Lab 3: Sequential Up/Down Counter

EE478 Fall 2019

Objective

In this lab, you will implement a complex sequential digital system in Behavioral VHDL using process statements. This system will take an input from a slider switch on the board as well as the on board 125MHz clock and will output a 4-bit 2's complement number displayed on four LEDs. The number will increment or decrement once per second based on the state of the slider switch, but the number will be clamped to the maximum or minimum numerical range in 2's complement with 4 bits. A push button input will be used to reset the counter to zero.

Deliverables

Lab start: Beginning of lab during the week of 9/30

Demo: Working four bit up down counter in a testbench showing incrementing and decrementing based on the up/down input signal and resetting to zero based on the reset signal. The testbench may update once per clock cycle. Working counter on the Zybo board, with up/down input driven by a slider switch and output on any consecutive four LEDs. The count should update once per second.

Demo due: End of lab session on during the week of 10/7 (2 week lab)

Design Document due:

- On UBLearns by 11:59PM on 10/18. *Submit one per team*.

Lab Contents

Part A: Up/Down Counter System Requirements

In this lab, you will develop an up/down counter system to be implemented on the Zybo board. This lab description will not cover Vivado usage; see Lab 1 for assistance with project setup, etc. The following requirements must be satisfied by your demo or design document.

1. The counter shall accept one up/down input driven by a slider switch, one reset input driven by a momentary push button, and one clock input connected to the 100MHz clock

source on the Zybo board. The counter shall display a 4-bit signed 2's complement count using four LEDs.

2. The output of the counter shall update once per second based on the following rules:
    a. If the up/down input is logic '0', the counter shall count *down* unless it reaches the most negative value expressible in 4-bit 2's complement, in which case it should remain the same.
    b. If the up/down input is logic '1', the counter shall count *up* unless it reaches the most positive value expressible in 4-bit 2's complement, in which case it should remain the same.
3. The output of the counter shall be reset to zero when a momentary push button on the board is pressed (any button is acceptable).

Appendix A: Using clock sources on the Zybo board

The Zybo board has a 125MHz oscillator circuit that can be used as a clock signal source. The clock signal can be accessed through pin **K17**, and you can connect a clock signal in your design to this pin just like any other input/output pins.

The 125MHz clock signal is too fast to use directly for the 1Hz update rate we need for this lab. As such, we need to divide the clock down to a 1Hz signal. The way to implement this is to use a counter that counts up to 125,000,000. Once the counter reaches this value, a slow clock should be driven high. If the counter is any other value, the slow clock should be low. A separate process can be used for this, sensitive to the main 125MHz clock. Then, the internally generated slow clock can be used to trigger a second process that implements the 4-bit signed counter connected to the LEDs. Because we want to reserve the rising_edge function for actual clock signals, you should implement this as a **clock enable**:

```
count_proc : process(clk)
begin
    if rising_edge(clk) then
        if slow_clk = '1' then
            --Fill in the rest of the logic here
```

Appendix B: Outputs from processes

Outputs from modules cannot be read by other logic. Best practice is therefore to use an internal signal for all outputs from processes, make all assignments to that signal, and then connect that signal through to the output with a concurrent statement.

For example, in this lab, suppose we have a 4-bit vector output called *leds*. Our process will assign to an internal version of this signal, say, *leds_int* which we can make *signed* for convenience. Then we will use a concurrent statement to assign to the actual output:

```
count_proc : process(clk)
begin
    if rising_edge(clk) then
        if slow_clk = '1' then
            --Fill in the rest of the logic here
            --Assign to leds_int

        end if;
    end if;
end process count_proc;

leds <= std_logic_vector(leds_int);
```

Appendix C: Simulating with clock signals

You will need to create a clock signal in your testbench when you simulate your sequential designs. The code below will create a clock signal with an 8ns period (a 125MHz signal). *Make sure you initialize clk to '0' when declared or this will not work!*

```
clk_proc : process
begin
    wait for 4 ns;
    clk <= not clk;
end process clk_proc;
```

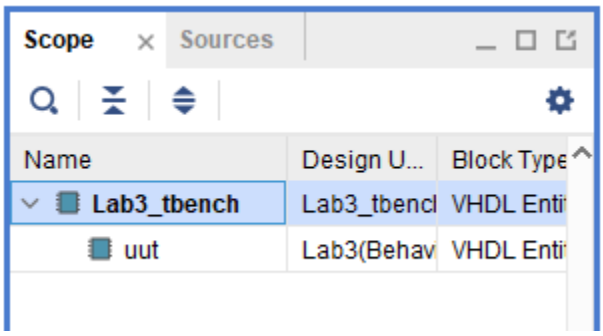
For synthesis, you need to use the slow clock, which divides the main clock by a factor of 125,000,000. However, simulating one second can be time consuming, so you can temporarily adjust your terminal count to 10 for simulation. You should then see the count change every 10 clock cycles.

