ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

EXPERIMENT NO	:	5
EXPERIMENT DATE	:	21.05.2020
LAB SESSION	:	FRIDAY - 13.30
GROUP NO	:	G11

GROUP MEMBERS:

150180064 : MELİS GÜNŞEBER

150170043 : EBRAR ÖMER

150140902 : PERİT JAN AYDEMİR

SPRING 2020

Contents

1 INTRODUCTION

The main goal of this experiment is to implement a 8-bit Full adder using the Verilog software. This is an introduction experiment for us to use and learn the usage and functions of the Verilog.

2 MATERIALS AND METHODS

2.1 Experiment – Part 1

In the first part first we implement the main AND, OR, NOT and XOR modules which is going to be used in following stages of the experiment.

2.1.1 AND Gate

```
1 module and_gate(
2          input A,
3          input B,
4          output C
5          );
6          assign C = A & B;
7 endmodule
```

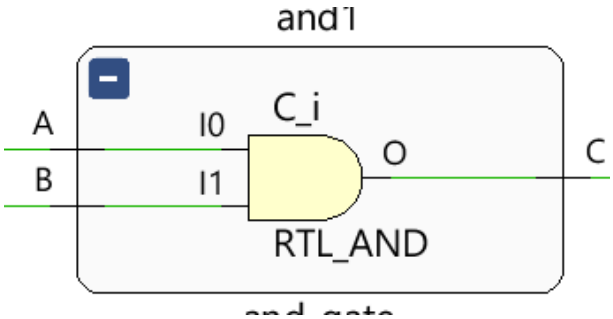

Figure 1: AND Gate

We have defined the inputs A, B and the output as C. And after that, we have assigned C as A B by using the bitwise AND operation.

2.1.2 NOT Gate

```
module not_gate(
        input A,
        output B
        );
        assign B =   A  ;
endmodule
```

We have defined the input as A and the output as B. And after that, we have assigned B as A by using the Bitwise NOT operation.

2.1.3 OR Gate

```
module or_gate(
        input A,
        input B,
        output C
        );
        assign C = A | B;
endmodule
```

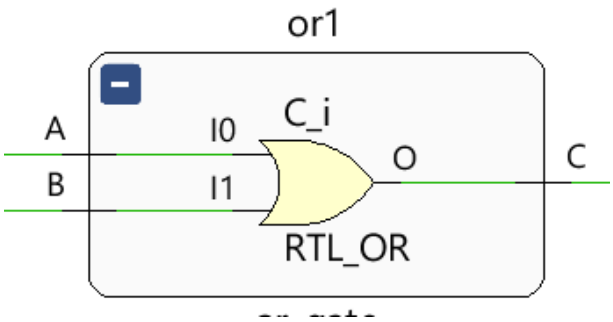

Figure 2: OR Gate

We have defined the inputs A, B and the output as C. And after that, we have assigned C as A — B by using the bitwise OR operation.

2.1.4 XOR Gate

```
module xor_gate2(
        input A,
        input B,
        output C
        );
        assign C = A^B;
endmodule
```

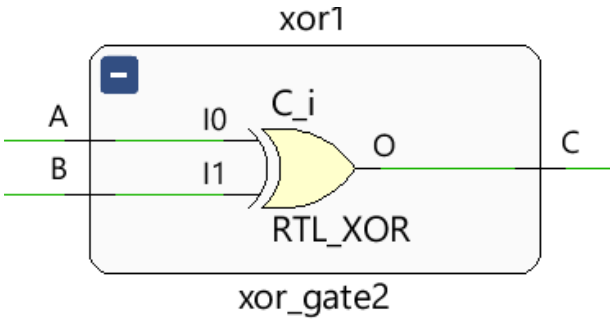

Figure 3: XOR Gate

We have defined the inputs A, B and the output as C. And after that, we have assigned C as A ^{B}*byusingthebitwiseXORoperation*.

