Digital System Design Applications

Experiment 4 Report

Hasan Enes Şimşek

040160221

1 Half Adder

1.1 Verilog Codes

```
module half_adder(
   input A,B,
   output C,S
   );

(* dont_touch *) assign C = A & B;
(* dont_touch *) assign S = A ^ B; //XOR

endmodule
```

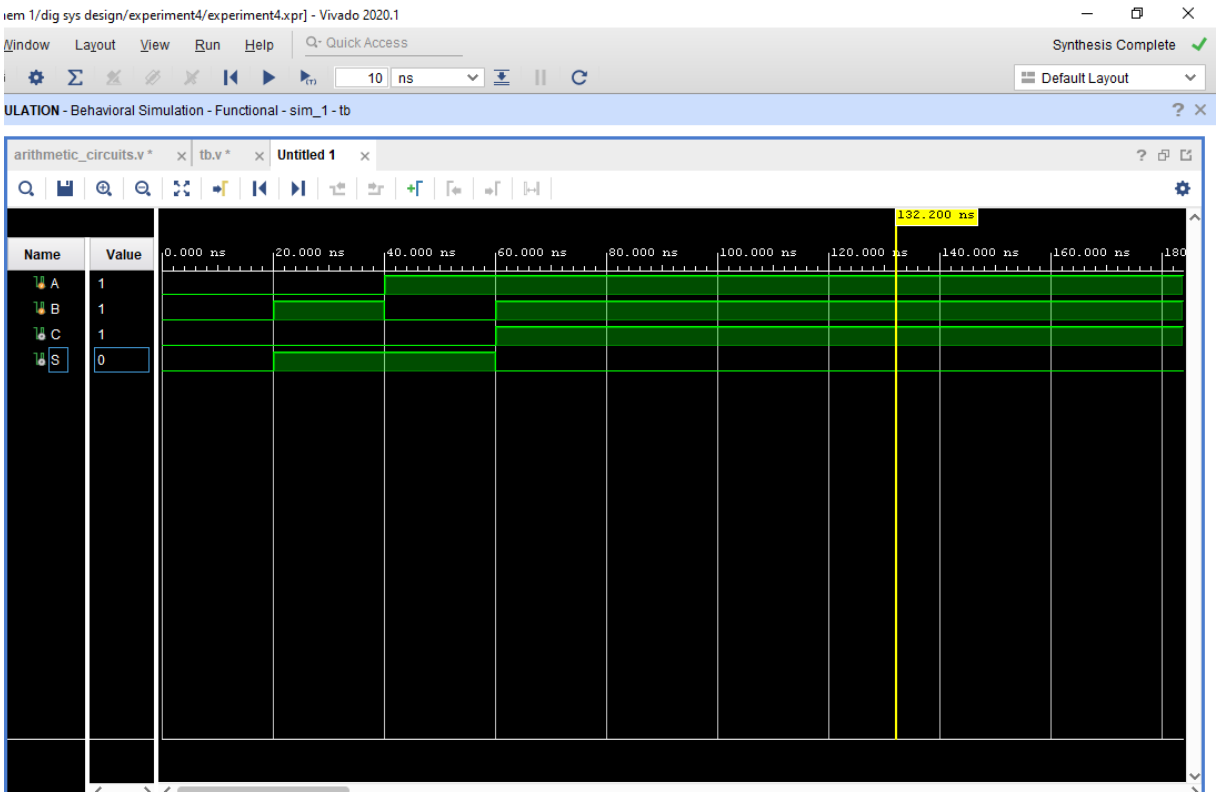
Arithmetic_circuits.v

```
module tb();
   reg A,B;
   wire C,S;

   half_adder HA(A,B,C,S); // HA

   initial begin

   A = 1'b0; B = 1'b0; #20;
   A = 1'b0; B = 1'b1; #20;
   A = 1'b1; B = 1'b0; #20;
   A = 1'b1; B = 1'b1; #20;

   end
endmodule
```

Tb.v

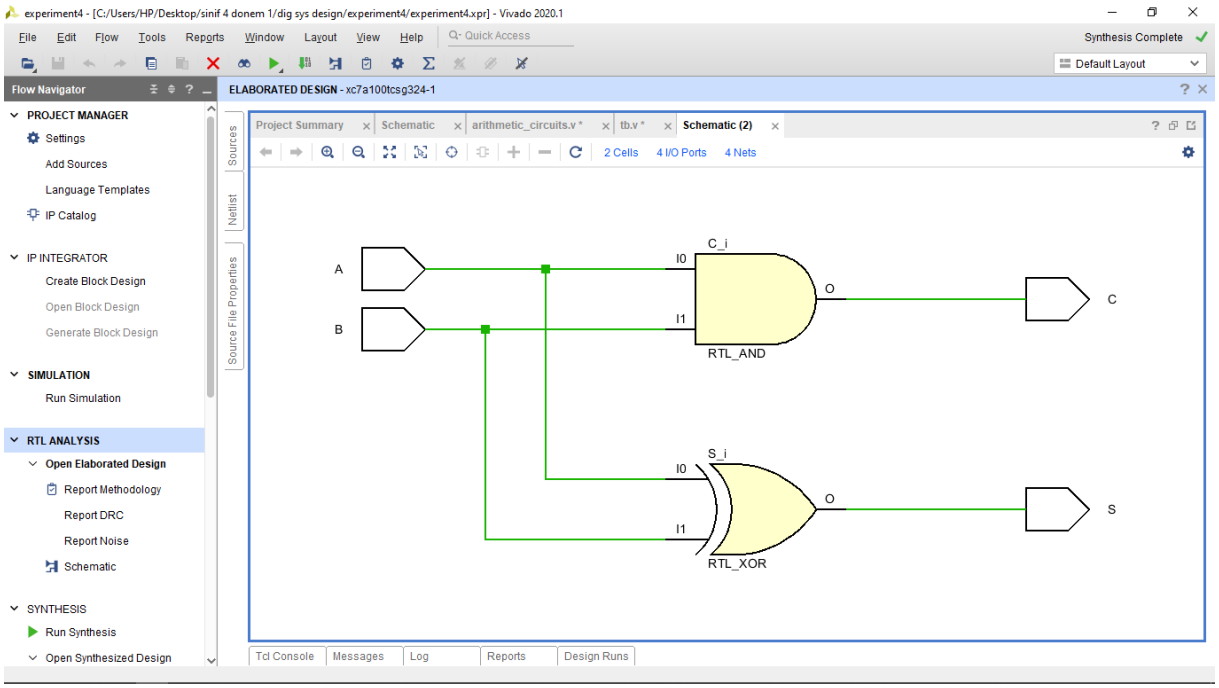
1.2 Behavioral Simulation

Simulation was obtained successfully.



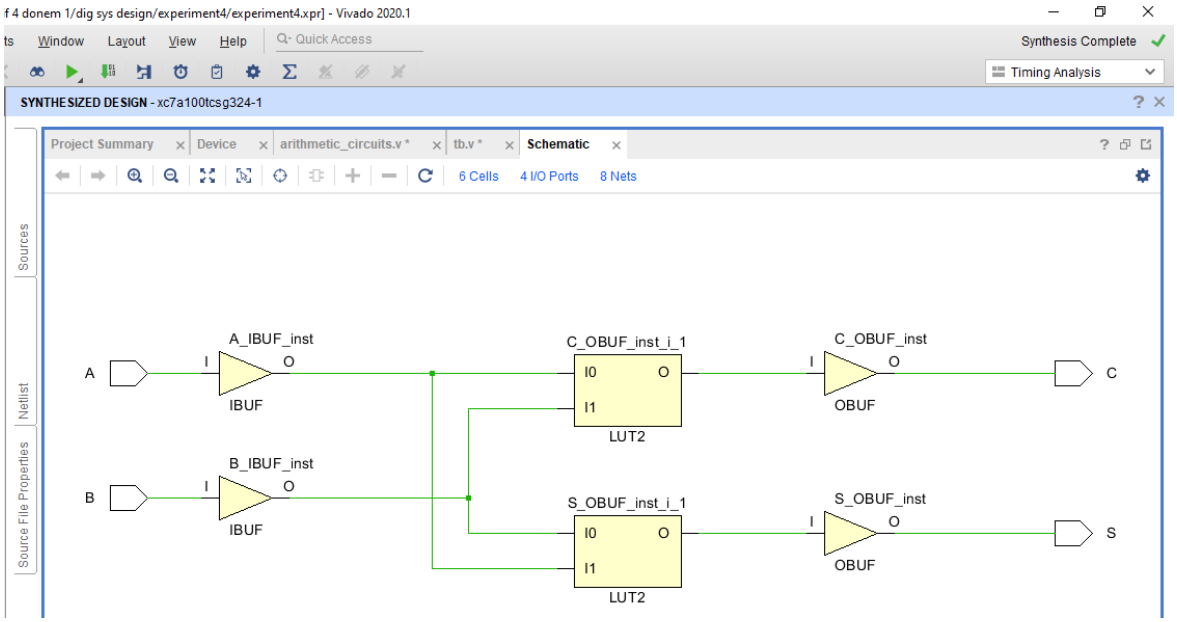
1.3 RTL Schematic

RTL Schematic was obtained as it should be.



1.4 Technology Schematic

Technology Schematic was obtained successfully. **There are 2 LUT2 in the design. One is for XOR, other is for AND gate. 4 buffer are for input and outputs.**

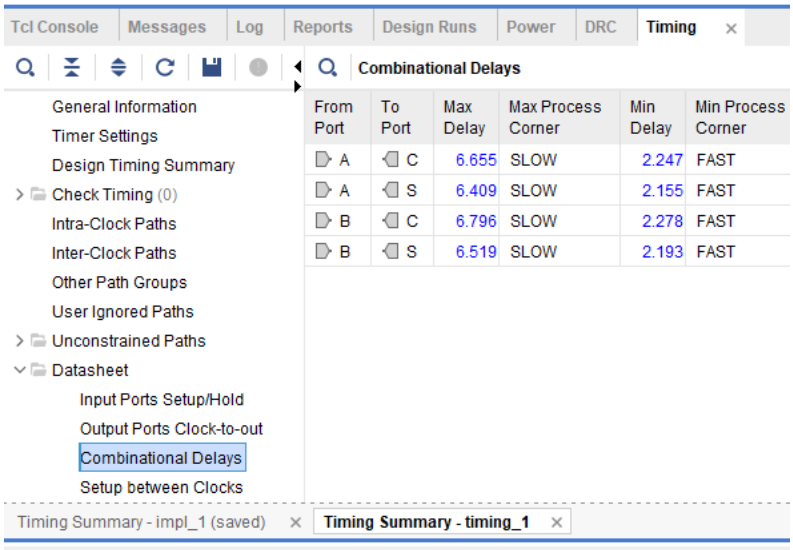


1.5 LUT Usage

There are 2 LUT2 in the design. One is for XOR, other is for AND gate.

1.6 Delays

After implementation following results are obtained in the figure. Max path delay occurred between B bit and Carry output and **its value is 6.796 ns.**



Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A	C	6.655	SLOW	2.247	FAST
A	S	6.409	SLOW	2.155	FAST
B	C	6.796	SLOW	2.278	FAST
B	S	6.519	SLOW	2.193	FAST

2 Full Adder

Full adder can be implemented with 2 half adder and or gate.