Appendix D: Viewing internal signals

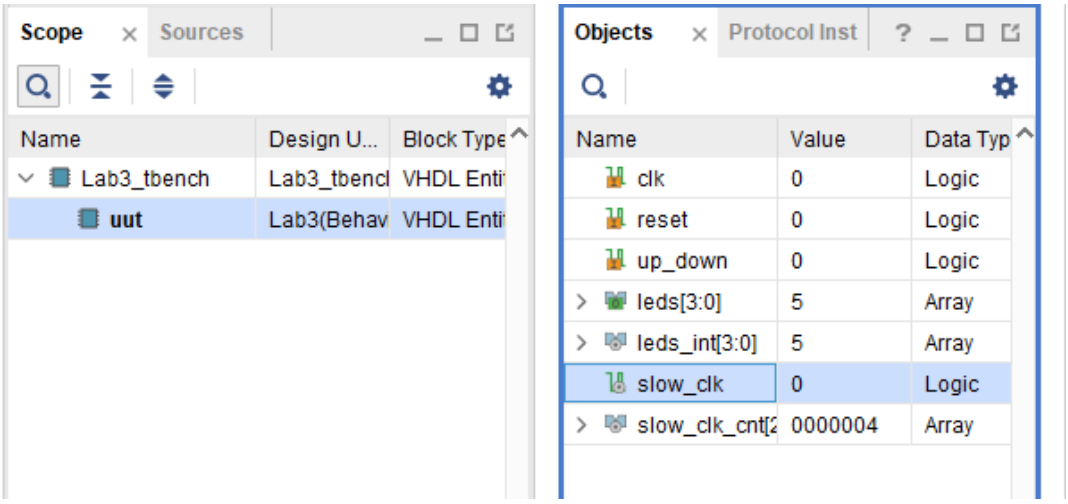
As mentioned in Lab2, by default, XSim plots input and output signals to the UUT module. We can plot any signal in the design, including internal signals by adding them to the waveform view and then rerunning the simulation.

Let's look at how we would plot the slow clock signal for Lab 3. I called the slow clock signal *slow_clk*, and for simulation I reset it every 10 cycles of the main clock (for synthesis we need to make sure to change that back to 125,000,000 so that we actually get the 1Hz slow clock).

To add the internal signal to the design, find the pane on the left of the XSim GUI that says "Scope", and press the arrow/plus sign next to the name of your testbench (I called mine *Lab3_tbench*)



Next, click on *uut* (or whatever you called your UUT instance), then look to the right under *Objects*, and find the signal you would like to view.



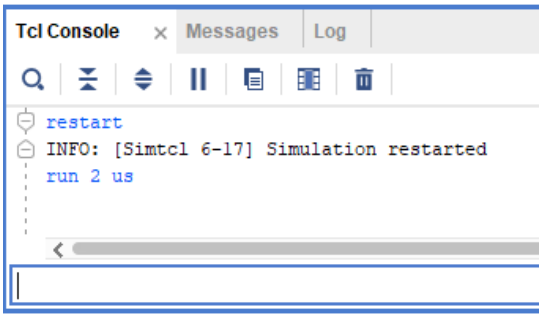
Drag the signal over from the Objects list to the waveform display. It should appear on the waveform signal list, but without a plot trace.

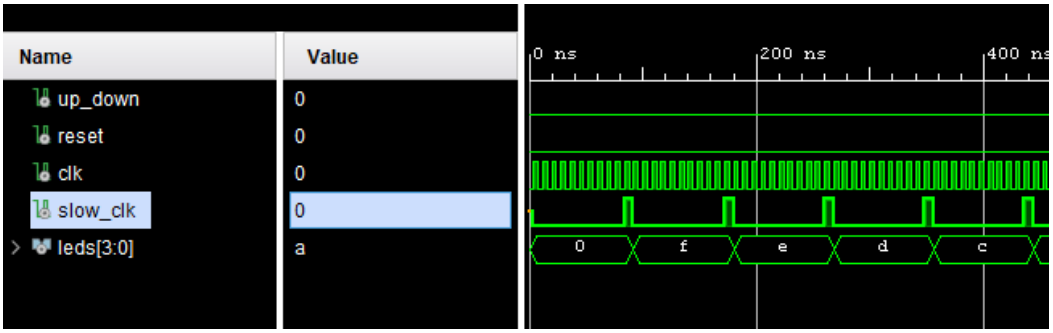
Now, in the Tcl Console at the bottom, enter the command “restart”, followed by a run command. For example

restart

run 2 us

You should now see your internal signal plotted alongside the I/O signals on the waveform display.





Note that slow_clk is a one-shot signal that triggers every 10 counts of the main clock.