CSCE 22104

Lab Report

Brent Marcus Orlina

ID: 011019116

Lab Section 001

Lab 1

Introduction

This lab's goal was to create a component, "10-Counter," in VHDL that counts from 0 to 9 that increments during the rising edge of a clock cycle. The counter also loops back to 0 after reaching 9. The second goal of this lab was to create a clock component that displays the hours, minutes, and seconds and increments by one second during the rising edge of a clock cycle. The clock also loops back to displaying all zeroes after reaching 24 hours.

Approach

For the 10-Counter component, there are three 1-bit input ports, `clk`, `rst`, and `en` and a 4-bit output port, `out`. When `clk` is in a rising edge and `en` is active, `out` is incremented. However, when `rst` is active, it overrides `en` and synchronously resets `out` to 0.

To implement this a process sensitive to all three input ports is used. `clk` is checked if it is in a rising edge with `clk'event and clk = '1'`. If the condition is satisfied, `rst` is checked if it is active with `rst = '1'`. If it is active, a 4-bit signal `count` is set to 0, with `count <= "0000"`. Otherwise, `en` is checked if it is active. If it is, the signal `count` is set back to 0 if it reaches 10, otherwise it is incremented with `count <= count + 1`. The library `IEEE.std_logic_unsigned.all` is required to be able to increment `count` by using addition. Outside of the process, the signal `count` is connected to `out` with `out <= count`. The signal `count` is necessary since it is illegal to set an output port to a value that uses that same output port in VHDL.

For the clock component, there are three 1-bit input ports, `clk`, `rst`, and `en`, one 1-bit output port `cout`, one 4-bit output port `H` and two 5-bit output ports `M` and `S`. Similar to the 10-Counter, when `clk` is in a rising edge and `en` is active, the output port `S` is incremented. When `S` reaches 60, it is set back to 0 and the output port `M` is incremented. Similarly, when `M` reaches 60, it is set back to 0 and the output port `H` is incremented. Finally, when `H` reaches 24, it is set back to 0. `cout` is active when `H` is incremented to 23.

To implement this, two more components are made, "60-Counter" and "24-Counter." These counters use "10-Counter" as the base. The two other counters instead count up to 60 and 24 respectively. Another difference is that these counters have an extra 1-bit output port `cout`. These are used to chain the counters together. In the clock, the 60-Counter responsible for `S` activates its `cout` when it would be incremented to 59. The `cout` is then connected to the 60-Counter

responsible for M's en input port. This is so that in the next cycle, when S is incremented to 60 and set back to 0, M is incremented, since there are 60 seconds in a minute. This is also the same for the M's 60-Counter and H's 24-Counter.

In the clock component two instances of 60-Counter and one instance of 24-Counter is created. The clock's clk and rst input ports are connected to each of the counter's clk and rst input ports. The clock's en is connected to the first 60-Counter's en. The first 60-Counter's out is connected to S and its cout is connected to the second 60-Counter's en using a signal sec_carry. The second 60-counter's cout is connected to a signal min_carry and its out is connected to M. The signals sec_carry and min_carry are connected to the 24-Counter's en through an AND gate. This is because an hour should only increment once both the second and minute gets incremented to 60. If only the min_carry signal was connected, it would increment the 24-Counter for 60 cycles when the second 60-Counter is incremented to 59 producing the wrong behavior.

Experimentation

The components' behaviors were tested using hand-crafted testbenches. The testbench for the 10-Counter tested all possible combinations of the ports en and rst signals as well as testing the behavior of the counter incrementing from 9. The counter worked as expected and incremented during the rising-edge of every clock cycle when en was active. The counter looped back to 0 once it incremented from 9. The reset behavior worked as expected with the counter resetting back to 0 when rst was active and taking priority over en.

The testbench for the clock component tested all possible combinations of the en and rst signals. It also tested the behavior of ports S and M incrementing during the correct times as well as looping back to 0 when they've reached their defined maximum limit. The behavior of port H was only tested for incrementing during the correct times. The component worked as expected and S incremented during the rising edge of every clock cycle when en was active. The port M correctly incremented only when S was incremented from 59 and looped back to 0. H correctly incremented only when S and M were both incremented from 59 and looped back to 0. Similarly, the reset behavior worked as expected with S, M, and H resetting back to 0 when rst was active and taking priority over en.

