Lab 7: FPGA/VHDL Exercises

UC Davis Physics 116B
Rev 2/26/2019

In this lab, you will expand your VHDL/FPGA skills by

- working with numbers and simple arithmetic in an up/down counter.
- modularizing code by instantiating VHDL modules in other VHDL modules.
- driving your circuit with a high speed external clock.

Reasonable familiarity with the designed tools will be assumed. You may want to use last week's lab as a reference.

8-bit Counter

Start the ISE design tools and open an new project. Use Right-click→"New source file..."→"New VHDL Module" to create a module called "counter" with the following inputs and outputs

```
entity counter is
    Port ( CLK : in  STD_LOGIC;
           EN : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           UPDOWN : in  STD_LOGIC;
           Q : out  STD_LOGIC_VECTOR (7 downto 0));
end counter;
```

All inputs are synchronous; that is, nothing will occur until there is a rising clock edge, at which time:

- If $RESET$ is asserted, all Q bits will be set to 0. Otherwise
- If EN is asserted, then

  if $UPDOWN =' 1'$, Q will increment by 1

  if $UPDOWN =' 0'$, Q will decrement by 1
- Otherwise do nothing.

As we discussed in class, you need to explicitly convert between integers and signal vectors. The easiest way to do this is to create an internal variable (between "process" and "begin")

```
variable count : integer range 0 to 255 :=0;
```

Use this in your calculations, then convert it to a logic vector at then end with

```
Q <= std_logic_vector(to_unsigned(count,8));
```

Note that in order to use these conversion functions, you'll need to uncomment the line

```
use IEEE.NUMERIC_STD.ALL;
```

at the top of the code.

Add a "pins.ucf" user constraints file to map the inputs and outputs as follows

- CLK→ Switch 3
- EN→ Switch 0
- RESET→ Switch 1
- UPDOWN→ Switch 2
- Q bits → LEDs[1]

As you learned last week, the switches are not tied to dedicated clock lines, so you'll also need to add the line

```
NET "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
```

to your .ucf file to generate a warning instead of a fatal error.

Compile your code, generate a .bit file, and download it to the board like you did last week. Check the pinout report to verify that the pins mapped correctly.

Verify that the circuit functions as intended and counts properly in both directions.

The signal from the switch is a bit "bouncy" and may increment or decrement by more than one count when you toggle it. This is OK.

Once you have the "counter" module working, you should not need to modify it for the remainder of the lab.

Instantiating the counter

In this section, we will "instantiate" the counter module inside another module. This is equivalent to adding a discrete component to a schematic.

Create the master module by defining a new VHDL module called "lab7" with the following inputs and outputs

[1]The UCF file has a different syntax than VHDL. To tie one element of a bus to a pin, use, e.g. "NET "LED<0>" LOC = ...".

```
entity Lab7 is
    Port ( ENABLE : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           UPDOWN : in  STD_LOGIC;
           CLK : in  STD_LOGIC;
          LED : out  STD_LOGIC_VECTOR (7 downto 0));
end Lab7;
```

As with programming languages, the VHDL compiler does the first pass compilation on each module independently, so if we are going to instantiate " counter " or any other module, we have to tell this module it exists and what sort of inputs and outputs it wants. This is done by placing "component" blocks between "architecture" and "begin". In this case

```
component counter port (
         CLK : in STD_LOGIC;
         EN : in STD_LOGIC;
         RESET : in STD_LOGIC;
         UPDOWN : in STD_LOGIC;
         Q : out STD_LOGIC_VECTOR (7 downto 0));
end component;
```

This is a little like prototyping in C or C++, but unlike those languages, both the order *and* the names of the signals must match those in your `counter.vhd` file.

Now you can instantiate the "counter" component by adding

`C:counter port map ( .... )`

after the "begin" of the architecture block, with the correct connections to the inputs of "lab7" connected to the inputs of "counter" between the parentheses.

Edit the names in your "pins.ucf" file to match the inputs of "lab7" where they are different than in the previous section.

We have to insure that both "counter" and "pins.ucf" are both associated with "lab7". The easiest way to do this is to right-click→"Remove" them, and then right-click→"Add source..." them to "lab7".

Compile the program, generate a bit file, and download it to the board.

Verify that the board behaves in the same way as it did in the first section.

Clock divider

In the second section, we just used a more complicated way to do the same thing we did in the first section; however, we would also be able to instantiate other components (or multiple instances of "counter", as well as to introduce additional functionality.

In this section, we will drive the circuit from the onboard 50 MHz clock, but we will reduce this to 1 Hz by dividing it by 50 million. You can do this by introducing a counter that increments every clock cycle, and only asserting the $ENABLE$ bit when it hits 50 million.

Change the CLK input from Switch 3 to the 50 MHz onboard clock (C9). Compile and load this configuration. Verify that the counter counts at 1 Hz. Verify the operation of the $ENABLE$, $RESET$, and $UPDOWN$ functions of the first three switches.

Two Speed Counter

Add another input to your "lab7" module

```
FAST: in STD_LOGIC;
```

and tie this to Switch 3 in the "pins.ucf" file.

Modify your process such that when the $FAST$ input is '1', the counter will count at 10 Hz instead of 1 Hz (hint: test on a different value of the delay counter in this case).

Compile and load this configuration. Verify that toggling Switch 3 changes the counting rate and also that the functionality of the other three switches is preserved.