University of Glasgow | School of Computing Science

Assessed Coursework

Course Name	**Computer Architecture 4**			
Coursework Number	**1 -- State machine circuit**			
Deadline	**Time:**	**5pm**	**Date:**	**Thurs, 12 Nov 2015**
% Contribution to final course mark	**10%**			
Solo or Group ✓	**Solo**	✓	**Group**	
Anticipated Hours	**5 hours**			
Submission Instructions	**Submission via Moodle**			
Please Note: This Coursework cannot be Re-Assessed				

Code of Assessment Rules for Coursework Submission

Deadlines for the submission of coursework which is to be formally assessed will be published in course documentation, and work which is submitted later than the deadline will be subject to penalty as set out below.

The primary grade and secondary band awarded for coursework which is submitted after the published deadline will be calculated as follows:

(i) in respect of work submitted not more than five working days after the deadline
   a. the work will be assessed in the usual way;
   b. the primary grade and secondary band so determined will then be reduced by two secondary bands for each working day (or part of a working day) the work was submitted late.

(ii) work submitted more than five working days after the deadline will be awarded Grade H.

Penalties for late submission of coursework will not be imposed if good cause is established for the late submission. You should submit documents supporting good cause via MyCampus.

Penalty for non-adherence to Submission Instructions is 2 bands

You must complete an "Own Work" form via https://studentltc.dcs.gla.ac.uk/ for all coursework

Computer Architecture 4

Assessed Exercise 1

State Machine Circuit

Introduction

The purposes of this exercise are to learn how to design and test a simple synchronous digital circuit that implements a state machine. The circuit specification and simulation will be carried out using the Hydra hardware description language.

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 on submission.

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. (Important: when your file is extracted with tar or unzip, it should produce a *directory* — it is unacceptable just to dump a lot of files into the user's directory.) The directory should contain 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: 2015-11-02
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.
```