CPE - 133
Fall 2018

Lab 6
Jared Rocha
Luis Gomez

Exp 11: A Simple Data Latching and Display Circuit

Summary: We implemented the sseg_dec.v module provided and created an a dflop with a button. So when the button was pressed the switch input would be latched into memory The input could only change if the button was pressed again to release the input being held by the latch.

Figure below is the BDD for exp 11

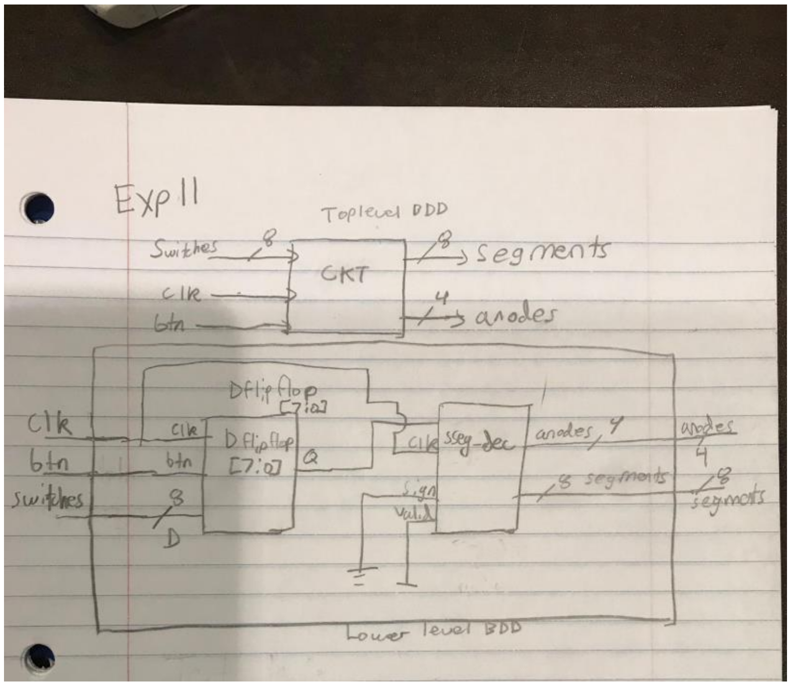


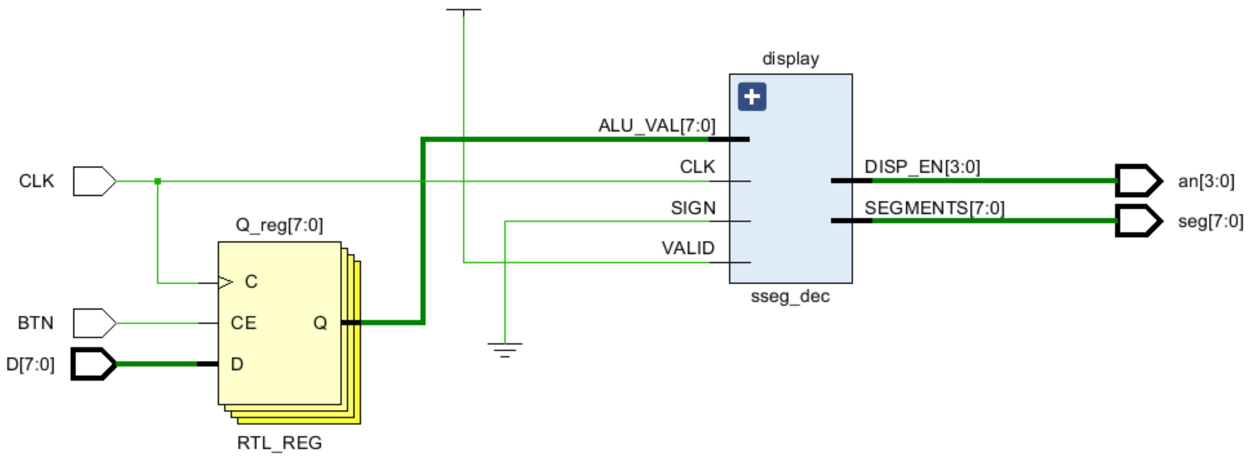
Figure below, Excitation Table for exp 11

