Class:	**CPE300L Digital Systems Architecture and Design 1001**		Semester:	**Fall 2024**
Points		Document author:	**Narek Kalikian**	
		Author's email:	**kalikn1@unlv.nevada.edu**	
		Document topic:	**Postlab 2**	
Instructor's comments:				

1. Introduction / Theory of Operation

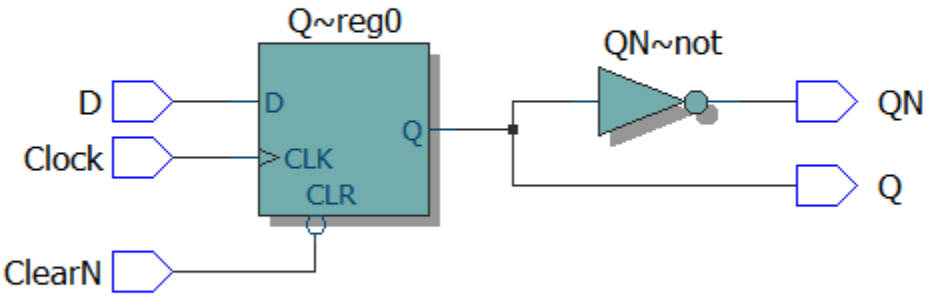
The experiments during this lab consisted of latches, flip-flops, synchronous, and asynchronous system designs. These elements were present in different experiments and some worked together to accomplish certain functions. A D flip-flop is a logic element that takes in one input and its output is driven to match this single input when the clock is high. A JK flip-flop has two inputs; its output is determined by the current values of these inputs and the previous state. Waveforms help us confirm that the elements we coded and implemented are functioning correctly. This is done by checking what the output(s) will be when the inputs (input signals, clock signal, clear/reset signals) have certain values. Testbenches enhance this process by making the waveform simulations faster and more automated. Rather than forcing values on all of the inputs and testing each of these combinations to view the respective outputs, a testbench can handle all of this functionality in one go.

2. Description of Experiments

Experiment 1 - D Flip Flop Logic Code:

```
1   module Dflipflop (D, Clock, ClearN, Q, QN) ;
2     input D, Clock, ClearN;
3     output Q, QN;
4     reg Q;
5     always @(posedge Clock, negedge ClearN)
6     begin
7       if(~ClearN)
8         Q = 1'b0;
9       else
10        Q = D;
11      end
12      assign QN = ~Q;
13    endmodule
14
```

Experiment 1 - D Flip Flop RTL Schematic:



Experiment 2 - Register Logic Code:

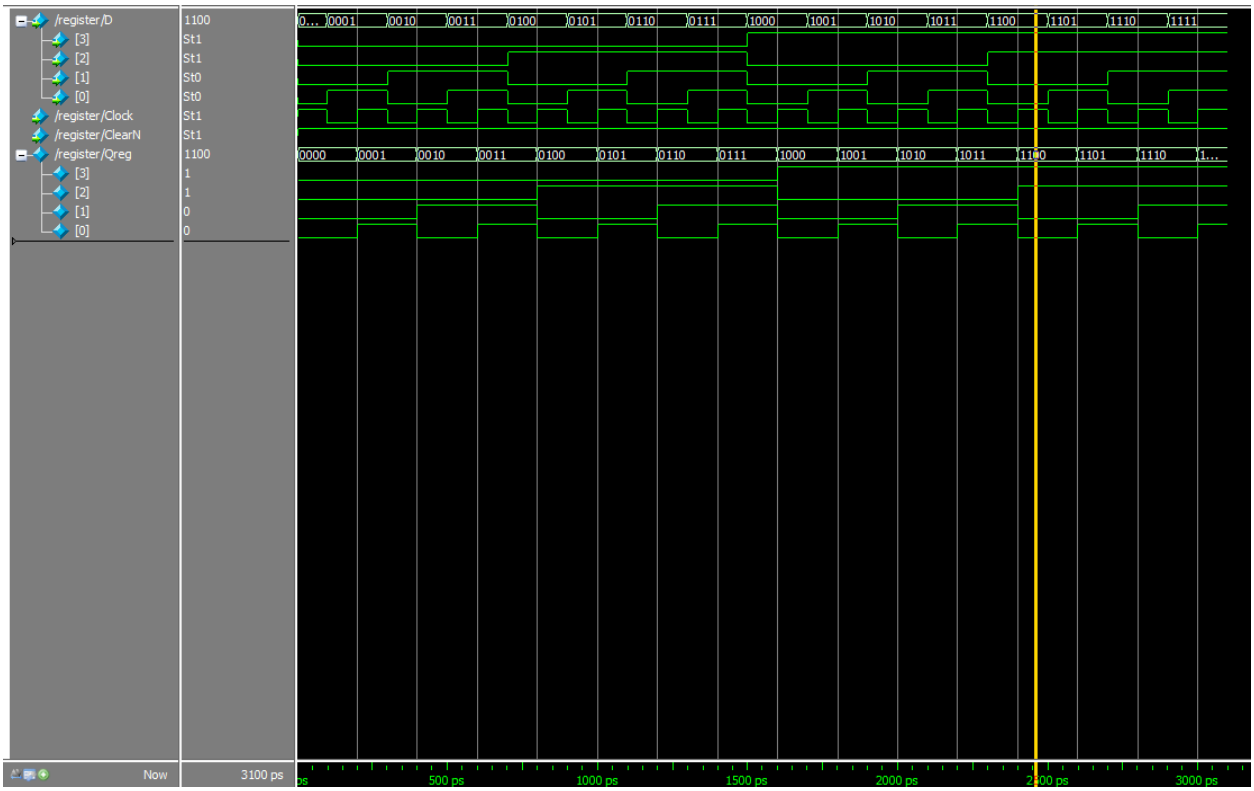
```
14  module register (D, Clock, ClearN, Q);
15
16      input [3:0] D;
17      input Clock, ClearN;
18      output [3:0] Q;
19
20      wire c1, c2, c3, c4;
21
22      DFF FF0(D[0], Clock, ClearN, Q[0]);
23      DFF FF1(D[1], Clock, ClearN, Q[1]);
24      DFF FF2(D[2], Clock, ClearN, Q[2]);
25      DFF FF3(D[3], Clock, ClearN, Q[3]);
26
27      reg[3:0] Qreg;
28
29      always @ (posedge Clock, negedge ClearN)
30      begin
31          if(~ClearN)
32              Qreg = 4'b0;
33          else
34              Qreg[0] <= D[0];
35              Qreg[1] <= D[1];
36              Qreg[2] <= D[2];
37              Qreg[3] <= D[3];
38      end
39
40  endmodule
```

Experiment 2 - Register Test Bench Code:

```
1   module DFFtb;
2
3     reg [3:0] D;
4     reg Clock;
5     reg ClearN;
6     wire [3:0] Q;
7
8   register uut (
9       .D(D),
10      .Clock(Clock),
11      .ClearN(ClearN),
12      .Q(Q)
13  );
14
15  initial
16    begin
17      Clock = 1;
18      ClearN = 1;
19      D = 4'b0000;
20      #100;
21
22      Clock = 0;
23      D = 4'b0001;
24      #100;
25
26      Clock = 1;
27      #100;
28
29      Clock = 0;
30      D = 4'b0010;
31      #100;
32
33      Clock = 1;
34      #100;
35
36      Clock = 0;
37      D = 4'b0011;
38      #100;
39
40      Clock = 1;
41      #100;
42
43      Clock = 0;
44      D = 4'b0100;
45      #100;
46
47      Clock = 1;
48      #100;
```

```
        Clock = 0;
        D = 4'b0101;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b0110;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b0111;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1000;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1001;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1010;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1011;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1100;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1101;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1110;
        #100;

        Clock = 1;
        #100;

        Clock = 0;
        D = 4'b1111;
        #100;

        Clock = 1;
        #100;

        $stop;
    end

initial
    begin
        $monitor("Time = %0t | D = %b | ClearN = %b | Q = %b", $time, D, ClearN, Q);
    end

endmodule
```

