Class:	CPE300L	Semester:	Fall 2021

Points		Document author:	Jerrod Batu
		Author's email:	batuj1@unlv.nevada.edu
		Document topic:	Postlab 4

Instructor's comments:

1. Introduction / Theory of Operation

Throughout this lab, I will continue Verilog review by learning more about latches and flip-flops, synchronous and asynchronous operations, and asynchronous system design. Some of these latches and flip-flops would include a gated D latch with asynchronous Set' and Clear', JK flip-flop with synchronous clear, D flip flop with asynchronous low clear, etc.

1. A **D flip-flop** is a digital electronic circuit that delays output state change until its next rising edge where an input would be called upon. Specifically, the D flip-flop behaves similar to memory as the output will stay constant until it is altered from the D input at the rising clock edge. They are, in fact, utilized as building block shift registers that can store clock cycles. A **JK flip-flop**, in general, is a gated S-R latch with the AND and NOR gates replaced as NAND gates. JK flip-flops are the most used flip-flop designs because they are very universal. The two inputs "J" and "K" are named after the inventor Jack Kilby. The JK flip-flop acts only at a rising clock edge, and the output would toggle from state to state. When J and K are both low, the output is retained and no changes would occur.
2. **Waveforms** are used to check the correctness of a circuit by comparing inputs to the outputs in a "form of a wave"; therefore, the inputs would be clocked at a specific time period that the output would follow based on the combinational logic of the electronic circuit. Waveforms would not be ideal for a large number of inputs as this would become a long and tedious process. **Testbenches** enhances this process by comparing the user's Verilog code to its expected output; hence, comparing a large number of inputs and outputs would allow the user to efficiently check the correctness of a circuit.

2. Prelab

https://docs.google.com/document/d/18KNm2l3hURA8lTZ1jOOj9OqBpmHYE14P/edit?usp=sharing&ouid=102808507017671072128&rtpof=true&sd=true
This is the link to my prelab 4 submission.

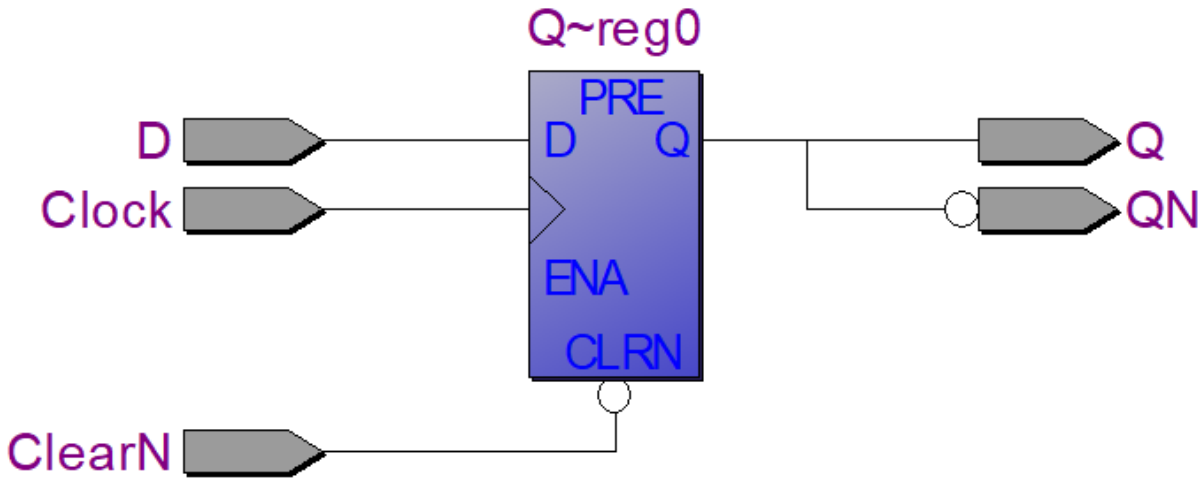
3. Results of Experiments

Experiment 1

a.

```
1   module asyncDFF (D, Clock, ClearN, Q, QN);
2      input D, Clock, ClearN;
3      output Q, QN;
4      reg Q;
5
6      always @ (posedge Clock, negedge ClearN)
7         begin
8            if (~ClearN)
9               Q = 0;
10           else
11              Q = D;
12        end
13        assign QN = ~Q;
14  endmodule
15
```

This is the Verilog code for a D flip flop with asynchronous low clear and complementary output.



This is the RTL view for my D flip flop with asynchronous low clear and complementary output.

