Texas A&M University

Counters and Clock Dividers

*Author:*

Eden Habtegebrial

April 11, 2017



**Objectives:**

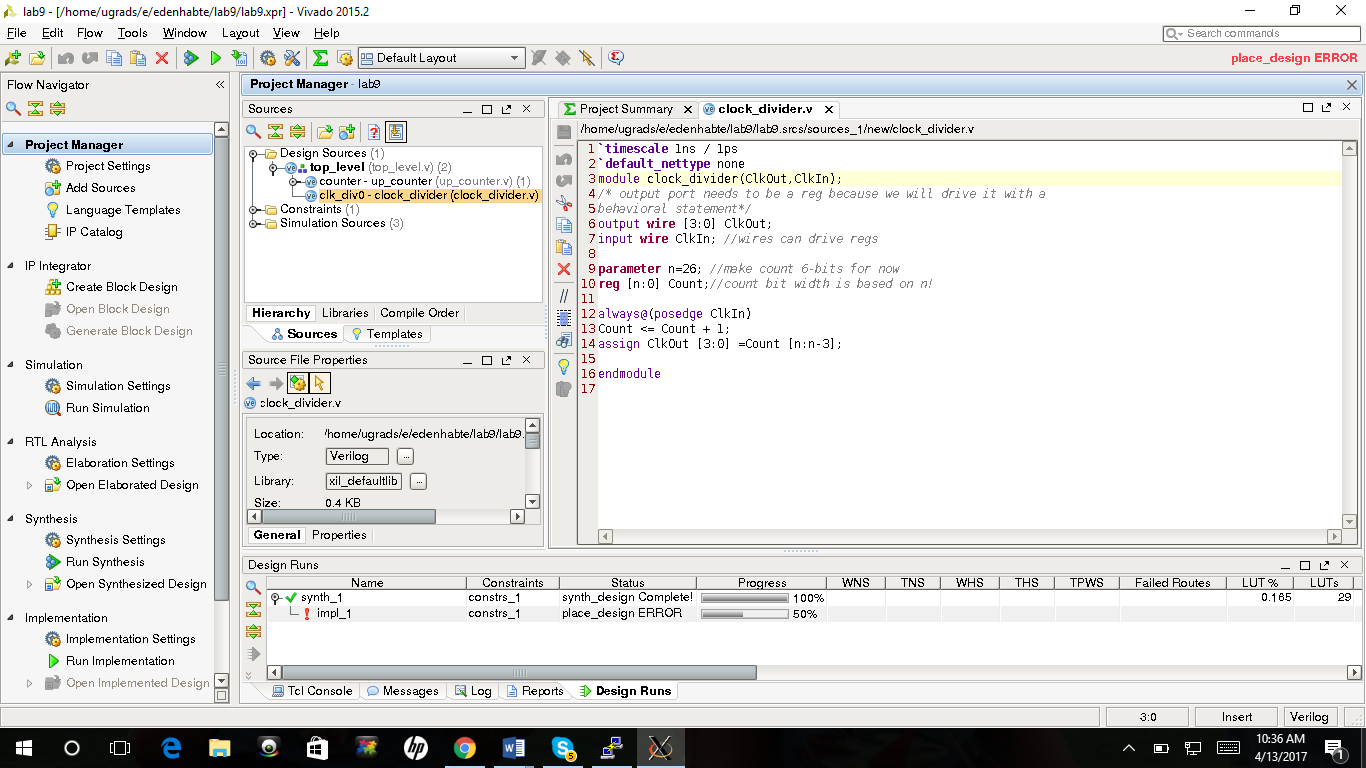
This objective of this lab is to reiterate what we know about sequential circuits by introducing synchronous sequential circuit, clock divider, up-counter and other components. The lab will show us how to properly implement these designs in order to get an effective display.

**Design:**

We utilized the clock divider, half-adder, up-counter, top-level, switch-bounce, noDebounce, and withdebounce designs. By implementing these circuits using Verilog we were able to simplify the process and synthesize the complex code onto the FPGA board and load our final design onto the ZYBO board to see the results.

**Code:**

*Clock Divider*



*Half Adder*

`timescale 1ns / 1ps

`default\_nettype none

//This is a simple Half Adder design

module half\_adder(S, Cout, A, B);

output wire S; //1bit wires

output wire Cout; //1bit wires

input wire A;

input wire B;

assign S = A ^ B;

assign Cout = A & B;

endmodule

*Up Counter*

`timescale 1ns / 1ps

`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);

//Count output needs to be a reg

output reg[3:0] Count; //4bit reg

output wire Carry3;

//inputs are wires

input wire En, Clk, Rst; //1bit wires

//intermediate nets

wire [3:0] Carry, Sum;

//wire up half-adders

half\_adder half0(Sum[0], Carry[0], Count[0], En);

half\_adder half1(Sum[1], Carry[1], Count[1], Carry[0]);

half\_adder half2(Sum[2], Carry[2], Count[2], Carry[1]);

half\_adder half3(Sum[3], Carry[3], Count[3], Carry[2]);

//wire up carry3

assign Carry3 = Carry[3];

//describe the positive edge triggered flip flops for count

//Including "posedge Rst" in the sensitivity list

//implies an asynchronous reset!

always@(posedge Clk or posedge Rst)

if(Rst) //if Rst == 1'b1

Count <= 0; //reset Count

else //otherwise, latch sum

Count <= Sum;

endmodule

*Top-Level*

`timescale 1ns / 1ps

`default\_nettype none

/\*This is the top-level module which wires all of our synchronous

components together. This module does NOT include synchronizers for the inputs

(we will discuss them shortly) so just don't use this in

real application\*/

module top\_level(LEDs, SWs, North, South, FastClk);

//all ports will be wires because we will use

//structural Verilog to wire everything up

output wire [4:0] LEDs;

input wire [1:0] SWs;

input wire North, South, FastClk;

//intermediate nets

wire [3:0] Clocks;

reg SlowClk; //will use an always block for MUX

/\*behavioral description of a MUX that

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 up\_counter

//Hint: if you want to wire a port to just the first 4 bits of a bus,

//you can do something like this: LEDS[3:0]

up\_counter up0(LEDs[3:0], LEDs[4], North, SlowClk, South);

//instantiate clock divider

clock\_divider clk\_div0(

.ClkOut(Clocks),

.ClkIn(FastClk)

);

endmodule