Lab 8: Counters, Clock Dividers and Debounce Circuits

ECEN 248 - 505

TA: Younggyun Cho

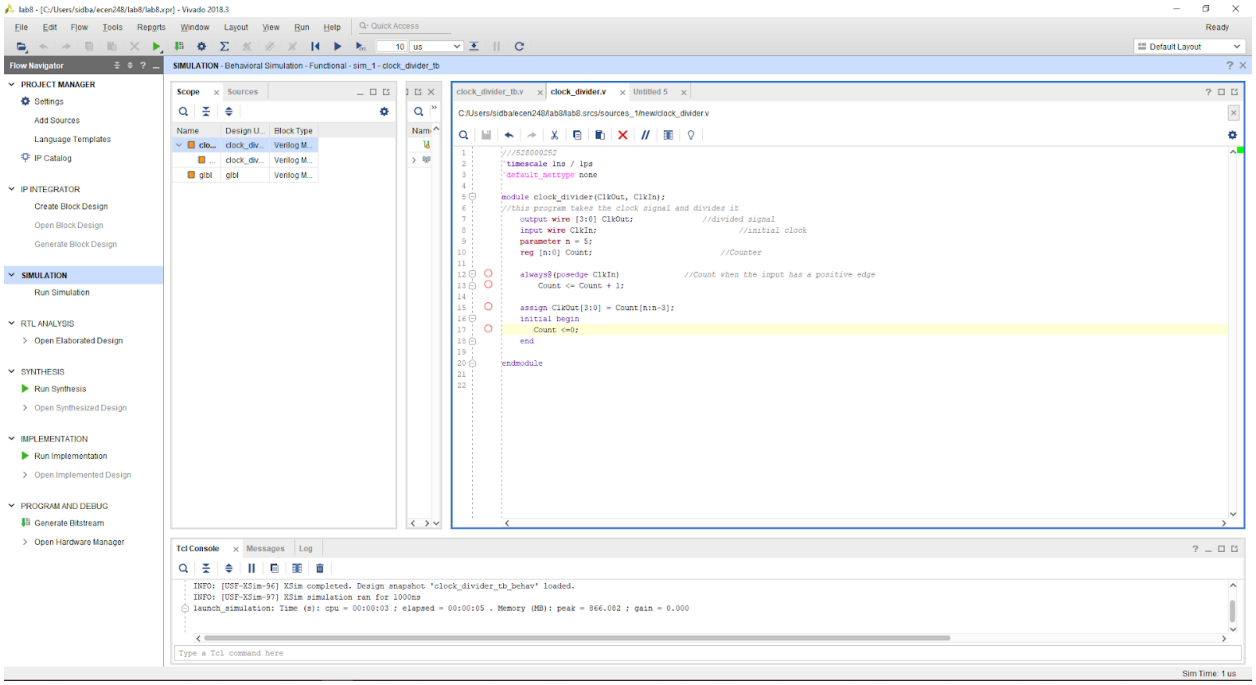
Date: October 27, 2020

Objectives

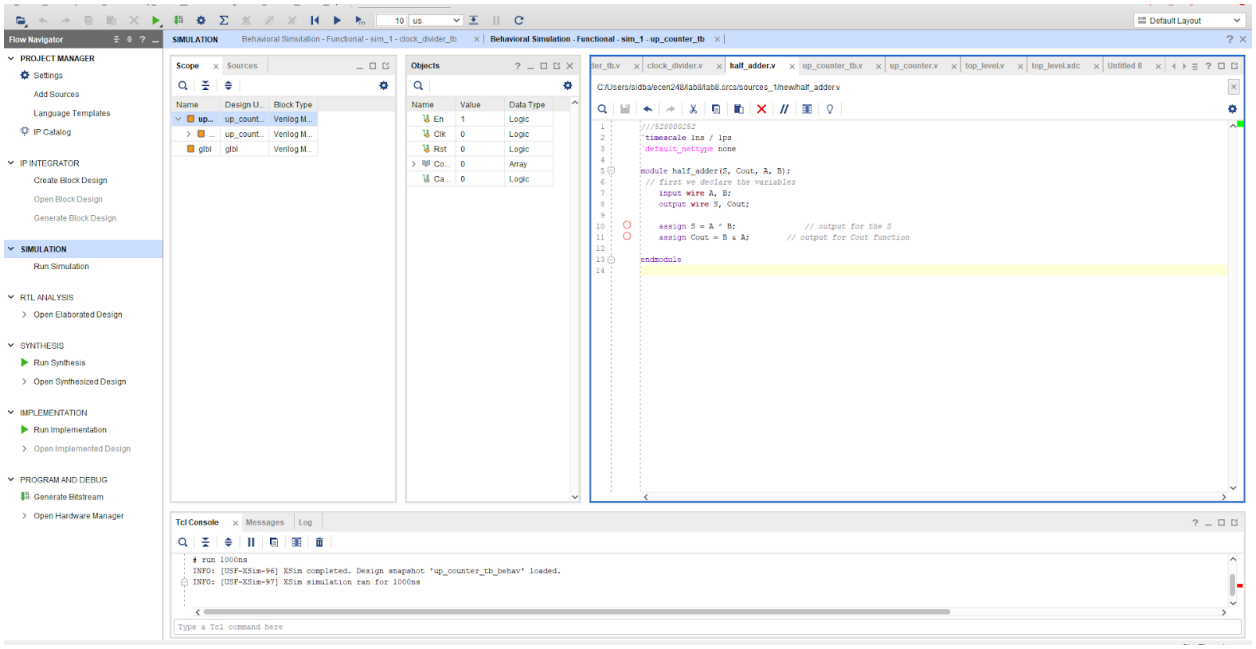
The purpose of this lab is to understand the way counters and clock dividers and debounce circuits work and their foundational functions. In this lab the students will utilize the FPGA board to perform clock dividers. Students will understand how to code counters, clock dividers, and debounce circuits using Verilog.