Experiment 2

a.

```
1  `timescale 1 ns / 100 ps
2
3  module regDFF (D, Clock ,ClearN, Q, QN);
4      input [3:0] D;
5      input Clock, ClearN;
6      output [3:0] Q, QN;
7
8      asyncDFF DFF0 (D[0], Clock, ClearN, Q[0], QN[0]);
9      asyncDFF DFF1 (D[1], Clock, ClearN, Q[1], QN[1]);
10     asyncDFF DFF2 (D[2], Clock, ClearN, Q[2], QN[2]);
11     asyncDFF DFF3 (D[3], Clock, ClearN, Q[3], QN[3]);
12 endmodule
13
14 // asynchronous D Flip Flop
15 module asyncDFF (D, Clock, ClearN, Q, QN);
16     input D, Clock, ClearN;
17     output Q, QN;
18     reg Q;
19
20     always @ (posedge Clock, negedge ClearN)
21         begin
22             if (~ClearN)
23                 Q <= 0;
24             else
25                 Q <= D;
26         end
27         assign QN = ~Q;
28 endmodule
29
```

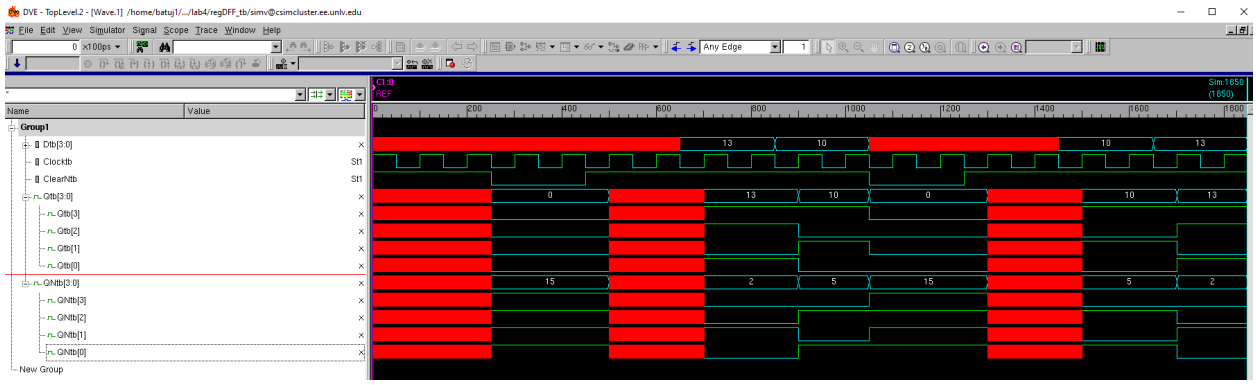
This is the Verilog code for instantiating the DFF from experiment 1 into a 4-bit register which uses D flip-flops.

b.

```
`timescale 1 ns / 100 ps

// register D Flip Flop Testbench
module regDFF_tb;
  reg [3:0] Dtb;
  reg Clocktb = 1'b1;
  reg ClearNtb = 1'b1;
  wire [3:0] Qtb, QNtb;

  `define PERIOD 10

  always
    #(`PERIOD/2) Clocktb = ~Clocktb;

    regDFF U0 (
      .D (Dtb),
      .Clock (Clocktb),
      .ClearN (ClearNtb),
      .Q (Qtb),
      .QN (QNtb)
    );

    initial begin
      $timeformat (-9, 1, "ns", 9 );
      $monitor ( "time=%t     Dtb = %b     ClrNtb = %b     Qtb = %b     QNtb = %b", $time,   Dtb,   ClearNtb,   Qtb,   QNtb);
      #(`PERIOD * 100)
      $display ( "TESTING TIMEOUT" );
      $finish;
    end

  task expectedResult (input [3:0] expected);
    if (Qtb !== expected)
      begin
        $display ( "Qtb=%b, but expected value is %b", Qtb, expected);
        $display ( "Test Failed" );
        $finish;
      end
  endtask

  task expectedResultN (input [3:0] expectedN);
    if (QNtb !== expectedN)
      begin
        $display ( "QNtb=%b, but expected value is %b", QNtb, expectedN);
        $display ( "Test Failed" );
        $finish;
      end
  endtask

  initial
    begin
      @(negedge Clocktb)
      { ClearNtb, Dtb } = 5'b1_XXXX; @(negedge Clocktb) expectedResult ( 4'bxxxx ); @(negedge Clocktb) expectedResultN ( 4'bxxxx );
      { ClearNtb, Dtb } = 5'b0_XXXX; @(negedge Clocktb) expectedResult ( 4'b0000 ); @(negedge Clocktb) expectedResultN ( 4'b1111 );
      { ClearNtb, Dtb } = 5'b1_XXXX; @(negedge Clocktb) expectedResult ( 4'bxxxx ); @(negedge Clocktb) expectedResultN ( 4'bxxxx );
      { ClearNtb, Dtb } = 5'b1_1101; @(negedge Clocktb) expectedResult ( 4'b1101 ); @(negedge Clocktb) expectedResultN ( 4'b0010 );
      { ClearNtb, Dtb } = 5'b1_1010; @(negedge Clocktb) expectedResult ( 4'b1010 ); @(negedge Clocktb) expectedResultN ( 4'b0101 );
      { ClearNtb, Dtb } = 5'b0_XXXX; @(negedge Clocktb) expectedResult ( 4'b0000 ); @(negedge Clocktb) expectedResultN ( 4'b1111 );
      { ClearNtb, Dtb } = 5'b1_XXXX; @(negedge Clocktb) expectedResult ( 4'bxxxx ); @(negedge Clocktb) expectedResultN ( 4'bxxxx );
      { ClearNtb, Dtb } = 5'b1_1010; @(negedge Clocktb) expectedResult ( 4'b1010 ); @(negedge Clocktb) expectedResultN ( 4'b0101 );
      { ClearNtb, Dtb } = 5'b1_1101; @(negedge Clocktb) expectedResult ( 4'b1101 ); @(negedge Clocktb) expectedResultN ( 4'b0010 );
      $display ( "*****Test PASSED*****" );
      $finish;
    end
endmodule
```

