lab report 2

Andrew Nady
900184042

Question 2:
Technical summary:

Steps:

- In the zero experiment,
  - we copied the files of the UART to Vivado.
  - Then downloaded tera term.
  - And then we tested the UART using the tera term by clicking the assigned keys" a, …".
- In the first experiment,
  - we added the full adder module
  - then a ripple carry module to call 4 full adders to make 4-bit full adder.
  - We added the seven-segment display of the previous lab and it constrain file.
  - Then we make a top module to connect the UART, ripple carry and the seven-segment display.
- In the second experiment,
  - we have 8 bits output from the UART, so we make the number in the BCD and the seven-segment display module 8 bits.
  - Second, we make a variable called comp which is 2's Complement =~num+1.
  - Then we made a condition if the most significant bit is 1 then pass the comp to the BCD. Otherwise, pass the num as normal.
  - In order to display the negative sign, we make instead of the thousand, an if statement that sees if the most significant bit is 1, then assign anode to 4'n0111 to activate the first digit, otherwise we gave it 4 1's. then we gave the LED_BCD a value 10. Then we added a case below which implements the '-', when the lED_BCD is activated and equals to 10.

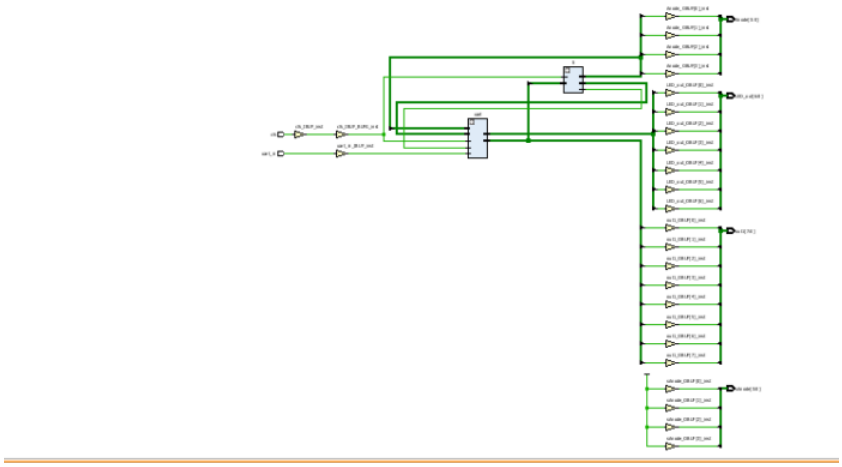
- In the third experiment,
  - We take UART the same as the other experiments
  - We take seven-segment display and only modified the bits of the input num and made it 4 bits.
  - We take the BCD and only modified the bits of the input num and made it 4 bits
  - Then made a top module to connect everything together.

Components:

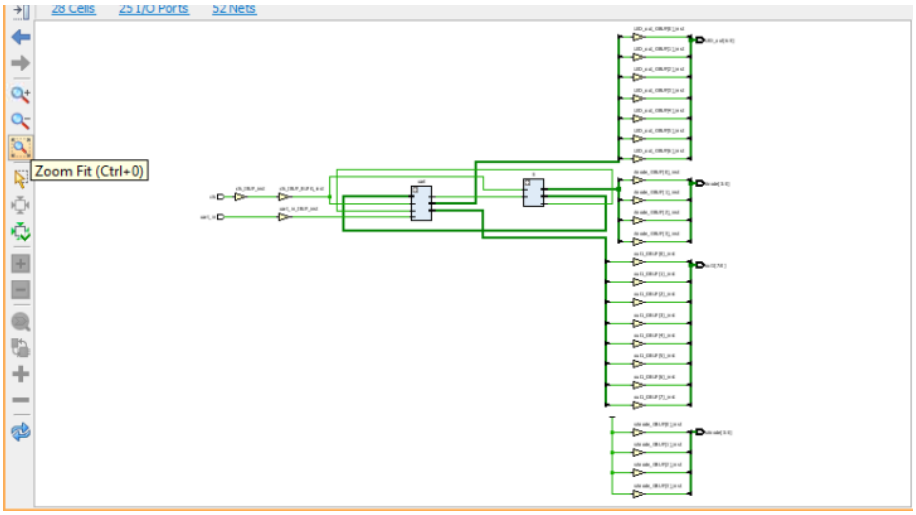
1. Nexys 100T a7
2. Vivado
3. Tera term

Schematic design:

Experiment 1:



Experiment 3:

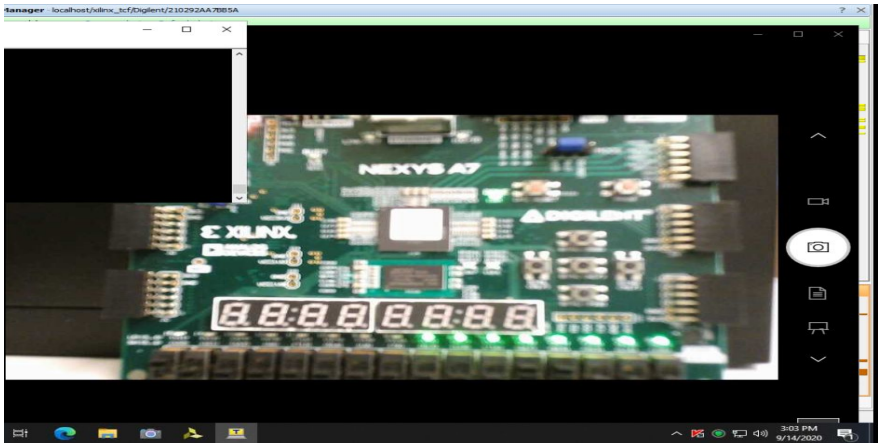


Code functionalities:

- In the zero experiment, when clicking on specific keys on keyboard it makes specific led on.
- In the first experiment, adding two 4 bits with each other and then display the result in the seven-segment display
- In the second experiment, displaying any number on the seven-segment display whether it was negative or positive. If it is negative the first digit from the left will display the sign '-'
- In the third experiment, adding any two numbers of four bits whether they are negative or positive. (eg -6+2=-4). In this example, it will display – 006 on the seven segment display

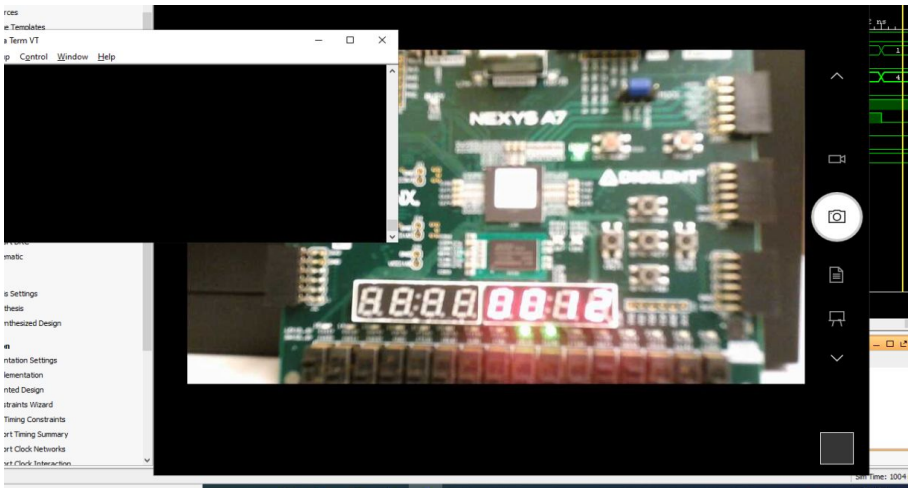
Question 3:
Results:

Experiment 0:



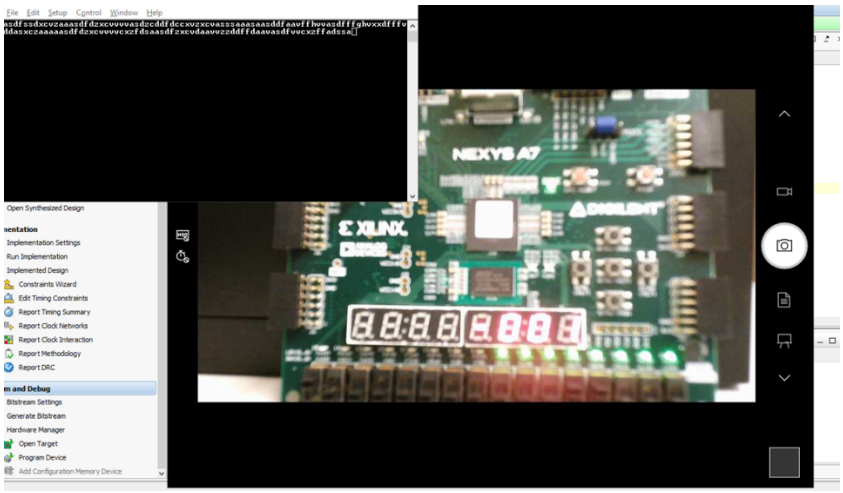
Experiment 1:

12+0 =12



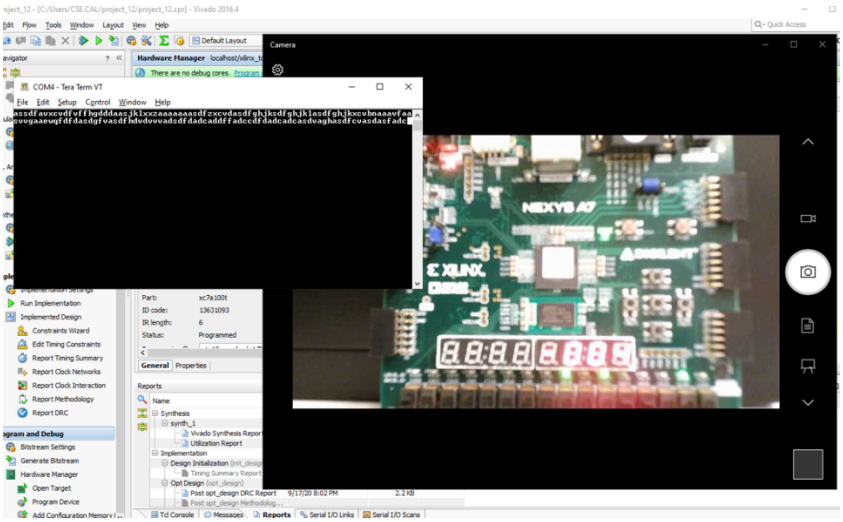
Experiment 2:

Displaying ‘-1’



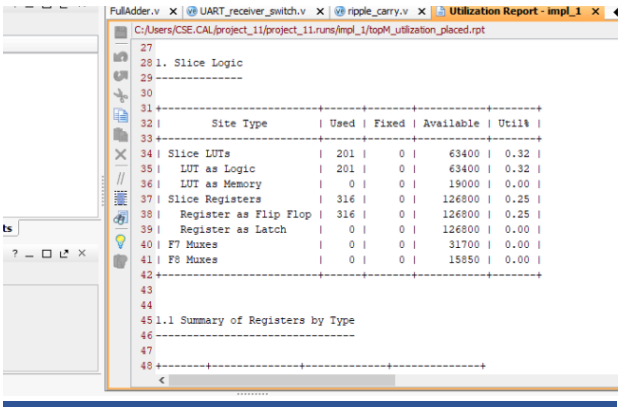
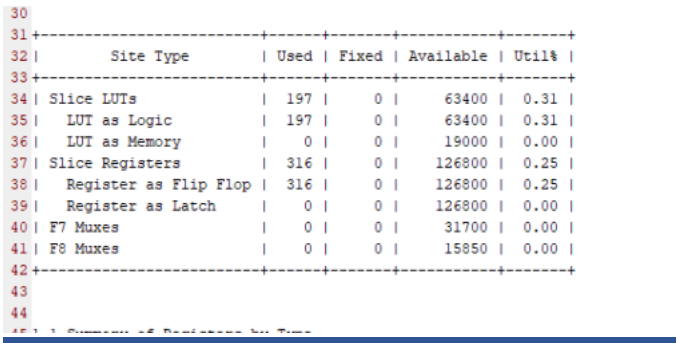
Experiment 3:

1010 + 0010 = -6 + 2 =-4



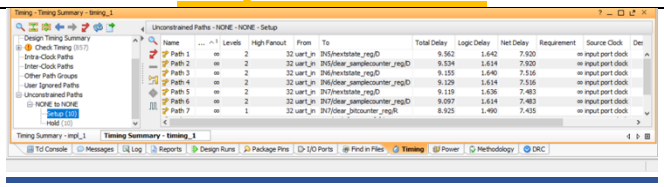
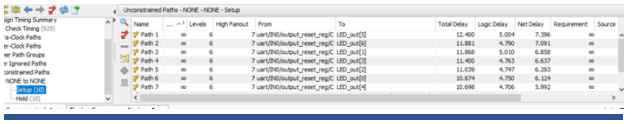
Problem (4)
Comparison:

Utilization:

	Experiment 1	Experiment 3
SCREENSHOTS		

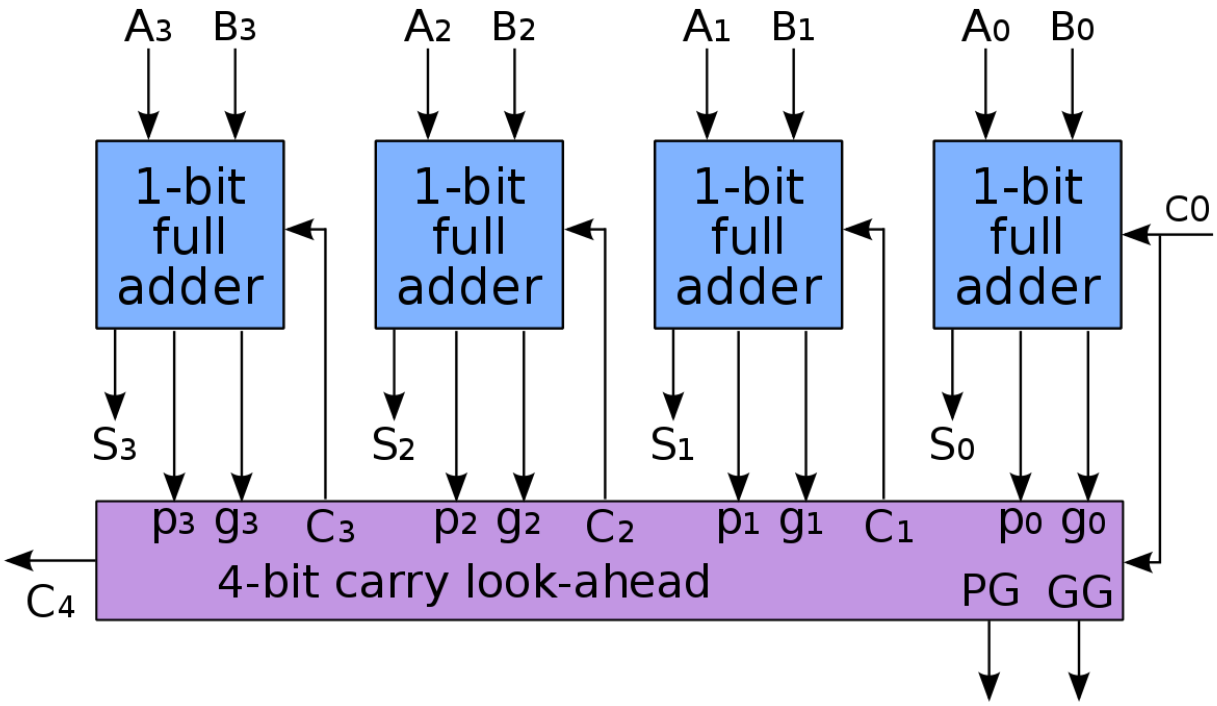
Site Type	Used	Fixed	Available	Util%
Slice LUTs	197	0	63400	0.31
LUT as Logic	197	0	63400	0.31
LUT as Memory	0	0	19000	0.00
Slice Registers	316	0	126800	0.25
Register as Flip Flop	316	0	126800	0.25
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Delay:

	Experiment 1	Experiment 3
SCREENSHOTS		

- Utilization is less than in experiment 3, because we decreased the input num from 13 bit to 4 bits only which decrease the hardware a little bit (I think so).

Problem (5)



what should be done is replacing the full adder with the lock ahead adder.

4-bit carry lookahead adder

```
// file: CLA.v
// author: @andrewhany

`timescale 1ns / 1ps

module CLA( input [3:0] x , input [3:0] y , output [4:0] res );
wire [4:0] C;
wire [3:0] G,P,sum;

FullAdder adder1 (x[0],y[0],C[0],sum[0]);
FullAdder adder2 (x[1],y[1],C[1],sum[1]);
FullAdder adder3 (x[2],y[2],C[2],sum[2]);
FullAdder adder4 (x[3],y[3],C[3],sum[3]);



assign G[0]= x[0] & y[0];
assign G[1]= x[1] & y[1];
assign G[2]= x[2] & y[2];
```

```
assign G[3]= x[3] & y[3];


assign P[0]= x[0] | y[0];
assign P[1]= x[1] | y[1];
assign P[2]= x[2] | y[2];
assign P[3]= x[3] | y[3];


assign C[0]=1'b0;
assign C[1]= G[0] | (P[0] & C[0]);
assign C[2]= G[1] | (P[1] & C[1]);
assign C[3]= G[2] | (P[2] & C[2]);
assign C[4]= G[3] | (P[3] & C[3]);

assign res ={C[4],sum};
endmodule
```

