CS2610: Computer Organization and Architecture
Lab Report 2

Devansh Singh(111701011), Himanshu Jain(111701013)

Objective:

To analyse the efficiency of different adder/subtractor topologies in terms of processing delay and power consumption.

Problem:

To implement the below mentioned scenarios in Verilog:

a.) Implement an adder that uses carry generate and propagate logic to add A and B.

b.) Implement a subtractor that uses carry generate and propagate logic to subtract B from A.

c.) Implement a subtractor that using ripple carry adder in Lab 1 to subtract B from A.

Theory and Approach:

Full Adder: *A full adder is a logical circuit that performs an addition operation on three one-bit binary numbers. The full adder produces a sum of the two inputs and carry value. It gives a two bit output. First bit as sum and the second bit is the carry bit. Both these represents the sum of the given three one bit inputs.*

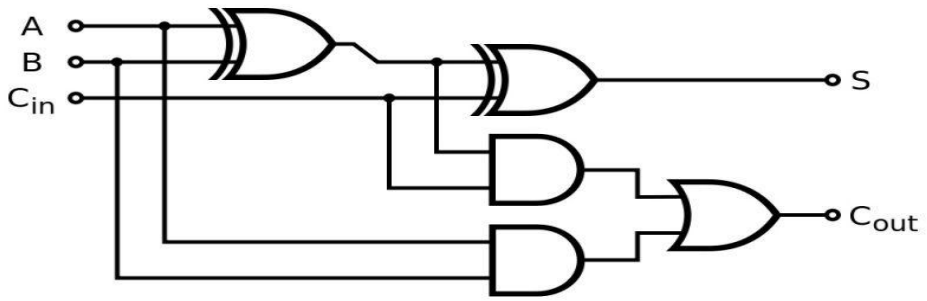


Figure 1: Full Adder Circuit

Code for full adder

```
module  full_adder(input a, input b, input c, output s, output C);

        assign  s =  a^b^c;
        assign  C = ((a^b)&c) | (a&b);

endmodule
```

1.) 4 bit adder using carry generate and propagate logic

In ripple carry adders, for each adder block, the two bits that are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block. So, it is not possible to generate the sum and carry of any block until the input carry is known. The ith block waits for the (i-1)th block to produce its carry. So there will be a considerable time delay which is carry propagation delay.

In this we initially calculates all the carry inputs and then pass it to the full adder .

$$C_1 = G_0 + P_0C_{in}$$
$$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_{in}$$
$$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{in}$$
$$C_4 = G_3 + P_3C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{in}$$

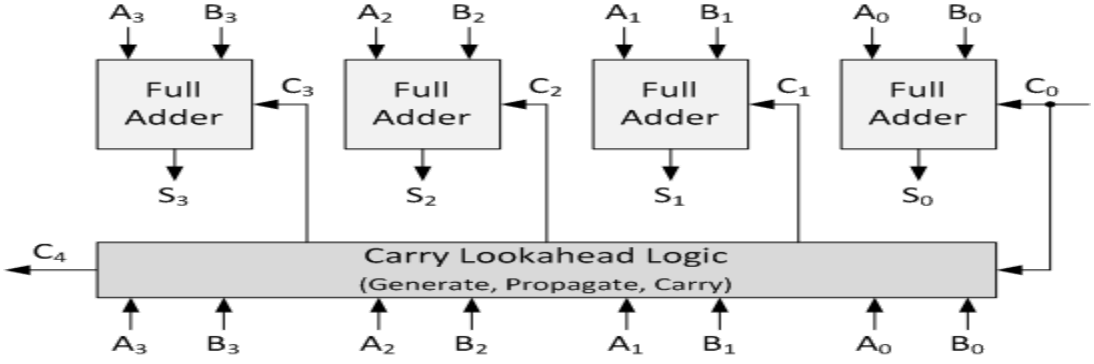
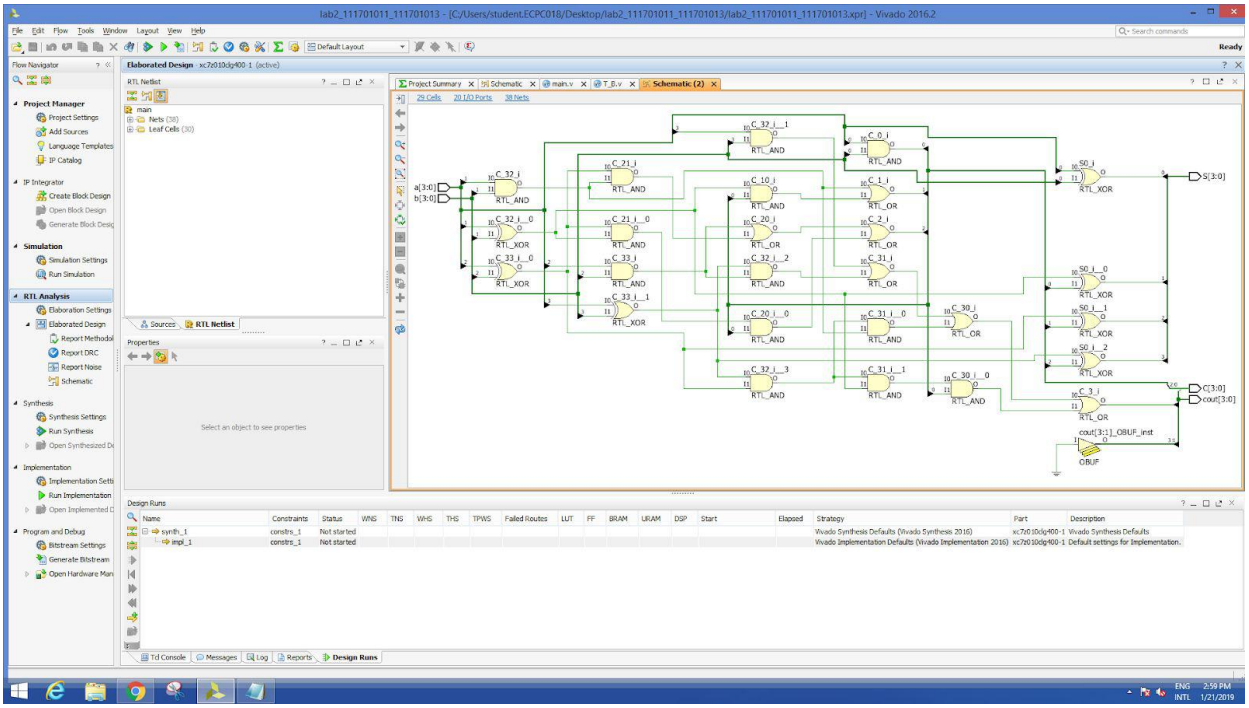