2.2 Experiment – Part 2

In this part we are going to implement a 1-bit Half Adder with the modules we have design before.

```
module half_adder(A,B,C,S);
    input wire A;
    input wire B;
    output wire C;
    output wire S;
    and_gate AND1(.A(A),.B(B),.C(C));
    xor_gate2 XOR1(.A(A),.B(B),.C(S));
endmodule
```

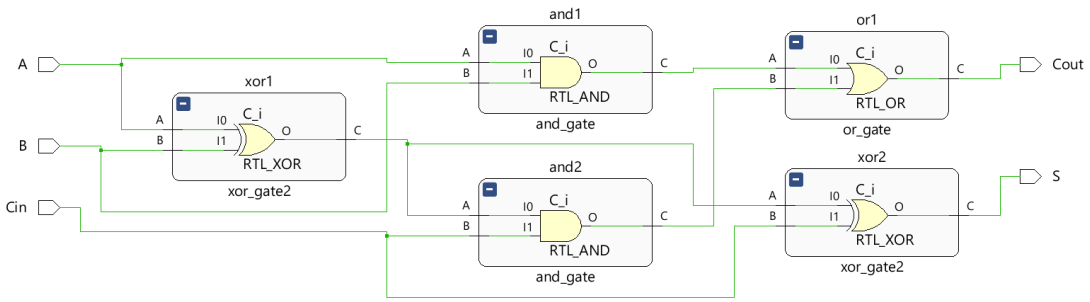

Figure 4: Half Adder

We have used our previously created circuit elements and assigned input wires A, B and output wires C, S for Cout and Sum.

2.3 Experiment – Part 3

In this part we are going to implement a 1-bit Full Adder with the Half Adder and OR gate we have design before.

2.3.1 1-Bit Full Adder

```
1  module full_adder(A,B,Cin,S,Cout);
2      input wire A;
3      input wire B;
4      input wire Cin;
5      output wire S;
6      output wire Cout;
7      wire D;
8      wire F;
9      wire G;
10     xor_gate2 xor1(.A(A),.B(B),.C(D));
11     xor_gate2 xor2(.A(D),.B(Cin),.C(S));
12     and_gate and1(.A(A),.B(B),.C(F));
13     and_gate and2(.A(D),.B(Cin),.C(G));
14     or_gate or1(.A(F),.B(G),.C(Cout));
15 endmodule
```

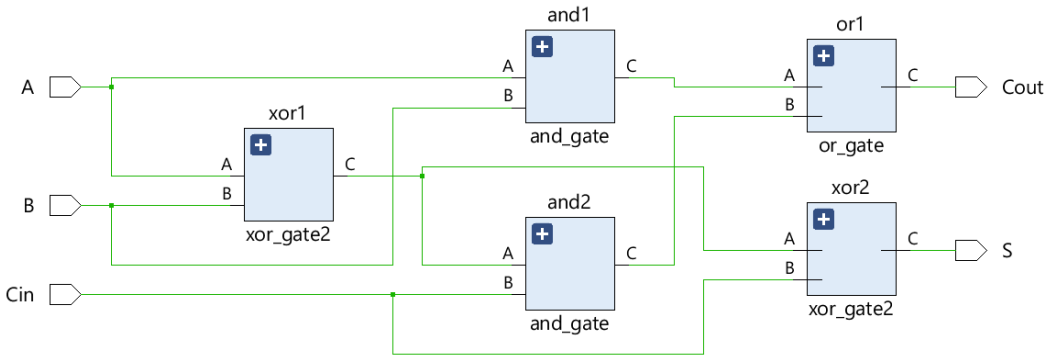

Figure 5: Full Adder Simulation

2.4 Experiment – Part 4

In this part, we have implemented an 8-Bit Full Adder by using 8 1-Bit Full Adder modules that we designed in the previous part.

2.4.1 8-Bit Full Adder

```
module eight_bitfull_adder(A,B,Cin,S,carryout);
    input [7:0] A;
    input [7:0] B;
    input Cin;
    output [7:0] S;
    output carryout;
    wire [6:0]Cout;
    full_adder f1(.A(A[0]),.B(B[0]),.Cin(Cin),.S(S[0]),.Cout(Cout[0]));
    full_adder f2(.A(A[1]),.B(B[1]),.Cin(Cout[0]),.S(S[1]),.Cout(Cout[1]));
    full_adder f3(.A(A[2]),.B(B[2]),.Cin(Cout[1]),.S(S[2]),.Cout(Cout[2]));
    full_adder f4(.A(A[3]),.B(B[3]),.Cin(Cout[2]),.S(S[3]),.Cout(Cout[3]));
    full_adder f5(.A(A[4]),.B(B[4]),.Cin(Cout[3]),.S(S[4]),.Cout(Cout[4]));
    full_adder f6(.A(A[5]),.B(B[5]),.Cin(Cout[4]),.S(S[5]),.Cout(Cout[5]));
    full_adder f7(.A(A[6]),.B(B[6]),.Cin(Cout[5]),.S(S[6]),.Cout(Cout[6]));
    full_adder f8(.A(A[7]),.B(B[7]),.Cin(Cout[6]),.S(S[7]),.Cout(carryout));
endmodule
```

