Name : salma soliman

Id : 900182325

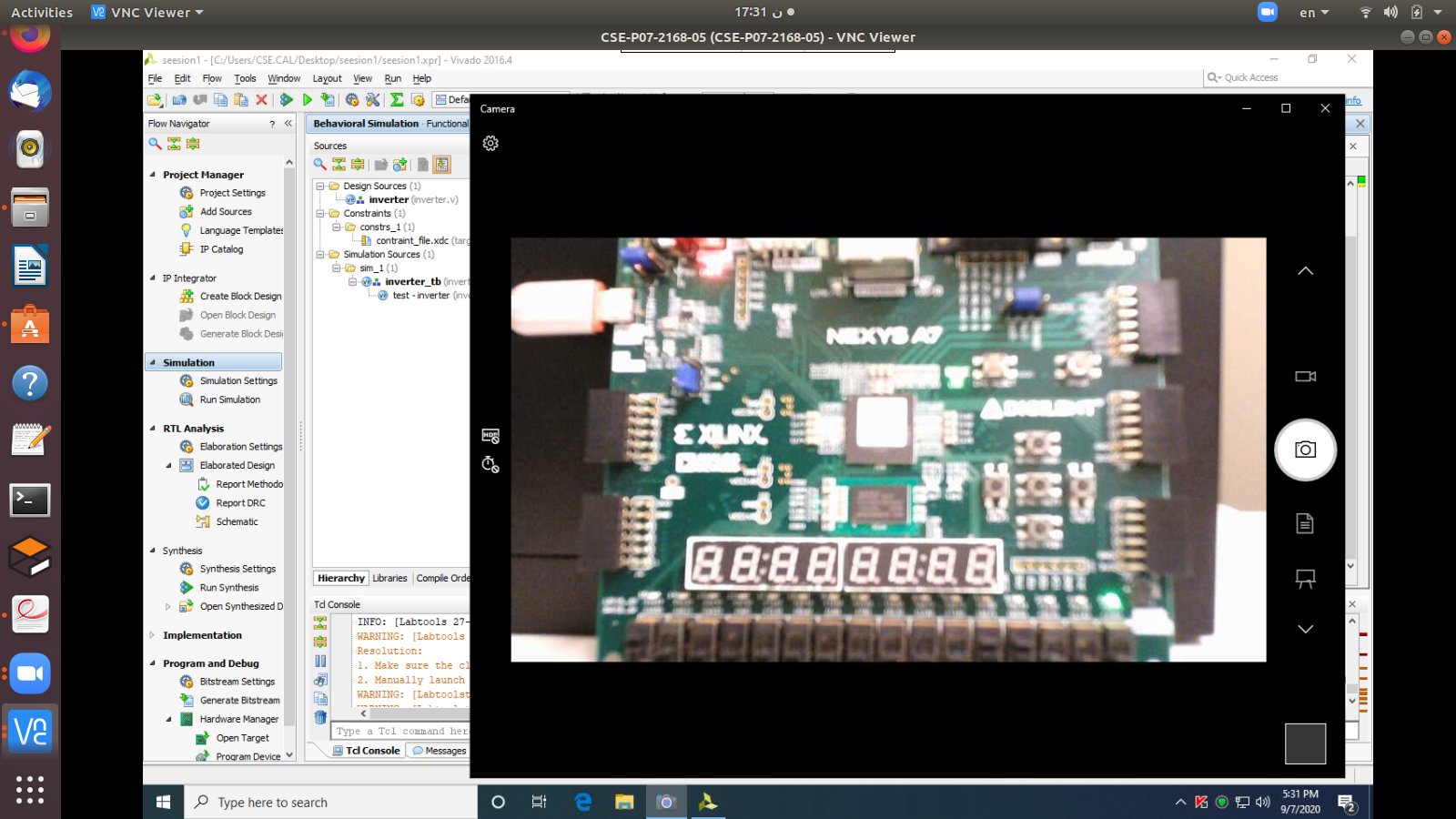
Lab1 Monday 3:30pm

-----------------------------------------------------------------------------------------------------------------------------

