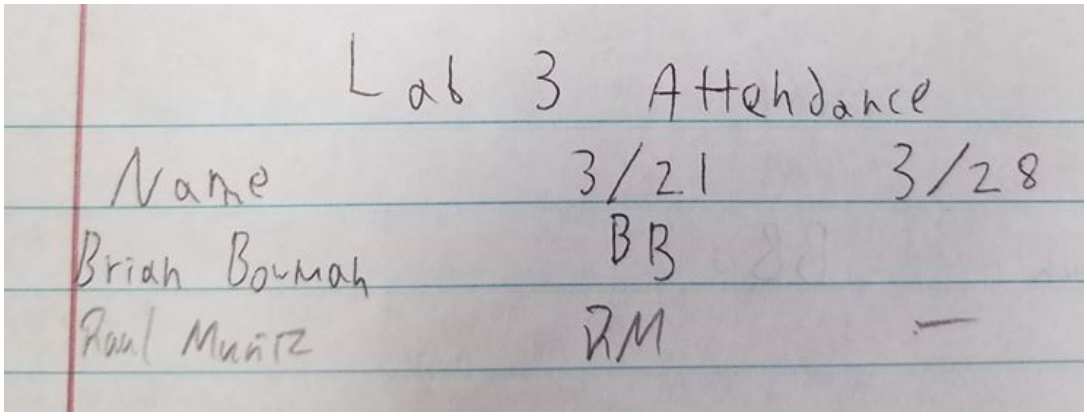
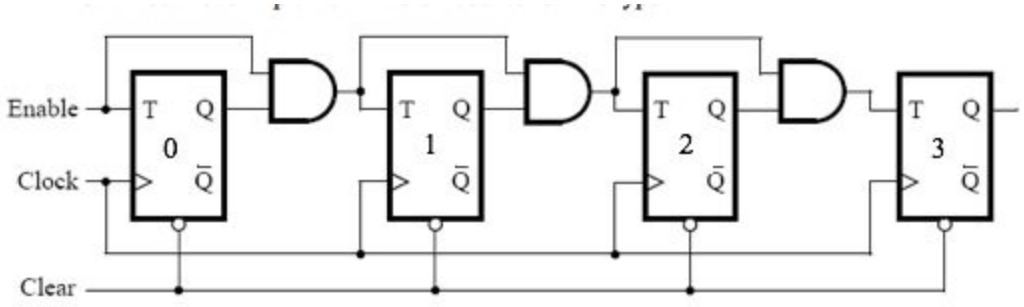


Part 1

I. **Problem:** Implement an 8-bit synchronous counter using T flip-flops and AND gates.

II. **Conceptual Design:**



III. **Verilog Design:**

#Logic Elements: 32

```
module lab3part1 (SW, KEY, LEDR, HEX1, HEX0);
    input [1:0] SW; // SW[1] as enable input; SW[0] as async reset
    input [0:0] KEY; // clock
    output [7:0] LEDR; // counter state in binary
    output [6:0] HEX1, HEX0; // counter state in hex

    wire [7:0] Q;

    assign LEDR = Q;

    counter8 U0 (SW[1], SW[0], KEY, Q);

    hex2seg7 H0 (Q[3:0], HEX0);
    hex2seg7 H1 (Q[7:4], HEX1);
endmodule

module counter8 (en, clear, clk, Q);
```

