Name: Joseph Hany **ID:900182870**

Architecture Lab 1 Report

Experiment 1:

- This experiment aims to implement a simple inverter that takes in an input A and inverts its value and shows the output in wire B.

- In order to implement such a circuit, we should do the following:

  1- write an “Inverter” module that takes an input A and an output B. However, in order to test the output, we should declare A as a wire inside the module and assign it to value since we don’t have access to the lab and can not connect real wires and give real inputs. As a result, the output will be exactly the opposite of A. If A is assigned to zero then B will be 1 and vice versa.

  2- write a “inverter_constraint” file in order to inform the software what physical pins on the Nexys A7-100T FPGA Board that we plan on using or connecting to in relation to the Verilog code that we wrote to describe the behavior of the FPGA. For example, connect the input wire to port J15 in order to visualize the real schematics, then comment it and only connect the output wire B to port H17 in order to see the output on a led in the board.

  3- write an “inverter_tb” testbench in order to simulate the output of the circuit using different cases to check the result.

- **Explaining code functionality by commenting on the code:**

```
module inverter(
//Input A,                // commenting the input until we need to see the
                          // actual circuit
output B);                // specifying the output as B
   wire A;                // hardwiring an input A
   assign A = 1'b0;       // assigning a zero bit to the wire A
   assign B=~A;           // inverting the the current value of A and assigning
                          // it to B
endmodule
```

source code

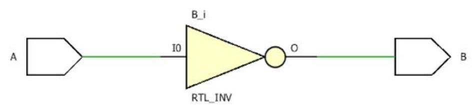
- **Testbench**

```
module inverter_tb();
wire b;
reg a;
inverter u1(.A(a),.B(b));
intial begin
a=1'b0;
#100;
a = 1'b1;
#100;
a = 1'b0;
#100;
a = 1'b1;
end
```

- **Constraint file**

```
#set_property package_pin J15 [get_ports A]
#set_property iostandard LVCMOS33 [get_ports A]

set_property package_pin H17 [get_ports B]
set_property iostandard LVCMOS33 [get_ports B]
```

Schematics:

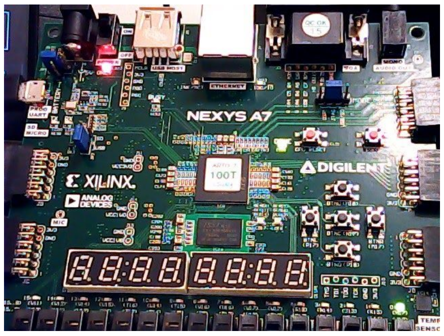


- The circuit is nothing but an inverter with an input and an inverted output.

<u>Results</u>

- As a result in this experiment, the led in the below-left corner is lit because the input A was equal to 0. (as shown in the screenshot below)

<u>Camera Screenshot:</u>



Experiment 2:

- This experiment aims to implement A 4-digit 7-segment display that takes a number and outputs it on the 4-digit 7-segment display.

- In order to implement such a circuit, we should do the following:

  1- Write a “Four_Digit_Seven_Segment_Driver” module that separates the input number into 4 digits one for the thousands digit, one for the hundreds digits, one for the tens, and one for the Ones digit.

  2- write an “FDSSD_constraint” file in order to inform the software what physical pins on the Nexys A7-100T FPGA Board that we plan on using or connecting to in relation to the Verilog code that we wrote to describe the behavior of the FPGA.

  3- write an “inverter_tb” testbench in order to simulate the output of the circuit using different cases to check the result.