2.1 Verilog Codes

```
module full_adder(
   input A,B,CI,
   output CO,S
   );

wire half_adder1_s,half_adder1_c;
wire half_adder2_s,half_adder2_c;
(* dont_touch *) half_adder half_adder1(A,B,half_adder1_c,half_adder1_s);
(* dont_touch *) half_adder half_adder2(CI,half_adder1_s,half_adder2_c,half_adder2_s);
(* dont_touch *) assign CO = half_adder1_c | half_adder2_c;
(* dont_touch *) assign S = half_adder2_s;
endmodule
```

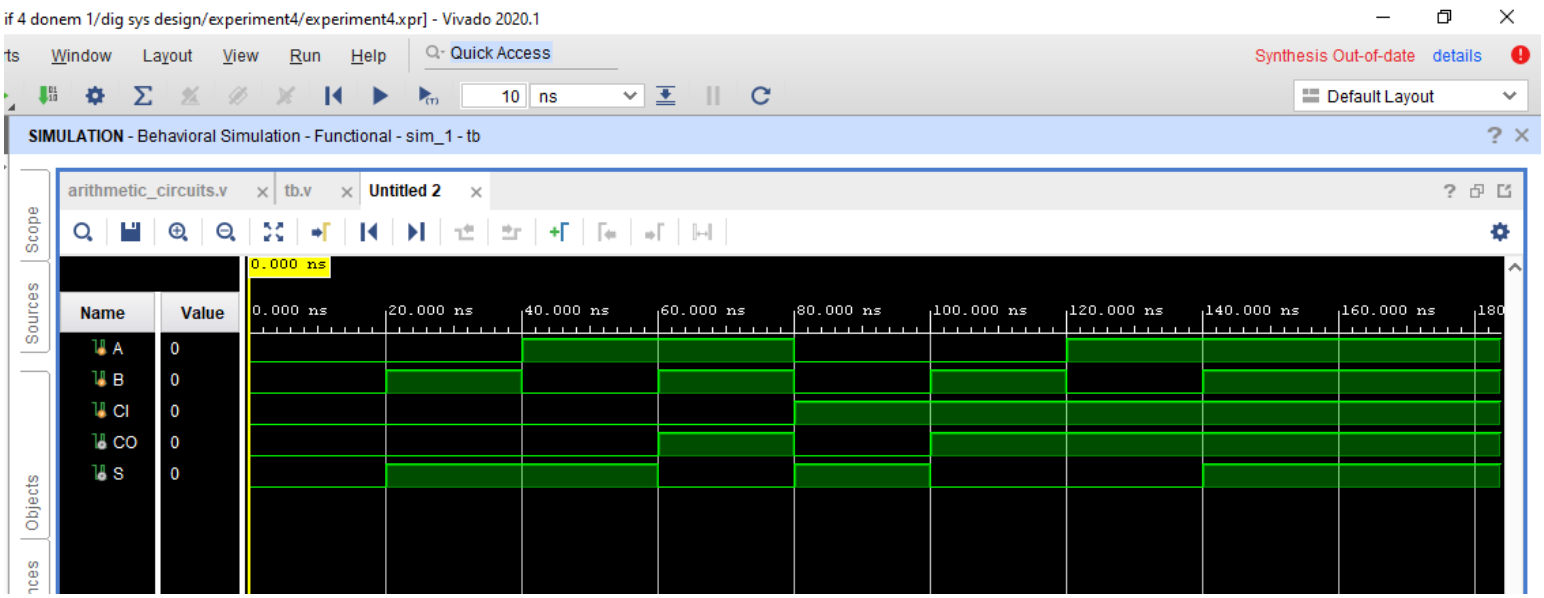
Arithmetic_circuits.v

```
module tb();
   reg A,B,CI;
   wire CO,S;
   full_adder FA(A,B,CI,CO,S);
   initial begin
   CI = 1'b0;
   A = 1'b0; B = 1'b0; #20;
   A = 1'b0; B = 1'b1; #20;
   A = 1'b1; B = 1'b0; #20;
   A = 1'b1; B = 1'b1; #20;
   CI = 1'b1;
   A = 1'b0; B = 1'b0; #20;
   A = 1'b0; B = 1'b1; #20;
   A = 1'b1; B = 1'b0; #20;
   A = 1'b1; B = 1'b1; #20;
   end
endmodule
```

Tb.v

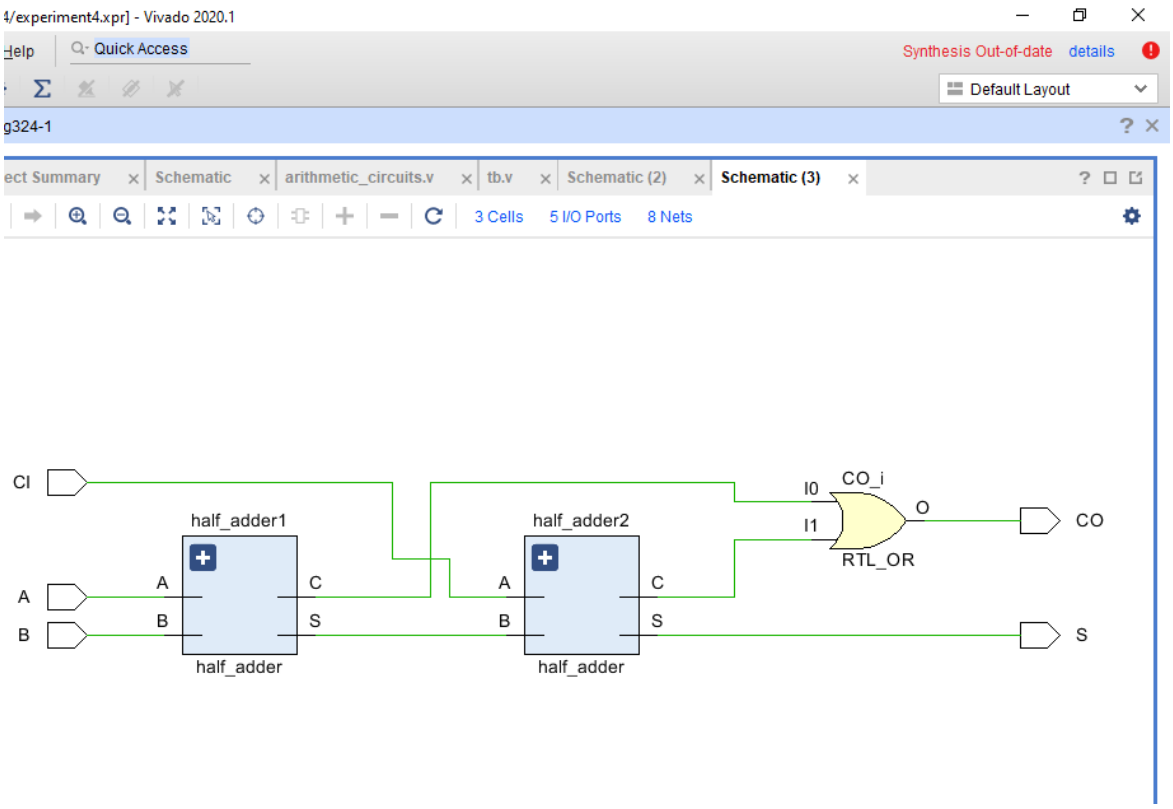
2.2 Behavioral Simulation

Obtained behavioral simulation is true according to truth table of full adder.



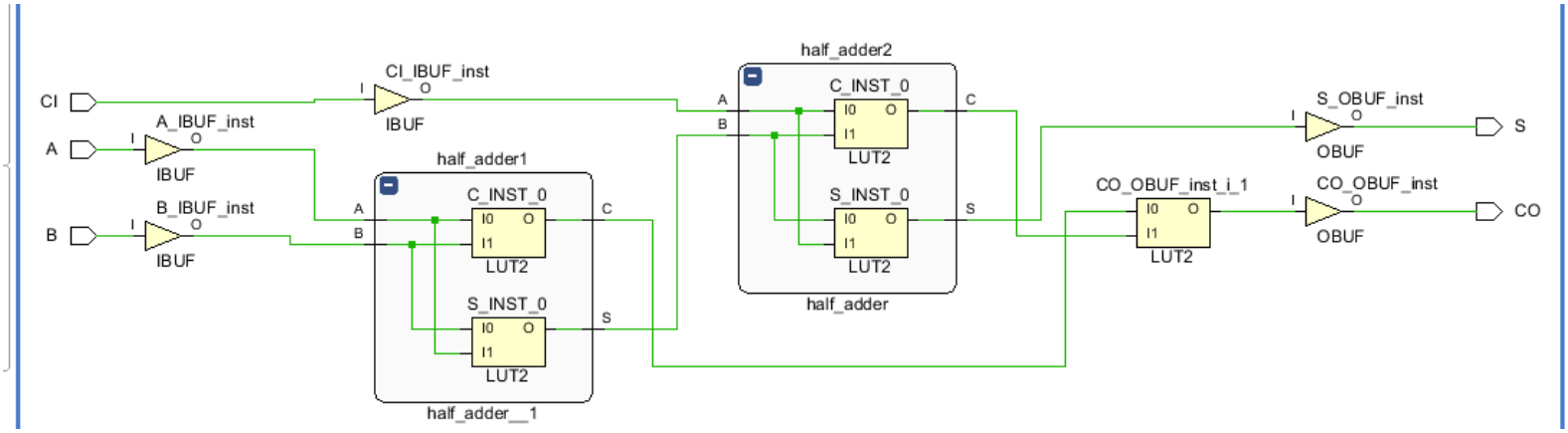
2.3 RTL Schematic

Obtained RTL Schematic is true according to planned circuit design.



2.4 Technology Schematic

As you can see in the figure below, technology schematic of the full adder was successfully obtained. I explained half adder LUTs in the previous section. In addition Full adder has extra LUT2 as OR gate. Therefore Full adder has 5 LUT2 and 5 buffers.



2.5 LUT usage

Full adder has **5 LUT2.**

Log | Reports | Design Runs | Power | DRC | Timing | **Utilization**

Hierarchy

Name	Slice LUTs (63400)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)
full_adder	5	2	5	5
half_adder1 (half_adder__1)	2	1	2	0
half_adder2 (half_adder)	2	1	2	0

2.6 Delays

Max pad to pad delay is between B and CO and it **is 7.930 ns.**

Tcl Console | Messages | Log | Reports | Design Runs | Power | DRC | **Timing**

- Inter-Clock Paths
- Other Path Groups
- User Ignored Paths
- Unconstrained Paths
- Datasheet
  - Input Ports Setup/Hold
  - Output Ports Clock-to-out
  - Combinational Delays
  - Setup between Clocks

Combinational Delays

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Pro Corner
B	CO	7.930	SLOW	2.435	FAST
A	CO	7.691	SLOW	2.346	FAST
CI	CO	7.667	SLOW	2.563	FAST
B	S	7.106	SLOW	2.401	FAST
A	S	6.866	SLOW	2.324	FAST
CI	S	6.841	SLOW	2.295	FAST

Timing Summary - impl_1 (saved) | **Timing Summary - timing_1**

3 Ripple Carry Adder

3.1 Codes

```
module ripple_carry_adder(
    input [3:0] A, B,
    input CI,
    output CO,
    output [3:0] S);

wire fac0, fac1, fac2;
(* dont_touch *) full_adder fa0(A[0],B[0],CI,fac0,S[0]);
(* dont_touch *) full_adder fa1(A[1],B[1],fac0,fac1,S[1]);
(* dont_touch *) full_adder fa2(A[2],B[2],fac1,fac2,S[2]);
(* dont_touch *) full_adder fa3(A[3],B[3],fac2,CO,S[3]);

endmodule
```