Design

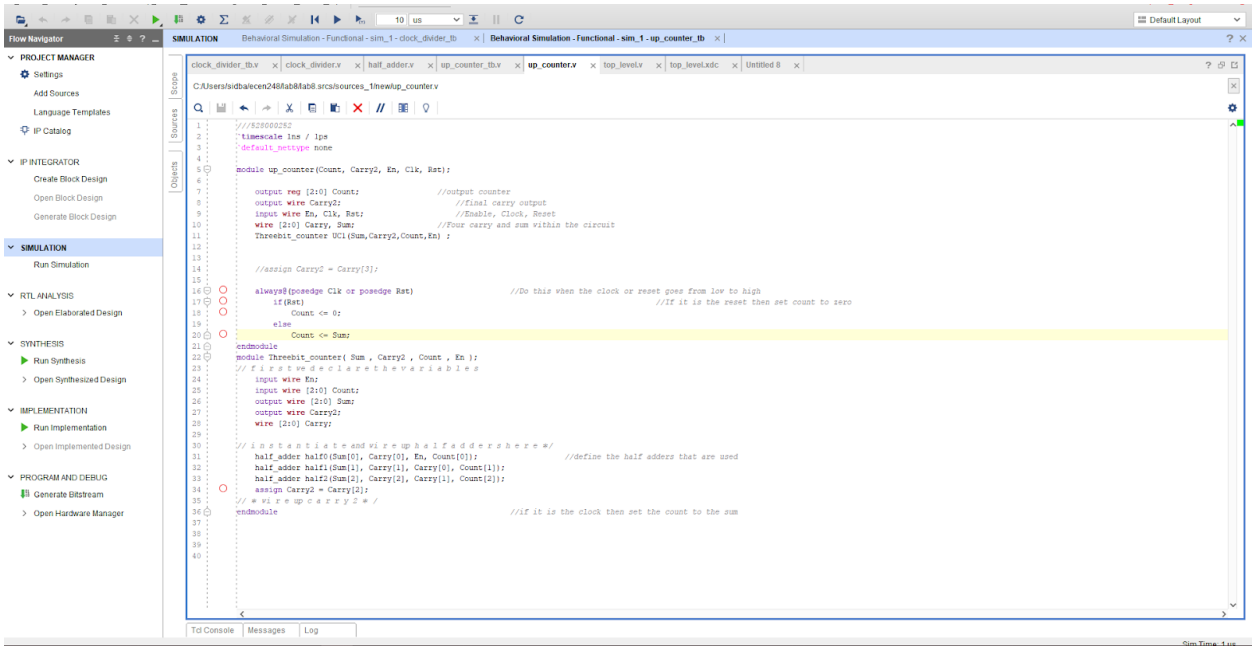
//clock divider



//Half adder

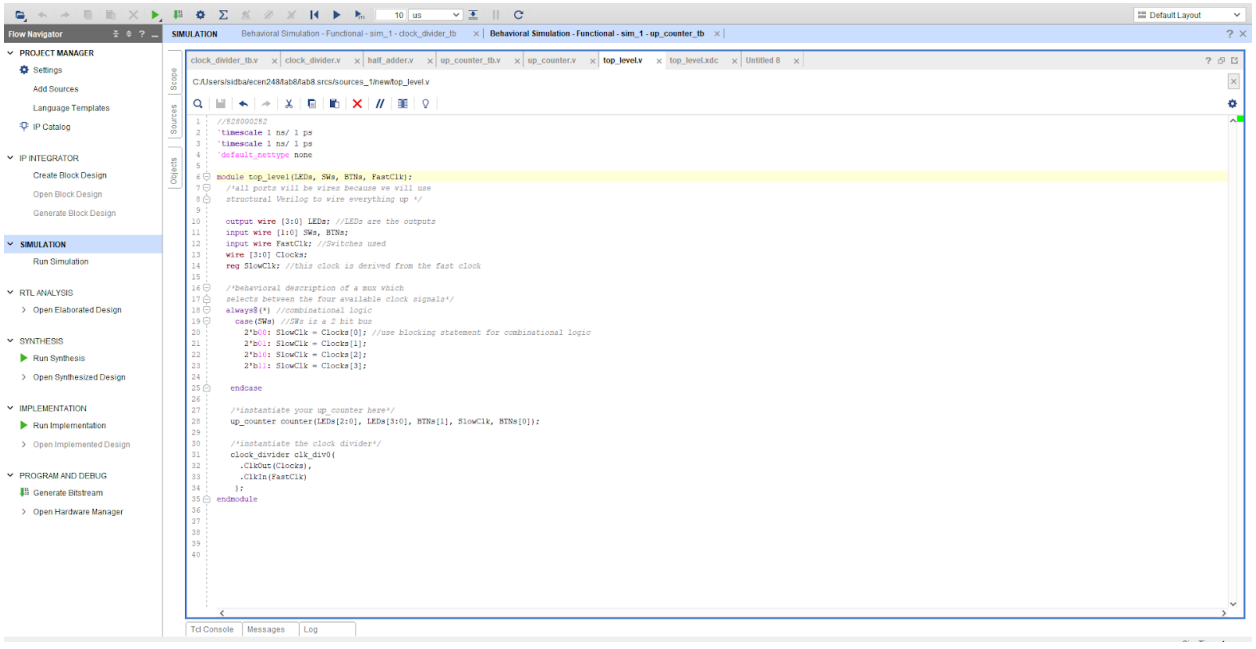


//up counter



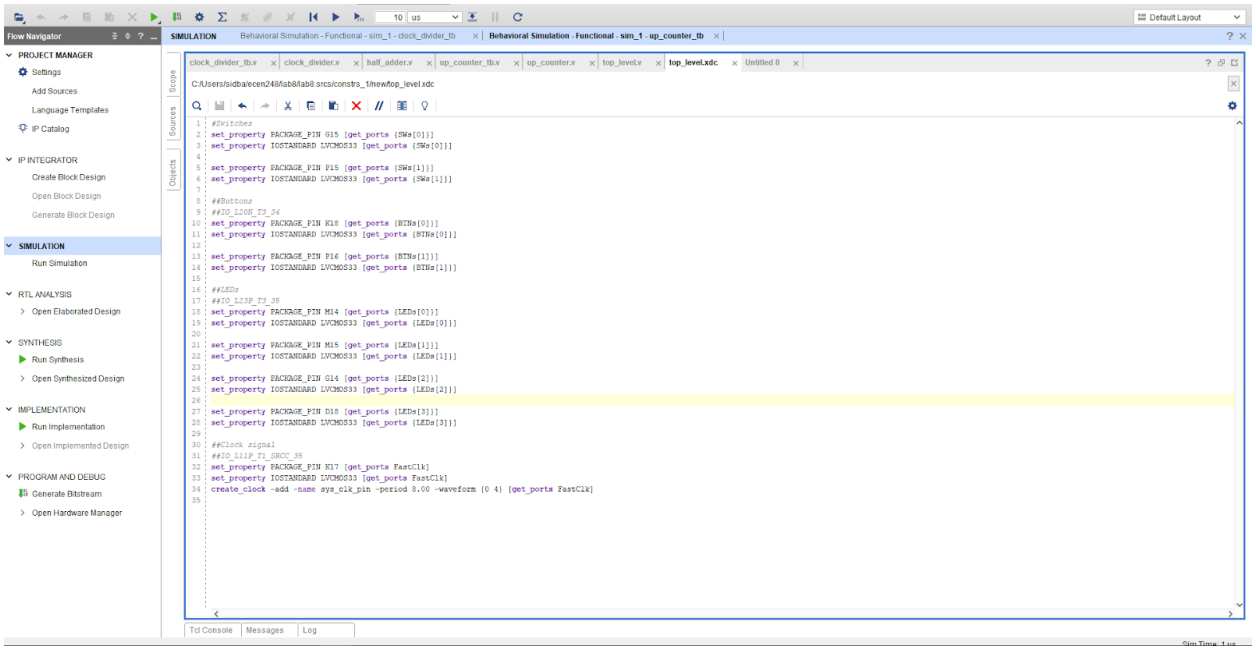
```
////528000252
`timescale 1ns / 1ps
`default_nettype none

module up_counter(Count, Carry2, En, Clk, Rst);

    output reg [2:0] Count;                 //output counter
    output wire Carry2;                     //final carry output
    input wire En, Clk, Rst;                //Enable, Clock, Reset
    wire [2:0] Carry, Sum;                  //Four carry and sum within the circuit
    Threebit_counter UC1(Sum,Carry2,Count,En) ;


    //assign Carry2 = Carry[3];

    always@(posedge Clk or posedge Rst)                 //Do this when the clock or reset goes from low to high
        if(Rst)                                                             //If it is the reset then set count to zero
            Count <= 0;
        else
            Count <= Sum;
endmodule
module Threebit_counter( Sum , Carry2 , Count , En );
// f i r s t w e d e c l a r e t h e v a r i a b l e s
    input wire En;
    input wire [2:0] Count;
    output wire [2:0] Sum;
    output wire Carry2;
    wire [2:0] Carry;

// i n s t a n t i a t e and w i r e up h a l f a d d e r s h e r e */
    half_adder half0(Sum[0], Carry[0], En, Count[0]);               //define the half adders that are used
    half_adder half1(Sum[1], Carry[1], Carry[0], Count[1]);
    half_adder half2(Sum[2], Carry[2], Carry[1], Count[2]);
    assign Carry2 = Carry[2];
// * w i r e up c a r r y 2 * /
endmodule                                                           //If it is the clock then set the count to the sum
```

