EELE 367 - Introduction to Logic Circuits

Lab 9(d) – BCD Counters and a Precision Clock Divider

Objective

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 6-symbol, binary coded decimal (BCD) counter that will be driven to the six character displays on the DE0-CV board. This will be accomplished using six 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 multi-symbol BCD counter using interdependent VHDL processes.

Deliverables

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

- Demonstration of your precision clock divider by measuring the frequency using the logic analyzer (45%).
- Demonstration of your 6-symbol, BCD counter displayed on the character displays of the DE0-CV FPGA board (45%).
- Upload your top.vhd file to the lab DropBox (10%).

Lab Work & Demonstration

Part 1 – 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-CV 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. You 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, you can perform a task such as toggling a bit. This allows you to create timing events that have a precision of +/- ½ T of the incoming clock.

As an example, let's say you 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 you create a counter based on this clock, it will increment every 20ns. If you 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, you 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 (SW(1) and SW(0)) that choose which clock is used in the system. You will create the counter using a single process with range checking. You will implement your divider as a subsystem (clock_div_prec.vhd) that will be instantiated in your top.vhd.

A) Create a new Quartus project by copying your one of your prior labs. Name it "Lab9d_BCD_Counters_n_Precision_Clock_Div".

B) Create the precision clock divider that outputs clock frequencies of 1 Hz, 10 Hz, 100 Hz, and 1 kHz. Use the following entity definition.

```
entity clock_div_prec is
   port (Clock_in  : in  std_logic;
         Reset     : in  std_logic;
         Sel       : 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). This will allow you to select the range that will be checked against in real time.

C) Instantiate your clock_div_prec.vhd in your top.vhd file. The input to your clock is the 50MHz crystal on the DE0-CV board. The reset to your divider will come from the "FPGA_RESET" push button on the DE0-CV. Use the SW(1) and SW(0) inputs to drive the select lines of your divider. You should route the divided down clock to the GPIO_0(0) pin and also the LEDR(0) for this part of the lab.

D) Compile your design. Fix any Errors.

E) Download and test your design. Fix any Errors.

F) Measure the frequency of your divided clock with the logic analyzer. Connect channel 0 of the Analog Discovery logic analyzer to GPIO_0(0). Show that you are successfully achieving clock frequencies of 1 Hz, 10 Hz, 100 Hz, and 1 kHz by measuring the frequency using HotTrack within the *Waveforms* application.

G) Demonstrate your design.

DEMO – Show the lab instructor your precision clock divider operation and logic analyzer measurement.

Part 2 – Creating a 6-Symbol 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.

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 6-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.

In this part of the lab, you are going to create a 6-symbol, BCD counter that will be displayed on the HEX characters of the DE0-CV board. You will accomplish this using six separate processes. Each subsequent symbol process will need to consider all of its preceding symbol processes.

A) In your top.vhd, create the six processes to implement the BCD counter. It is a good idea to get the least significant symbol working first, compile, download and test. Once it is operational, move onto the next higher order symbol. Each 4-bit BCD code should be driven through your char_decoder.vhd subsystem in order to drive the HEX displays on the DE0-CV board.

B) Compile your design. Fix any Errors.

C) Download and test your design. Fix any Errors.

D) Demonstrate your design.

DEMO – Show the lab instructor your BCD counter. Once checked off, upload your top.vhd file to the lab DropBox..

Lab Grading

Demo 1 – Proper operation of your precision clock divider	____________	**/ 45**
Demo 2 – Proper operation of your BCD counter	____________	**/ 45**
Review of your top.vhd file	____________	**/ 10** (uploaded to DropBox)
Total	____________	**/ 100**

Post Lab Survey (Just for your own knowledge. This is not graded)

1) Can you create a precision clock divider in VHDL using a process and range checking?
2) Can you create a multi-symbol, BCD counter using interdependent processes in VHDL?