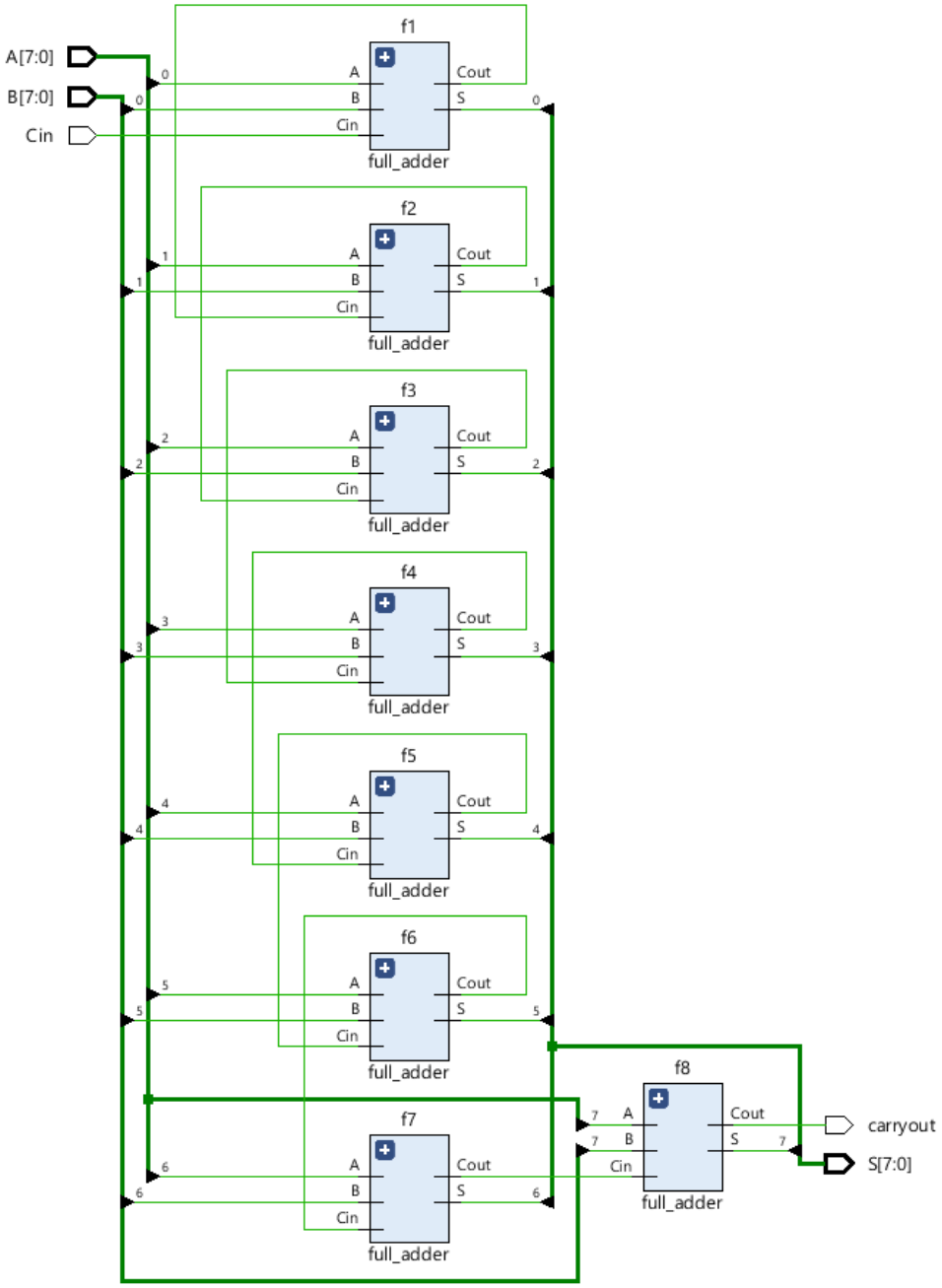

Figure 6: 8-bit Full Adder

2.5 Experiment - Part 5

In this part, we have implemented an 8-Bit Full Adder-Subtractor by using eight 1-Bit Full Adder, NOT, XOR, AND and OR modules that we designed in the previous parts.