Q	Qnext	D
0	0	0
0	1	1
1	0	0
1	1	1

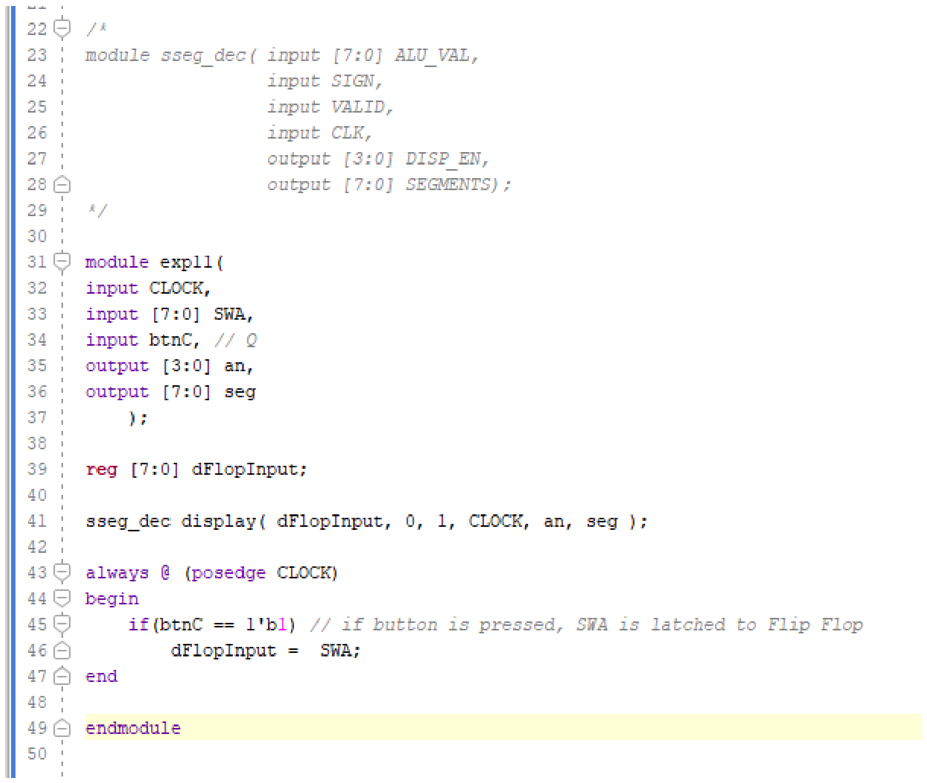
Verification: Our verification process entailed extensive physical board tests as demonstrated in the video link below.

Video: https://youtu.be/nwjEh1qY0zE

Figure below is the Circuit Diagram of exp 11 provided by Vivado



Code



```
/*
module sseg_dec( input [7:0] ALU_VAL,
                 input SIGN,
                 input VALID,
                 input CLK,
                 output [3:0] DISP_EN,
                 output [7:0] SEGMENTS);
*/

module exp11(
input CLOCK,
input [7:0] SWA,
input btnC, // Q
output [3:0] an,
output [7:0] seg
    );

reg [7:0] dFlopInput;

sseg_dec display( dFlopInput, 0, 1, CLOCK, an, seg );

always @ (posedge CLOCK)
begin
    if(btnC == 1'b1) // if button is pressed, SWA is latched to Flip Flop
        dFlopInput =  SWA;
end

endmodule

```

Questions:

1. When you press the button, how many times is the data effectively "latched" in the associated D flip-flops? HINT: realize that the clock input from the development board operates at 100Mhz

100000000 times if button is pushed for 1 second. f(t)= t(100000000/seconds).

2. Knowing what you know about Verilog, do you think there is an easier approach to designing the set of eight flip-flops? Briefly speculate and explain.

Create an always block sensitive to button change with a series of if statement within the always block.

Exp 12: T Flip-Flop Based Four-Bit Counter

Summary: In exp 12 we attempted to create a Finite State Machine made up by three modules: the T-flop, 4bit counter, and the provided sseg-module. For this exp we were able to create a T-Flop module integrating the sseg module, but we were not able to complete step 2 as our counter module did not work and requires more design.