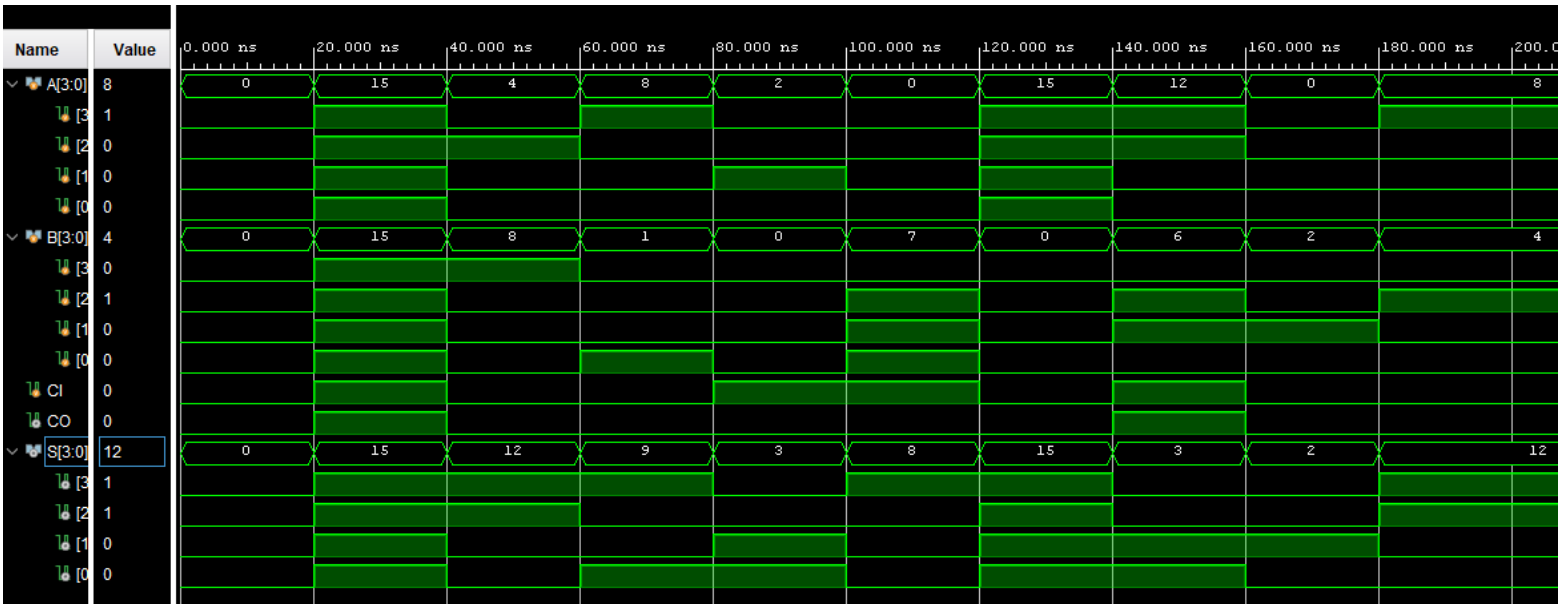
Arithmetic_circuits.v

```
module tb();
    //ripple carry
    reg [3:0] A,B;
    reg CI;
    wire CO;
    wire [3:0] S;
    ripple_carry_adder RAD(A,B,CI,CO,S);
    initial begin

    A = 4'b0000; B = 4'b0000; CI = 1'b0; #20; //min
    A = 4'b1111; B = 4'b1111; CI = 1'b1; #20; //max
    A = 4'b0100; B = 4'b1000; CI = 1'b0; #20;
    A = 4'b1000; B = 4'b0001; CI = 1'b0; #20;
    A = 4'b0010; B = 4'b0000; CI = 1'b1; #20;
    A = 4'b0000; B = 4'b0111; CI = 1'b1; #20;
    A = 4'b1111; B = 4'b0000; CI = 1'b0; #20;
    A = 4'b1100; B = 4'b0110; CI = 1'b1; #20;
    A = 4'b0000; B = 4'b0010; CI = 1'b0; #20;
    A = 4'b1000; B = 4'b0100; CI = 1'b0; #20;

    end
endmodule
```

Tb.v

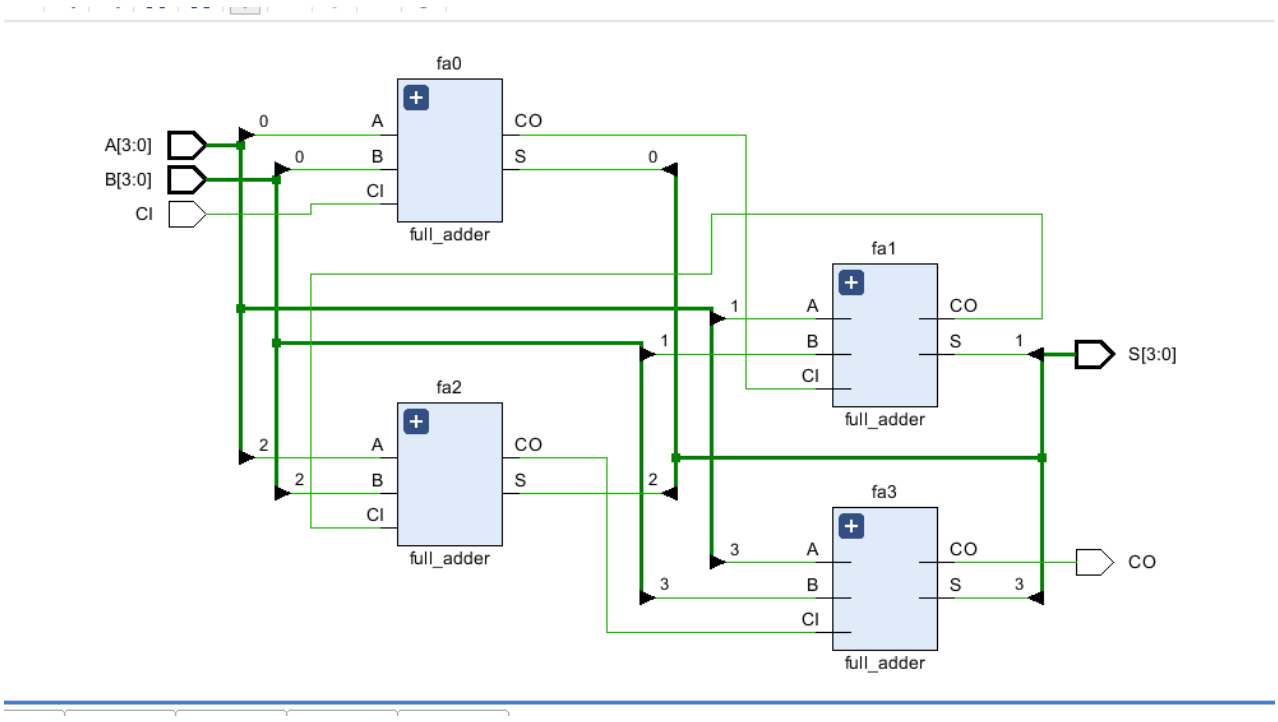
3.2 Behavioral Simulation

Behavioral Simulation was obtained successfully with minimum, maximum and other 8 random input values. Design was true according to randomly checked math operations.



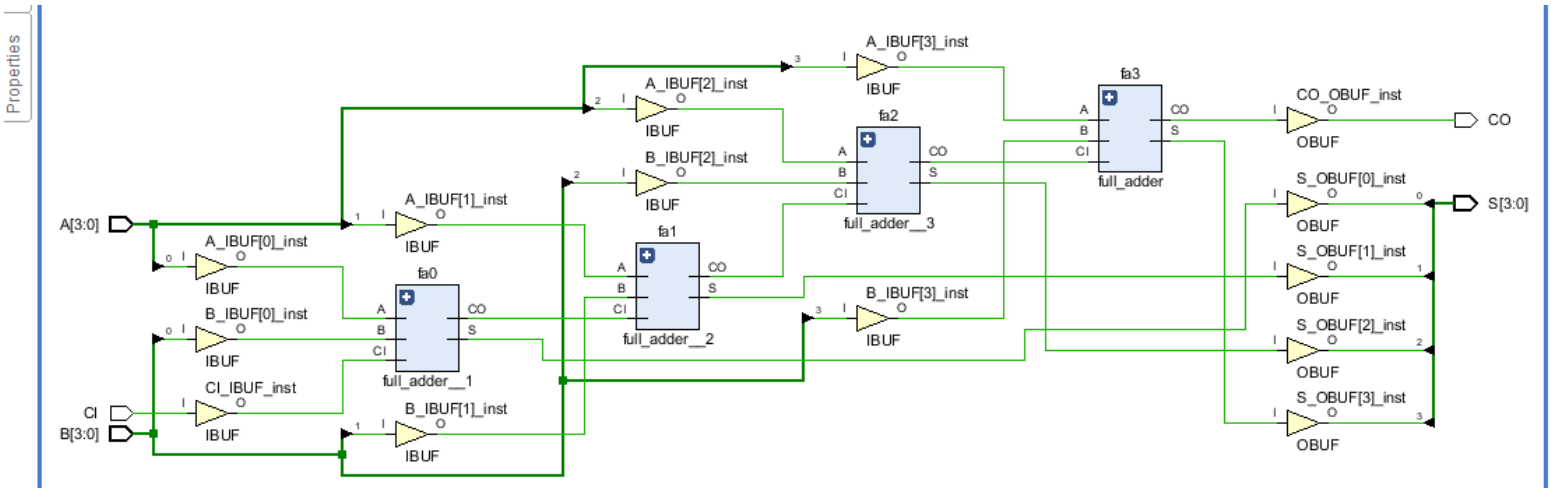
3.3 RTL Schematic

Obtained RTL Schematic is little bit confusing because of the components' places but it is true according to given circuit design.



3.4 Technology Schematic

As you can see in the figure below, technology schematic of the ripple carry adder was successfully obtained.



3.5 LUT usage

In total, there are 20 LUT2 and 14 IO buffers used. (Every full adder has 5 LUT2)

```
8. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| LUT2     |   20 |                 LUT |
| IBUF     |    9 |                  IO |
| OBUF     |    5 |                  IO |
+----------+------+---------------------+
```

3.6 Delays

There are too many input and output path are exist, but the path has the most delay is between **A[0] and CO and it is 12.950 ns.**

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	CO	12.950	SLOW	4.005	FAST
B[0]	CO	12.941	SLOW	4.047	FAST
A[0]	S[3]	12.158	SLOW	3.733	FAST
B[0]	S[3]	12.149	SLOW	3.776	FAST
CI	CO	12.056	SLOW	4.097	FAST
B[1]	CO	11.456	SLOW	3.614	FAST
A[1]	CO	11.325	SLOW	3.554	FAST
CI	S[3]	11.263	SLOW	3.825	FAST
A[0]	S[2]	10.774	SLOW	3.267	FAST
B[0]	S[2]	10.765	SLOW	3.310	FAST
B[1]	S[3]	10.663	SLOW	3.342	FAST
A[2]	CO	10.558	SLOW	3.479	FAST
A[1]	S[3]	10.532	SLOW	3.282	FAST
B[2]	CO	10.522	SLOW	3.261	FAST
CI	S[2]	9.879	SLOW	3.359	FAST
A[2]	S[3]	9.765	SLOW	3.207	FAST
B[2]	S[3]	9.729	SLOW	2.989	FAST
A[0]	S[1]	9.447	SLOW	2.806	FAST
B[0]	S[1]	9.438	SLOW	2.848	FAST
B[1]	S[2]	9.279	SLOW	2.876	FAST
A[1]	S[2]	9.148	SLOW	2.816	FAST
CI	S[1]	8.553	SLOW	2.897	FAST
A[3]	CO	8.274	SLOW	2.634	FAST
B[1]	S[1]	8.212	SLOW	2.768	FAST
A[1]	S[1]	8.081	SLOW	2.782	FAST
A[0]	S[0]	7.873	SLOW	2.675	FAST
B[0]	S[0]	7.864	SLOW	2.663	FAST

ning Summary - timing_1

In addition, I investigated CO, carry output bit of last full adder. As inputs are getting closer to CO, path delays between them are getting lower and lower. Lowest path delay between CO and a input is 7.786 ns and corresponding input is B[3] that is the input of full adder 3.