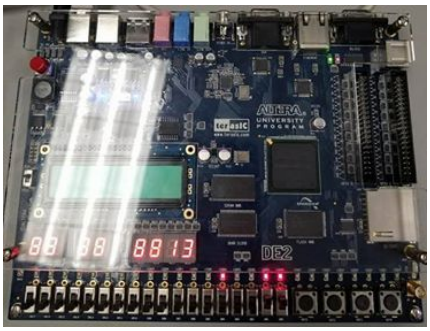
```
    input en, clear, clk;
    output [7:0] Q;

    wire [7:0] T;

    tflipflop U0 (clear, T[0], clk, Q[0]);
    tflipflop U1 (clear, T[1], clk, Q[1]);
    tflipflop U2 (clear, T[2], clk, Q[2]);
    tflipflop U3 (clear, T[3], clk, Q[3]);
    tflipflop U4 (clear, T[4], clk, Q[4]);
    tflipflop U5 (clear, T[5], clk, Q[5]);
    tflipflop U6 (clear, T[6], clk, Q[6]);
    tflipflop U7 (clear, T[7], clk, Q[7]);

    assign T[0] = en;
    assign T[1] = T[0] && Q[0];
    assign T[2] = T[1] && Q[1];
    assign T[3] = T[2] && Q[2];
    assign T[4] = T[3] && Q[3];
    assign T[5] = T[4] && Q[4];
    assign T[6] = T[5] && Q[5];
    assign T[7] = T[6] && Q[6];
endmodule

module tflipflop (resetn, t, clk, q);
    input resetn, t, clk;
    output reg q;

    always @(posedge clk or negedge resetn) begin
      if (~resetn)
           q <= 0;
      else
           q <= t ? ~q : q;
    end
endmodule

module hex2seg7 (bin, seg7);
    input [3:0] bin;
    output reg [6:0] seg7;

    always @ (*) begin
      case (bin)
           4'b0000 : seg7 = 7'b1000000; // 0
```

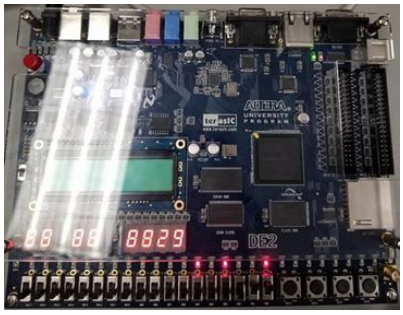
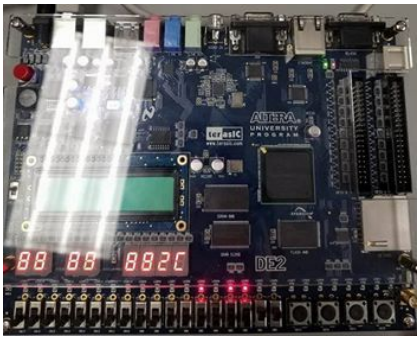
```
            4'b0001 : seg7 = 7'b1111001; // 1
            4'b0010 : seg7 = 7'b0100100; // 2
            4'b0011 : seg7 = 7'b0110000; // 3
            4'b0100 : seg7 = 7'b0011001; // 4
            4'b0101 : seg7 = 7'b0010010; // 5
            4'b0110 : seg7 = 7'b0000010; // 6
            4'b0111 : seg7 = 7'b1111000; // 7
            4'b1000 : seg7 = 7'b0000000; // 8
            4'b1001 : seg7 = 7'b0010000; // 9
            4'b1010 : seg7 = 7'b0001000; // A
            4'b1011 : seg7 = 7'b0000011; // B
            4'b1100 : seg7 = 7'b1000110; // C
            4'b1101 : seg7 = 7'b0100001; // D
            4'b1110 : seg7 = 7'b0000110; // E
            4'b1111 : seg7 = 7'b0001110; // F
            default : seg7 = 7'b1111111; // blank
        endcase
    end
endmodule
```

