Report 1

Technical summary of experiments conducted in the lab (Steps, Results components (a screenshot with Schematic design might help), code functionality, etc...]

Experiment 1: A simple inverter

The main purpose of these experiments was to review Verilog and FPGA Refresher, as we did a simple inventor to turn on FPGA LED.

➔ **Steps**

After opening VIVADO, we should create design and constraint files. In the design file whose extension is **.v**. Then, we should specify the input and the output, and write the main code inside the body of the module we should assign x = ~ y. In the constraint file whose extension is **.xdc**, we should connect the input with the output (the input bin with the output LED), by writing this code for the bin and the LED successfully.(set_property -dict { PACKAGE_PIN J15 IOSTANDARD LVCMOS33 } [get_ports { SW[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0], #set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { LED[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]).

➔ **Results**

The bin labeled J15 will turn on the LED labeled H17. The LED will turn on when the bin is 0 because the FPG is actively low.

Experiment 2: A 4-digit 7-segment display driver

The main purpose of these experiments was to design and implement a 4-digit 7-segment display driver module controlled by the 13 input switches.

➔ Steps

After opening VIVADO, we should create design and constraint files. In the design file, whose extension is .v, we should specify the input and output and write the main code inside the body of the module. We should write the Four_Digit_Seven_Segment_Driver module with the code given in the lap manual.

```
module Four_Digit_Seven_Segment_Driver (
   input clk,
   input [12:0] num,
   output reg [3:0] Anode,
   output reg [6:0] LED_out
   );

   reg [3:0] LED_BCD;
   reg [19:0] refresh_counter = 0; // 20-bit counter
   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;
            end
         2'b01: begin
            Anode = 4'b1011;
            LED_BCD = (num % 1000)/100;
            end
         2'b10: begin
            Anode = 4'b1101;
            LED_BCD = ((num % 1000)%100)/10;
            end
         2'b11: begin
            Anode = 4'b1110;
            LED_BCD = ((num % 1000)%100)%10;
            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
```

In the constraint file, whose extension is **.xdc**, we should connect the output with the variables to set up the FPGA. The files correctly contain the written code.

➔ Result

The 7-segment display driver module is controlled by the 13 input switches and will work correctly by the switches.

Experiment 3: A 4-digit 7-segment display driver with optimized Divisor

This experiment is basically like the previous one (Experiment 2) with slight differences.

➔ Steps

We will use the same constraint file without any changes. However, we will use two design files Four_Digit_Seven_Segment_Driver_Optimized.v and BCD.v. The BCD.v file contains the following code

```
module BCD (
      input [7:0] num,
      output reg [3:0] Hundreds,
      output reg [3:0] Tens,
      output reg [3:0] Ones
      );
integer i;
always @(num)
begin
      //initialization
      Hundreds = 4'd0;
      Tens = 4'd0;
      Ones = 4'd0;
      for (i = 7; i >= 0 ; i = i-1 )
      begin
            if(Hundreds >= 5 )
                  Hundreds = Hundreds + 3;
            if (Tens >= 5 )
                  Tens = Tens + 3;
            if (Ones >= 5)
                  Ones = Ones +3;

            //shift  left  one
            Hundreds = Hundreds << 1;
            Hundreds [0] = Tens [3];
            Tens = Tens << 1;
            Tens [0] = Ones[3];
            Ones = Ones << 1;
            Ones[0] = num[i];
      end
end
endmodule
```

This module helped us in the Four_Digit_Seven_Segment_Driver_Optimized.v instead of write the code by this format

```
always @(*)
   begin
      case(LED_activating_counter)
      2'b00: begin
         Anode = 4'b0111;
         LED_BCD = num/1000;
         end
      2'b01: begin
         Anode = 4'b1011;
         LED_BCD = (num % 1000)/100;
         end
      2'b10: begin
         Anode = 4'b1101;
         LED_BCD = ((num % 1000)%100)/10;
         end
      2'b11: begin
         Anode = 4'b1110;
         LED_BCD = ((num % 1000)%100)%10;
         end
      endcase
   end
```

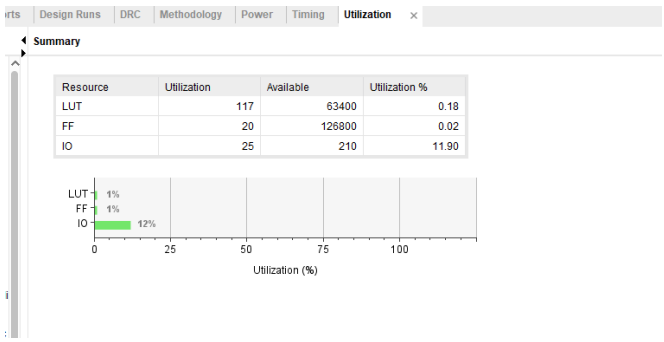
We wrote Ones, Tens, Hundreds, and Thousands.

