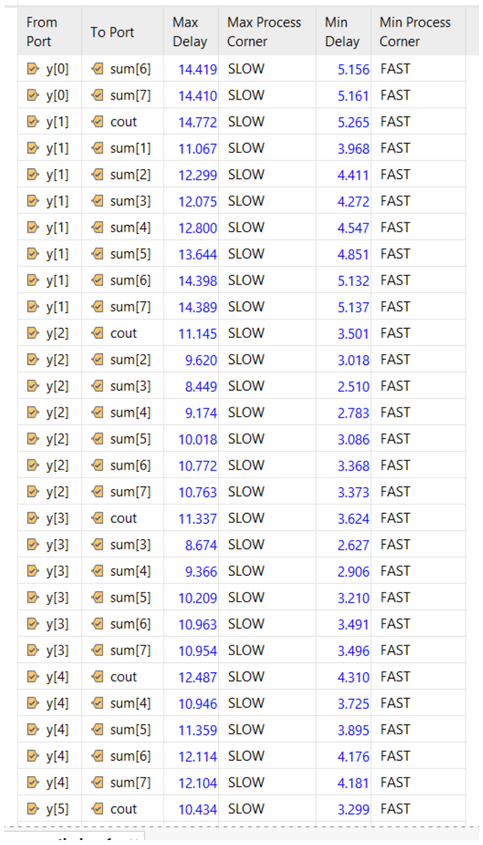
## Timing Report



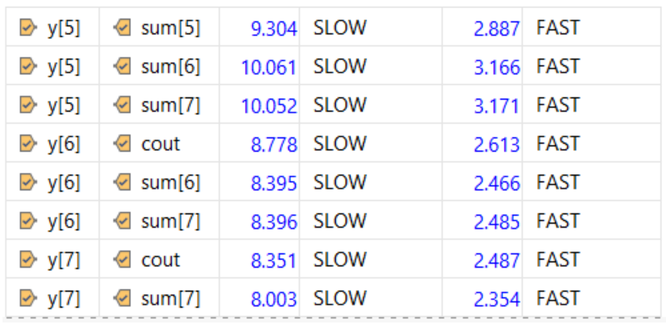
| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| cin | cout | 12.312 | SLOW | 3.884 | FAST |
| cin | sum[0] | 8.627 | SLOW | 2.615 | FAST |
| cin | sum[1] | 8.609 | SLOW | 2.588 | FAST |
| cin | sum[2] | 9.840 | SLOW | 3.030 | FAST |
| cin | sum[3] | 9.616 | SLOW | 2.891 | FAST |
| cin | sum[4] | 10.341 | SLOW | 3.166 | FAST |
| cin | sum[5] | 11.184 | SLOW | 3.469 | FAST |
| cin | sum[6] | 11.939 | SLOW | 3.751 | FAST |
| cin | sum[7] | 11.930 | SLOW | 3.755 | FAST |
| x[0] | cout | 12.525 | SLOW | 4.025 | FAST |
| x[0] | sum[0] | 9.321 | SLOW | 2.882 | FAST |
| x[0] | sum[1] | 8.854 | SLOW | 2.723 | FAST |
| x[0] | sum[2] | 10.053 | SLOW | 3.170 | FAST |
| x[0] | sum[3] | 9.829 | SLOW | 3.032 | FAST |
| x[0] | sum[4] | 10.554 | SLOW | 3.306 | FAST |
| x[0] | sum[5] | 11.397 | SLOW | 3.610 | FAST |
| x[0] | sum[6] | 12.152 | SLOW | 3.891 | FAST |
| x[0] | sum[7] | 12.143 | SLOW | 3.896 | FAST |
| x[1] | cout | 12.723 | SLOW | 4.041 | FAST |
| x[1] | sum[1] | 9.018 | SLOW | 2.744 | FAST |
| x[1] | sum[2] | 10.251 | SLOW | 3.187 | FAST |
| x[1] | sum[3] | 10.027 | SLOW | 3.049 | FAST |
| x[1] | sum[4] | 10.752 | SLOW | 3.323 | FAST |
| x[1] | sum[5] | 11.595 | SLOW | 3.627 | FAST |
| x[1] | sum[6] | 12.350 | SLOW | 3.908 | FAST |
| x[1] | sum[7] | 12.340 | SLOW | 3.913 | FAST |
| x[2] | cout | 11.110 | SLOW | 3.482 | FAST |
| x[2] | sum[2] | 8.886 | SLOW | 2.739 | FAST |
| x[2] | sum[3] | 8.416 | SLOW | 2.489 | FAST |



| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| x[2] | sum[5] | 9.983 | SLOW | 3.067 | FAST |
| x[2] | sum[6] | 10.737 | SLOW | 3.349 | FAST |
| x[2] | sum[7] | 10.728 | SLOW | 3.354 | FAST |
| x[3] | cout | 11.278 | SLOW | 3.534 | FAST |
| x[3] | sum[3] | 8.582 | SLOW | 2.544 | FAST |
| x[3] | sum[4] | 9.307 | SLOW | 2.816 | FAST |
| x[3] | sum[5] | 10.150 | SLOW | 3.120 | FAST |
| x[3] | sum[6] | 10.904 | SLOW | 3.401 | FAST |
| x[3] | sum[7] | 10.895 | SLOW | 3.406 | FAST |
| x[4] | cout | 9.633 | SLOW | 2.961 | FAST |
| x[4] | sum[4] | 8.450 | SLOW | 2.503 | FAST |
| x[4] | sum[5] | 8.471 | SLOW | 2.550 | FAST |
| x[4] | sum[6] | 9.260 | SLOW | 2.827 | FAST |
| x[4] | sum[7] | 9.251 | SLOW | 2.832 | FAST |
| x[5] | cout | 10.078 | SLOW | 3.127 | FAST |
| x[5] | sum[5] | 8.948 | SLOW | 2.710 | FAST |
| x[5] | sum[6] | 9.705 | SLOW | 2.993 | FAST |
| x[5] | sum[7] | 9.696 | SLOW | 2.998 | FAST |
| x[6] | cout | 8.276 | SLOW | 2.414 | FAST |
| x[6] | sum[6] | 7.894 | SLOW | 2.268 | FAST |
| x[6] | sum[7] | 7.895 | SLOW | 2.288 | FAST |
| x[7] | cout | 9.184 | SLOW | 2.811 | FAST |
| x[7] | sum[7] | 8.804 | SLOW | 2.680 | FAST |
| y[0] | cout | 14.793 | SLOW | 5.289 | FAST |
| y[0] | sum[0] | 10.859 | SLOW | 3.939 | FAST |
| y[0] | sum[1] | 11.090 | SLOW | 3.990 | FAST |
| y[0] | sum[2] | 12.320 | SLOW | 4.435 | FAST |
| y[0] | sum[3] | 12.097 | SLOW | 4.296 | FAST |
| y[0] | sum[4] | 12.821 | SLOW | 4.571 | FAST |



| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| y[0] | sum[6] | 14.419 | SLOW | 5.156 | FAST |
| y[0] | sum[7] | 14.410 | SLOW | 5.161 | FAST |
| y[1] | cout | 14.772 | SLOW | 5.265 | FAST |
| y[1] | sum[1] | 11.067 | SLOW | 3.968 | FAST |
| y[1] | sum[2] | 12.299 | SLOW | 4.411 | FAST |
| y[1] | sum[3] | 12.075 | SLOW | 4.272 | FAST |
| y[1] | sum[4] | 12.800 | SLOW | 4.547 | FAST |
| y[1] | sum[5] | 13.644 | SLOW | 4.851 | FAST |
| y[1] | sum[6] | 14.398 | SLOW | 5.132 | FAST |
| y[1] | sum[7] | 14.389 | SLOW | 5.137 | FAST |
| y[2] | cout | 11.145 | SLOW | 3.501 | FAST |
| y[2] | sum[2] | 9.620 | SLOW | 3.018 | FAST |
| y[2] | sum[3] | 8.449 | SLOW | 2.510 | FAST |
| y[2] | sum[4] | 9.174 | SLOW | 2.783 | FAST |
| y[2] | sum[5] | 10.018 | SLOW | 3.086 | FAST |
| y[2] | sum[6] | 10.772 | SLOW | 3.368 | FAST |
| y[2] | sum[7] | 10.763 | SLOW | 3.373 | FAST |
| y[3] | cout | 11.337 | SLOW | 3.624 | FAST |
| y[3] | sum[3] | 8.674 | SLOW | 2.627 | FAST |
| y[3] | sum[4] | 9.366 | SLOW | 2.906 | FAST |
| y[3] | sum[5] | 10.209 | SLOW | 3.210 | FAST |
| y[3] | sum[6] | 10.963 | SLOW | 3.491 | FAST |
| y[3] | sum[7] | 10.954 | SLOW | 3.496 | FAST |
| y[4] | cout | 12.487 | SLOW | 4.310 | FAST |
| y[4] | sum[4] | 10.946 | SLOW | 3.725 | FAST |
| y[4] | sum[5] | 11.359 | SLOW | 3.895 | FAST |
| y[4] | sum[6] | 12.114 | SLOW | 4.176 | FAST |
| y[4] | sum[7] | 12.104 | SLOW | 4.181 | FAST |
| y[5] | cout | 10.434 | SLOW | 3.299 | FAST |



| | | | | | |
|---|---|---|---|---|---|
| y[5] | sum[5] | 9.304 | SLOW | 2.887 | FAST |
| y[5] | sum[6] | 10.061 | SLOW | 3.166 | FAST |
| y[5] | sum[7] | 10.052 | SLOW | 3.171 | FAST |
| y[6] | cout | 8.778 | SLOW | 2.613 | FAST |
| y[6] | sum[6] | 8.395 | SLOW | 2.466 | FAST |
| y[6] | sum[7] | 8.396 | SLOW | 2.485 | FAST |
| y[7] | cout | 8.351 | SLOW | 2.487 | FAST |
| y[7] | sum[7] | 8.003 | SLOW | 2.354 | FAST |

There are maximum delays between the input paths y and output paths count. The greatest delay is between y[0] and the cout path.

## PARAMETRIC CARRY LOOKAHEAD ADDER

### Design Sources

Parametric Carry Lookahead Adder design source code

```
module CLA #(parameter SIZE = 8)
(
    input [SIZE-1:0] x,
    input [SIZE-1:0] y,
    input cin,
    output cout,
    output [SIZE-1:0] s

);

    wire [SIZE:0] p;
    wire [SIZE:0] g;

    wire [SIZE+1:0] c;
    assign c[0] = cin;

    genvar i;
    genvar j;
    generate
        for(i = 0; i<SIZE; i = i+1) begin: gen_p_g
            assign p[i] = x[i] ^ y[i];
            assign g[i] = x[i] & y[i];
        end

    endgenerate

    generate
        for(j = 0; j<SIZE+1; j = j+1) begin : gen_c
            assign c[j+1] = g[j]|(p[j] & c[j]);
        end
    endgenerate

    assign cout = c[SIZE];

    generate
        for(j = 0; j<SIZE; j = j+1) begin: gen_s
              assign s[j] = p[j] ^ c[j];
        end
    endgenerate
endmodule
```

### Simulation Sources

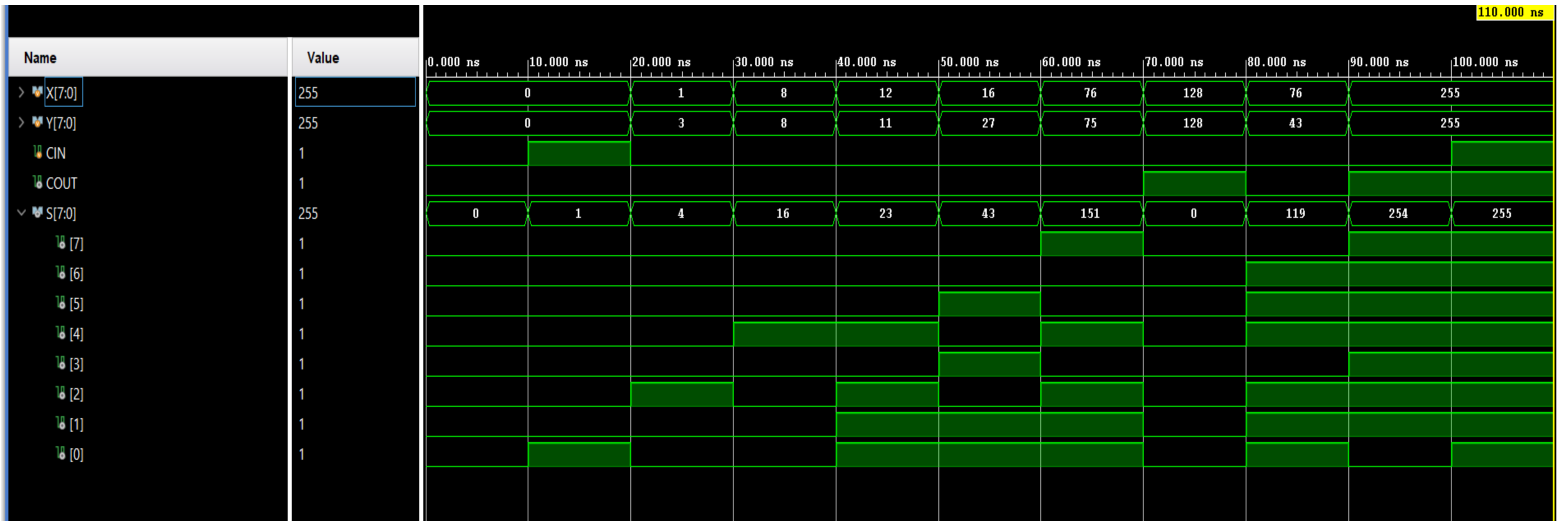
Parametric Carry Lookahead Adder simulation source codes

```verilog
module CLA_tb();

parameter SIZE = 8;

    reg [SIZE-1:0] X;
    reg [SIZE-1:0] Y;
    reg CIN;
    wire COUT;
    wire [SIZE-1:0] S;

    CLA #(.SIZE(SIZE)) uut
    (
        .x(X),
        .y(Y),
        .cin(CIN),
        .cout(COUT),
        .s(S)
    );

    initial
    begin
        X = 8'b00000000;
        Y = 8'b00000000;
        CIN = 0;
        #10;

        X = 8'b00000000;
        Y = 8'b00000000;
        CIN = 1;
        #10;


        X = 8'b00000001;
        Y = 8'b00000011;
        CIN = 0;
        #10;


        X = 8'b00001000;
        Y = 8'b00001000;
        CIN = 0;
        #10;


        X = 8'b00001100;
        Y = 8'b00001011;
        CIN = 0;
        #10;


        X = 8'b00010000;
        Y = 8'b00011011;
        CIN = 0;
        #10;


        X = 8'b001001100;
        Y = 8'b01001011;
        CIN = 0;
        #10;


        X = 8'b10000000;
        Y = 8'b10000000;
        CIN = 0;
        #10;

        X = 8'b01001100;
        Y = 8'b00101011;
        CIN = 0;
        #10;
```

```verilog
        X = 8'b11111111;
        Y = 8'b11111111;
        CIN = 0;
        #10;

        X = 8'b11111111;
        Y = 8'b11111111;
        CIN = 1;
        #10;

        $finish;
    end
endmodule
```

