EELE 367 – Logic Design
Lab #7 – Precision Clock Divider & BCD Counter

Overview

This lab will give more practice modeling counters in VHDL using single processes. In the first part of the lab, you will create a precision clock divider that is capable of outputting four different clock frequencies: 1 Hz, 10 Hz, 100 Hz, and 1 kHz. This will be accomplished using a counter process and selectable range checking. In the second part, you will be designing a 4-symbol, binary coded decimal (BCD) counter that will be driven to the four character displays on the I/O shield. This will be accomplished using four separate, but interdependent processes.

Outcomes

After completing this lab you should be able to:

- Create a precise timing event using a counter modeled with a VHDL process and range checking.
- Create a single symbol BCD counter using a VHDL process and range checking.
- Create the higher order symbols of a BCD counter using VHDL processes that are dependent on the values of prior BCD symbol values.

Deliverables

The deliverable(s) for this lab are as follows:

- Pre-lab: n/a.
- Demonstration of a precision clock divider (30%).
- Measurement of the frequency of the output of the precision clock divider (10%).
- Demonstration of a four symbol BCD counter shown on the four character displays of the I/O shield (50%).
- Uploading your top.vhd file for this lab to the course DropBox (10%).

Lab Work & Demonstration

Part 1 – Precision Clock Divider (30% + 10%)

A. Create a New Quartus II Project

You are going to create a new Quartus II project for this lab. Create a new folder called "Lab07_Precision_Clock_Divider_n_BCD_Counter". Open one of your prior labs and copy it into your new project folder. You can choose an appropriate lab to copy. Remember that you will need your decoder_7seg_4in.vhd component. Open the new project.

B. Creating a Precision Clock Divider

Our prior labs have been using a selectable, 2^n, clock divider (clock_div_2ton.vhd) to create a slower version of the 50 MHz clock from the DE0-nano board. The divided down clock is suitable for clocking logic slow enough for the human eye to see. This 2^n clock divider was created using the outputs of a ripple counter. The disadvantage of this approach is that the clock frequencies available are only powers of 2. Using this approach it is difficult to get an exact clock frequency such as 1 Hz or 10 Hz. The output frequencies available are *only* powers of 2 and thus you can only get so close to a desired frequency.

Another approach to creating a clock divider is using a single process counter and range checking. We can create precise timing events by simply counting up to a certain value and then setting the counter back to zero. Each time the counter reaches its maximum range, we can perform a task such as toggling a bit. This allows us to create timing events that have a precision of +/- ½ T of the incoming clock.

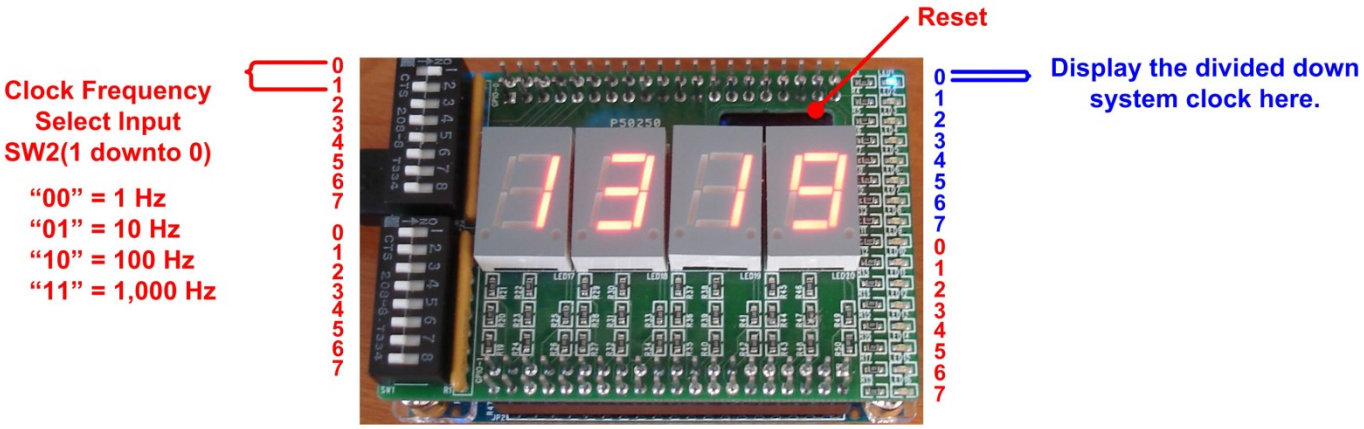
As an example, let's say we wanted to create a divided down clock with a period of 200 ns. We know that the incoming clock to the system is 50MHz (T=20ns). If we create a counter based on this clock, it will increment every 20ns. If we create a counter that increments up to 5 and then rolls over, the roll over event will occur every 5*20ns=100ns. Each time the counter reaches its maximum value and needs to be manually set back to zero, we can also have it complement a signal that will be used as the divided down clock output. This results in a HIGH time of 100ns, a LOW time of 100ns, and an overall period of 200ns.

