Name: _____Servando_Olvera_____ ID# _____1001909287_____

Date Submitted: _____01-25-2024____ Time Submitted _____4:16_pm_____

CSE 3341 Digital Logic Design II

CSE 5357 Advanced Digital Logic Design

Spring Semester 2024

Lab Assignment #1 – Knight Rider Flasher

Due Date – January 25, 2024 (11:59 PM)

Submit on Canvas Assignments

DESIGN REQUIREMENTS

Design in SystemVerilog, implement on the DE10-Lite, and demonstrate a sequential machine that flashes the red LEDs on the DE10-Lite in a continuous back and forth pattern reminiscent of the action lights on the Knight Rider's Firebird Trans Am, KITT. Once you complete this assignment, you will have demonstrated an ability to design and model sequential logic circuits in SystemVerilog that meet specified requirements and to implement the design using a Field Programmable Gate Array (FPGA) employing the following features.

- Clock_50 50-MHz clock
- On/Off toggle
- Clock divider
- Up/Down counter
- Binary to 7-segment decoder
- Module instantiation
- Pin assignment

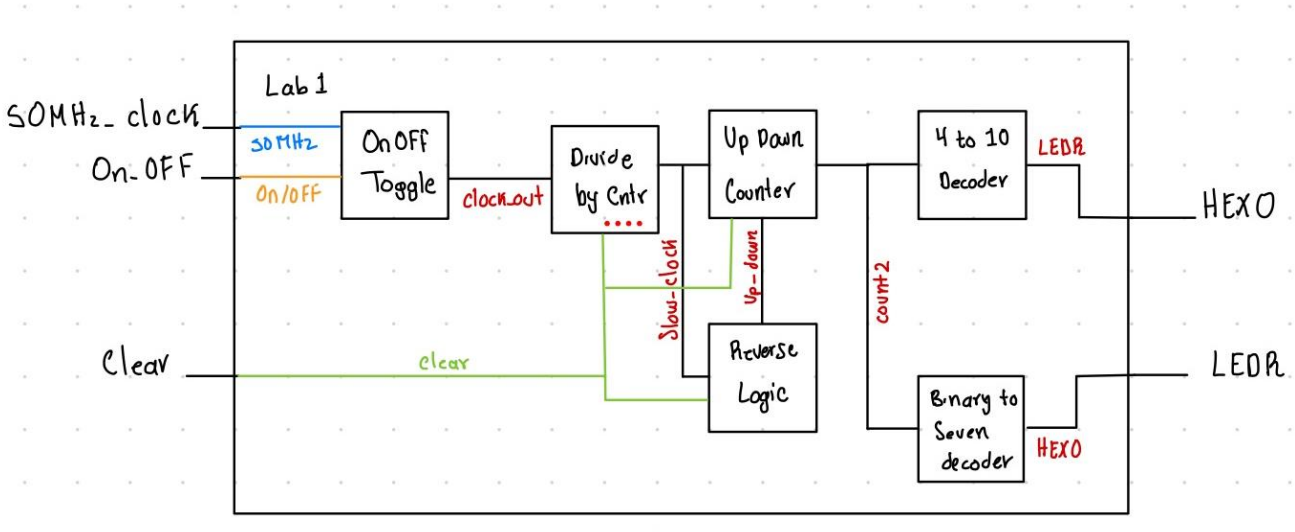
The DE10-Lite board provides a 50-MHz clock (CLOCK_50) that can be used as the time-based for the Flasher. You will need to design a clock divider to provide an appropriate frequency for the machine. The LEDs should flash in a continuous back and forth pattern, reversing direction about once per second, twice per second, and once every two seconds. An up/down counter can be used to realize the LED output pattern. Employ an on/off toggle for your On/Off switch.

Design, implement, and demonstrate the sequential machine described above. Your design must employ a top-down approach and include the following modules.

- Top level
- On/off toggle
- Clock divider
- Up/down counter
- Seven-segment decoder

Lower-level modules should be instantiated in the top-level module.