2.5.1 8-Bit Full Adder-Subtractor

```
module addersubtractor(A,B,Cin,S,carryout);
    input [7:0] A;
    input [7:0] B;
    input [1:0] Cin;
    output [7:0] S;
    output carryout;
    wire [7:0] Bnew;
    xor_gate2 B0(.A(B[0]),.B(Cin),.C(Bnew[0]));
    xor_gate2 B1(.A(B[1]),.B(Cin),.C(Bnew[1]));
    xor_gate2 B2(.A(B[2]),.B(Cin),.C(Bnew[2]));
    xor_gate2 B3(.A(B[3]),.B(Cin),.C(Bnew[3]));
    xor_gate2 B4(.A(B[4]),.B(Cin),.C(Bnew[4]));
    xor_gate2 B5(.A(B[5]),.B(Cin),.C(Bnew[5]));
    xor_gate2 B6(.A(B[6]),.B(Cin),.C(Bnew[6]));
    xor_gate2 B7(.A(B[7]),.B(Cin),.C(Bnew[7]));
   eight_bitfull_adder fonk(.A(A),.B(Bnew),.Cin(Cin),.S(S),.carryout(carryout
endmodule
```

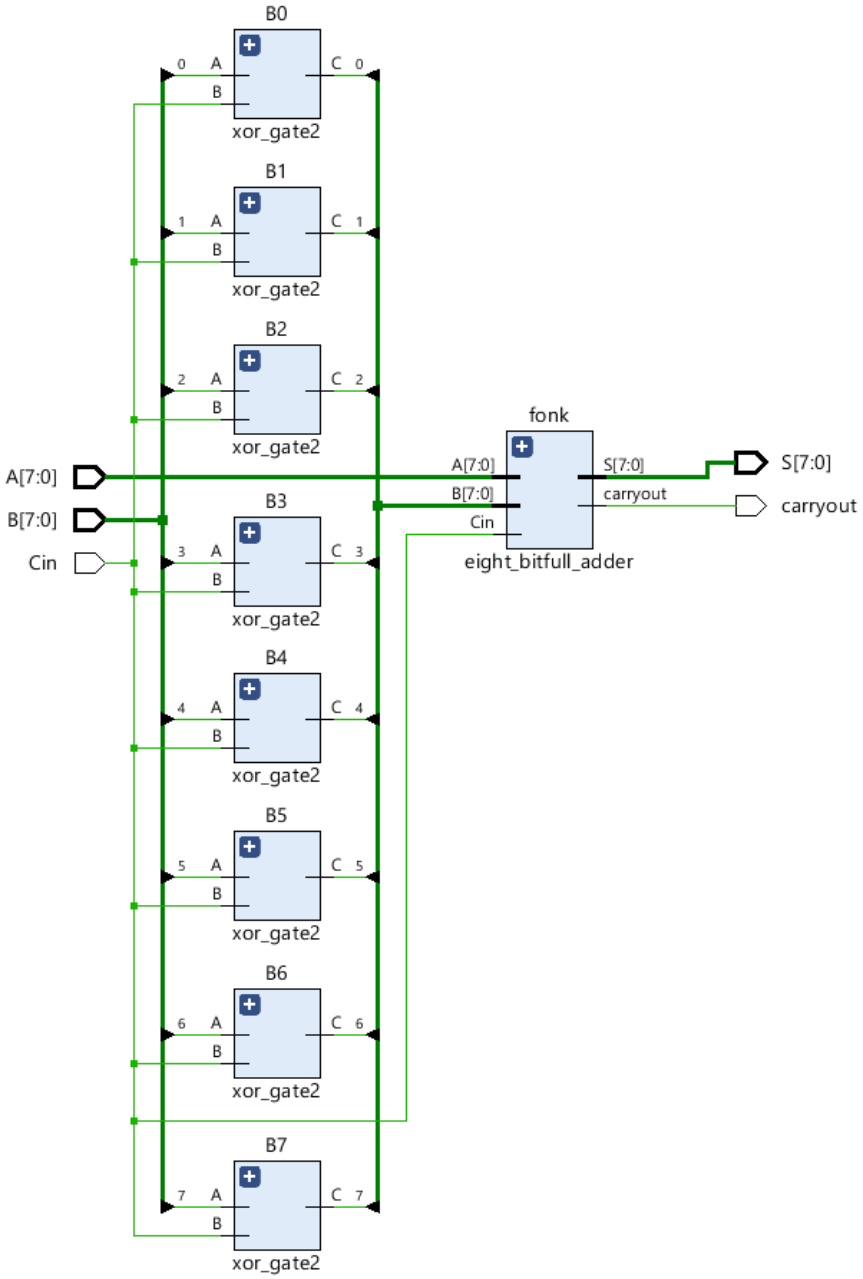

Figure 7: 8-bit Full Adder-Subtractor

2.6 Experiment - Part 6

In this part, we have implemented a module which calculates the 2B - 3A by using 8-Bit Adder-Subtractor, Adder, NOT, XOR, AND, and OR modules.

2.6.1 2B - 3A

```
module part6(A,B,Cin,Output,Cout);
    input [7:0] A;