IV. Lab Procedure:

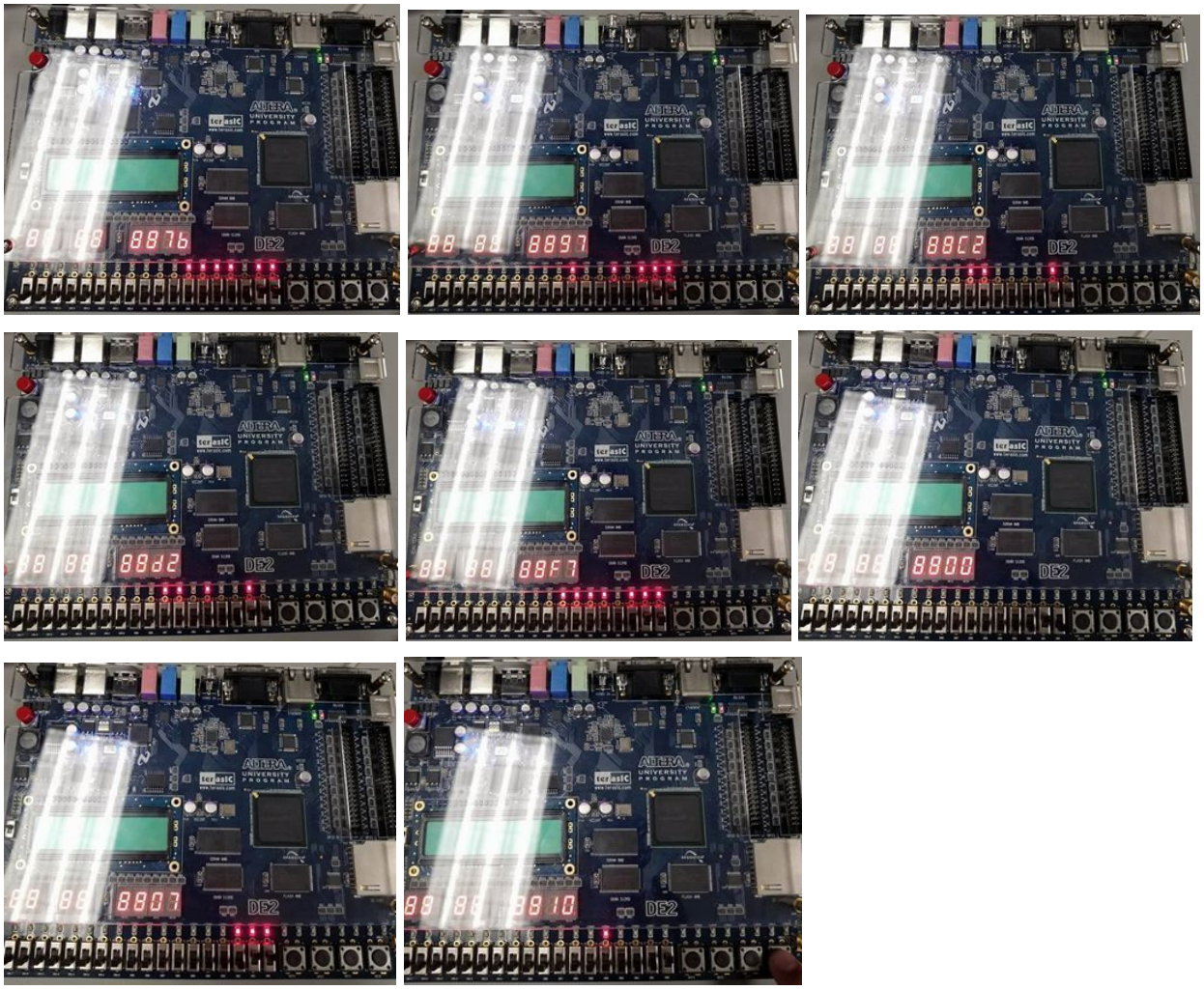
Pictures





Lab3
Brian Bowman
Raul Muniz



Troubleshooting: When we first ran the code on the board, we pushed the button but nothing happened. Then we remembered we had to switch on the enable and low-active reset switches. We did that, but still nothing happened. We eventually figured out that we had to change the line `'input KEY;' to 'input [0:0] KEY;'` in order for the switch to respond.

Conclusion: Many modules had to be implemented and in sync with each other in order to perform the specific task. "Lab3part1," basically declared the proper variables, SW, LEDR, etc.. counter8 began the count with each time the button being pressed. Tflipflop to properly reset the count. Finally, hex2seg7 to demonstrate the proper hexadecimal number according to the switch.

Part 2

I. **Problem:** Implement an 8-bit synchronous counter using addition and assignment operators in Verilog.

II. **Conceptual Design:** 8-bit synchronous counter using Q <=Q+1
The functionality of the circuit is the same as part 1. The counter is accomplished using the statement: Q <= Q+1;