Combinational Delays

From Port	To Po 1	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	CO	12.950	SLOW	4.005	FAST
A[1]	CO	11.325	SLOW	3.554	FAST
A[2]	CO	10.558	SLOW	3.479	FAST
A[3]	CO	8.274	SLOW	2.634	FAST
B[0]	CO	12.941	SLOW	4.047	FAST
B[1]	CO	11.456	SLOW	3.614	FAST
B[2]	CO	10.522	SLOW	3.261	FAST
B[3]	CO	7.786	SLOW	2.451	FAST
CI	CO	12.056	SLOW	4.097	FAST
A[0]	S[0]	7.873	SLOW	2.675	FAST

4 (8-bit) Ripple Carry Adder with Parameters

In this section, we are going to use generate structure and parameter feature of verilog. Instead creating adders with fixed size, we will determine the size of the adder later. (while IDE is working)

4.1 Codes

Wire fac[SIZE-1:0] are the wires that connect carry input and outputs between full adders.

```
module parametric_RCA #(parameter SIZE = 8)
(   input [SIZE-1:0] A, B,
    input CI,
    output CO,
    output [SIZE-1:0] S);

    wire fac[SIZE-1:0];//ith full adder carry bit

    genvar i;
    generate
        for (i = 0; i < SIZE; i = i + 1) begin: generated_fa
            //first fa gets CI as input
            if (i==0) begin: test1
                (* dont_touch *) full_adder (A[i],B[i],CI,fac[i],S[i]);
            end else
            //carry of the last fa is the CO
            if (i==SIZE-1) begin: test2
                (* dont_touch *) full_adder (A[i],B[i],fac[i-1],CO,S[i]);
            end else
            //fa's in the middle gets prev carry from prev fa,
            //send own carry to next fa
            (* dont_touch *) full_adder fa(A[i],B[i],fac[i-1],fac[i],S[i]);
        end
    endgenerate


endmodule
```

Arithmetic_circuit.v

```
module tb();
    parameter integer SIZE = 8; //parameter makes it constant
    reg [SIZE-1:0] A,B;
    reg CI;
    wire CO;
    wire [SIZE-1:0] S;
    parametric_RCA RAD(A,B,CI,CO,S);
    initial begin
```

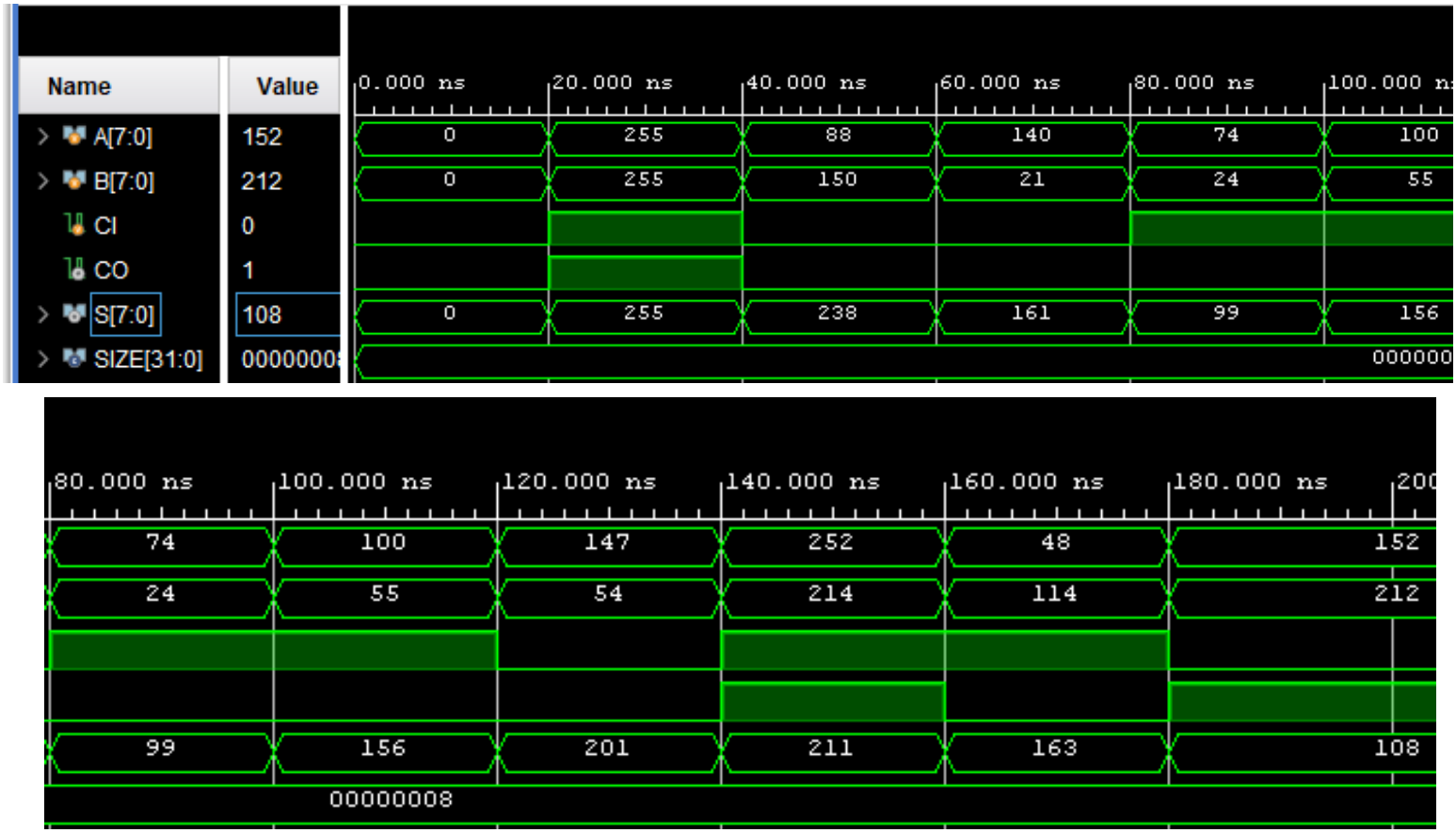
```
    A = 8'b00000000; B = 8'b00000000; CI = 1'b0; #20; //min
    A = 8'b11111111; B = 8'b11111111; CI = 1'b1; #20; //max
    A = 8'b01011000; B = 8'b10010110; CI = 1'b0; #20;
    A = 8'b10001100; B = 8'b00010101; CI = 1'b0; #20;
    A = 8'b01001010; B = 8'b00011000; CI = 1'b1; #20;
    A = 8'b01100100; B = 8'b00110111; CI = 1'b1; #20;
    A = 8'b10010011; B = 8'b00110110; CI = 1'b0; #20;
    A = 8'b11111100; B = 8'b11010110; CI = 1'b1; #20;
    A = 8'b00110000; B = 8'b01110010; CI = 1'b1; #20;
    A = 8'b10011000; B = 8'b11010100; CI = 1'b0; #20;

    end


endmodule
```

Tb.v

4.2 Behavioral Simulation

Behavioral Simulation was obtained successfully with minimum, maximum and other 8 random input values. Design was true according to randomly checked math operations.

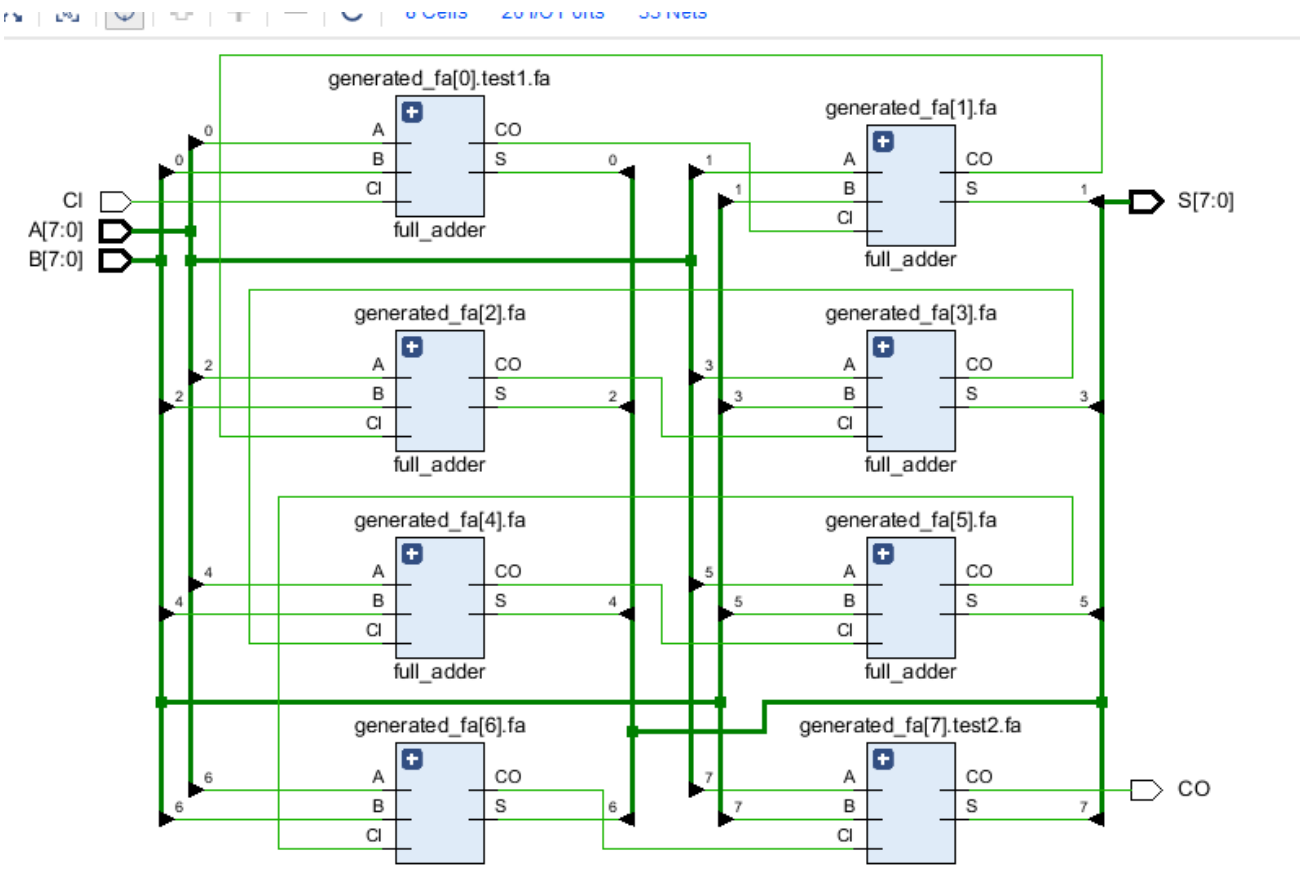
For example 252 + 214 + CI(1) = CO(256) + 211 is correct.



4.3 RTL schematic

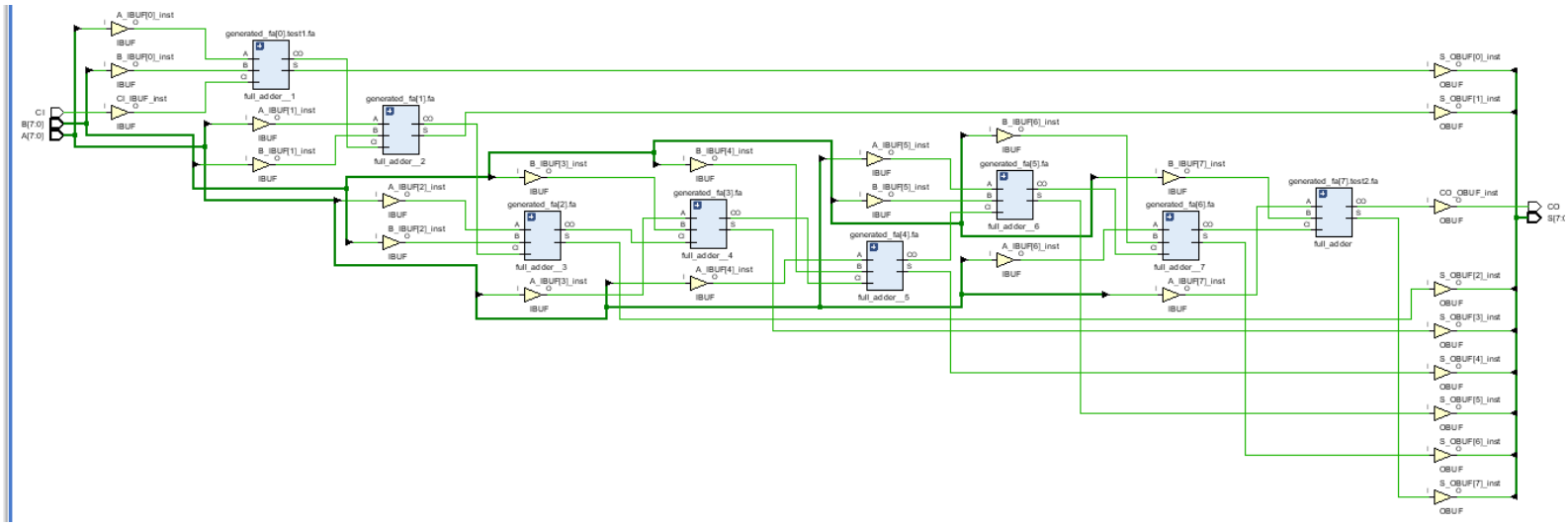
What is different while comparing with the 4-b ripple carry adder?