//top level



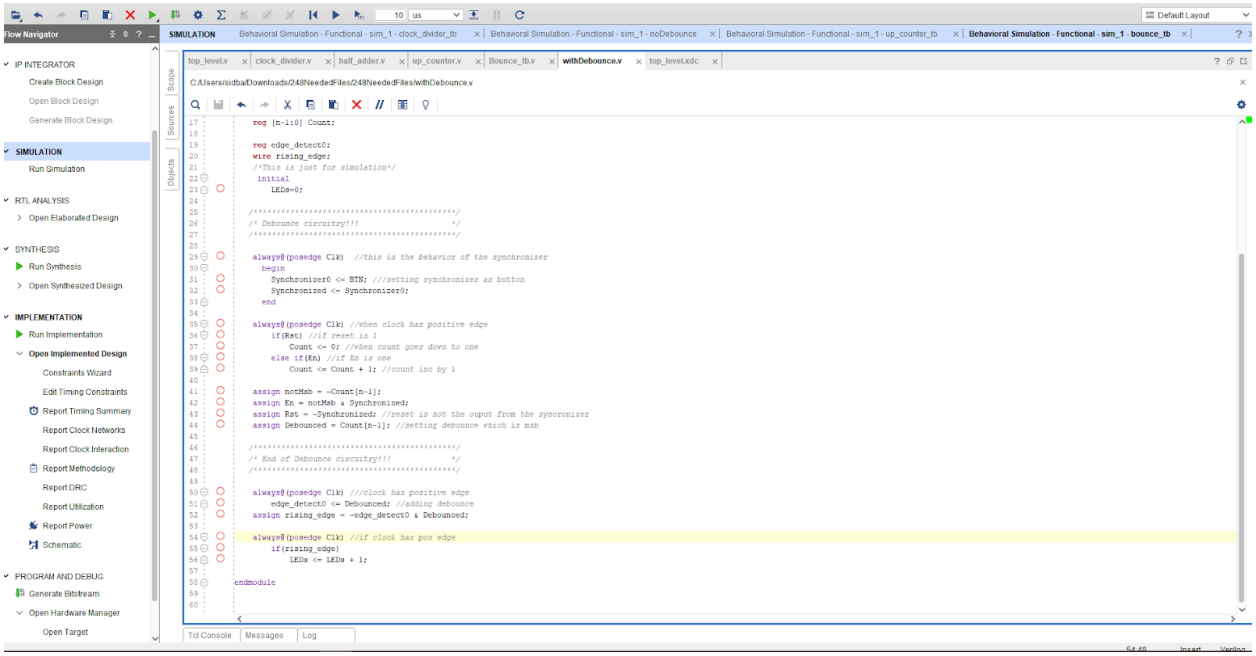
```
//528000252
`timescale 1 ns/ 1 ps
`timescale 1 ns/ 1 ps
`default_nettype none

module top_level(LEDs, SWs, BTNs, FastClk);
  /*all ports will be wires because we will use
  structural Verilog to wire everything up */

  output wire [3:0] LEDs; //LEDs are the outputs
  input wire [1:0] SWs, BTNs;
  input wire FastClk; //Switches used
  wire [3:0] Clocks;
  reg SlowClk; //this clock is derived from the fast clock

  /*behavioral description of a mux which
  selects between the four available clock signals*/
  always@(*) //combinational logic
    case(SWs) //SWs is a 2 bit bus
      2'b00: SlowClk = Clocks[0]; //use blocking statement for combinational logic
      2'b01: SlowClk = Clocks[1];
      2'b10: SlowClk = Clocks[2];
      2'b11: SlowClk = Clocks[3];

    endcase

    /*instantiate your up_counter here*/
    up_counter counter(LEDs[2:0], LEDs[3:0], BTNs[1], SlowClk, BTNs[0]);

    /*instantiate the clock divider*/
    clock_divider clk_div0(
      .ClkOut(Clocks),
      .ClkIn(FastClk)
    );
endmodule
```

//top level xdc



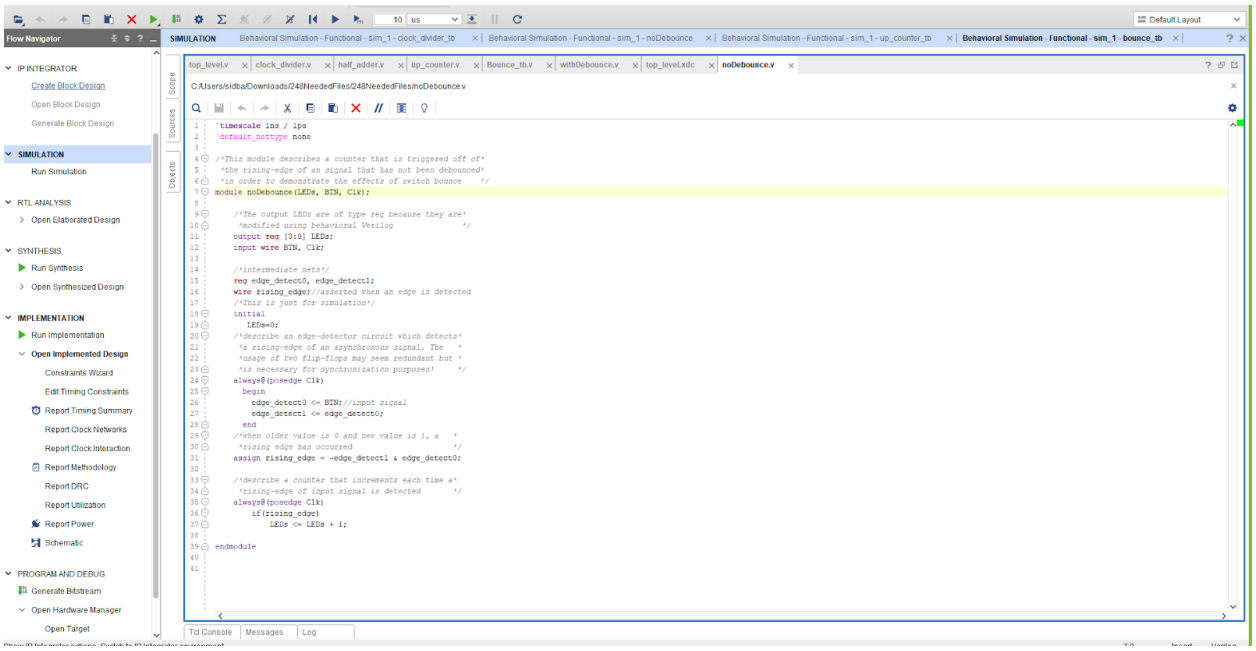
```
#Switches
set_property PACKAGE_PIN G15 [get_ports {SWs[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWs[0]}]

set_property PACKAGE_PIN P15 [get_ports {SWs[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SWs[1]}]

##Buttons
##IO_L20N_T3_34
set_property PACKAGE_PIN K18 [get_ports {BTNs[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTNs[0]}]

set_property PACKAGE_PIN P16 [get_ports {BTNs[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTNs[1]}]

##LEDs
##IO_L23P_T3_35
set_property PACKAGE_PIN M14 [get_ports {LEDs[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[0]}]

set_property PACKAGE_PIN M15 [get_ports {LEDs[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[1]}]

set_property PACKAGE_PIN G14 [get_ports {LEDs[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[2]}]

set_property PACKAGE_PIN D18 [get_ports {LEDs[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LEDs[3]}]

##Clock signal
##IO_L11P_T1_SRCC_35
set_property PACKAGE_PIN K17 [get_ports FastClk]
set_property IOSTANDARD LVCMOS33 [get_ports FastClk]
create_clock -add -name sys_clk_pin -period 8.00 -waveform {0 4} [get_ports FastClk]
```

