ECE 212 – Spring 2006
HW 1 Verilog Exercise: Introduction to Verilog Simulation

E. Rotenberg

1. Overview

The purpose of this exercise is to familiarize you with the Verilog XL simulator which we will use in this class to test digital designs.

In this exercise, you will learn how to:

- specify a logic design using two styles of verilog code: *structural* verilog and *dataflow* verilog
- apply test inputs to a design under test
- compile and simulate a verilog design
- view waveforms produced by a simulation

2. Getting Started

You have to be working on Unix computer (either Solaris or Linux operating system) to run the Cadence tools. You can do that directly in the appropriate EOS lab.

If desired, you can run the Cadence tools remotely from a Windows PC (either from home or from an EOS lab). This can be done by setting up an X-windows session (e.g., X-Win32) from the PC to an on-campus Unix computer (requires high bandwidth access). For more information on setting up remote access, see http://www.eos.ncsu.edu/remoteaccess.

To get started, make a new directory and "add cadence" to get access to the cadence tools:

```
mkdir hw1
cd hw1
add cadence
```

You may notice some information come on the screen. We will be using verilog and simvision in this exercise.

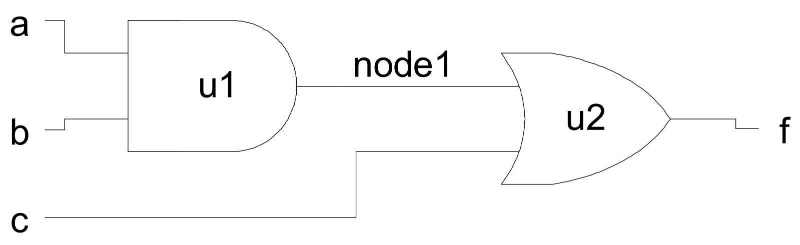
3. Example verilog design: Circuit #1

Download circuit1.v, prebuilt_gates.v, and test.v from the website.

The first circuit we want to test is shown in Figure 1a (Circuit #1). The file circuit1.v implements this circuit. Actually, circuit1.v contains two different versions of Circuit #1 in order to show you two different verilog coding styles:

1. The verilog module called **circuit1_structural**, shown in Figure 1b, implements Circuit #1 using what is called the *structural verilog style*. The term *structural* means the verilog code matches the structure of the circuit depicted in Figure 1a, gate for gate. You can see this in the code. The code "instantiates" two gates, given the names u1 and u2, of types AND2 (a two input AND gate) and OR2 (a two input OR gate), respectively. There are three inputs to the circuit (a, b, c) and one output (f). A single wire called node1 is used to connect the output of u1 to the first input of u2. Hence you can see the direct correspondence between gate instantiations and wires in the verilog code and gates/wires in the Circuit #1 diagram. This is what we mean by structural verilog.

2. The verilog module called **circuit1_dataflow**, shown in Figure 1c, implements Circuit #1 using what is called the *dataflow verilog style*. This style is quite similar to the structural style, except we don't literally use gate instantiations to implement logic. Instead, logic is implemented using built-in verilog operators such as & (AND), | (OR), and ~ (NOT), for example.



(a) Circuit #1

```
module circuit1_structural (a, b, c, f);
    input a, b, c;
    output f;
    wire node1;

    AND2 u1 (a, b, node1);
    OR2 u2 (node1, c, f);
Endmodule
```

(b) Structural verilog implementation of Circuit #1.

```
module circuit1_dataflow (a, b, c, f);
    input a, b, c;
    output f;
    wire node1;

    assign node1 = (a & b);
    assign f = (node1 | c);
endmodule
```

(c) Dataflow verilog implementation of Circuit #1.

Figure 1. Circuit #1 (in file circuit1.v).

4. Compiling and simulating Circuit #1

Make sure you have downloaded circuit1.v, prebuilt_gates.v, and test.v from the website.

Examine the contents of test.v. It contains a verilog module called **test_fixture** that instantiates two copies of Circuit #1. The two instantiations are made near the bottom of the module as shown in Figure 2. The first instance is given the name u3 and it uses the structural version of Circuit #1 (circuit1_structural). The second instance is given the name u4 and it uses the dataflow version of Circuit #1 (circuit1_dataflow). To test both instances of Circuit #1 at the same time, we apply the same test inputs to both circuits. However, notice that the two circuits have unique outputs f1 and f2. Otherwise they would conflict. Later we will want to observe the behavior of f1 and f2 as we apply different input combinations.

```
module test_fixture;

        . . .   // bunch of stuff here for testing circuits

        circuit1_structural u3 (test_input[2], test_input[1], test_input[0], f1);
        circuit1_dataflow u4 (test_input[2], test_input[1], test_input[0], f2);
endmodule    // test_fixture
```

