Experiment #2 - Clock Adjusting and Monitoring

Borna
Tavassoli,
810198374

***Keywords*— clock, monitor, set period, noise eliminator, frequency divider, clock adjusting, min/max bound**

I. Introduction

In this lab, we will be working with unsteady clock signals.

First, we learn to put a frequency regulator along with our previously built frequency divider. In the second step, we need to add a clock monitoring unit to generate our "SetPeriod" signal. Finally, after connecting said modules, we'll add a Noise Eliminator to take care of possible noises.

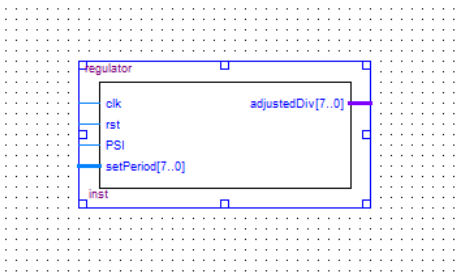
II. Clock Adjusting Unit

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

A. *Frequency Regulator*

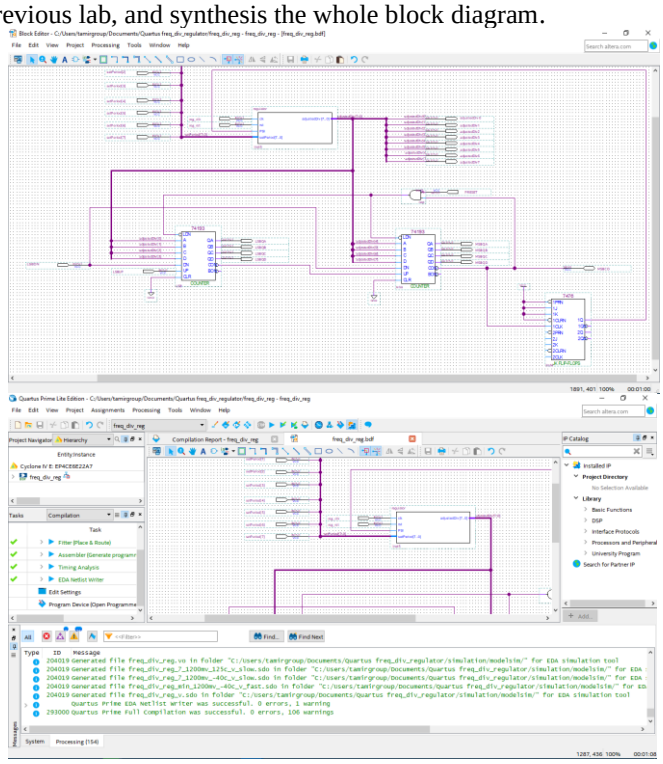
```
module regulator(input clk, rst, PSI, input[7:0] setPeriod, output reg [7:0] adjustedDiv);
        reg PSI_CURR, flag_fall, flag_rise, flag_done;
        reg [15:0] data;
        always @(posedge clk, posedge rst) begin: do_something
                if(rst)
                        PSI_CURR <= 1'b0;
                else begin
                        case({PSI_CURR, PSI})
                                2'b01: begin data <= 8'b00000000; end
                                2'b11: begin data <= data + 1; end
                        endcase
                        PSI_CURR <= PSI;
                end
        end
        always @(PSI) begin: comparison
                {flag_rise, flag_fall} <= 2'b00;
                if(PSI_CURR && ~PSI) begin: hey
                        if (data > setPeriod)
                                flag_rise <= 1'b1;
                        else if (data < setPeriod)
                                flag_fall <= 1'b1;
                        else
                                flag_done <= 1'b1;
                end
        end
         always @(posedge clk, posedge rst) begin: inc_dec
                 if(rst)
                         adjustedDiv <= 8'd127;
                 else if(PSI_CURR && ~PSI) begin
                         if (flag_fall)
                                 adjustedDiv <= adjustedDiv - 1;
                         else if (flag_rise)
                                 adjustedDiv <= adjustedDiv + 1;
                 end
         end


endmodule
```

We synthesis the above code as a top-lvl entity and create its symbol:



B. *Frequency Divider Unit*

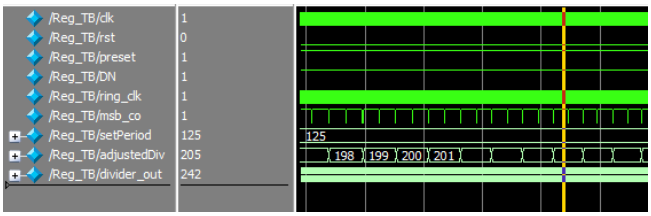
We add the regulator to the Frequency Divider from the previous lab, and synthesis the whole block diagram.



