Verilog & FPGA Fundamentals

ENEE 245: Digital Circuits and Systems Laboratory, Fall 2014
Lab 5

Objectives

The objectives of this laboratory are the following:

- To learn how to represent your schematics in Verilog
- To learn how to program an FGPA
- To start to understand the benefits of designing hardware via software

Verilog is a powerful language, and writing code that produces hardware makes design and debugging of that hardware far simpler than using a breadboard. As a result, you will find that it is easy to design circuits of incredible complexity. Software design of hardware (i.e., CAD, or computer-aided design) is the only way that modern computer chips could be built. To try to design and test them by hand would be nothing short of impossible.

In this lab you will design a simple 3-bit ALU in Verilog, for the Xilinx FPGA board, and interface it with the manual switches and LED lights as output. You will also design a simple 1-bit full adder so that you can do an in-depth analysis of the efficiency of synthesized circuits.

Pre-Lab Preparation

Full Adder

Design a 1-bit full adder, using either structural Verilog or behavioral Verilog or any combination of the two. As you should remember, the 1-bit full adder has the following inputs and outputs:

- **A** (a 1-bit input operand)
- **B** (a 1-bit input operand)
- **C_in** (a 1-bit input operand)
- **Sum** (a 1-bit output)
- **C_out** (a 1-bit output)

Sum is the 1-bit sum of the three input operands, and C_out (carry-out) indicates the overflow case in which more than one of the input operands is a 1.

The inputs should be tied to the switch inputs of the FPGA (these are called "sw<i>" in the user constraint file). The outputs should be tied to the LED outputs of the FPGA (these are called "Led<i>" in the user constraint file).

Design a test harness to test every possible input to your adder and verify that, for each, it produces correct output.

3-bit ALU

Design the following Verilog modules, using behavioral code:

- a 3-bit adder (has a 3-bit output and 1-bit overflow indicator)
- a 3-bit subtractor (has a 3-bit output and 1-bit overflow indicator)

- a 3-bit bitwise ANDer (has a 3-bit output)
- a 3-bit bitwise ORer (has a 3-bit output)
- an ALU that instantiates the preceding four modules and selects one for output based upon the input function code

Your ALU should have the following inputs:

- a 2-bit function code: 00 = ADD; 01 = SUB; 10 = AND; 11 = OR
- two 3-bit inputs A and B

These inputs should be tied to the switch inputs of the FPGA (these are called "sw<i>" in the user constraint file).

Your ALU should have the following output:

- a 3-bit result
- a 1-bit overflow bit that only lights up if the adder or subtractor overflows

These outputs should be tied to the LED outputs of the FPGA (these are called "Led<i>" in the user constraint file).

The ALU should use the 2-bit function input to choose which module's output gets connected to the ALU's output.

Note that the 1-bit overflow can simply be the topmost bit of a 4-bit sum/difference value, but it should not stay lit when switching from an add/subtract to an AND/OR function.

Write a test harness that checks every possible input and output combination and verifies the design's correctness. Simulate your code and bring a pre-lab write-up showing the code and simulation results.

In-Lab Procedure

Bring flash drives to store your data.

Ask the TA questions regarding any procedures about which you are uncertain.

Complete the following tasks:

- **Program your full adder onto the FPGA board.**
- Test every possible input and verify correctness. Demonstrate your working adder to the TA.
- Look at the detailed RTL schematics produced by the software; save it for your post-lab report.
- Look at the timing report that gives the pin-to-pin delays for input/output combinations of every pin. Save these reports and tables for your post-lab report.
- **Program your ALU onto the FPGA board.**
- Test every possible input and verify correctness. Demonstrate your working ALU to the TA.
- Look at the detailed RTL schematics produced by the software; save it for your post-lab report.
- Look at the timing report that gives the pin-to-pin delays for input/output combinations of every pin in every module. Save these reports and tables for your post-lab report.

Post-Lab Report

Write up your code, schematics, and lab procedures. Demonstrate the correctness of your designs through your pre-lab simulations and note any differences between what you simulated and how the circuits behaved in the lab.

Full Adder

Regarding the RTL schematic produced by the software—how did the design software synthesis your code? Where did it choose poorly, and how could it have done better? Could you have better specified your design to get more efficient results? Given the simplicity of the design, and the many possible ways of generating the circuit, there is likely to be much you can say about this aspect. Moreover, because it is a small design, you should be able to do an extremely thorough analysis. In general, what can you say about synthesis of high-level code?

Regarding the timing report that gives the pin-to-pin delays for input/output combinations of every pin—what does the design software say for the timing? How fast is it?

3-bit ALU

Regarding the RTL schematic produced by the software—how did the design software synthesis your code? Where did it choose poorly, and how could it have done better? Could you have better specified your design to get more efficient results?

Regarding the timing report that gives the pin-to-pin delays for input/output combination of every module—what does the design software say for the timing of each component? How fast is your design? How fast is each component, and how fast is the overall ALU? Calculate the delay through the MUX at the end of the ALU.