**Jumping back in**

The last time I had written any Verilog was when I worked as a Digital Logic Teaching Assistant back in the Spring 2021 semester of school. Since then, I have done a lot of work on Iowa State’s solar car team, PrISUm, learning many skills related to PCB design and system wide testing. As I look ahead towards getting my Masters at Iowa State, I have thought about learning more about FPGA’s since I enjoyed my time in Digital Logic learning about combinational logic, sequential logic, and state machines.

To get myself more familiar with the basics of digital design and Verilog, I decided to buy the Basys 3 Artix 7 FPGA development board sold by Diligent. This board comes with a 100 MHz clock, 16 switches, 16 LED’s, 5 buttons, a USB Host, UART Bridge, VGA output, and seven segment displays. Between all of these features, I figured that I would be able to apply what I have learned in my digital logic class and go beyond it to more topics based on my personal interests and future classes.

On this specific page, both videos cover demos and conversation about my time working with designing modules and testbenches related to combination logic, sequential logic, finite state machines, UART communication, and PS2 keyboard receiving interfaces. As a guide, I have been following through the first 9 chapters of the textbook “FPGA Prototyping by Verilog Examples” by Pong P. Chu. This book has helped expand on the syntax that is both synthesizable and not for Verilog, while also explaining useful design practices. I have found this extremely beneficial for modules involving UART, PS2 communication, and complex state machines. Normally in each chapter, the concepts and basic outline of a module will be defined, with extra experiments added at the bottom to expand upon what is given. Through my work over two months, the code presented in these demos are from those experiments. If you are interested in the modules and testbenches I have designed, feel free to access my GitHub repository below!

**Creating a good testbench**

One of the largest challenges working with Verilog was designing quality testbenches. In the beginning when I was testing 1-bit comparators or combinational logic, the testbenches were fairly straightforward. As my code became more complex though, that is where I think the learning began to speed up. When working with modules involving UART or PS2 receiving systems, I found a steep learning curve in deriving effective code in Verilog. I started to design separate always blocks to drive a clock based on a parameter designated period. I also had my asynchronous reset be set in an initial block once at the beginning of each test. I also thought about edge cases where I would force the largest and smallest inputs to see what would happen.

As I moved to testing state machines, things also became trickier (in a fun way). When the output data I was expecting came out wrong, I had to think about both why and where that data was outputting from. This is because the code could have forced my module into the wrong state at the wrong time! I started to add more output ports with useful information like the state register or additional internal signals that were not meant to be outputted temporarily. This was extremely beneficial, as sometimes I would find errors such as not instantiating the state register with enough bits, so it would only alternate between the first two states instead of all four. That one made me facepalm pretty hard after staring at waveforms for an hour.

Looking ahead, I would like to try and automate more with my testbenches. I am beginning to see a trend that I am spending much more time writing code for my testbench than I am designing the hardware. I would like to incorporate more tasks and event triggers within my code, and take full advantage of the done ticks of my state machines for hold times. The way I see it, the more I can test before synthesizing the better, since it is much easier to debug waveforms and edit the testbench. I do think that it is extremely satisfying when tests come out just like you expected it to. This is especially the case with the nature of HDL and how every little building block comes together to build something interesting and challenging.

**Synthesizing to my first board**

After taking and teaching my Digital Logic class online during COVID-19 at Iowa State, I assumed that synthesizing would be my biggest challenge while working through the textbook I had. To start out with synthesizing and creating bitstream files on my FPGA, I began with small designs such as driving the LED’s or connecting them to buttons. I believe that if I started with more than I could chew, then I would just have gotten slowed down later down the line. By being confident in the small building blocks that I created, I was able to work myself up to creating more and more fun modules!

Working with a 100 MHz clock made me consider many problems when interfacing a board with the real world. Before this, I never considered clock dividers too much on hardware. I now had to consider if I wanted something to display or update slower so that it would be practical to interface and test a module. This is very apparent when looking at the basic state machines I created that make fun patterns to display on the 4 7 segment display module on my Basys 3 board.

Another problem I had to break down was the idea of debouncing my button inputs to avoid glitches. I have previously done this while working with Arduino projects or microcontrollers by adding in hold times, but as I learned with FPGA’s is that you cannot wait like that easily in hardware! So, I had to design a state machine that would wait a set amount of time before reading the next input to create a debounced output. I also worked on creating a module to provide a rising or falling edge tick based on a button press, so that I could manipulate state machines easier without worrying about an input running through for too long.

All of these problems were super interesting to work through, and I love the idea of combining coding in a “hardware” mindset. I believe that with my skills between hardware, software, and systems design, I would excel at working with FPGA’s in the future. In upcoming semesters, I am planning on taking some classes relating to signals, computer architecture, and embedded systems. With all of these, I hope I can further improve my skills involving FPGA’s, because this type of work is very rewarding.