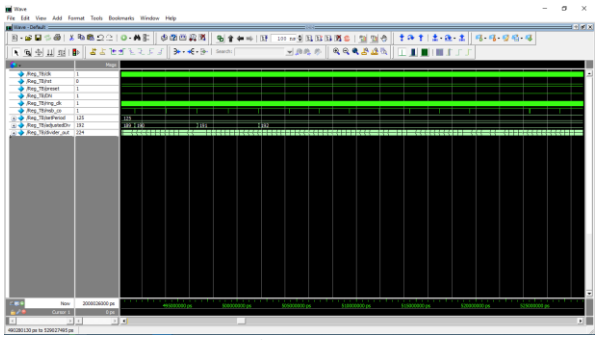
C. *Simulation in Modelsim*

Having generated the ".vo" files, we now test the whole circuit in Modelsim with two different clocks.

In the first test, Ring Oscillator frequency is set to 20MHz, Desired Frequency is 400kHz, Initial parallel load is 127 and set period is 125; As it is shown in the picture below, in the end, the final parallel load becomes 205 and that's because the clock needs to change 50 times (255 – 205) because our original frequency is $(20*10^6) / (400*10^3) = 50$ times greater than the desired frequency. Note that the duration for this test was 0.002 seconds.



In the second test, Ring Oscillator frequency is set to 16MHz, Desired Frequency is 400kHz, Initial parallel load is 127 and set period is 125; As it is shown in the picture below, in the end, the final parallel load becomes 192 and that's because the clock needs to change 62.5 times. Note that our original frequency is $(25*10^6) / (400*10^3) = 62.5$ times greater than the desired frequency. So the clock needs to alternate for 62 times to keep our frequency as close as possible to the desired frequency. Also note that the duration for this test was 0.002 seconds.



Zoom for better quality

First test

ROF	DF	FPL	IPL	SP
20MHz	400kHz	205	127	125

Second test

ROF	DF	FPL	IPL	SP
25MHz	400kHz	192	127	125

Third test

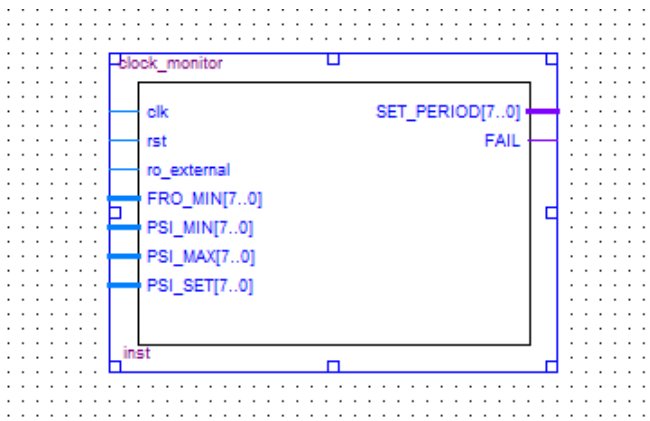
ROF	DF	FPL	IPL	SP
16.66MHz	400kHz	213	127	125

III. Clock Monitoring Unit

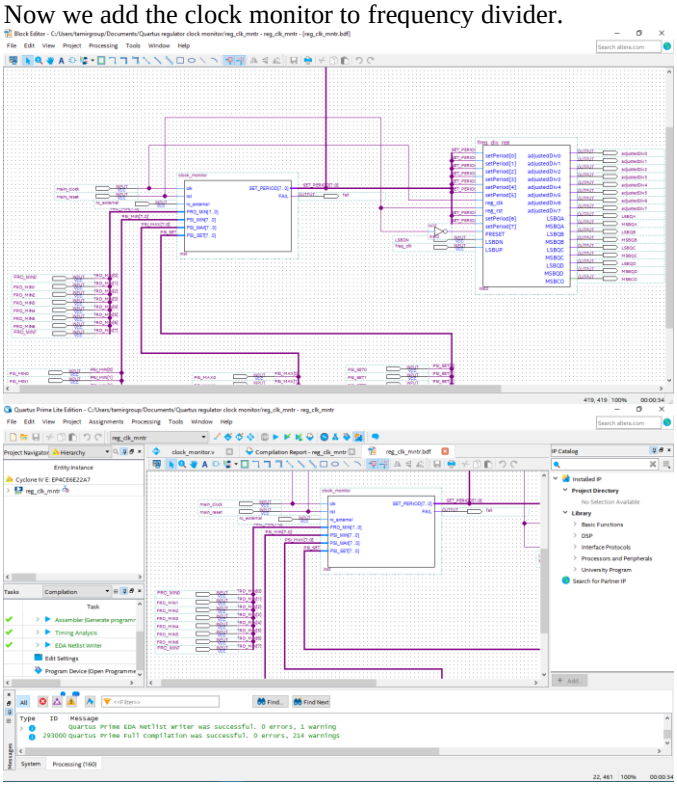
In this part, we design the clock monitoring unit, and then use it to generate different setPeriods for our regulator.