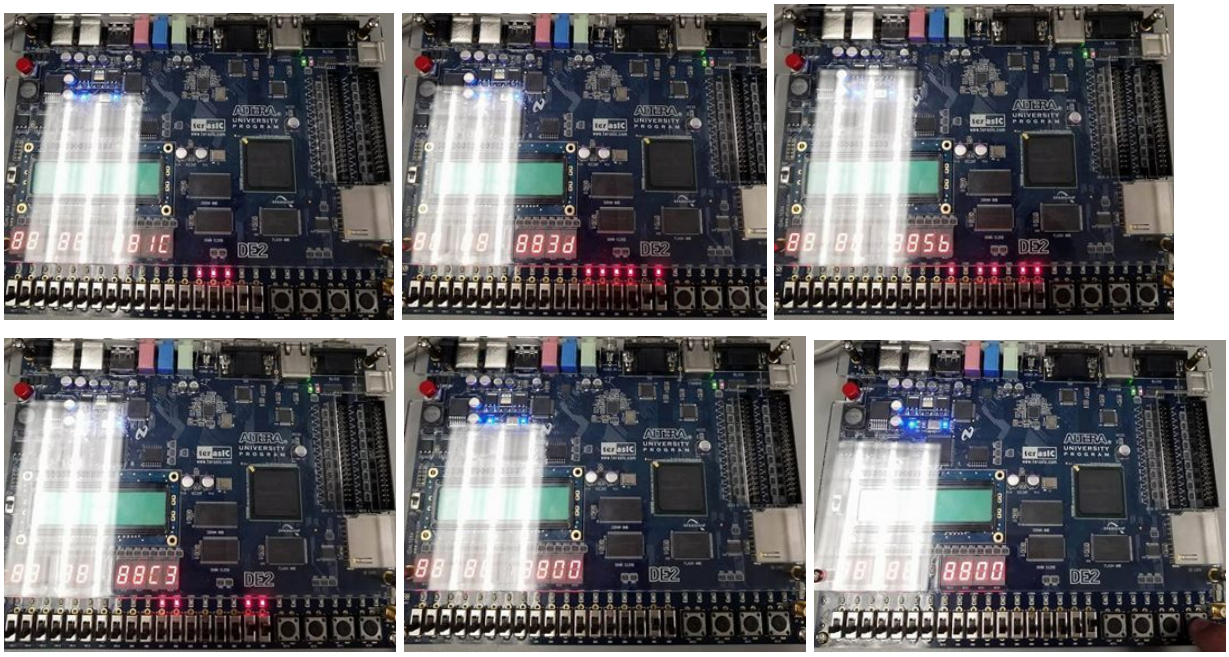
III. Verilog Design:

#Logic Elements: 23

```
module lab3part2 (SW, KEY, LEDR, HEX1, HEX0);
    input [1:0] SW; // SW[1] as enable input; SW[0] as async reset
    input [0:0] KEY; // clock
    output [7:0] LEDR; // counter state in binary
    output [6:0] HEX1, HEX0; // counter state in hex

    wire [7:0] Q;

    assign LEDR = Q;

    counter8new U0 (SW[1], SW[0], KEY, Q);

    hex2seg7 H0 (Q[3:0], HEX0);
    hex2seg7 H1 (Q[7:4], HEX1);
endmodule

module counter8new (en, clear, clk, Q);
    input en, clear, clk;
    output reg [7:0] Q;

    always @(posedge clk or negedge clear) begin
      if (!clear)
           Q <= 0;
      else if (en) begin
           Q <= Q + 1;
      end
    end
endmodule
```

IV. Lab Procedure:

Pictures



Troubleshooting: The design functioned correctly after we performed the same modification as in part one of specifying "`[0:0]`" for the KEY input.
Conclusion: Pretty much the same as part1 of the lab, but implemented the 8-bit synchronous counter using Q <=Q+1 this time. Not too difficult.

