Class:	CPE-300L	Semester:	Spring 2020

Points		Document author:	Brysen Kokubun
		Author's email:	Brysenkokubun96@gmail.com
		Document topic:	Postlab 4

Instructor's comments:

1. Introduction / Theory of Operation

- For this lab experiment we implemented flip-flop, registers, counters, PRNG, and ALU models. These models were implemented synchronously and asynchronously. We verified our models by using the RTL viewer, Waveforms and DE2 board implementations.

a) Provide basic info about D and JK flip-flops

- The D flip flop has a single input therefore it is dependent on that input, whatever the input is, will be the output on a given clock cycle. The JK flip flop has 2 inputs, either J or K, 00 means no change, 01 means Reset, 10 means Set, and 11 means toggle.

b) Write few sentences: explain how waveforms are used to check the correctness of a circuit

- The waveforms are used to check the correctness of a circuit because it allows us to visually see how data is being sent to each component and what the signals are at given times. So if the person testing the circuit knows how about the signals and how they should act for certain outputs then the waveform will help clarify the circuit.

2. Prelab Report

- Attached

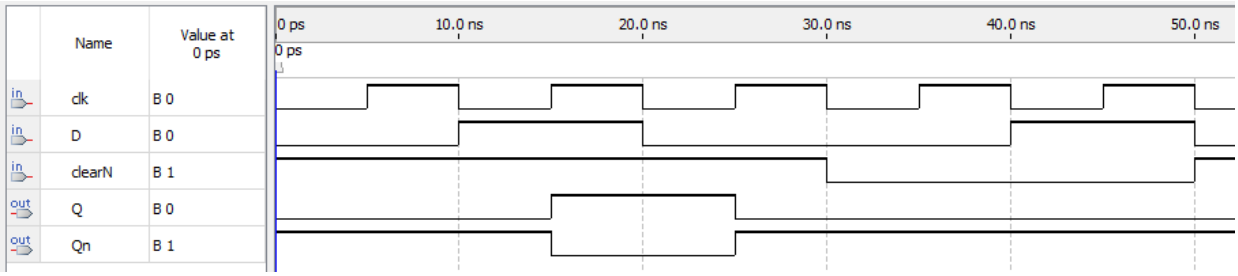
3. Experiment Results

Experiment #1: Design D flip-flop with asynchronous low clear

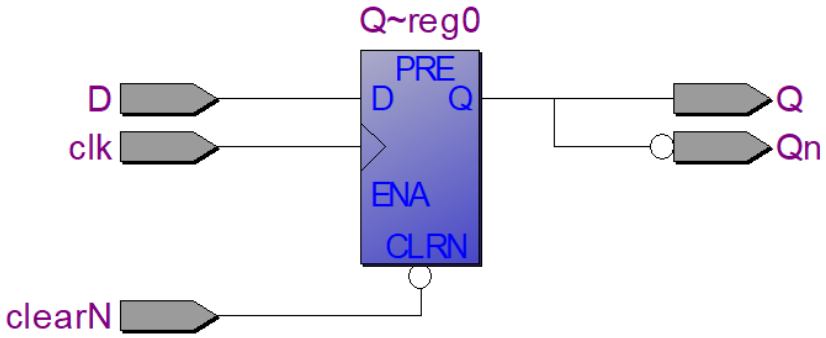
a. Verilog Code

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

b. Waveform



c. RTL View

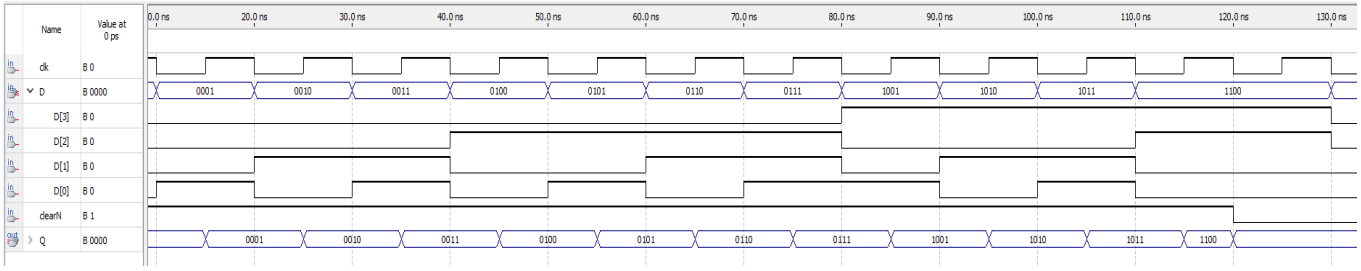


Experiment #2: 4-bit register of D flip-flops

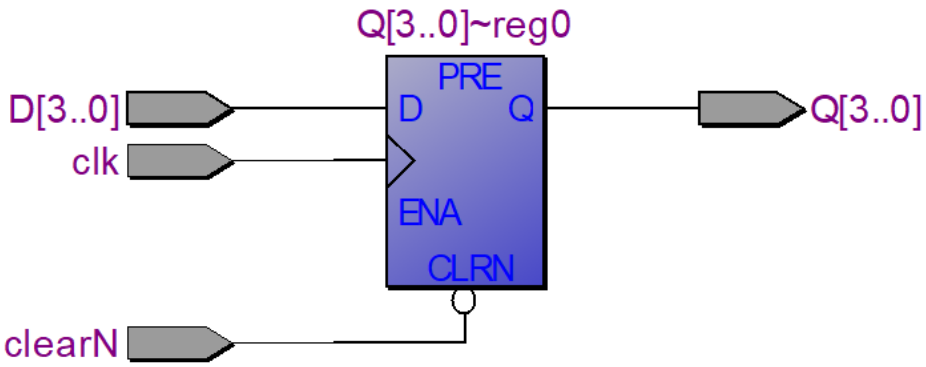
a. Verilog Code

```
1   module D_reg (clk, clearN, D, Q);
2      input clk, clearN;
3      input [3:0] D;
4      output [3:0] Q;
5      reg [3:0] Q;
6
7      always @(posedge clk, negedge clearN)
8      begin
9         if (~clearN)
10           Q = 4'b0000;
11        else
12           Q = D;
13     end
14
15  endmodule
```

b. Waveform



c. RTL View



Experiment #3: 4-bit mod-13 counter

a. Verilog Code

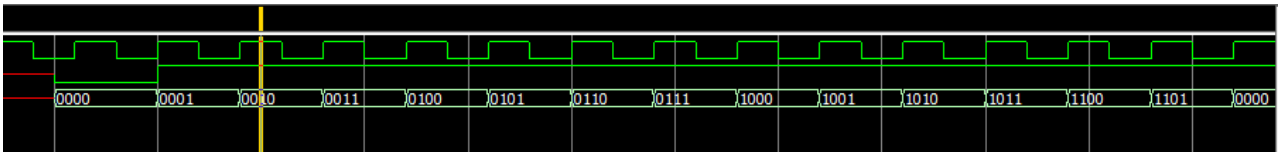
```
module Counter (clk, clearN, Q, display1, display2, clk_50mhz, clk_1hz);