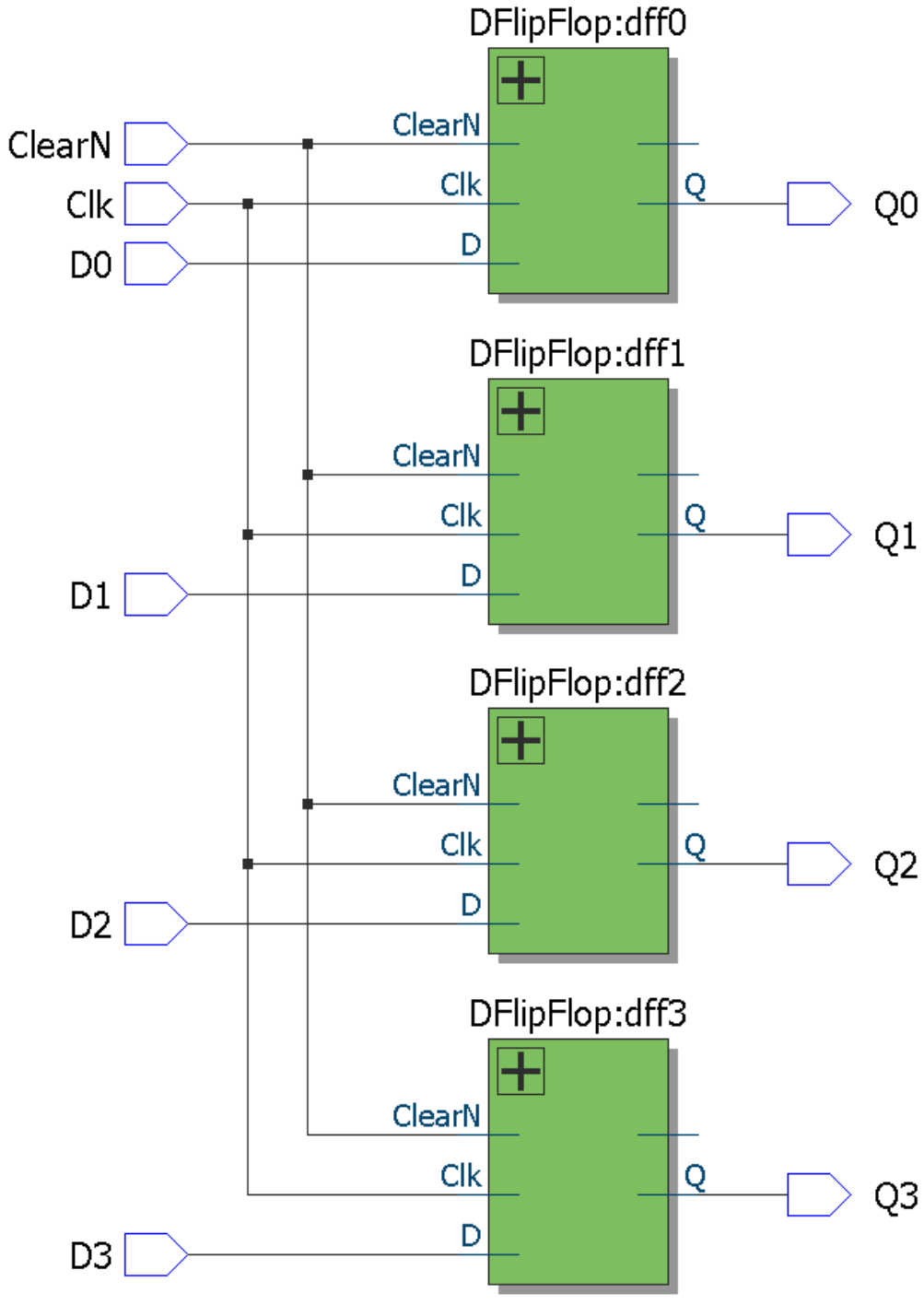
Experiment 2 - Register Waveform Simulation:



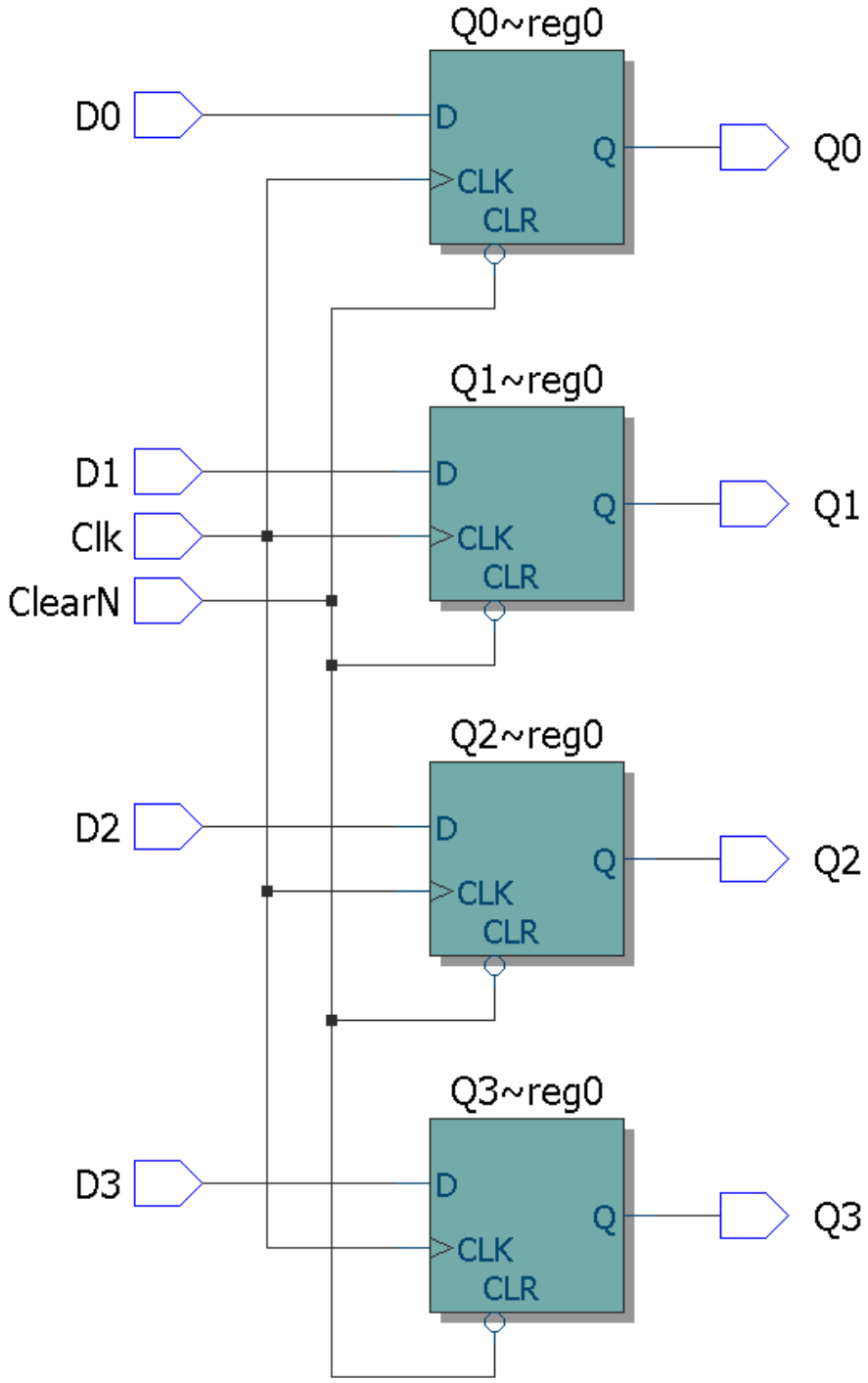
Experiment 3 - Behavioral Code:

```
17  module register (
18      input Clk, ClearN, D0, D1, D2, D3,
19      output reg Q0, Q1, Q2, Q3
20  );
21
22  always @(posedge Clk or negedge ClearN) begin
23      if (ClearN == 0) begin
24          Q0 <= 0;
25          Q1 <= 0;
26          Q2 <= 0;
27          Q3 <= 0;
28      end else begin
29          Q0 <= D0;
30          Q1 <= D1;
31          Q2 <= D2;
32          Q3 <= D3;
33      end
34  end
35
36  endmodule
```

Experiment 3 - Structural RTL Schematic:



Experiment 3 - Behavioral RTL Schematic:



The structural RTL view provides a more detailed, gate-level view of each of the D Flip Flop's connections whereas the behavioral RTL view provides a high-level, abstract view of the component and its functionality.