Results & Discussion

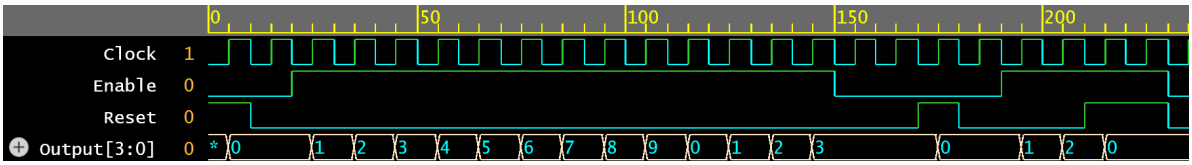

Figure 1: The waveform for the 10-Counter component implementation.

The 10-Counter works as expected as shown in figure 1. The counter has to first be reset, by activating signal `Reset`, so that the output is initialized to 0. The signal `Enable` is then enabled. It shows that the signal `Output` increments during the rising edge of every clock cycle when `Enable` is active. It also shows that the counter resets back to 0 it would increment to 10 in the 115th nanosecond. In the 150th nanosecond, `Enable` deactivates and the counter correctly stops incrementing. In the 175th nanosecond, it shows that when `Reset` is active, it synchronously resets the output back to 0. It can also be seen in the 215th nanosecond that when both `rst` and `en` are both active, `rst` takes priority and sets `output` to 0.

The testbench for this component is sufficient since it tests all the possible combinations of `en` and `rst`. It also checks the behavior of the clock when it is incremented from 9, with the expected behavior that it loops back to 0.

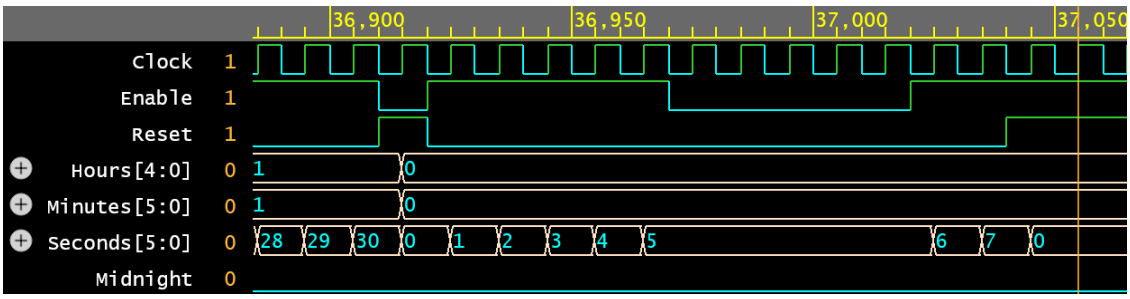

Figure 2: Segment of the waveform for the clock component implementation. Shows how seconds part increments. Also highlights that the clock is reset when `rst` is active.

The clock component also works as expected. Figure 2 shows the reset and enable functionality of the clock component. In the 36,915th nanosecond, the signal on the testbench `Reset` activates and the clock correctly and synchronously resets all of its output ports back to 0. In the 36,970th nanosecond, the signal on the testbench `Enable` deactivates and the clock correctly stops incrementing. In the 37,045th nanosecond, `Reset` activates while `Enable` is also active, and the clock

resets back to 0, correctly taking priority over `Enable`.

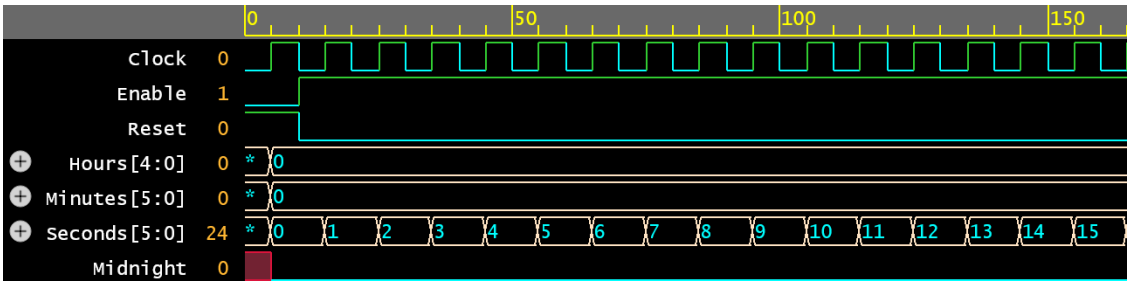

Figure 3: Segment of the waveform for the clock component implementation. Shows how seconds part increments. Also highlights that the clock is reset when `rst` is active.