    input clk, clearN;
    output[3:0] Q;
    reg[3:0] Q;
    reg[3:0] count;
    output[6:0] display1, display2;
    reg[6:0] display1, display2;

    input clk_50mhz;
    output clk_1hz;
    reg clk_1hz;
    reg [24:0] c_meg;

    always @ (posedge clk_50mhz)
    begin
    if(c_meg == 24999999)
    begin
    c_meg <= 0;
     $dumpfile("f.vcd");
     clk_1hz <= ~clk_1hz;
     end
     else begin
     c_meg <= c_meg + 1;
     end
    end

    always @(posedge clk_1hz, negedge clearN)
    begin
        if (~clearN)
            Q = 4'b0000;
        else
          begin
            count = count+1;
            case(count)
              4'b1101: count = 4'b0000;
              default: Q = count;
             endcase
           end
    end
    always@(*)
    case(Q)
        4'b0000: begin
                    display1 = 7'b1000000; // 0
                    display2 = 7'b1000000;
                    end
        4'b0001: begin
                    display1 = 7'b1111001; // 1
                    display2 = 7'b1000000;
                    end
        4'b0010: begin
                    display1 = 7'b0100100; // 2
                    display2 = 7'b1000000;
                    end
        4'b0011: begin
                    display1 = 7'b0110000; // 3
                    display2 = 7'b1000000;
                    end
        4'b0100: begin
                    display1 = 7'b0011001; // 4
                    display2 = 7'b1000000;
                    end
        4'b0101: begin
                    display1 = 7'b0010010; // 5
                    display2 = 7'b1000000;
                    end
        4'b0110: begin
                    display1 = 7'b0000010; // 6
                    display2 = 7'b1000000;
                    end
        4'b0111: begin
                    display1 = 7'b1111000; // 7
                    display2 = 7'b1000000;
                    end
        4'b1000: begin
                    display1 = 7'b0000000; // 8
                    display2 = 7'b1000000;
                    end
        4'b1001: begin
                    display1 = 7'b0010000; // 9
                    display2 = 7'b1000000;
                    end
```

