Lab 3: Clock Design Methodology

CS M152A Fall 2020, Majid Sarrafzadeh

Lab 6: Rajas Mhaskar

Kenta Hayakawa

11/08/2020

Introduction and Requirement

The purpose of this lab was to use the Xilinx ISE software to create, simulate, and test various clock waveforms on a digital system. This lab goes over basic system clocking principles, as well as techniques to generate them from a system clock. The lab was based on simulation only with no FPGAs involved at all, and was implemented in the Verilog HDL.

The 4 main design task modules to design were:

1) Various divide by power of 2 clocks
    a) Divide by 2 clock
    b) Divide by 4 clock
    c) Divide by 8 clock
    d) Divide by 16 clock
2) Even division clock using counters (divide by 28 clock).
3) Odd division clock using counters (divide by 5 clock).
4) Single pulse strobe.

In addition to the main 4 design tasks, 9 verification tasks and their waveforms were required. The final code clock_gen.v only includes code for the main 4 design tasks and their respective submodules; therefore, the 9 verification tasks are waveforms only (exceptions are some of the design tasks which are also verification tasks, such as clk_div_5).

Design Description

1) Design task 1 (submodule clock_div_two)

For design task 1, we were required to implement a divide by 2 clock, a divide by 4 clock, a divide by 8 clock, and a divide by 16 clock. All are powers of 2, so we are essentially making a divide by power of 2 clock. I implemented this by first making the divide by 2 clock, which was as simple as flipping the clk_div_2 clock every positive edge of the input clock clk_in. Then, I implemented the divide by 4 clock by simply flipping the clk_div_4 clock every positive edge of the clk_div_2 clock this time. This is because since we are making a power of 2 clock, the waveform for a certain power of 2 clock should be triggered on the positive edge of the previous power of 2 clock, since that will double the slowness of the output clock. Using this same logic, I created a divide by 8 clock by flipping the clk_div_8 clock every positive edge of the clk_div_4 clock (previous power of 2). Then, a divide by 16 clock was generated by flipping clk_div_16 clock on every positive edge of the preceding clk_div_8 clock.

2) Design task 2 (submodule clock_div_twenty_eight)

For design task 2, I created a divide by 28 clock using counters. A 4-bit counter counted from 0 to 13 in decimal, and on the 14th edge the output clk_div_28 flipped. Since there are 14 clock ticks underneath the waveform, this means that it is 28 times slower than the original input clock.

3) Design task 3 (submodule clock_div_5)

For design task 3, I created a divide by 5 clock using if statements and counters. I used two always blocks, one that triggers on the positive edge and another that triggers on the negative edge. Each always block has its separate counters and round counters. For every counter value that is supposed to flip on the positive edge of the input clock, I flipped the output. However, for every period of the clock this counter value changes. So I created a roundcounter variable to keep track of which period round we are on. Eventually this period pattern returns to its original form, so we set the roundcounter to 0 at that point (meaning we do not have to manually code the roundcounter and counter flips forever). This same method was used to implement the counter values flip the output on the negative edge of the input clock. These two always blocks together form the final divide by 5 clock with a 50% duty cycle.

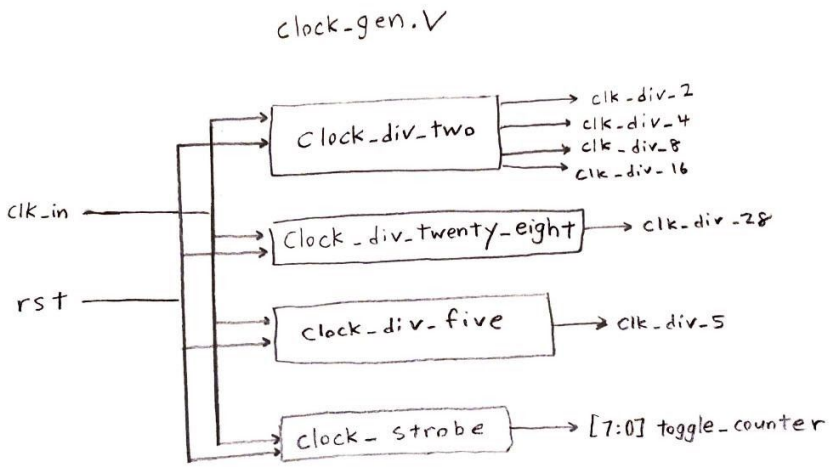
4) Design task 4 (submodule clock_strobe)