Figure 2: 4 bit adder using carry generate and propagate logic

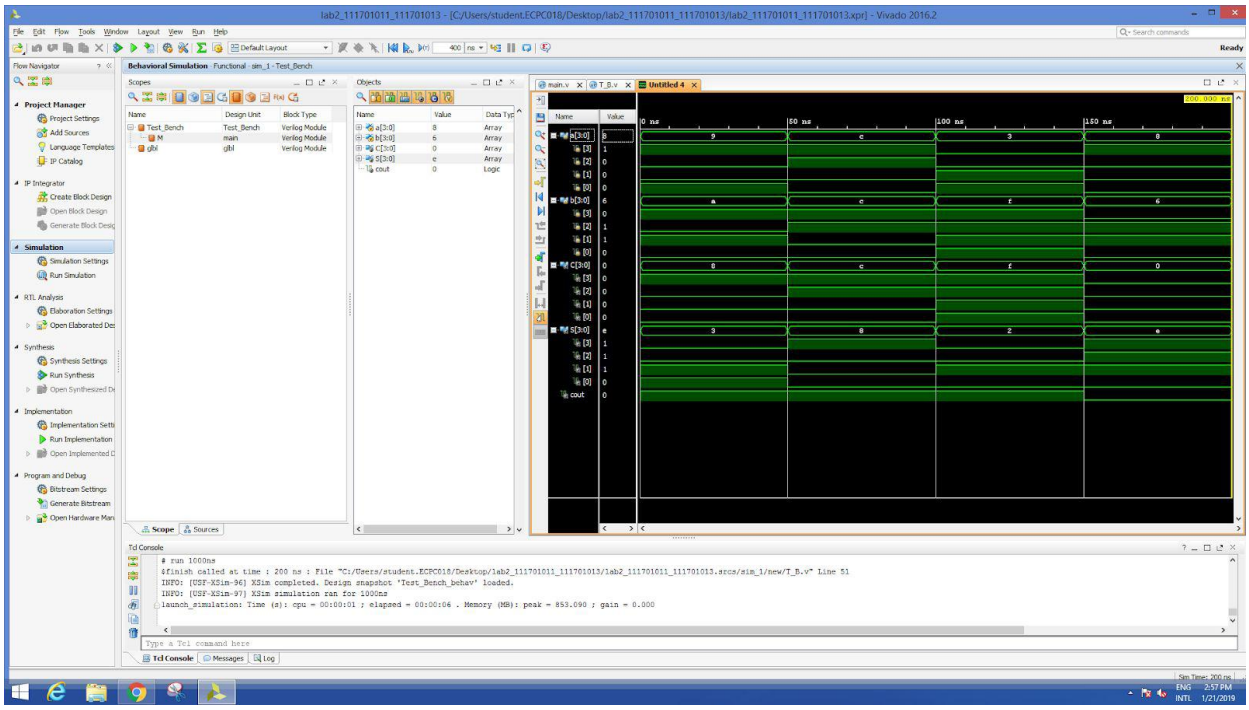
Code for 4 bit adder using carry generate and propagate logic

```
module main(input [3:0] a,b, output [3:0] C, S, cout);

    assign C[0] = a[0]&b[0];
    assign C[1] = (a[1]&b[1]) | ((a[1]^b[1])&(a[0]&b[0]));
    assign C[2] = (a[2]&b[2]) | ((a[2]^b[2])&(a[1]&b[1])) | ((a[2]^b[2])&(a[1]^b[1])&(a[0]&b[0]));
    assign C[3] = (a[3]&b[3]) | ((a[3]^b[3])&(a[2]&b[2])) | ((a[3]^b[3])&(a[2]^b[2])&(a[1]&b[1])) |
((a[3]^b[3])&(a[2]^b[2])&(a[1]^b[1])&(a[0]&b[0]));

    assign S[0] = a[0]^b[0];
    assign S[1] = a[1]^b[1]^C[0];
    assign S[2] = a[2]^b[2]^C[1];
    assign S[3] = a[3]^b[3]^C[2];
    assign cout = C[3];

endmodule
```