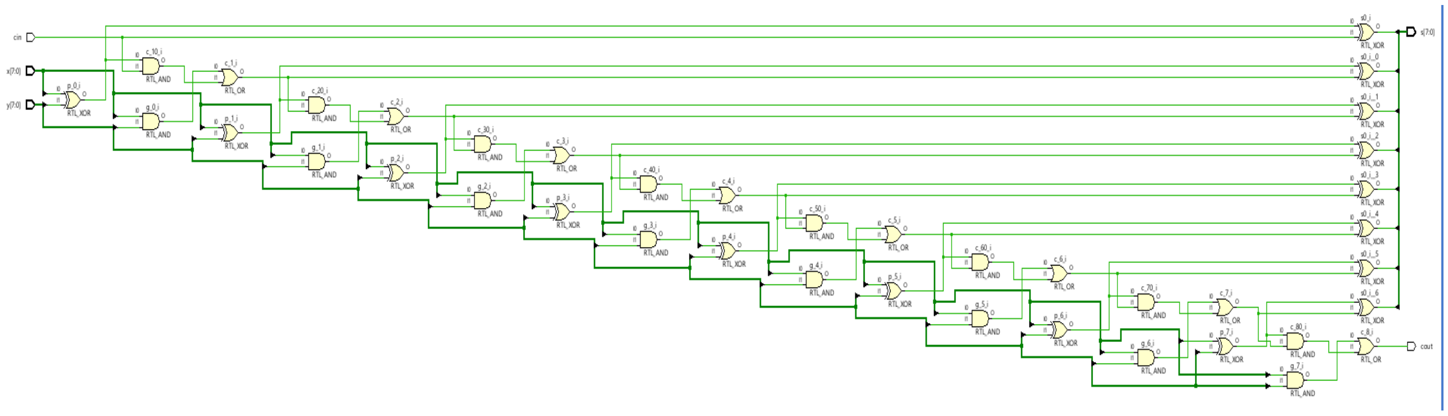
## Simulation Wave

Behavioral simulation wave screenshot

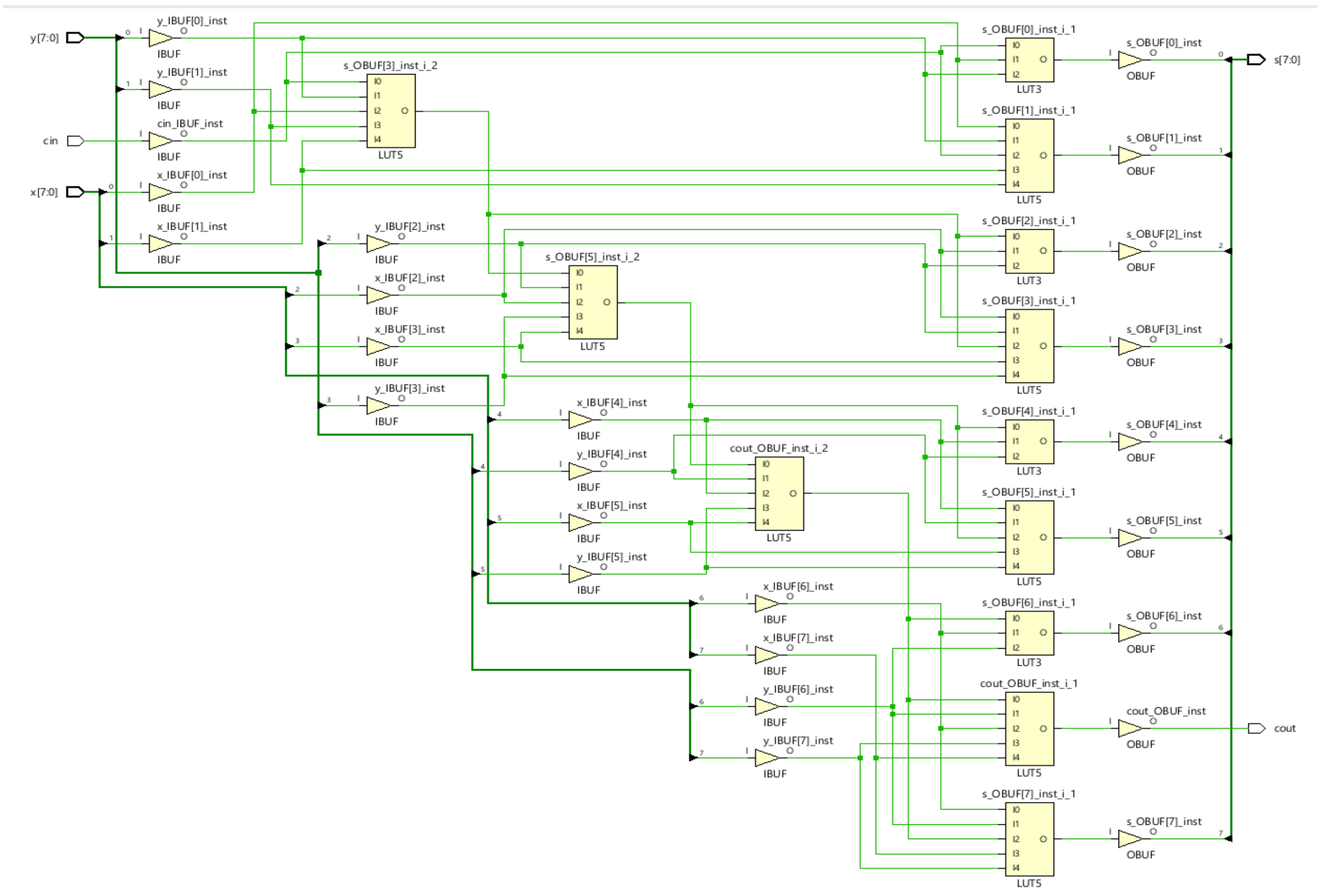


As seen in the simulation, the module works as it should. There are not any differences between parametric ripple carry adder with parametric carry lookahead adder between results. There are differences in the algorithm for these two adders.

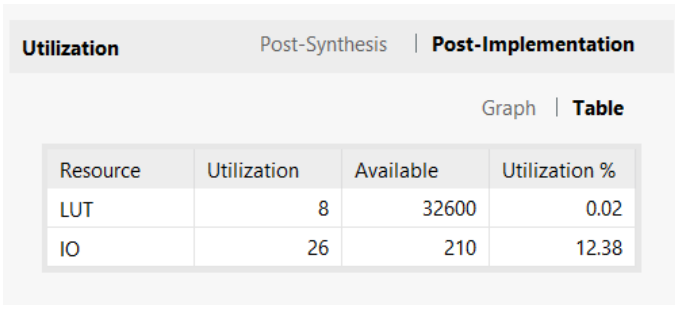
## RTL Schematic



## Technology Schematic



## Utilization Report

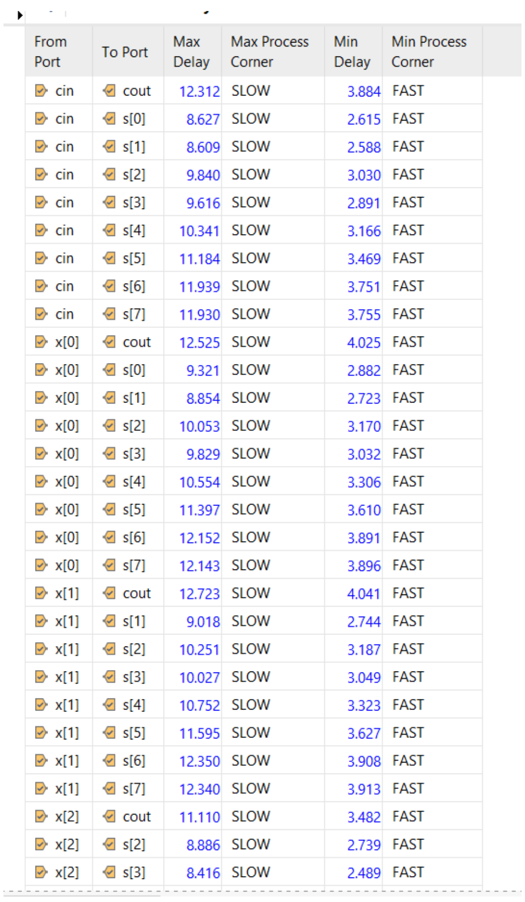


**Utilization** Post-Synthesis | **Post-Implementation**

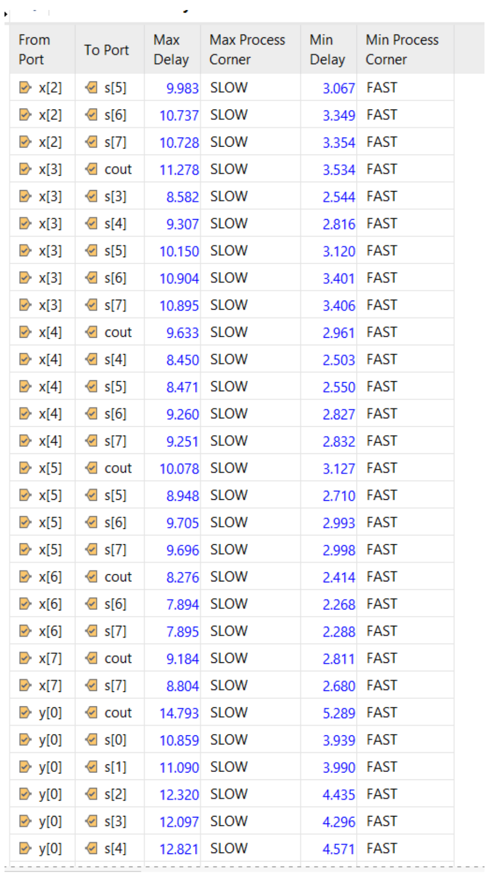
Graph | **Table**

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 8 | 32600 | 0.02 |
| IO | 26 | 210 | 12.38 |

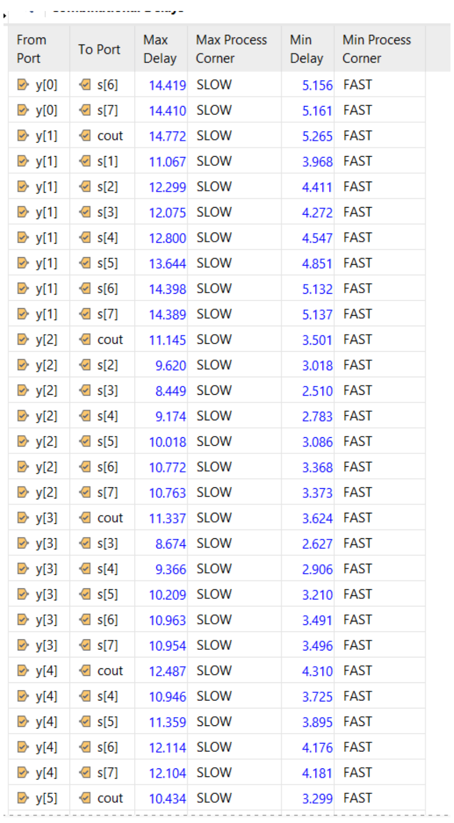
## Timing Report



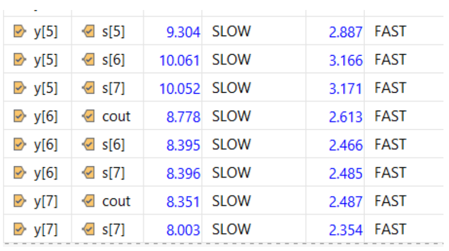
| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| cin | cout | 12.312 | SLOW | 3.884 | FAST |
| cin | s[0] | 8.627 | SLOW | 2.615 | FAST |
| cin | s[1] | 8.609 | SLOW | 2.588 | FAST |
| cin | s[2] | 9.840 | SLOW | 3.030 | FAST |
| cin | s[3] | 9.616 | SLOW | 2.891 | FAST |
| cin | s[4] | 10.341 | SLOW | 3.166 | FAST |
| cin | s[5] | 11.184 | SLOW | 3.469 | FAST |
| cin | s[6] | 11.939 | SLOW | 3.751 | FAST |
| cin | s[7] | 11.930 | SLOW | 3.755 | FAST |
| x[0] | cout | 12.525 | SLOW | 4.025 | FAST |
| x[0] | s[0] | 9.321 | SLOW | 2.882 | FAST |
| x[0] | s[1] | 8.854 | SLOW | 2.723 | FAST |
| x[0] | s[2] | 10.053 | SLOW | 3.170 | FAST |
| x[0] | s[3] | 9.829 | SLOW | 3.032 | FAST |
| x[0] | s[4] | 10.554 | SLOW | 3.306 | FAST |
| x[0] | s[5] | 11.397 | SLOW | 3.610 | FAST |
| x[0] | s[6] | 12.152 | SLOW | 3.891 | FAST |
| x[0] | s[7] | 12.143 | SLOW | 3.896 | FAST |
| x[1] | cout | 12.723 | SLOW | 4.041 | FAST |
| x[1] | s[1] | 9.018 | SLOW | 2.744 | FAST |
| x[1] | s[2] | 10.251 | SLOW | 3.187 | FAST |
| x[1] | s[3] | 10.027 | SLOW | 3.049 | FAST |
| x[1] | s[4] | 10.752 | SLOW | 3.323 | FAST |
| x[1] | s[5] | 11.595 | SLOW | 3.627 | FAST |
| x[1] | s[6] | 12.350 | SLOW | 3.908 | FAST |
| x[1] | s[7] | 12.340 | SLOW | 3.913 | FAST |
| x[2] | cout | 11.110 | SLOW | 3.482 | FAST |
| x[2] | s[2] | 8.886 | SLOW | 2.739 | FAST |
| x[2] | s[3] | 8.416 | SLOW | 2.489 | FAST |



| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| x[2] | s[5] | 9.983 | SLOW | 3.067 | FAST |
| x[2] | s[6] | 10.737 | SLOW | 3.349 | FAST |
| x[2] | s[7] | 10.728 | SLOW | 3.354 | FAST |
| x[3] | cout | 11.278 | SLOW | 3.534 | FAST |
| x[3] | s[3] | 8.582 | SLOW | 2.544 | FAST |
| x[3] | s[4] | 9.307 | SLOW | 2.816 | FAST |
| x[3] | s[5] | 10.150 | SLOW | 3.120 | FAST |
| x[3] | s[6] | 10.904 | SLOW | 3.401 | FAST |
| x[3] | s[7] | 10.895 | SLOW | 3.406 | FAST |
| x[4] | cout | 9.633 | SLOW | 2.961 | FAST |
| x[4] | s[4] | 8.450 | SLOW | 2.503 | FAST |
| x[4] | s[5] | 8.471 | SLOW | 2.550 | FAST |
| x[4] | s[6] | 9.260 | SLOW | 2.827 | FAST |
| x[4] | s[7] | 9.251 | SLOW | 2.832 | FAST |
| x[5] | cout | 10.078 | SLOW | 3.127 | FAST |
| x[5] | s[5] | 8.948 | SLOW | 2.710 | FAST |
| x[5] | s[6] | 9.705 | SLOW | 2.993 | FAST |
| x[5] | s[7] | 9.696 | SLOW | 2.998 | FAST |
| x[6] | cout | 8.276 | SLOW | 2.414 | FAST |
| x[6] | s[6] | 7.894 | SLOW | 2.268 | FAST |
| x[6] | s[7] | 7.895 | SLOW | 2.288 | FAST |
| x[7] | cout | 9.184 | SLOW | 2.811 | FAST |
| x[7] | s[7] | 8.804 | SLOW | 2.680 | FAST |
| y[0] | cout | 14.793 | SLOW | 5.289 | FAST |
| y[0] | s[0] | 10.859 | SLOW | 3.939 | FAST |
| y[0] | s[1] | 11.090 | SLOW | 3.990 | FAST |
| y[0] | s[2] | 12.320 | SLOW | 4.435 | FAST |
| y[0] | s[3] | 12.097 | SLOW | 4.296 | FAST |
| y[0] | s[4] | 12.821 | SLOW | 4.571 | FAST |



| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| y[0] | s[6] | 14.419 | SLOW | 5.156 | FAST |
| y[0] | s[7] | 14.410 | SLOW | 5.161 | FAST |
| y[1] | cout | 14.772 | SLOW | 5.265 | FAST |
| y[1] | s[1] | 11.067 | SLOW | 3.968 | FAST |
| y[1] | s[2] | 12.299 | SLOW | 4.411 | FAST |
| y[1] | s[3] | 12.075 | SLOW | 4.272 | FAST |
| y[1] | s[4] | 12.800 | SLOW | 4.547 | FAST |
| y[1] | s[5] | 13.644 | SLOW | 4.851 | FAST |
| y[1] | s[6] | 14.398 | SLOW | 5.132 | FAST |
| y[1] | s[7] | 14.389 | SLOW | 5.137 | FAST |
| y[2] | cout | 11.145 | SLOW | 3.501 | FAST |
| y[2] | s[2] | 9.620 | SLOW | 3.018 | FAST |
| y[2] | s[3] | 8.449 | SLOW | 2.510 | FAST |
| y[2] | s[4] | 9.174 | SLOW | 2.783 | FAST |
| y[2] | s[5] | 10.018 | SLOW | 3.086 | FAST |
| y[2] | s[6] | 10.772 | SLOW | 3.368 | FAST |
| y[2] | s[7] | 10.763 | SLOW | 3.373 | FAST |
| y[3] | cout | 11.337 | SLOW | 3.624 | FAST |
| y[3] | s[3] | 8.674 | SLOW | 2.627 | FAST |
| y[3] | s[4] | 9.366 | SLOW | 2.906 | FAST |
| y[3] | s[5] | 10.209 | SLOW | 3.210 | FAST |
| y[3] | s[6] | 10.963 | SLOW | 3.491 | FAST |
| y[3] | s[7] | 10.954 | SLOW | 3.496 | FAST |
| y[4] | cout | 12.487 | SLOW | 4.310 | FAST |
| y[4] | s[4] | 10.946 | SLOW | 3.725 | FAST |
| y[4] | s[5] | 11.359 | SLOW | 3.895 | FAST |
| y[4] | s[6] | 12.114 | SLOW | 4.176 | FAST |
| y[4] | s[7] | 12.104 | SLOW | 4.181 | FAST |
| y[5] | cout | 10.434 | SLOW | 3.299 | FAST |



| | | | | | |
|---|---|---|---|---|---|
| y[5] | s[5] | 9.304 | SLOW | 2.887 | FAST |
| y[5] | s[6] | 10.061 | SLOW | 3.166 | FAST |
| y[5] | s[7] | 10.052 | SLOW | 3.171 | FAST |
| y[6] | cout | 8.778 | SLOW | 2.613 | FAST |
| y[6] | s[6] | 8.395 | SLOW | 2.466 | FAST |
| y[6] | s[7] | 8.396 | SLOW | 2.485 | FAST |
| y[7] | cout | 8.351 | SLOW | 2.487 | FAST |
| y[7] | s[7] | 8.003 | SLOW | 2.354 | FAST |

There are maximum delays between the input paths y and output paths count. The greatest delay is between y[0] and the cout path. Carry Lookahead Adder calculates all results almost simultaneously. On the contrary, Ripple Carry Adder performs parallel computation, for subsequent computations it must wait for the previous computation to finish. There are not any differences for delay time for this code. However, had their input been greater, perhaps we would have achieved different results.

