Professor Fearing Debugging Checklist Fall 1998

1 Introduction

This document is written under the assumption that you would like to minimize debugging time. Minimizing debugging time requires thought, effort, hard work, and being well rested. You may be able to get your project working by spending lots of time and trying random things without thinking very hard. (If you would prefer not to think in depth about your design, this document won't be very helpful to you).

Before starting your next session of simulating and debugging, re-read Synchronous Design Rules. Sure you just read them before your last session, but hasn't your design changed a little bit? Did that quick fix your partner put in on page 7 without telling you violate one of the design rules?

There is no point trying to debug your design if you violate any of the Synchronous Design Rules listed below. In past semesters students have said things like "I know I am not supposed to have coupled Mealey machines but":
"I am just trying to get this module working first, then I will fix it" or
"I am too busy/tired to take care of that now".
Hardware won't let you get away with this approach. Poor software debugging techniques, such as changing a line of code, recompiling, praying, and seeing what happens, won't get you very far, because in hardware, local modifications aren't necessarily local.

2 Synchronous Design Rules

1. Use only one clock in your design, (using $CLOCK$ and $\overline{CLOCK}$ is fine). If you need a slower signal, such as for the serial output, use clock enables rather than a separate slower clock.

2. All inputs to your design must be synchronized, including VSYNC, CSYNC, Spare button, etc.

3. Never gate the clock, use clock enables instead.

4. Avoid clock skew, use BUFGS to globally distribute your single clock throughout the Xilinx part.

5. The only safe asynchronous control signal is the Global Set/Reset using the STARTUP block. (This signal should be driven by an appropriate RC filtered switch to ensure adequate RESET pulse width).

6. Never build your own latches in Xilinx. Coupled Mealey machines create undesirable feedback paths through the output decoders during any state transition. Even if the output is not supposed to change during a state transition, remember that the output decoders are just look up tables, and will glitch when address lines change. Synchronous Mealey machines and Moore machines are fine.

7. The minimum period for a FSM is $T_{worst-case-delay} + T_{SETUP} + T_{CKO}$. Use the *Timing Analyzer* in the Xilinx Design Manager to verify that your worst case net routing of the compiled design is less than your smallest clock period. (Note that the delay changes everytime you recompile your design, since the compilation uses non-deterministic algorithms).

3 Debugging Commandments

Anything that can go wrong will go wrong. Which means that you need to follow this checklist every time you want to debug.

1. Visually Inspect the Board
   A quick check to verify that chips are still plugged in, no wires have broken, or pins bent, is worth a minute, before you start, but is unlikely to find most bugs.

2. Check Software Functionality
   Is the computer system and network functioning? Are your files missing pieces or corrupted? (You do have floppy backups, right?) Verify that the software tools are working as expected. Compile a known working schematic; do the tools still work, such as ViewSim and EDIF? (Please send mail to `cs150@po` and `cs150support@deepthought` if software tools malfunction).

3. Check Hardware Functionality
   Are all cables, oscilloscope and logic probes working? (You can check the oscilloscope probes by using the Calib output on the front of the oscilloscope. You should see a nice square wave). Is test equipment set up for desired sweep rate, volts-per-division, etc. (Please notify your TA about bad cables so other students won't have to find them the hard way). Do you really have +5V on your Xilinx board? Verify that download cable, Xilinx board, RAM, camera, camera software, A/D and serial output are still working by running the TA project file on your board.

4. Verify that you are using the correct version of your design. Did you run `check -p design-name` to make sure you updated all the .WIR files?

5. Check System Essentials
   Is the clock running at the expected frequency? Do voltage levels seem reasonable (using oscilloscope to measure)? Are your state machines running?

6. Divide and Conquer[1]
   "Complex systems usually present designers with complex problems to solve during system debugging. At times overwhelming, may of these problems are easily solved when addressed in an orderly manner. A *divide and conquer* technique is commonly used to debug complex systems.
   First, concentrate on problem identification. Next, isolate the questionable circuitry from the rest of the system. Isolation allows you to observe output-signal integrity."

   The three keys to efficient divide and conquer debugging are:

   (a) Model Expected Behavior. What do you expect to see? ViewSim tells you what you have (with unrealistic unit delay model). It can *NOT* tell you what you need to see.
   (b) Hypothesize possible errors.
   (c) Controllability and Observability. Force module inputs to verify hypotheses, and observe outputs. Are outputs consistent with your model, or is something not functioning as expected?

[1]portions adapted from J. Ziegler, T. Hornback, A. Jordan, "The Ten Commandments of Debugging", *Electronic Design* , pp. 61-68, Sept. 3, 1992.

7. Understand Component Devices
   Are you sure you understood the data sheet correctly on that part?

8. Know your Software
   Is the serial protocol what you expect? Also, logic without loads or sources may be eliminated, and routing is non-deterministic — are you expecting that? Look at warning messages in the log file. Sometimes they matter. Make life easier for yourself by labelling every component such as U5-1 for component 5 on page 1.

9. Know Your Test Equipment
   The digital scope will give nonsense if the sweep rate is too high or too low. The logic analyzer can not tell you that a signal which appears to be random is actually just floating at 1.5V, and not connected to any output. When in doubt about a signal, verify that you have proper logic levels using the oscilloscope. For example, the oscilloscope can find a short between two outputs pretty easily since there can be an intermediate voltage level, neither zero nor one. The logic analyzer can't tell you this.

10. Take a Break/Get a New Perspective
    Debugging is mentally taxing, and we tend to focus in narrowly on particular suspect areas. Taking a break opens the mind to a wider range of possibilities and can get you out of a rut.