This is the Verilog code for the 4-bit register which uses D flip-flops testbench.

c.



These are my VCS waveforms for the 4-bit DFF register. The blocks of red indicate delay or if the output is a "don't care (X) value".

d.

```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Sep 21 23:24 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab4/regDFF_tb/inter.vpd' was opened successfully.
time=     0.0ns      Dtb = xxxx      ClrNtb = 1      Qtb = xxxx      QNtb = xxxx
time=    25.0ns      Dtb = xxxx      ClrNtb = 0      Qtb = 0000      QNtb = 1111
time=    45.0ns      Dtb = xxxx      ClrNtb = 1      Qtb = 0000      QNtb = 1111
time=    50.0ns      Dtb = xxxx      ClrNtb = 1      Qtb = xxxx      QNtb = xxxx
time=    65.0ns      Dtb = 1101      ClrNtb = 1      Qtb = xxxx      QNtb = xxxx
time=    70.0ns      Dtb = 1101      ClrNtb = 1      Qtb = 1101      QNtb = 0010
time=    85.0ns      Dtb = 1010      ClrNtb = 1      Qtb = 1101      QNtb = 0010
time=    90.0ns      Dtb = 1010      ClrNtb = 1      Qtb = 1010      QNtb = 0101
time=   105.0ns      Dtb = xxxx      ClrNtb = 0      Qtb = 0000      QNtb = 1111
time=   125.0ns      Dtb = xxxx      ClrNtb = 1      Qtb = 0000      QNtb = 1111
time=   130.0ns      Dtb = xxxx      ClrNtb = 1      Qtb = xxxx      QNtb = xxxx
time=   145.0ns      Dtb = 1010      ClrNtb = 1      Qtb = xxxx      QNtb = xxxx
time=   150.0ns      Dtb = 1010      ClrNtb = 1      Qtb = 1010      QNtb = 0101
time=   165.0ns      Dtb = 1101      ClrNtb = 1      Qtb = 1010      QNtb = 0101
time=   170.0ns      Dtb = 1101      ClrNtb = 1      Qtb = 1101      QNtb = 0010
*****Test PASSED*****
$finish called from file "regDFF_tb.v", line 62.
$finish at simulation time   185.0ns
Simulation complete, time is 185000 ps.
```

This is my VCS simulation console for the 4-bit DFF register.