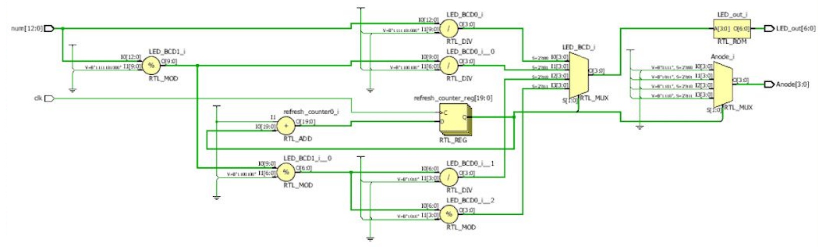
- **Explaining code functionality by commenting on the code:**

```
module Four_Digit_Seven_Segment_Driver (
 input clk,                    // clk is our clock
// input [12:0] num,           // this input is used when we implement the real circuit in
                               // ordet to see the real circuit
 output reg [3:0] Anode,
 output reg [6:0] LED_out      // these seven wires are the seven inputs for the seven
                               // segmet display
 );

 wire [12:0] num=1234;         // the number we want to view on the screen
 reg [3:0] LED_BCD;            // this represents each number on each 7-segment display
```

```
                                        // and acts as a selector for the LED_OUT

reg [19:0] refresh_counter = 0; // 20-bit counter that shifts the display of the current
                                                  //number from one 7-segment to the other very fast
wire [1:0] LED_activating_counter;
always @(posedge clk)
begin
refresh_counter <= refresh_counter + 1;
end

assign LED_activating_counter = refresh_counter[19:18];

always @(*)
begin
case(LED_activating_counter)
2'b00: begin
Anode = 4'b0111;
LED_BCD = num/1000;       // dividing 1234 by 1000 gives us “1” the thousands digit
end
2'b01: begin
Anode = 4'b1011;
LED_BCD = (num % 1000)/100; // 1234 modulus 1000 gives us “234” divided by
                                                  // 100 gives us 2 the hundredth digit
end
2'b10: begin
Anode = 4'b1101;
LED_BCD = ((num % 1000)%100)/10; //1234 modulus 1000 gives us “234” modulus
                              //100 gives us 34 divided by 10 equal 3 which is the tenth digit
end
2'b11: begin
Anode = 4'b1110;
LED_BCD = ((num % 1000)%100)%10;  //1234 modulus 1000 gives us “234” modulus
                              //100 gives us 34 modulus 10 equals 4 which is the ones digit
end
endcase
end
always @(*)
begin
case(LED_BCD)
4'b0000: LED_out = 7'b0000001; // "0"
4'b0001: LED_out = 7'b1001111; // "1"
4'b0010: LED_out = 7'b0010010; // "2"
4'b0011: LED_out = 7'b0000110; // "3"
```

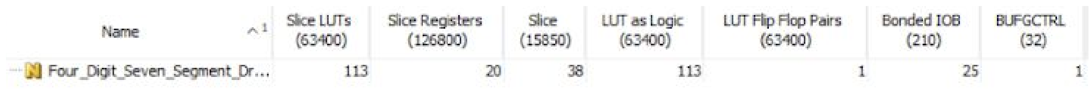
```
4'b0100: LED_out = 7'b1001100; // "4"
4'b0101: LED_out = 7'b0100100; // "5"
4'b0110: LED_out = 7'b0100000; // "6"
4'b0111: LED_out = 7'b0001111; // "7"
4'b1000: LED_out = 7'b0000000; // "8"
4'b1001: LED_out = 7'b0000100; // "9"
default: LED_out = 7'b0000001; // "0"
endcase
end
Endmodule
```

Schematics:



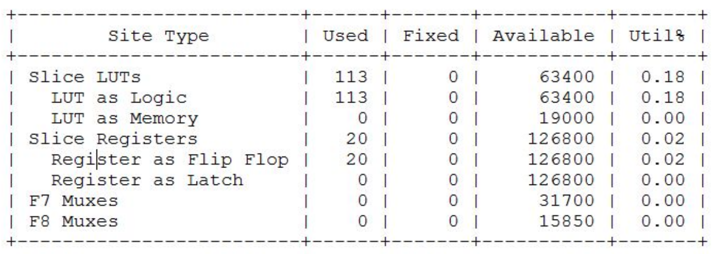
- It would be apparent that this implementation is more complex than the optimized divisor in experiment 3.