HIERARCHY DIAGRAM



SYSTEM VERILOG CODE FOR ALL MODULE

- Top Module

```
module Lab1
(
    input CLK, ON_OFF, CLEAR,
    output [9:0] LEDR,
    output [0:6] HEX0
);

       logic clock_out;

    OnOffToggle toggle
    (
       .OnOff(ON_OFF),
       .IN(CLK),
       .OUT(clock_out)
    );

       logic slow_clock, up_down;
       logic [3:0] count1, count2;

    // 5000000  --> Once every sec
    // 2500000  --> Twice every sec
    // 10000000 --> Once every 2 sec

    divideXN #(5000000, 32) divByCntr
    (
       .CLK(clock_out),
       .CLEAR(CLEAR),
       .COUNT(count1),
       .OUT(slow_clock)
    );

    reverseLogic RvrsLgc
    (
       .CLK(slow_clock),
       .CLEAR(CLEAR),
       .UP_DOWN(up_down)
    );

    decadeUpDownCounter UpDwnCntr
    (
       .CLK(slow_clock),
       .CLEAR(CLEAR),
       .UP(up_down),
       .COUNT(count2)
    );

    four2ten fourDecoder
    (
       .BIN(count2),
       .OUT(LEDR)
    );

    binary2seven B2Sev
    (
       .BIN(count2),
       .SEV(HEX0)
    );

endmodule
```

- On and Off Toggle Module

```
module OnOffToggle
(
    input OnOff, IN,
    output OUT
);
    reg state, nextstate;
    parameter ON = 1'b1, OFF = 1'b0;
       always @ (negedge OnOff)
          state <= nextstate;
       always @ (state)
          case(state)
             OFF: nextstate = ON;
             ON: nextstate = OFF;
          endcase
    assign OUT = state*IN;
endmodule
```

- Divide By Counter Module

```
module divideXN #(parameter N = 10, parameter M = 4)     // 5,000,000 for 1 sec
(
    input CLK, CLEAR,
    output logic [M-1:0] COUNT,                                     // COUNT is defined as a M-bit register
    output logic OUT
);
    always_ff @ (negedge CLK, negedge CLEAR)
    if (CLEAR == 1'b0) COUNT <= 0;                                  // COUNT is loaded with all 0's
       else
       begin
          if (COUNT == N-2'd2) begin OUT <= 1'b1; COUNT <= N-1'd1; end      // Once COUNT = N-2 OUT = 1
             else
                if (COUNT == N-1'd1) begin OUT <= 1'b0; COUNT <= 0; end     // Once COUNT = N-1 OUT=0
                   else begin OUT <= 1'b0; COUNT <= COUNT + 1'b1; end       // COUNT is incremented
       end
endmodule
```

- Reverse Logic

```
// Toggle output every 10 clock cycles

module reverseLogic
(
  input CLK, CLEAR,
  output logic UP_DOWN
);

  logic [3:0] counter = 4'b0;

  always_ff @(negedge CLK, negedge CLEAR) begin
    if(CLEAR == 1'b0) begin
      counter <= 4'b0;
      UP_DOWN <= 1'b0;
    end else begin
      counter <= counter + 4'b1;

      if(counter == 4'd8) begin  // 9 does one more cylce than require
        UP_DOWN <= ~UP_DOWN;     // so had to go w 8
        counter <= 4'b0;
      end
    end
  end

endmodule
```

- Up Down Counter Module

```
module decadeUpDownCounter
(
    input CLK, CLEAR, UP,
    output logic [3:0] COUNT
);

    always_ff @ (negedge CLK, negedge CLEAR)
        if(CLEAR == 1'b0) begin
            COUNT <= 4'b0;
        end else begin
            if(UP == 0)
                COUNT <= COUNT + 1'b1;      // if UP increment by 1
            else
                COUNT <= COUNT - 1'b1;      // else, decrease by 1
        end
endmodule
```