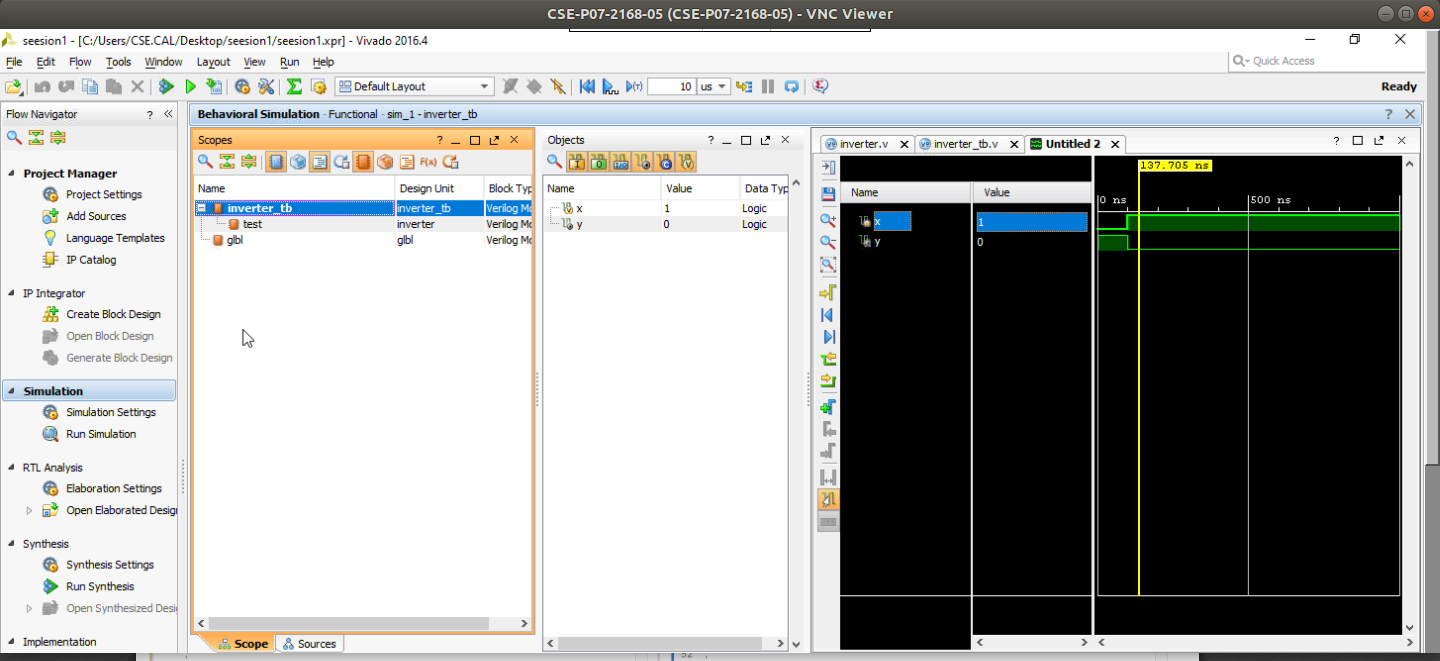
Experiment (1):

Summary : in this experiment, we designed a module called ‘inverter’. It takes a signal and invert it. We saw that on the FPGA when the led switched on and off. The source code and the constraint file are attached to the whole zip file.

Output screenshots:



Simulation :



Experiment (2):

Summary : in this experiment, we were provided with its code in the lab manual.