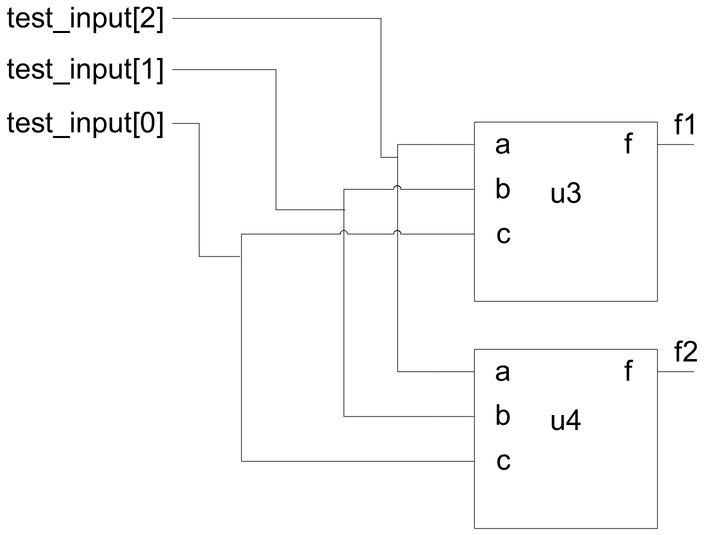


Figure 2. The module test_fixture in test.v instantiates two copies of Circuit #1 (u3 is structural version and u4 is dataflow version).

Inside the **test_fixture** module there is something called an "initial" block. *In short, the initial block allows us to apply test input combinations sequentially and run a simulation once.* As you can see inside the initial block, we apply a series of test input combinations. Each #5 directive inserts a delay of 5 time units (nanoseconds in our case – see timescale at top of test.v) before proceeding to the next test input combination.

Enough babbling, let's run it. Compile and simulate the three files together:

```
verilog test.v circuit1.v prebuilt_gates.v
```

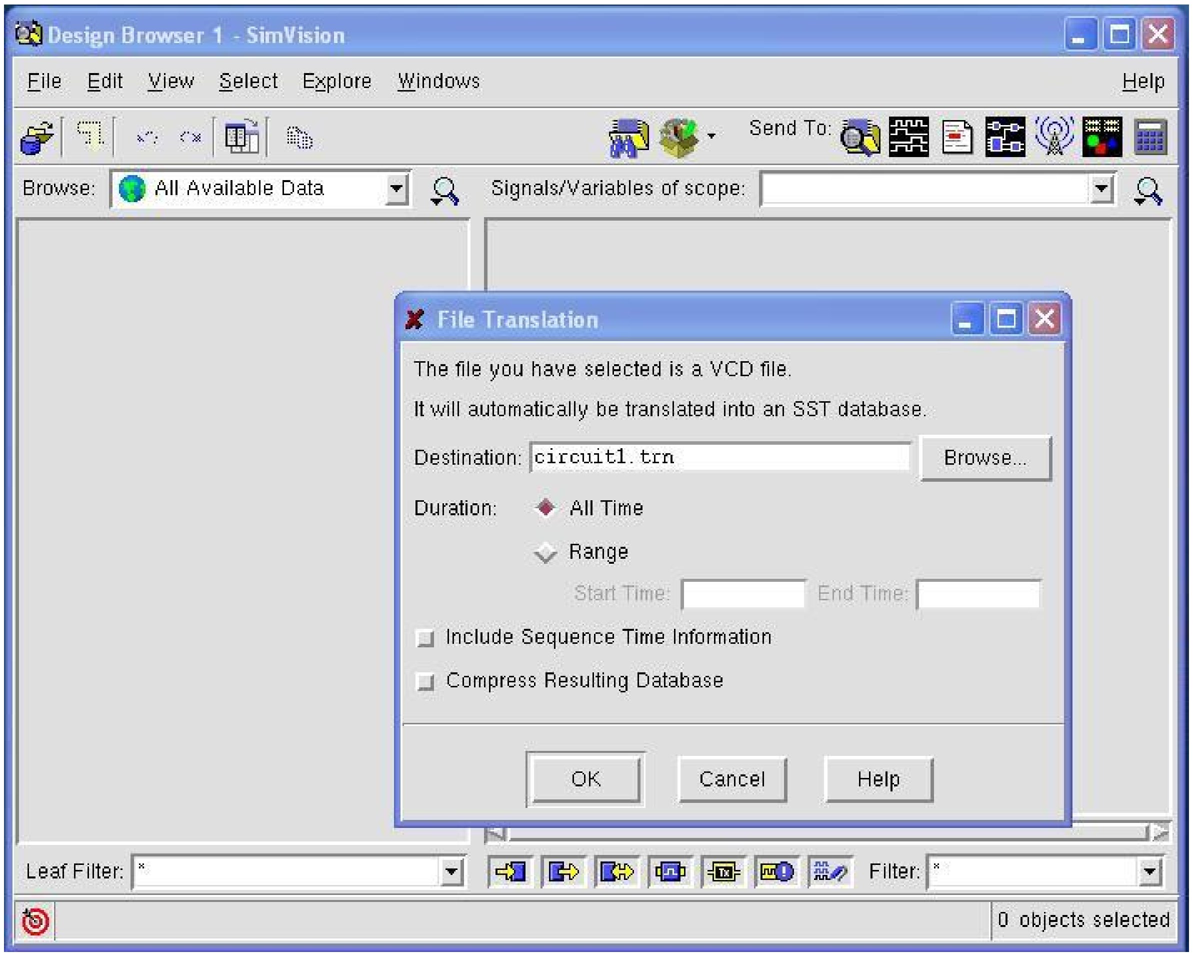
Note that test.v must be listed first because it contains the timescale command which tells the verilog compiler/simulator the unit of time. The verilog compiler/simulator should have created a new file containing the results of simulation: circuit1.vcd.

