CPE/EEE 64

Lab Two: Verilog and Behavioral Modeling

Instructor:
Dennis DHALQUIST

Abstract

Two input logic gates are synthesized for the Altera Cyclone IV FPGA using the Quartus IDE. The logic gates are verified using a System Verilog testbench and Mentor's Modelsim HDL simulator

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Contents

List of Figures

I. Introduction

THIS lab will introduce Verilog's behavioral modeling ability. It is a powerful tool that allows the programmer to abstract themselves from the burdons of stuctural modeling. In the previous lab we used whats called Structual modeling, we created indivual gate "structures" and wired them together to implement the design. This is the most basic use of Verilog but bmagine creating Karnaugh maps for all 72GPIO pins, or better yet the 548 user configurable pins on the Stratix [1]. This is simply unreasonable, behavioral modeling allows the use of higher level statements like If's and Cases. if you don't know what these are, don't worry, we will explore them throughly in this lab. The purpose of this lab is to introduce the following concepts:

- Verilog behavioral modeling
- Construction of adders and Comparitors
- Verilog's constant syntax
- Verilog behavioral blocks
- Testbench assertions
- Instantiate a System Verilog module
- Use a System Verilog Testbench
- Synthesize Verilog code for a FPGA

A. *Verilog Design Entry*

Verilog is a powerful way to describe circuits. Logic diagrams like those being used in lecture can become cumbersome in large designs. "Text based design entry" can be less prone to error because it is easier to track differences in large designs. Verilog is a text based hardware descriptive language the begun being used in ASIC(Application Specific Integrated Circuit) and now is the language of choice for FPGAs(Field Programmable Gate Array). Quartus provides a comprehensive solution for testing verilog and synthesyzing it for use on a FPGA.

B. *Verilog Modularity*

One of the most important features of Verilog is it's ability to reuse a design. Reusing code allows you to rapidly assemble and test new designs. The ability to rapidly prototype a design is one the biggest advantages of the FPGA. This section will involve provided logic gate Verilog modules which will be reused in later labs to build more complex structures. Reusing these modules is very similar to how you would reuse code in the workplace to be more productive. The lab documentation comes with Verilog implementations of the four logic ates in **Source.zip** The demo for the lab will be implementing these modules with the DE0-Nano development board and testing the design on a breadboard. You could think of this as the source libraries that would be available at the company that you might work for.

C. *Test Bench for automated debugging*

Verilog roughly breaks into two halves synthesizable and non-synthesizable. FPGAs synthesis cantake a very long time, using a simulator to verify individual modules can be much faster thanresynthesizing the entire design. The Testbench also offers a unique ability to check expectedoutputs and generate test stimulus. We will use a test bench to check the provided Verilog modulesare providing the desired operation in part C of the procedure. This simulation should be verifiedagainst the known truth table for the logic gate to ensure the module is accurate.

II. Procedure: Verilog Testbench

VERIFICATION is more than half the battle when working with Verilog. Foruntunataly the language offers a number of tools to make checking your code easier. The first of which is the assertion; it will run two different blocks of code depending on if a logical condition is met, It works much like an if statement that might be more familiar.

```
//| This assertion will list an error if not met
assert (Logical statement)
  <code for true case>
else
  <code for false case>

```

The same code is used to test the adder from this lab's example code. The true case is used to display the valid output of the module. The false case throws a simulation error and shows the user the case. The logical statement in this code block checks to see if the output of the module is equal to the sum of specified constant and Number.Many designers write the test bench from specification in advance of the verilog module. Testing should be an integral part of Verilog development from the beginning.

```
//| This assertion will list an error if not met
assert (Sum == SpecifiedConstant + Number)
  $display("Case %d: Pass", Number);
else
  $error("Case %d: FAIL:%d + %d /= %d", Number,
    Number, SpecifiedConstant, Sum);

```

References

[1] Altera. (2013) Stratix family overview. [Online]. Available: http://www.altera.com/devices/fpga/stratix-fpgas/stratix/stratix-gx/overview/sgx-overview.html