18-360
Intro to Computer-Aided Digital Design
ECE Department

Carnegie Mellon

Out: 2/26/04 Project 1 Debugging Due: 3/30/04

So far we have spent a considerable amount of time in the area of modeling, simulating, and synthesis of digital systems. Part of the material had to do with writing a test bench for a system, and a few constructs and suggestions were made regarding how to do that. This is where you get your chance to try out some of those techniques — and maybe some of your own. After all, there is a certain amount of art to this.

General Description:

The project will be done in groups of two; but you can work alone if you want to. We will make a Verilog description available to you for your debugging pleasure — it is a simple calculator. The full description will not be available to you as some of the submodules are hidden. However, we will provide a description of what the circuit is supposed to be and do. You are to build a testbench for the description and find the bugs. The goal here isn't to play with it, find the *n* bugs, and then give us a list of *n* test vectors that uncover the bugs. Rather, the idea is to develop a testing strategy that would uncover a wide range of bugs, including the *n* that are in there.

The buggy description, not to be confused with some Spring Carnival literature, has "a handful" of bugs that are common to all groups. However, another handful of bugs will be individual to your group. We implement this by having a Verilog parameter in the design that turns on certain bugs. Each group will receive a different parameter number to use when instantiating this design. Later, at the demonstration, we'll turn on a few more bugs (these will be common to all). Will your tester find these too?

Each group will turn in a write-up. Also, I will ask a few groups to present what they did to the class (this part will be ungraded).

Some Particulars

We will point you to the files with the modules in it. You are to modify the tester.v file — don't change any of the others. Do not change any of the names because we will be taking your tester.v file and running it with our design. You can add in some assert modules as well — be sure to let us know so we copy those too.

If your tester finds a bug, it should print out a message with the string "ERROR" in it. When we run your tester on a calculator with different bugs, we are not going to dig through waveforms to check the results: If your tester doesn't print "ERROR, <something informative>", then we won't know you found an error! This further suggests that it is not sufficient to give inputs that cause bugs. You also need to write code that checks the output of the device under test (DUT), and flags an error if the output is not correct (i.e., you feed it 3, 4, and + and your tester should know to expect a result of 7).

When demoing, you can't endlessly debug this system. You only get 1000 clock cycles of debugging. Pick your test vectors carefully.

If you find that some errors mask others, making it difficult to find the others, send pam@andrew.cmu.edu a description of what the error is. If you've described an error activated in your design, we'll turn it off by giving you another parameter value. We'll probably set up times to do this since Pete can't be available 24/7 to make new parameters. There may be some time delay getting that updated vector, possibly 24 hours. At the final demo, we'll use your original parameter. We'll also use others to see how general your solution is.

The description of what you're testing will be somewhat vague at points. Welcome to the real world.

What to turn in

You should turn in:

- Your tester.v file.
- A report on how you approached this problem. What features did you test? How complete are your tests? Why did.you choose the tests you did? It's OK to talk about things you tried but didn't work — just explain why you tried it and why it didn't work. Or why something else worked better.

- List the bugs you found and write a brief paragraph on each bug. What is the problem found? How'd you find it?

You will be graded on…

- The generality of your approach.
- The bugs found, including the ones in the demo.
- The demo and the questions asked during it.
- Your coding style. (Be sure to help us out with comments!)
- Your write-up.
- Whether your buggy wins the sweepstakes.

Tools:

For this project, you will need to use modelsim verilog. You are encouraged to play with the waveform viewer in modelsim. It will be very useful to see the values of the signals inside the design. Documentation for modelsim (including a tutorial) is available at:

http://www.ece.cmu.edu/~ece347/verilog

To run it, type "vsim".

Setup:

To get your bug number, email pam@andrew.cmu.edu (Pete Milder), and you will get a reply with your number. Put this number into the top.v file on line 61. We'll keep track of the numbers and expect to see that number in your description.

To get the files you need for this project, copy them from

/afs/ece/class/ee360/proj1/handout

To do this, cd to the directory that you wish to work in, and type

```
cp -r /afs/ece/class/ee360/proj1/handout   .
```

In this directory, you will find several verilog files, and a directory named "work". Modelsim compiles each verilog module separately, and then simulates the design by combining the compiled modules. The compiled modules are put into the "work" directory. We did not give you the verilog source for every module in the design. Instead, we have compiled these modules, and placed the results of the compilation in the work directory.

Hand in directories will be made in the class afs space. You should copy your files to

/afs/ece/class/ee360/handin/<your userid>

before the due date. If you work with a partner, you only need to hand in to one person's directory.