Getting to Know your Environment!

I initially prepared for this lab by watching the four Diligent Xilinx video tutorials that were posted on the class website. After I had a grasped a basic understanding of the Xilinx ISE toolset, I read the lab handout that detailed the procedure for this lab. When I arrived at lab I observed Jim Frenzel give a presentation on how to use the Xilinx ISE to complete the assigned lab activity. I then opened the Xilinx ISE myself and created a new project while being careful to select the correct device family, device, and package type. I began by drawing the gate level logic needed to implement the assignment using the Xilinx Schematic tool. This schematic included a global input buffer (IBUFG) wired directly to a NOT gate. The input of the circuit was connected to IBUFG and the output was connected to the NOT gate. I also printed the schematic once it was complete.

I then created a “user constraints file” to ensure that my input was connected to P15 on the D2XL board and my output was connected to P38, which connects to the LED. I then ran a pre-synthesis simulation and printed the results. After I had successfully ran my simulation and verified that the results were consistent with the goals of the assignment, I prepared to generate the programming file by setting the FPGA start-up clock to the “JTAG Clock” in the “startup options tab.” I then generated the programming the file and opened the program needed to upload the programming file to the D2XL board. After I had successfully uploaded the file, I tested the board to ensure that the button had the desired affect on the LED. I then generated and printed a design summary for my project and had the lab TA verify my work. This lab was relatively straight forward and I did not have any major problems completing it. The most difficult part of this lab was discovering how the Xilinx ISE required me to select a certain file in the upper box before it would show me the desired operations in the bottom box.
Digital Logic Lab 1, Getting to Know your Environment, focused on learning how to use the tools used to program the Xilinx Spartan II FPGA. These tools, the Xilinx ISE toolset, are used in programming the D2XL boards used in the ECE241 lab. Thus, it is important to become comfortable with the toolset that will be used throughout the semester.

After reading the lab instructions, I viewed the training films online to familiarize myself with the tools in a general sense. I also read the online documentation of both the D2XL FPGA board and the DIO I/O board. These utilities are extremely helpful, as they give the student a glimpse of how to use the ISE toolset before we entered the lab environment. This pre-knowledge is much more powerful than a in-lab demonstration, because the student can process what is learned outside of lab for much longer than what is learned in lab, thus the student is more familiar with tools.

The in-lab demonstration of how to use the tools to program the FPGA to control a LED off of a pushbutton input was also very helpful, as it demonstrated the proper procedure of how to go through the lab. It also showed us, the students, exactly where we had to go to accomplish the objective for this lab, as well as continue in the student’s familiarization with the Xilinx ISE toolset.

Then the student attempted to program the FPGA with the ISE toolset. The student created a schematic in the toolset to define exactly the logic combination that was needed to implement the design. The student then defined a simulation waveform file, and simulated the design. This simulation predicted that the design would work as expected.

The student then synthesized the design. However, due to an oversight, the student forgot to define the pins needed for the FPGA to perform its duties properly. This oversight was corrected, and the design was re-synthesized. The student then printed the Design overview, and has the instructor sign the design overview. Finally, the student programmed the FPGA and verified that the design was working properly.

The Xilinx ISE tools are very easy to use, and apart with some difficulties finding the design overview report, the student had no major problems. By far the most helpful utility to learning the toolset was seeing the instructor demonstrate the software before the student used the toolset. This demonstration, coupled with a strong rounding in other CAD programs, such as OrCAD and Cadence, allowed the student to avoid most major problems.
Introduction

This lab incorporates the use of a finite state machine as a means of controlling a microprocessor. The FSM in many ways serves as the “brain” of the microprocessor in this design, handling the routing of signals into, out of, and through the arithmetic logic unit designed previously. Output signal routing and clock division is given in lab instruction, allowing students to focus on the generation of a FSM.

Description

For my FSM, I began by analyzing the code given by the professor, allowing me to determine the boundaries I was working in to construct this FSM. Next, I constructed a simple state diagram showing the “big picture” of how data would need to flow through the system, using the functional block diagram as a reference.