## BEHAVIORAL ADDER

### Design Sources

Behavioral Adder design source code

```
module Behav_adder #(parameter SIZE = 8)
(
    input [SIZE-1:0] x,
    input [SIZE-1:0] y,
    output cout,
    output [SIZE-1:0] sum
);
    assign {cout,sum} = x + y;

endmodule
```

## Simulation Sources

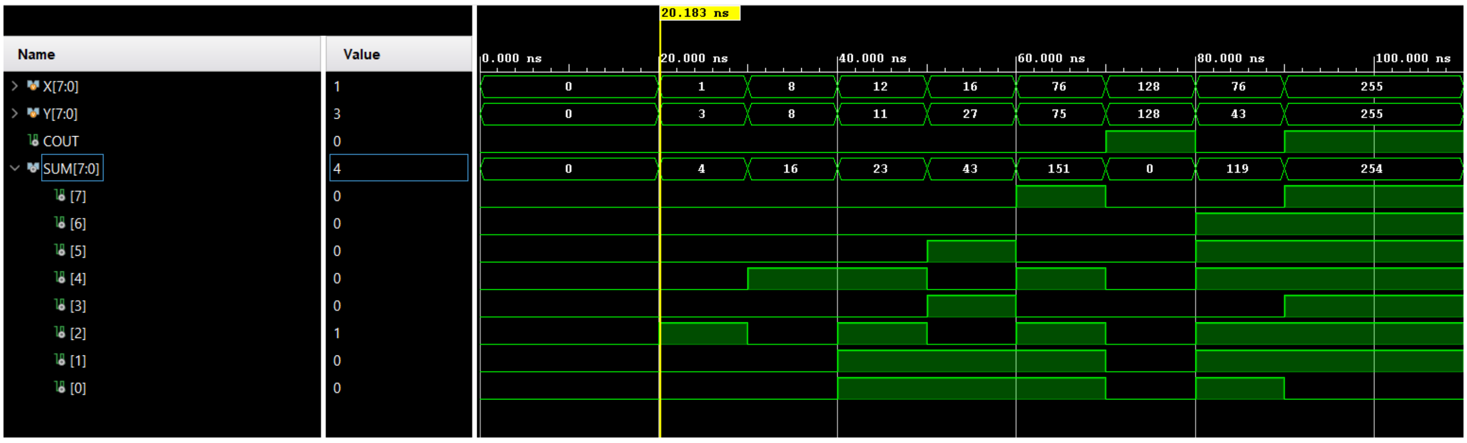
Behavioral Adder simulation source codes

```verilog
module Behav_adder_tb();


parameter SIZE = 8;

    reg [SIZE-1:0] X;
    reg [SIZE-1:0] Y;
    wire COUT;
    wire [SIZE-1:0] SUM;

    Behav_adder #(.SIZE(SIZE)) uut
    (
        .x(X),
        .y(Y),
        .cout(COUT),
        .sum(SUM)
    );

    initial
    begin
        X = 8'b00000000;
        Y = 8'b00000000;
        #10;

        X = 8'b00000000;
        Y = 8'b00000000;
        #10;


        X = 8'b00000001;
        Y = 8'b00000011;
        #10;


        X = 8'b00001000;
        Y = 8'b00001000;
        #10;
```

```verilog
        X = 8'b00001100;
        Y = 8'b00001011;
        #10;


        X = 8'b00010000;
        Y = 8'b00011011;
        #10;


        X = 8'b001001100;
        Y = 8'b01001011;
        #10;


        X = 8'b10000000;
        Y = 8'b10000000;
        #10;

        X = 8'b01001100;
        Y = 8'b00101011;
        #10;

        X = 8'b11111111;
        Y = 8'b11111111;
        #10;

        X = 8'b11111111;
        Y = 8'b11111111;
        #10;

        $finish;
    end
endmodule
```

## Simulation Wave

Behavioral simulation wave screenshot

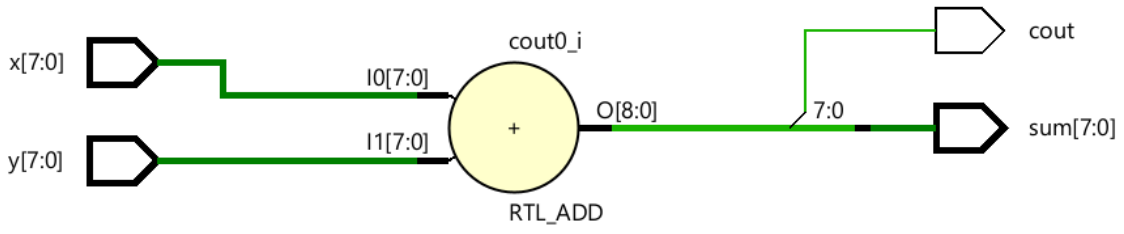


```
Name       Value
X[7:0]     1      0 | 1 | 8  | 12 | 16 | 76  | 128 | 76  | 255
Y[7:0]     3      0 | 3 | 8  | 11 | 27 | 75  | 128 | 43  | 255
COUT       0
SUM[7:0]   4      0 | 4 | 16 | 23 | 43 | 151 | 0   | 119 | 254
[7]        0
[6]        0
[5]        0
[4]        0
[3]        0
[2]        1
[1]        0
[0]        0
```

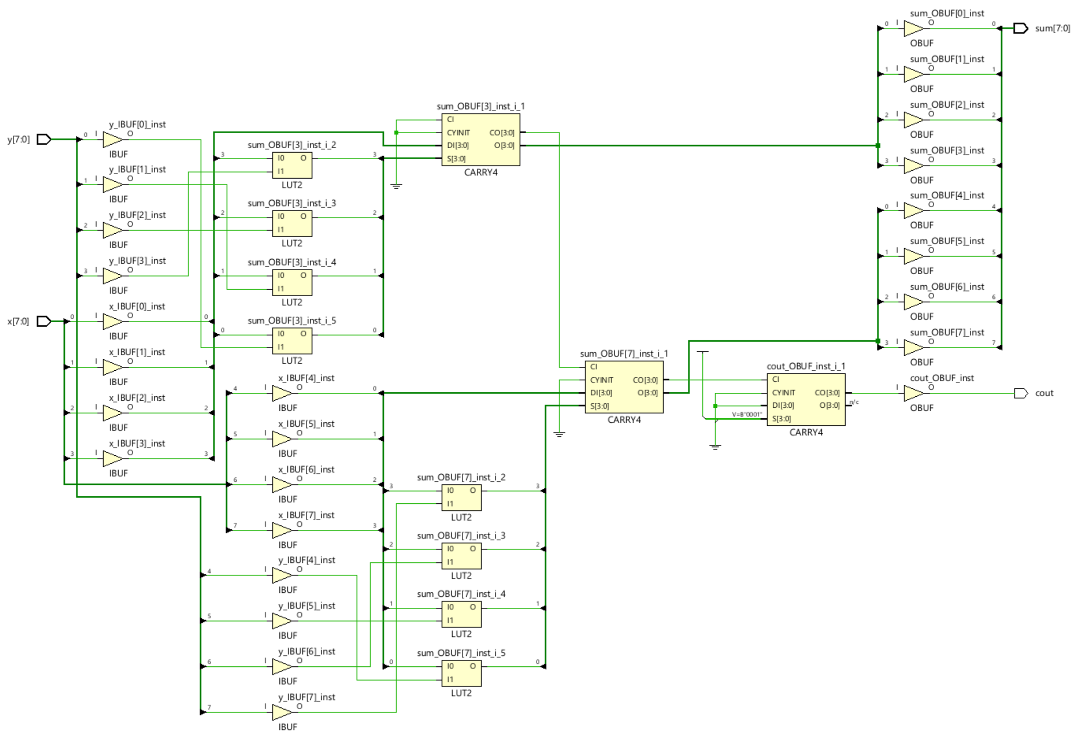
As seen in the simulation, the module works as it should. There are not any differences between parametric ripple carry adder with behavioral adder between results. There are differences in the algorithm for these two adders.

Salih Ömer Ongün
040220780

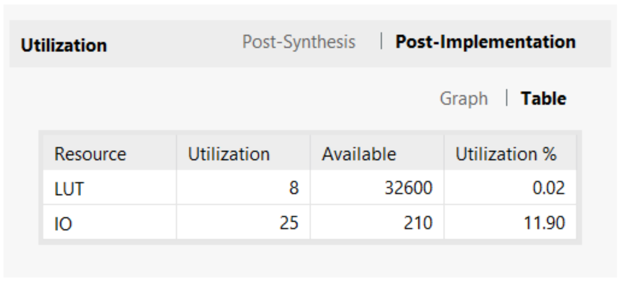
## RTL Schematic



## Technology Schematic



## Utilization Report

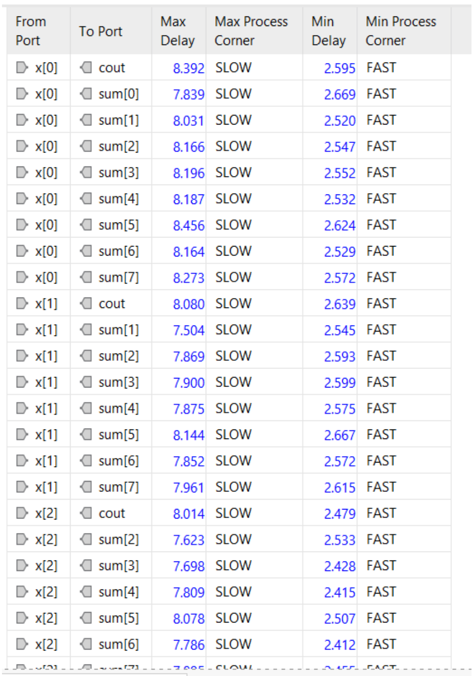
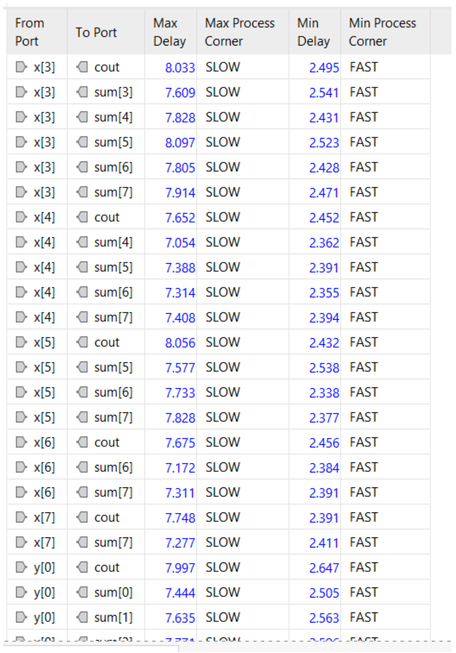
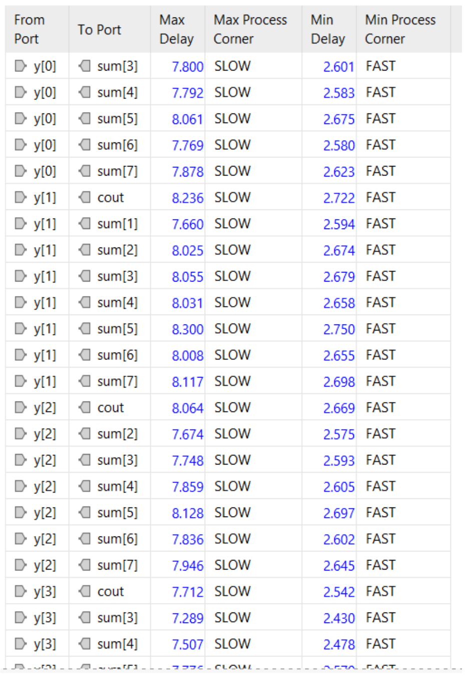
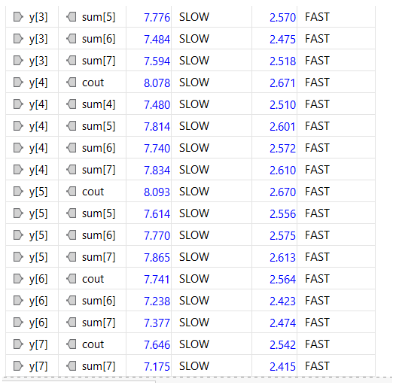


Utilization    Post-Synthesis | **Post-Implementation**

Graph | **Table**

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 8 | 32600 | 0.02 |
| IO | 25 | 210 | 11.90 |

# Timing Report

| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| x[0] | cout | 8.392 | SLOW | 2.595 | FAST |
| x[0] | sum[0] | 7.839 | SLOW | 2.669 | FAST |
| x[0] | sum[1] | 8.031 | SLOW | 2.520 | FAST |
| x[0] | sum[2] | 8.166 | SLOW | 2.547 | FAST |
| x[0] | sum[3] | 8.196 | SLOW | 2.552 | FAST |
| x[0] | sum[4] | 8.187 | SLOW | 2.532 | FAST |
| x[0] | sum[5] | 8.456 | SLOW | 2.624 | FAST |
| x[0] | sum[6] | 8.164 | SLOW | 2.529 | FAST |
| x[0] | sum[7] | 8.273 | SLOW | 2.572 | FAST |
| x[1] | cout | 8.080 | SLOW | 2.639 | FAST |
| x[1] | sum[1] | 7.504 | SLOW | 2.545 | FAST |
| x[1] | sum[2] | 7.869 | SLOW | 2.593 | FAST |
| x[1] | sum[3] | 7.900 | SLOW | 2.599 | FAST |
| x[1] | sum[4] | 7.875 | SLOW | 2.575 | FAST |
| x[1] | sum[5] | 8.144 | SLOW | 2.667 | FAST |
| x[1] | sum[6] | 7.852 | SLOW | 2.572 | FAST |
| x[1] | sum[7] | 7.961 | SLOW | 2.615 | FAST |
| x[2] | cout | 8.014 | SLOW | 2.479 | FAST |
| x[2] | sum[2] | 7.623 | SLOW | 2.533 | FAST |
| x[2] | sum[3] | 7.698 | SLOW | 2.428 | FAST |
| x[2] | sum[4] | 7.809 | SLOW | 2.415 | FAST |
| x[2] | sum[5] | 8.078 | SLOW | 2.507 | FAST |
| x[2] | sum[6] | 7.786 | SLOW | 2.412 | FAST |
| x[3] | cout | 8.033 | SLOW | 2.495 | FAST |
| x[3] | sum[3] | 7.609 | SLOW | 2.541 | FAST |
| x[3] | sum[4] | 7.828 | SLOW | 2.431 | FAST |
| x[3] | sum[5] | 8.097 | SLOW | 2.523 | FAST |
| x[3] | sum[6] | 7.805 | SLOW | 2.428 | FAST |
| x[3] | sum[7] | 7.914 | SLOW | 2.471 | FAST |
| x[4] | cout | 7.652 | SLOW | 2.452 | FAST |
| x[4] | sum[4] | 7.054 | SLOW | 2.362 | FAST |
| x[4] | sum[5] | 7.388 | SLOW | 2.391 | FAST |
| x[4] | sum[6] | 7.314 | SLOW | 2.355 | FAST |
| x[4] | sum[7] | 7.408 | SLOW | 2.394 | FAST |
| x[5] | cout | 8.056 | SLOW | 2.432 | FAST |
| x[5] | sum[5] | 7.577 | SLOW | 2.538 | FAST |
| x[5] | sum[6] | 7.733 | SLOW | 2.338 | FAST |
| x[5] | sum[7] | 7.828 | SLOW | 2.377 | FAST |
| x[6] | cout | 7.675 | SLOW | 2.456 | FAST |
| x[6] | sum[6] | 7.172 | SLOW | 2.384 | FAST |
| x[6] | sum[7] | 7.311 | SLOW | 2.391 | FAST |
| x[7] | cout | 7.748 | SLOW | 2.391 | FAST |
| x[7] | sum[7] | 7.277 | SLOW | 2.411 | FAST |
| y[0] | cout | 7.997 | SLOW | 2.647 | FAST |
| y[0] | sum[0] | 7.444 | SLOW | 2.505 | FAST |
| y[0] | sum[1] | 7.635 | SLOW | 2.563 | FAST |
| y[0] | sum[3] | 7.800 | SLOW | 2.601 | FAST |
| y[0] | sum[4] | 7.792 | SLOW | 2.583 | FAST |
| y[0] | sum[5] | 8.061 | SLOW | 2.675 | FAST |
| y[0] | sum[6] | 7.769 | SLOW | 2.580 | FAST |
| y[0] | sum[7] | 7.878 | SLOW | 2.623 | FAST |
| y[1] | cout | 8.236 | SLOW | 2.722 | FAST |
| y[1] | sum[1] | 7.660 | SLOW | 2.594 | FAST |
| y[1] | sum[2] | 8.025 | SLOW | 2.674 | FAST |
| y[1] | sum[3] | 8.055 | SLOW | 2.679 | FAST |
| y[1] | sum[4] | 8.031 | SLOW | 2.658 | FAST |
| y[1] | sum[5] | 8.300 | SLOW | 2.750 | FAST |
| y[1] | sum[6] | 8.008 | SLOW | 2.655 | FAST |
| y[1] | sum[7] | 8.117 | SLOW | 2.698 | FAST |
| y[2] | cout | 8.064 | SLOW | 2.669 | FAST |
| y[2] | sum[2] | 7.674 | SLOW | 2.575 | FAST |
| y[2] | sum[3] | 7.748 | SLOW | 2.593 | FAST |
| y[2] | sum[4] | 7.859 | SLOW | 2.605 | FAST |
| y[2] | sum[5] | 8.128 | SLOW | 2.697 | FAST |
| y[2] | sum[6] | 7.836 | SLOW | 2.602 | FAST |
| y[2] | sum[7] | 7.946 | SLOW | 2.645 | FAST |
| y[3] | cout | 7.712 | SLOW | 2.542 | FAST |
| y[3] | sum[3] | 7.289 | SLOW | 2.430 | FAST |
| y[3] | sum[4] | 7.507 | SLOW | 2.478 | FAST |
| y[3] | sum[5] | 7.776 | SLOW | 2.570 | FAST |
| y[3] | sum[6] | 7.484 | SLOW | 2.475 | FAST |
| y[3] | sum[7] | 7.594 | SLOW | 2.518 | FAST |
| y[4] | cout | 8.078 | SLOW | 2.671 | FAST |
| y[4] | sum[4] | 7.480 | SLOW | 2.510 | FAST |
| y[4] | sum[5] | 7.814 | SLOW | 2.601 | FAST |
| y[4] | sum[6] | 7.740 | SLOW | 2.572 | FAST |
| y[4] | sum[7] | 7.834 | SLOW | 2.610 | FAST |
| y[5] | cout | 8.093 | SLOW | 2.670 | FAST |
| y[5] | sum[5] | 7.614 | SLOW | 2.556 | FAST |
| y[5] | sum[6] | 7.770 | SLOW | 2.575 | FAST |
| y[5] | sum[7] | 7.865 | SLOW | 2.613 | FAST |
| y[6] | cout | 7.741 | SLOW | 2.564 | FAST |
| y[6] | sum[6] | 7.238 | SLOW | 2.423 | FAST |
| y[6] | sum[7] | 7.377 | SLOW | 2.474 | FAST |
| y[7] | cout | 7.646 | SLOW | 2.542 | FAST |
| y[7] | sum[7] | 7.175 | SLOW | 2.415 | FAST |

There are different algorithms. Vivado includes these algorithms. If we apply any of these algorithms, Vivado will perform the design with the algorithms we applied. If we use arithmetic operators like add (+) for input, Vivado will choose the most optimal algorithm. The algorithm has optimum delay, power usage and resource usage for our code.  Therefore, Behavioral Adder has the minimum delays in all paths. It uses fewer I/O ports than other algorithms.

