SOFTENG 251 Software Engineering 1
Assignment Two: Digital Simulator

The University of Auckland

September 17, 2002

WORTH: 6.5% towards final mark for course
Due: 9pm, Thursday 19th September 2002 (see Section 7 for bonus details)

1 Introduction

This assignment aims to give you some further experience with object-oriented programming using a test-first approach. It involves adding extra capability to the Digital Simulator that was covered in lectures.

2 What you are to do

The assignment is broken into a sequence of stages of varying complexity. Please complete the assignment in this order, using a Test-Driven Development approach. The marks for each stage are included with the stage; those marks total 60. See section 6 for general marking details.

You are to provide a complete record of your test-driven development of the assignment, by retaining a copy of your system at the following times:

- After you've written a test but before it is passed
- After you've passed the test and carried out any necessary refactoring

Save a copy of the folder containing all the files. Name the copies folders according to the Stage (2 digits) and the step within the Stage (1+ digits). Eg, the folder for the first step for Stage One will be named "01-1". The final step you submit is to be in a folder called "assignment2". The easiest way to manage this is to do all your development in the folder "assignment2", and to make a copy of that folder at each time, as specified above.

Keep a record of how long you spend on each stage (to the nearest 5 minutes) in a spreadsheet, as outlined in section 5. A statistical summary of the results from all the spreadsheets, and other results, will be supplied to the class once the assignment has been marked, so you can see the range and how you compare to others in the class.

Ensure that you follow the directions carefully, because the correctness of your program will be marked automatically. An acceptance test program will be supplied on the server, which you can use as a large-scale test of your code. Do not change those acceptance tests. The final marking of your code may be more comprehensive than those acceptance tests. These tests differ in nature from unit tests, so they won't be helpful in driving your design.

3 Help

If you are not sure what's required at any stage below, check the forum, and/or to email Rick to ask, by providing a test case and asking whether it will succeed. Received test cases will most

Figure 1: Delay Gate

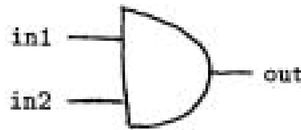

Figure 2: And Gate

probably not be published on the forum – please do not post them there yourself.

If you believe that there is an error in this document or in the acceptance tests, please email Rick about it as soon as possible. It is quite possible that there are errors - the only means of seriously testing the tests is to test them against an application. Of course, you may find errors which are really in your code or which arise because you have misunderstood what was required.

3.1 Stage One: Extract Package [5 marks]

Extract the classes *Agenda*, *TimedTask* and *TestAgenda*, along with the Java interface *Task* into a package *simulate*. Make the class *TimedTask* package-private, as it shouldn't be visible outside the *simulate* package.

All other classes remain outside a package and use the package *simulate* where necessary. Check that all the tests still pass.

3.2 Stage Two: Add a Delay gate [5 marks]

Add a class *DelayGate*, which passes it's input signal to its output wire after a specified delay. The constructor has arguments as follows: *DelayGate(int delay, Wire in, Wire out, simulate.Agenda agenda)*.

A *DelayGate* is shown in Fig. 1.

3.3 Stage Three: AndGate [5 marks]

Add a class *AndGate*. Its constructor has the same arguments as an *OrGate*. It has a delay of 6, and computes the logical *and* of the inputs to produce the output signal.

An *AndGate* is shown in Fig. 2.

3.4 Stage Four: Signal Generator [10 marks]

Add a class *Generator*, which generates a square wave, with a specified cycle time, but only when the *control* wire is 1. If the *control* wire changes to 0, the output goes to 0. The *Generator* responds to any change of the *control* after a delay of 3.

The constructor has arguments as follows: *Generator(int cycleTime, Wire control, Wire out, Agenda agenda)*. A *Generator* is shown in Fig. 3.

A sample timing diagram for a generator with a cycle time of 4 is shown in Fig. 4.

Figure 3: Generator

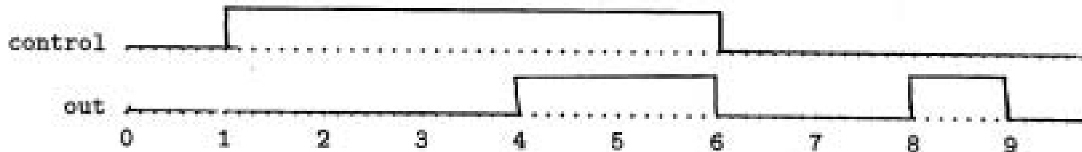

Figure 4: Timing Diagram for Generator

3.5 Stage Five: Nand Composite Gate [5 marks]

Introduce a class *CompositeGate*, with a ***static void*** method *nandGate()* that takes the same arguments as the constructor of an *OrGate*. It builds a Nand gate by wiring together an *AndGate* and an *Inverter* (so it effectively has a delay of 10).