    input [7:0] B;
    input [1:0] Cin;
    output [7:0] Output;
    output Cout;

    wire [7:0] Anew;
    wire [7:0] Anew2;
    wire [7:0] Bnew;
    wire cout;

   eight_bitfull_adder fb(.A(B), .B(B), .Cin(Cin), .S(Bnew), .carryout(cout)
   eight_bitfull_adder fa(.A(A), .B(A),  .Cin(Cin), .S(Anew), .carryout(cout
   eight_bitfull_adder fa2(.A(A), .B(Anew), .Cin(Cin), .S(Anew2) ,.carryout(
   wire [1:0] Cin2;
   assign Cin2=1;
   addersubtractor ar(.A(Bnew),.B(Anew2), .Cin(Cin2),.S(Output),.carryout(Cou
endmodule
```

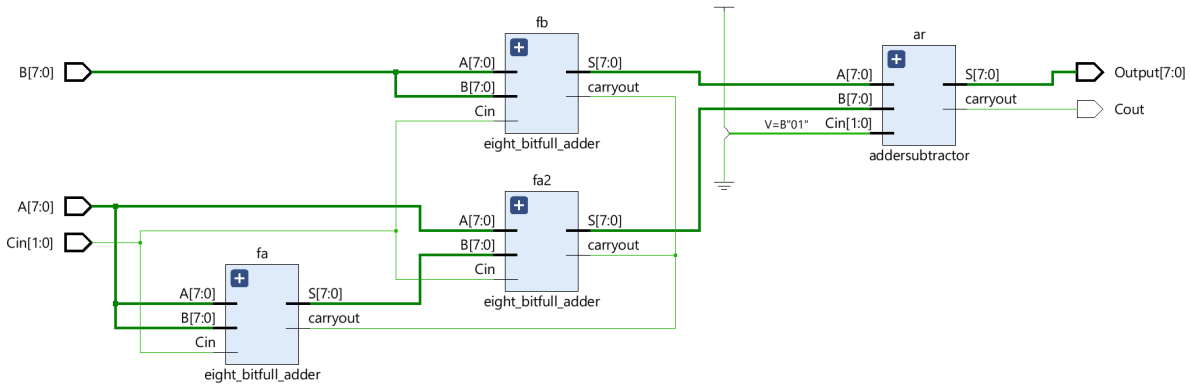

Figure 8: 8-bit Full Adder-Subtractor

3 3.RESULTS

3.1 The Simulation of Half Adder

```
module half_adder_test();
    reg  Input1;
    reg  Input2;
    wire  Output;
    wire  Carry;
    half_adder uut(.A(Input1),.B(Input2), .C(Carry),.S(Output));
    initial begin
        Input1=0;    Input2=0; #250;
        Input1=0;    Input2=1; #250;
        Input1 =1;   Input2=0; #250;
        Input1 =1;   Input2=1; #250;
     end
endmodule
```