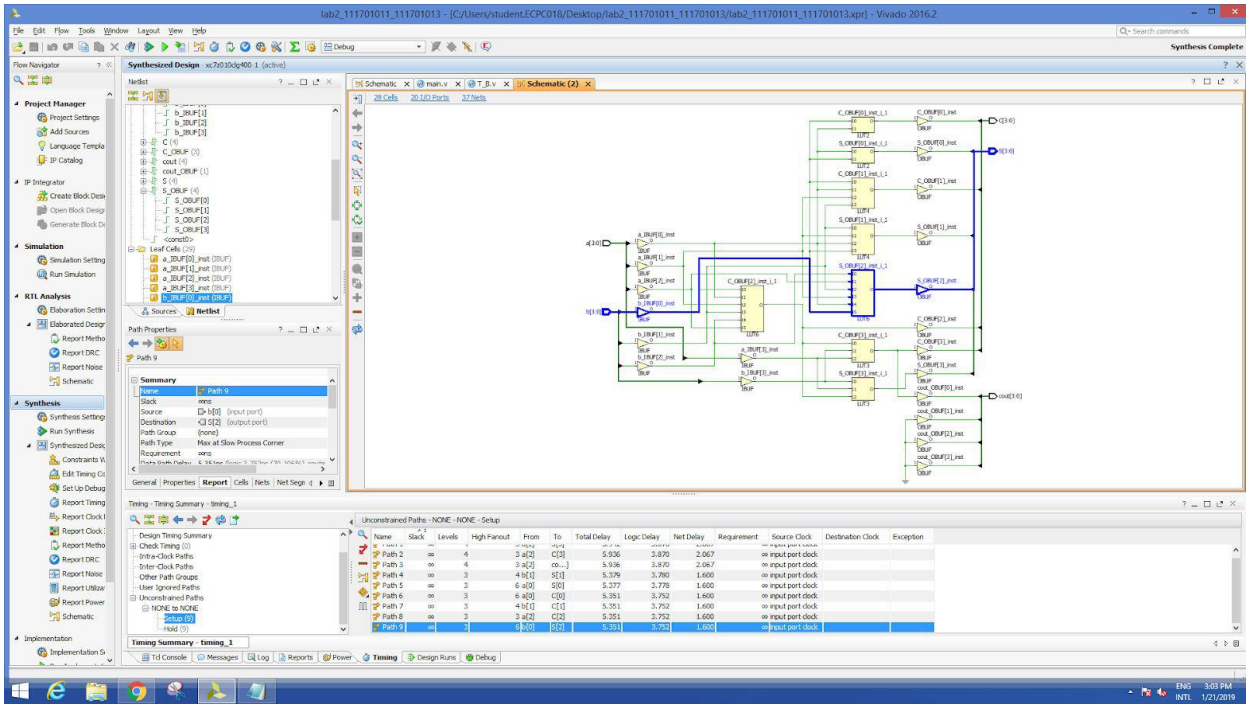
Elaborated Design



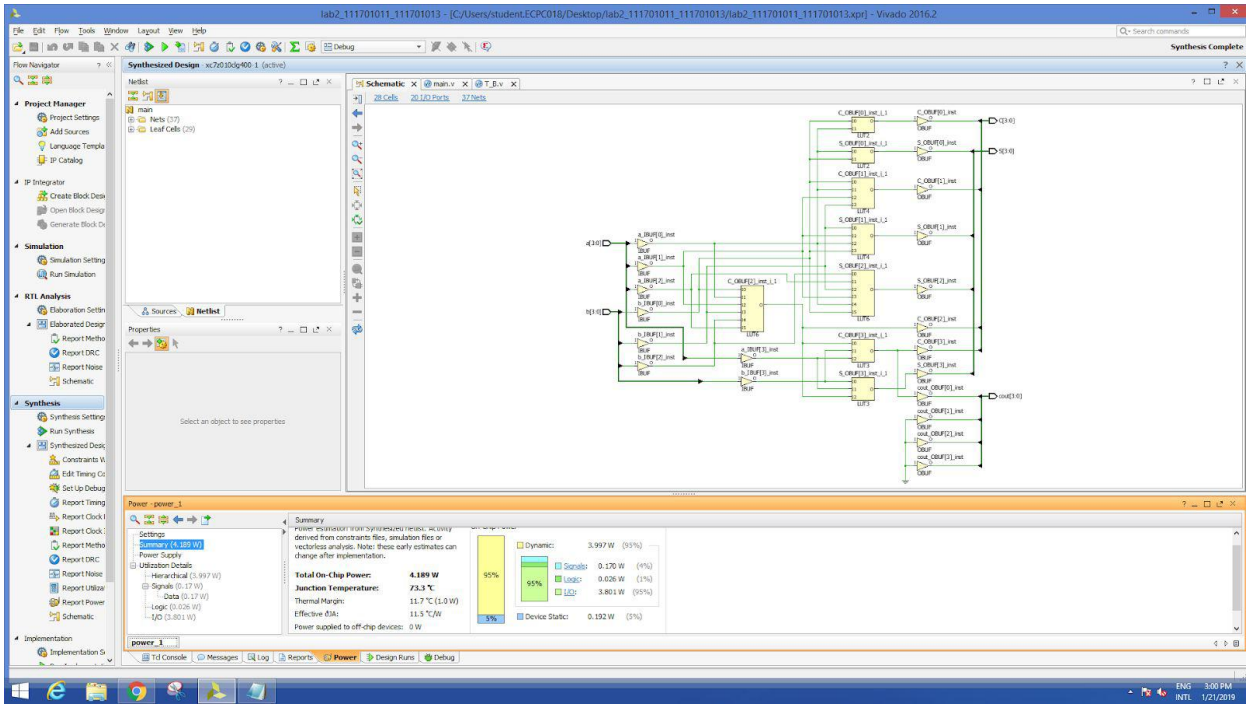
Simulation Results



Timing Results



Schematic and On chip power Results



2.) 4 bit subtractor using carry generate and propagate logic

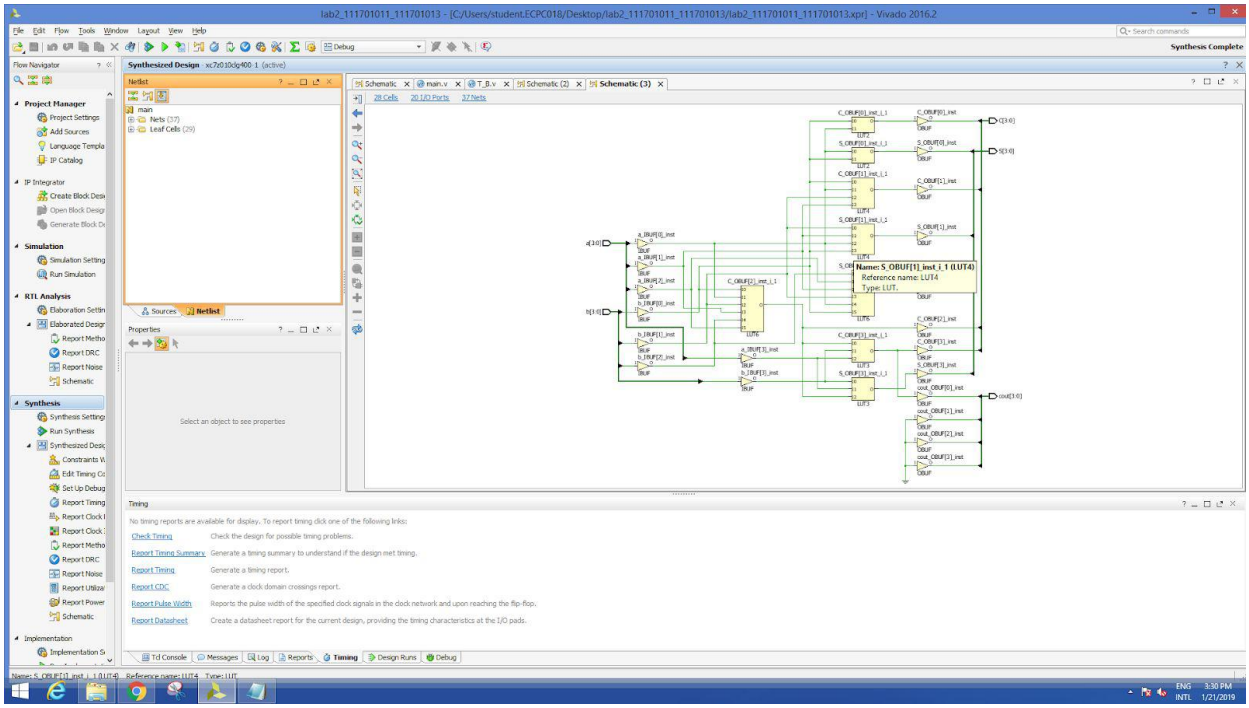
In ripple carry adders, for each adder block, the two bits that are to be added are available instantly. However, each adder block waits for the carry to arrive from its previous block. So, it is not possible to generate the sum and carry of any block until the input carry is known. The ith block waits for the (i-1)th block to produce its carry. So there will be a considerable time delay which is carry propagation delay.
In this we initially calculates all the carry inputs and then pass it to the subtractor.