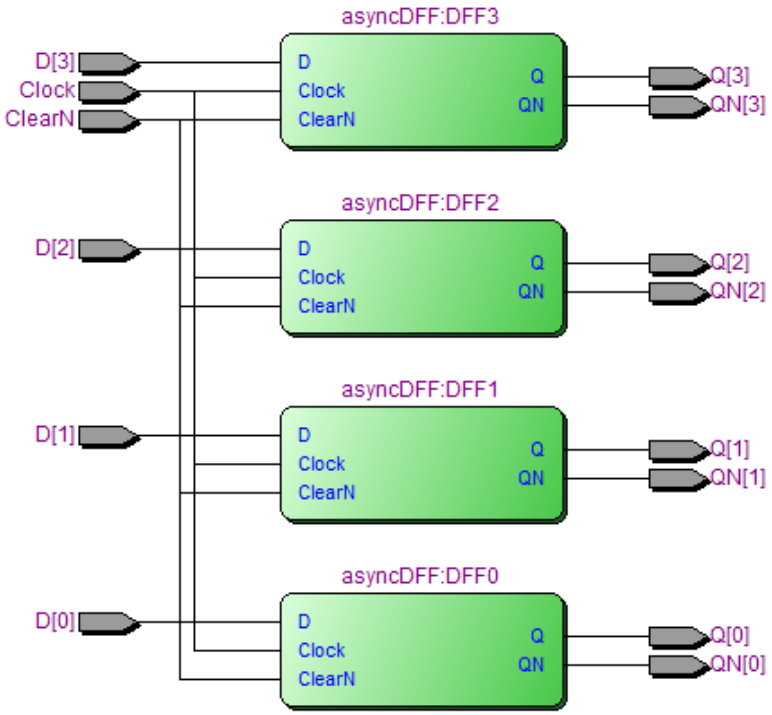
Experiment 3

a.

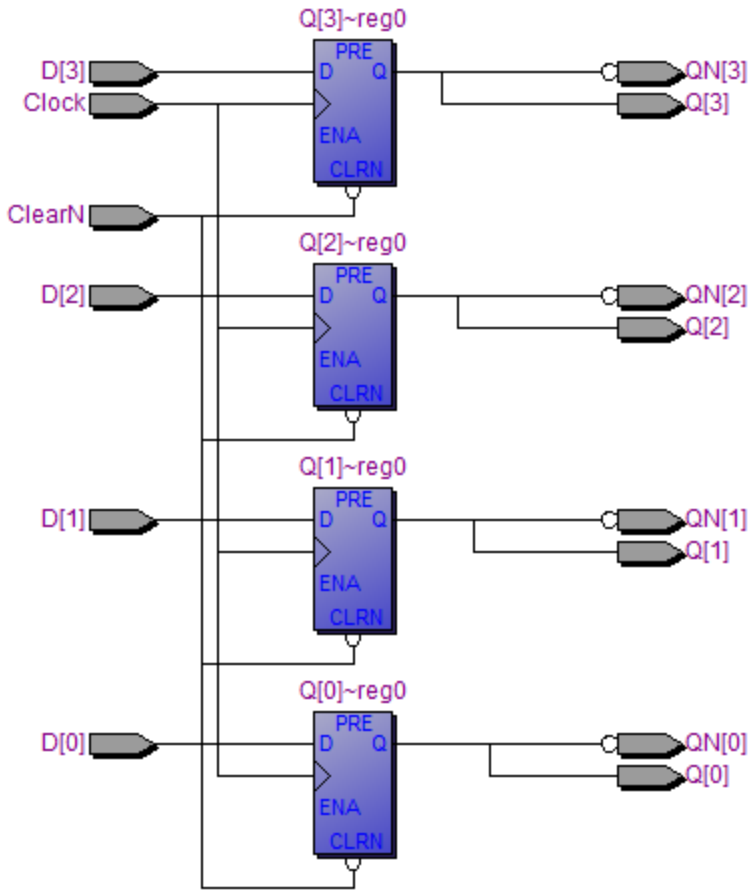
```
1   `timescale 1 ns / 100 ps
2
3   module regBehaveDFF (D, Clock, ClearN, Q, QN);
4      input [3:0] D;
5      input Clock, ClearN;
6      output reg [3:0] Q;
7      output [3:0] QN;
8
9      always @ (posedge Clock, negedge ClearN)
10        begin
11           if (~ClearN)
12              Q = 4'b0000;
13           else
14              Q = D;
15        end
16        assign QN = ~Q;
17
18  endmodule
19
```

This is my behavioral Verilog code for instantiating the DFF from experiment 1 into a 4-bit register which uses D flip-flops.

b.



This is the RTL View of the structural 4-bit DFF register.



This is the RTL View of the behavioral 4-bit DFF register.

c.

From the RTL view of both behavioral and structural, both are quite similar in the lower hierarchy. For example, by clicking on each "green" block of the structural RTL view, the instantiation of the DFF is the same as the behavioral RTL view. The reason for the "green" blocks for the structural RTL view is due to the instantiation of the DFF module. Since there is no instantiation for the behavioral model, the D flip-flops can easily be seen in the behavioral view because the 4 bits are designated to each individual D flip-flop within the code. The "green" blocks of the structural code simply just show instantiation of another module; whereas, the behavioral code would show the 4 bits spread to each D flip-flop.