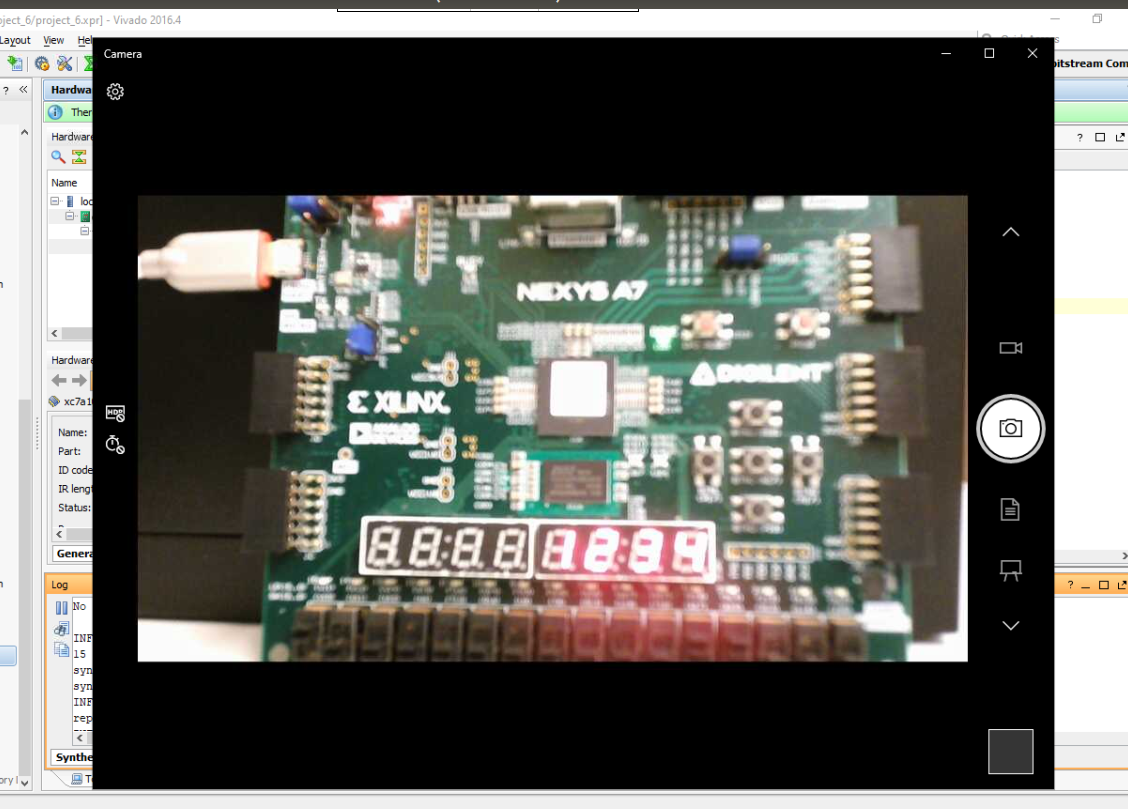
The code developed a four seven segment driver that takes a number of 13 bits and displays it on the 7segment display on the FPGA.

At each +ve edge of the clock we increment the refresh counter. It is counting the frequency of the clock. Then we are using the led activating counter as the most significant bit (like a clock with low frequency)

