ECEN 248: Introduction to Digital Design
Department of Electrical and Computer Engineering
Texas A&M University
Lab Report

Laboratory Exercise #8

Counters, Clock Dividers, and Debounce Circuits

ECEN-248-509

Joaquin Salas

731000141

11-08-2022

TA: Sri Hari Pada Chandanam Kodi

Objectives:

The primary objective of this week's lab assignment is to reinforce our knowledge of sequential circuits by introducing important synchronous sequential circuits known as binary counters. The lab will guide us to design a binary up-counter using familiar combinational components also using sequential components discussed in the last lab that we did. Also, this lab will demonstrate two important use cases for binary counters, namely clock frequency division and I/O debouncing. Due to the nature of this lab, we will be testing all of our designs on an FPGA Board, also creating .xdc files.

Lab Procedures:

The following lab exercises will guide us through the implementation of the circuit, for the first two experiments, we will be introduced to the behavioral description before using structural Verilog.

Experiment Part 1:

First, we will be describing a synchronous sequential circuit in Verilog and load it onto the FPGA board. Second, we want to have hands-on experience dealing with counters and clock frequency division These are important concepts that we will need to understand when we design more complex digital circuits.

1. Write a source file and save it as 'clock_divider.v':

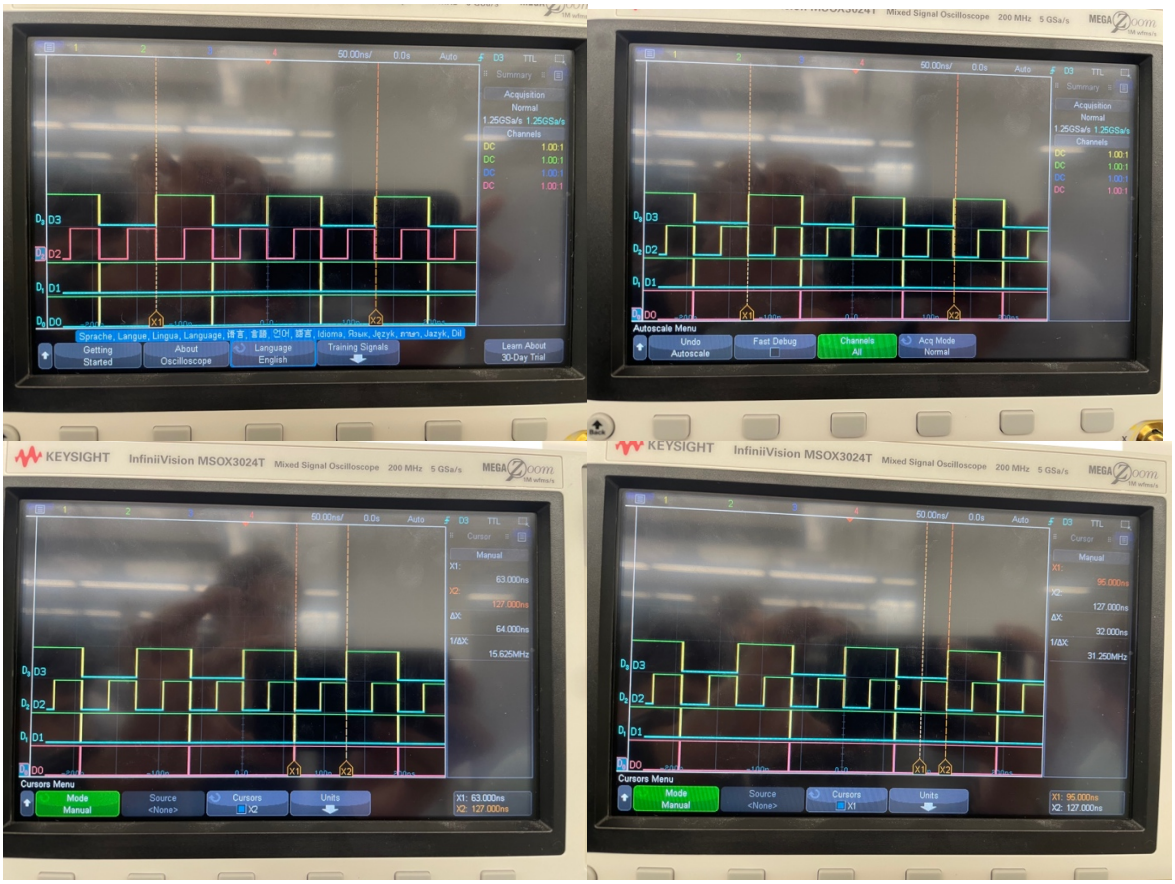
Code: *clock_divider*

```
`timescale 1ns/1ps
`default_nettype

/*This simple module will demonstrate the concept of clock frequency*
*division using a simple counter. We will use behavioral Verilog    *
*for our circuit description.                                       */

module clock_divider(ClkOut, ClkIn);

    output wire [3:0] ClkOut;
    input wire ClkIn; //wires can drive regs

    /*-this is a keyword we have not seen yet! *
    * as the name implies, it is a parameter   *
    * that can be changed at compile time...   */
```

```
    parameter n = 5; //make count 6-bits for now...

    reg [n:0] Count; //count bit width is based on n! how cool is that!

    /* simple behavioral construct to describe a counter...    */
    always@(posedge ClkIn) //should look familiar...
        Count <= Count + 1; //worlds shortest always block

    assign ClkOut[3:0] = Count[n:n-3];

endmodule
```

