B EE 427 PA #2
Due: 02/13/2023 3:30pm

Note: There are two parts in this PA. Create a Quartus project file for each part. You should submit your Verilog code (design source code and test bench code) to Canvas. In addition, you should show me your demo using the DE1-SoC board. There is no report for this PA.

Background

In Verilog programming, we can describe a variable-size counter by using a parameter declaration. An example of an n-bit counter is shown below:

```
module  counter (Clock, Reset_n, Q);
    parameter n = 4;

    input Clock, Reset_n;
    output [n-1:0] Q;
    reg [n-1:0] Q;

    always @(posedge Clock or negedge Reset_n)
    begin
        if (!Reset_n)
            Q <= 1'd0;
        else
            Q <= Q + 1'b1;
    end
endmodule
```

The parameter n specifies the number of bits in the counter. When instantiating (including) this counter in a higher-level module, a particular value of the parameter n can be specified by using the defparam keyword or #(). By using parameters, we can instantiate counters of different sizes in a logic circuit, without having to create a new module for each counter.

Part 1

Create a modulo-k counter by modifying the forementioned code to contain an additional parameter. The counter should count from 0 to $k-1$. When the counter reaches the value $k-1$, then the next counter value should be 0. Include an output from the counter called *rollover* and set this output to 1 in the clock cycle where the count value is equal to $k-1$. After that, the *rollover* value should be reset to 0.

Perform the following steps:

1. Create a new Quartus project using the system builder program. You'll use switches, buttons, and LEDs.

2. Write Verilog code that specifies the circuit for k = 20, and an appropriate value of n. Your circuit should use SW9 as a reset and KEY0 as a manual clock input. The contents of the counter should be displayed on the red lights LEDR. Also display the *rollover* signal on one of the LEDR lights.
3. Include the Verilog file in your project and compile the circuit.
4. Simulate the designed circuit to verify its functionality (submit your test bench file together).
5. Verify that your circuit works correctly by observing the lights.

Part 2

Design and implement a circuit that acts as a real-time clock. It should display the minutes (from 0 to 59) on HEX5-4, the seconds (from 0 to 59) on HEX3-2, and hundredths of a second (from 0 to 99) on HEX1-0. Use the switches SW7-0 to preset the minute part of the time (SW7-4 for HEX5 and SW3-0 for HEX4) when SW8 is toggled up. Stop counting whenever SW9 is toggled up and continue when SW9 is toggled down.

The DE1-SoC board has four 50 MHz (20ns period) clocks (you can use one of them). For Part 2, you should include a clock divider module. Modify the following example code to get an appropriate clock frequency.

```
// This module is used to lower the clock frequency (from 50MHz to 0.25Hz) or increase the clock
// period (from 1/(50 x 10^6) seconds to 4 seconds)
module clock_divider (
    input          i_clock,
    input          reset_n,
    output reg     o_clock
);

    reg [27:0] cnt;

    always @(posedge i_clock) begin
        if (~reset_n) begin
            cnt <= 0;
        end else begin
            if (cnt == 100000000) cnt <= 0;
            else cnt <= cnt + 1; // the cnt value increases by 1 whenever i_clock changes from 0 to 1
        end
    end

    always @(posedge i_clock) begin
        if (~reset_n) begin
            o_clock <= 0;
        end else begin
            if (cnt == 100000000) begin
                o_clock <= ~o_clock; // o_clock is flipped when the cnt value is 100000000
            end
        end
    end

endmodule
```

Perform the following steps:

1. Create a new Quartus project using the system builder program. You'll use clock, switches and 7-segment displays.
2. Write a Verilog file.
3. Include the Verilog file in your project and compile the circuit.
4. Simulate the designed circuit to verify its functionality (you don't have to submit your test bench file).
5. Verify that your circuit works correctly by observing the 7-segment displays.

Reference

[1] Intel, "Teaching Materials for the Intel FPGA Academic Program"