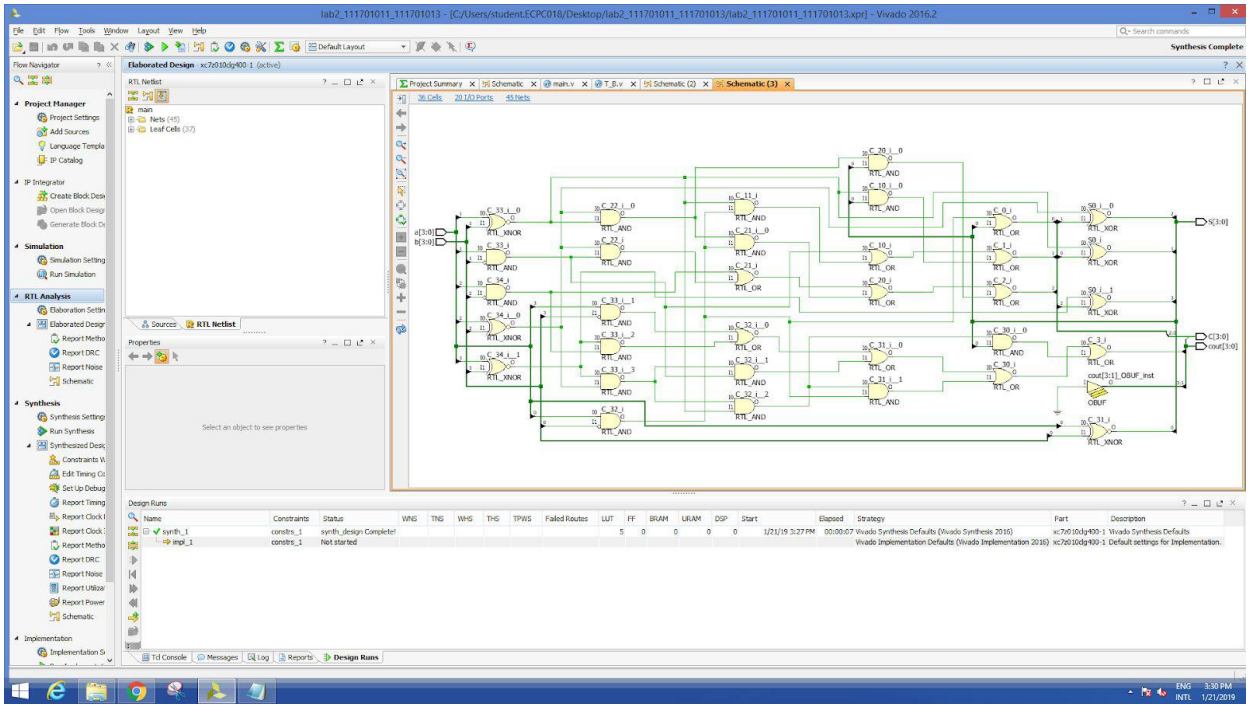
A	B	C	C +1	Condition
0	0	0	0	
0	0	1	0	No Carry
0	1	0	0	Generate
0	1	1	1	
1	0	0	0	No Carry
1	0	1	1	Propogate
1	1	0	1	Carry
1	1	1	1	Generate

Code for second question

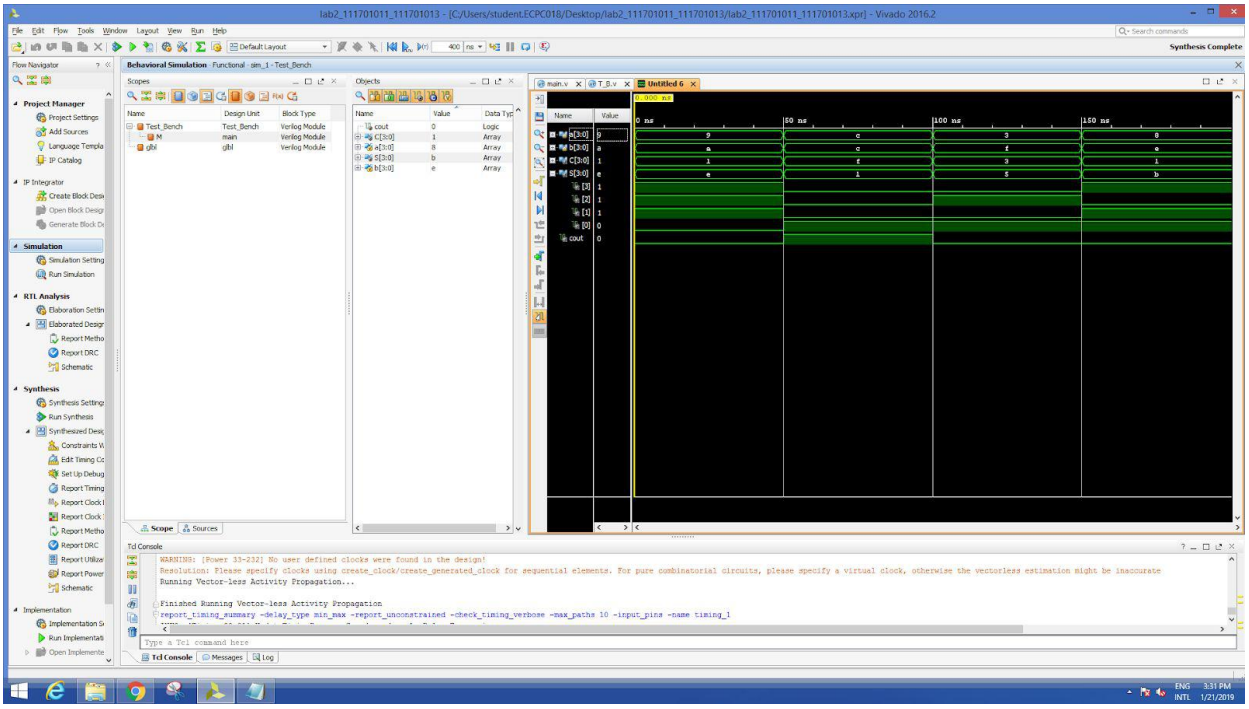
```
module main(input [3:0] a,b, output [3:0] C, S, cout);

   assign C[0] = (a[0]&~b[0]) | (a[0]^~b[0]);
   assign C[1] = (a[1]&~b[1]) | ((a[1]^~b[1])&(a[0]&~b[0])) | ((a[1]^~b[1])&(a[0]^~b[0]));
   assign C[2] = (a[2]&~b[2]) | ((a[2]^~b[2])&(a[1]&~b[1])) |
((a[2]^~b[2])&(a[1]^~b[1])&(a[0]&~b[0])) | ((a[2]^~b[2])&(a[1]^~b[1])&(a[0]^~b[0]));
   assign C[3] = (a[3]&~b[3]) | ((a[3]^~b[3])&(a[2]&~b[2])) |
((a[3]^~b[3])&(a[2]^~b[2])&(a[1]&~b[1])) |
((a[3]^~b[3])&(a[2]^~b[2])&(a[1]^~b[1])&(a[0]&~b[0])) |
((a[3]^~b[3])&(a[2]^~b[2])&(a[1]^~b[1])&(a[0]^~b[0]));

   assign S[0] = a[0]^~b[0];
   assign S[1] = a[1]^~b[1]^C[0];
   assign S[2] = a[2]^~b[2]^C[1];
   assign S[3] = a[3]^~b[3]^C[2];
   assign cout = C[3];

endmodule
```