- Decoder (4-10) Module

```
module four2ten
(
    input [3:0] BIN,
    output logic [9:0] OUT
);
    always_comb
        case ({BIN[3:0]})
            4'b0000: {OUT[9:0]} = 10'b0000000001;
            4'b0001: {OUT[9:0]} = 10'b0000000010;
            4'b0010: {OUT[9:0]} = 10'b0000000100;
            4'b0011: {OUT[9:0]} = 10'b0000001000;
            4'b0100: {OUT[9:0]} = 10'b0000010000;
            4'b0101: {OUT[9:0]} = 10'b0000100000;
            4'b0110: {OUT[9:0]} = 10'b0001000000;
            4'b0111: {OUT[9:0]} = 10'b0010000000;
            4'b1000: {OUT[9:0]} = 10'b0100000000;
            4'b1001: {OUT[9:0]} = 10'b1000000000;
            default: {OUT[9:0]} = 10'b0000000000;
        endcase
endmodule
```

- Decoder (10-7 seg) Module

```
module binary2seven
(
    input [3:0] BIN,
    output logic [0:6] SEV
);
    always_comb
        case ({BIN[3:0]})
            4'b0000: {SEV[0:6]} = 7'b0000001;   //0
            4'b0001: {SEV[0:6]} = 7'b1001111;   //1
            4'b0010: {SEV[0:6]} = 7'b0010010;   //2
            4'b0011: {SEV[0:6]} = 7'b0000110;   //3
            4'b0100: {SEV[0:6]} = 7'b1001100;   //4
            4'b0101: {SEV[0:6]} = 7'b0100100;   //5
            4'b0110: {SEV[0:6]} = 7'b0100000;   //6
            4'b0111: {SEV[0:6]} = 7'b0001111;   //7
            4'b1000: {SEV[0:6]} = 7'b0000000;   //8
            4'b1001: {SEV[0:6]} = 7'b0001100;   //9
            4'b1010: {SEV[0:6]} = 7'b0001000;   //A
            4'b1011: {SEV[0:6]} = 7'b1100000;   //b
            4'b1100: {SEV[0:6]} = 7'b0110001;   //C
            4'b1101: {SEV[0:6]} = 7'b1000010;   //d
            4'b1110: {SEV[0:6]} = 7'b0110000;   //E
            4'b1111: {SEV[0:6]} = 7'b0111000;   //F
        endcase
endmodule
```

DE-10 PIN ASSIGMENT

	tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✔		CLK	Location	PIN_P11	Yes			
2	✔		ON_OFF	Location	PIN_B8	Yes			
3	✔		HEX0[0]	Location	PIN_C14	Yes			
4	✔		HEX0[1]	Location	PIN_E15	Yes			
5	✔		HEX0[2]	Location	PIN_C15	Yes			
6	✔		HEX0[3]	Location	PIN_C16	Yes			
7	✔		HEX0[4]	Location	PIN_E16	Yes			
8	✔		HEX0[5]	Location	PIN_D17	Yes			
9	✔		HEX0[6]	Location	PIN_C17	Yes			
10	✔		LEDR[0]	Location	PIN_A8	Yes			
11	✔		LEDR[1]	Location	PIN_A9	Yes			
12	✔		LEDR[2]	Location	PIN_A10	Yes			
13	✔		LEDR[3]	Location	PIN_B10	Yes			
14	✔		LEDR[4]	Location	PIN_D13	Yes			
15	✔		LEDR[5]	Location	PIN_C13	Yes			
16	✔		LEDR[6]	Location	PIN_E14	Yes			
17	✔		LEDR[7]	Location	PIN_D14	Yes			
18	✔		LEDR[8]	Location	PIN_A11	Yes			
19	✔		LEDR[9]	Location	PIN_B11	Yes			
20	✔		CLEAR	Location	PIN_A7	Yes			
21		<<new>>	<<new>>	<<new>>					

DEMOED IN PERSON

Demoed on 01/24/2024

To: TA (Madison)