

Extending the Digital Simulator

Rick Mugridge

September 2002

1 Introduction

1.1 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.

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

Add a class *DelayGate*, which passes its 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.

1.3 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.

1.4 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 1: Delay Gate

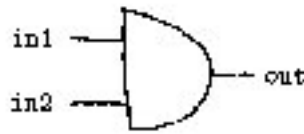


Figure 2: And Gate

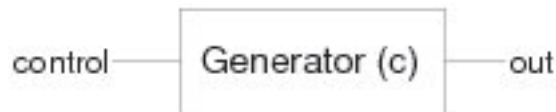


Figure 3: Generator

1.5 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

1.6 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

1.7 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);
    scheduleSignal(in2,2,true);
}
```

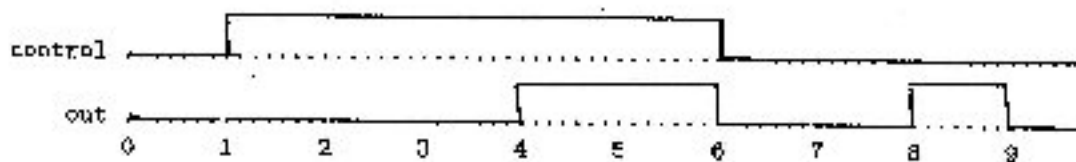


Figure 4: Timing Diagram for Generator

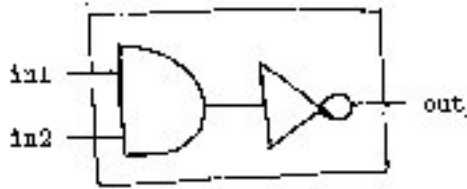


Figure 5: Nand Composite Gate

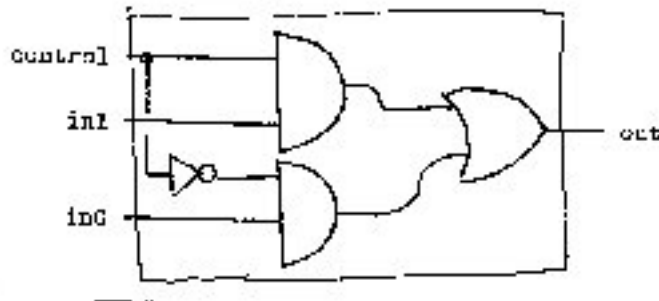


Figure 6: Mux Composite Gate

```

    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.

1.8 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.

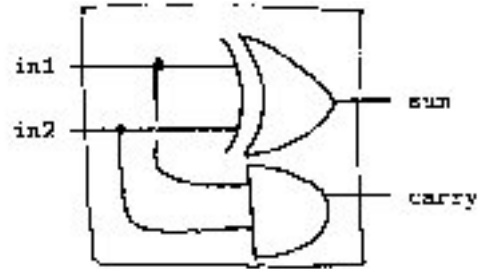


Figure 7: Half Adder

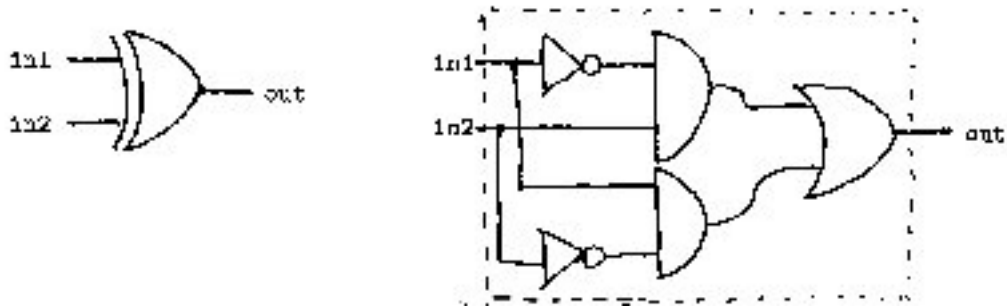


Figure 8: XOR Composite Gate

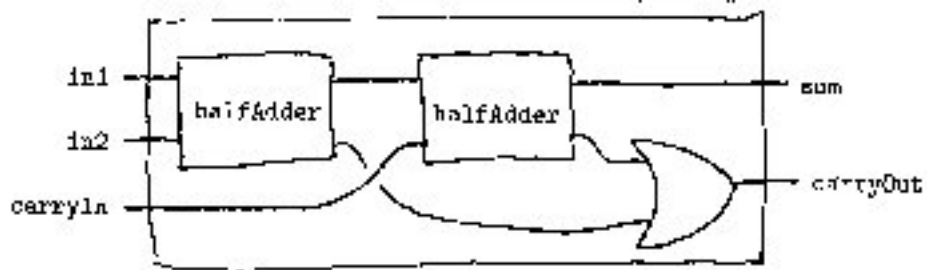


Figure 9: Full Adder

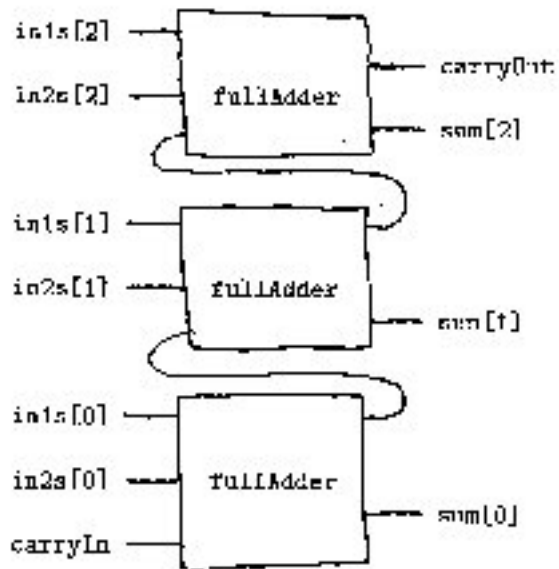


Figure 10: 3-bit Adder

1.9 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.