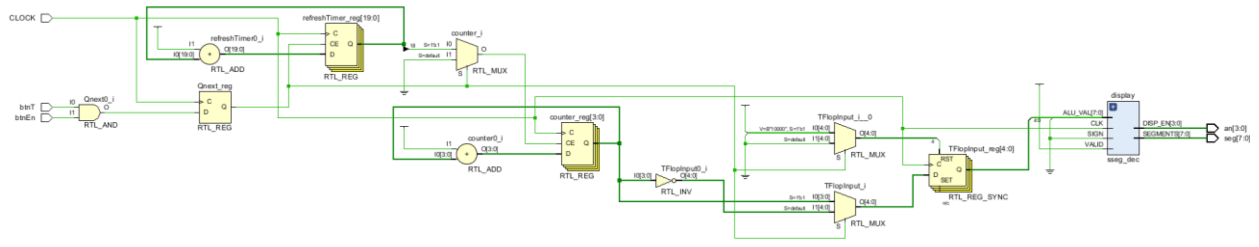
Characteristic Equation for T-Flop: *Qnext = T & EN*

Q	Qnext	T	EN
0	Q	0	X
0	**~Q**	0	1
1	**~Q**	1	1
1	Q	0	X

Figure below, K-Map for exp 12, generated using Excitable Table above

Q/T,EN	00	01	11	10
0	Q	Q	**~Q**	X
1	Q	Q	**~Q**	X

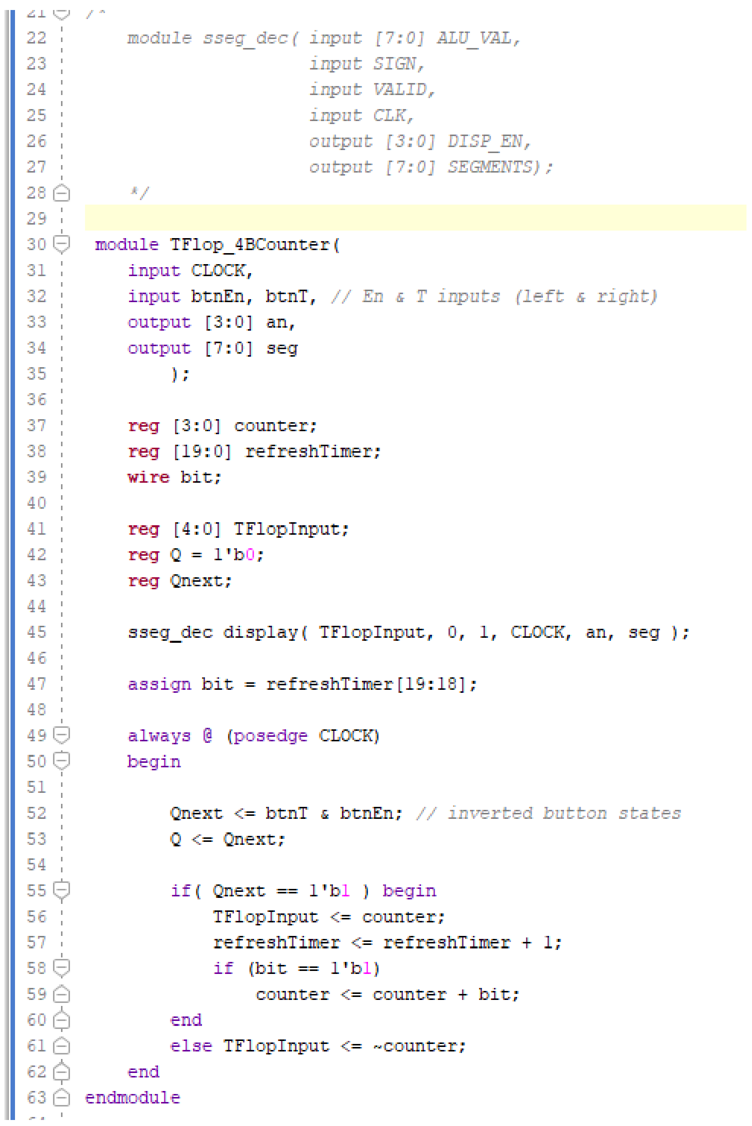
Figure below is the circuit diagram for our attempt of exp 12



Verification: We unfortunately struggled to successfully implement a real-time counter that would increment at a rate of 1-bit per second and that could display its change in value real-time. Our attempt at creating the module failed at Step 2, but we were able to integrate the sseg-module to visualize the results of our work. The module we did create implemented an internal counter variable that incremented while pressed but unfortunately did not display its increment value on the BCD unless the enable was released and then pressed again. We included evidence of our work in the demo video below and in the code/Summary sections of the report.