RTL schematic is not different than I expect. Only structural difference between 8-b and 4-b ripple carry adder are the number of the full adders. Like it should be, there are 8 full adder used in the 8-b ripple carry adder. Another difference but not structural is the name of the components, name of the adders were determined by the label I gave in the generate block.

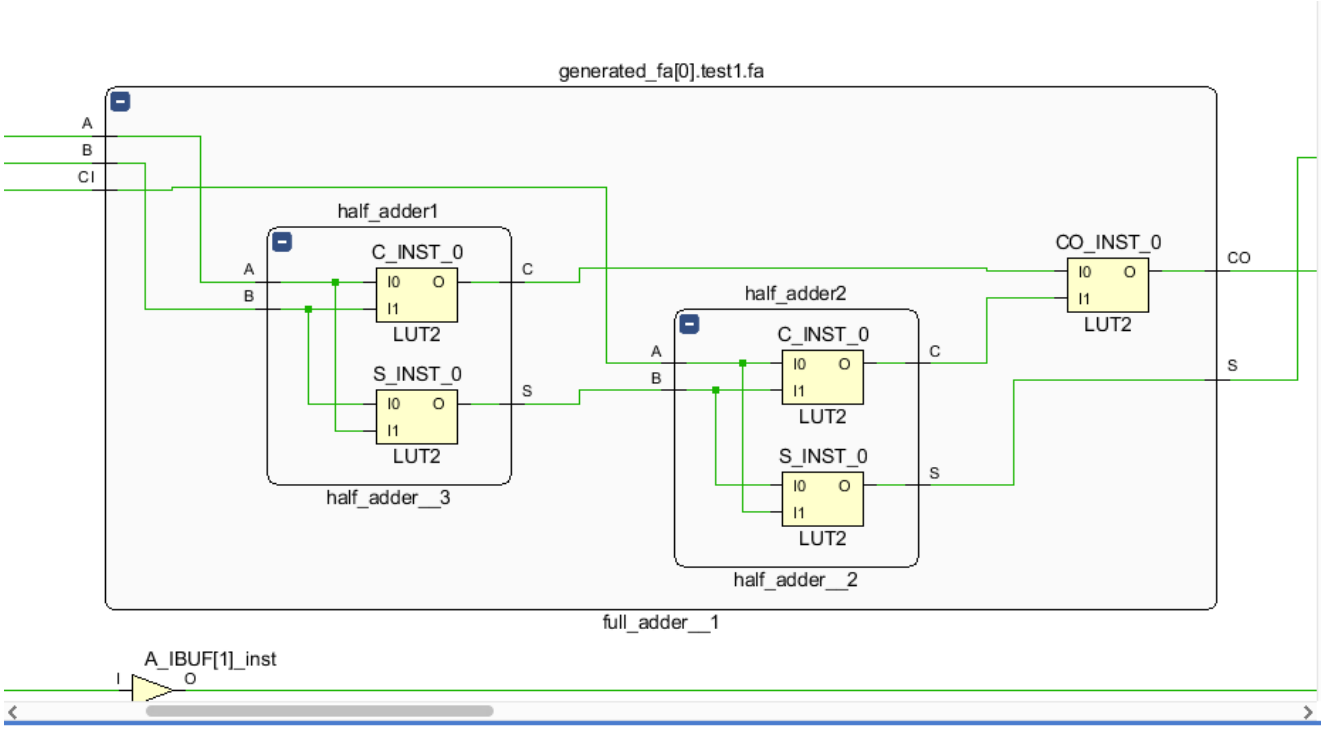


4.4 Technology Schematic

Like the RTL schematic there is no big difference between technology schematic of the 4-b adder and 8-b ripple carry adder. Number of the full adder component are 8 as expected. Like RTL schematics, name of the components were determined by the label I gave in the generate phase.



Example full adder is shown in the below.



4.5 Usage

There are 40 LUT2 were used in the design. 26 IO buffers were used as well.

Schematic (2) | impl_1_place_report_utilization_0 - impl_1

esign/experiment4/experiment4.runs/impl_1/parametric_RCA_utilization_placed.rpt

Read-only

```
+-------------+------+-------+-----------+-------+


8. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| LUT2     |   40 |                 LUT |
| IBUF     |   17 |                  IO |
| OBUF     |    9 |                  IO |
+----------+------+---------------------+


9. Black Boxes
--------------
```

ematic (2) | impl_1_place_report_utilization_0 - impl_1

ign/experiment4/experiment4.runs/impl_1/parametric_RCA_utilization_placed.rpt

Read-only

```
+-------------------------------------------+------+-------+---
|                 Site Type                 | Used | Fixed | Av
+-------------------------------------------+------+-------+---
| Slice                                     |   10 |     0 |
|   SLICEL                                  |   10 |     0 |
|   SLICEM                                  |    0 |     0 |
| LUT as Logic                              |   36 |     0 |
|   using O5 output only                    |    0 |       |
|   using O6 output only                    |   32 |       |
|   using O5 and O6                         |    4 |       |
| LUT as Memory                             |    0 |     0 |
|   LUT as Distributed RAM                  |    0 |     0 |
|   LUT as Shift Register                   |    0 |     0 |
| Slice Registers                           |    0 |     0 |
|   Register driven from within the Slice   |    0 |       |
|   Register driven from outside the Slice  |    0 |       |
| Unique Control Sets                       |    0 |       |
+-------------------------------------------+------+-------+---
```

ematic (2) | impl_1_place_report_utilization_0 - impl_1

gn/experiment4/experiment4.runs/impl_1/parametric_RCA_utilization_placed.rpt

Read-only

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|         Site Type       | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |   36 |     0 |     63400 |  0.06 |
|   LUT as Logic          |   36 |     0 |     63400 |  0.06 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |    0 |     0 |    126800 |  0.00 |
|   Register as Flip Flop |    0 |     0 |    126800 |  0.00 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+


1.1 Summary of Registers by Type
```

5 Carry Look Ahead Adder

5.1 Design

$G_i = A_i B_i \Rightarrow G_0 = A_0 B_0 \quad G_1 = A_1 B_1 \quad G_2 = A_2 B_2 \quad G_3 = A_3 B_3$

$P_i = A_i \oplus B_i \Rightarrow P_0 = A_0 \oplus B_0 \quad P_1 = A_1 \oplus B_1 \quad P_2 = A_2 \oplus B_2 \quad P_3 = A_3 \oplus B_3$

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

$C_1 = G_0 + P_0 C_{IN}$

$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_{IN}) = G_1 + P_1 G_0 + P_1 P_0 C_{IN}$

$C_3 = G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 (G_0 + P_0 C_{IN})) = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{IN}$

$C_4 = G_3 + P_3 C_3 = \cdots = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{IN}$

$CO = C_4$

Every C_i depends G_i, P_i, C_{IN}

5.2 Codes

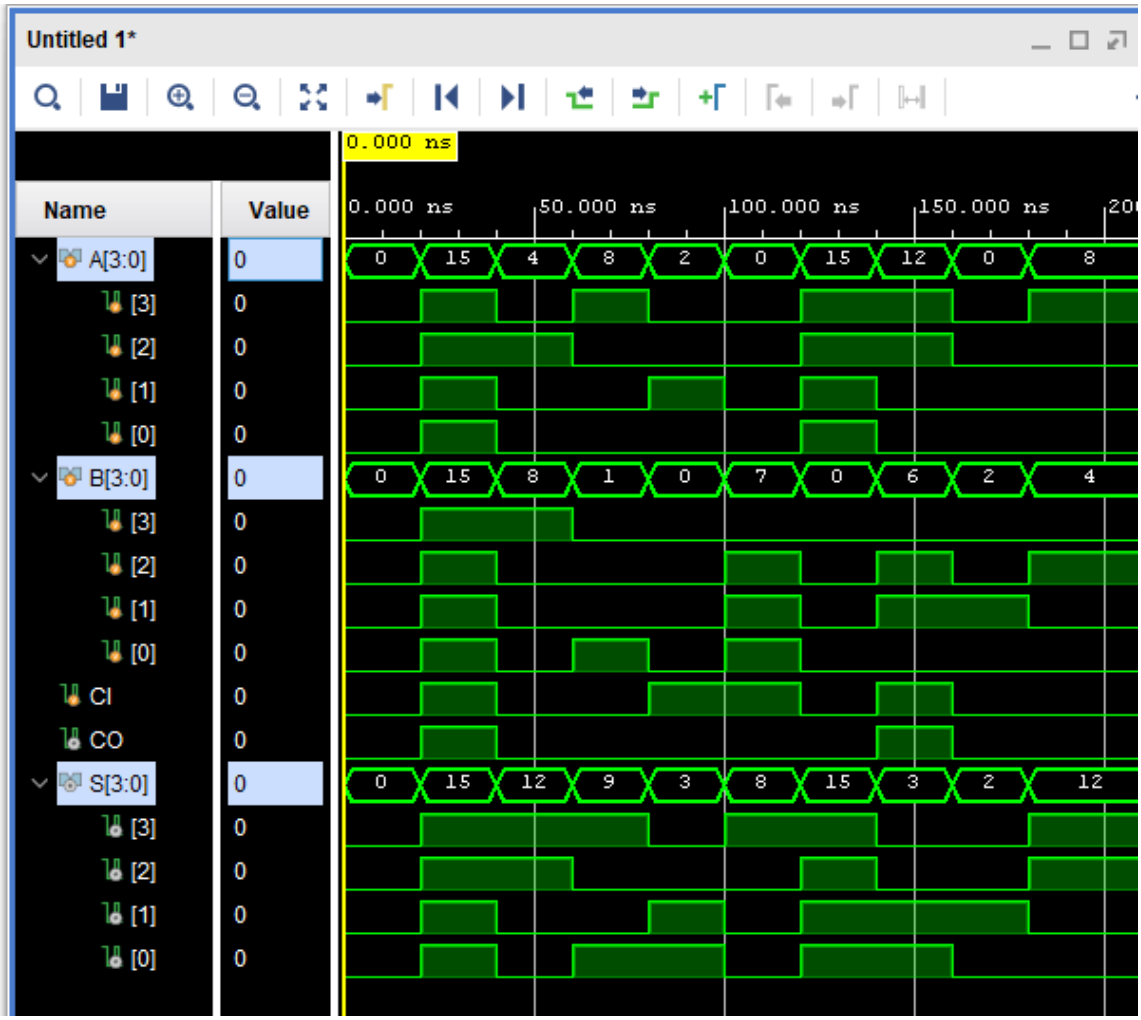
```
module CLA (
   input [3:0] A, B,
   input CI,
   output CO,
   output [3:0] S);

wire p0,p1,p2,p3;
(* dont_touch *) assign p0 = A[0] ^ B[0];
(* dont_touch *) assign p1 = A[1] ^ B[1];
(* dont_touch *) assign p2 = A[2] ^ B[2];
(* dont_touch *) assign p3 = A[3] ^ B[3];

wire g0,g1,g2,g3;
(* dont_touch *) assign g0 = A[0] & B[0];
(* dont_touch *) assign g1 = A[1] & B[1];
(* dont_touch *) assign g2 = A[2] & B[2];
```

```
(* dont_touch *) assign g3 = A[3] & B[3];

wire c1,c2,c3;
(* dont_touch *) assign c1 = g0 | (p0&CI);
(* dont_touch *) assign c2 = g1 | (p1&g0) | (p1&p0&CI);
(* dont_touch *) assign c3 = g2 | (p2&g1) | (p2&p1&g0) | (p2&p1&p0&CI);
(* dont_touch *) assign CO = g3 | (p3&g2) | (p3&p2&g1) | (p3&p2&p1&g0) |
(p3&p2&p1&p0&CI);

(* dont_touch *) assign S[0] = p0 ^ CI;
(* dont_touch *) assign S[1] = p1 ^ c1;
(* dont_touch *) assign S[2] = p2 ^ c2;
(* dont_touch *) assign S[3] = p3 ^ c3;
endmodule
```

