Computer Architecture 4

Assessed Exercise 1
Circuit Design

Handout: Monday, January 27, 2014
Handin: Friday, February 14, 2014
See Moodle page for instructions on submitting the solution.

Introduction

The purposes of this exercise are to learn how to design and test a simple synchronous digital circuit, and to introduce you to using the Hydra hardware description language.

Preparation

This exercise can be carried out either using the Linux machines in the level 4 lab, or your own machine. You will need two pieces of software: the Haskell compiler ghc, and the Hydra library. All of this is free software, and it works on Linux, Windows, and Macintosh. The Moodle page explains how to install and use the software. After installation, try running the adder and register examples on the Moodle page.

The exercise

Design and implement

- Two circuits, as described below, which should all be in a file TrafficLights.hs.
- A simulation driver (also called a test bench) for each circuit; both of these should be in a file named TestTrafficLights.hs
- Suitable test data that demonstrates the correct functioning of each circuit, which should be included in TestTrafficLights.hs.
- There should be a main function in TestTrafficLights.hs that runs all of your test cases.

Informal specification of the circuit

The circuits are traffic light controllers. There are two versions.

Version 1

The circuit controller1 has one input, a bit called reset. This would be connected to a pushbutton. The reset button should be pushed once to start the circuit, and then it should never be pressed again. We model this by defining the value of the reset input bit to be 1 during the first clock cycle, and 0 thereafter.

There are three outputs, each of which is a bit. The outputs correspond to green, amber, and red, and they determine whether the corresponding traffic light is on. At all times (after reset has been pressed and the circuit is running) one of the three output bits should be 1, indicating that the corresponding traffic light should be on, and the other two bits should be 0. The outputs should run through a fixed sequence: green, green, green, amber, red, red, red, red, amber, and then it repeats.

Version 2

This version, controller2, is intended for a pedestrian crossing. There are three lights for traffic (green, amber, and red) and also two lights for the pedestrian (wait and walk). There are two input bits: a reset pushbutton, and a walkRequest which is 1 when a pedestrian presses the Walk pushbutton.

Normally the outputs indicate green/wait. However, when the walkRequest button is pressed, the traffic light changes to amber, and then the traffic light changes to red and the pedestrian light turns to walk for three clock cycles. Then the system displays amber/wait and then returns to its normal state.

Furthermore, the traffic engineers want to know how often the walkRequest button is pressed. To measure this, there is a 16-bit counter walkCount. When the Reset button is pressed, the value of WalkCount is set to 0 at the next clock tick. When the walkRequest button is pressed, walkCount should be incremented at the next clock tick.

To summarise, the inputs are: reset (a pushbutton) and walkRequest (a pushbutton). The outputs are: green, amber, red, wait, walk (each is 1 bit), and walkCount (a 16-bit binary integer).

In the real world, the reset button and the walkCount display would be hidden inside the box containing the electronics, while the walkRequest button would be out where a pedestrian could see and press it, and the various light outputs would control the actual light bulbs.

Assessment

The exercise will be marked out of 100 marks, and it counts for 10% of the CA4 assessment. The two exercises together are worth 20% of the assessment, while the examination is the other 80%.

On this exercise, Version 1 and Version 2 have equal weight (each is worth 50 marks). Assessment will be based on (1) handing something in; (2) a reasonable approach to the problem; (3) a correct solution; (4) a working simulation driver; (5) suitable test data.

Handin

The handin will be via the Moodle page; see the page for detailed instructions.

Your handin should consist of a single file with a name of the form Smith-John.tgz (but change it to your name, not John Smith's). (If you cannot produce a tgz file on your system, you can use zip. Other formats, such as rar, are not acceptable.) This file defines a directory named Smith-John containing the following files:

- StatusReport.txt — a text file containing your status report
- TrafficLight.hs — the circuit definition file, containing the two circuit specifications
- Testbench.hs — definition of the testbench as well as suitable test data. This should define main :: IO () so that running main will execute your tests for both circuits.

The status report, StatusReport.txt, should be a plain text file. The opening lines should follow a key-value format, where each keyword is followed by a colon : and a space, and then the value is just text. You should define the keys that are illustrated in the following example of a status report:

```
course: CA4
exercise: 1
date: 2014-02-14
surname: Smith
forename: John
email: johnsmith@your.email.address

Now, after a blank line, is a status report.  This should say
whether each part of the exercise has been completed, and its
status: Does it compile?  Does it appear to work correctly?  How
did you test it?  Are there any aspects that appear to be not quite
right?  Are there any especially good aspects you would like to
highlight?

If your status report contains several paragraphs, separate them
with a blank line.  You can explain your approach to solving the
problem in the status report, or you can include your documentation
as comments in the source program.
```