Do not introduce a new class for the Nand gate itself. The class *CompositeGate* will grow to include several functions for composing composite gates from elementary ones.

A Nand gate is shown in Fig. 5

3.6 Stage Six: Mux Composite Gate [5 marks]

Add ***static void*** method *mux()* (a multiplexer) to the class *CompositeGate*. The method arguments are as follows: *mux(Wire in1, Wire in0, Wire control, Wire out, Agenda agenda)*. When the *control* signal becomes 1, the *in1* signal appears on *out* after a delay. When the *control* signal becomes 0, the *in0* signal appears on *out* after a delay. The delays are determined by the underlying circuit.

A Mux gate is shown in Fig. 6

3.7 Stage Seven: Simultaneous Changes to a Wire [10 marks]

Consider the following test:

```
public void testTwoChangesAtSameTime() {
    new AndGate(in1,in2,out,agenda);
    scheduleSignal(in1,1,true);
```

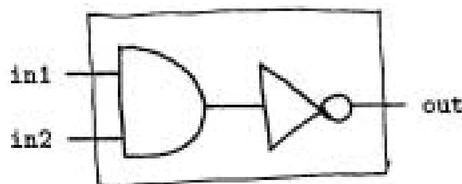

Figure 5: Nand Composite Gate

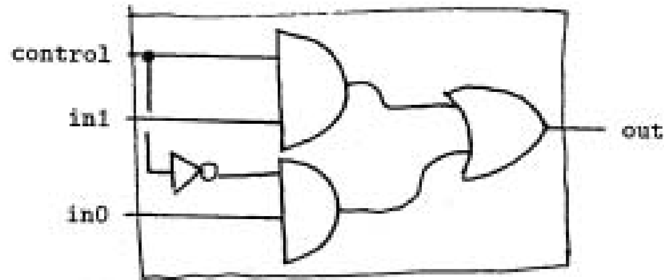

Figure 6: Mux Composite Gate

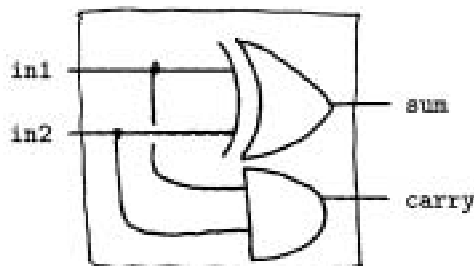

Figure 7: Half Adder

```
        scheduleSignal(in2,2,true);
        scheduleSignal(in1,2,false);
        agenda.run();
        assertEquals("(0,false)",probe.getLog());
    }
```

This fails with the supplied code, giving a log of: "(0,false)(8,true)(8,false)". That's because the wire signal changes are processed one at a time.

Fix this problem without changing the *Probe*, and without changing the code of the different gates.

3.8 Stage Eight: Half Adder and Full Adder [10 marks]

Add *static void* methods *halfAdder()* and *fullAdder()* to the class *CompositeGate*. The method arguments are as follows:

- *halfAdder(Wire in1, Wire in2, Wire sum, Wire carry, Agenda agenda)*
- *fullAdder(Wire in1, Wire in2, Wire carryIn, Wire sum, Wire carryOut, Agenda agenda)*

The Half Adder circuit is as shown in Fig. 7. This makes use of an XOR gate, which is defined by the circuit shown in Fig. 8.

The Full Adder circuit is shown in Fig. 9.