## DONT_TOUCH Implementation

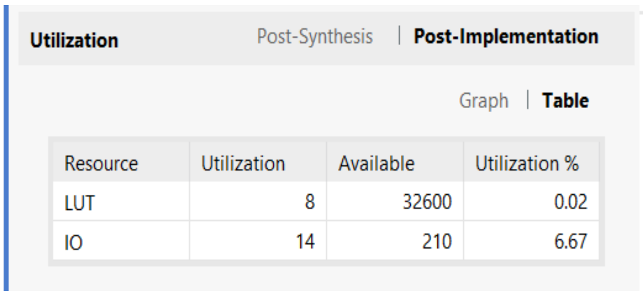
Design Source Code

```
(* DONT_TOUCH = "yes" *)
module Behav_adder #(parameter SIZE = 8)
(
    input [SIZE-1:0] x,
    input [SIZE-1:0] y,
    output cout,
    output [SIZE-1:0] sum
);
    assign {cout,sum} = x + y;

endmodule
```

## Behavioral Adder

There are not any differences in RTL and Technology Schematics between with and without "DONT_TOUCH". They have same delays and resource usage. A small number of inputs may have resulted in the same results. If we will have greater input, we can see effect of "DONT_TOUCH".

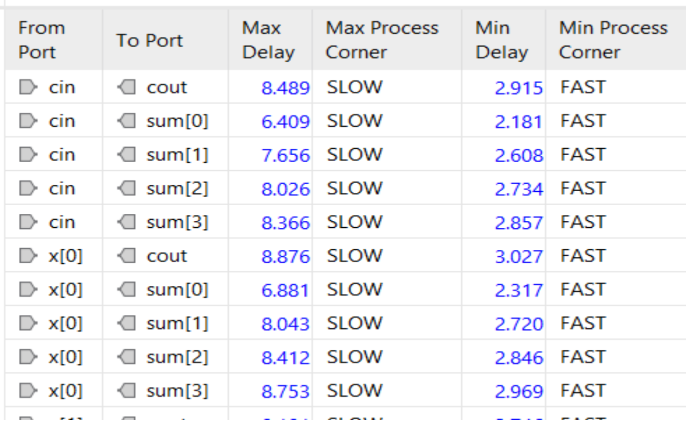
**RCA** With DONT_TOCUH



Utilization Post-Synthesis | **Post-Implementation**

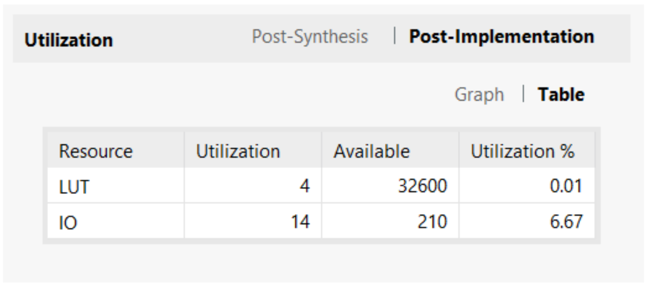
Graph | **Table**

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 8 | 32600 | 0.02 |
| IO | 14 | 210 | 6.67 |



| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| ▷ cin | ◁ cout | 8.489 | SLOW | 2.915 | FAST |
| ▷ cin | ◁ sum[0] | 6.409 | SLOW | 2.181 | FAST |
| ▷ cin | ◁ sum[1] | 7.656 | SLOW | 2.608 | FAST |
| ▷ cin | ◁ sum[2] | 8.026 | SLOW | 2.734 | FAST |
| ▷ cin | ◁ sum[3] | 8.366 | SLOW | 2.857 | FAST |
| ▷ x[0] | ◁ cout | 8.876 | SLOW | 3.027 | FAST |
| ▷ x[0] | ◁ sum[0] | 6.881 | SLOW | 2.317 | FAST |
| ▷ x[0] | ◁ sum[1] | 8.043 | SLOW | 2.720 | FAST |
| ▷ x[0] | ◁ sum[2] | 8.412 | SLOW | 2.846 | FAST |
| ▷ x[0] | ◁ sum[3] | 8.753 | SLOW | 2.969 | FAST |

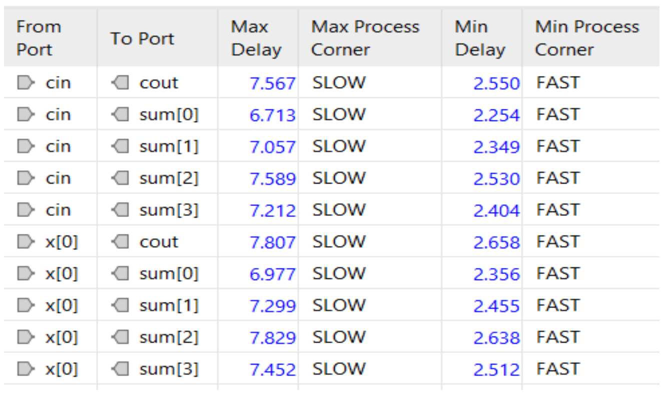
Without DONT_TOCUH



Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 4 | 32600 | 0.01 |
| IO | 14 | 210 | 6.67 |

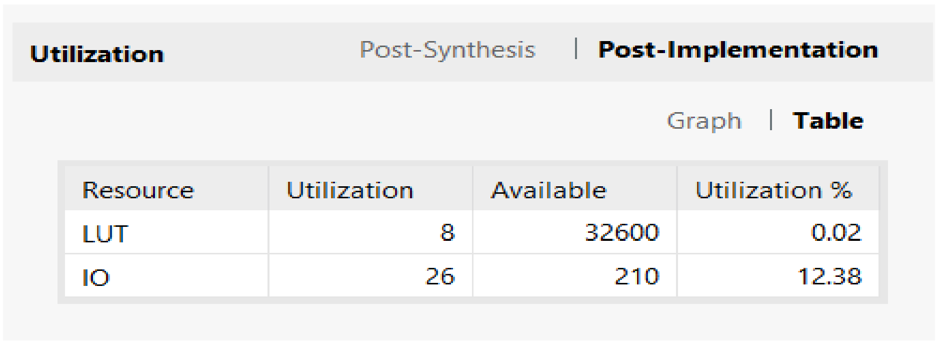


| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| ▷ cin | ◁ cout | 7.567 | SLOW | 2.550 | FAST |
| ▷ cin | ◁ sum[0] | 6.713 | SLOW | 2.254 | FAST |
| ▷ cin | ◁ sum[1] | 7.057 | SLOW | 2.349 | FAST |
| ▷ cin | ◁ sum[2] | 7.589 | SLOW | 2.530 | FAST |
| ▷ cin | ◁ sum[3] | 7.212 | SLOW | 2.404 | FAST |
| ▷ x[0] | ◁ cout | 7.807 | SLOW | 2.658 | FAST |
| ▷ x[0] | ◁ sum[0] | 6.977 | SLOW | 2.356 | FAST |
| ▷ x[0] | ◁ sum[1] | 7.299 | SLOW | 2.455 | FAST |
| ▷ x[0] | ◁ sum[2] | 7.829 | SLOW | 2.638 | FAST |
| ▷ x[0] | ◁ sum[3] | 7.452 | SLOW | 2.512 | FAST |

There are differences between with and without DONT_TOUCH. Without DONT_TOUCH implementation have less delays and less resource usage.

**CLA**

Without DONT_TOUCH

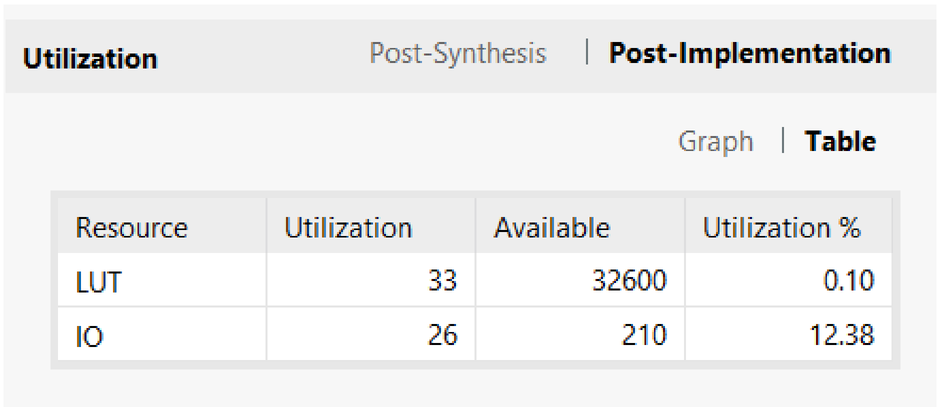


**Utilization** Post-Synthesis | **Post-Implementation**

Graph | **Table**

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 8 | 32600 | 0.02 |
| IO | 26 | 210 | 12.38 |

With DONT_TOUCH



**Utilization** Post-Synthesis | **Post-Implementation**

Graph | **Table**

| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 33 | 32600 | 0.10 |
| IO | 26 | 210 | 12.38 |

There aren't any significant differences in terms of delay for with and without DON'T_TOUCH CLA implementation. However, there are big differences in terms of resource usage. Without the DON'T_TOUCH implementation, much less resources are used.

# ADDER – SUBTRACTOR CIRCUIT

## Design Sources

Adder – Subtractor Circuit design source code

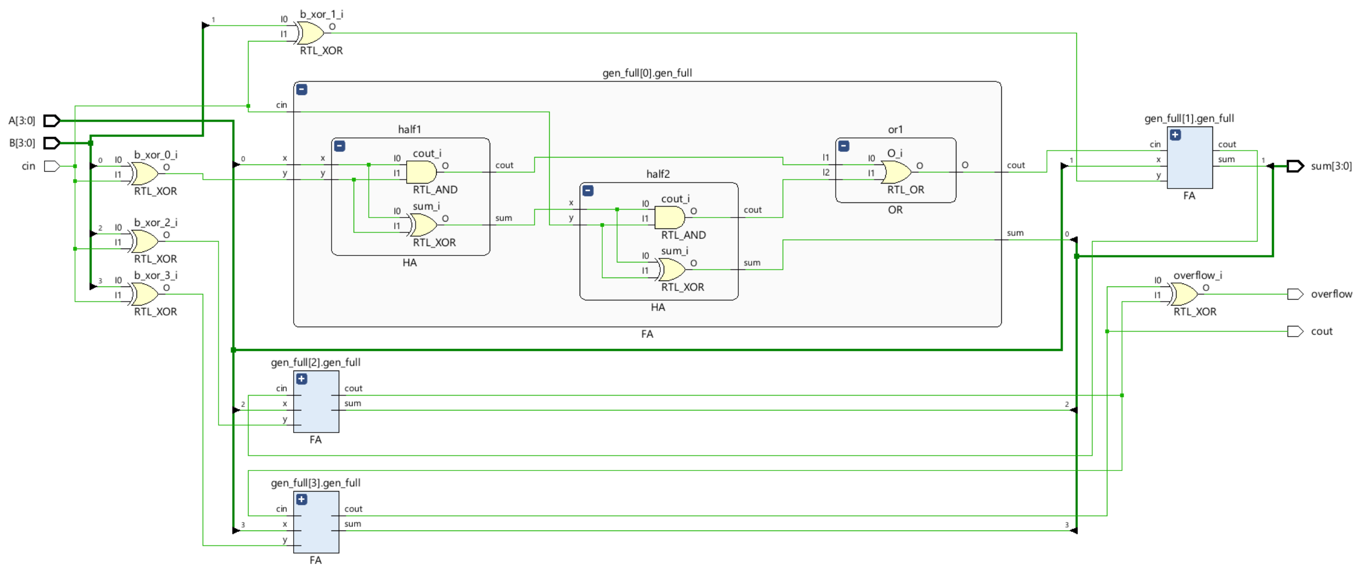
```
module Add_Sub
(
    input [3:0] A,
    input [3:0] B,
    input cin,
    output [3:0] sum,
    output cout,
    output overflow
);

    wire [3:0] b_xor;
    wire [4:0] c_gen;
    assign c_gen[0] = cin;

    genvar i;


    generate
        for(i = 0; i<4; i=i+1) begin: gen_xor
            assign b_xor[i] = B[i] ^ cin ;
        end

        for(i = 0; i<4; i=i+1) begin: gen_full
            FA gen_full
            (
                A[i],
                b_xor[i],
                c_gen[i],
                c_gen[i+1],
                sum[i]
            );
        end
    endgenerate

    assign cout = c_gen[4];

    assign overflow = c_gen[4] ^ c_gen[3];
endmodule
```

## Simulation Sources

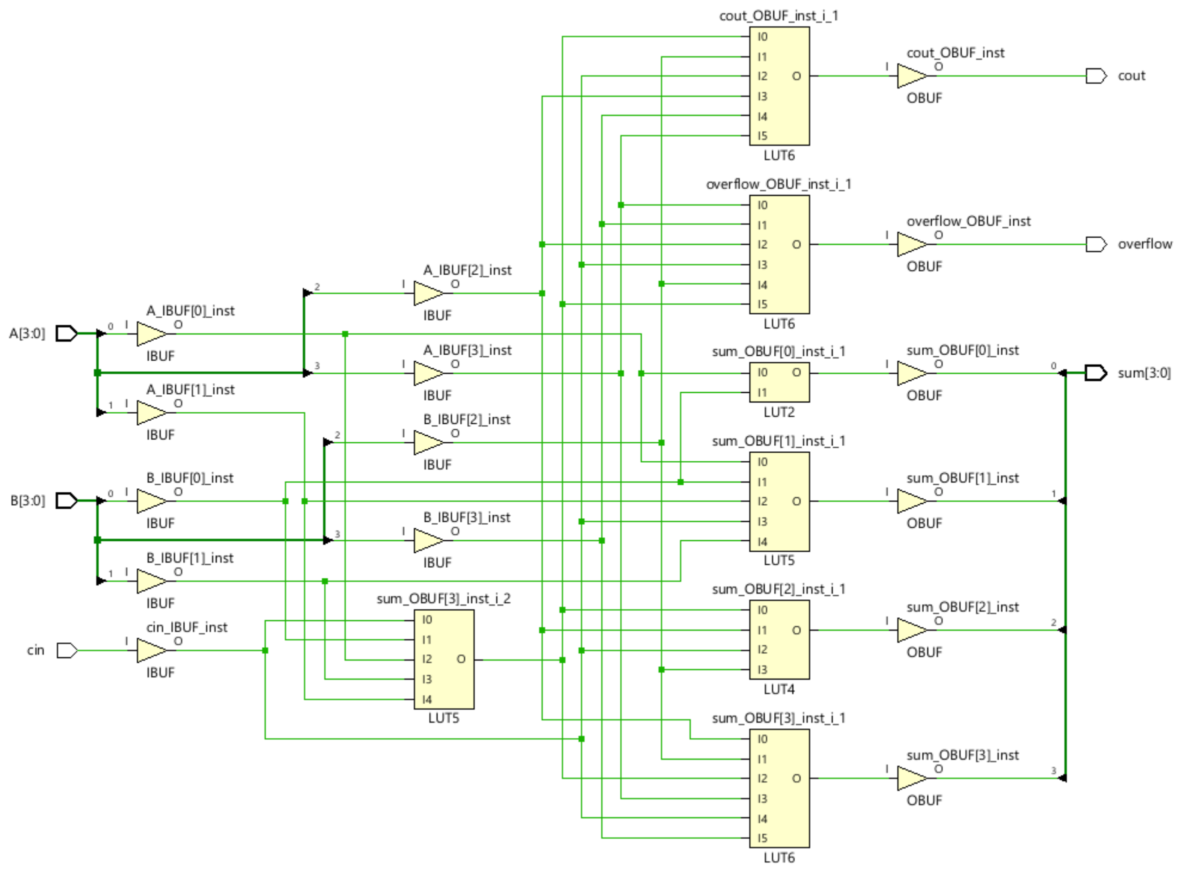
Adder – Subtractor Circuit simulation source code

```
module Add_Sub_tb();

    reg [3:0] A= 4'b0;
    reg [3:0] B= 4'b0;
    reg CIN= 1'b0;
    wire COUT;
    wire [3:0] SUM;
    wire OVERFLOW;

    Add_Sub uut
    (
        .A(A),
        .B(B),
        .cin(CIN),
        .cout(COUT),
        .overflow(OVERFLOW),
        .sum(SUM)
    );
    integer i;
    integer k;
    integer j;

    initial
    begin
        for(k = 0; k<2; k = k+1) begin
          CIN = k;
            for(i = -8; i<8; i = i+1) begin
             A = i;
                for(j = -8; j<8; j = j+1) begin
                    B = j;
                    #5;
                end
            end
        end
        $finish;
    end
endmodule
```