Experiment 4

a.

```
`timescale 1 ns / 100 ps

module modCount (inclk, rst, ud, load, data, count, segments1, segments2);
    input inclk, rst, ud, load;
    input [4:0] data;
    output reg [4:0] count = 0;
    output reg [6:0] segments1, segments2;
    wire outclk;

    onehertz U0(                                        // calls to the onehertz module
        .clk_50mhz (inclk),
        .clk_1hz (outclk)
    );

    always @ (posedge outclk, posedge rst)
        begin
            if (rst == 1)                               // dont count if reset
                count = 0;
            else if (load)                              // load input into output
                count = data;
            else
                if (ud == 1)                            // count up
                    if (count == 25)
                        begin
                            count = 0;                  // counter is 0 when reached 25
                        end
                    else
                        begin
                            count = count + 1;      // increment counter
                        end
                else                                            // count down
                    begin
                        if (count == 0)             // if there is not count...
                            begin
                                count = 25;             // counter starts at 25
                            end
                        else
                            begin
                                count = count - 1;  // decrement counter
                            end
                    end
        end

        // 7-segment display
        always @ (count) begin
            case (count)
                0 : begin
                        segments1 = 7'b0000001;
                        segments2 = 7'b0000001;
                    end
                1 : begin
                        segments1 = 7'b1001111;
                        segments2 = 7'b0000001;
                    end
                2 : begin
                        segments1 = 7'b0010010;
                        segments2 = 7'b0000001;
```

```
58                  end
59              3 : begin
60                      segments1 = 7'b0000110;
61                      segments2 = 7'b0000001;
62                  end
63              4 : begin
64                      segments1 = 7'b1001100;
65                      segments2 = 7'b0000001;
66                  end
67              5 : begin
68                      segments1 = 7'b0100100;
69                      segments2 = 7'b0000001;
70                  end
71              6 : begin
72                      segments1 = 7'b0100000;
73                      segments2 = 7'b0000001;
74                  end
75              7 : begin
76                      segments1 = 7'b0001111;
77                      segments2 = 7'b0000001;
78                  end
79              8 : begin
80                      segments1 = 7'b0000000;
81                      segments2 = 7'b0000001;
82                  end
83              9 : begin
84                      segments1 = 7'b0000100;
85                      segments2 = 7'b0000001;
86                  end
87              10 : begin
88                      segments1 = 7'b0000001;
89                      segments2 = 7'b1001111;
90                   end
91              11 : begin
92                      segments1 = 7'b1001111;
93                      segments2 = 7'b1001111;
94                   end
95              12 : begin
96                      segments1 = 7'b0010010;
97                      segments2 = 7'b1001111;
98                   end
99              13 : begin
100                     segments1 = 7'b0000110;
101                     segments2 = 7'b1001111;
102                  end
103             14 : begin
104                     segments1 = 7'b1001100;
105                     segments2 = 7'b1001111;
106                  end
107             15 : begin
108                     segments1 = 7'b0100100;
109                     segments2 = 7'b1001111;
110                  end
111             16 : begin
112                     segments1 = 7'b0100000;
113                     segments2 = 7'b1001111;
114                  end
```

```
                17 : begin
                        segments1 = 7'b0001111;
                        segments2 = 7'b1001111;
                     end
                18 : begin
                        segments1 = 7'b0000000;
                        segments2 = 7'b1001111;
                     end
                19 : begin
                        segments1 = 7'b0000100;
                        segments2 = 7'b1001111;
                     end
                20 : begin
                        segments1 = 7'b0000001;
                        segments2 = 7'b0010010;
                     end
                21 : begin
                        segments1 = 7'b1001111;
                        segments2 = 7'b0010010;
                     end
                22 : begin
                        segments1 = 7'b0010010;
                        segments2 = 7'b0010010;
                     end
                23 : begin
                        segments1 = 7'b0000110;
                        segments2 = 7'b0010010;
                     end
                24 : begin
                        segments1 = 7'b1001100;
                        segments2 = 7'b0010010;
                     end
                25 : begin
                        segments1 = 7'b0100100;
                        segments2 = 7'b0010010;
                     end
                26 : begin
                        segments1 = 7'b0100000;
                        segments2 = 7'b0010010;
                     end
                default:
                        begin
                            segments1 = 7'b1111111;
                            segments2 = 7'b1111111;
                        end
            endcase
        end

endmodule

module onehertz(clk_50mhz, clk_1hz);
    input clk_50mhz;
    output clk_1hz;
    reg clk_1hz;
    reg [24:0] count;
    always @ (posedge clk_50mhz)
        begin
```

```
            if(count == 24999999) begin
                count <= 0;
                $dumpfile("f.vcd");
                clk_1hz <= ~clk_1hz;
            end
            else begin
                count <= count + 1;
            end
        end
    endmodule
```