A. Design and Synthesis

Below is the symbol generated to represent a clock monitor.

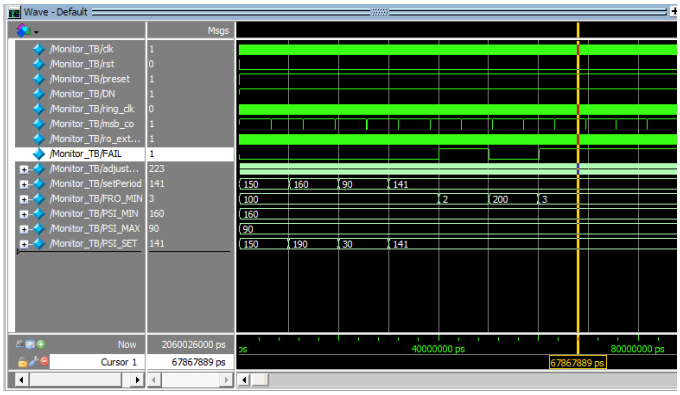


Now we add the clock monitor to frequency divider.



B. Simulation in Modelsim

After synthesising is complete, we write a test bench for our module and check the functions in modelsim:

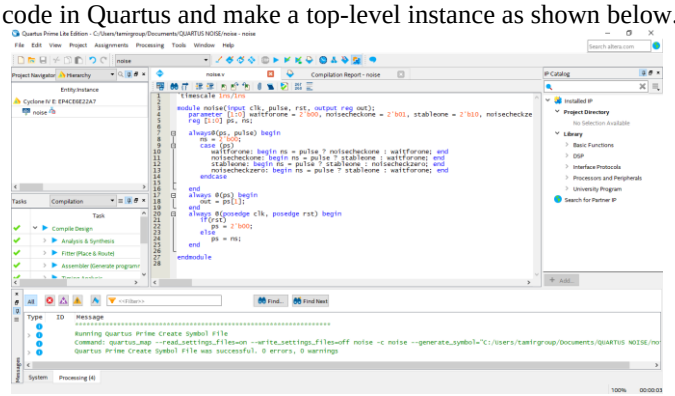


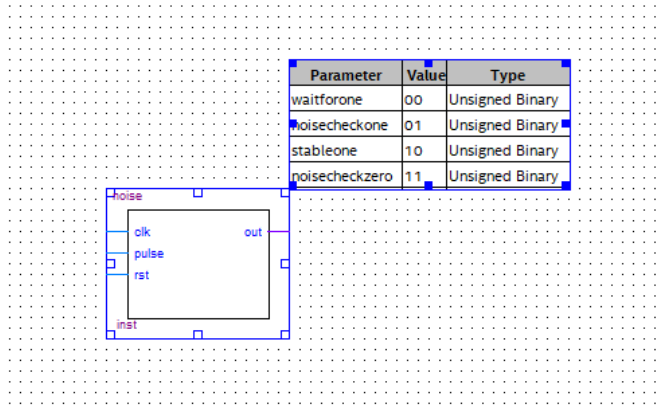
Explanation: We have set PSImin and PSImax to 160 and 90 repectively, the main clock has 50MHz frequency and the external ring oscillator frequency is near 24MHz. When PSI_SET is greater than PSI_MIN, or less than PSI_MAX, the circuit does the necessary changes. Also, when 6 * FRO_MIN is less than the ROfrequency (i.e. 24), the FAIL flag is issued.

IV. Noise Eliminator unit

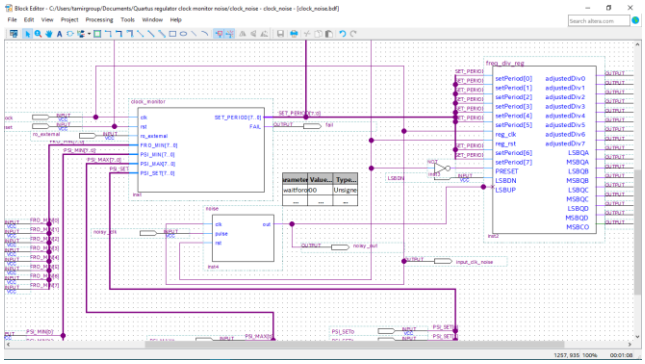
A. Design and Synthesis

First, we write the Verilog code. Then we synthesis that code in Quartus and make a top-level instance as shown below.

