**ECEN 340**

**Lab #6**

Clock Generation and Module Instantiation

**Purpose:**

1. To learn how to design and simulate clock dividers

2. To learn how to port signals to connectors to view with an oscilloscope

3. To learn how to instantiate multiple modules into one project

4. To learn how to trust simulation for sub modules

Overview:

The objective of this lab is to drive all 4 of the seven segment LEDs on the Basys 3 board so that each of them displays a different number. All sixteen DIP switches will be used to generate four hex numbers.

This lab will require enabling each of the 4 seven-segment LED columns one at a time at a very high rate so that it appears as though they are all on at once. If the update rate is faster than 30Hz, the flicker is not detectable to the human eye (Figures 1a and 1b).



Figure 1a – Seven Segment Functional Overview

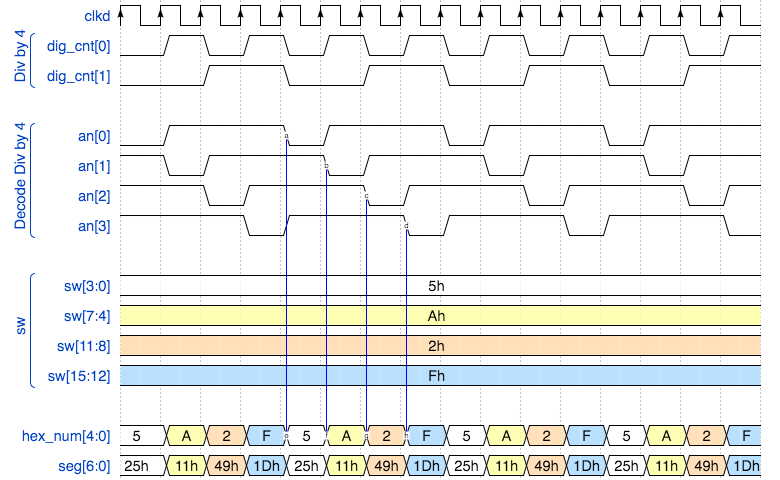


Figure 1b – Seven Segment Timing Overview

Procedure:

Part 1a. Create a new project called “sseg\_x4 \_top”. The ports will be the 16 dip switches (sw), the center pushbutton as a divider reset (btnC), the 7-seg LEDs (seg), all 4 column selectors (an), the decimal point (dp), five of the PMOD JA pins (JA) and the clock input (clk). The flip-flops in the project will all be reset by the middle push-button on the Basys board (btnC). This reset signal will need to be passed to all modules that use flip-flops (Figure 2).



Figure 2 – Block Diagram of Top Level Module creation

Part 1b. As a separate file, write a Verilog module called “clk\_gen”. This clock generator will be implemented as a counter. It will count from 0 to (2^26)-1 (how many bits will this take?). There will be a flip-flop for each bit. Each bit of this counter may be used as a divided clock of a different frequency.

You will use the 100MHz clock as an input to this module, so you will need to pass this clock to this module from your top level module.

If the input clock is 100MHz, what is the frequency of the counter’s lsb? msb?

If you want to update the displays faster than 30Hz but slower than 200Hz which outputs could be used to generate a proper clock frequency? Pick an appropriate output and designate it as your clock output port. It will be used to drive other modules.

Instantiate this module in your top-level module. At this point, you will only have this one module instantiated in your top-level design (Figure 3).



Figure 3 – Block Diagram of Instantiation of “clk\_gen” module

Part 1c. Create a testbench to simulate this module and verify its functionality. At a minimum, the testbench should include signals to drive “clk”, “btnC”, and "sw[15:0]” (Figure 4). Include a screenshot of the simulation results in the report.



Figure 4 – Block Diagram of Test Bench With Top-Level Module Instantiated

Part 1d. As a separate file, write a module called “digit\_selector.” The digit selector will receive the clock from the “clk\_gen” cell, and will output a four-bit signal called “digit\_sel”. This digit selector will decode the output of a 2-bit counter so that only one of four outputs is enabled at a time.

Instantiate the digit\_selector module in your top-level module, and connect it up to your clock\_gen module with port connections (Figure 5).

Simulate with the same test bench to verify that digit selector is functioning correctly.



Figure 5 – Block Diagram of “digit\_selector” instantiation

Part 2a. In your Project Manager, add the Verilog source file you’re your previous seven-segment lab and instantiate your seven-segment module into sseg\_x4\_top (Figure 6). YOU DON’T NEED TO MODIFY THIS! If the name of your seven segment lab were “sseg”, it would look something like this:

wire [3:0] not\_used;

reg [3:0] hex\_num;

// seg and dp are already declared as ports

sseg inst1 (

.seg (seg), // drives the seven LEDs for the display

.an (not\_used), // assigned to 4’b1110, so I am not using it

.dp (dp), // decimal place assigned permanently to 1’b1

.sw (hex\_num) // can’t be called sw any more so I’ll call it hex\_num

);

Notes:

1. This instantiation uses an improved naming convention for the ports. The port names are no longer positional. Instead, we reference the port with the original name preceded with a “.”, and we map the original name to a new name found in the (). The names in the () must be declared if they are not already declared as a port.

2. The LED column selector “an” is a four-bit word. Each bit enables one column. We were only using one column in the last lab, so we permanently tied this to 4’b1110, where the “0” enables the left column. We now want to drive the “an” port with “digit\_sel” from part 1c of this lab.

3. The decimal place “dp” was not used in the last lab so it was set to 1’b1. We are still not using it in this lab, so we can keep the same name.

4. The variable “sw” was only 4 bits wide in the last lab, and was used to generate a hex number with four switches. We are now going to generate the four-bit hex number by rotating through all 16 switches, four at a time. To avoid confusion, we must change the port name to something other than “sw”, like “hex\_num”. Remember that sw will be used on the top level as a 16-bit input.



Figure 6 – Block Diagram of “sseg\_display” Instantiation

Part 2b. Now you are only missing one element of this design. You have a digit selector, but you will need a way to select the corresponding group of four bits from the 16-switch input and write that hex number to the “hex\_num” nibble. It must be in sync with the digit selector. How are you going to do this? Are you going to write a new module or just place it in your digit selector? This will be your call. Figure 7 shows the creation of a new module called “hex\_num\_gen” to accomplish this. “hex\_num\_gen” takes the input from the 16 switches and creates the input (hex\_num) to the “sseg\_display” module. “hex\_num” is four bits wide and it cycles through all 16 switch values four bits at a time.



Figure 7 – Block Diagram of the Complete Hierearchy of the Seven Segment Display Driver

You can use the same test bench from part 1, and modify it as needed to be sure everything is working.

Part 2c. Generate the bitstream and test your design! Use the JA[4:0] pins to view critical waveforms on an oscilloscope.

**Submitting your report:**

You will submit a combined report for both parts of this lab which should include the following:

1) A copy of the Verilog code used to implement the design(s). The Verilog code (including test bench code) should be complete with useful comments (5 pts).

2) Include a brief description of the unique (new) tools, technologies, or methods used to implement this lab (5 pts).

3) The report should be professional quality—meaning it will be neat and use proper English. Include Vivado screen shots of simultations, oscilloscope waveforms, etc., to document your work (5 pts).

4) In your conclusion statement, discuss your results, the method of testing, and include the level of functionality of the lab (10 pts).

Note: To grasp the Lab 6 concepts, you must follow the recommended hierarchy. Expect point reductions for not following the prescribed hierarchy!