Elaborated design

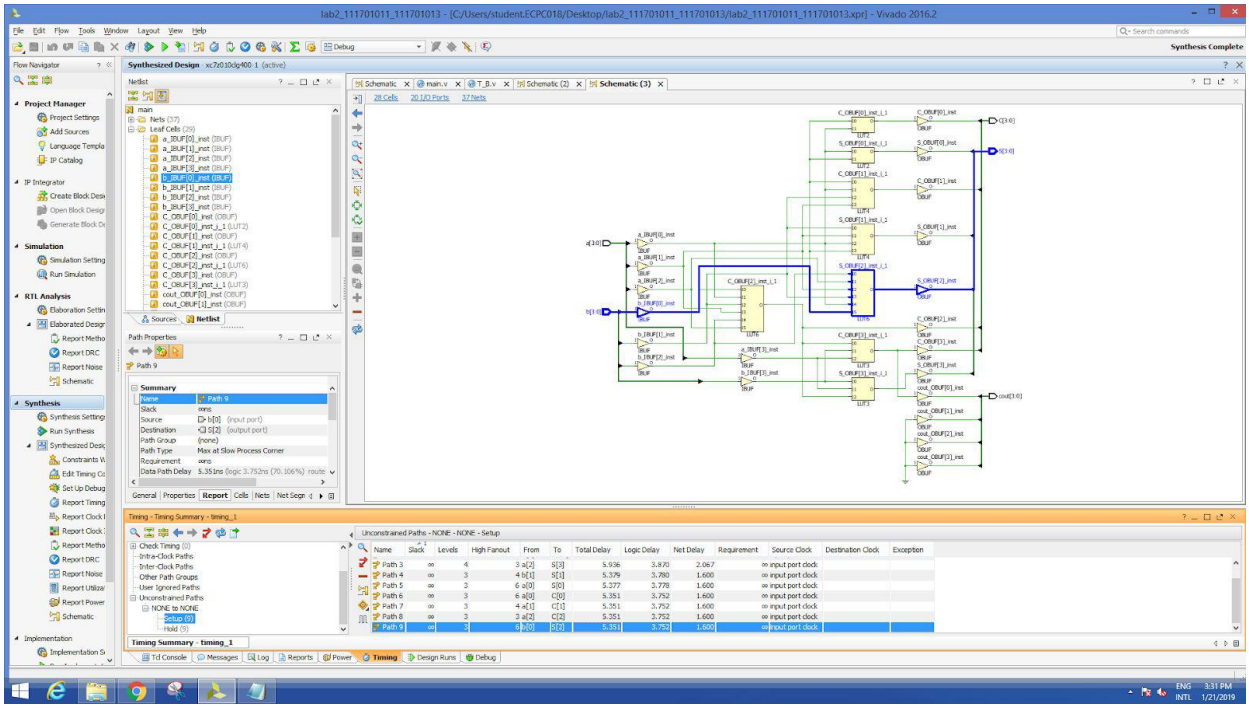




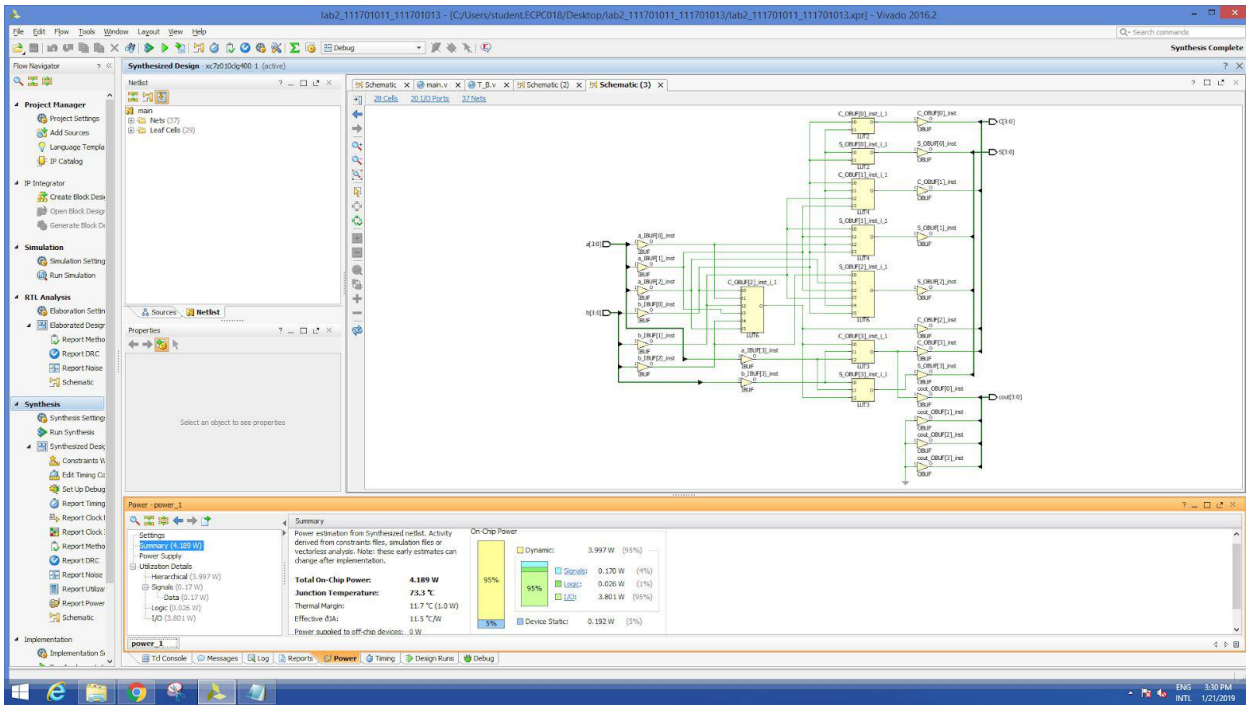
Simulation Results



Timing Results

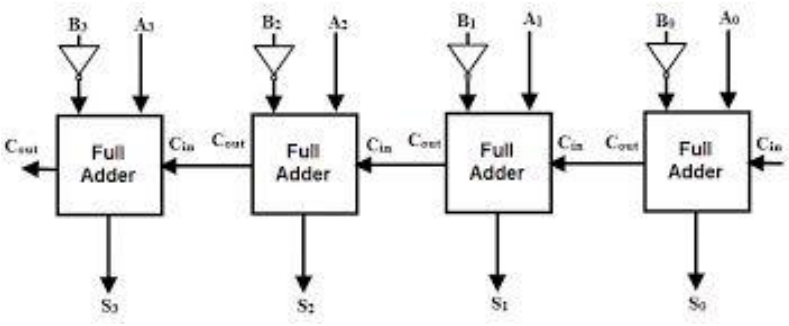


Schematic and Power Results



3.) 4 bit subtractor using ripple carry adder

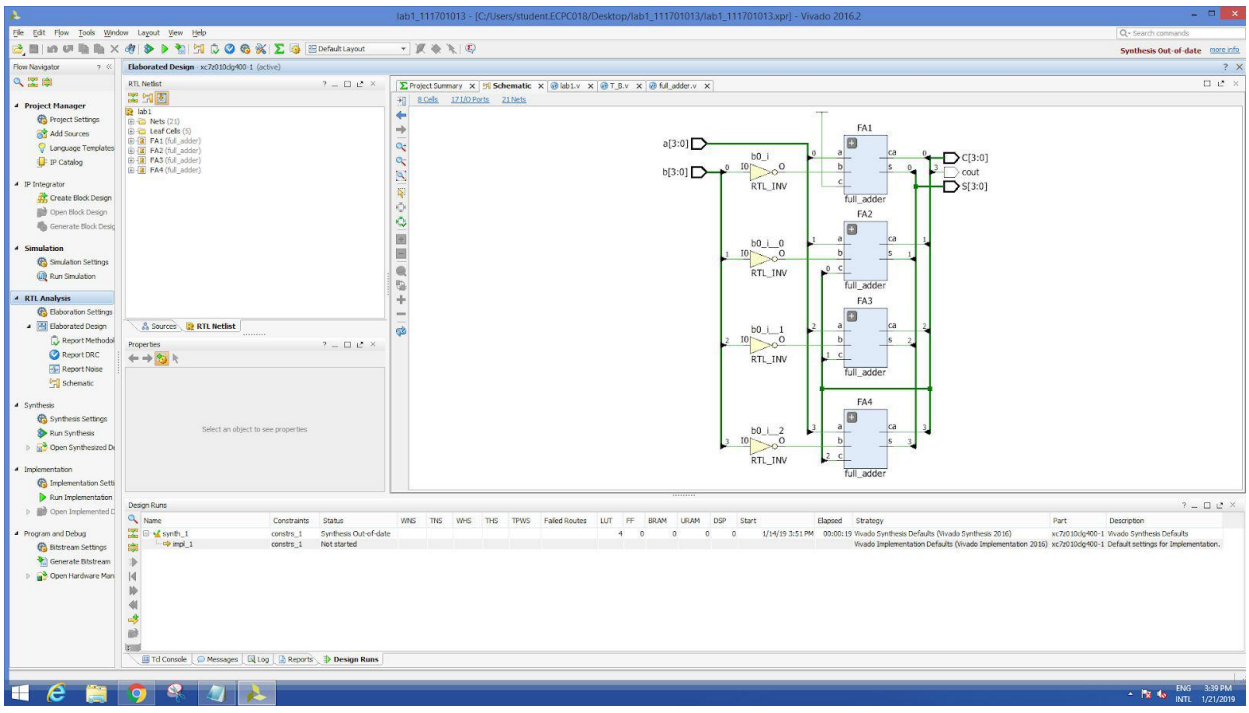
We have used four full adder to subtract the two given four bit numbers by using 2's complement of the second number and adding it to the first number. In the first full adder to add the initial digit of the two numbers the third digit given as carry is 1. After that it provided us the sum and the carry, the sum is stored in the register and the carry is passed to the next full adder. Similarly next three full adders are used to add the rest three bits of the two numbers. The last full adder gives us the final carry which may be 0 or 1 which is also the output of the sum of numbers including the sums from all the full adders. In this we need to have four full adders so that we can perform the addition. Each bit from both numbers was passed to different full adder and the output was generated in one go.