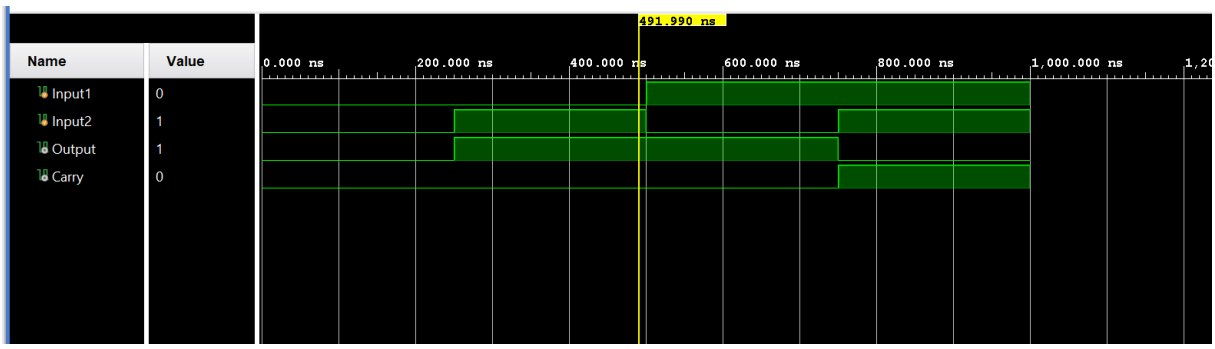

Figure 9: Half Adder Simulation

As we can see above all possible input combinations have been covered. Output C becomes one when there is a positive feed from only one input and when there are two positive feeds the output becomes 0 and the carry output becomes one.

3.2 Simulation of 1-Bit Full Adder

```
module full_adder_test();
    reg  Input1;
    reg  Input2;
    reg  Cin;
    wire  Sum;
    wire  Cout;
    full_adder uut(.A(Input1),.B(Input2), .Cin(Cin),.S(Sum),.Cout(Cout));
    initial begin
        Input1=0;    Input2=0; Cin=0; #125;
        Input1=0;    Input2=0; Cin=1; #125;
        Input1=0;    Input2=1; Cin=0; #125;
        Input1=0;    Input2=1; Cin=1; #125;
        Input1=1;    Input2=0; Cin=0; #125;
        Input1=1;    Input2=0; Cin=1; #125;
        Input1=1;    Input2=1; Cin=0; #125;
        Input1=1;    Input2=1; Cin=1; #125;
     end
endmodule
```

;

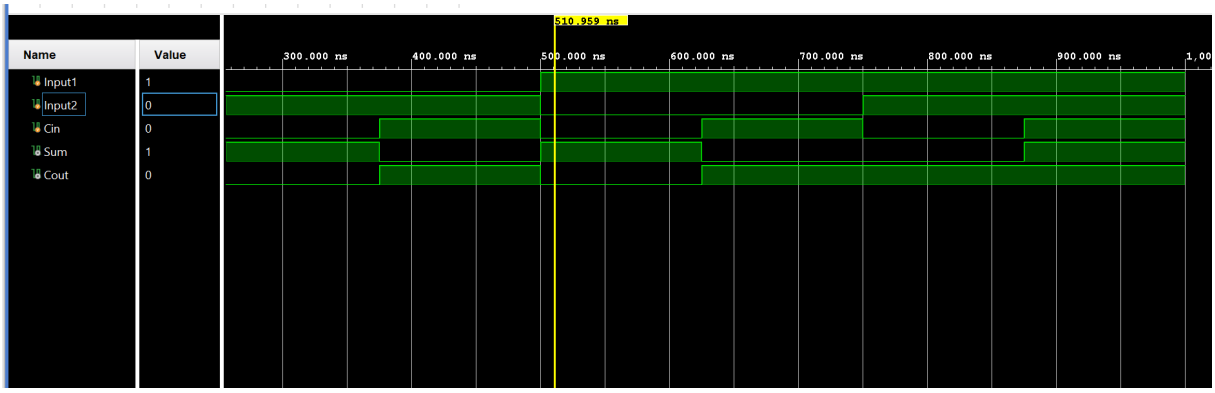

Figure 10: Full Adder Simulation

As we can see above all possible input combinations have been covered. Output C becomes one when there is a positive feed from only one input and when there are two positive feeds the output C becomes 0 and the carry output becomes one. When we feeded the circuit with all three inputs A,B and Cin we got both Sum and Cout as one.

