Eugene Ngo

1965514

10/14/2022

EE 271 Lab 1 Report

Task 1 Screenshot:

Graphical user interface, application, table

Description automatically generated

Task 2 Screenshot:

Graphical user interface

Description automatically generated

Task 3 Screenshots:

Full Waveform

Timeline

Description automatically generated with medium confidence

Partial (and clearer) Waveform

A picture containing graphical user interface

Description automatically generated

**Brief Reflection:**

In Task 1, I created a schematic diagram and simulated a full adder in quartus. The resulting waveform can be seen in screenshot 1. Then we moved up a level with Task 2 and implemented digital logic using Verilog, recreating the fulladder. In Task 2, we programmed the DE1\_SoC to fit the fulladder.sv we had created, then uploaded this to LabsLand where we could see the fulladder working in real-time on an FPGA. Then Task 3 was creating a 4 bit full adder in Verilog and then simulating it in modelsim, which can be seen in the screenshots above, and then uploaded onto the FPGA to see it working on a board.

**Reflection**

The Understanding -> SystemVerilog -> Synthesis -> Simulation -> FPGA workflow is interesting. If I break it down into sections for my thoughts then I would say that the entire workflow is very logical. It makes me feel like I am thinking like an engineer. I am curious if this is how embedded systems engineers also think about problems out in the real world. The ‘Understanding’ part is obvious: we have to make sure we understand the problem before trying to approach it and fix it. The ‘SystemVerilog’ part is probably the most confusing part for me. The step itself isn’t confusing but the language is. The next step is to code our solution in System Verilog, but learning how to code in Verilog is confusing and took the most work out of all of the parts of this workflow. The ‘Synthesis’ is when it all comes together. The ‘Simulation’ step is when your work is checked and ensures that the expected outputs are being achieved with our controlled inputs. Then finally, probably the most satisfying step is to the ‘FPGA,’ where we see the entire process come together as our work is loaded onto a FPGA board which can be interacted and played with.

I noticed that SystemVerilog is unlike any other coding language I’ve dealt with before. The testbenches in particular are interesting to me. I also had questions on minor details such as declaring bits before and after a variable which makes it unpacked vs packed. I’m curious what that means, why that’s important, and why it affects the code, since this is one of the issues I ran into. Overall though, I wish there were more resources to learn Verilog. It doesn’t seem as accessible as say python or java, software languages in general.

My main questions are: “How would you recommend getting better or more comfortable at SystemVerilog?” and “Are there supplementary resources or online resources I can use to either practice or improve?”