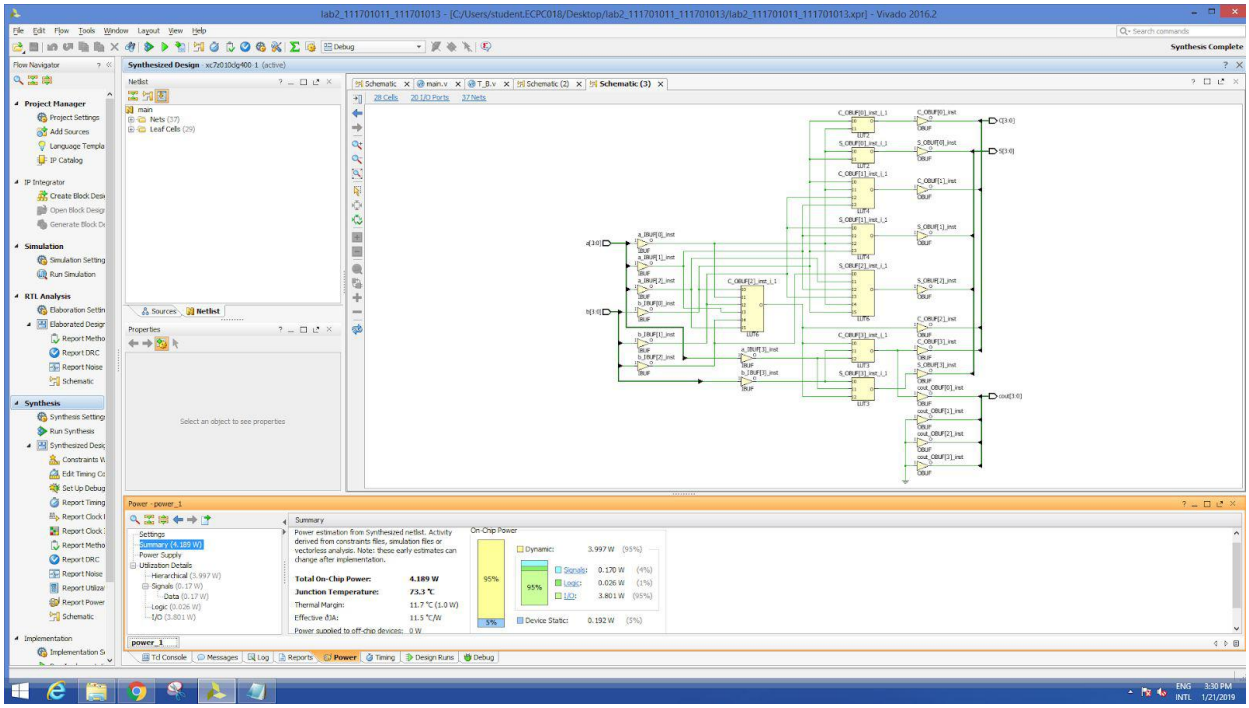


Code for 4 bit ripple carry subtractor

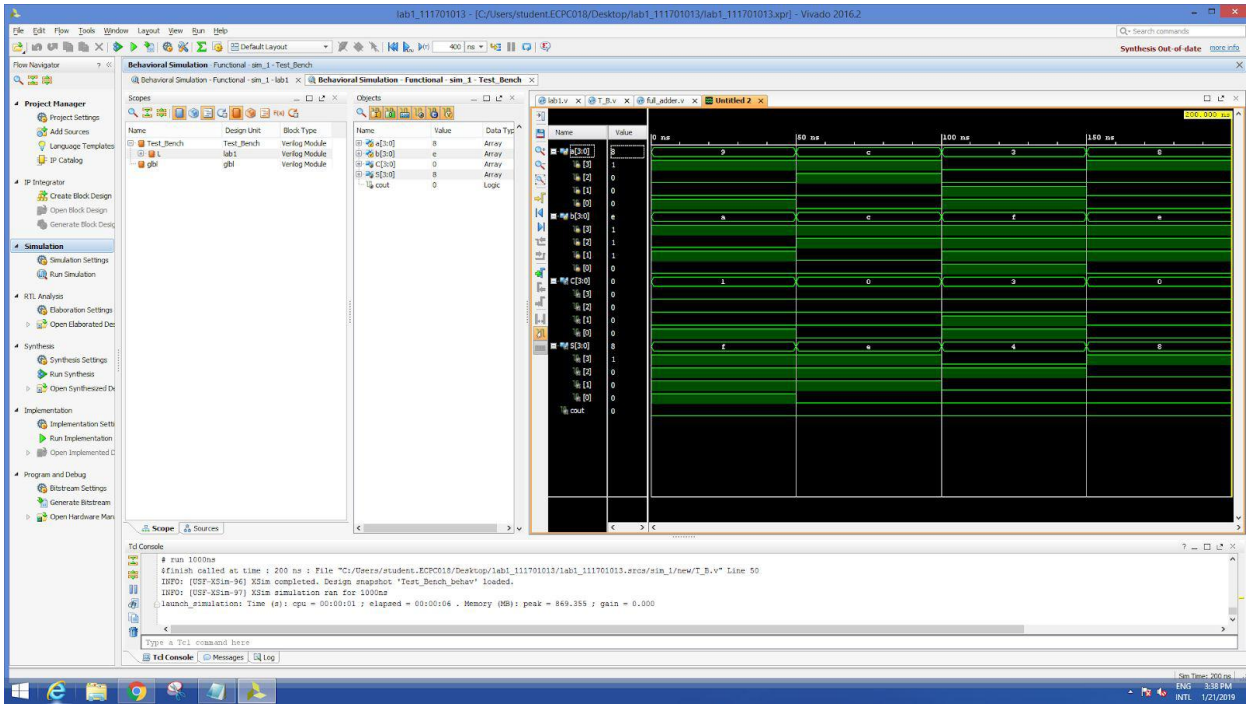
```
module subtractor(input [3:0] a, input [3:0] b, output [3:0] S, output [3:0] C,output cout);


    full_adder FA1 (a[0],~b[0],1,S[0],C[0]);
    full_adder FA2 (a[1],~b[1],C[0],S[1],C[1]);
    full_adder FA3 (a[2],~b[2],C[1],S[2],C[2]);
    full_adder FA4 (a[3],~b[3],C[2],S[3],C[3]);
    assign cout=C[3];

endmodule
```