```
module tb();
reg [3:0] A,B;
   reg CI;
   wire CO;
   wire [3:0] S;
   //ripple_carry_adder RAD(A,B,CI,CO,S);
   CLA cla(A,B,CI,CO,S);
   initial begin

   A = 4'b0000; B = 4'b0000; CI = 1'b0; #20; //min
   A = 4'b1111; B = 4'b1111; CI = 1'b1; #20; //max
   A = 4'b0100; B = 4'b1000; CI = 1'b0; #20;
   A = 4'b1000; B = 4'b0001; CI = 1'b0; #20;
   A = 4'b0010; B = 4'b0000; CI = 1'b1; #20;
   A = 4'b0000; B = 4'b0111; CI = 1'b1; #20;
   A = 4'b1111; B = 4'b0000; CI = 1'b0; #20;
   A = 4'b1100; B = 4'b0110; CI = 1'b1; #20;
   A = 4'b0000; B = 4'b0010; CI = 1'b0; #20;
   A = 4'b1000; B = 4'b0100; CI = 1'b0; #20;

   End
endmodule
```

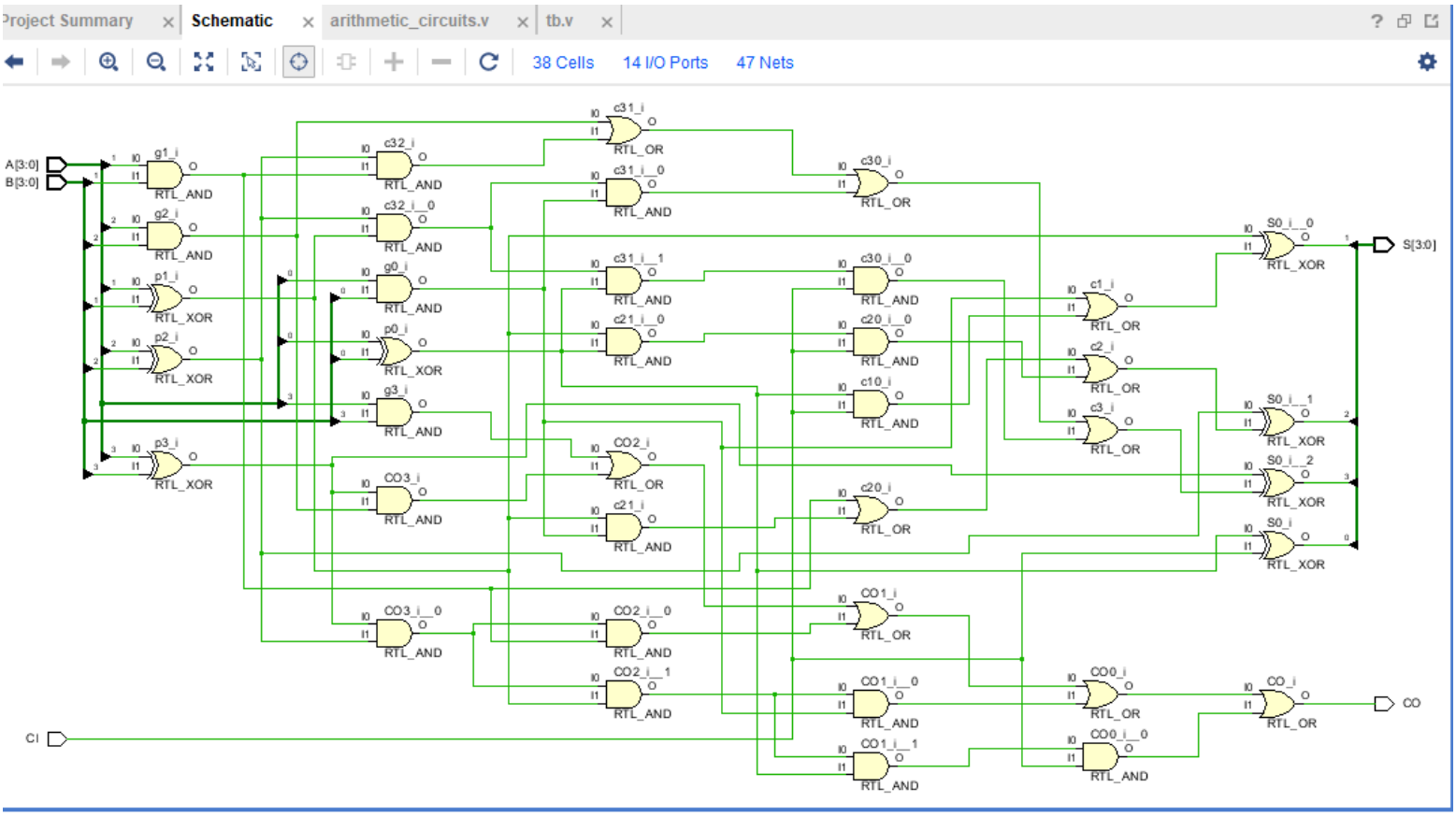
5.3 Behavioral Simulation

Behavioral simulation was obtained successfully.



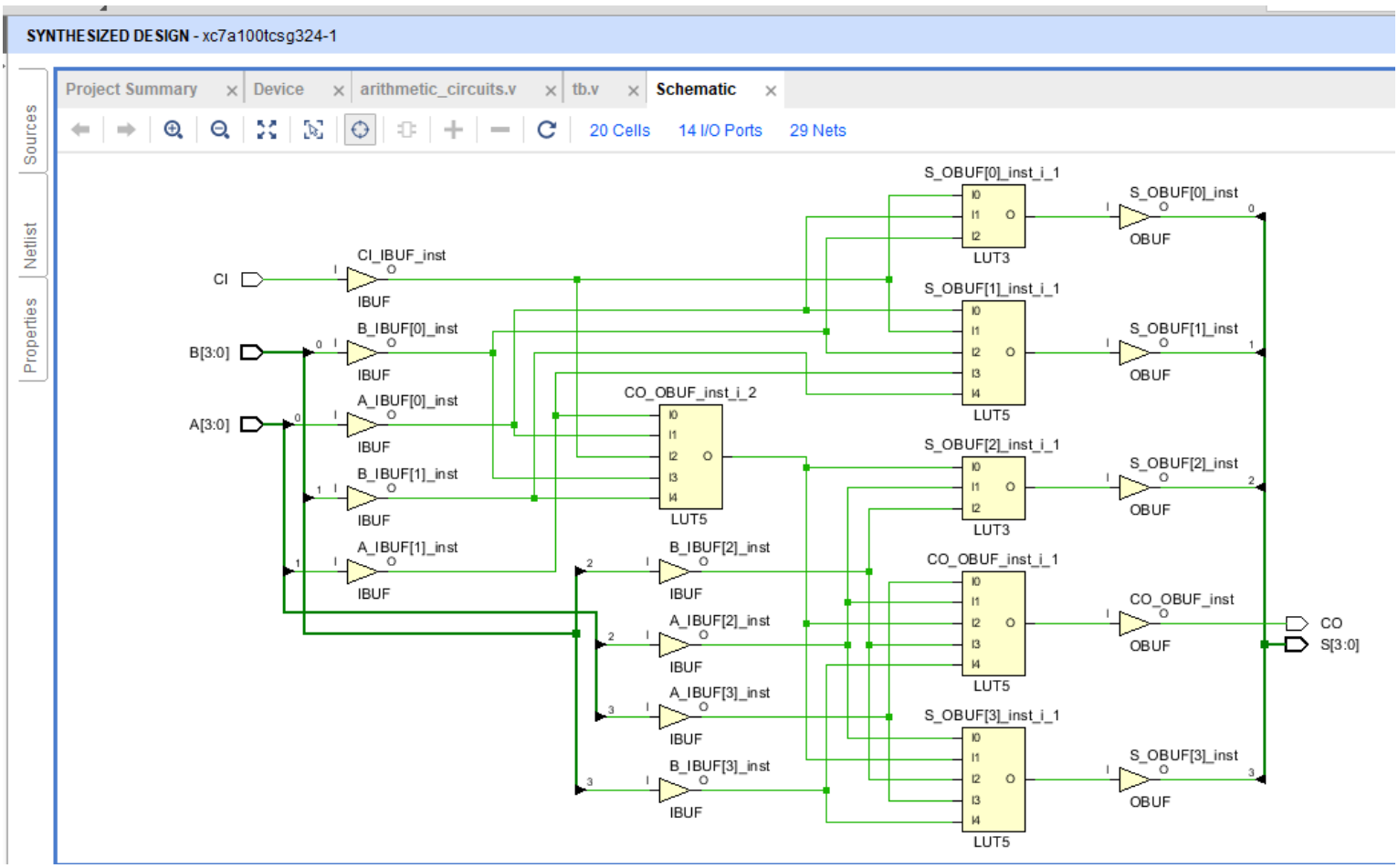
5.4 RTL Schematic

As you can see in the figure, CLA looks more parallel circuit than ripple carry adder. It looks exactly what I was planning to design.



5.5 Technology Schematic

As I understand, I even used “don’t touch” words to make them unoptimized, Vivado made optimization operation on my circuit design and used bigger LUTs instead using small LUT2’s to realize OR, AND and XOR operations.



5.6 Usage

As it is possible to count on the implemented design**, there are 6 LUTs used.** Details like distribution are in the following figures.

experiment4 - [C:/Users/HP/Desktop/sinif 4 donem 1/dig sys design/experiment4/experiment4.xpr] - Vivado 2020.1

File Edit Flow Tools Reports Window Layout View Help Quick Access

Implementation Complete

Timing Analysis

Flow Navigator

Schematic

SYNTHESIS

Run Synthesis

Open Synthesized Design

Constraints Wizard

Edit Timing Constraints

Set Up Debug

Report Timing Summary

Report Clock Networks

Report Clock Interaction

Report Methodology

Report DRC

Report Noise

Report Utilization

Report Power

Schematic

IMPLEMENTATION

Run Implementation

Open Implemented Design

Constraints Wizard

Edit Timing Constraints

IMPLEMENTED DESIGN - xc7a100tcsg324-1

Sources Netlist Source File Properties

CLA

Nets (29)

Leaf Cells (20)

Device arithmetic_circuits.v tb.v impl_1_place_report_utilization_0 - impl_1

C:/Users/HP/Desktop/sinif 4 donem 1/dig sys design/experiment4/experiment4.runs/impl_1/CLA_utilization_placed.rpt

Read-only

```
29  --------------
30
31  +-------------------------+------+-------+-----------+-------+
32  |        Site Type        | Used | Fixed | Available | Util% |
33  +-------------------------+------+-------+-----------+-------+
34  | Slice LUTs              |    4 |     0 |     63400 | <0.01 |
35  |   LUT as Logic          |    4 |     0 |     63400 | <0.01 |
36  |   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
37  | Slice Registers         |    0 |     0 |    126800 |  0.00 |
38  |   Register as Flip Flop |    0 |     0 |    126800 |  0.00 |
39  |   Register as Latch     |    0 |     0 |    126800 |  0.00 |
40  | F7 Muxes                |    0 |     0 |     31700 |  0.00 |
41  | F8 Muxes                |    0 |     0 |     15850 |  0.00 |
42  +-------------------------+------+-------+-----------+-------+
43
44
45  1.1 Summary of Registers by Type
46  --------------------------------
47
48  +-------+--------------+-------------+--------------+
```

Tcl Console Messages Log Reports Design Runs Power DRC Timing

Report	Type	Options	Modified	Size
impl_1_place_report_io_0	report_io		12/6/20, 6:38 PM	96.6 KB
impl_1_place_report_utilization_0	report_utilization		12/6/20, 6:38 PM	8.0 KB

1:1 Read-only File Text

4.xpr] - Vivado 2020.1

Quick Access

Implementation Complete

Timing Analysis

Device arithmetic_circuits.v tb.v impl_1_place_report_utilization_0 - impl_1

C:/Users/HP/Desktop/sinif 4 donem 1/dig sys design/experiment4/experiment4.runs/impl_1/CLA_utilization_placed.rpt

Read-only