For design task 4, I created an 8-bit counter that increases by 2 on every

positive edge, but decreases by 5 on every strobe. First I created an always block that generates the strobe. Similar to previous submodules, a counter was used to keep track, and when counter is 3 the strobe output flips, and immediately after on counter=4 the output flips again. This essentially generated a 25% duty cycle strobe used in the final toggle counter.

The second always block generates the final toggle counter output. The toggle counter simply increases by 2, and on every strobe subtracts by 5. This was implemented simply by passing in the strobe output variable from the previous always block as the argument to the if statement in the second always block that subtracts the toggle counter by 5.

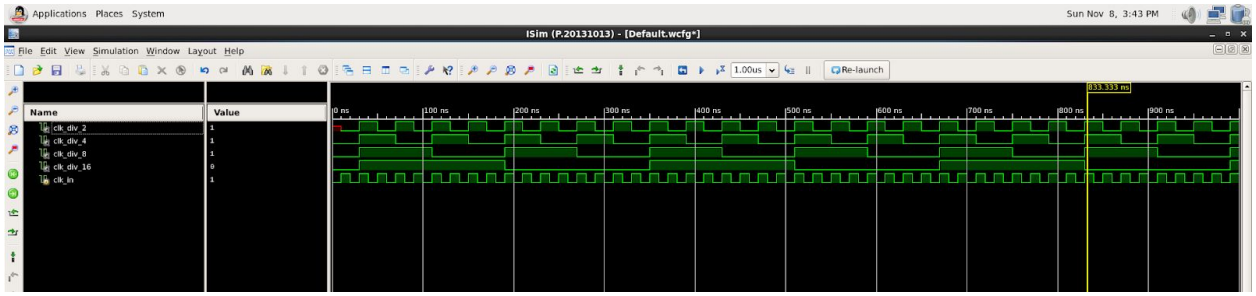
High level schematic of clock_gen.v:



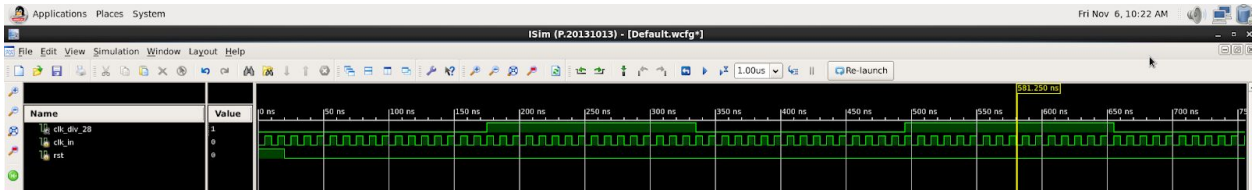
Simulation Documentation

For this lab, a testbench was created from scratch. There was not much to the testbench file, just code to generate the input clock signal and the reset signal, and a 10 unit delay before the input clock flips (since my time unit was 1ns, this will create a 100MHz clock signal required for tasks such as verification task 8). The following are all the 9 verification task simulation outputs generated by this testbench, as well as a single compilation of all the design tasks:

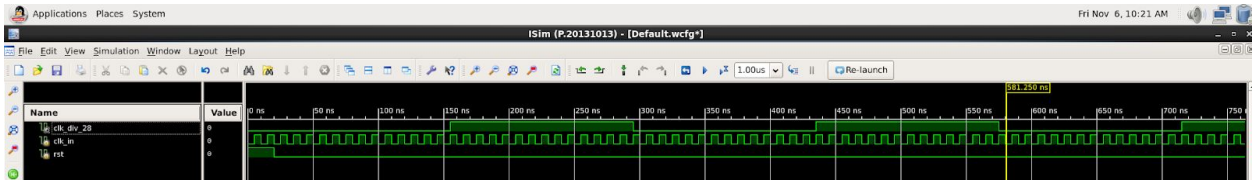
1) Verification task 1 (divide by power of 2 clock)