Testbench to test it:

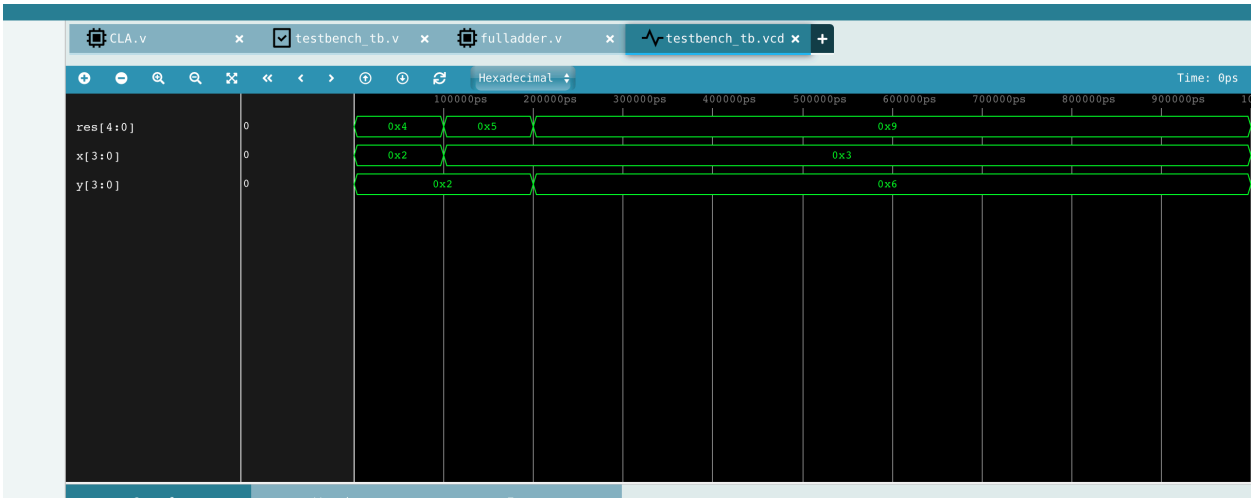
```
// file: testbench_tb.v
// author: @andrewhany
// Testbench testbench_tb

`timescale 1ns/1ns

module testbench_tb;
 reg [3:0] x;
 reg [3:0] y;
wire  [4:0] res;
CLA  cla (  x , y , res );

initial
begin
x[3:0]=4'b0010;
y[3:0]=4'b0010;
#100
x[3:0]=4'b0011;
y[3:0]=4'b0010;
#100
x[3:0]=4'b0011;
y[3:0]=4'b0110;
end
```

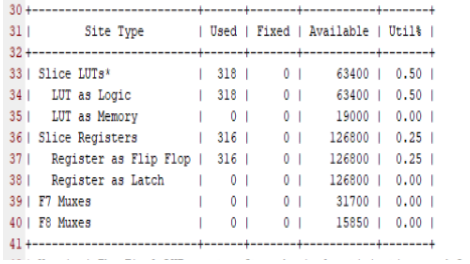
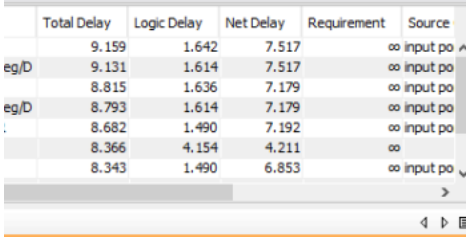
```
endmodule
```

Simulation:

0x2+0x2 =0x4, 0x2+0x3=0x5, 0x6+0x3=0x9

So, it is working properly



utilization	delay
	

There is a folder for Question 5 which contains the whole project.

Problem (6)

- Utilization in the CLA increases because the gates used in CLA is more than used in RCA, this can be noticed when seen the above figure which indicates that the gates and the hardware used in CLA is more than that used in RCA
- The delay is almost the same. However it is a bit faster in CLA than RCA