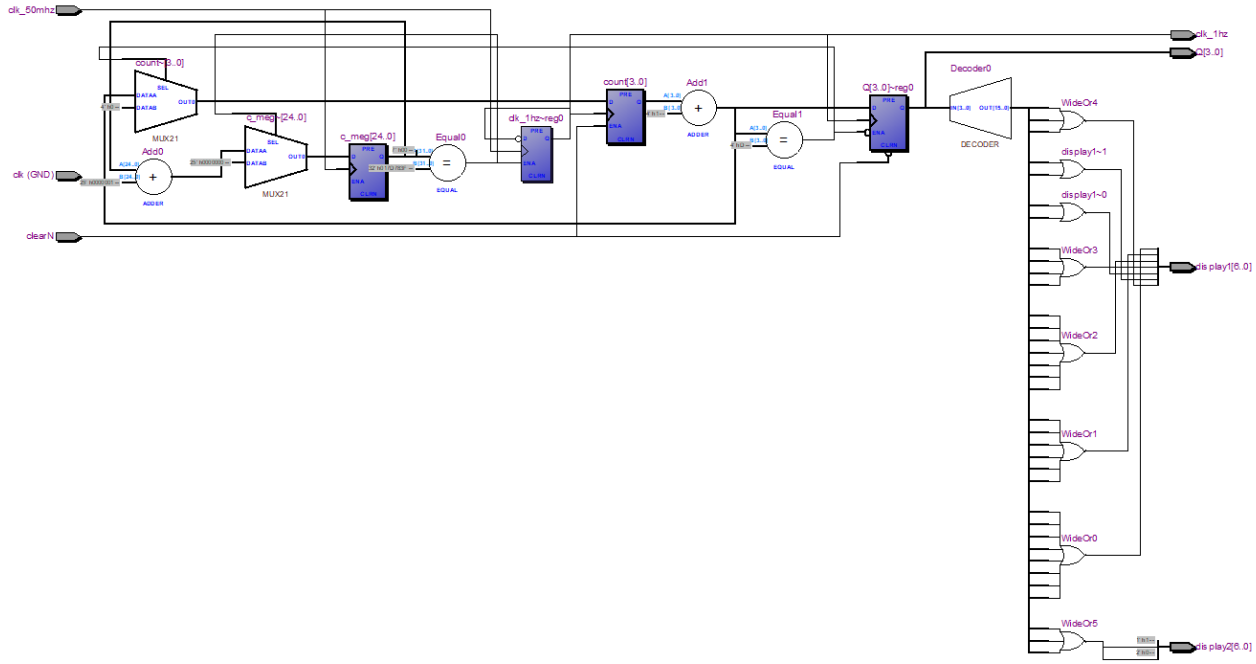
```
        4'b1010: begin
                    display1 = 7'b1000000; // 10
                    display2 = 7'b1111001;
                    end
        4'b1011: begin
                    display1 = 7'b1111001; // 11
                    display2 = 7'b1111001;
                    end
        4'b1100: begin
                    display1 = 7'b0100100; // 12
                    display2 = 7'b1111001;
                    end
        4'b1101: begin
                    display1 = 7'b1000000; // When 13
                    display2 = 7'b1000000;
                    end
        default: begin
                    display1 = 7'b1000000; // Default 0
                    display2 = 7'b1000000;
                    end
    endcase

endmodule
```