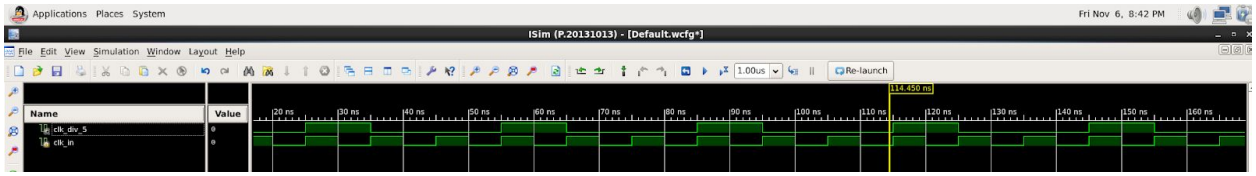
2) Verification task 2 (divide by 32 clock)



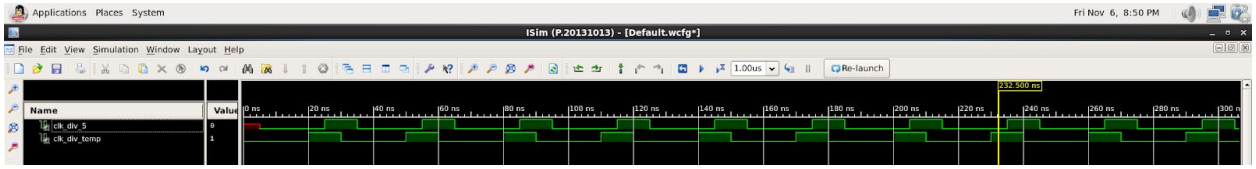
3) Verification task 3 (divide by 28 clock)



4) Verification task 4 (33% duty cycle clock)

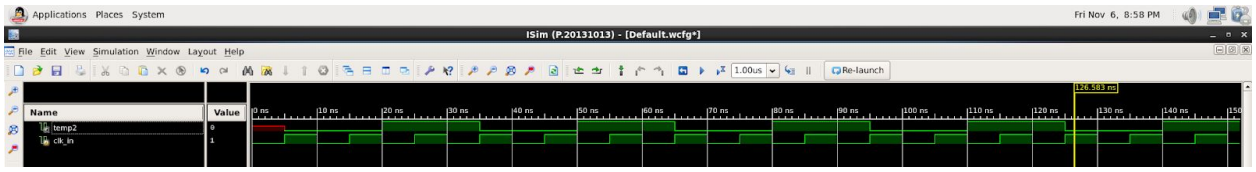


5) Verification task 5 (Two 33% duty cycle waveform comparison)

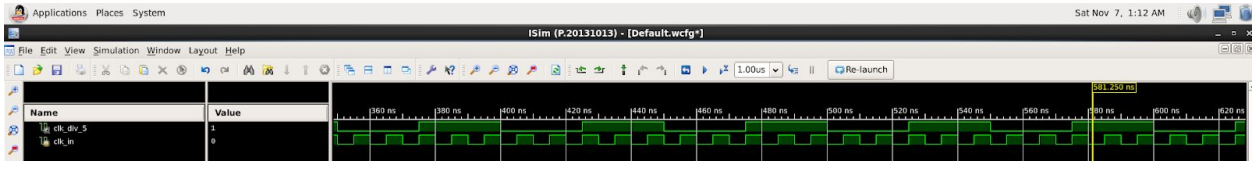


6) Verification task 6 (logical OR of both waveforms)