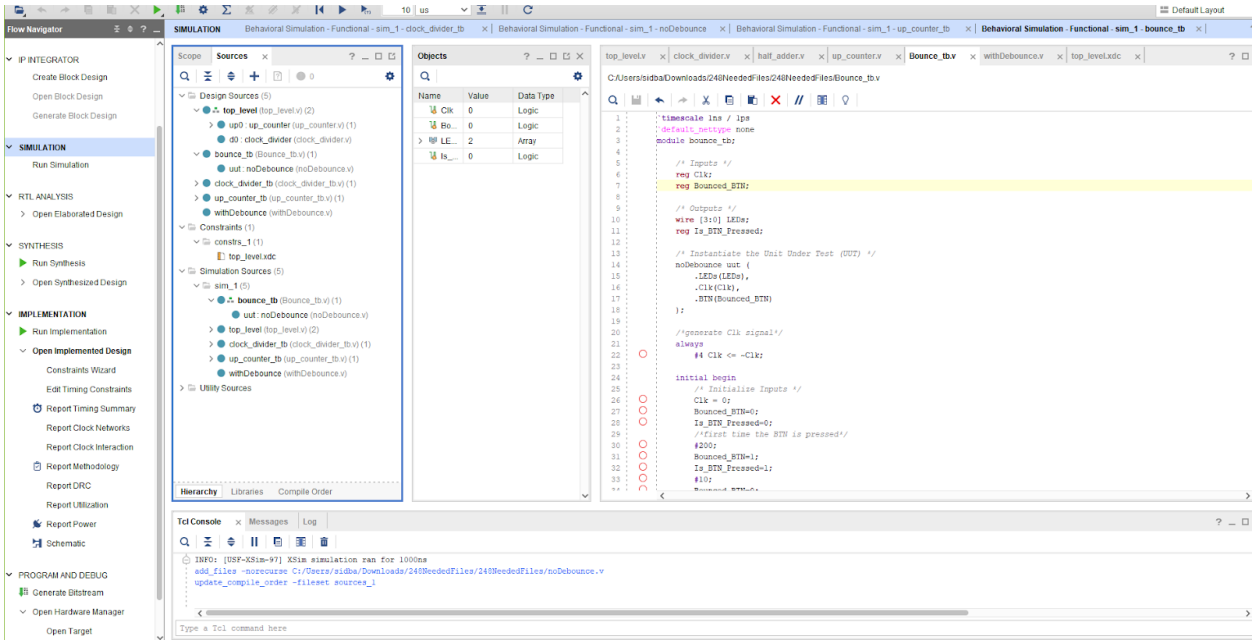
//with debounce comments



```
    reg [n-1:0] Count;

    reg edge_detect0;
    wire rising_edge;
    /*This is just for simulation*/
     initial
        LEDs=0;

    /*****************************************/
    /* Debounce circuitry!!!                 */
    /*****************************************/

    always@(posedge Clk)  //this is the behavior of the synchronizer
      begin
        Synchronizer0 <= BTN; ///setting synchronizer as button
        Synchronized <= Synchronizer0;
      end

    always@(posedge Clk) //when clock has positive edge
        if(Rst) //if reset is 1
            Count <= 0; //when count goes down to one
        else if(En) //if En is one
            Count <= Count + 1; //count inc by 1

    assign notMsb = ~Count[n-1];
    assign En = notMsb & Synchronized;
    assign Rst = ~Synchronized; //reset is not the ouput from the syncronizer
    assign Debounced = Count[n-1]; //setting debounce which is msb

    /*****************************************/
    /* End of Debounce circuitry!!!          */
    /*****************************************/

    always@(posedge Clk) ///clock has positive edge
        edge_detect0 <= Debounced; //adding debounce
    assign rising_edge = ~edge_detect0 & Debounced;

    always@(posedge Clk) //if clock has pos edge
        if(rising_edge)
            LEDs <= LEDs + 1;

endmodule
```