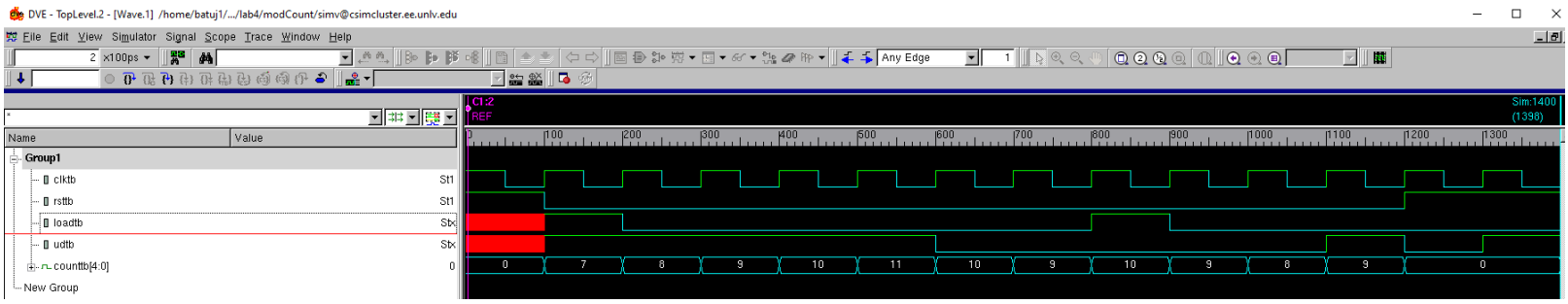
This is my Verilog code for the 5-bit mod-25 up/down counter including a clock divider and a 7 segment display.

b.