5. Viewing the waveforms for Circuit #1

Simvision is the current waveform viewer. View the waveforms by running simvision on the just-created circuit1.vcd:

```
simvision circuit1.vcd &
```

The following window will appear with a prompt about file translation:



The file you have selected is a VCD file.

It will automatically be translated into an SST database.

Destination: circuit1.trn

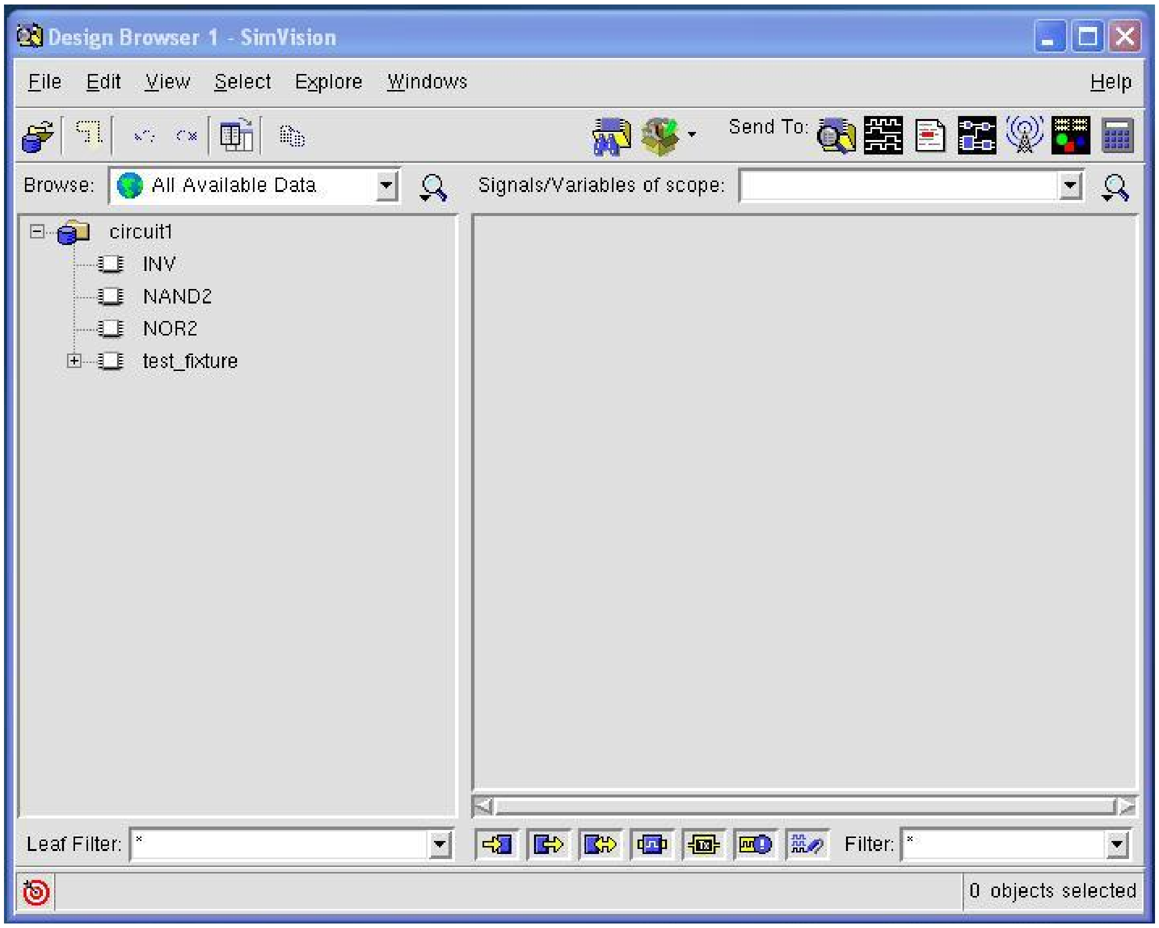
Duration: ◆ All Time ◇ Range

Start Time: End Time:

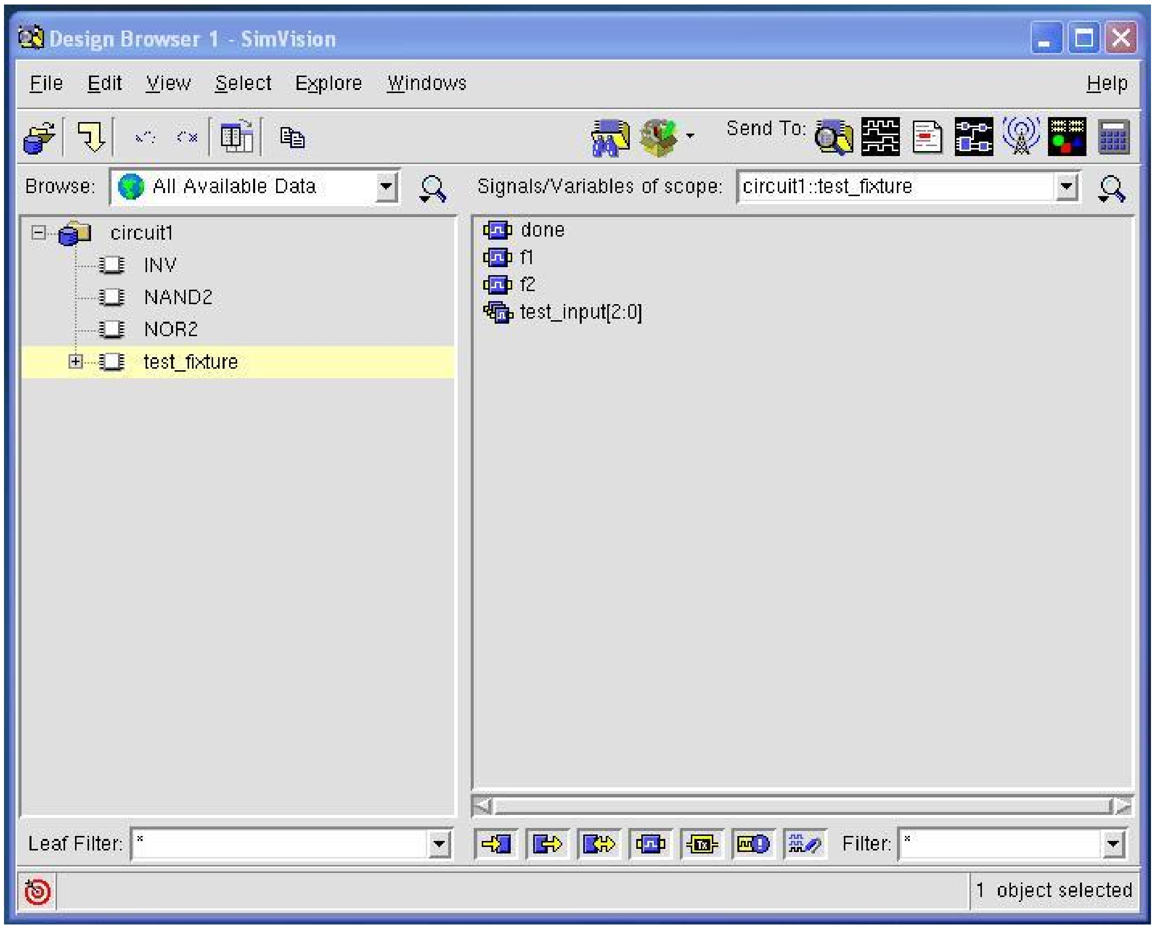
☐ Include Sequence Time Information

☐ Compress Resulting Database

Choose OK. If it also prompts you to overwrite because of an update to the database choose Yes.