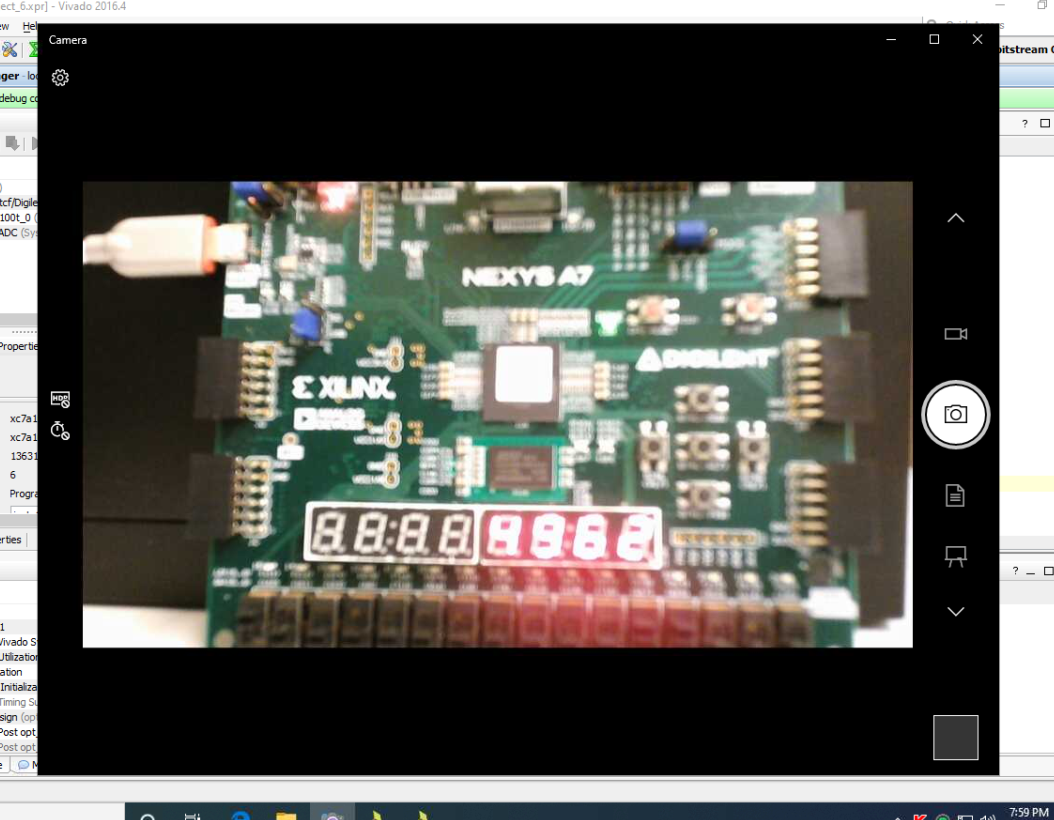
The second part is the conversion part. The code divides the number over 1000 to get the thousands part and then divides it by 100 to get the hundreds part and then divides by 10 to get the tens part and the rest is the ones part by taking the modulus with 10.

The third part is the display part where it makes a switch case over the LED\_OUT to display each digit according to its value.

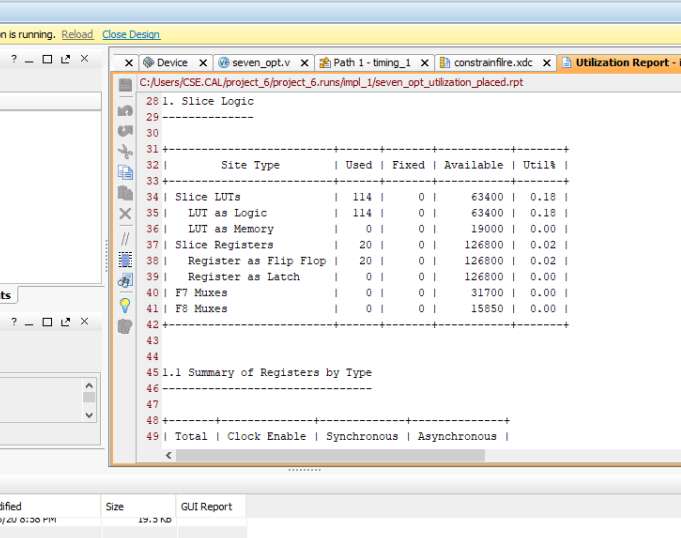
Output screenshot :



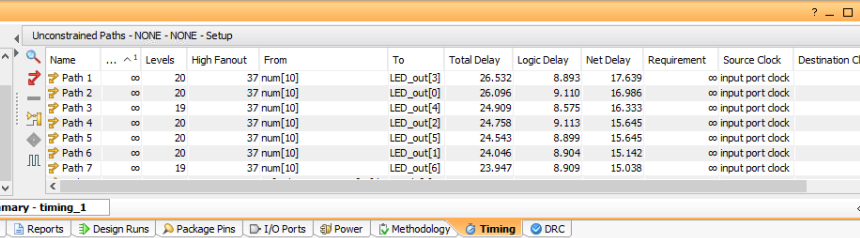
Output screenshot displaying another number:



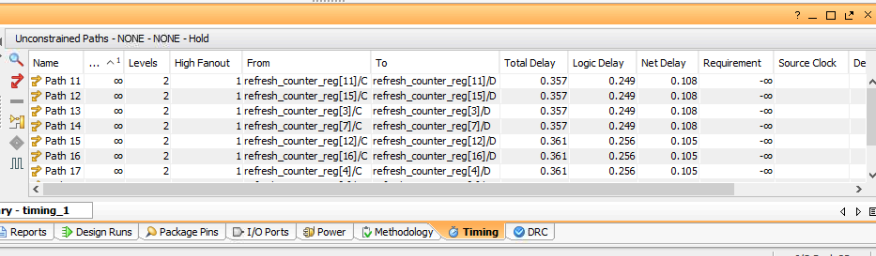
Utilization:



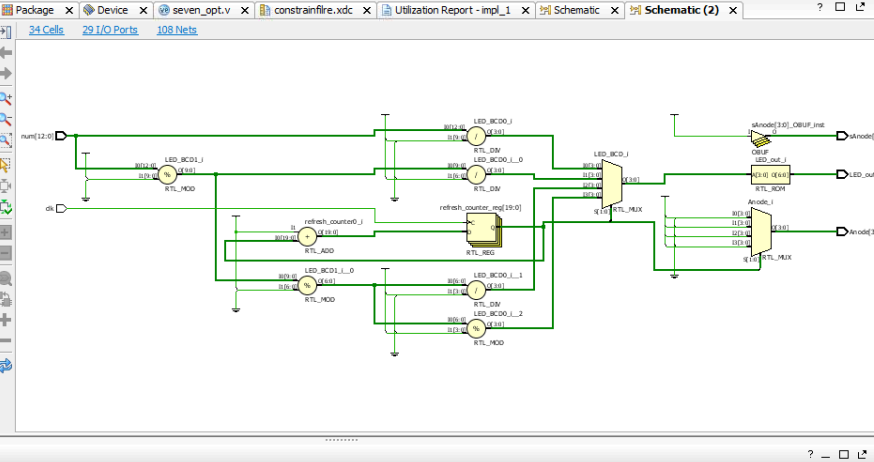
Delay setup:



Hold time:



Schematic:



Experiment (3):

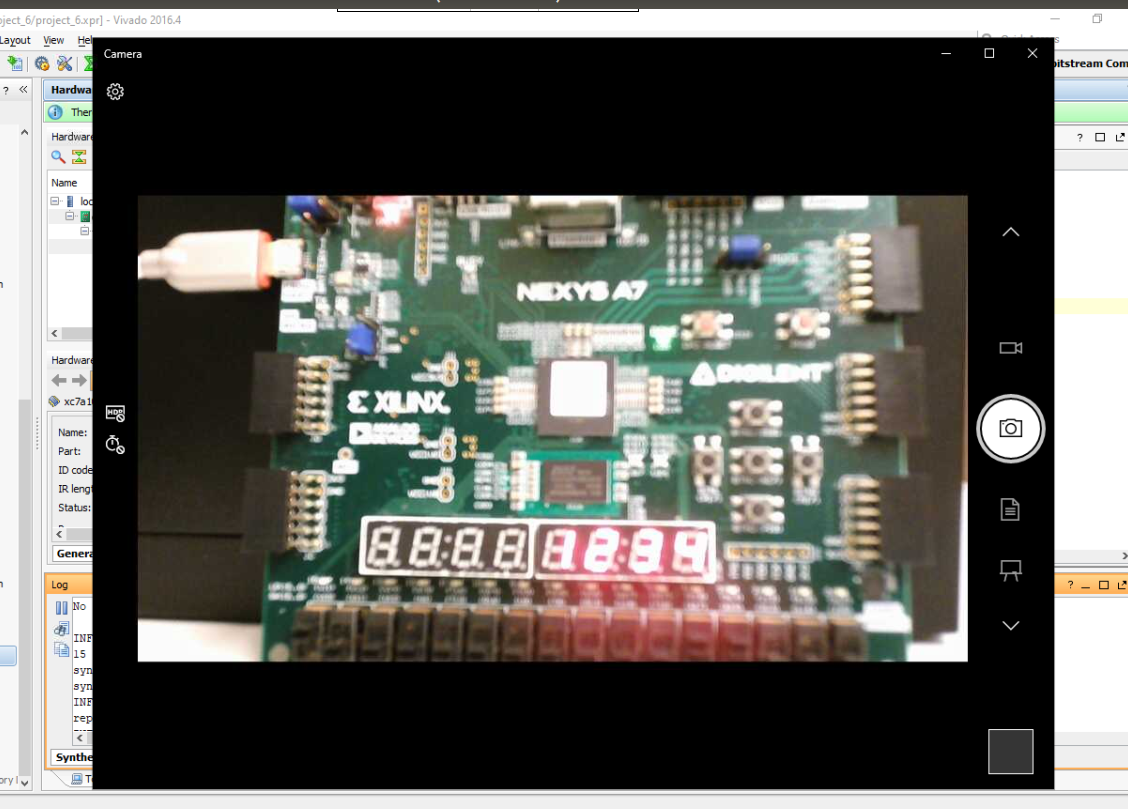
In this experiment, we were provided with its code in the lab manual.