## RTL Schematic

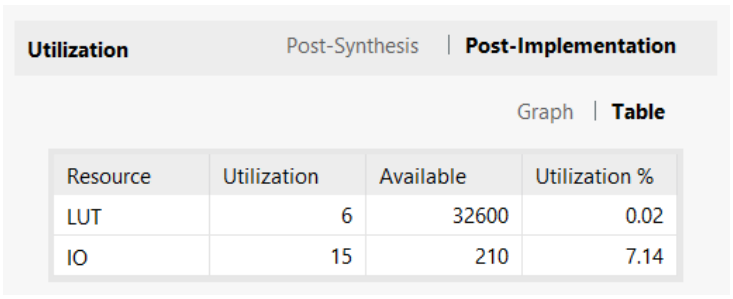


## Technology Schematic



Salih Ömer Ongün
040220780

## Utilization Report

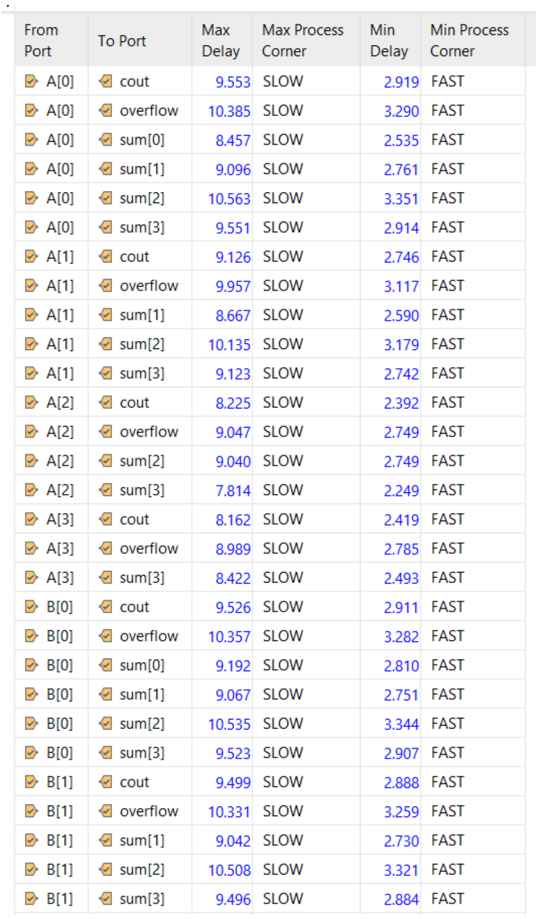
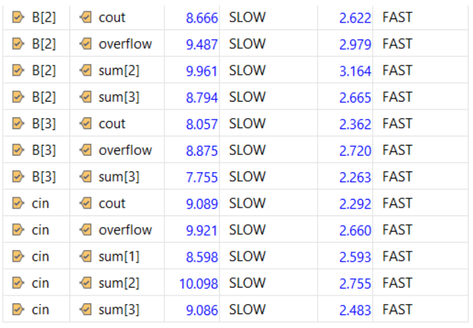


**Utilization** Post-Synthesis | **Post-Implementation**

Graph | **Table**

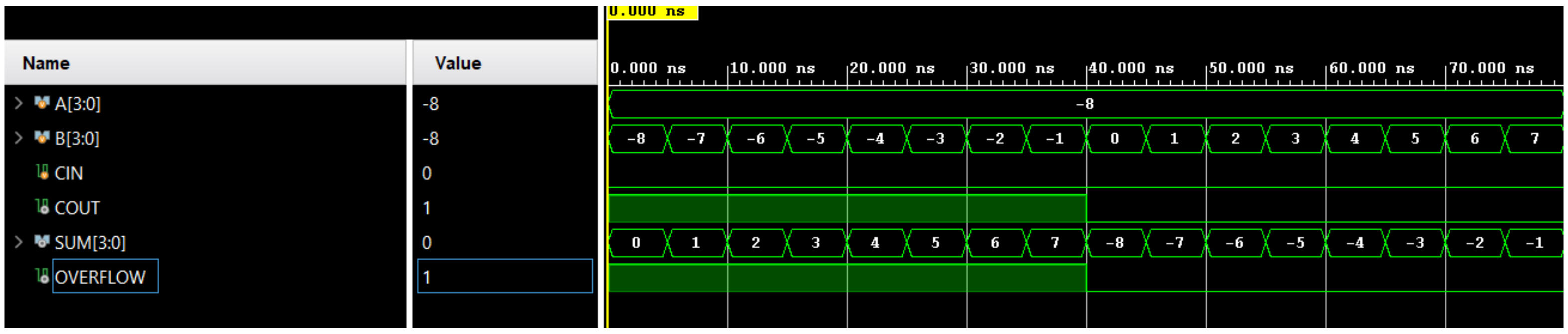
| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 6 | 32600 | 0.02 |
| IO | 15 | 210 | 7.14 |

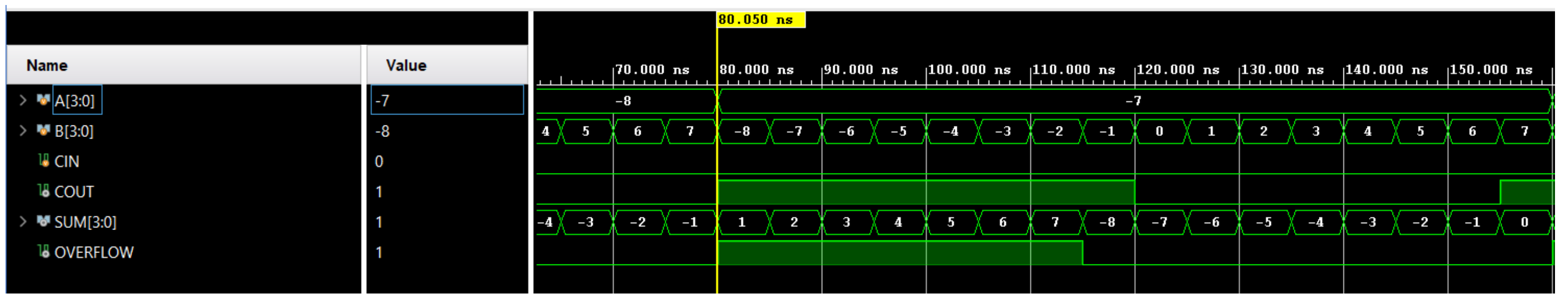
## Timing Report

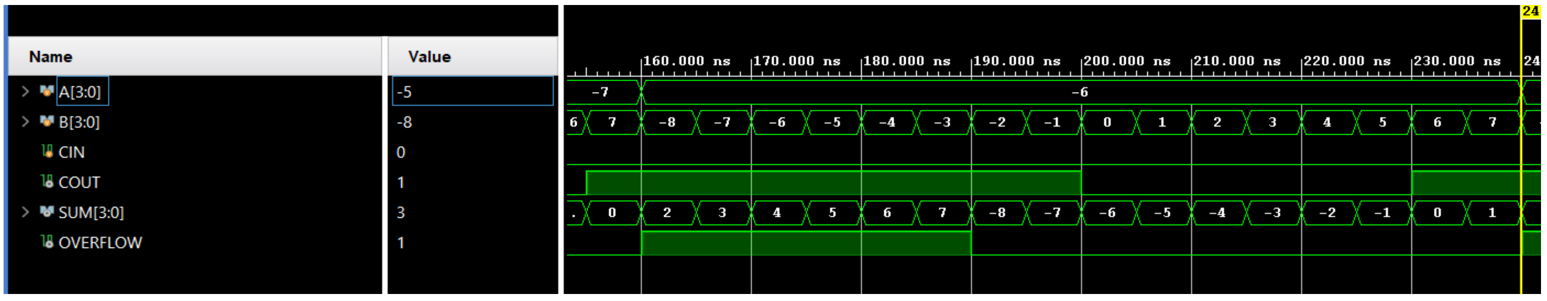



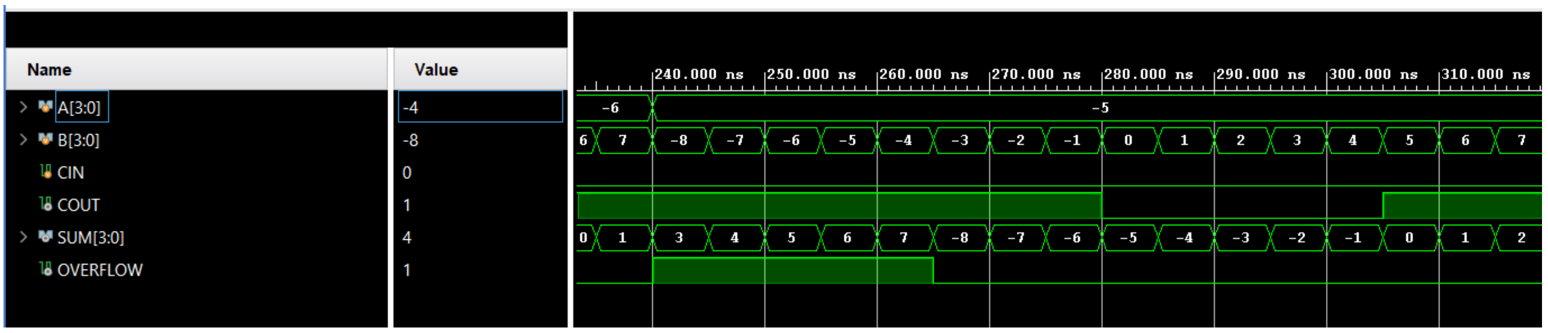
| From Port | To Port | Max Delay | Max Process Corner | Min Delay | Min Process Corner |
|---|---|---|---|---|---|
| A[0] | cout | 9.553 | SLOW | 2.919 | FAST |
| A[0] | overflow | 10.385 | SLOW | 3.290 | FAST |
| A[0] | sum[0] | 8.457 | SLOW | 2.535 | FAST |
| A[0] | sum[1] | 9.096 | SLOW | 2.761 | FAST |
| A[0] | sum[2] | 10.563 | SLOW | 3.351 | FAST |
| A[0] | sum[3] | 9.551 | SLOW | 2.914 | FAST |
| A[1] | cout | 9.126 | SLOW | 2.746 | FAST |
| A[1] | overflow | 9.957 | SLOW | 3.117 | FAST |
| A[1] | sum[1] | 8.667 | SLOW | 2.590 | FAST |
| A[1] | sum[2] | 10.135 | SLOW | 3.179 | FAST |
| A[1] | sum[3] | 9.123 | SLOW | 2.742 | FAST |
| A[2] | cout | 8.225 | SLOW | 2.392 | FAST |
| A[2] | overflow | 9.047 | SLOW | 2.749 | FAST |
| A[2] | sum[2] | 9.040 | SLOW | 2.749 | FAST |
| A[2] | sum[3] | 7.814 | SLOW | 2.249 | FAST |
| A[3] | cout | 8.162 | SLOW | 2.419 | FAST |
| A[3] | overflow | 8.989 | SLOW | 2.785 | FAST |
| A[3] | sum[3] | 8.422 | SLOW | 2.493 | FAST |
| B[0] | cout | 9.526 | SLOW | 2.911 | FAST |
| B[0] | overflow | 10.357 | SLOW | 3.282 | FAST |
| B[0] | sum[0] | 9.192 | SLOW | 2.810 | FAST |
| B[0] | sum[1] | 9.067 | SLOW | 2.751 | FAST |
| B[0] | sum[2] | 10.535 | SLOW | 3.344 | FAST |
| B[0] | sum[3] | 9.523 | SLOW | 2.907 | FAST |
| B[1] | cout | 9.499 | SLOW | 2.888 | FAST |
| B[1] | overflow | 10.331 | SLOW | 3.259 | FAST |
| B[1] | sum[1] | 9.042 | SLOW | 2.730 | FAST |
| B[1] | sum[2] | 10.508 | SLOW | 3.321 | FAST |
| B[1] | sum[3] | 9.496 | SLOW | 2.884 | FAST |
| B[2] | cout | 8.666 | SLOW | 2.622 | FAST |
| B[2] | overflow | 9.487 | SLOW | 2.979 | FAST |
| B[2] | sum[2] | 9.961 | SLOW | 3.164 | FAST |
| B[2] | sum[3] | 8.794 | SLOW | 2.665 | FAST |
| B[3] | cout | 8.057 | SLOW | 2.362 | FAST |
| B[3] | overflow | 8.875 | SLOW | 2.720 | FAST |
| B[3] | sum[3] | 7.755 | SLOW | 2.263 | FAST |
| cin | cout | 9.089 | SLOW | 2.292 | FAST |
| cin | overflow | 9.921 | SLOW | 2.660 | FAST |
| cin | sum[1] | 8.598 | SLOW | 2.593 | FAST |
| cin | sum[2] | 10.098 | SLOW | 2.755 | FAST |
| cin | sum[3] | 9.086 | SLOW | 2.483 | FAST |