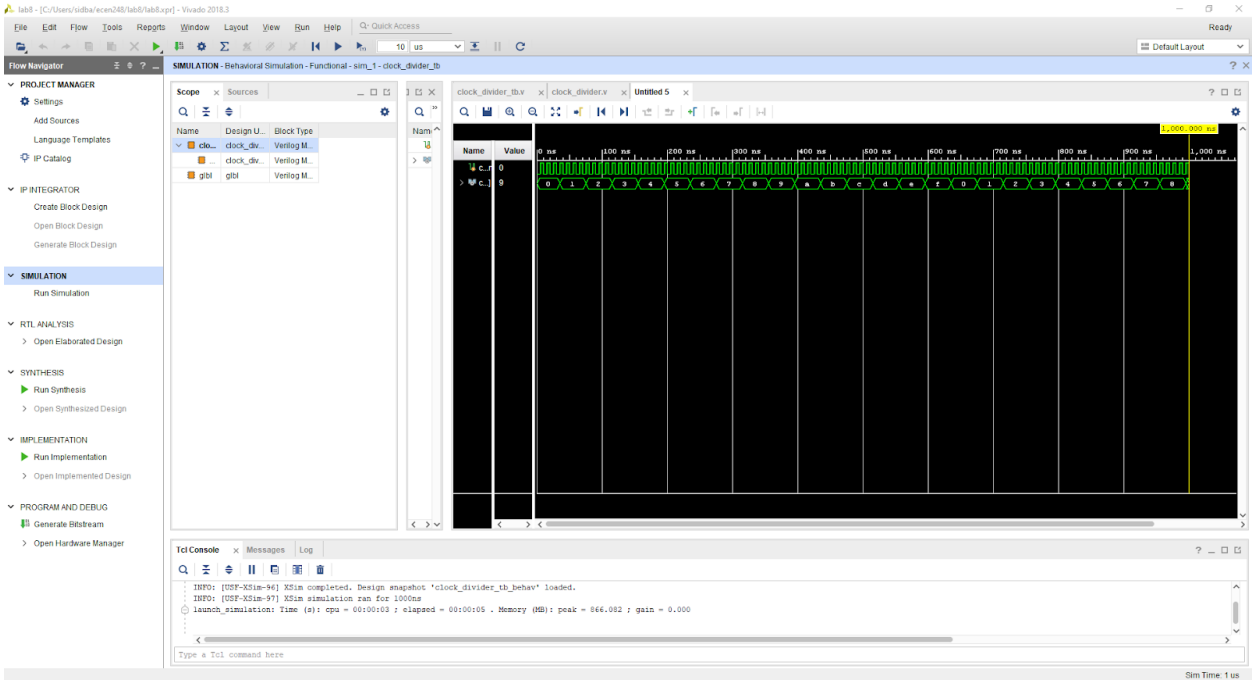
//no debounce



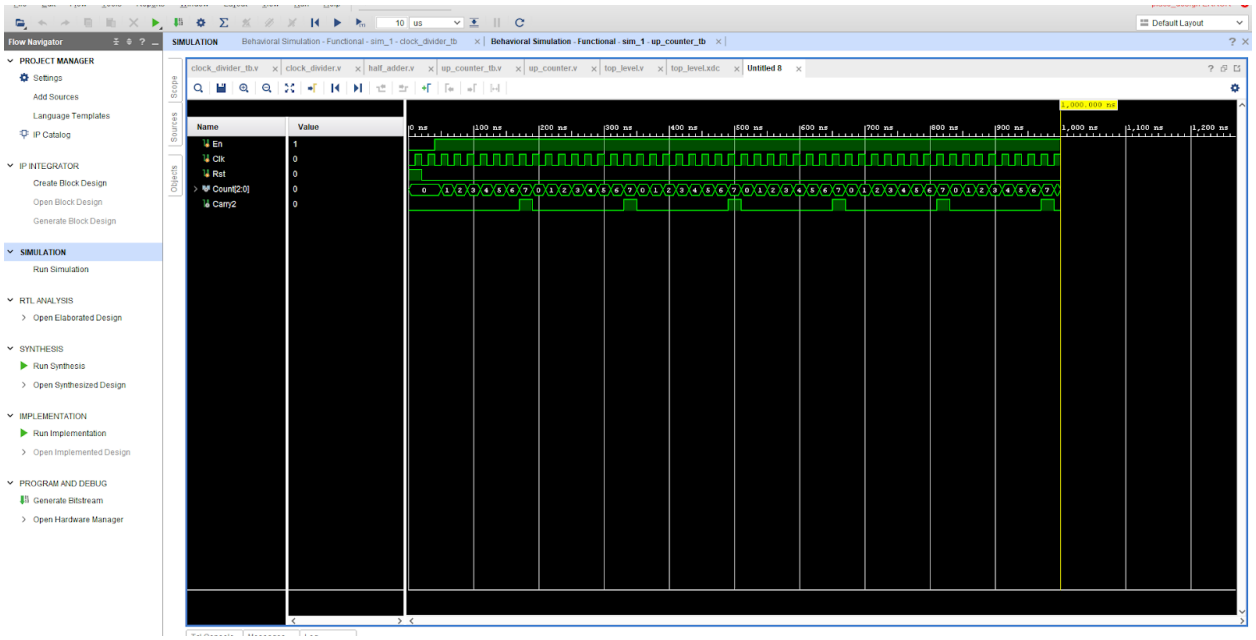


Results

//clock divider



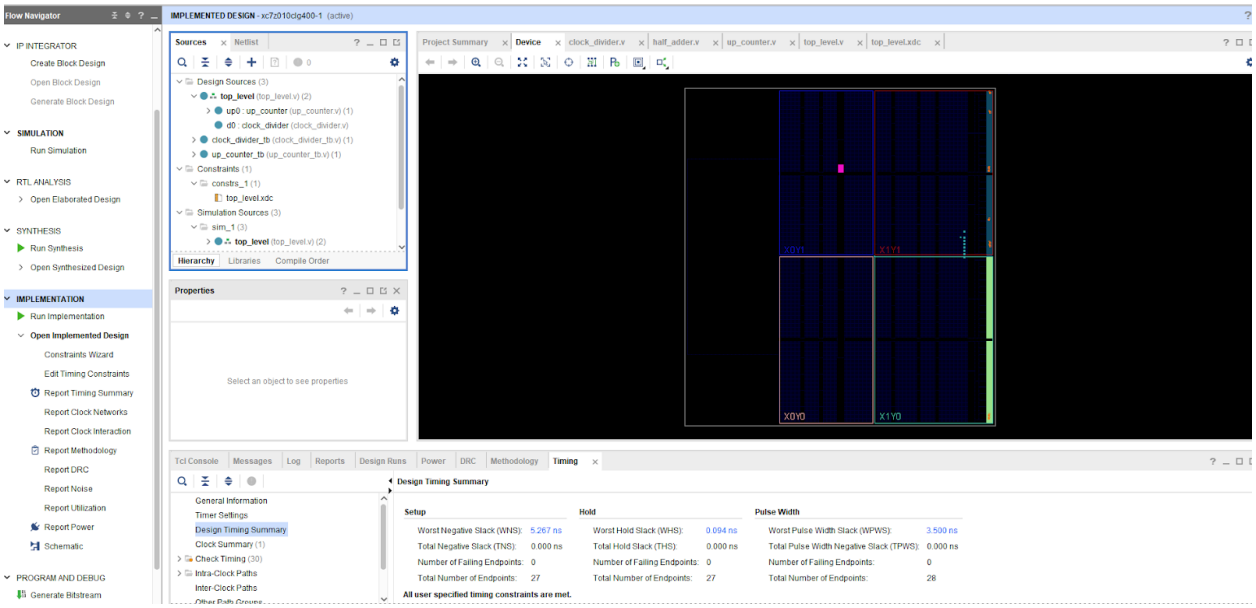
//up counter



```
if { [string length $curr_wave] == 0 } {
  if { [llength [get_objects]] > 0} {
    add_wave /
    set_property needs_save false [current_wave_config]
  } else {
    send_msg_id Add_Wave-1 WARNING "No top level signals found. Simulator will start without a wave window. If you want to open a wave window go to 'File->New Waveform Configuratio
  }
}
run 1000ns
INFO: [USF-XSim-96] XSim completed. Design snapshot 'up_counter_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:02 ; elapsed = 00:00:07 . Memory (MB): peak = 866.082 ; gain = 0.000
```