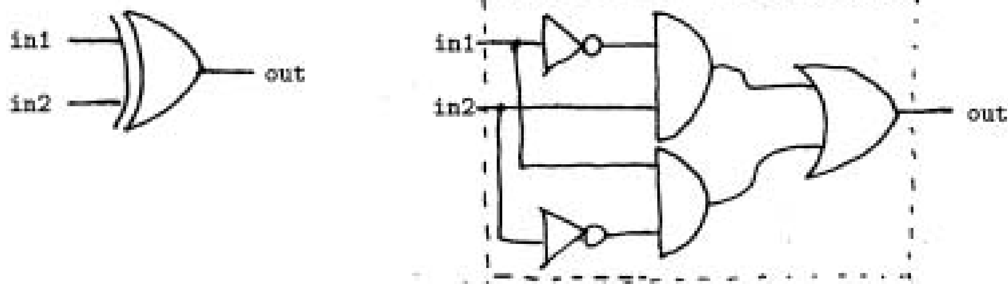

Figure 8: XOR Composite Gate

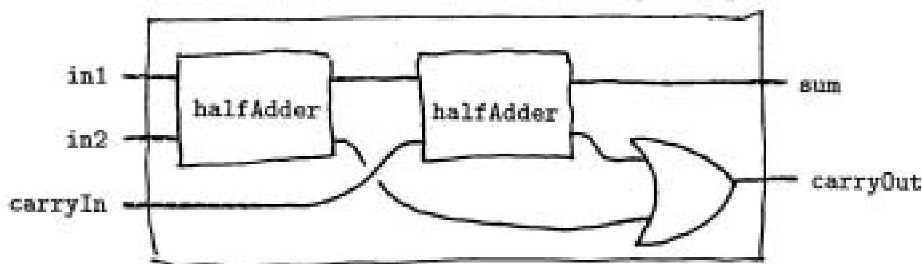

Figure 9: Full Adder

3.9 Stage Nine: N-bit Adder [5 marks]

Add *static void* method *adderChain()* to the class *CompositeGate*, where the number of bits of adder are determined by the length of the input and sum arrays. The method arguments are as follows: *adderChain(Wire[] in1s, Wire[] in2s, Wire carryIn, Wire[] sums, Wire carryOut, Agenda agenda)*. There is a constraint on the arguments that *in1s.length == in2s.length == sum.length*.

A 3-bit adder is composed as shown in Fig. 10.

4 Resources

Files for the *Simulator* application and the acceptance tests are available on the 251 web page under *Resources*.

5 What to Submit

Please submit:

- All the Java source in folders for each of the steps of your assignment (see Section 2 for details). If a stage is incomplete, ensure that your final program still works for earlier stages.
- A spreadsheet, saved in comma-separated format (CSV), with the following values:
  - A1: your upi

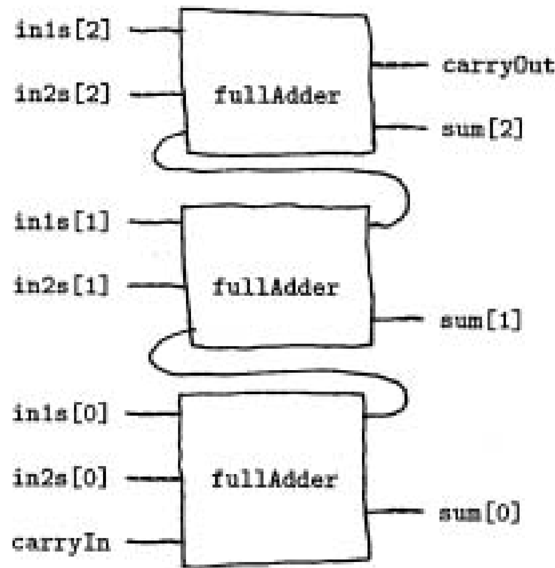

Figure 10: 3-bit Adder

- A2, B2, C2, D2, E2, F2, G2, H2, I2: the time it took you, in minutes (rounded to the nearest 5) to complete the test cases and implementation for each of stages One to Nine. If you have not started a stage, enter 0.
- H2: =sum(A2:G2)

Submit your Java source files and the spreadsheet (CSV) file through the Assignment Drop Box (ADB) at any time from the first submission date up to the final date. You must be logged into *NetLogin* under your own login to use the ADB. Do not hand in a printout of your assignment for marking. Submit all source files (ie, unchanged, changed and new)

You may resubmit if you find that you have left out a file or made a silly mistake. Bonus-Penalty marks will be calculated as of the date that you last submitted.

6 Marking

Marks will be given out of 65, awarded as follows:

- Spreadsheet: 5 marks for completing and submitting it
- The 60 marks for each of the stages will be split according to the following proportions:
  - Correctness: 60%
  - Unit tests: 20%
  - Code Quality: 20%

This assignment is to be done by you alone. No marks will be given to students who provide solution code to others or who accept such code. If you have any doubts as to what counts as individual work, please read the "Examinations Regulations" of the university Calendar and the School of Engineering Handbook and/or see the lecturer.

7 Bonus and Penalty Details

This assignment will not be accepted before Monday 16th September. It will not be accepted after 9pm, Friday 20th September. Bonuses and Penalties will be calculated as follows for the last submitted assignment:

By	**Bonus or Penalty**
9pm, Mon 16th September	10% Bonus
9pm, Thu 19th September	No Bonus or Penalty
9pm, Fri 20th September	25% Penalty