Next, I spent a couple of minutes contemplating how to differentiate between the different operations to be performed. A system was quickly recognized that there are only four basic operations: stop, load, move, and arithmetic. The multiple arithmetic functions performed by the ALU are similar in the fact that all will require the same sequence of loading registers.

Using this knowledge, as well as the knowledge that I could define a new type for additional readability of code, I set off using an eight bit binary vector to encode the states. With the simple transitions, and due to some timing hazards encountered later in the design process, the load ir and load in registers are loaded at the time the go button is pressed, regardless of what operation is called. This simplifies the circuit design as well as shortened the cycle time for the load a operation.

Final testing done in Modelsim and on the FPGA chip was successful. One interesting note here was I discovered the synthesis report can be helpful as a debugging tool, as it allowed me to see a mistake in my code where I had “unreachable” states in my FSM, due to an error entering a state in the code.

Conclusion

This was the best lab as far as developing a better understanding of the operation of a microprocessor. I did not find this lab to be too much of a challenge, although I was considering writing my own interface code as well (light the lights divide the clock and such). The only advantages I can see to doing this (students write the code themselves) would be the re-use of the seven segment display code (again) as well as the use of shift registers and clock dividers, which seems to be a concept not very well understood. Maybe a portion of the lab could be devoted to experimenting with clock dividers (say, along with the ff/latch lab) where students would run longer models to see the effects of 3 bit to 8 bit clock dividers on how much the clock is divided. They would also explore how the xor locations affect the clock division.
Introduction

For our 9th laboratory session, we explored the differences between a flip-flop and a latch, created with VHDL. Both the simulated flip-flop and latch were to accept two inputs: a single "D" input shared between both components, and a shared "Clock" input. The premise was to observe the difference in behaviors between the flip-flop and latch using several different simulations. Those results will be discussed in the proceeding sections.

Procedure

To prepare for the laboratory exercise, I read section 4.4 in our text, which demonstrated how a flip-flop could be modeled in VHDL using a simple process. Upon arriving in the laboratory the next day, I created a new VHDL module with \( d \) and \( clk \) inputs, and two outputs, \( q_{ff} \) and \( q_{latch} \). Each of the outputs corresponded, respectively, to the flip-flop and latch outputs. After declaring the necessary outputs, I formulated two separate processes: one to dictate the action of the flip-flop, and the other to govern the latch. The flip-flop process was an exact replica of the one demonstrated in the textbook, using the \texttt{RISING\_EDGE(clk)} notation to dictate when the inputs would be sampled. For the latch, I had to draw upon my previous knowledge to construct a process that accurately modeled the activities of a latch. This turned out to be relatively simple, however, since the latch process differed from the flip-flop process only in that the latch copied the value of \( d \) onto \( q \) whenever \( clk = '1' \), instead of only on the rising edge of the clock. After running my first two behavioral and post-route simulations, I noticed my latch was not accurately copying the \( d \) input while the clock was high. This was quickly remedied when it was
explained that, along with the $clk$ input, the $d$ input for the latch needed to be included in the processes sensitivity list. That is, just like the $clk$, the input $d$ would be able to change the $q_{latch}$ output whenever $clk = '1'$. Finally, with both the behavioral and post-route simulations working perfectly, it was time to perform the last simulation, which had been created by Jim Frenzel to model what happens when the timing requirements for a latch or flip-flop are not satisfied. In other works, what occurs when the clock period is small enough to violate the setup time, and the input changes close enough to the clock edge to potentially violate hold time. From the waveform, both the flip-flop and latch outputs change somewhat erratically in response to the $d$ input because of the close proximity of rising clock edges. At one point, the input fell too close to a rising clock edge, so the latch output can be seen going metastable. Violating the timing requirements for the flip-flop and latch had a drastic effect on the outputs, causing the circuit to malfunction.

**Conclusion**

Modeling these sequential circuit elements with VHDL was an interesting application of the topics we have been learning in lecture. Beyond simply demonstrating how a flip-flop and latch are created, the exercise showed how powerful and useful the VHDL process is in creating sequential logic. Also, being able to observe actual timing violations in a circuit was useful in that it allowed me to actually visualize the concepts we had most recently discussed. Timing of sequential logic seems less ambiguous now that I have observed it in a working circuit.