Experiment 4 - Counter Verilog Code

```
module counter (
    input clk,
    input reset,
    input load,
    input [4:0] data,
    input up,
    input down,
    output reg [4:0] count,
    output reg [6:0] segments,
    output reg [6:0] segments2
);

    reg [24:0] clk_divider;
    reg slow_clk;

    always @(posedge clk) begin
        clk_divider <= clk_divider + 1;
        slow_clk <= clk_divider[24];
    end

    always @(posedge slow_clk or posedge reset) begin
        if (reset) begin
            count <= 5'b0;
        end
        else if (load) begin
            count <= data;
        end
        else begin
            if (up) begin
                if (count == 5'd25)
                    count <= 5'b0;
                else
                    count <= count + 1;
            end
            else if (down) begin
                if (count == 5'b0)
                    count <= 5'd25;
                else
                        count <= count - 1;
              end
          end
     end


always @(count)
  case (count)
    0:
    begin
    segments = 7'b100_0000;
    segments2 = 7'b111_1111;
    end
    1:
    begin
    segments = 7'b111_1001;
    segments2 = 7'b111_1111;
    end
    2:
    begin
    segments = 7'b010_0100;
    segments2 = 7'b111_1111;
    end
    3:
    begin
    segments = 7'b011_0000;
    segments2 = 7'b111_1111;
    end
    4:
    begin
    segments = 7'b001_1001;
    segments2 = 7'b111_1111;
    end
    5:
    begin
    segments = 7'b001_0010;
    segments2 = 7'b111_1111;
    end
```