3.3 8-Bit Full Adder Simulation

```
module eight_adder_test();
    reg [7:0] A;
    reg [7:0] B;
    reg Cin;
    wire [7:0] O;
    wire  carryout;

    eight_bitfull_adder uut(.A(A),.B(B),.Cin(Cin),.S(O),.carryout(carryout))
    initial begin
        A =8'd27;   B=8'd5;   Cin=1'd0; #125;
        A =8'd19;   B=8'd92;  Cin=1'd0; #125;
        A =8'd16;   B=8'd34;  Cin=1'd0; #125;
        A =8'd196;  B=8'd2;   Cin=1'd0; #125;
        A =8'd200;  B=8'd95;  Cin=1'd0; #125;
        A =8'd48;   B=8'd15;  Cin=1'd0; #125;
        A =8'd78;   B=8'd255; Cin=1'd0; #125;
        A =8'd14;   B=8'd53;  Cin=1'd0; #125;
     end
endmodule
```

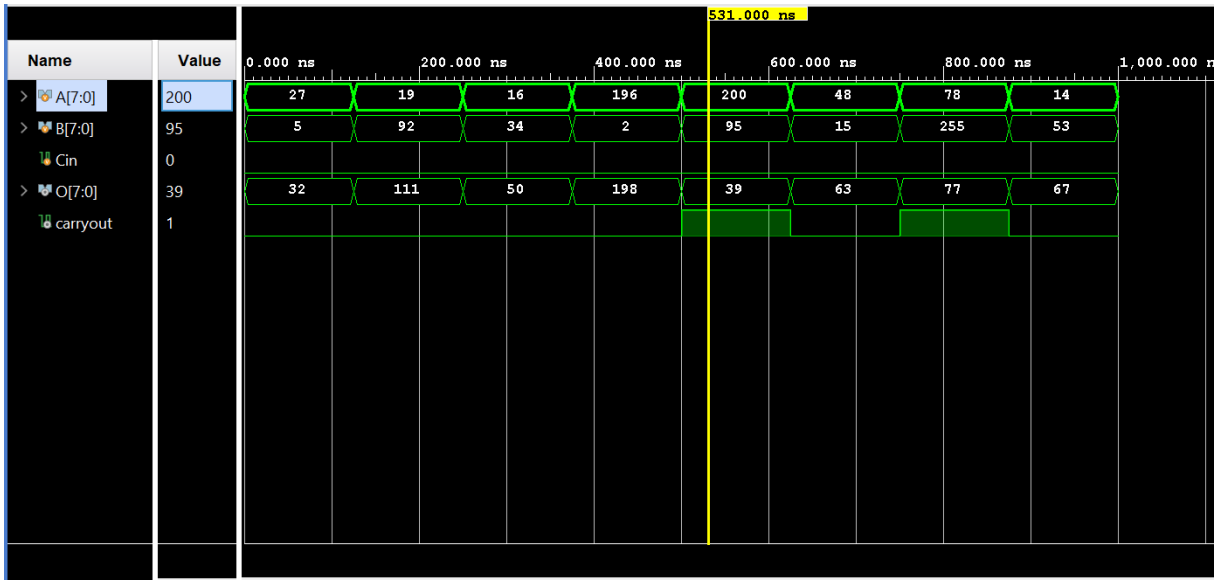

Figure 11: 8-bit Full Adder

3.4 8-Bit Full Adder-Subtractor Simulation

```
module addsub_test();
    reg [7:0] A;
    reg [7:0] B;
    reg [1:0] Cin;
    wire [7:0] O;\
    wire  carryout;
    addersubtractor uut(.A(A),.B(B),.Cin(Cin),.S(O),.carryout(carryout));\
    initial begin
        A =8'd27;   B=8'd5;   Cin=1'd0; #125;
        A =8'd19;   B=8'd92;  Cin=1'd0; #125;
        A =8'd16;   B=8'd34;  Cin=1'd1; #125;
        A =8'd196;  B=8'd2;   Cin=1'd1; #125;
        A =8'd200;  B=8'd95;  Cin=1'd0; #125;
        A =8'd48;   B=8'd165; Cin=1'd0; #125;
        A =8'd78;   B=8'd255; Cin=1'd0; #125;
        A =8'd14;   B=8'd53;  Cin=1'd1; #125;
     end
endmodule
```