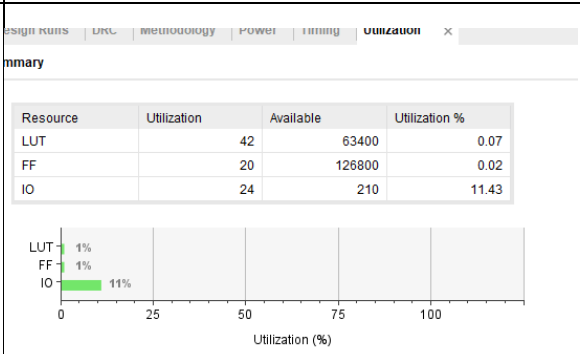
➔ Results

The result of this experiment is the same result of Experiment 2.

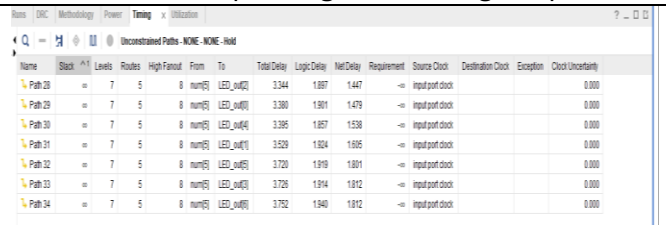
Sara Ali **Report1** **900204902**

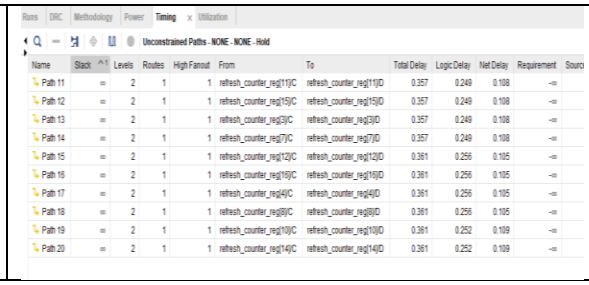
Compare the utilization and delay of experiment 2 vs experiment 3

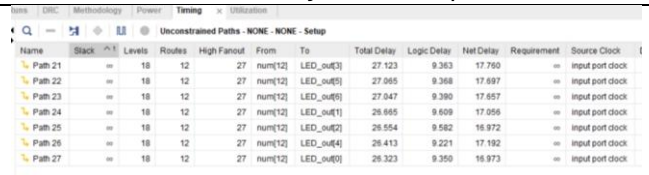
<table>
<tr><th></th><th>Experiment 1</th><th>Experiment 2</th></tr>
<tr><td>Utilization</td><td>


</td><td>


</td></tr>
<tr><td>Report</td><td colspan="2">The LUT Utilization of the second experiment is more than the third experiment as the second is 117 and the third is 42 (0.18% for the second experiment and 0.07% for the third one). However, the availability is the same for both experiments. The FF in Utilization and availability for both experiments is the same. IO in the second experiment (25) is more than the third experiment (24) by 1, which is 11.90% for the second experiment and 11.43% for the third one.

• Efficiency: The third experiment shows lower LUT utilization (0.07% vs. 0.18%), which might indicate better efficiency or less complexity in the design.
• Detail: The second summary provides more detailed breakdowns of specific logic blocks and configurations, which could be useful for deeper analysis.
• Specifics: The third experiment provides more details on IO and memory, which could be crucial depending on the design requirements.</td></tr>
<tr><td>Timing summary hold</td><td>


</td><td>


</td></tr>
<tr><td>Report</td><td colspan="2">There is a significant difference between both experiments in the total, logic, and net delays, as it shown that the the delay of third experiment is less that the delay of the second one.</td></tr>
<tr><td>Timing summary Setup</td><td>


</td><td>


</td></tr>
<tr><td>Report</td><td colspan="2">There is a significant difference between both experiments in the total, logic, and net delays, as it shown that the the delay of third experiment is less that the delay of the second one.

This means that in terms of timing and delay, the third experiment is more efficient than the second experiment.</td></tr>
</table>