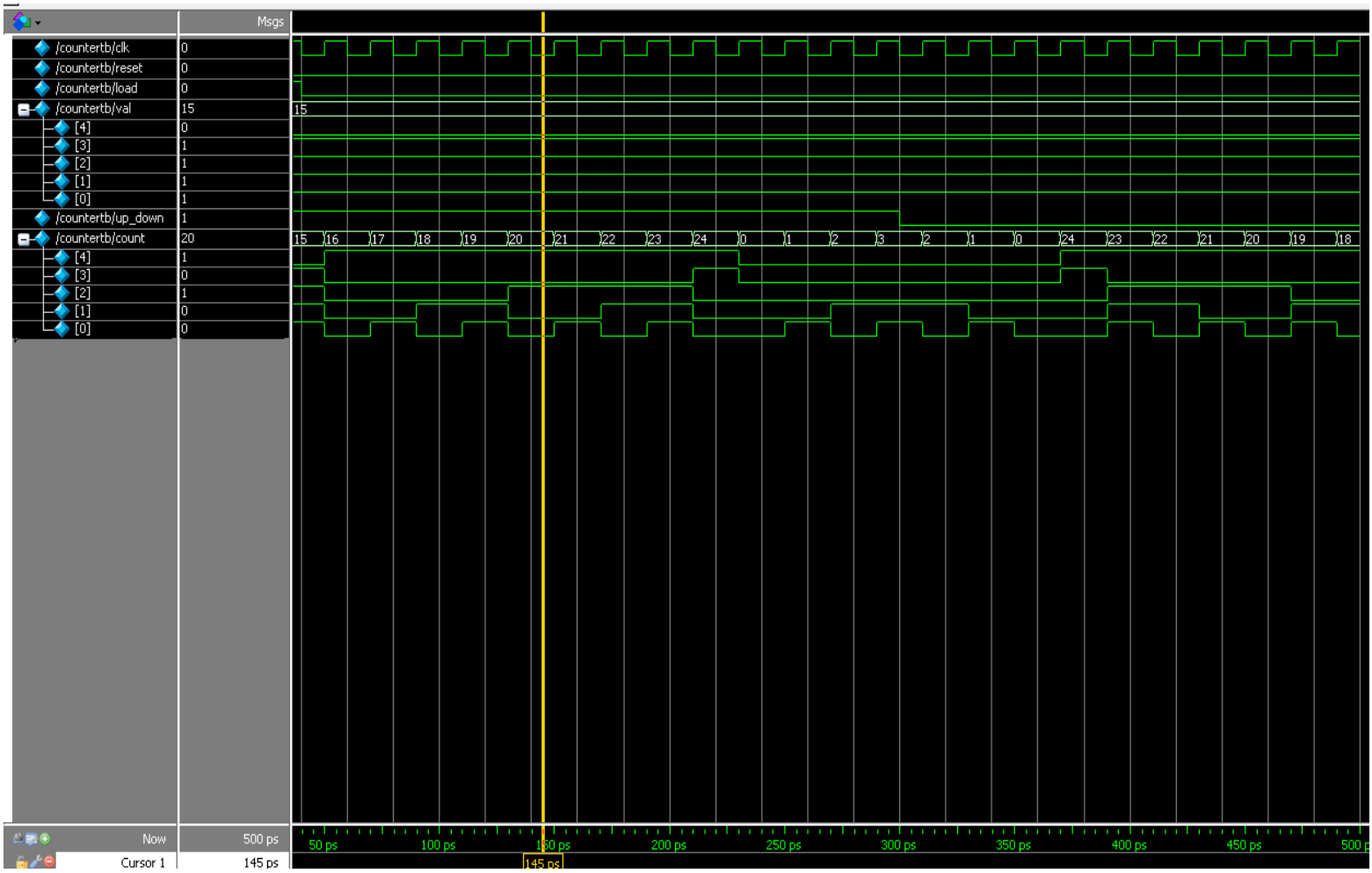
```
77          6:
78          begin
79          segments = 7'b000_0010;
80          segments2 = 7'b111_1111;
81          end
82          7:
83          begin
84          segments = 7'b111_1000;
85          segments2 = 7'b111_1111;
86          end
87          8:
88          begin
89          segments = 7'b000_0000;
90          segments2 = 7'b111_1111;
91          end
92          9:
93          begin
94          segments = 7'b001_1000;
95          segments2 = 7'b111_1111;
96          end
97          10: begin
98                  segments = 7'b100_0000;
99                  segments2 = 7'b111_1001;
100         end
101       11 :
102         begin
103          segments = 7'b111_1001;
104          segments2 = 7'b111_1001;
105         end
106       12 :
107         begin
108          segments2 = 7'b111_1001;
109          segments = 7'b010_0100;
110         end
111       13 :
112         begin
113          segments2 = 7'b111_1001;
114          segments = 7'b011_0000;
115         end
116       14 :
117         begin
118          segments2 = 7'b111_1001;
119          segments = 7'b001_1000;
120         end
121       15 :
122         begin
123          segments2 = 7'b111_1001;
124          segments = 7'b001_0010;
125         end
126       16 :
127         begin
128          segments2 = 7'b111_1001;
129          segments = 7'b000_0010;
130         end
131       17 :
132         begin
133          segments2 = 7'b111_1001;
134          segments = 7'b111_1000;
135         end
136       18 :
137         begin
138          segments2 = 7'b111_1001;
139          segments = 7'b000_0000;
140         end
141       19 :
142         begin
143          segments2 = 7'b111_1001;
144          segments = 7'b001_1000;
145         end
146       20 :
147         begin
148          segments2 = 7'b010_0100;
149          segments = 7'b100_0000;
150         end
151       21 :
152         begin
```

```
153             segments2 = 7'b010_0100;
154             segments = 7'b111_1001;
155           end
156         22 :
157           begin
158             segments2 = 7'b010_0100;
159             segments =  7'b010_0100;
160           end
161         23 :
162           begin
163             segments2 = 7'b010_0100;
164             segments =  7'b011_0000;
165           end
166         24 :
167           begin
168             segments2 = 7'b010_0100;
169             segments =  7'b001_1001;
170           end
171         25 :
172           begin
173             segments2 = 7'b010_0100;
174             segments =  7'b001_0010;
175           end
176           default: begin
177               segments = 7'b111_1111;
178               segments2 = 7'b111_1111;
179           end
180       endcase
181   endmodule
```