Utilization Hierarchy:



Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
Four_Digit_Seven_Segment_Dr…	113	20	38	113	1	25	1

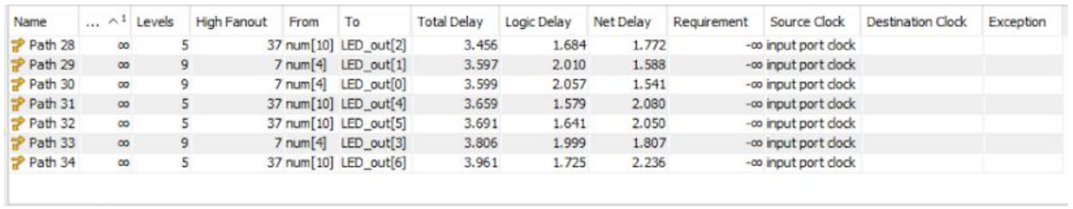
Utilization Report:



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |  113 |     0 |     63400 |  0.18 |
|   LUT as Logic          |  113 |     0 |     63400 |  0.18 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   20 |     0 |    126800 |  0.02 |
|   Register as Flip Flop |   20 |     0 |    126800 |  0.02 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Total slice LUTs used = 113
Total slice Registers used = 20

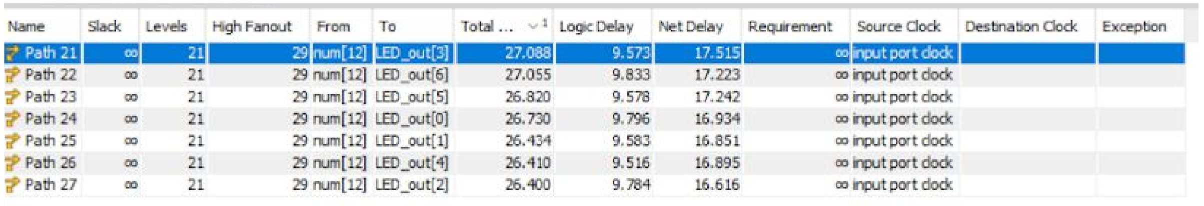
Hold time:



Name	... ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 28	∞	5	37	num[10]	LED_out[2]	3.456	1.684	1.772	-∞	input port clock		
Path 29	∞	9	7	num[4]	LED_out[1]	3.597	2.010	1.588	-∞	input port clock		
Path 30	∞	9	7	num[4]	LED_out[0]	3.599	2.057	1.541	-∞	input port clock		
Path 31	∞	5	37	num[10]	LED_out[4]	3.659	1.579	2.080	-∞	input port clock		
Path 32	∞	5	37	num[10]	LED_out[5]	3.691	1.641	2.050	-∞	input port clock		
Path 33	∞	9	7	num[4]	LED_out[3]	3.806	1.999	1.807	-∞	input port clock		
Path 34	∞	5	37	num[10]	LED_out[6]	3.961	1.725	2.236	-∞	input port clock		

Hold time= 3.961

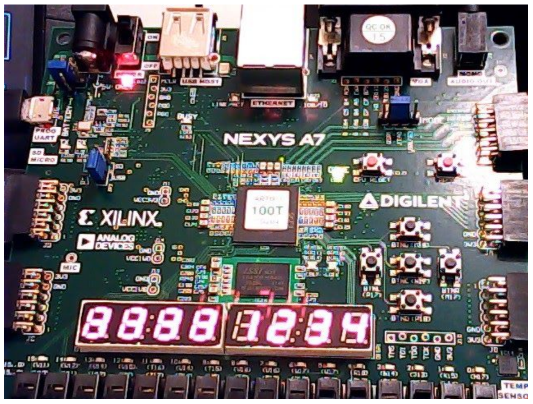
Setup time:



Name	Slack	Levels	High Fanout	From	To	Total ... ∨1	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 21	∞	21	29	num[12]	LED_out[3]	27.088	9.573	17.515	∞	input port clock		
Path 22	∞	21	29	num[12]	LED_out[6]	27.055	9.833	17.223	∞	input port clock		
Path 23	∞	21	29	num[12]	LED_out[5]	26.820	9.578	17.242	∞	input port clock		
Path 24	∞	21	29	num[12]	LED_out[0]	26.730	9.796	16.934	∞	input port clock		
Path 25	∞	21	29	num[12]	LED_out[1]	26.434	9.583	16.851	∞	input port clock		
Path 26	∞	21	29	num[12]	LED_out[4]	26.410	9.516	16.895	∞	input port clock		
Path 27	∞	21	29	num[12]	LED_out[2]	26.400	9.784	16.616	∞	input port clock		

Setup time = 27.088

Camera Screenshot:



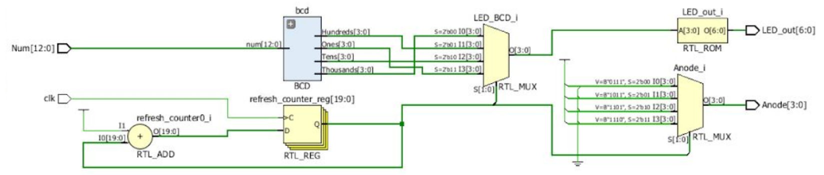
Experiment 3:

- This experiment aims to improve the division part in experiment two using bits operations.
- In order to implement such a circuit, we should define a new module called BCD in which we will use an algorithm called the **double-dabble algorithm** also known as the shift and add 3 which is an efficient algorithm that converts binary numbers to decimal in binary coded decimal (BCD) format by a series of shifts and addition.

Explaining code functionality by commenting on the code:

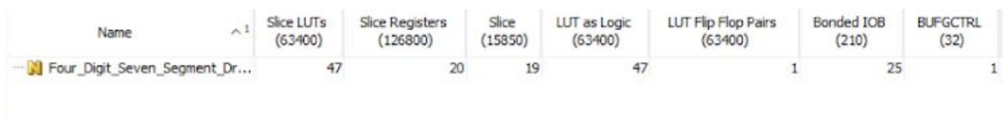
```
module BCD (
input [12:0] num,
output reg [3:0] Thousands,
output reg [3:0] Hundreds,
output reg [3:0] Tens,

output reg [3:0] Ones
);
integer i;
always @(num)
begin
//initialization              //here we instialize the four main 4 bits registers with zeros
 Thousands = 4'd0;
 Hundreds = 4'd0;
 Tens = 4'd0;
 Ones = 4'd0;

for (i = 12; i >= 0 ; i = i-1 )
begin
if(Thousands  >= 5)                      // everytime wecheck whether the number contained in the
                                         //register is equal to or greater than 5 then we ad 3 to it
 Thousands  = Thousands + 3;
if(Hundreds >= 5)
 Hundreds = Hundreds + 3;
if (Tens >= 5)
 Tens = Tens + 3;
if (Ones >= 5)
 Ones = Ones +3;
//shift left one
 Thousands = Thousands <<1;              // we shift each register one bit to the left and insert in the
                                         // LSB the MSB inside the register below it
                                         // until we reach the end of the number bus
 Thousands [0] = Hundreds[3];
 Hundreds = Hundreds << 1;
 Hundreds [0] = Tens [3];
 Tens = Tens << 1;
 Tens [0] = Ones[3];
 Ones = Ones << 1;
 Ones[0] = num[i];
end

end
endmodule
```

Schematic:



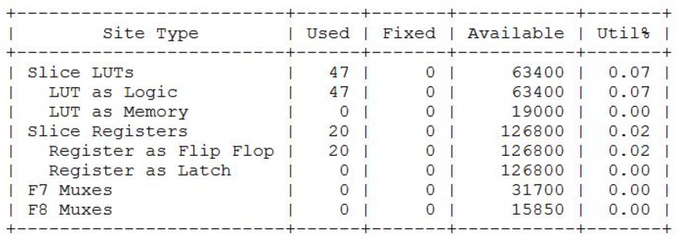
Here, it can be clearly seen that there are fewer components in the implementation of the same circuit but with optimized divisor, since bits operations are cheaper than dividing and taking the modulus of a number.

Utilization Hierarchy:



Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	Bonded IOB (210)	BUFGCTRL (32)
Four_Digit_Seven_Segment_Dr...	47	20	19	47	1	25	1

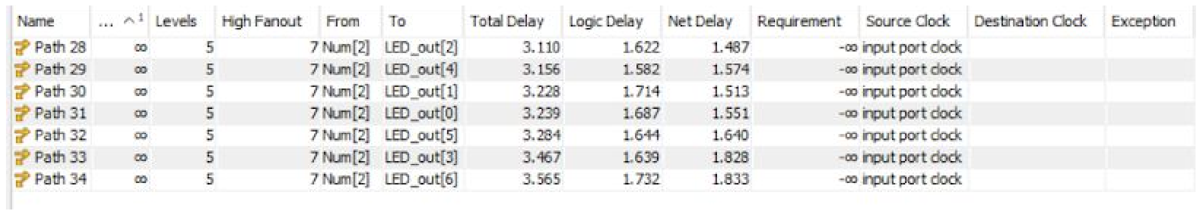
Utilization Report:



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |   47 |     0 |     63400 |  0.07 |
|   LUT as Logic          |   47 |     0 |     63400 |  0.07 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   20 |     0 |    126800 |  0.02 |
|   Register as Flip Flop |   20 |     0 |    126800 |  0.02 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Total slice LUTs used = 47
Total slice Registers used = 20

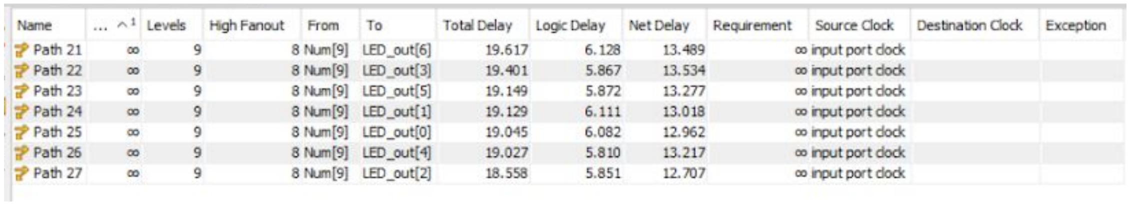
Hold time:



Name	... ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 28	∞	5	7	Num[2]	LED_out[2]	3.110	1.622	1.487	-∞	input port clock		
Path 29	∞	5	7	Num[2]	LED_out[4]	3.156	1.582	1.574	-∞	input port clock		
Path 30	∞	5	7	Num[2]	LED_out[1]	3.228	1.714	1.513	-∞	input port clock		
Path 31	∞	5	7	Num[2]	LED_out[0]	3.239	1.687	1.551	-∞	input port clock		
Path 32	∞	5	7	Num[2]	LED_out[5]	3.284	1.644	1.640	-∞	input port clock		
Path 33	∞	5	7	Num[2]	LED_out[3]	3.467	1.639	1.828	-∞	input port clock		
Path 34	∞	5	7	Num[2]	LED_out[6]	3.565	1.732	1.833	-∞	input port clock		

Hold time= 3.565

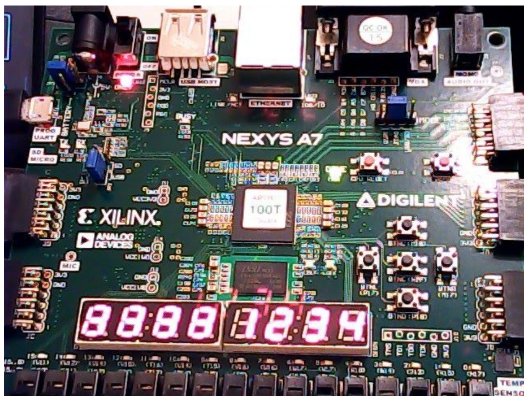
Setup time:



Name	... ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 21	∞	9	8	Num[9]	LED_out[6]	19.617	6.128	13.489	∞	input port clock		
Path 22	∞	9	8	Num[9]	LED_out[3]	19.401	5.867	13.534	∞	input port clock		
Path 23	∞	9	8	Num[9]	LED_out[5]	19.149	5.872	13.277	∞	input port clock		
Path 24	∞	9	8	Num[9]	LED_out[1]	19.129	6.111	13.018	∞	input port clock		
Path 25	∞	9	8	Num[9]	LED_out[0]	19.045	6.082	12.962	∞	input port clock		
Path 26	∞	9	8	Num[9]	LED_out[4]	19.027	5.810	13.217	∞	input port clock		
Path 27	∞	9	8	Num[9]	LED_out[2]	18.558	5.851	12.707	∞	input port clock		

Setup time = 19.617

Camera Screenshot:



Conclusion

	Not optimized divisor (Experiment 2)	**Optimized divisor (Experiment 3)**
Total slice LUTs used	113	47
Total slice Registers used	20	20
Hold time	3.961	3.565
Setup time	27.088	19.617

The optimized divisor in experiment 3 clearly uses fewer components (slice LUTs) and takes less hold and setup time.