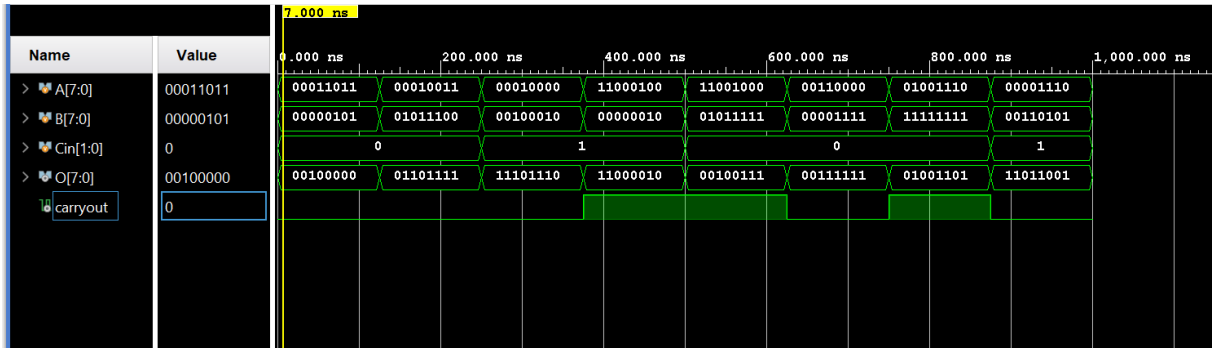

Figure 12: 8-bit Full Adder-Subtractor Simulation

3.5 Simulation of 2A - 3B

```
module part6test();
    reg [7:0] A;
    reg [7:0] B;
    reg [1:0] Cin;
    wire [7:0] O;
    part6 uut(.A(A),.B(B),.Cin(Cin),.Output(O),.Cout(carryout));
    initial begin
        A =8'd27;   B=8'd5;   Cin=1'd0; #125;
        A =8'd19;   B=8'd92;  Cin=1'd0; #125;
        A =8'd16;   B=8'd34;  Cin=1'd0; #125;
        A =8'd196;  B=8'd2;   Cin=1'd0; #125;
        A =8'd200;  B=8'd95;  Cin=1'd0; #125;
        A =8'd48;   B=8'd165; Cin=1'd0; #125;
        A =8'd78;   B=8'd255; Cin=1'd0; #125;
        A =8'd14;   B=8'd53;  Cin=1'd0; #125;
     end
endmodule
```

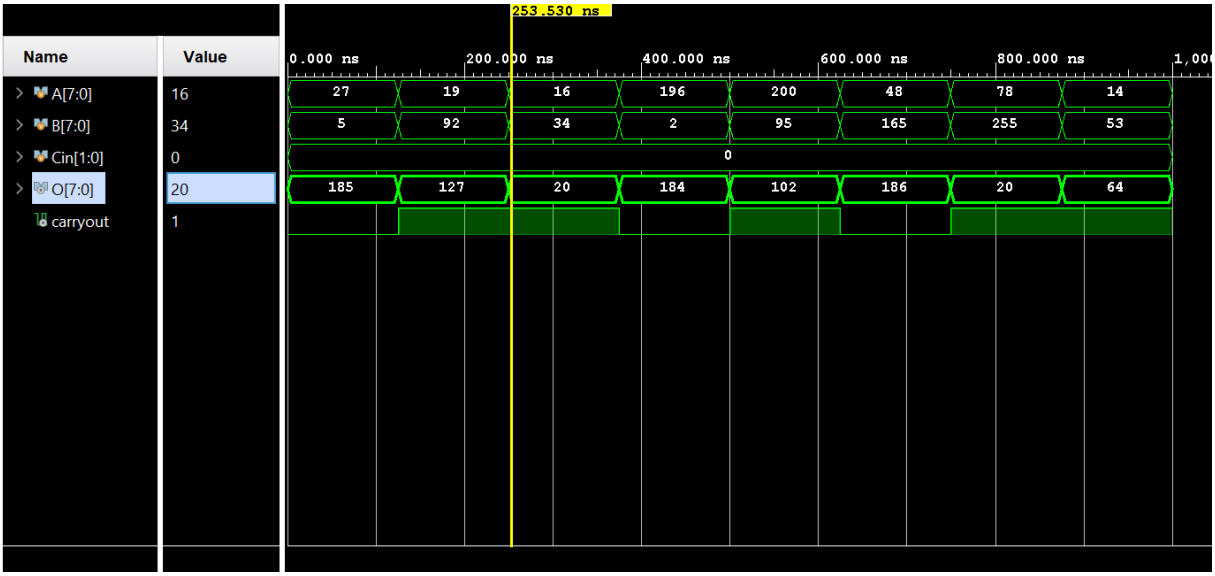

Figure 13: 8-bit Full Adder-Subtractor

4 CONCLUSION

We experienced to use Verilog for the first time. We implemented the circuits with writing some piece of design codes for the gates and some simulation codes for testing the outputs. In simulation phases we saw that the adder and subtractors that we implemented worked correctly. We learned that these circuits can also be implemented in a program and can be seen how they are working.

5 REFERENCES