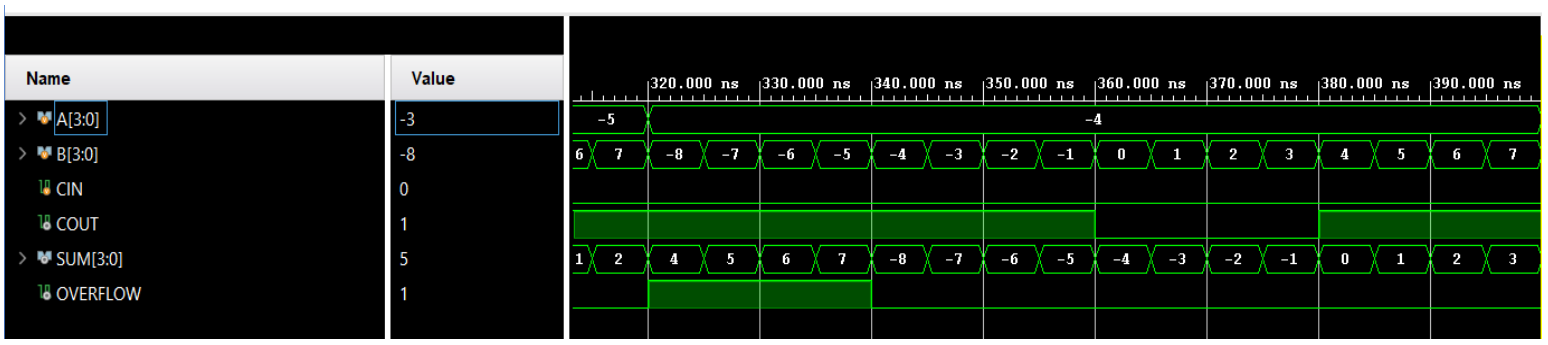
## Simulation Waves

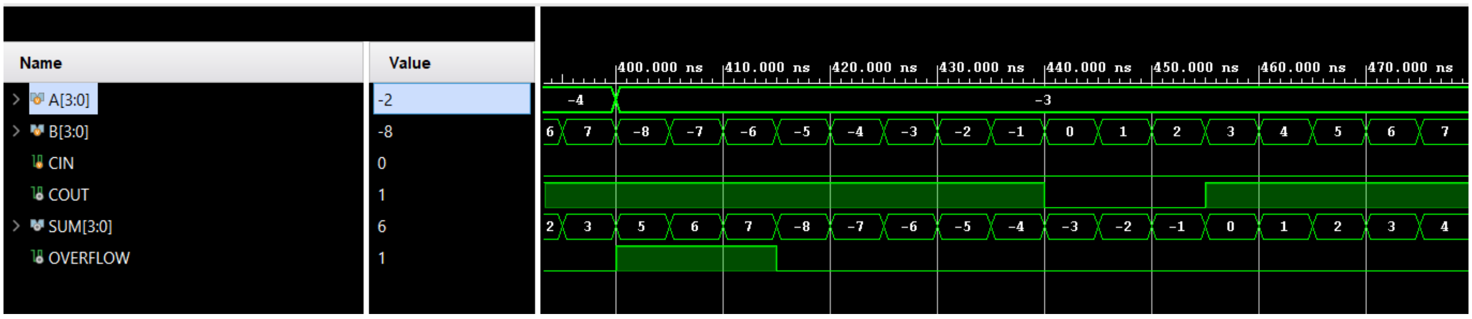




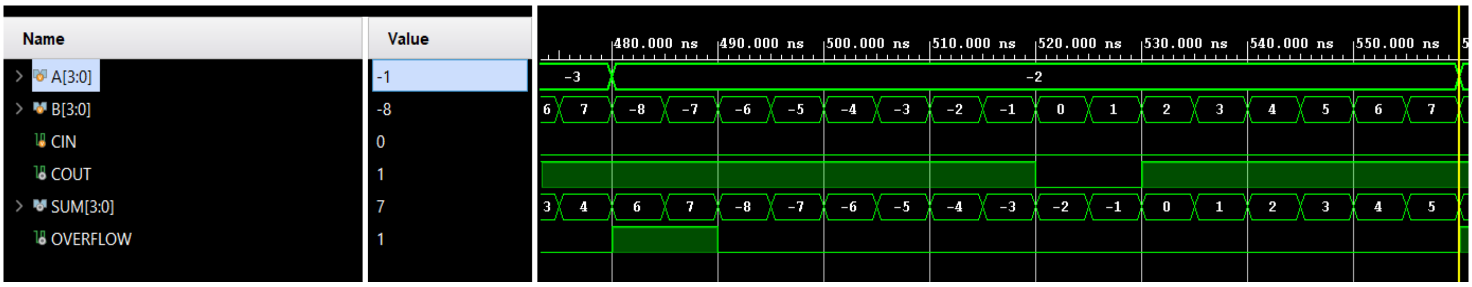




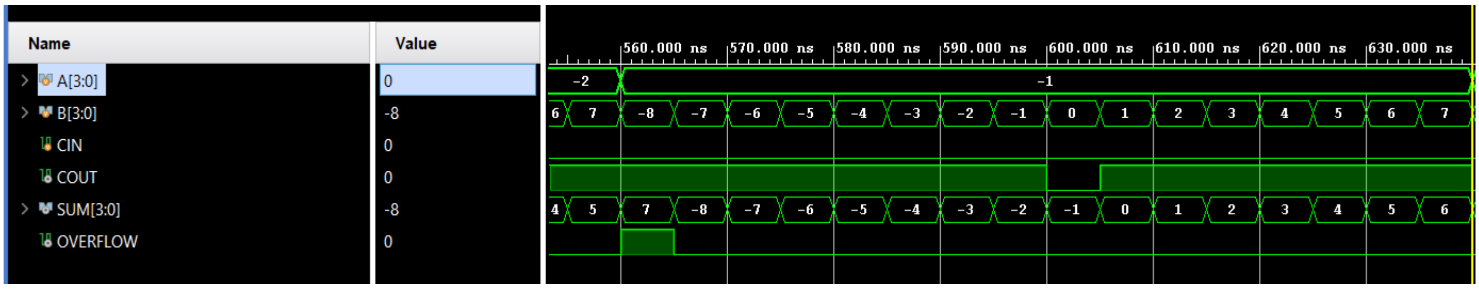




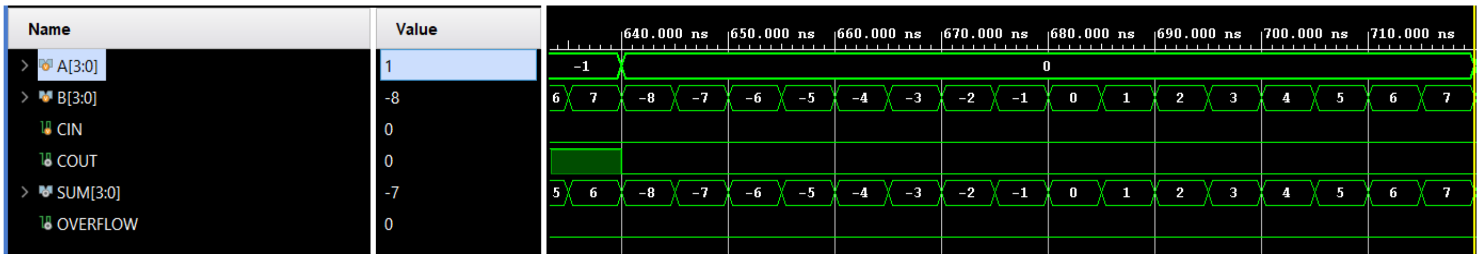
| Name | Value |
|---|---|
| A[3:0] | -2 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 1 |
| SUM[3:0] | 6 |
| OVERFLOW | 1 |



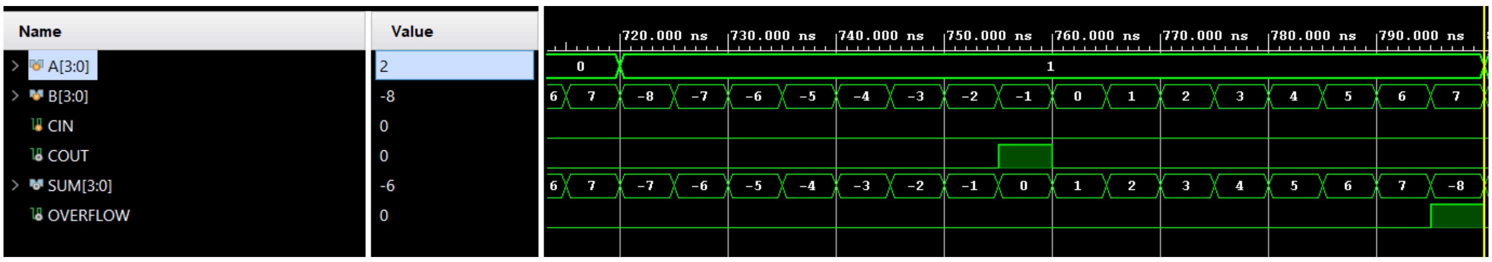
| Name | Value |
|---|---|
| A[3:0] | -1 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 1 |
| SUM[3:0] | 7 |
| OVERFLOW | 1 |



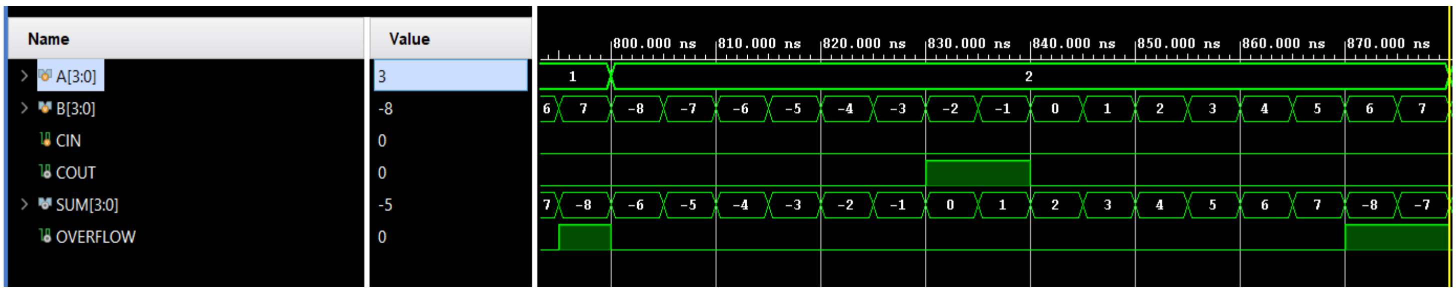
| Name | Value |
|---|---|
| A[3:0] | 0 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -8 |
| OVERFLOW | 0 |



| Name | Value |
|---|---|
| A[3:0] | 1 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -7 |
| OVERFLOW | 0 |

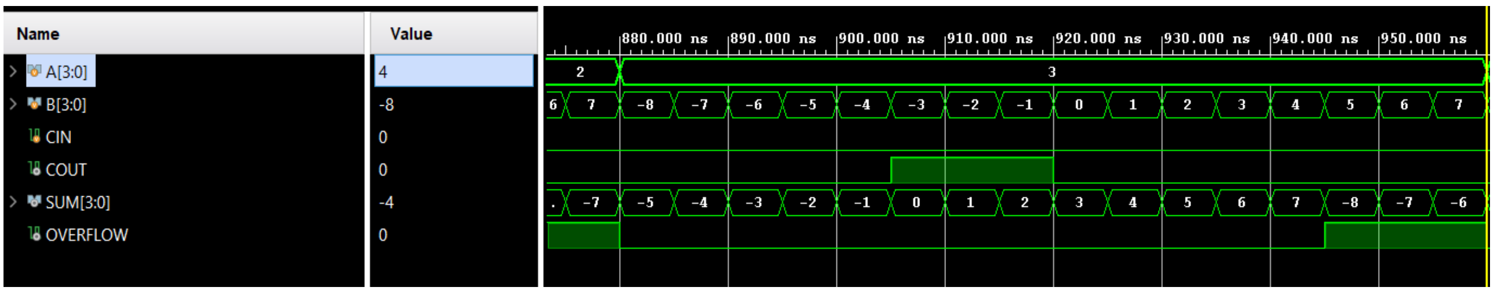


| Name | Value |
|---|---|
| A[3:0] | 2 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -6 |
| OVERFLOW | 0 |

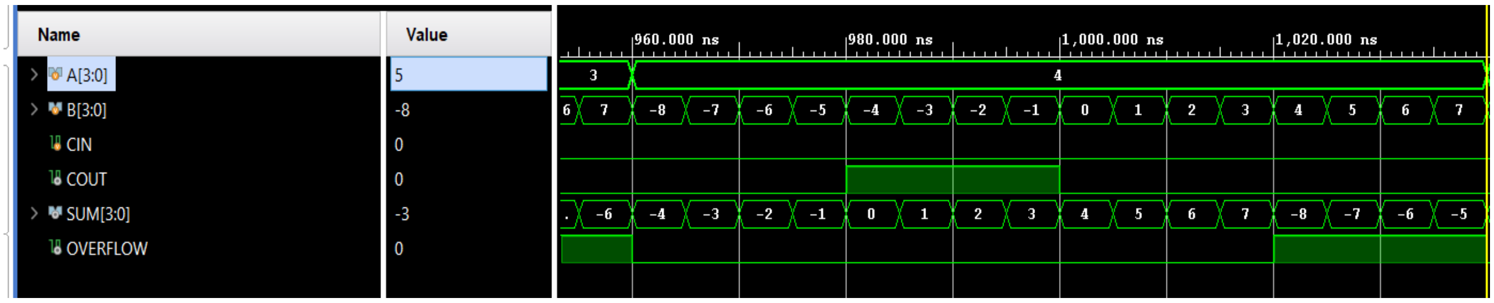


| Name | Value |
|---|---|
| A[3:0] | 3 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -5 |
| OVERFLOW | 0 |

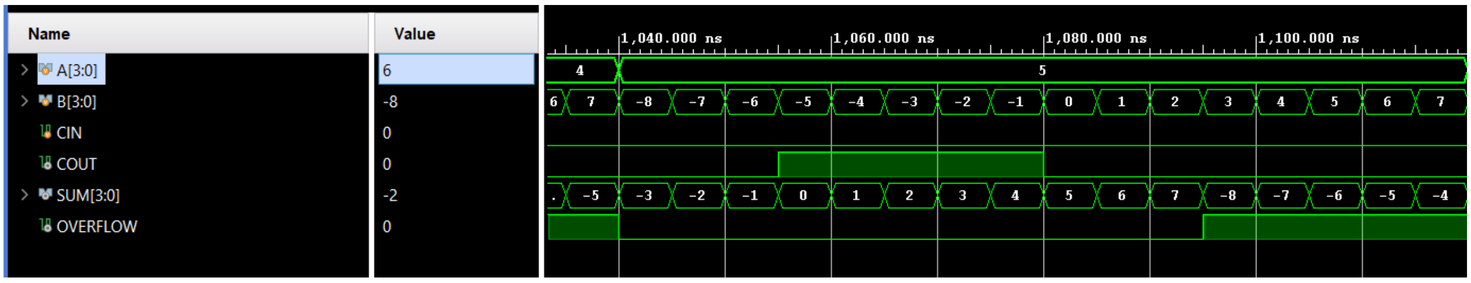
Salih Ömer Ongün
040220780



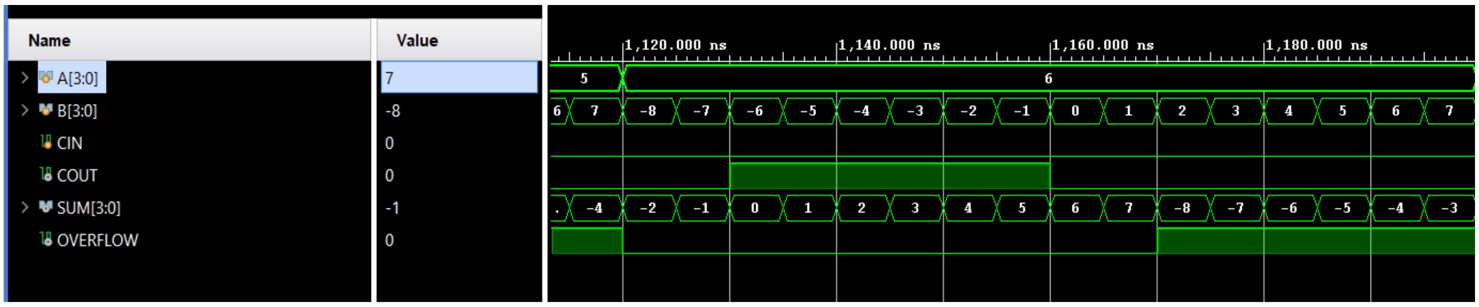
| Name | Value |
|---|---|
| A[3:0] | 4 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -4 |
| OVERFLOW | 0 |



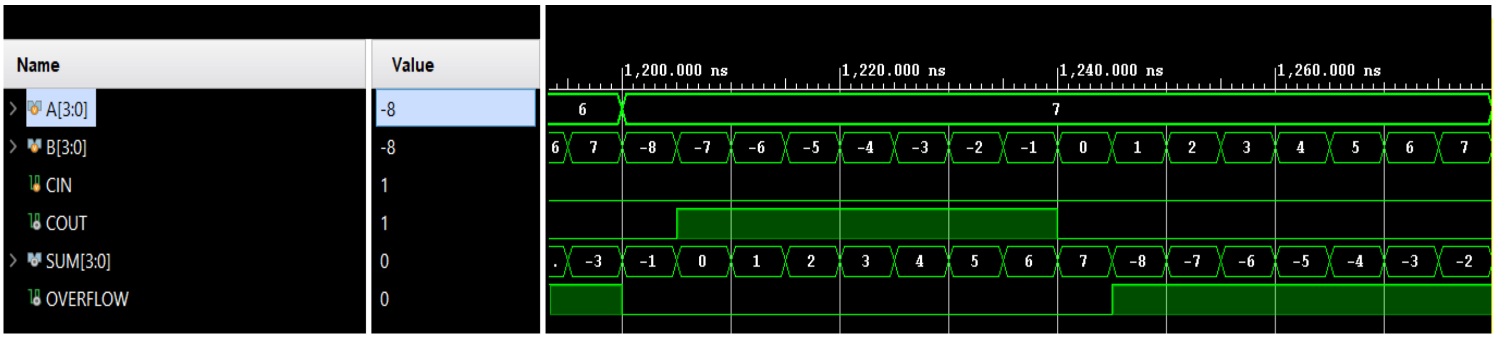
| Name | Value |
|---|---|
| A[3:0] | 5 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -3 |
| OVERFLOW | 0 |



| Name | Value |
|---|---|
| A[3:0] | 6 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -2 |
| OVERFLOW | 0 |

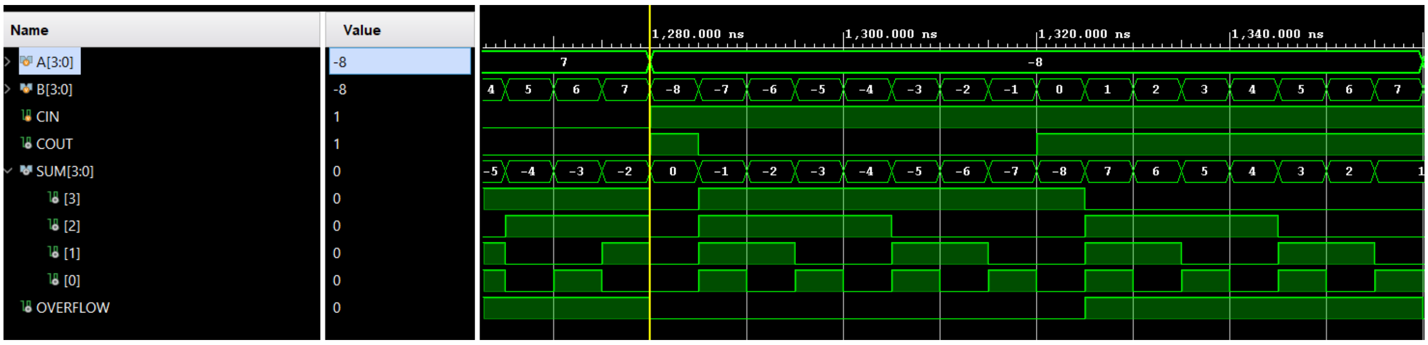


| Name | Value |
|---|---|
| A[3:0] | 7 |
| B[3:0] | -8 |
| CIN | 0 |
| COUT | 0 |
| SUM[3:0] | -1 |
| OVERFLOW | 0 |