Part 3

I. **Problem:** Implement an 8-bit synchronous counter that counts up every second.
II. **Conceptual Design:** The circuit, driven by a 50MHz clock (CLOCK_50) (on-board), will display two-digit hexadecimal numbers from 00 to FF on two 7-seg displays HEX1 and HEX0 at a refreshing rate of 1Hz. The circuit consists of 3 parts: 1) 1 Hz clock signal generator; 2) 8-bit binary counter with reset (SW[0]) and enable (SW[1]); and 3) binary-seg decoder for the display (seg7.v in lab2)
III. **Verilog Design**

#Logic Elements: 80

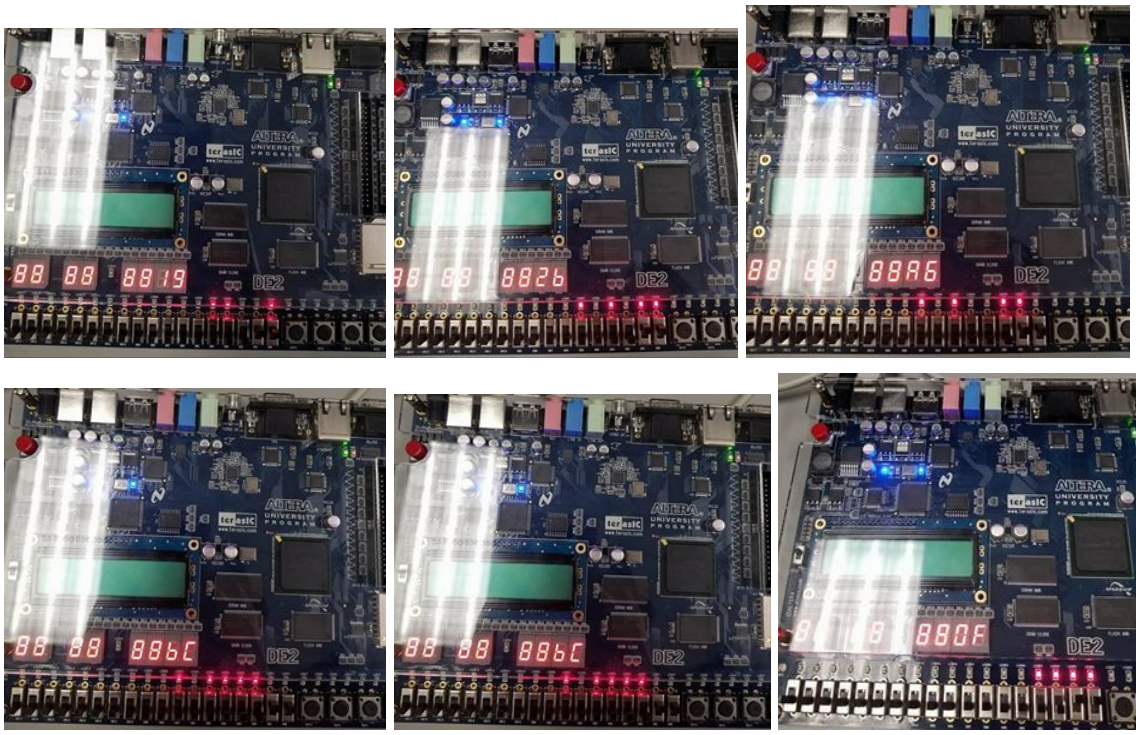
```
module lab3part3 (SW, CLOCK_50, LEDR, HEX1, HEX0);
    input [1:0] SW; // SW[1] as enable input; SW[0] as async reset
    input CLOCK_50; // DE2 50MHz clock
    output [7:0] LEDR; // counter state in binary
    output [6:0] HEX1, HEX0; // counter state in hex

    wire clk1Hz;
    wire [7:0] Q;

    assign LEDR = Q;
```

```
    clock_div V0 (CLOCK_50, 25_000_000, clk1Hz);

    counter8new U0 (SW[1], SW[0], clk1Hz, Q);

    hex2seg7 H0 (Q[3:0], HEX0);
    hex2seg7 H1 (Q[7:4], HEX1);
endmodule

// output frequency = clk_in frequency / (2 * clkscale)
module clock_div (clk_in, clkscale, clk_out);
    input clk_in;
    input [31:0] clkscale;
    output reg clk_out;

    reg [31:0] clkq = 0;

    always @ (posedge clk_in) begin
       clkq = clkq + 1;
       if (clkq == clkscale) begin
            clk_out = ~clk_out;
            clkq = 0;
       end
    end
endmodule
```

IV. Lab Procedure:

Pictures

Lab3
Brian Bowman
Raul Muniz



Troubleshooting: n/a
Conclusion: We implemented a clock to start the count automatically using the verilog files used previously in the other parts of the lab.