b. Testbench Code

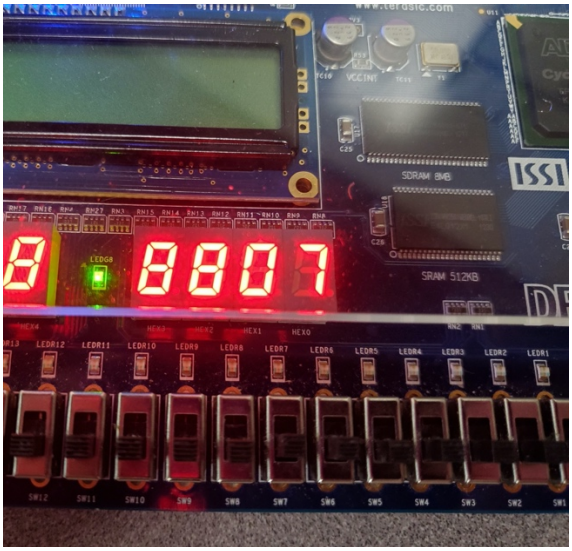
```
module CounterTB;
  reg clk, clearN;
  wire[3:0] Q;
  Counter C1(clk, clearN, Q);

  initial begin
    clk = 1'b0;
    clearN = 1'b0;
    #5;
    clearN = 1'b1;
    #5;
    end

   always
    begin
      #1;
      clk = ~clk;
    end

endmodule
```

c. Simulation Waveform & RTL View




clk_50mhz
clk (GND)
clearN
count[3..0]
MUX21
Add0
ADDER
c_meg[24..0]
MUX21
c_meg[24..0]
Equal0
EQUAL
clk_1hz~reg0
count[3..0]
Add1
ADDER
Equal1
EQUAL
Q[3..0]~reg0
Decoder0
DECODER
clk_1hz
Q[3..0]
WideOr4
display1~1
display1~0
WideOr3
WideOr2
WideOr1
WideOr0
WideOr5
display1[6..0]
display2[6..0]

d. DE2 board

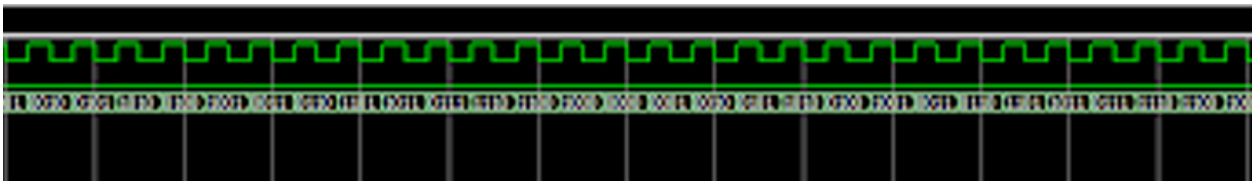

SDRAM 8MB
SRAM 512KB
LEDG8
LEDR17
LEDR16
LEDR15
LEDR14
LEDR13
LEDR12
LEDR11
LEDR10
LEDR9
LEDR8
LEDR7
LEDR6
LEDR5
LEDR4
LEDR3
LEDR2
LEDR1
SW17
SW16
SW15
SW14
SW13
SW12
SW11
SW10
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2
SW1
SW0

Experiment #4: 4-bit Pseudo Random Number Generator (PRNG) using linear feedback shift register