The code developed a four seven segment driver that takes a number of 13 bits and displays it on the 7 segment display on the FPGA.

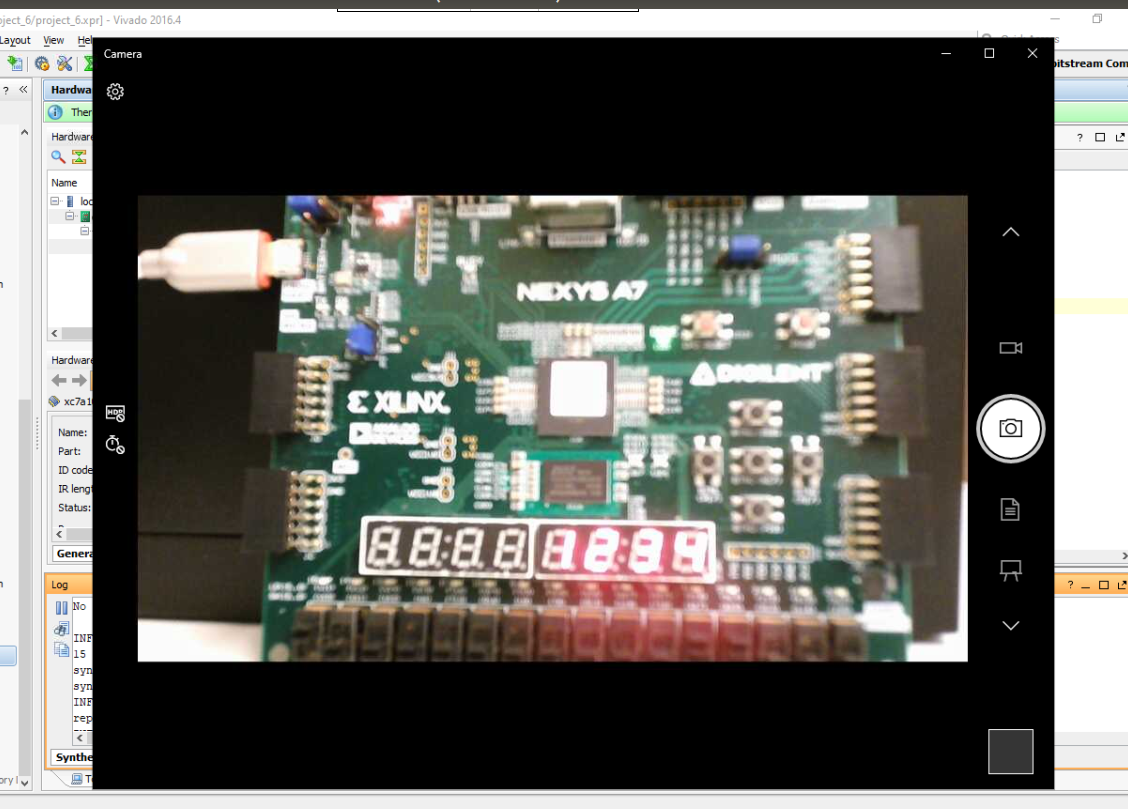
At each +ve edge of the clock, we increment the refresh counter. It is counting the frequency of the clock. Then we are using the led activating counter as the most significant bit (like a clock with low frequency)