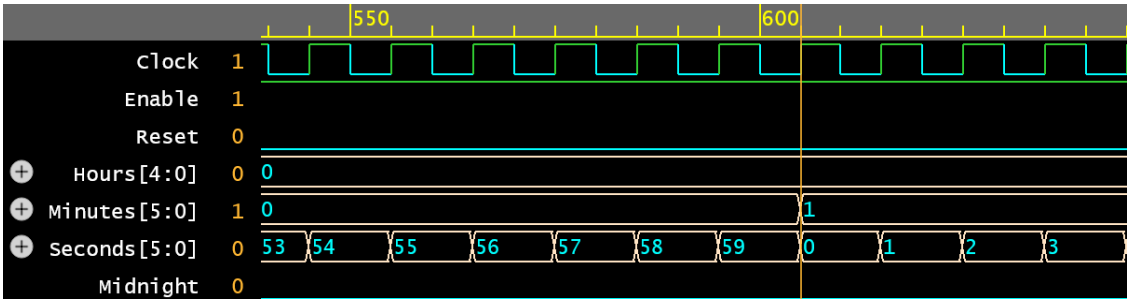

Figure 4: Segment of the waveform for the clock component implementation. Shows how seconds part increments. Also highlights that the clock is reset when `rst` is active.

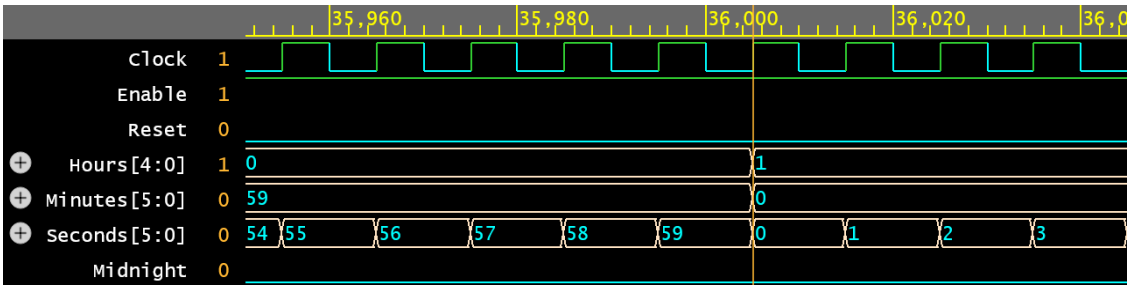

Figure 5: Segment of the waveform for the clock component implementation. Shows how seconds part increments. Also highlights that the clock is reset when `rst` is active.

In figure 3, the seconds increments during the rising edge of every cycle when `Enable` is on, which is the correct behavior. It also shows that the clock is set back to 0 when `rst` is active. Figure 4 shows that the `Minutes` increments from 0 to 1 when the seconds is incremented from 59 back to 0. This is the correct behavior since there are 60 seconds in 1 minute. Similarly, figure 5 shows that the `Hours` increments from 0 to 1 when both the `Minutes` and `Seconds` increment from 59 back to 0 since there are 60 minutes in 1 hour.

The testbench for the clock component is not completely sufficient. It tests the `Reset` and `Enable` functionality. It tests for the behaviors for when `Seconds`, `Minutes`, and `Hours` should increment. It also tests for the behavior of when `Seconds` and `Minutes` increments from 59. However, `Hours` incrementing from 23 and the output for `Midnight`, which should be active when `Hours` is incremented to 23, was not verified. This was because the software used was not able to simulate the clock cycling all the way to 24+ hours worth of clock cycles.

Conclusions

Both the 10-Counter component and clock component displayed the correct behavior during testing. The knowledge learned from this lab was learning counters worked and how they can be constructed as well as chaining different counters together. The skills practiced in this lab was learning how to build components behaviorally and structurally, and creating sufficient testbenches that test the components behaviors with different inputs. Debugging VHDL code was also an important skill practiced.