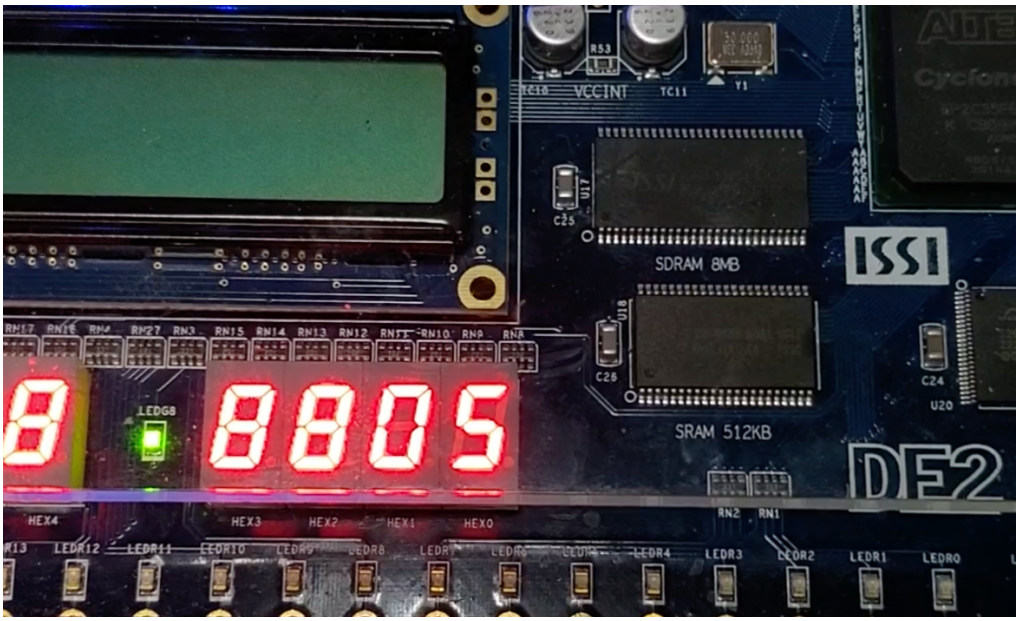
a. Verilog Code

```
1   module Rando (clk, clear,rdm);
2       input clk, clear;
3       output reg [3:0] rdm;
4       wire num;
5
6       assign num = ~(rdm[3] ^ rdm[0]);
7
8       always @(posedge clk, posedge clear)
9         begin
10          if (clear)
11             rdm = 4'b0;
12          else
13             rdm = {rdm[2:0],num};
14        end
15      endmodule
```