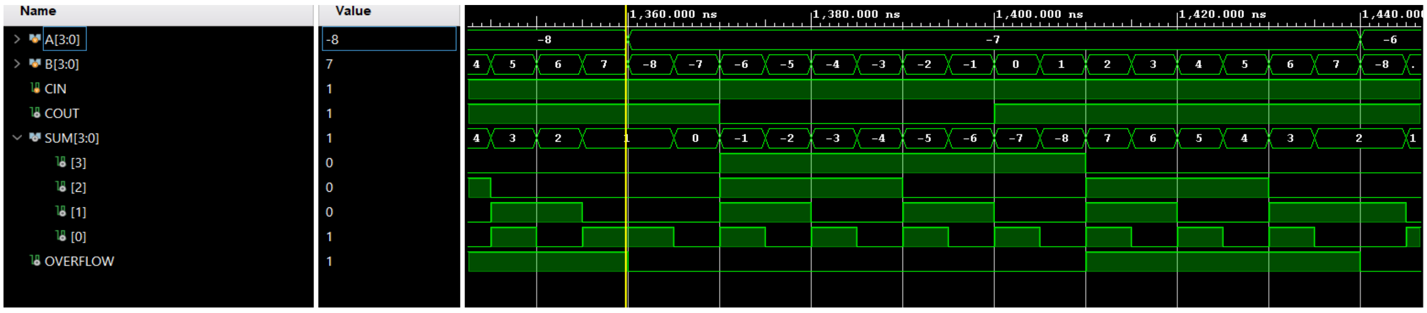


| Name | Value |
|---|---|
| A[3:0] | -8 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 0 |
| OVERFLOW | 0 |

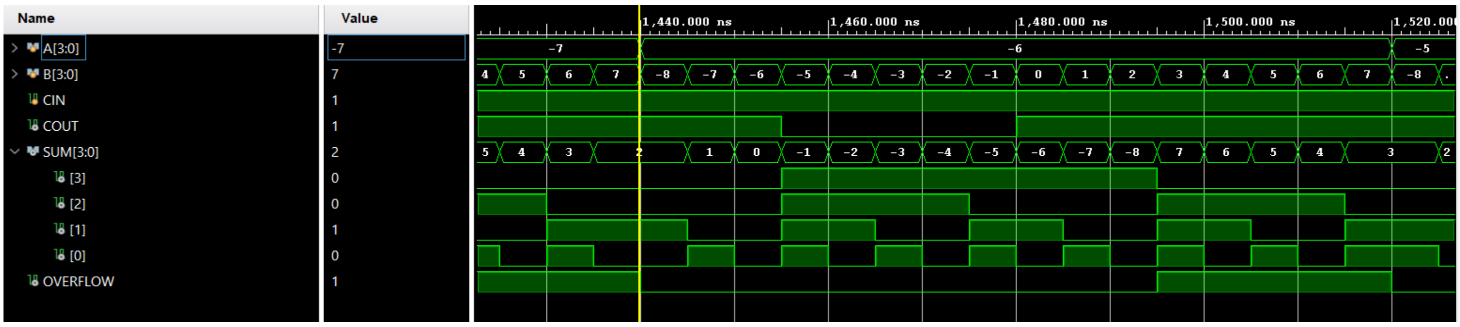
Salih Ömer Ongün
040220780



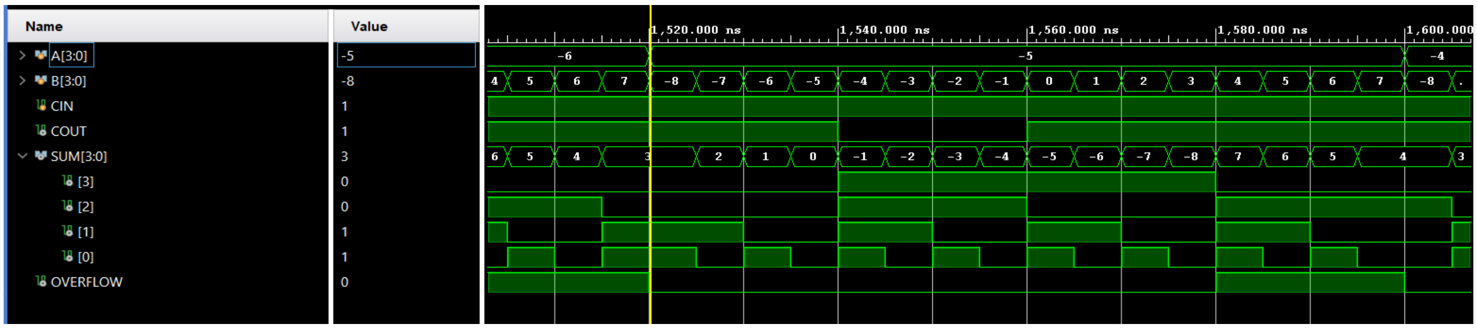
| Name | Value |
|---|---|
| A[3:0] | -8 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 0 |
| [3] | 0 |
| [2] | 0 |
| [1] | 0 |
| [0] | 0 |
| OVERFLOW | 0 |



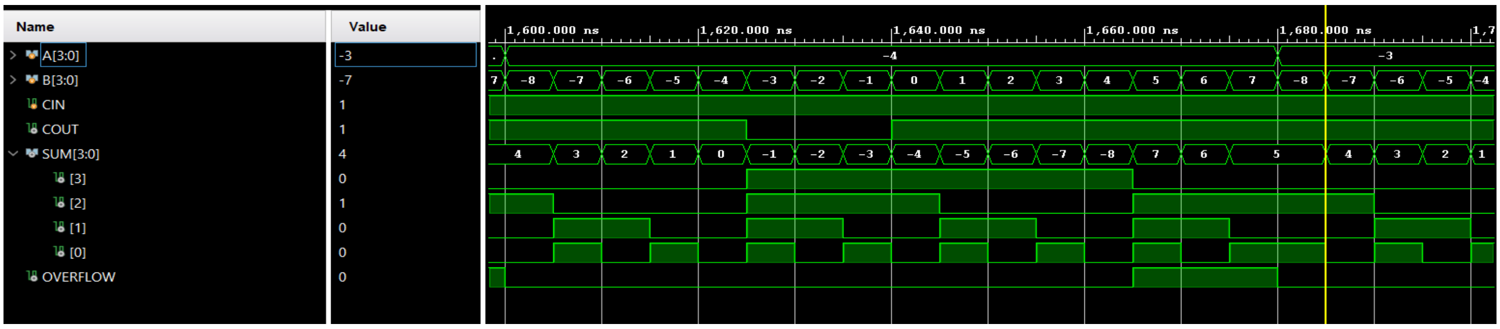
| Name | Value |
|---|---|
| A[3:0] | -8 |
| B[3:0] | 7 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 1 |
| [3] | 0 |
| [2] | 0 |
| [1] | 0 |
| [0] | 1 |
| OVERFLOW | 1 |



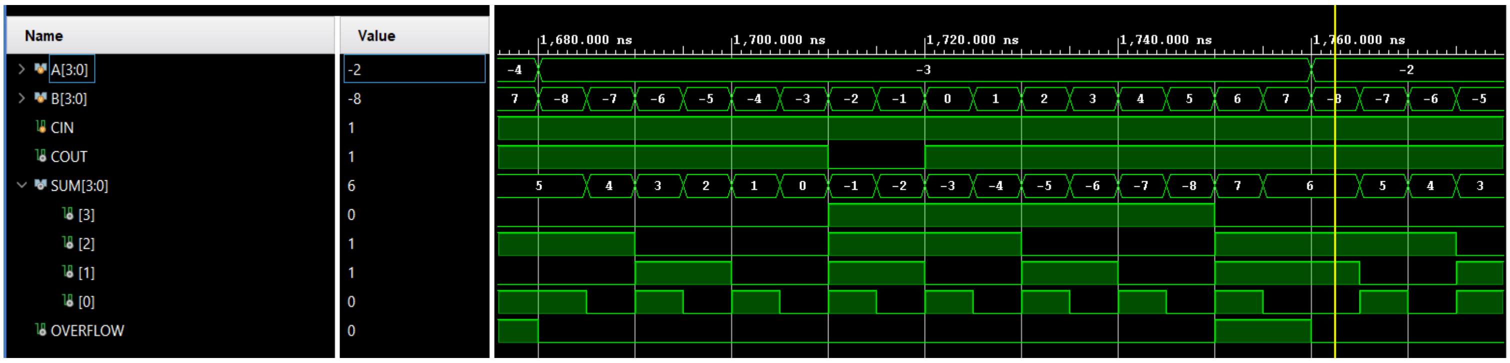
| Name | Value |
|---|---|
| A[3:0] | -7 |
| B[3:0] | 7 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 2 |
| [3] | 0 |
| [2] | 0 |
| [1] | 1 |
| [0] | 0 |
| OVERFLOW | 1 |



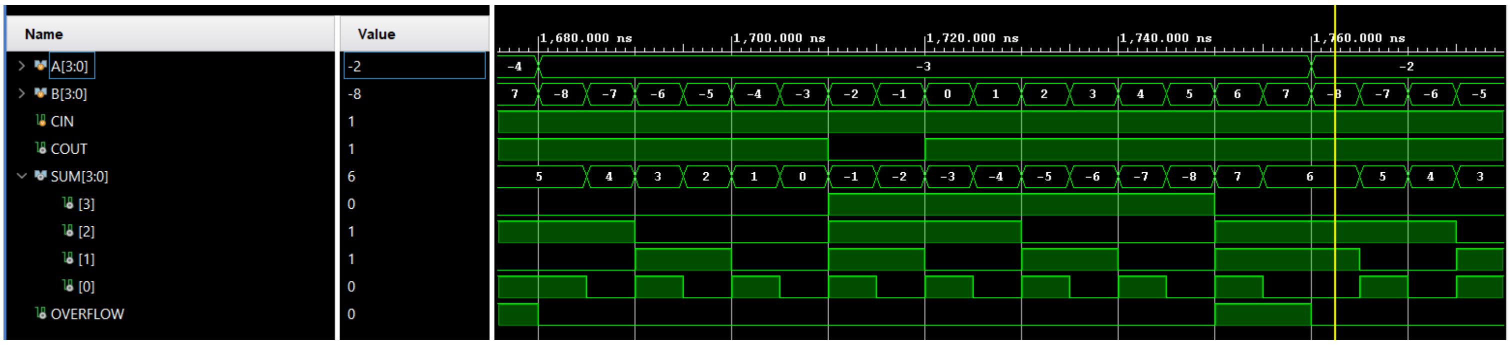
| Name | Value |
|---|---|
| A[3:0] | -5 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 3 |
| [3] | 0 |
| [2] | 0 |
| [1] | 1 |
| [0] | 1 |
| OVERFLOW | 0 |



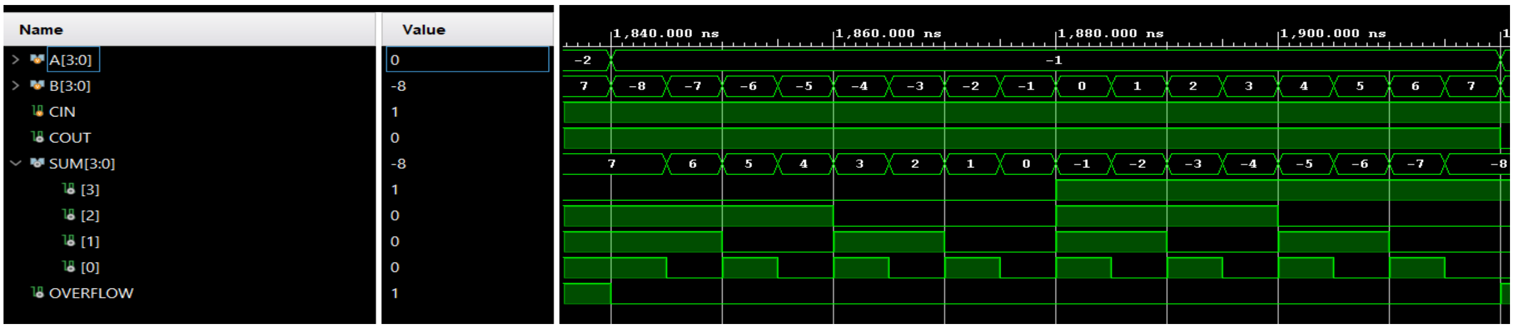
| Name | Value |
|---|---|
| A[3:0] | -3 |
| B[3:0] | -7 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 4 |
| [3] | 0 |
| [2] | 1 |
| [1] | 0 |
| [0] | 0 |
| OVERFLOW | 0 |



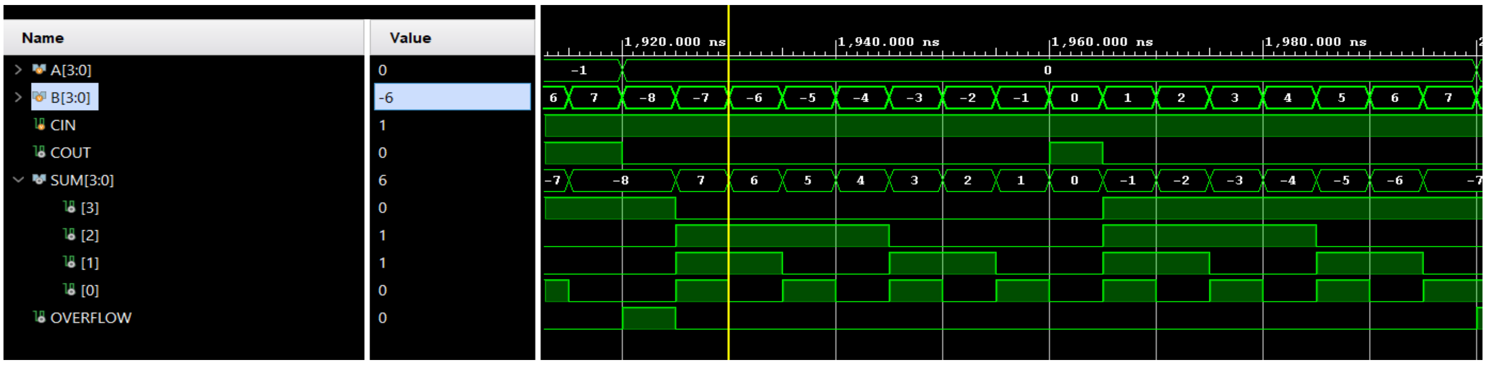
| Name | Value |
|---|---|
| A[3:0] | -2 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 6 |
| [3] | 0 |
| [2] | 1 |
| [1] | 1 |
| [0] | 0 |
| OVERFLOW | 0 |



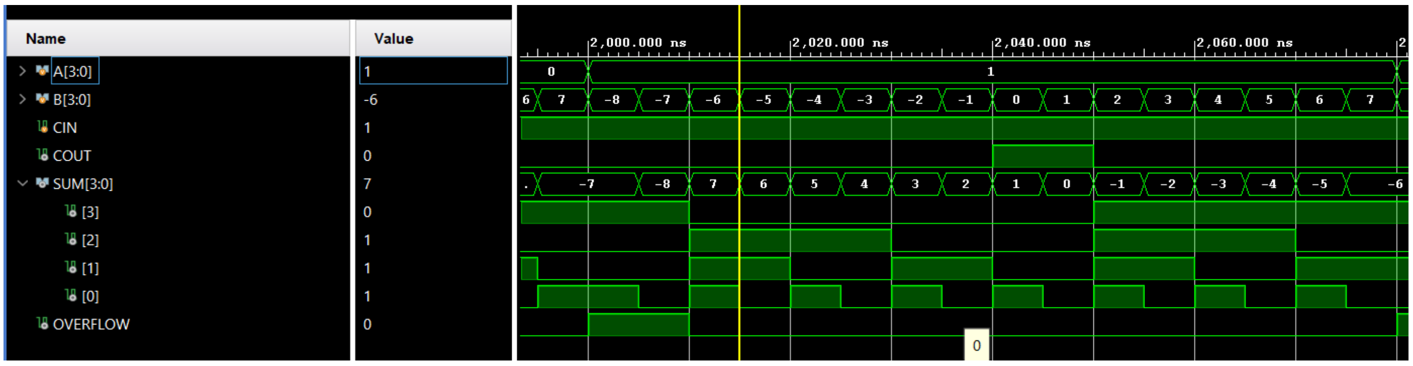
| Name | Value |
|---|---|
| A[3:0] | -2 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 6 |
| [3] | 0 |
| [2] | 1 |
| [1] | 1 |
| [0] | 0 |
| OVERFLOW | 0 |