Experiment 4 - Testbench Code:

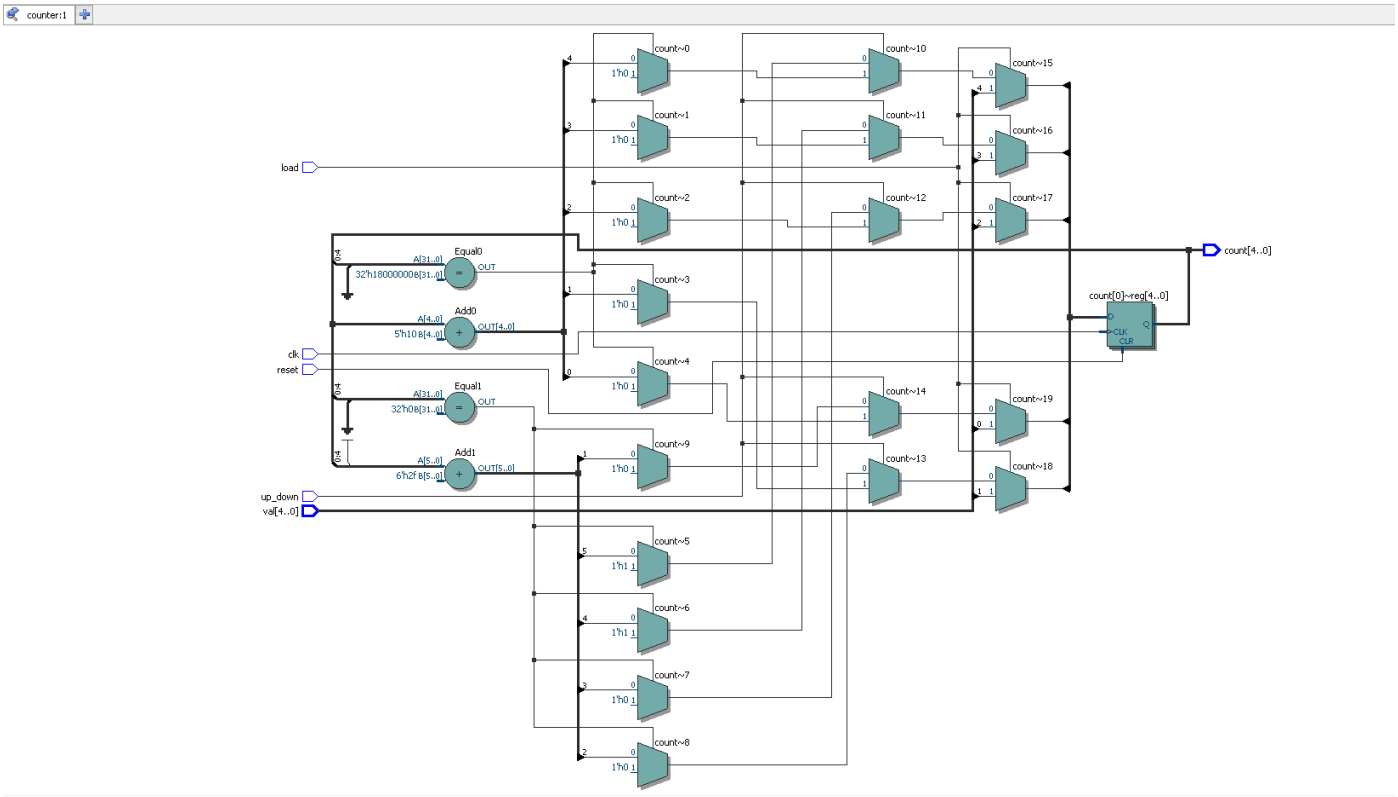
```
1   module countertb;
2       reg clk;
3       reg reset;
4       reg load;
5       reg [4:0] val;
6       reg up_down;
7       wire [4:0] count;
8
9       counter uut (
10          .clk(clk),
11          .reset(reset),
12          .load(load),
13          .val(val),
14          .up_down(up_down),
15          .count(count)
16      );
17
18      always #10 clk = ~clk;
19
20      initial begin
21
22          clk = 0;
23          reset = 0;
24          load = 0;
25          val = 5'b0;
26          up_down = 1;
27
28          reset = 1;
29          #20;
30          reset = 0;
31
32          load = 1;
33          val = 5'd15;
34          #20;
35          load = 0;
36
37          up_down = 1;
38          #260;
39
40          up_down = 0;
41          #200;
42
43          $stop;
44      end
45  endmodule
```