b. Simulation waveforms



c. DE2 board



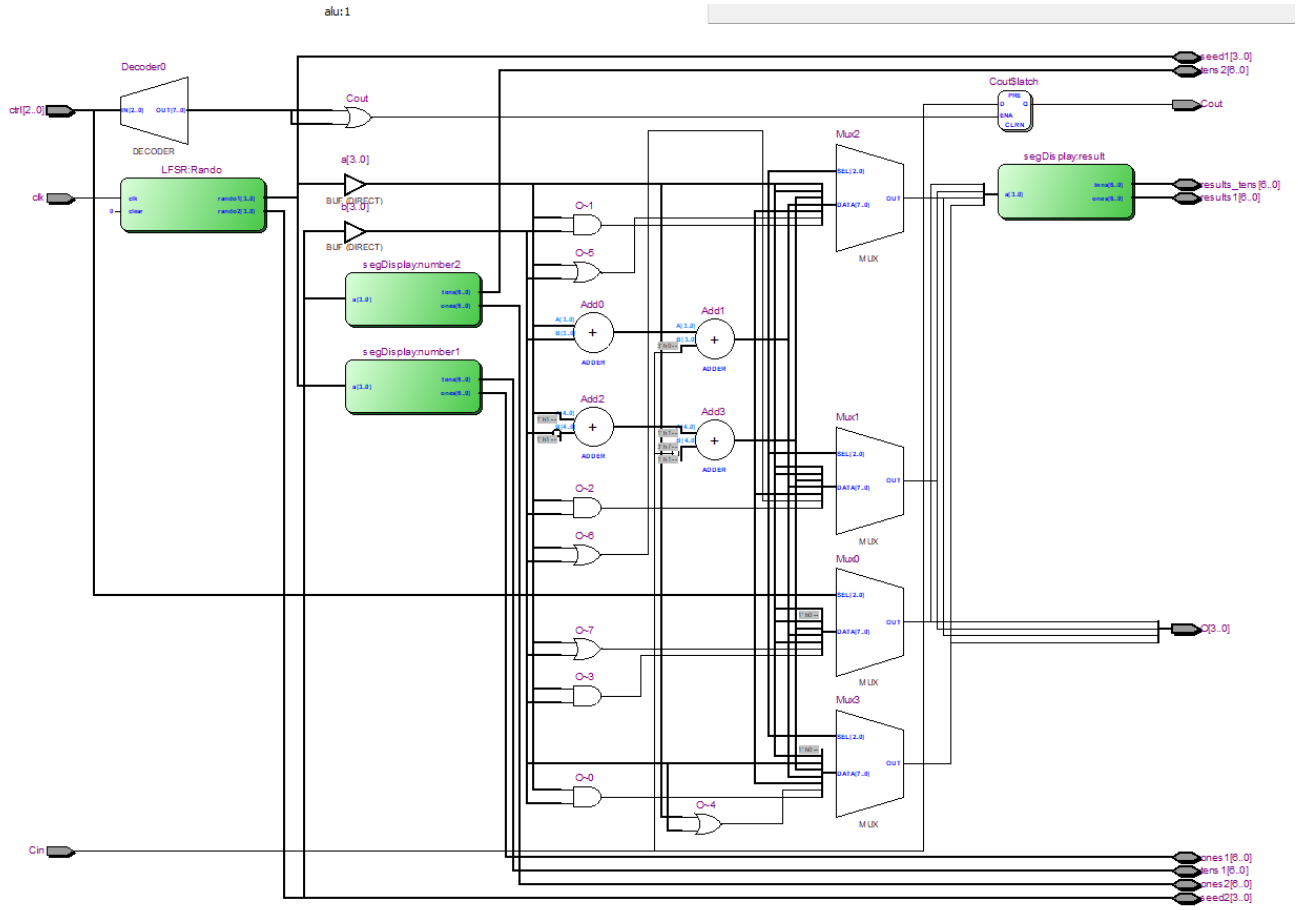
Experiment #5: Use LFSR from Exp #3, on ALU of Lab#3 Exp #4