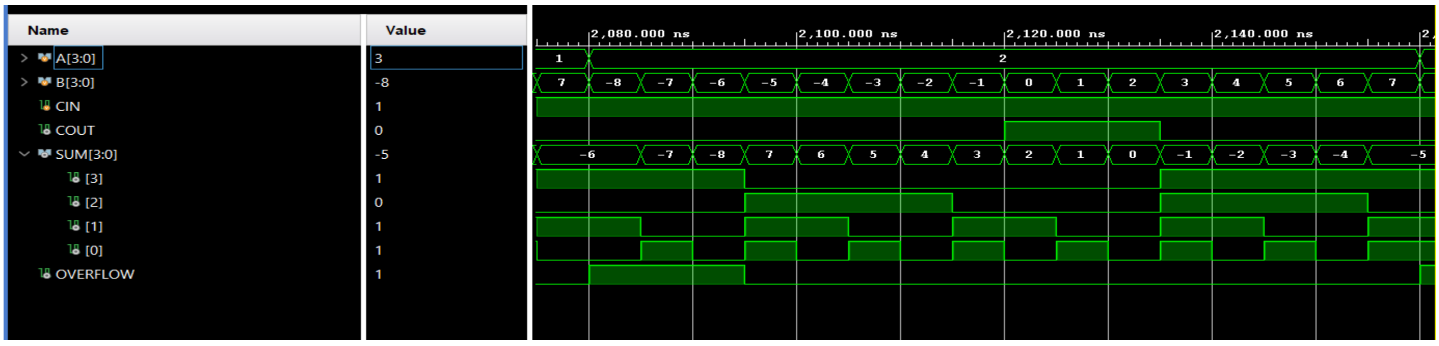
| Name | Value |
|---|---|
| A[3:0] | 0 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | -8 |
| [3] | 1 |
| [2] | 0 |
| [1] | 0 |
| [0] | 0 |
| OVERFLOW | 1 |



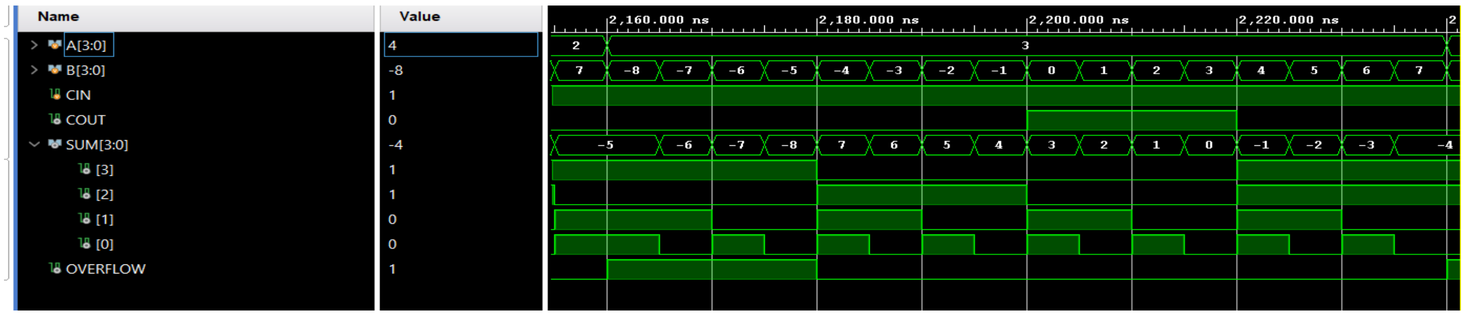
| Name | Value |
|---|---|
| A[3:0] | 0 |
| B[3:0] | -6 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | 6 |
| [3] | 0 |
| [2] | 1 |
| [1] | 1 |
| [0] | 0 |
| OVERFLOW | 0 |



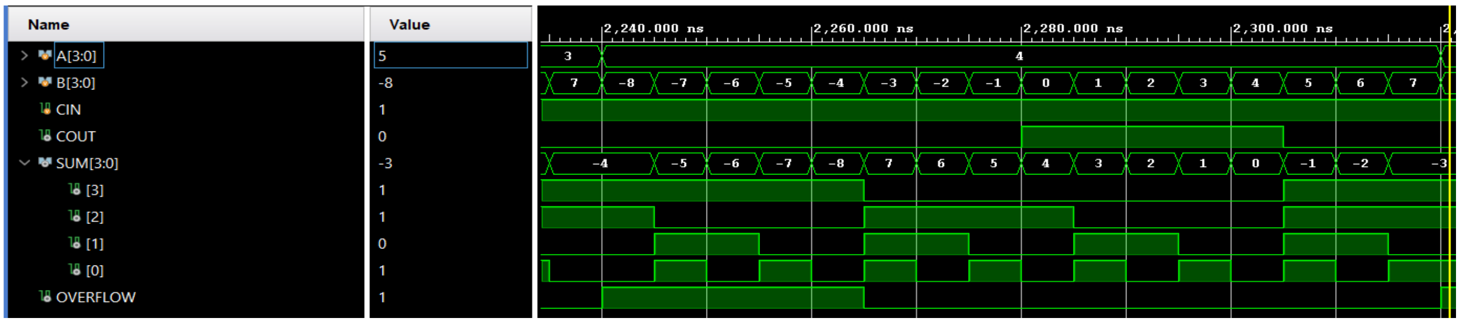
| Name | Value |
|---|---|
| A[3:0] | 1 |
| B[3:0] | -6 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | 7 |
| [3] | 0 |
| [2] | 1 |
| [1] | 1 |
| [0] | 1 |
| OVERFLOW | 0 |



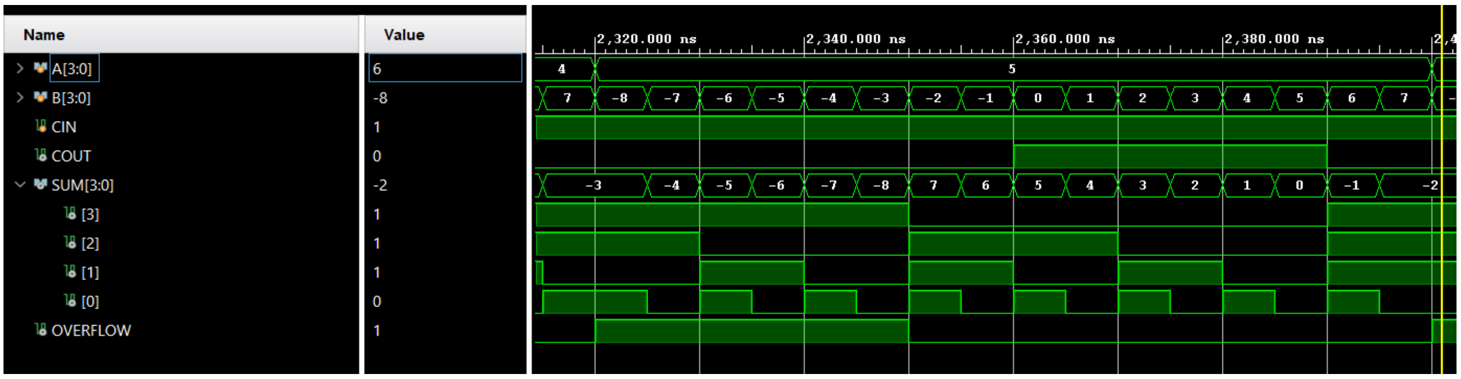
| Name | Value |
|---|---|
| A[3:0] | 3 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | -5 |
| [3] | 1 |
| [2] | 0 |
| [1] | 1 |
| [0] | 1 |
| OVERFLOW | 1 |



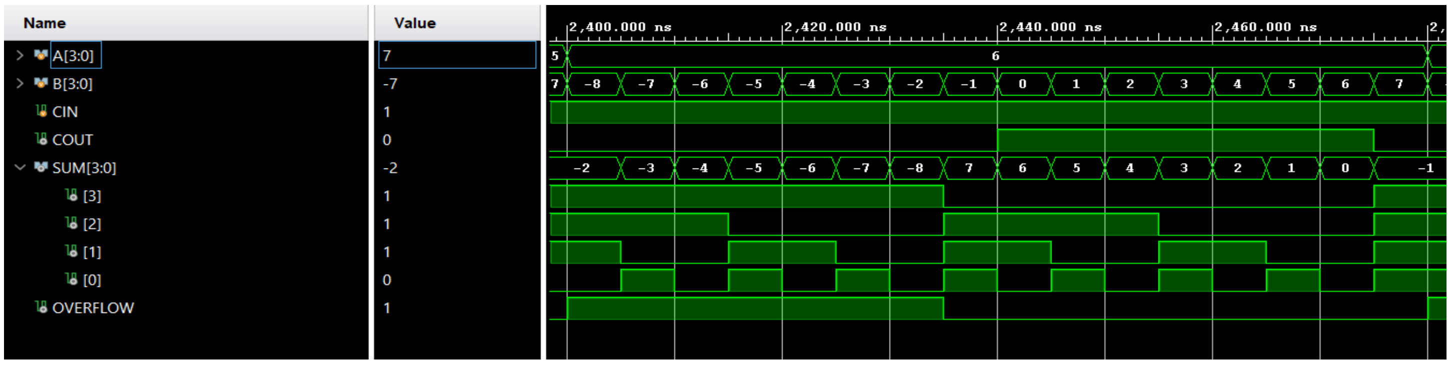
| Name | Value |
|---|---|
| A[3:0] | 4 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | -4 |
| [3] | 1 |
| [2] | 1 |
| [1] | 0 |
| [0] | 0 |
| OVERFLOW | 1 |



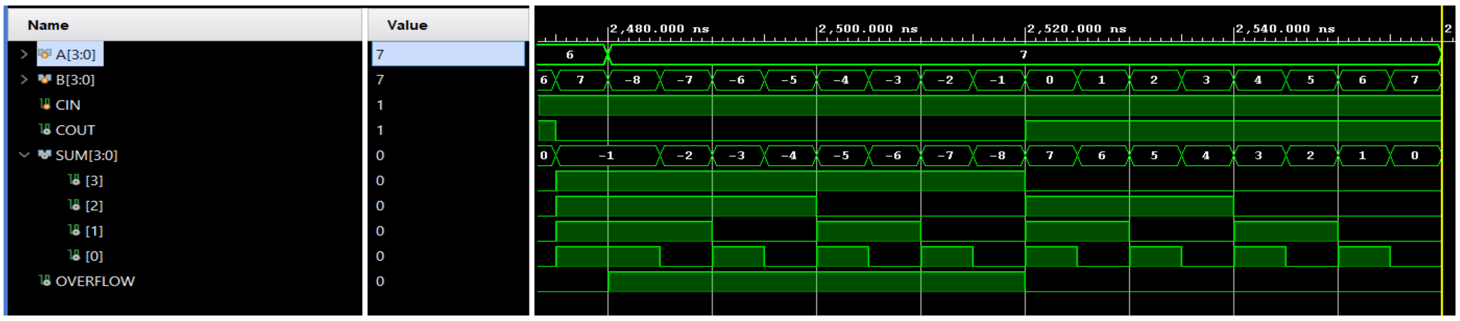
| Name | Value |
|---|---|
| A[3:0] | 5 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | -3 |
| [3] | 1 |
| [2] | 1 |
| [1] | 0 |
| [0] | 1 |
| OVERFLOW | 1 |



| Name | Value |
|---|---|
| A[3:0] | 6 |
| B[3:0] | -8 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | -2 |
| [3] | 1 |
| [2] | 1 |
| [1] | 1 |
| [0] | 0 |
| OVERFLOW | 1 |



| Name | Value |
|---|---|
| A[3:0] | 7 |
| B[3:0] | -7 |
| CIN | 1 |
| COUT | 0 |
| SUM[3:0] | -2 |
| [3] | 1 |
| [2] | 1 |
| [1] | 1 |
| [0] | 0 |
| OVERFLOW | 1 |



| Name | Value |
|---|---|
| A[3:0] | 7 |
| B[3:0] | 7 |
| CIN | 1 |
| COUT | 1 |
| SUM[3:0] | 0 |
| [3] | 0 |
| [2] | 0 |
| [1] | 0 |
| [0] | 0 |
| OVERFLOW | 0 |

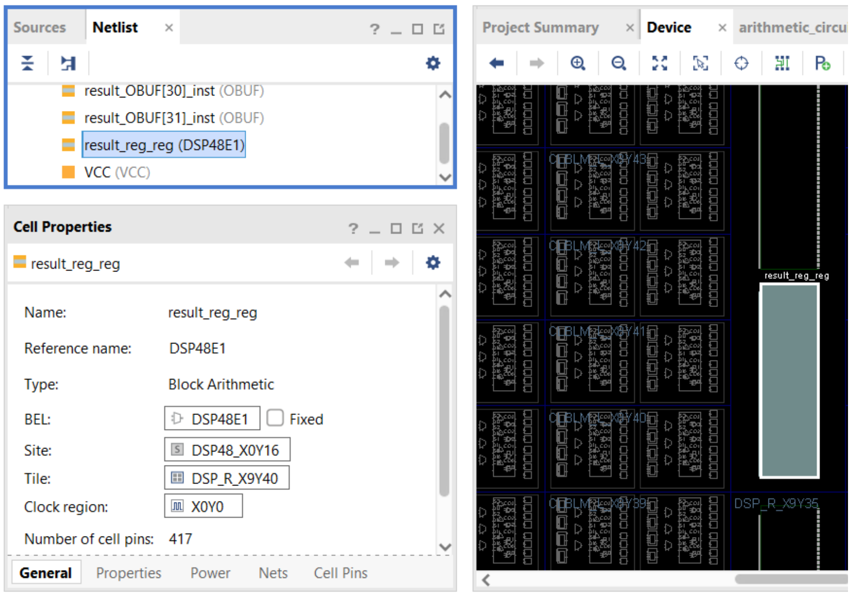
Salih Ömer Ongün
040220780

# RESEARCH

**1)** DSP Block has different tasks in FPGA. One of them multipliers implementation. Especially, DSP Blocks accelerate and facilitate multiplying. Therefore, multiplication and multiplication related operations are performed in DSP Blocks. If operations cannot be performed in a single DSP Blocks, Vivado use different DSP Blocks or use DSP Blocks and Slice logic together. [1]

```
module DSP_IMP
(
    input clk,
    input [15:0] a,
    input [15:0] b,
    input [15:0] c,
    output [31:0] result
);
    reg [31:0] result_reg;

    always @(posedge clk) begin
        result_reg <= (a * b) + c;
    end

    assign result = result_reg;

endmodule
```

*Example code for usage DSP Blocks*



Sources | Netlist

result_OBUF[30]_inst (OBUF)
result_OBUF[31]_inst (OBUF)
result_reg_reg (DSP48E1)
VCC (VCC)

Cell Properties

result_reg_reg

| | |
|---|---|
| Name: | result_reg_reg |
| Reference name: | DSP48E1 |
| Type: | Block Arithmetic |
| BEL: | DSP48E1 ☐ Fixed |
| Site: | DSP48_X0Y16 |
| Tile: | DSP_R_X9Y40 |
| Clock region: | X0Y0 |
| Number of cell pins: | 417 |

General | Properties | Power | Nets | Cell Pins

***Placement***

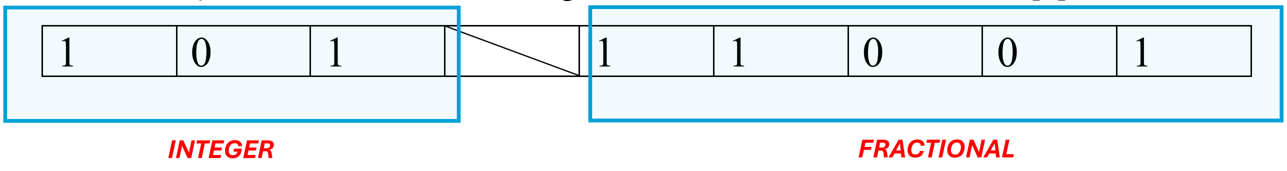
DSP Blocks accelerate and facilitate multiplying. They use FPGA resources efficiently. They have optimal results in term of power consumption.

**2)** Fixed point representation is a method used to represent fractional numbers. It is used in DSP operations and arithmetic operations. It has significant advantages compared with other representations for fractional numbers.
It is represented with Qm.n format.
Q: Format of fixed point. M

m : Number of integer bits

n : Number of fractional bits

For Q3.5 format, it have 3 bit integer number, 5 bit fractional number. [2]



5 is integer part

0.5+0.25+0+0+0.03125=0.78125 is fractional part

101.11001 = 5.78125

Fixed point hardware have less complex than floating points. Therefore, it reduces the size of the chip and results in less power consumption. Generally, fixed-point representation uses fewer bits. Therefore, it requires less memory usage. It may increase the speed of FPGA. The size of the chip and the use of less memory can reduce the price of the FPGA. [3]

## References

[1] *DSP Block Implementation • VivaDo Design Suite User Guide: Synthesis (UG901) • Reader • AMD Technical Information Portal*. (n.d.). https://docs.amd.com/r/en-US/ug901-vivado-synthesis/DSP-Block-Implementation

[2] Harvie, L. (2024, August 22). How to perform Fixed-Point Arithmetic on an FPGA. *RunTime Recruitment*. https://runtimerec.com/how-to-perform-fixed-point-arithmetic-on-an-fpga/

[3] *Benefits of Fixed-Point Hardware*. (n.d.). https://www.mathworks.com/help/fixedpoint/gs/benefits-of-fixed-point-hardware.html