Design Browser 1 - SimVision
File Edit View Select Explore Windows Help
Send To:
Browse: All Available Data
Signals/Variables of scope:
circuit1
INV
NAND2
NOR2
test_fixture
Leaf Filter: *
Filter: *
0 objects selected

This is the Design Browser which allows you to select signals for viewing. You can look at signals inside test_fixture (e.g., test_input[2:0], f1, f2) by selecting test_fixture. Your Design Browser should now look as follows.


Design Browser 1 - SimVision
File Edit View Select Explore Windows Help
Send To:
Browse: All Available Data
Signals/Variables of scope: circuit1::test_fixture
circuit1
INV
NAND2
NOR2
test_fixture
done
f1
f2
test_input[2:0]
Leaf Filter: *
Filter: *
1 object selected

Click on the button "Send selected object(s) to target waveform window" to view the signals.

When the Waveform viewer opens click on the "Zoom out full" to get a full view of the waveform.

6. Important information about rerunning verilog compiler/simulator

If you make changes and rerun the verilog compiler/simulator, and you want to view the modified waveforms, I recommend you exit the current simvision session and restart it from scratch. This is the most reliable way to ensure you view the new waveform and not the old one.

7. Exercises

Include answers to the following in your HW 1 assignment:

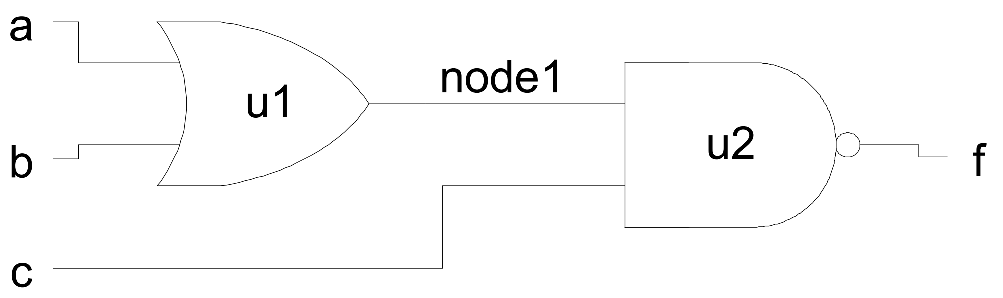
A. Use simvision to view and compare the outputs f1 and f2. **Do the circuits u3 (output f1) and u4 (output f2) behave exactly the same or differently? Is this result to be expected? Why or why not?**

B. In the file circuit1.v, within the circuit1_structural module, swap the order of the two gates u1 and u2. Recompile/resimulate the modified design by rerunning the verilog command. Then terminate your current simvision session. Then restart simvision to view the new waveforms. **How does swapping the order of the verilog statements affect the f1 output? What can you conclude?**

C. In the file circuit1.v, within the circuit1_dataflow module, swap the order of the two assign statements. Recompile/resimulate the modified design by rerunning the verilog command. Then terminate your current simvision session. Then restart simvision to view the new waveforms. **How does swapping the order of the verilog statements affect the f2 output? What can you conclude?**

D. In the file circuit1.v, modify the circuit1_dataflow module to use only *one* assign statement instead of two assign statements. Also get rid of the node1 as you will no longer need it. *Hint: You can write arbitrarily complex Boolean algebra expressions, just like we do on paper.* To confirm your verilog code is correct: recompile/resimulate the modified design by rerunning the verilog command; terminate your current simvision session; restart simvision to view the new waveforms; check that the f2 output behaves the same as before. What to hand in: **Hand in a printout of your modified circuit1_dataflow module.**

E. In the file circuit1.v, modify both the structural and dataflow implementations of Circuit #1 to implement the new circuit below. Hints: For your structural implementation, look at prebuilt_gates.v for available gate types; for your dataflow implementation, note that ~x is the proper expression for NOT X. Recompile/resimulate the modified design by rerunning the verilog command. Then terminate your current simvision session. Then restart simvision to view the new waveforms. What to hand in: **(1) Hand in a printout of your modified circuit1_structural and circuit1_dataflow modules, (2) Hand in a printout of the waveforms.**



8. Acknowledgments

This document is derived in part from “lab1.doc” from ECE 406 (authored by Dr. P. Franzon and updated by J. Abell and A. Woodson). Dr. E. Rotenberg made major additions/deletions/modifications to retool the document for ECE 212. Please e-mail bugs, fixes, oversights, suggestions, etc. to Eric Rotenberg at ericro@ece.ncsu.edu.