Experiment 4 - Counter Waveform:




```
# Loading work.countertb
# Loading work.counter
VSIM 20> run -all
# Break in Module countertb at C:/altera/13.0sp1/countertb.v line 43

VSIM 21>
```

Experiment 4 - Counter RTL Schematic:



Experiment 4 - Count From 15 Up to 25, Back to 0 & Count From 10 Down to 0, Back to 25:

Attached to Canvas submission along with this report

Experiment 4 - Compilation Report:

```
        ************************************************************
        Running Quartus II 64-Bit Analysis & Synthesis
        Command: quartus_map --read_settings_files=on --write_settings_files=off test1 -c test1
 20028 Parallel compilation is not licensed and has been disabled
 12021 Found 1 design units, including 1 entities, in source file test1.v
 12127 Elaborating entity "counter" for the top level hierarchy
 10230 Verilog HDL assignment warning at test1.v(17): truncated value with size 32 to match size of target (25)
 10230 Verilog HDL assignment warning at test1.v(33): truncated value with size 32 to match size of target (5)
 10230 Verilog HDL assignment warning at test1.v(39): truncated value with size 32 to match size of target (5)
 13024 Output pins are stuck at VCC or GND
286030 Timing-Driven Synthesis is running
 16010 Generating hard_block partition "hard_block:auto_generated_inst"
 21057 Implemented 101 device resources after synthesis - the final resource count might be different
        Quartus II 64-Bit Analysis & Synthesis was successful. 0 errors, 6 warnings
        ************************************************************
        Running Quartus II 64-Bit Fitter
        Command: quartus_fit --read_settings_files=off --write_settings_files=off test1 -c test1
        qfit2_default_script.tcl version: #1
        Project  = test1
        Revision = test1
 20028 Parallel compilation is not licensed and has been disabled
119006 Selected device EP4CE115F29C7 for design "test1"
 21077 Core supply voltage is 1.2V
 21077 Low junction temperature is 0 degrees C
 21077 High junction temperature is 85 degrees C
171003 Fitter is performing an Auto Fit compilation, which may decrease Fitter effort to reduce compilation time
292013 Feature LogicLock is only available with a valid subscription license. You can purchase a software subscription to gain full access to this feature.
176444 Device migration not selected. If you intend to use device migration later, you may need to change the pin assignments as they may be incompatible with other devices
176445 Device EP4CE40F29C7 is compatible
176445 Device EP4CE40F29I7 is compatible
176445 Device EP4CE30F29C7 is compatible
```

3. Questions and Answers

1. Logic Utilization: An ALUT is an adaptive look-up table. These are used to implement digital logic from the Verilog code onto the FPGA. The number of ALUTs generated and listed in the Quartus compilation report will tell you how much of the FPGA's logic resources will be used. Dedicated logic registers are used to values. Similar to ALUTs, they are listed in the Quartus compilation report. However, they tell you how much of the FPGA's sequential resources will be used.

2. Clock Gating: Clock gating is a technique used in FPGA and ASIC designs that selectively disables clock signals to unused circuitry. We need them because they significantly improve power consumption, reliability, and the overall performance of these designs.

3. Testbenches: A function can only return a single value and is executed in zero simulation time. A task on the other hand does not return any value but can perform complex operations like timing delays.

4. Conclusions

In this lab, we were able to continue getting comfortable with programming in Verilog, simulating waveforms and schematics, and implementing them on an FPGA development board. We learned about latches, flip-flops, and registers and tested their functionality using Quartus and ModelSim.