Lab 4: Computer Organization and Data Path Design

CSCE 312.506

Akash Kundu, Julian Immanuel, Cole Boggus, Robert Paraliticci

Abstract: In this lab, we were tasked with three distinct problems. In the first task, we learned how to analyze a problem, create a corresponding finite state diagram, and then derive a design from that. In the second task, we learned how to use memory components such as RAM and ROM, how to design basic IO interface and how software can mimic and even work together with hardware. Finally, in the third task, we learned how to implement and design basic ALU components such as the adder, subtractor, comparator, and barrel shifter. By doing all these tasks, we learned about the basic process and considerations for a microprocessor based system.

Results and Discussion:

Problem 1:

- **Part 1:** In part 1 of this problem, we created a Finite State Diagram that showed if the right signal was activated then the last LED would be activated and then in descending order one after another, the rest of the LED's would be activated until either the right signal was turned off or the last LED was activated and then the first LED is activated to loop back and start again. The same process would happen if the left signal was turned on, but instead of the last LED being activated, the first one would be activated and instead of the LED's activating in descending order they would be in ascending order order.
- **Part 2:** The design is made up of 4 major components: inputs, counter, decoder, and LED array. The inputs are made up of a switch for left and a switch for right and a clock. Next, the counter is made up of two main components: an incrementer for when the left switch is activated and a decrementer for when the right switch is activated. Next, the decoder contains the logic that turns on and off the appropriate lights. Lastly, the LED array which consists of 20 LED's. The counter was the key to this design. The counter starts at the first or last LED depending on if the right or left signal was on and then activates the next adjacent LED in order to get the effect we were tasked to achieve. However, just like in the State Diagram if the right or left single is switched off, then the LED's will be deactivated and the counter will be reset.
- **Part 3:** This design is very similar to the design from part 2, except for the counter and instead of a decoder, there is a ROM. The ROM is different than the decoder because instead of using logic to turn on the appropriate LED's, the ROM contains addresses that correspond to each LED. Moreover, the counter was modified so that when it was going from LED to LED, it simply increments or decrements the addresses to go from one LED to the next.

Problem 2:

- **Hardware:** In the design, there are 7 main components: an 8 bit unidirectional address bus, an 8 bit bidirectional data bus, ROM, RAM, I/O block, and 6 bit control line. The ROM is a memory block that can only be read from. The data read from the ROM is used with the RAM which is also a memory block that can be written to and read from. The 8 bit unidirectional address bus is essentially a one-way highway for address information to travel by. Moreover, this address bus carries information from the ROM to RAM and then to the I/O block. The 8 bit bidirectional data bus is similar to the unidirectional bus except data can travel both ways on this bus. Furthermore, the bidirectional data bus can take information from the ROM, RAM, and I/O block and deliver it to the CPU as well as deliver data to the I/O and RAM. The control line dictates whether the RAM is being read from or written to and if the ROM is being read from. Lastly, the I/O block takes input and output and trades data with the CPU using the data and address buses.
- **Software:** The software written for problem two is a behavioral model written in C of the implemented design. Because the circuitry we were tasked to complete still requires the user to tell it whether the ROM or RAM should be accessed and where the user wants to write in the RAM, the program mimics this behavior by prompting the user to choose whether to write to RAM or read from RAM or ROM. Notice there is no option for ROM to be written on as the actual hardware ROM is read-only. In the design, data is read from the ROM and then written into a specified address in the RAM. The program functions the same way. Information can only

be read from a specified address in ROM and information can be read and written from specified addresses in RAM. Since ROM only has memory addresses from 0 to 255, the program will keep asking for another address if the inputted address is outside that range. Moreover, since the value of the data can be from 0 to 256, if the inputted data is outside that range the program will keeping asking for another value.

Problem 3:

- **Adder:** The adder takes two eight bit inputs and adds them together bit by bit using one half adder and seven adders in a ripple-carry style. After all signals have propagated, the adder will output the addition of both inputs. If desired, one can check the output for integer overflow.
- **Subtractor:** The subtractor uses the design of the adder but first converts the subtrahend to two's complement form by inverting all of the bits and then adding one to the subtrahend. After this, the minuend and subtrahend are added together using the adder design, giving the result of subtraction.
- **Comparator:** The comparator checks each bit of two inputs starting from the greatest bit to the least bit and checks for inequality. Each bit is compared and then the comparison is output to the next bit comparison. If an inequality is found, then there are no more checks that can change this and the end result is passed all the way to the final output. If no inequality is found, then the inputs are equal.
- **Shifter:** The barrel shifter uses a combination of a one-bit shifter, a two-bit shifter, and a four-bit shifter. The combination of these shifters can shift all permutations from 0 to 7 bits. For example, if three bits need to be shifted, the first shifter shifts one bit, and then the second shifter shifts two bits. This gives a total of three bits shifted. This can be done for all permutations. There is also a direction bit that indicates if the bits are to be shifted to the left or to the right.

Conclusion: In this lab, we designed several larger and smaller circuits that taught us va lot about microprocessor design and computer organization. Problem one allowed us to take a problem, rationalize it and draw it into a finite state machine which we then turned into a circuit design and even learned about how to use ROM memory and its advantages over a decoder. Problem two allowed us to analyze a practical example of a computer circuit and recreate the circuitry involved to transport the data to where it is needed. Furthermore, it allowed us to create the same circuit programmatically in C, mimicking the behavior of the hardware circuit created in Logisim. Lastly, problem three allowed us to implement smaller, but very useful computational circuitry such as adders and shifters that are utilized in ALU's and other computational components. This lab taught us about how data moves in a simplified microprocessor circuit, what each component's role is in this circuit and how software relates to hardware in the context of a microprocessor based system.

References:

TA: Pritam Majumder
http://www.cburch.com/logisim/docs.html
http://cs.nyu.edu/~gottlieb/courses/2000s/2000-01- fall/arch/class-notes.html
http://courses.cse.tamu.edu/yum/Spring2017/312/Lectures.html
http://en.wikipedia.org/wiki/Three-state_logic
http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/CompOrg/tristate.html