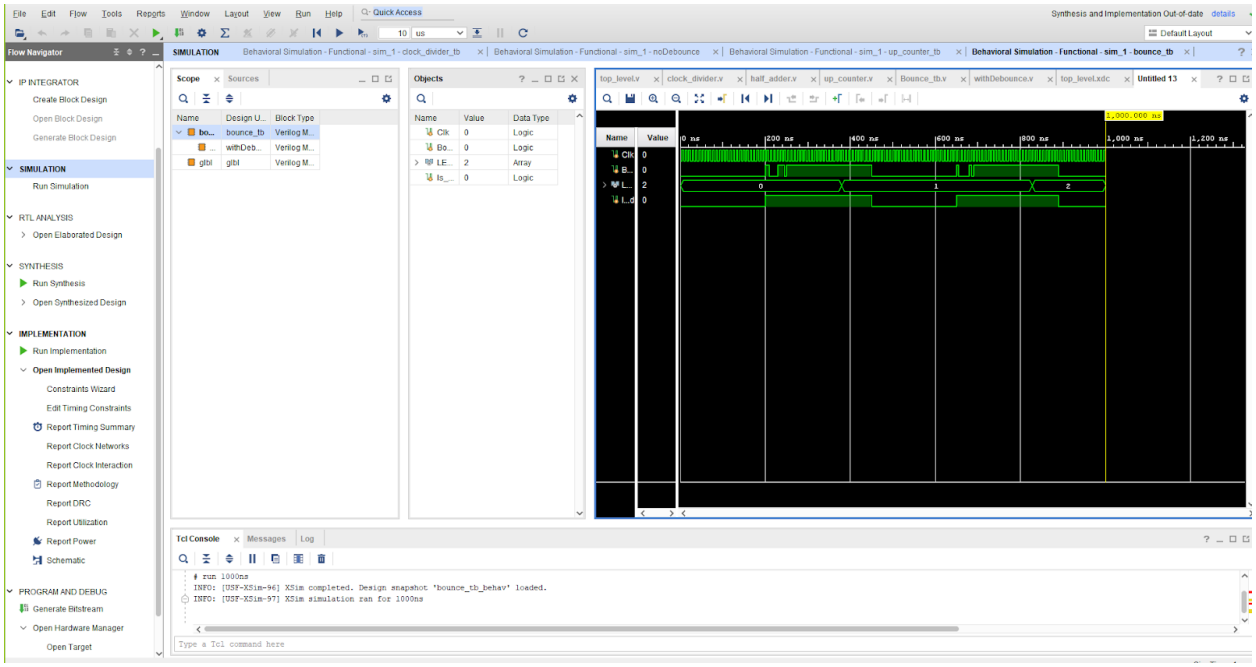
//top level and top level xdc

After synthesize and implementation

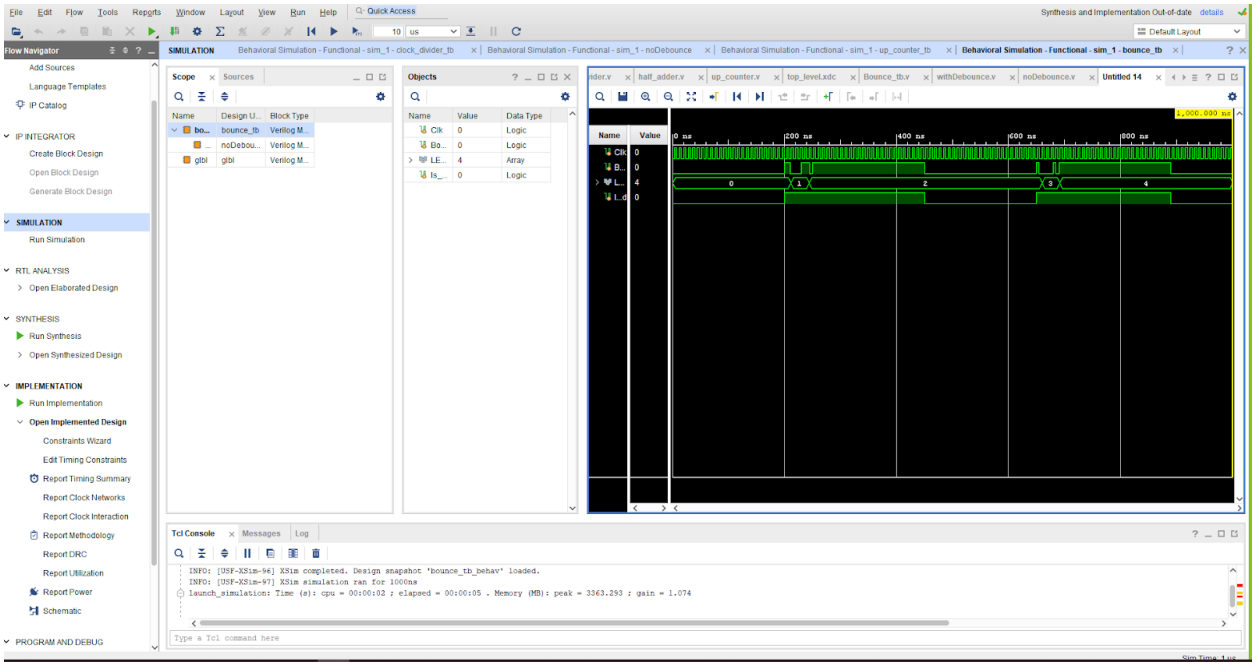


Rest is in the demo

//with debounce



//no debounce



Conclusion

In conclusion, this lab allows students to gain an understanding on counters, clock dividers and debounce. The design section contains all the source codes that are shown for the different tests we were required to do. In the results it contains all the waveforms for source codes like clock divider, up counter, debouncer, etc. The codes compiled successfully. My understanding of the topic overall improved after this hands on performance of the lab.

Post - Lab Questions

1. Include the source code with comments for all modules in the lab. You do not have to include the test bench code. The code without comments will not be accepted!

This is in the design section

2. Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation.

This is in the results section

3. Page 11 2.(b) (c) (d) (f)

(b) The delay of the clock signal is 10 which is 10ns, so the Frequency = 1/10ns = 100Mhz

(c) The up counter the reset is 20ns.

(d) The up-counter holds enable low for 20 ns

(f) The maximum count is 111, and the rollover goes from 1 to 0.

4. Page 11 3.(a)

frequency(f1) is 125 MHz
n = 26
The rate of MSB to oscillate is
F2 = f1/226
F2 = 125Mhz/67108864
F2 = 1.8 MHz
The frequency MSB to oscillate is 1.8 MHz

5. Page 15 1.(b)

It worked as expected because in the simulation noDebounce, the signals have some of the electrical chatter since there were no additional circuits to prevent the chatters.

6. Page 15 2.(a) (b) (c)

(a) It did work as expected because the counter with the debounce have an expected oscillation as it removes the inconsistencies form the no debounce circuits.

(b) The source file with the comment is above in design section

(c) The synchronizer is implemented as the positive clock edge. When the clock edge is positive, and if the reset is enabled then the counter goes to 0, but if the circuit is enabled then the count adds up to 1. The output from the Msb noted with synchronizer outputs for the res, reducing the chatter in the circuit.

Feedback

1. I liked the clear instructions on how to code the counters, clock dividers, etc. I did not like the unclarity on what to demo.
2. The lab manual was clear.
3. I wouldn't change anything about the lab.