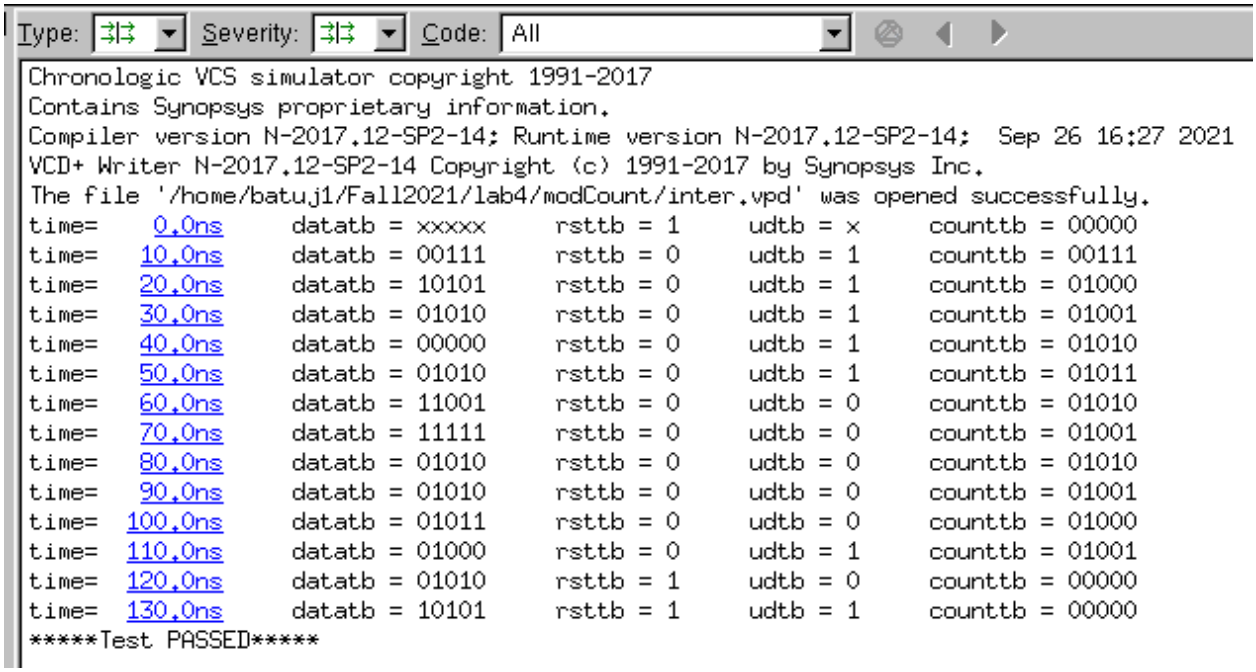
```
`timescale 1 ns / 100 ps

// modCount Testbench
module modCount_tb;
  reg clktb = 1'b1;
  reg rsttb = 1'b1;
  reg udtb;
  reg loadtb;
  reg [4:0] datatb;
  wire [4:0] counttb;
  wire [6:0] seg1, seg2;
  `define PERIOD 10

  always
    #(`PERIOD/2) clktb = ~clktb;

  modCount U0 (
    .clk (clktb),
    .rst (rsttb),
    .ud (udtb),
    .load (loadtb),
    .data (datatb),
    .count (counttb),
    .segments1 (seg1),
    .segments2 (seg2)
  );

  initial begin
    $timeformat (-9, 1, "ns", 9 );
    $monitor ( "time=%t     datatb = %b     rsttb = %b     udtb = %b     counttb = %b", $time,   datatb,   rsttb,   udtb,   counttb);
    #(`PERIOD * 100)
    $display ( "TESTING TIMEOUT" );
    $finish;
  end

task expectedResult (input [4:0] expected);
  if (counttb !== expected)
    begin
      $display ( "counttb=%b, but expected value is %b", counttb, expected);
      $display ( "Test Failed" );
      $finish;
    end
endtask

initial
  begin
    @(posedge clktb)
    { rsttb, loadtb, udtb, datatb } = 8'b0_1_1_00111; @(posedge clktb) expectedResult ( 5'b00111 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_1_10101; @(posedge clktb) expectedResult ( 5'b01000 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_1_01010; @(posedge clktb) expectedResult ( 5'b01001 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_1_00000; @(posedge clktb) expectedResult ( 5'b01010 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_1_01010; @(posedge clktb) expectedResult ( 5'b01011 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_0_11001; @(posedge clktb) expectedResult ( 5'b01010 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_0_11111; @(posedge clktb) expectedResult ( 5'b01001 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_1_0_01010; @(posedge clktb) expectedResult ( 5'b01010 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_0_01010; @(posedge clktb) expectedResult ( 5'b01001 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_0_01011; @(posedge clktb) expectedResult ( 5'b01000 );
    { rsttb, loadtb, udtb, datatb } = 8'b0_0_1_01000; @(posedge clktb) expectedResult ( 5'b01001 );
    { rsttb, loadtb, udtb, datatb } = 8'b1_0_0_01010; @(posedge clktb) expectedResult ( 5'b00000 );
    { rsttb, loadtb, udtb, datatb } = 8'b1_0_1_10101; @(posedge clktb) expectedResult ( 5'b00000 );
    $display ( "*****Test PASSED*****" );
    $finish;
  end
endmodule
```

This is my Verilog code for the 5-bit mod-25 up/down counter testbench.

c.