Video: https://youtu.be/x8SDC0OMgwc

Code



```
/*
    module sseg_dec( input [7:0] ALU_VAL,
                     input SIGN,
                     input VALID,
                     input CLK,
                     output [3:0] DISP_EN,
                     output [7:0] SEGMENTS);
    */

 module TFlop_4BCounter(
    input CLOCK,
    input btnEn, btnT, // En & T inputs (left & right)
    output [3:0] an,
    output [7:0] seg
        );

    reg [3:0] counter;
    reg [19:0] refreshTimer;
    wire bit;

    reg [4:0] TFlopInput;
    reg Q = 1'b0;
    reg Qnext;

    sseg_dec display( TFlopInput, 0, 1, CLOCK, an, seg );

    assign bit = refreshTimer[19:18];

    always @ (posedge CLOCK)
    begin

        Qnext <= btnT & btnEn; // inverted button states
        Q <= Qnext;

        if( Qnext == 1'b1 ) begin
            TFlopInput <= counter;
            refreshTimer <= refreshTimer + 1;
            if (bit == 1'b1)
                counter <= counter + bit;
        end
        else TFlopInput <= ~counter;
    end
endmodule
```

Questions:

1. How many flip-flops did this FSM design use?

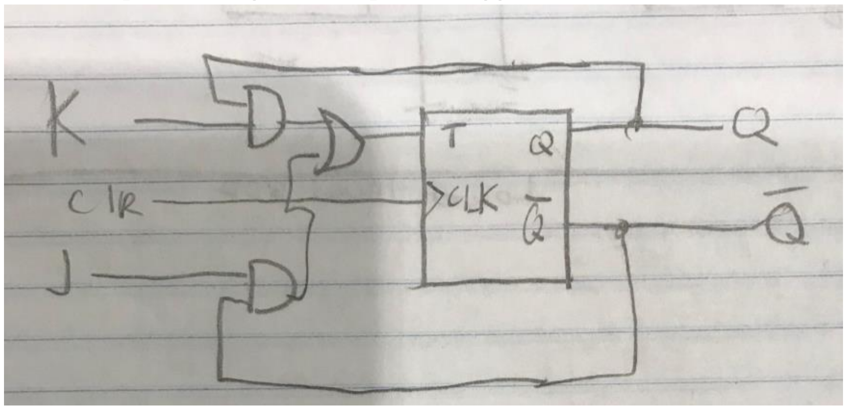
Although we were unsuccessful in implementing the required design we believe that the total number of FF's in the FSM would be at least 12 + n, where n is the bit-size of your timer variable in the counter module. Specifically, the FF's arose from the following: T-flop module, 4-FF's, Counter-Module includes a variable number depending on the bit-size of your refresh-Timer variable, and the sseg module included 8-FF's.

2. Without thinking about it, this counter simply "rolled over" when the highest count was reached. What was it that made it magically roll over? Briefly explain

By counting to 15 it rolls back because there are only 4 bits. So when the value is 15 = 1111, adding a single bit results in a carryout and the value becomes 0000.

3. Describe the changes that would be required to implement this counter using JK flip-flops instead of T flip-flops.

Add an AND gate that inputs are Q and K. Add an and gate that inputs are ~Q and J. OR the two AND gates outputs. Make the output of OR gate the input for toggle button.



4. This counter is referred to as an "up counter" (counts from low binary number to high binary numbers). Describe the changes that would be required in order to make it a "down counter".

We suppose a down-counter could be created by inverting the T-flop result so that pressing T results in the default value not the inverted result. Then what you would see on the BCD output from the sseg module is the inverted result decreasing in value as the default value increases.