```
68  |                  Site Type                  | Used | Fixed | Available | Util% |
69  +---------------------------------------------+------+-------+-----------+-------+
70  | Slice                                       |    1 |     0 |     15850 | <0.01 |
71  |   SLICEL                                    |    1 |     0 |           |       |
72  |   SLICEM                                    |    0 |     0 |           |       |
73  | LUT as Logic                                |    4 |     0 |     63400 | <0.01 |
74  |   using O5 output only                      |    0 |       |           |       |
75  |   using O6 output only                      |    2 |       |           |       |
76  |   using O5 and O6                           |    2 |       |           |       |
77  | LUT as Memory                               |    0 |     0 |     19000 |  0.00 |
78  |   LUT as Distributed RAM                    |    0 |     0 |           |       |
79  |   LUT as Shift Register                     |    0 |     0 |           |       |
80  | Slice Registers                             |    0 |     0 |    126800 |  0.00 |
81  |   Register driven from within the Slice     |    0 |       |           |       |
82  |   Register driven from outside the Slice    |    0 |       |           |       |
83  | Unique Control Sets                         |    0 |       |     15850 |  0.00 |
84  +---------------------------------------------+------+-------+-----------+-------+
85  * * Note: Available Control Sets calculated as Slice * 1, Review the Control Sets Report for
86
87
```

Device arithmetic_circuits.v tb.v impl_1_place_report_utilization_0 - impl_1

C:/Users/HP/Desktop/sinif 4 donem 1/dig sys design/experiment4/experiment4.runs/impl_1/CLA_utilization_placed.rpt

Read-only

```
164 | STARTUPE2   |    0 |     0 |         1 |  0.00 |
165 | XADC        |    0 |     0 |         1 |  0.00 |
166 +-------------+------+-------+-----------+-------+
167
168
169 8. Primitives
170 -------------
171
172 +----------+------+---------------------+
173 | Ref Name | Used | Functional Category |
174 +----------+------+---------------------+
175 | IBUF     |    9 |                  IO |
176 | OBUF     |    5 |                  IO |
177 | LUT5     |    4 |                 LUT |
178 | LUT3     |    2 |                 LUT |
179 +----------+------+---------------------+
180
181
182 9. Black Boxes
183 --------------
```

n Runs Power DRC Timing

Type	Options	Modified	Size
report_io		12/6/20, 6:38 PM	96.6 KB
report_utilization		12/6/20, 6:38 PM	8.0 KB

5.7 Delays

Max path delay is 7.862 ns and it is between CI and S[2]. It was 12.950 ns for 4-b ripple carry adder. It is clear that CLA is much faster than ripple carry adder. Main reason behind this advantage is that CLA enables parallel design instead the design needs values as a sequence.

Combinational Delays

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
CI	S[2]	7.862	SLOW	2.662	FAST
CI	CO	7.825	SLOW	2.669	FAST
B[0]	S[2]	7.733	SLOW	2.632	FAST
A[0]	S[2]	7.701	SLOW	2.619	FAST
B[0]	CO	7.697	SLOW	2.638	FAST
A[0]	CO	7.665	SLOW	2.625	FAST
A[1]	S[2]	7.649	SLOW	2.603	FAST
A[1]	CO	7.612	SLOW	2.609	FAST
CI	S[3]	7.478	SLOW	2.531	FAST
A[3]	CO	7.435	SLOW	2.511	FAST
A[2]	CO	7.424	SLOW	2.509	FAST
B[0]	S[3]	7.350	SLOW	2.500	FAST
A[0]	S[3]	7.317	SLOW	2.487	FAST
CI	S[1]	7.314	SLOW	2.463	FAST
B[1]	S[2]	7.265	SLOW	2.458	FAST
A[1]	S[3]	7.265	SLOW	2.471	FAST
B[2]	CO	7.239	SLOW	2.421	FAST
B[1]	CO	7.229	SLOW	2.464	FAST
B[0]	S[1]	7.186	SLOW	2.432	FAST
A[0]	S[0]	7.163	SLOW	2.441	FAST
A[0]	S[1]	7.151	SLOW	2.417	FAST
A[1]	S[1]	7.099	SLOW	2.405	FAST
A[2]	S[2]	7.073	SLOW	2.404	FAST
A[3]	S[3]	7.054	SLOW	2.377	FAST
A[2]	S[3]	7.045	SLOW	2.375	FAST
B[3]	CO	6.943	SLOW	2.321	FAST
B[1]	S[3]	6.881	SLOW	2.326	FAST

In addition, output carry bit CO and its path delays are shown in the figure below. All path delays are almost equal each other. This can be possible with parallel design.

Combinational Delays

From Port	To Po	Max Delay	Max Process Corner	Min Delay	Min Pr Corne
A[0]	CO	7.665	SLOW	2.625	FAST
A[1]	CO	7.612	SLOW	2.609	FAST
A[2]	CO	7.424	SLOW	2.509	FAST
A[3]	CO	7.435	SLOW	2.511	FAST
B[0]	CO	7.697	SLOW	2.638	FAST
B[1]	CO	7.229	SLOW	2.464	FAST
B[2]	CO	7.239	SLOW	2.421	FAST
B[3]	CO	6.943	SLOW	2.321	FAST
CI	CO	7.825	SLOW	2.669	FAST
A[0]	S[0]	7.163	SLOW	2.441	FAST

6 Ripple Carry Adder for Signed Numbers

Before the report of 6th section, I need to brief problems in this section. Firstly what I understand from this section is

- With using ripple carry adder design
- Create a adder circuit that adds two 4-b signed numbers
- Without overflow problem (and underflow)
- With changing the definition of CO (carry out bit) **only** using with carry bits of last two full adders. (In the instruction, it doesn't say **only** but says by interpreting bits of last two full adders. So I understand I need to use only these two)

However, it **is not possible** to create such a circuit.

Proof:

```
 1111 (-5)
 0010 (2)
+-------------     in this sum, c3=0 and c2=0. Fifth bit should be 1
 11101 (-3)

 0011 (3)
 0010 (2)
+-------------     in this sum, c3=0 and c2=0. Fifth bit should be 0
 00101 (5)

Fifth bit is CO according to the instruction and CO depends c3 and c2. CO=f(c3,c2)
But CO has different outputs while its inputs are the same (0,0)
Therefore, it is not possible to define a function CO that only depends c3 and c2 to
create this adder circuit.
```

However, it is possible to define the CO with using c3,c2 and s[3]. We need another information like s[3].

6.1 Codes

CO = f(c3,c2,s[3]) solution is as follows:

```
module signed_RCA(
   input [3:0] A, B,
   input CI,
   output CO,
   output [3:0] S);
```

```
wire fac0, fac1, fac2, fac3;
(* dont_touch *) full_adder fa0(A[0],B[0],CI,fac0,S[0]);
(* dont_touch *) full_adder fa1(A[1],B[1],fac0,fac1,S[1]);
(* dont_touch *) full_adder fa2(A[2],B[2],fac1,fac2,S[2]);
(* dont_touch *) full_adder fa3(A[3],B[3],fac2,fac3,S[3]);

//to prevent overflow
 (* dont_touch *) assign CO = ((fac3 ^ fac2) & fac3) |(!(fac3 ^ fac2) & S[3]);
endmodule
```

Arithmetic_circuits.v

To clarify how CO was defined in the code above we need to look behavior the summation in terms of overflow.

There are two options while adding two numbers: sum either has overflow/underflow or doesn't have overflow/underflow.

- If the sum has overflow/underflow, result should be the output of the ripple carry adder. (5th bit,CO is c3).
- If the sum doesn't have overflow/underflow, result should be the output of the ripple carry adder without c3. But we need to define 5th bit because output should be 5-b signed number.
    - We need to define 5th bit 0, if sum is positive in order not to change number and its positivity. (5th bit,CO is 0) In addition 4th bit is 0 so CO=s[3]
    - We need to define 5th bit 1, if sum is negative in order not to change number and its negativity. (5th bit,CO is 1) In addition 4th bit is 1 so CO=s[3]
- How to detect overflow/underflow is occurred: (fac3 ^ fac2)

With using information above we can defined CO as

CO = (owerflow & fac3) | (!owerflow & S[3]);

CO = ((fac3 ^ fac2) & fac3) | (!(fac3 ^ fac2) & S[3]);

Testbench:

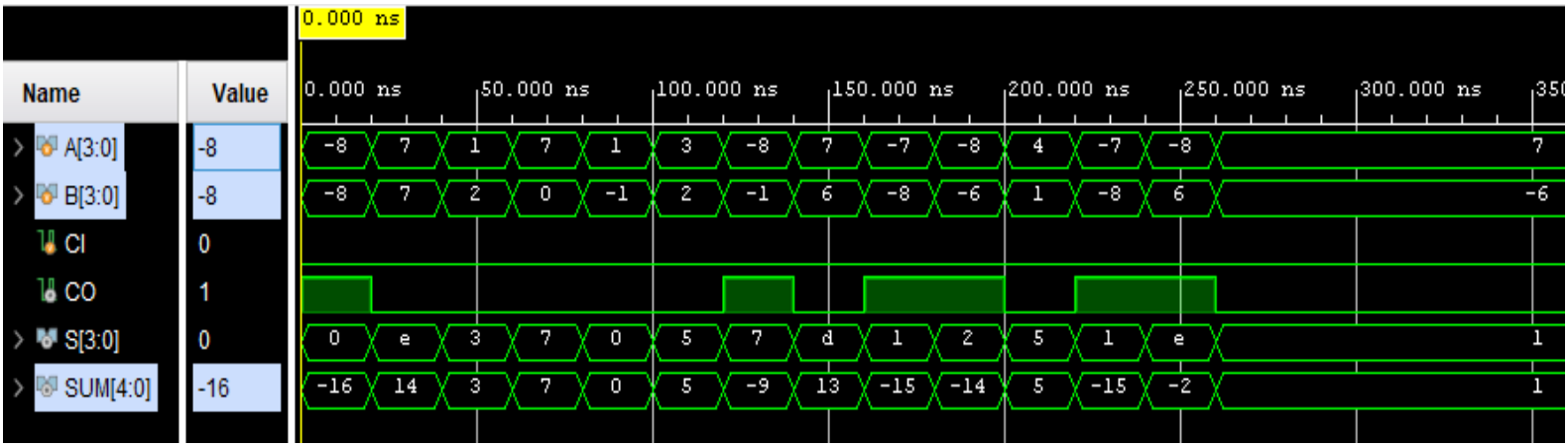
```
module tb();
  //signed RCA

   reg [3:0] A,B;
   reg CI;
   wire CO;
   wire [3:0] S;
```

```
    wire [4:0] SUM = {CO,S}; //signed sum
    signed_RCA srca(A,B,CI,CO,S);

    initial begin

    CI = 0;
    A = -4'd8; B = -4'd8; #20; //min
    A = 4'd7; B = 4'd7; #20; //max
    A = 4'd1; B = 4'd2; #20;
    A = 4'd7; B = 4'd0; #20;
    A = 4'd1; B = -4'd1; #20;
    A = 4'd3; B = 4'd2; #20;
    A = -4'd8; B = -4'd1; #20;
    A = 4'd7; B = 4'd6; #20;
    A = -4'd7; B = -4'd8; #20;
    A = -4'd8; B = -4'd6; #20;