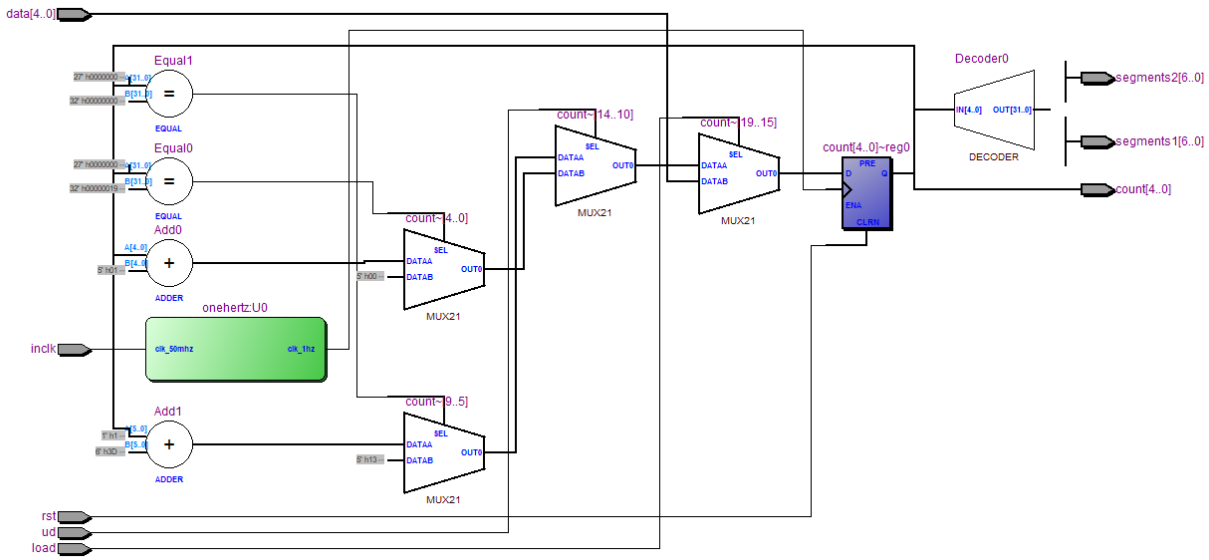


These are my VCS waveforms for the 5-bit mod-25 up/down counter.



This is my VCS console for the 5-bit mod-25 up/down counter.

d.



This is my RTL view for the 5-bit mod-25 up/down counter including a clock divider and a 7 segment display.

e.

Fitter Status	Successful - Sun Sep 26 16:58:40 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	modCount
Top-level Entity Name	modCount
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	95 / 33,216 (< 1 %)
Total combinational functions	95 / 33,216 (< 1 %)
Dedicated logic registers	31 / 33,216 (< 1 %)
Total registers	31
Total pins	28 / 475 (6 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

This is my compilation report of the total utilization, combinational ALUTs, and dedicated logic registers for the 5-bit mod-25 up/down counter including a clock divider and a 7 segment display.

f. https://drive.google.com/file/d/1hoacqdml03irNyL9vXW9phvNSH5imaPM/view?usp=sharing

This is the link to my video delivery of 4(b). The video will also be included within the batuj1_postlab_4.zip file.

4. Answers to questions

Question 1:

Regarding *Logic Utilization* from the *Quartus Compilation Report*, the *ALUT* is utilized to show half-ALMS (half-adaptive logic modules). Half-ALMS are half-adaptive logic modules, meaning that the ALUT has 2 combinational logic look up tables (LUTs) and 2 registers becoming one total adaptive look up table (ALUT). ALUTs show the actual number of partial or final half-ALMs in the design after it is placed. In general, the ALUT reveals the combinational logic needed for each register or flip-flop to form. The *Dedicated Logic Registers* are the two registers within the ALM. The ALM is the essential building block of supported device families and is made to maximize the performance of registers as well as resource usage. Each adaptive logic module can hold up to 8 inputs and 8 outputs. The reason for the ALM is to show the logic registers used to complete the total logic elements within the Verilog code design.

Question 2:

Clock-gating is when the clock is directed to each flip-flop within the design to reveal that each flip-flop is clocked continuously rather than individually. The purpose of clock-gating is to reduce unnecessary clocking to each register; thus, registers do not need to be clocked if the input data does not change. Clock-gating is inserted in two ways: local clock-gating and global clock-gating. Local clock-gating has the logic synthesizer find and utilize local gating opportunities and the RTL code would have the clock-gating cell instantiated within it. Global clock-gating specifically specifies the clock gating within the RTL code and follows the local clock-gating where the clock-gating cell is instantiated within it. Overall, clock-gating saves unnecessary clocking to each register by continuous clocking rather than individual clocks assigned to each individual register.

Question 3

The key difference between tasks and functions is that a function is to return a single value in regards to processing the input; whereas, a task can return multiple values and return these values using the output arguments. Tasks can also contain timing simulations where functions cannot utilize any timing. Tasks and functions are used when an operation is continuously repeated throughout the Verilog code. Instead of rewriting that same code, the task or function operation can be used. This reduces copy and paste errors and permits faster development time as well as making a code cleaner and easier to read.

5. Conclusions & Summary

This lab was one of the easier labs, and I was able to go more in depth with my knowledge of flip-flops and registers. In the CPE 200 lab, I had a hard time implementing the clock within my code; therefore, the provided clock divider definitely helped for the 4th experiment. The main issues within this lab occurred with experiment 4. When I originally coded my design, I did not take into account the clock behaving differently than my own interpretation of it; therefore, I used the clock divider. Although I had already completed the testbench, VCS console, and the VCS waveforms of my code, the clock divider gave several issues within my assignment. Without the clock divider, I would not be able to properly output my code into the DE2 board and make it properly output. I hope this issue does not continue into more labs, and I will be able to fix it for the future.