Elaborated design

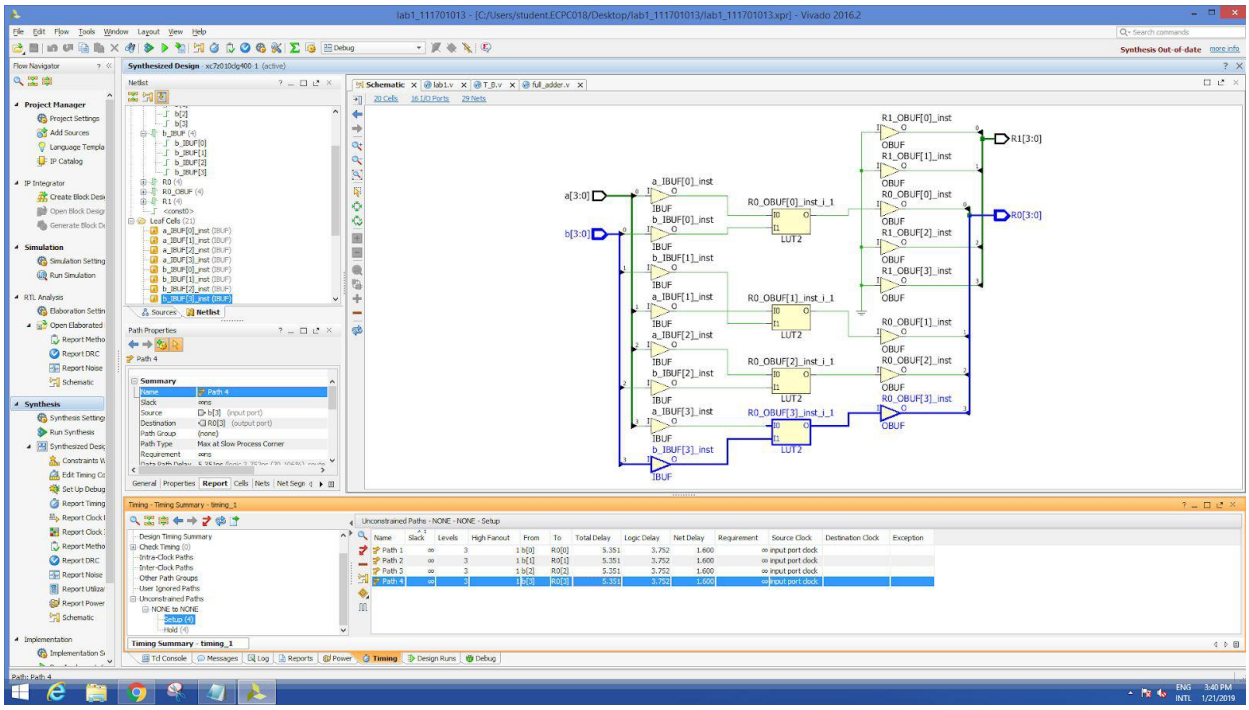




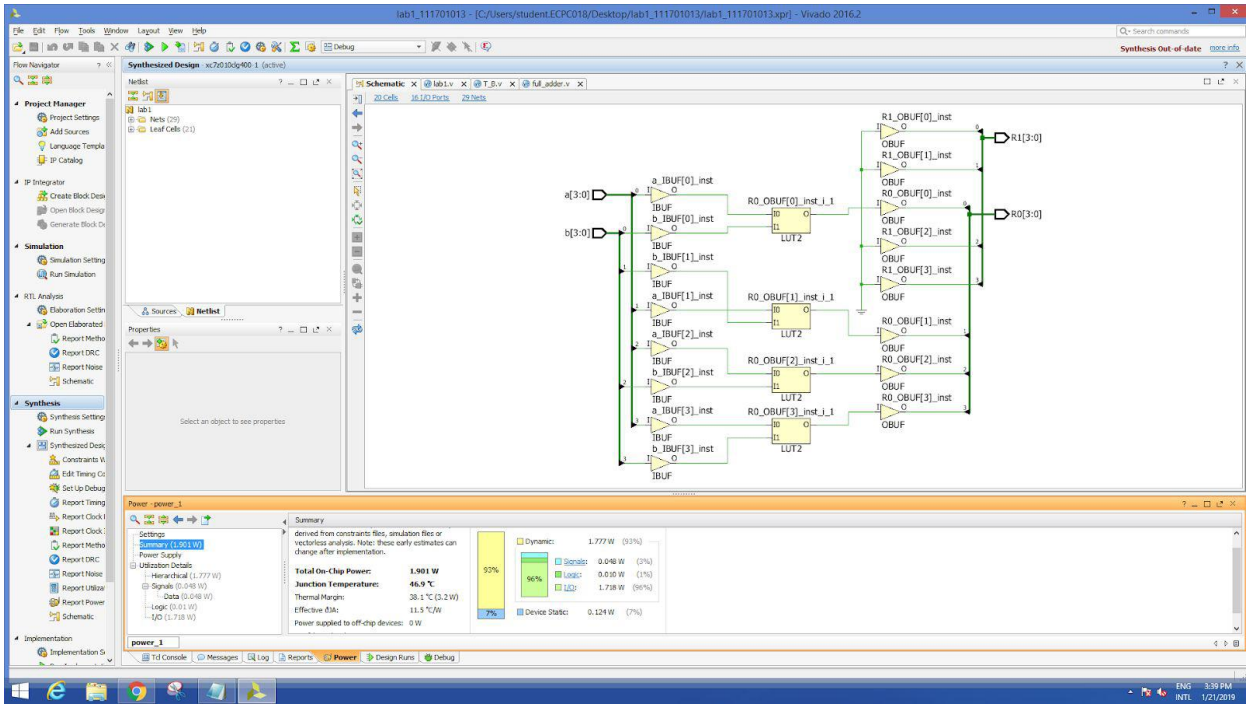
Simulation Results



Timing Results



Schematic and Power Results



Observations:

An adder that uses carry generate and propagate logic to add A and B:

Dynamic Power used : 3.997W

Static Power used : 0.192W

Total On Chip Power used : 4.189W

Static Power is 5% of the total power while dynamic power is 95%.

In dynamic power,

95% is used for I/O operations = 3.801W

1% is used in logic = 0.026W

4% is used in signals = 0.170W

Junction Temperature was 73.3°C

A subtractor that uses carry generate and propagate logic to subtract B from A:

Dynamic Power used : 3.997W

Static Power used : 0.192W

Total On Chip Power used : 4.189W

Static Power is 5% of the total power while dynamic power is 95%.

In dynamic power,

95% is used for I/O operations = 3.801W

1% is used in logic = 0.026W

4% is used in signals = 0.170W

Junction Temperature was 73.3°C

A subtractor that using ripple carry adder in Lab 1 to subtract B from A:

Dynamic Power used : 1.777W

Static Power used : 0.124W

Total On Chip Power used : 1.901W

Static Power is 7% of the total power while dynamic power is 93%.

In dynamic power,

96% is used for I/O operations = 1.718W

1% is used in logic = 0.010W

3% is used in signals = 0.048W

Junction Temperature was 46.9°C

1.) *In the first question (Adder using Generate and Propagate Logic), power consumption was more as compared to Adder circuits developed in previous labs .*
2.) *In the second one, the power consumption was higher than the power consumption in the third one.*
3.) *Junction Temperature for the first question is also higher than the Junction temperatures in adder circuits in the previous lab questions.*
4.) *Junction Temperature for the second question is higher than the Junction Temperature in the third question.*
5.) *In the first question, dynamic power percentage is higher than the adder circuits in the previous lab.*
6.) *In second question, dynamic power percentage is higher than the dynamic power percentage in the third one.*
7.) *Most of the dynamic power is consumed by the I/O.*
8.) *Rest of the dynamic power is used in logic and signals which is very less than input output dynamic power.*
9.) *More buffers and logic was there in the second logic as compared to the third one.*

Results:

1.) *Power consumption is less when there are less number of components are used at the same time.*
2.) *Power consumption reduces by a great factor when the number of components working parallely is decreased.*
3.) *But the time delay increases as the parallel logic is transformed into the pipeline logic.*

4.) But the static power remains constant in both the logic.

5.) Time delay is more in second program than in the third program.

6.) Time delay is more in first program than in the last lab adder programs.

Conclusions:

1.) More power is consumed in the parallel programming as compared to sequential (pipeline) programming.

2.) Static power consumed remains same as it is consumed when there is no activity on the circuit so it is independent on the circuit.

3.) Processing delay is more in case of pipeline fashion while it is less in parallel fashion.