

University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Science

EECS 61C, Fall 2003

Lab 11: Verilog Adder

Goals

In this lab exercise you will gain experience designing and simulating a combinational logic circuit, a 4-bit adder. It will give you practice using Verilog for logic simulation and an appreciation for the issues involved in designing, testing, and debugging combinational logic circuits.

Reading

Sections 1–5 of “CS61c: Verilog Tutorial”.

Background

Binary adders are explained in detail in P&H section 4.5, and in Wawrzynek’s lecture notes on “Combinational Logic Blocks”. Both sources show how multi-bit adders are built from 1-bit adders; in this lab, you will implement a 4-bit adder out of 1-bit adders.

A truth table for a 1-bit adder provides a prescription for the operation and guides the rest of the design. Assume that the single bits to be added are represented as *a* and *b*. We denote the carry-in signal as *cin*, the carry-out as *cout*, and the result bit as *s*. The truth table comes directly from the definition of addition and appears in Figure 4.11 of P&H.

<i>a</i>	<i>b</i>	<i>cin</i>	<i>s</i>	<i>cout</i>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Inspecting the table, we see that *s* is 1 if and only if the number of 1’s in the input is odd. We may recognize this as the xor of *a*, *b*, and *cin*. The carry-out signal is also an easily recognizable function. It is the majority function—*cout* takes on the value of the majority of its inputs. As a Boolean function, $cout = a \cdot b + a \cdot cin + b \cdot cin$. This function says that the output *cout* is 1 if at least two of its inputs are 1.

For this exercise, you also need to know how to generate the overflow signal for an *n*-bit 2’s complement adder. There is an overflow if the carry-out from the most significant adder stage is different from the carry-in to that stage.

At this point, you have everything you need to know to work out the details of the adder circuit. Remember, the best circuit designs take advantage of hierarchy, and an adder circuit exhibits a natural hierarchy.

The Exercise

Based on the truth table and boolean expressions presented above, and your understanding of multi-bit ripple-adders, *design and implement in Verilog a 4-bit adder*. Your design should include three Verilog modules (all in one file). You will have a 1-bit adder module, a 4-bit adder module, and a test-bench module for your 4-bit adder.

Your design must include **gate delay**. Use a value of 1ns for every gate in your design.

Use the following module and port names for your **4-bit adder**:

```
module add4 (A, B, R, overflow);
    input [3:0] A,B;
    output [3:0] R;
    output      overflow;
```

Your **test-bench** should print out the adder input and output values, and the expected output values. For this exercise, it is simplest to display, specify in your test-bench, and print your input and output values all in binary instead of decimal. Debug and verify the correct operation of your adder with at least the following test cases:

- a) $0 + 0$
- b) $1 + (-1)$
- c) positive + positive (without overflow)
- d) positive + positive (with overflow)
- e) positive + negative
- f) negative + positive
- g) negative + negative (without overflow)
- h) negative + negative (with overflow)

Compiling and Simulating Your Design

Verilog files have the extension “.v”. The command to compile such files is

```
iverilog -tvvp -Wall -o <output file>.vvp <input file>.v
```

Option “-tvvp” indicates that the output is a complete program that simulates the design but must be run using the command vvp. By convention, we name the output file with the extension “.vvp”. After your modules compile successfully, you can simulate the design with the following command:

```
vvp <file>.vvp
```

For more information on iverilog and vvp, try “man iverilog” and “man vvp”.