a. Verilog Code & RTL View

```
module alu(seed1, seed2, Cin, clk, O, Cout, ctrl, results_tens, results1,  tens1, ones1, tens2, ones2);
   input Cin;
   input [2:0] ctrl;
   input clk;
   inout [3:0] seed1, seed2;
   inout [6:0] results_tens, results1, tens1, ones1, tens2, ones2;
   output [3:0] O;
   output Cout;
   wire [3:0] a, b;
   reg [3:0] O;
   reg Cout;

   segDisplay result(O, results_tens, results1);
   segDisplay number1(seed1, tens1, ones1);
   segDisplay number2(seed2, tens2, ones2);

   LFSR Rando(seed1, seed2, clk, clear);
   assign a = seed1;
   assign b = seed2;

   always @(*)
   begin
      case (ctrl)
         3'b000:
         begin
            O = a & b;
         end
         3'b001:
         begin
            O = a | b;
         end
         3'b010:
         begin
            O = {a[0], b[3:1]};
         end
         3'b011:
         begin
            O = a + b + Cin;
            Cout = Cin;
         end
         3'b100:
         begin
            O = a - b - Cin;
            Cout = Cin;
         end
         3'b101:
         begin
            O = {a[3:1], b[0]};
         end
         3'b110:
         begin
            O = a >> 1;
         end
         3'b111:
         begin
            O = a << 1;
         end
      endcase
   end
endmodule

module LFSR (rando1, rando2, clk, clear, tens1, ones1, tens2, ones2);
   output reg [3:0] rando1, rando2;
   input clk, clear;
   inout [6:0] tens1, ones1, tens2, ones2;
   wire slowclk;
   wire num1, num2;
   assign num1 = ~(rando1[3] ^ rando1[0]);
   assign num2 = ~(rando2[2] ^ rando2[0]);

   segDisplay number1(rando1, tens1, ones1);
   segDisplay number2(rando2, tens2, ones2);
   onehertz slow(clk, slowclk);

always @(posedge slowclk, posedge clear)
   begin
      if (clear) begin
        rando1 = 4'b0;
        rando2 = 4'b0; end
      else begin
        rando1 = {rando1[2:0],num1};
        rando2 = {rando2[2:0],num2}; end
   end
endmodule

module segDisplay(a, tens, ones);
 input [3:0]a;
 output reg [6:0]ones;
 output reg [6:0]tens;
```

```
output reg [6:0]tens;
always@(*)
case (a)
4'b0000: begin tens = 7'b1000000; //0
         ones = 7'b1000000; //0
         end
4'b0001: begin tens = 7'b1000000; //0
         ones = 7'b1111001; //1
         end
4'b0010: begin tens = 7'b1000000; //0
         ones = 7'b0100100; //2
         end
4'b0011: begin tens = 7'b1000000; //0
         ones = 7'b0110000; //3
         end
4'b0100: begin tens = 7'b1000000; //0
         ones = 7'b0011001; //4
         end
4'b0101: begin tens = 7'b1000000; //0
         ones = 7'b0010010; //5
         end
4'b0110: begin tens = 7'b1000000; //0
         ones = 7'b0000010; //6
         end
4'b0111: begin tens = 7'b1000000; //0
         ones = 7'b1111000; //7
         end
4'b1000: begin tens = 7'b1000000; //0
         ones = 7'b0000000; //8
         end
4'b1001: begin tens = 7'b1000000; //0
         ones = 7'b0011000; //9
         end
4'b1010: begin tens = 7'b1111001; //1
         ones = 7'b1000000; //0
         end
4'b1011: begin tens = 7'b1111001; //1
         ones = 7'b1111001; //1
         end
4'b1100: begin tens = 7'b1111001; //1
         ones = 7'b0100100; //2
         end
4'b1101: begin tens = 7'b1111001; //1
         ones = 7'b0110000; //3
         end
4'b1110: begin tens = 7'b1111001; //1
          ones = 7'b0011001; //4
          end
4'b1111: begin tens = 7'b1111001; //1
          ones = 7'b0010010; //5
          end

default: begin tens = 7'b1111111; // turn off
         ones = 7'b1111111; // turn off
         end
 endcase
 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
```

RTL View



4. Questions

1. What is the meaning of component count from Quartus compilation report?

- The meaning of component count from Quartus compilation report is the number of components that Quartus will create from the given Verilog code after compilation.

2. What is the advantage of using asynchronous Set and Reset in case of D flip-flop?

- The advantage of using asynchronous Set and Reset for D flip-flop is because the Reset/Set of the flip flop does not have to wait for the next rising/falling edge of the clock, it can happen at any time. Also, for synchronous implementation, the signal must stay low or high in order for the flip flop to work, and then if a reset or set is to occur then the signal must sent at the perfect time so it does not miss the edge of the clock, making it trickier to implement.

3. Regarding Logic Utilization from the Quartus Compilation Report – what are ALUT and Dedicated Logic Registers?

- ALUT and Dedicated Logic Registers within the Logic Utilization of the Compilation Report shows how “Full” the device is and how much of its resources are being used. Therefore, the ALUT indicates the size of the function as well as amount of usages. The Logic Registers shows the amount of registers are being used in order to execute the given code.

5. Conclusions

- To conclude this lab, we learned how to take the D flip-flop and use it as a component in order to make a 4-bit register, counter, PRNG, and ALU. We verified our Verilog code by simulating it with waveforms and checking the RTL viewer. Some issues that we came across was, creating the "Randomness" for the PRNG. But after looking at some references provided by the lab experiment, we were able to create the Randomness. Overall, we gained a better understanding of the basic Verilog modules, how to create testbenches for them, and how to implement them on the DE2 board.