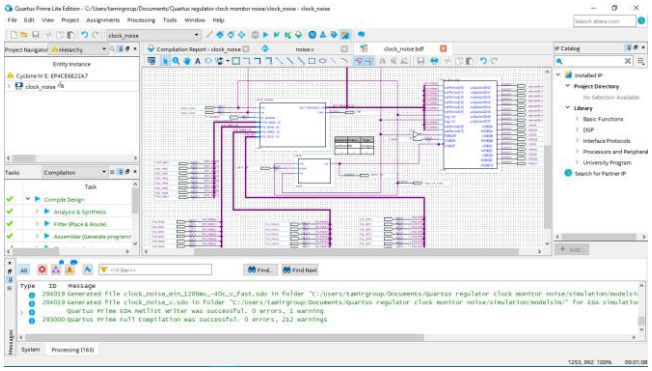




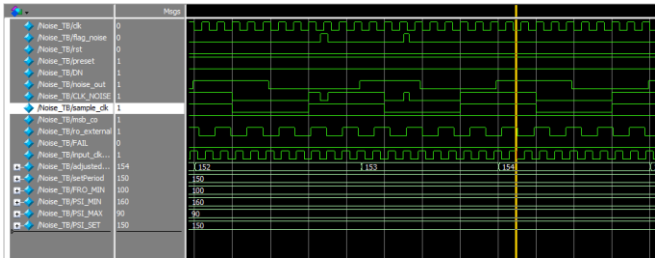
We add the noise eliminator to the circuit from the previous part.



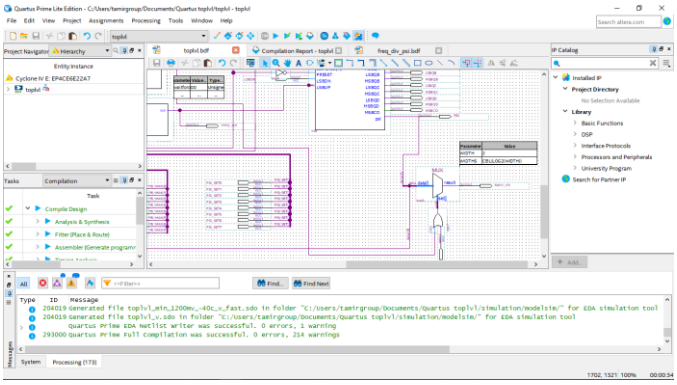
Now we synthesis the new circuit.



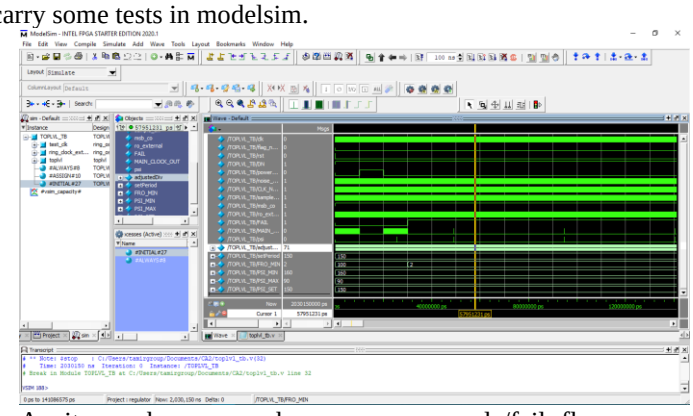
B. Simulation in Modelsim



After injecting two noises on the input clock of the noise eliminator (CLK_NOISE) we notice that after some delay, the output of said module has no noises whatsoever. The injected noises are from both to-one and to-zero types. Note that we've used a flag to generate the noises. Note that PSI_SET is set to 150 (less than PSI_MIN, more than PSI_MAX), and external clock frequency is equal to 24MHz which is less than 6 * FRO_MIN (100). Internal ring oscillator changes at 5MHz rate.



For the last part, after synthesising the required module, we carry some tests in modelsim.



As it can be seen, whenever powermode/fail flags are issued, the Main clock follows PSI, otherwise it copies the external RO. Note that the PSI clock duty cycle is NOT 50%.

V. Conclusions

In this Lab, we worked with different clocks at the same time. We designed circuits that needed three or more different clocks to operate! We handled different clock errors and defined clock boundaries. We learned about Noise Eliminators and their functions. And finally, we used *Powermode* for better processing in our circuit.