The Stopwatch

Magnus Björk
mab@cs.chalmers.se

Revised by Emil Axelsson

April 3, 2008

1 Introduction

The purpose of this lab is to give an introduction to VHDL programming and (to a larger extent) functional verification.
The homepage has a page with guidelines for design and verification, which should be followed in this lab. It is found in the "Labs and Exams" section of the Assignments page, just above the link to the lab. Note that you are provided with a partial solution called StopwatchRemoved.vhdl. It was created from a good correct solution by deleting parts (or rather replacing them by ...), so we strongly advise you to use it!

2 The Circuit

You are supposed to create the control circuit for a stopwatch. The circuit has three inputs: a 100 kHz clock, a start/stop switch and a lap/reset switch input. The circuit has outputs to drive the display consisting of six digits (two for minutes, two for seconds, and two for hundredths of seconds).

The stopwatch counter is initially reset to 00'00'00, with the display showing the counter time. In this state, pressing the start/stop button starts counting, with the display showing the counter time. Pressing the start/stop button again stops the counting. Successive presses of start/stop continue or stop counting, with the display showing the counter time.

If the lap/reset button is pressed while the counter is stopped and the display is showing the counter time, the counter is reset to 00'00'00. If the lap/reset button is pressed while the counter is running, the display freezes at the time at which the lap/reset button was pressed, but the counter continues to run. If the start/stop button is pressed, the counter stops, and the display remains unchanged. Successive presses of start/stop continue or stop counting, with the display unchanged. Pressing the lap/reset button while the display is frozen causes it to return to displaying the current counter time, whether the counter is running or stopped.

The switches are active high, which means that the input signal is high while a switch is pressed. You should use signals of type `std_logic` (for switches, the clock, etc) and `unsigned`, (for the digits).

3 Implementation

Read the "guidelines" page for more information about what is required here. Make two different implementations of the stopwatch (two architectures for the same entity):

3.1 The Behavioral Model

Start by making a behavioral model. You should keep in mind that it should be easy to understand. The behavioral model is your specification of how the circuit should work. Ideally, it should be easy to see the functional behavior of the circuit by looking at the code.

3.2 Implementation at the Register Transfer Level

Make an RTL implementation of the circuit. The partial solution provided uses the design shown in figure 1, and we strongly recommend that you do too. If you use a finite state machine (as shown in the figure), you must describe its states and transitions. A short description of the modules in figure 1 follows. Since all of these modules are sequential, they all have a clock input, which is not mentioned in the descriptions. The parameters of Count and Hold should be implemented using generic parameters.

- **Pos_Edge** outputs `'1'` if its input has gone from low to high during the last clock cycle. It may be tempting to use signal attributes like `'delayed` or `'stable` to achieve this, but remember that such attributes are generally not synthesizable, so this is not allowed. A synthesizable solution needs to use some kind of state machine.
- **Finite State Machine** decodes the button presses and determines which state the stopwatch is in.
- **Count(n, b)** has two inputs: `increase` and `reset` (both `std_logic`), and two outputs: `counter`, which is an `unsigned` with `b` bits), and `carry` (which is a `std_logic`). Initially `counter` is 0 and `carry` is low. If `increase` is low, `counter` has the same value as it had at last positive clock edge. When `increase` is high, the value of `counter` is one higher than it was at last positive clock edge, except if it was n-1 then. In that case, the new value should be 0 and carry should be high. When `reset` goes high, the counter should be reset to 0.
- **Hold(n)** has an n-bit `unsigned` output and an identical input, and another `std_logic` input named `enable`. Whenever `enable` is low, the n-bit