-Creates a divide by 3 clock with 50% duty cycle

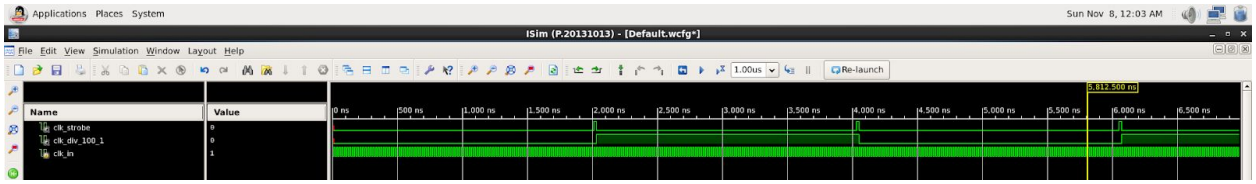


7) Verification 7 (50% duty cycle divide by 5 clock)

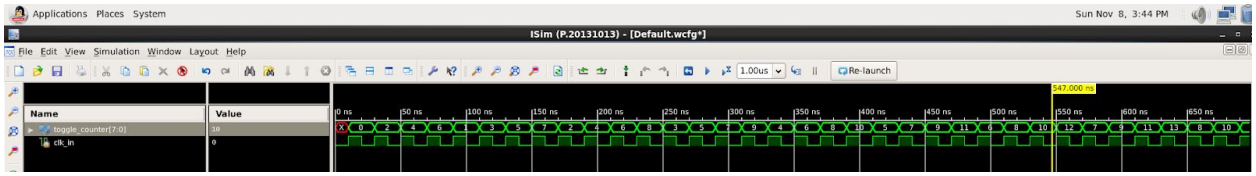


8) Verification task 8 (50% duty cycle divide by 200 clock at 500khz)

-clk_strobe is the strobe signal, clk_div_5 is final output



9) Verification task 9 (8-bit counter using divide by 4 strobe)



10) Final design task waveform outputs

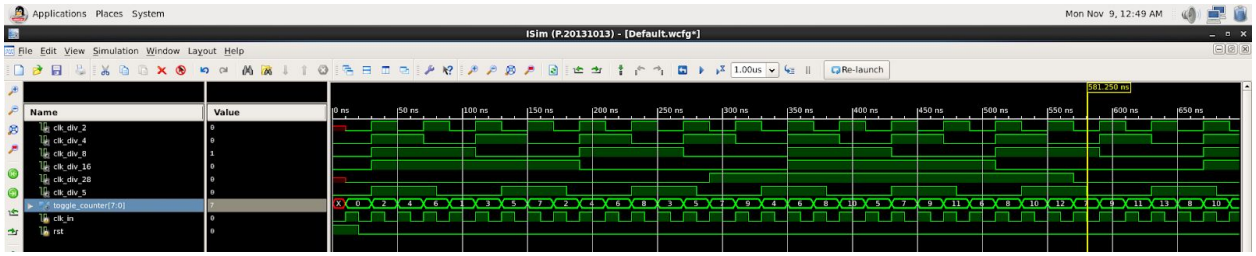
-First four outputs from top are design task 1

-Fifth output from top is design task 2

-Sixth output from top is design task 3

-Seventh output from top is design task 4

-Last two outputs at bottom are input clock and reset



Conclusion

The lab gave me first hands-on experience with Verilog coding and Xilinx simulations concerning clocking. We designed various waveforms from simulations, each corresponding to different techniques of clocking a system. There were a total of 9 verification tasks, as well as 4 main design tasks to accomplish.

The lab was a good exercise in learning how to clock a system. In industry, designers can run through waveform simulation or on physical devices by stepping one clock signal and check for expected behavior. Clocks and clocking techniques are an important tool to have, especially since synchronous data transmissions have become widely used in the real world, which allows signals to be communicated between different devices. One such example are timer systems, used in many digital devices we use on a daily basis without even thinking, such as traffic lights, monitor screens, digital stopwatches and phones.

I believe this lab was a simple and good enough introduction to digital clocking techniques, and hopefully I will get better at it with practice.