ECEN2350 Fall 2020

Jumpstart for Project2 – Clock Divider

One of the modules you need to create for Project2 is a clock divider module. We haven't talked about clock dividers yet in class, but as you will see, a clock divider is nothing more than a counter.

Consider an 8 bit up counter, with a 10Mhz input clock. If you connect a frequency counter to the least significant bit of the counter, the frequency counter would indicate 5Mhz. The next bit would measure 2.5Mhz, then 1.25Mhz, and so on. Each bit of the counter increments at half the rate of the previous bit. If you need to divide a clock by a factor that is a power of 2, a counter will work.

If you want to divide by a factor that is a random integer value, you need to use the counter to count to a certain value, then invert your clock signal each time the factor/2 is reached.

In Project2 you need to create a 2Hz clock and also a 5Hz clock. The input clock you should use comes from the DE10-Lite board, and is called ADC_CLK_10. You will need to add the following lines to your Quartus project file (.qpf) assuming you don't have ADC_CLK_10 in your file already:

```
set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to ADC_CLK_10
set_location_assignment PIN_N5 -to ADC_CLK_10
```

and include ADC_CLK_10 as an input port in your top level Verilog file.

How does one create a clock divider? First, determine the divide ratio needed in the clock divider. If you want to create a 2Hz clock from a 10Mhz clock, you need to divide the input clock by 5,000,000. You will need to create a counter that has enough bits to hold the divide ratio / 2, or 2,500,000. Quick math says that 20 bits can hold a number just over a million, so 22 bits should be able to hold a value of 2,500,000. It doesn't matter if you create a divide_by counter that is too wide, the compiler will simply optimize away any unneeded registers. However, if you build the counter with too few bits, you will never reach the desired count to value and your clock divider will not function properly.

Let's start building the divider module.

```
module clock_divider (input clock_in, reset_n, output reg clock_out);
reg [22:0] clock_divider;
parameter divide_by = 0;
```

Notice the parameter divide_by. Since you will need to create two clock dividers in the project (to create the 2Hz and 5Hz clocks), use of the divide_by parameter will allow you to set the divide ratio when you instantiate the clock_divider module (this will be shown shortly). By setting the divide_by parameter to 0, this will result in an error if you forget to set the divide_by value when you instantiate the module.

The operation of the clock divider is simple. You increment clock divider until it reaches the divide_by value. When the divide_by value is reached, you toggle the output clock, and reset the clock_divider back to 0.

I suggest that the clock divider module is the first module you create and test.

```
always @ (posedge clock_in, negedge reset_n)
    begin
        if (~reset_n)
            begin
                clock_out = 0;
                clock_divider = 0;
            end
        else       // test clock_divider value and increment if not done
            begin
                if (clock_divider != divide_by - 1)
                    clock_divider <= clock_divider + 1;
                else
                    begin
                    // fill in the last few lines yourself

                    end
            end
    end
endmodule
```

How do you set the divide_by value when you instantiate the module?

```
clock_divider #(2_500_000) U0 (.clock_in(), .reset_n(), .clock_out());
```