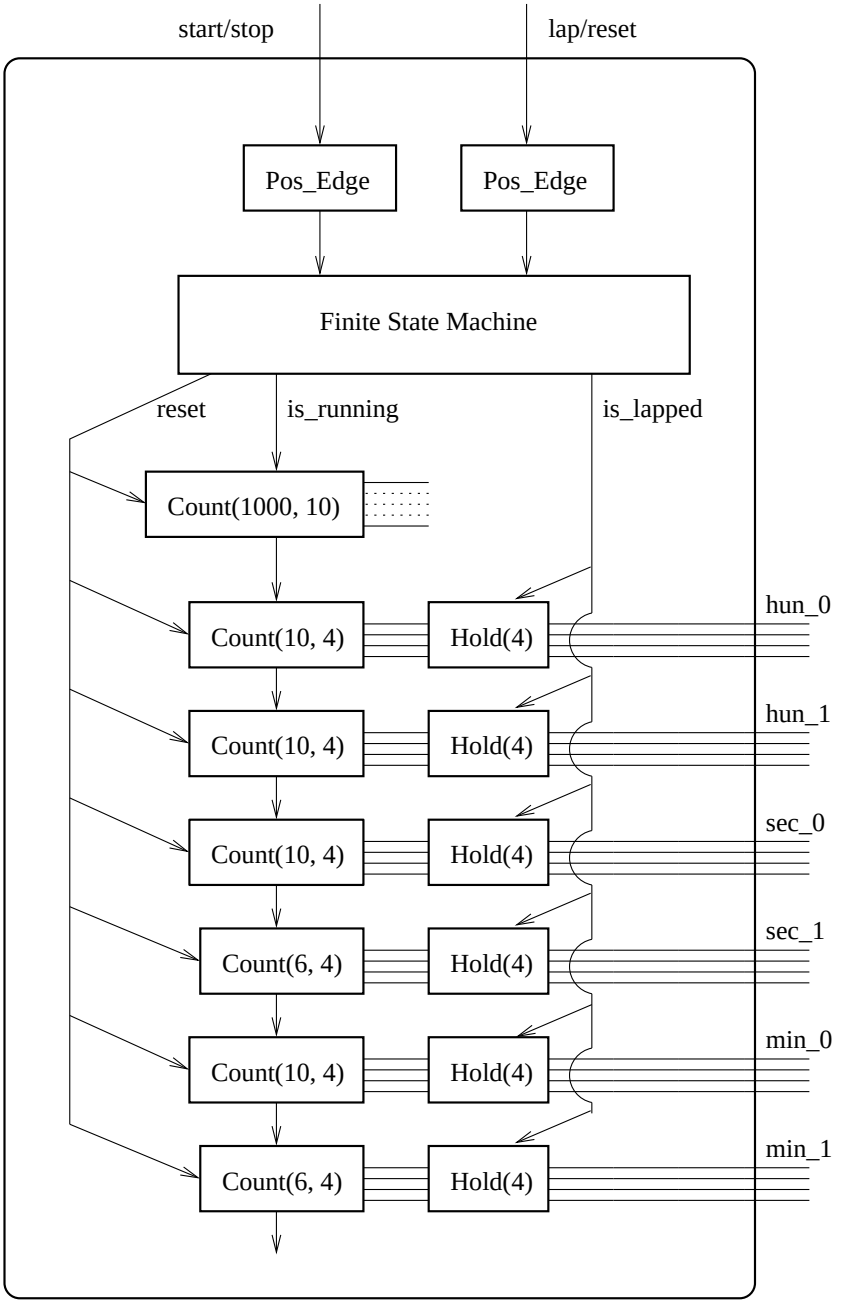

Figure 1: Suggested design of the circuit.

output is a copy of the input, and when `enable` is high, the output remains unchanged until `enable` goes low (no matter what happens on the input). Alternatively, the module can output whatever value was on the output signal at the last positive clock edge, when `enable` is high.

Note 1: In previous years, students have discovered that there is a counter element similar to the one describe above available in Sjöholm & Lindh. But that counter has a major difference: there is a flipflop between the carry input and the carry output (why?). The problem with this is when several digits change value at the same time (for example, in the transition 59'59'99 → 00'00'00). A flipflop on the carry will then cause one clock cycle delay between each digit's transition. This is probably not a desired behavior, and as you can imagine, it is not very easy to write a specification to match this behavior.

Using the two-process method described by Gaisler (see his notes on the Literature page) is a very good idea here. This method proposes two processes in each architecture: one which is sensitive *only* to the clock (and perhaps reset), and one which is sensitive to input signals and internal state, but not the clock. In the combinational process, it is possible to immediately affect output signals without going through the clocked process. This lets you get rid of the flipflop on the carry output.

Note 2: In previous years, students have implemented "ripple counters" – that is, counters which are sensitive to edges on the carry signal. First of all, this does not follow the specification above. Second, it goes against the guidelines on the homepage, which state that there should be only one global clock signal. We have also seen implementations were the clock is shut down by the carry signal (clock gating). This should be avoided for the same reason.

4 Verification

Read the "guidelines" page for more information about what is required here.

4.1 Verifying the Behavioral Model

Start by verifying that your behavioral model works as it should. Here, you will probably write your own test benches and study the input and output signals in the simulator. Perhaps you will also be able to specify some properties and use Jasper Gold (but this is not necessary, especially since the behavioral model doesn't have to be synthesizable).

4.2 Verifying the RTL Implementation

If you want, you can do the same verification on the RTL implementation as you did on the behavioral model (but not only). However, the emphasis here is to show that the RTL implementation is equivalent to the behavioral model.

Here, you will probably only study the output signals if you actually find bugs. It's excellent if you can use Jasper Gold in this step, but not required.

4.3 Formal Verification of the Counter

Verify that the counter module works as it should, using Jasper Gold. Try to use the description of **Count(n, b)** as a source for your properties. Make sure that your properties are complete, in the sense that they capture all aspects of the counter elements.

4.4 Formal Verification of the Finite State Machine (FSM)

Verify that the Finite State Machine in your implementation works as it should, using Jasper Gold.

4.5 Report

Write a short report on all of the verification that you have done (approximately $\frac{1}{2} - 1$ page). Argue for why you think that your verification is enough, and describe what properties you have verified. If there are aspects that you haven't tested, write that down and explain why. Think of it as if you were constructing this circuit for a company that you work for, and write this report to explain to your boss or co-workers why they should trust your circuit.

The important thing here is not to write a fancy report, but to describe what you did in the verification.

5 Formalia

This is a normal laboration: You are allowed to work in pairs. You may not copy solutions (or parts of solutions) between different groups, but you may discuss your ideas (see more about cheating on the course homepage). Make sure that both people in a pair work; in the take home exam, you'll be doing something similar on your own. During the lab you are encouraged to ask questions, but during the week of the take home exam we will not answer questions about the lab (and only urgent questions about the take home exam, with the possibility of deduction of marks for help given by us).

Details of how and when to submit your solutions will be available on the course page.

6 Summary

Submit

- All relevant VHDL code (all parts of the stopwatch and your test benches)
- PSL files, and other relevant parts of the verification

- Verification report
- If you have any other files than VHDL and PSL files, you must describe them.

7 Acknowledgements

Parts of this text are copied from an exercise in

Peter J Ashenden. *Designer's Guide to Vhdl.* Morgan Kaufmann, 2001.