    A = 4'd4; B = 4'd1; #20;
    A = -4'd7; B = 4'd8; #20;
    A = -4'd8; B = 4'd6; #20;
    A = 4'd7; B = -4'd6; #20;
    end

endmodule
```

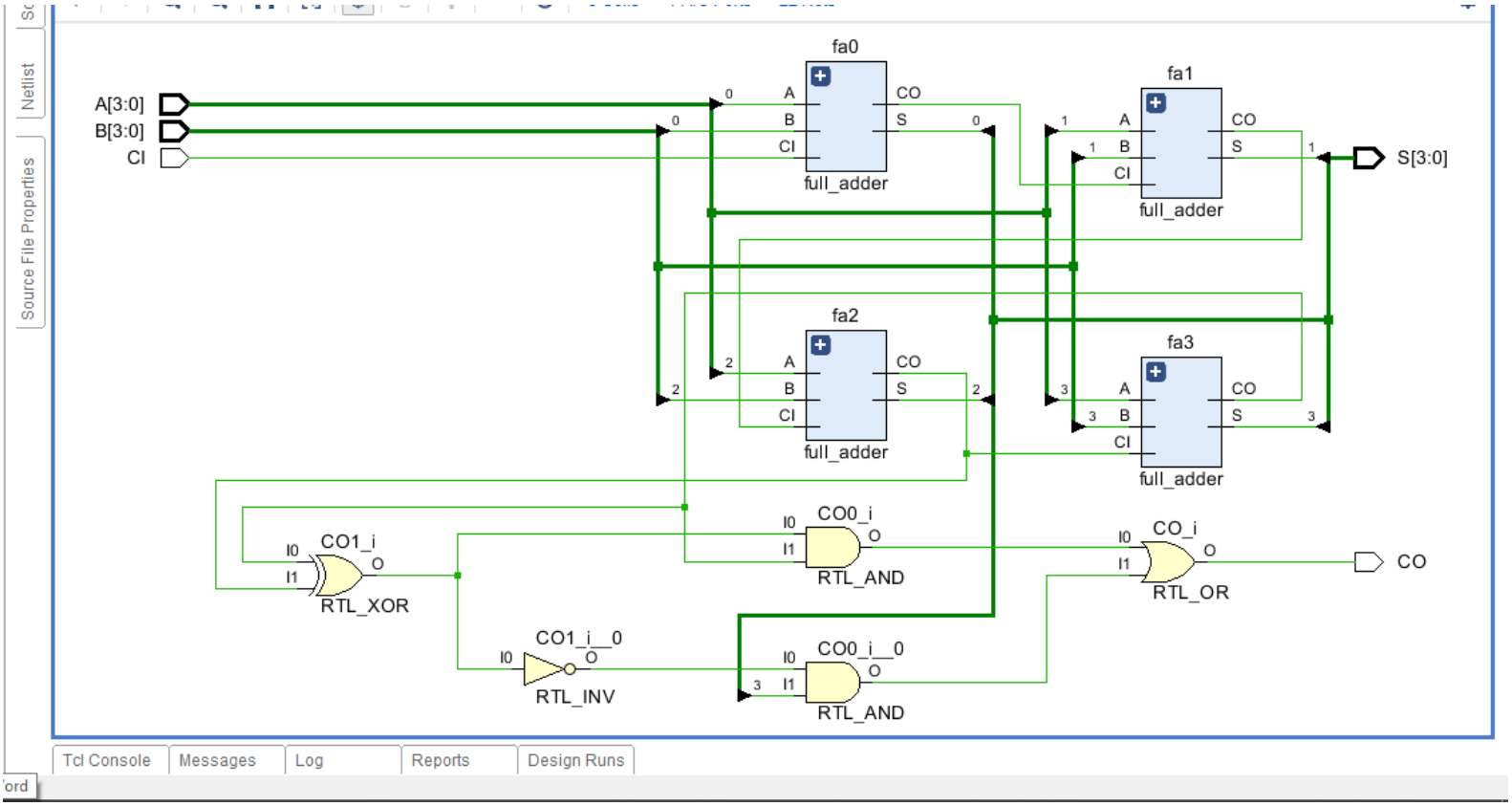
Tb.v

6.2 Behavioral Simulaton

Behavioral simulation was obtained **completely successfully.** All the results are **true**. A and B are the 4-b signed inputs and SUM is the 5-b signed output of sum. S is the 4-b ripple carry result. It was concatenated with CO and became SUM.

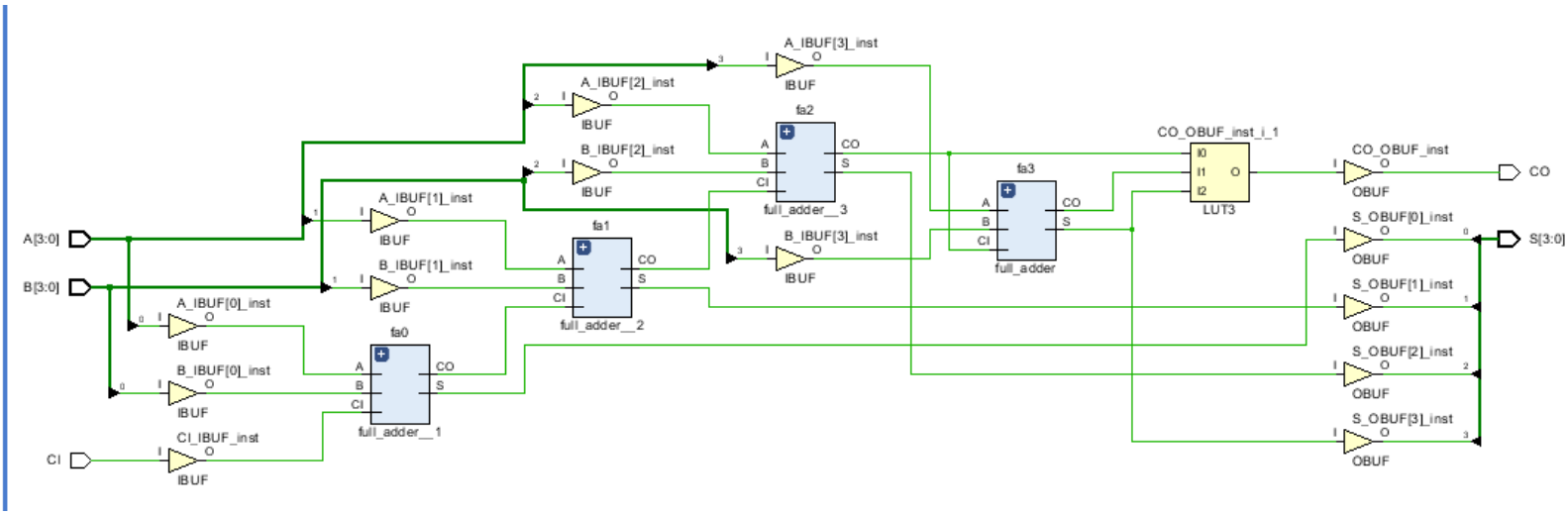


6.3 RTL Schematic



6.4 Technology Schematic

As you can see in the below, CO circuit was realized with one LUT3.



6.5 Timing

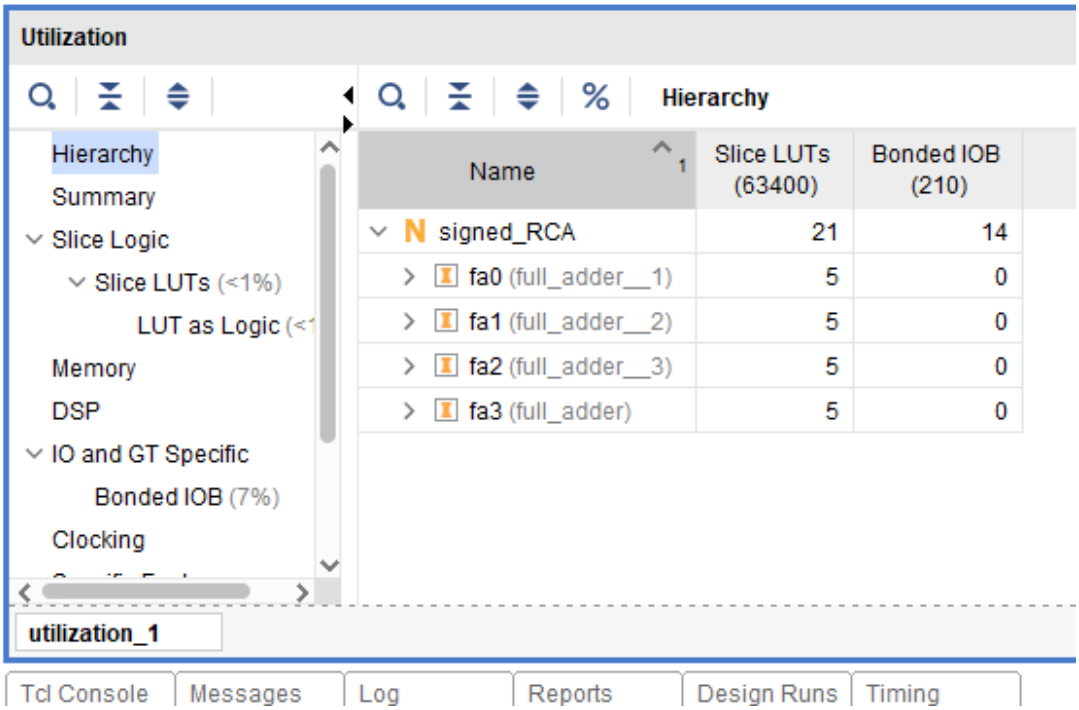
Because of the nature of the ripple carry adder, path delays between inputs and outputs are high while comparing the inputs and full adders in the middle. Process is working sequentially not parallel.

Combinational Delays

From Port	To Port	Ma	Max Process Corner	Min Delay	Min Process Corner
A[0]	CO	10.543	SLOW	3.602	FAST
B[0]	CO	10.543	SLOW	3.602	FAST
CI	CO	9.959	SLOW	3.615	FAST
A[0]	S[3]	9.397	SLOW	3.483	FAST
B[0]	S[3]	9.397	SLOW	3.483	FAST
A[1]	CO	9.356	SLOW	3.129	FAST
B[1]	CO	9.356	SLOW	3.129	FAST
CI	S[3]	8.813	SLOW	3.495	FAST
A[0]	S[2]	8.233	SLOW	3.007	FAST
B[0]	S[2]	8.233	SLOW	3.007	FAST
A[1]	S[3]	8.210	SLOW	3.010	FAST
B[1]	S[3]	8.210	SLOW	3.010	FAST
A[2]	CO	8.199	SLOW	2.656	FAST
B[2]	CO	8.199	SLOW	2.656	FAST
CI	S[2]	7.649	SLOW	3.019	FAST
A[0]	S[1]	7.076	SLOW	2.533	FAST
B[0]	S[1]	7.076	SLOW	2.533	FAST
A[2]	S[3]	7.053	SLOW	2.536	FAST
B[2]	S[3]	7.053	SLOW	2.536	FAST
A[1]	S[2]	7.046	SLOW	2.533	FAST
B[1]	S[2]	7.046	SLOW	2.533	FAST

6.6 Usage

Number of the LUTs were used in the circuit is 21 as you can see in the summary below. Every full adder uses 5 LUTs and CO uses 1 LUT at the end of the circuit. These results are expected.



Question:

Sometimes LUT number in the utilization report and primitive LUT implemented report are not the same. How is it possible?

Information obtained from https://forums.xilinx.com/t5/Implementation/Vivado-utilization-report/td-p/317517 in the below explain much.

In order to understand this report, you have to understand the structure of the 7 Series slice. The slice contains 4 LUTs and 8 flip-flops. Furthermore, in roughly 1 slice in 4, the LUTs can also be used as distributed memory.

The first section (Slice Logic) tells you what your design is using. Ignore the numbers beside the first line (the 21%). Within each slice you have

- 4 LUTs
- 8 FFs
- 2 F7MUX
- 1 F8MUX

These are the resources that the synthesis tool can use to build combinatorial functions (and a few other things). This first section tells you how many of these resources your design uses. If any one of these is approaching 100%, you are in trouble - your design is getting too big to fit in your part.

At the placement stage, these resources will get placed in slices - since there are multiple resources in a slice, you may end up with a very lightly used slice or a very heavily used slice. This is what the "Slices" number in the "Slice Logic Distribution" section is telling you - it tells you how many slices have at least one thing (LUT, FF, etc...) used in them. It is not necessarily a problem when this utilization gets high - you may have nearly 100% slice utilization but still not have a very full design.

The next sections in the "Slice Logic Distribution" tell you how the slices are used. In rougly 1 in 4 slices, the LUTs can also be used as distributed RAMs - that's what the "LUT as Memory" line is telling you. Note that the "LUT as Logic" and "LUT as Memory" numbers here are the same as they were in the "Slice Logic" section.

Utilization report indicates how much Slice Logic are full, not LUT distrubition as logic. It is possible to combine LUT inside of each other.