2. The output of the counter changes rapidly to view with LEDs we will connect the logic analyzer to the workbench. The logic analyzer we will use is integrated in InfiniVision 2000 X-Series Oscilloscope. The logic analyzer can probe up to 16 digital signals at once and can group signals into buses. These waveforms will resemble the ones we have seen in simulation through testbench. Below are the results I gathered from the logic analyzer:

Logic Analyzer: *clock_divider*



Experiment Part 2:

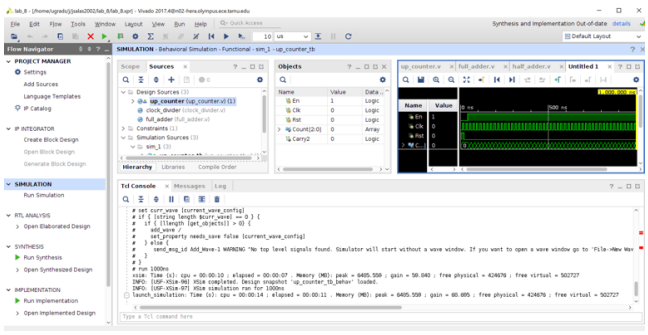
For this part, we will use our learned Verilog skills to design the up-counter discussed from the background section. We will drive an up-counter with the clock divider created in the previous experiment and wire the output of the up-counter to the LEDs on the FPGA board. The En signal will use the BTN0(R18) push button.

1. Design an up-counter.

Code: *up-counter*

```
`timescale 1ns / 1ps //specify 1 ns for each delay unit
`default_nettype none

// this module describes a simple 4-bit up counter using
//  half-adder modules built in the previous step

module up_counter(Count, Carry3, En, Clk, Rst);

    output reg [3:0] Count;
    output wire Carry3;
    input wire En, Rst, Clk;

    //intermediate nets
    wire [3:0] Carry, Sum;

    half_adder halfadder0(Sum[0], Carry[0], En, Count[0]);
    half_adder halfadder0(Sum[1], Carry[1], Carry[0], Count[1]);
    half_adder halfadder0(Sum[2], Carry[2], Carry[1], Count[2]);
    half_adder halfadder0(Sum[3], Carry[3], Carry[2], Count[3]);

    assign Carry3 = Carry[3]; //wire up carry3

    //positive edge triggered flip-flops for count
    //posedge Rst implies asynchronous reset
    always@(posedge Clk or posedge Rst)
        if(Rst)
            Count<=0; //reset the count
        else
            Count<=Sum; //otherwise, latch sum
endmodule
```

2. Now simulate the up-counter and observe the output waveform to ensure the counter is working properly.



3. (a) Create a top-level Verilog module that can be used to drive our counter on the FPGA and load the final design onto the ZYBO Z7-10 board.

Code: *top_level*

```
`timescale 1ns/1ps
`default_nettype

module top_level(LEDs, SWs, BTNs, FastClk);

  //all ports will be wires
  output wire [3:0] LEDs;
  input wire [1:0] SWs, BTNs;
  input wire FastClk;
  //Fast Clk is 1 bit wire
  //intermediate nets
  wire [3:0] Clocks;
  reg SlowClk; //will use an always block for mux
  //behavioral 4 way MUX
  always@(*) //combinational logic
    case(SWs)
      2'b00: SlowClk = Clocks [0];
      2'b01: SlowClk = Clocks [1];
      2'b10: SlowClk = Clocks [2];
      2'b11: SlowClk = Clocks [3];
    endcase
  //use up counter to connect to LEDs
  up_counter COUNTER0(LEDs[2:0], LEDs[3], BTNs[0], SlowClk, BTNs[1]);
  //use clock divider
  clock_divider clk_div0(
    .ClkOut(Clocks),
    .ClkIn(FastClk)
  );
endmodule
```

(b) Create a file called ‘top_level.xdc’ and finish the code provided.

Code: *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_T4
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]
```

Results:

This lab resulted in completing all parts of the lab and connecting the Zybo-10 board to complete all tasks and procedures on the lab, This lab was difficult but we did it.

Conclusion:

This lab gave us a better understanding of synchronous circuits and using a binary counter such as clock frequency and I/O debouncing. We also tested our code with the FPGA.

Post-lab Deliverables:

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

The code is shown above and clearly labeled.

2. Include any XDC files that you wrote or modified.

The XDC files we wrote are shown above and clearly labeled.

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

All waveforms are shown above and clearly labeled.

4. Answer all questions throughout the lab manual.

• Experiment Part 1, 4.a (clock periods and answer the question)

• Experiment Part 2, 2.b
100 MHz
• Experiment Part 2, 2.c
20 ns
• Experiment Part 2, 2.d
20 ns
• Experiment Part 2, 2.f
Max count value is 7. The carry2 signal when its equal to 1 to know when the counter is going to roll over.
• Experiment Part 2, 3.a
Frequency = 1.86 Hz
Time = .5368 s
• Experiment Part 3, 1.b
The noDebounce waveform made the LEDs output random when the button changes
• Experiment Part 3, 2.a
It is in the code
• Experiment Part 3, 2.c
The LED outputs were affected by the no Debounce and debounce operations

Important Student Feedback

1. What did you like most about the lab assignment and why? What did you like least about it and why?

I enjoyed learning the difference between the three types of ways that code is written in Verilog, structural, dataflow, and behavioral, the pre-lab did take me a fair amount of time but I believe it was much worth the work. I did not like how the Zybo-10 board did not work with our computer at the end of the lab.

2. Were there any sections of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?

All parts of the lab manual were very clear, except for the last part on experiment 3 where there was a crucial step that was left out to get the Zybo board to work correctly.

3. What suggestions do you have to improve the overall lab assignment?

I have no suggestions for improving the overall lab assignment.