The second part is the conversion part. This experiment is similar to the previous one but improved a little bit. The improvement is made by using the shift and add algorithm instead of keeping dividing the number over 1000 or 100 or 10. It keeps shifting the number till filling the thousands,hundreds,tens and ones parts.

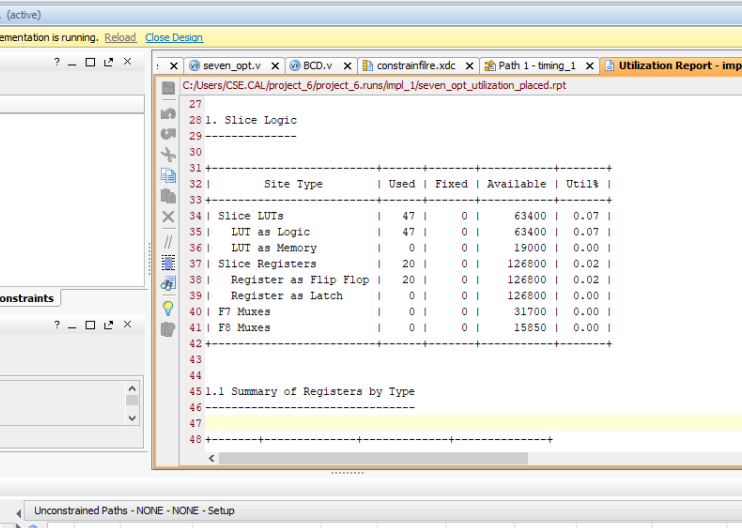
Output screenshot :



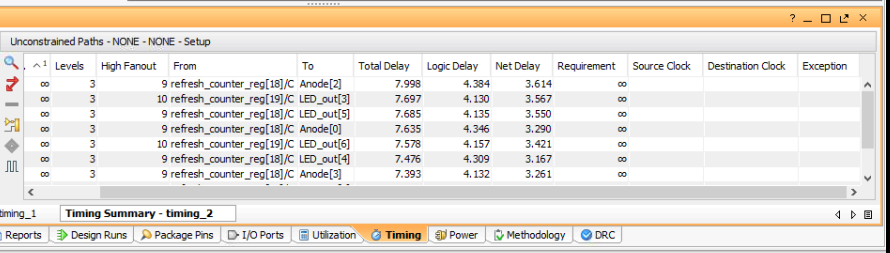
Another screenshot showing another number:



Utilization :



Delay :



Comment on utilization :

In experiment 2, there were more components involved in the schematic of the circuit,therefore the utilization report contained higher values. However, in experiment 3, the add and shift algorithm reduced the number of components involved in the circuit so the utilization report contained less values.

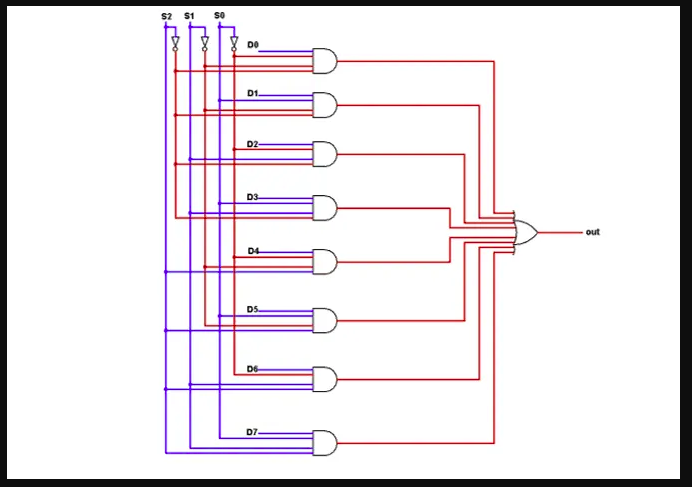
Comment on the delay:

In experiment 2, there were more components involved in the schematic of the circuit,therefore the delay was higher than experiment 3 as the add and shift algorithm reduced the number of components involved in the circuit so the delay reduced.

5)

In experiments 2&3, we used a variable num which was 13 bit to represent a four digit number on the FPGA. In order to represent an 8 digit number we will need 26 bits and we will modify the code accordingly. Modifying the code will be done through dividing the number over 10000 and then 100000 and then 1000000 to place each digit in one display in experiment2. However, modifying the code in experiment3 will be through increasing the number of iterations in the BCD module (inside the for loop).

The 8x1 multiplexer circuit



1. Using gate-level modeling

module (input in1 ,in2 ,in3 ,in4 ,in5 ,in6 ,in7,s1 , s2 ,s3 , output y );

wire w1,w2,w3,w4,w5,w6,w7;

wire s1\_bar , s2\_bar , s3\_bar;

not (s1\_bar,s1);

not(s2\_bar, s2);

not(s3\_bar, s3);

and (w1,in1,s1\_bar,s2\_bar,s3\_bar);

and(w2,in2,s1,s2\_bar,s3\_bar);

and (w3,in3,s1\_bar,s2,s3\_bar);

and (w4,in4,s1,s2,s3\_bar);

and (w5,in5,s1\_bar,s2\_bar,s3);

and (w6,in6,s1,s2\_bar,s3);

and (w7,in7,s1\_bar,s2,s3);

and (w8,in8,s1,s2,s3);

or (y,w1,w2,w3,w4,w5,w6,w7,w8);

endmodule

---------------------------------------------------------------------------------------------------------------------

1. Using dataflow modeling (e.g. using logical equations and operators)

module (input in1 ,in2 ,in3 ,in4 ,in5 ,in6 ,in7,s1 , s2 ,s3 , output y );

wire s1\_bar , s2\_bar , s3\_bar;

assign s1\_bar = ~s1;

assign s2\_bar = ~s2;

assign s3\_bar =~s3;

assign y = (in1&s1\_bar&s2\_bar&s3\_bar)|(in2&s1&s2\_bar&s3\_bar)|(in3&s1\_bar&s2&s3\_bar)|(in4&s1&s2&s3\_bar)|(in5&s1\_bar&s2\_bar&s3)|(in6&s1&s2\_bar&s3)|(in7&s1\_bar&s2&s3)|(in8&s1&s2&s3);

endmodule

---------------------------------------------------------------------------------------------

1. Using behavioral modeling (e.g. using always/case/if)

|  |  |  |  |
| --- | --- | --- | --- |
| s1 | s2 | s3 | output |
| 0 | 0 | 0 | in1 |
| 0 | 0 | 1 | in2 |
| 0 | 1 | 0 | in3 |
| 0 | 1 | 1 | in4 |
| 1 | 0 | 0 | in5 |
| 1 | 0 | 1 | in6 |
| 1 | 1 | 0 | in7 |
| 1 | 1 | 1 | in8 |

module( in1,in2.in3,in4,in5,in6,in7,in8,s1,s2,s3,y);

input wire in1,in2.in3,in4,in5,in6,in7,in8,s1,s2,s3;

Output reg y;

always@(\*);

Begin

Case (s1,s2,s3)

3’b000: y=in1;

3’b001: y=in2;

3’b010: y=in3;

3’b011: y=in4;

3’b100: y=in5;

3’b101: y=in6;

3’b110: y=in7;

3’b111: y=in8;

endcase

end

endmodule

---------------------------------------------------------------------------------------------