In this part of the lab, you are going to create a precision clock divider that outputs four different clock frequencies: (1 Hz, 10 Hz, 100 Hz, and 1 kHz). Your divider will have two select lines that choose which clock it outputs. You will create the counter using a single process with range checking. Implement your divider as a subsystem that will be implemented in your top.vhd. The entity definition for your system is as follows:

```
entity clock_div_prec is
   port (Clock_in  : in  std_logic;
         Reset     : in  std_logic;
         Sel_in    : in  std_logic_vector (1 downto 0);
         Clock_out : out std_logic);
end entity;
```

HINT: Is it possible to have the range that you are going to check against be a signal instead of a hard coded value? If it is, then you can change the value that this signal has based on the input select lines (Sel_in). This will allow you to select the range that will be checked against in real time.

Instantiate your clock divider in your top.vhd. The input to your clock is the 50MHz system clock on the DE0-nano board. The reset to your divider will come from the push button on the DE0-nano. Use the SW2(1:0) inputs to drive the select lines of your divider. Display the divided output clock on LED_Blue(0). Use the select settings in the following figure for the appropriate output frequency.



C. Synthesize and Test Your Counter

Implement and download your design to the DE0-nano board. You should be able to see the clock toggling on the LED_Blue(0) for the 1 Hz and 10 Hz frequencies.

D. Measure the Frequency using the Logic Analyzer

Connect the logic analyzer to the LED_Blue(0) pin of the header on the I/O shield. Setup the logic analyzer to take a measurement on the divided down clock.

DEMO

E. Demonstrate the proper operation of your clock divider (30%) and show that its frequency is precise for all four select settings using the logic analyzer measurement (10%).

Part 2 – Creating a 4-Symbol BCD Counter (50% + 10%)

A. Create the First Character of the BCD Counter

A BCD code is a 4-bit binary code that represents the decimal symbols from 0 to 9. This means a BCD counter will never increment above "1001" (e.g., decimal 9). A BCD counter must continually check if it has reached 9. If it has, it must reset the BCD code to 0. The first symbol of a BCD counter can be easily created in VHDL using a single process and range checking.

Create a BCD counter using a single process in your top.vhd. Use its outputs to drive the LED20 character display on the I/O shield. Note that the output of the BCD counter is simply 4-bits that can be driven directly into your 7-segment decoder. You should clock your counter off of the divided down clock coming out of your precision clock divider.

At this point, it is a good idea to compile, synthesize, implement, and test your design. You should see a counter that goes from 0 to 9 and then starts back at 0. Set your clock frequency to 1 Hz so that you can see the counter easily.

B. Create the Remaining Characters of the 4-symbol BCD Counter

One of the challenges of a multi-symbol BCD counter is that higher order symbols don't simply increment on each clock like the least significant symbol. For example, consider the symbol in the 10's position of the 4-symbol counter. It will only increment from a 0 to a 1 once the symbol in the 1's position reaches 9 and rolls over (i.e., 08, 09, 10). This means that when implementing the 10's position counter, its process must include logic to look at the 1's position count value. When looking at any input using a process, the input should NEVER be included in the sensitivity list. Only clock and reset are in the sensitivity list for synchronous systems. Instead, the count value from the 1's position is used as an additional if/then clause within the clock-synchronous portion of the process.

To continue this example, consider the symbol in the 100's position. It will only increment from a 0 to a 1 once BOTH the 10's position and the 1's position reach 9 and roll over (i.e., 098, 099, 100). This means that when implementing the 100's position counter, its process must include logic to look at the 10's and 1's position count values.

Create the BCD counters for the symbols in the 10's, 100's, and 1000's position for the 4-symbol counter. Implement and test your design.

DEMO

C. Demonstrate the proper operation of your 4-symbol BCD counter (50%)

D. Upload your top.vhd to the Lab #7 DropBox (10%)