Lab 5 - Basic CPU

Authors: Catherine Nguyen and Jonathon Watson

Documentation: EI with Lt Col Trimble

Purpose: The purpose of this lab is to write, test, and implement in hardware, a basic CPU that is capable of addition, subtraction, AND, OR, and shift operations.

Design: The CPU in this lab works by performing operations in multiple cycles. It uses four cycles using the center button of the BASYS3 board to complete an operation. The first push of the btnC activates a 4-digit seven-segment display and loads the 1st operand in a register and displays it. The second push stores the 2nd operand in a register and displays it. A third push displays the output of the operation and lights LEDs if there is a sign, zero, or carryout. Lastly, a fourth push will clear the seven-segment display.

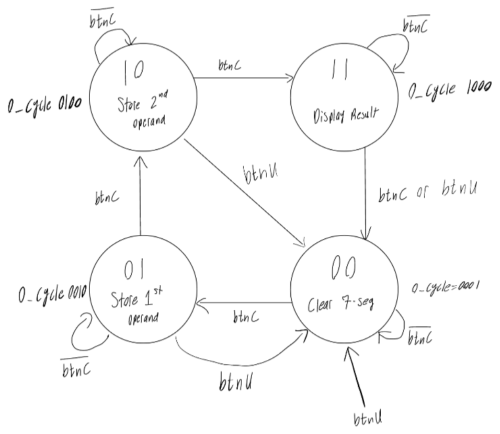

Diagram 1: Finite State Machine

Our CPU is driven by a clk input and a cycle output from the controller_fsm. The controller is essential to the entire design because it drives which inputs are passed to the ALU. For example, during state 0001 (equivalent to state 00 in diagram 1), the display is cleared, and nothing is passed. However, during state 0010 the input value from the switches is stored in register A and then passed to the ALU and displayed on the seven seg decoder. After a signal is passed to the ALU, a calculation is completed based off the opcode implemented in the ALU logic. This result is then passed to a component called twoscomp_decimal which breaks the result into a sign, hundreds place, tens place, and ones place. These different components of the result are passed to the TDM, which are then sent to the 7-segment decoder and its' anodes using MUXs.

Our group took a multiple-demo approach to this lab. Task A involved implementing the ALU's addition, carry out flag, and zero flag functionality. We used our completed prelab schematic's show in diagram's 2 and 3 below to implement the top-level code as well as the individual code for the ALU component. Sadly, we did not read task A correctly and displayed decimal values instead of hex.

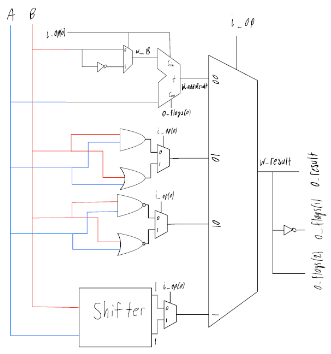

Diagram 2: ALU Schematic

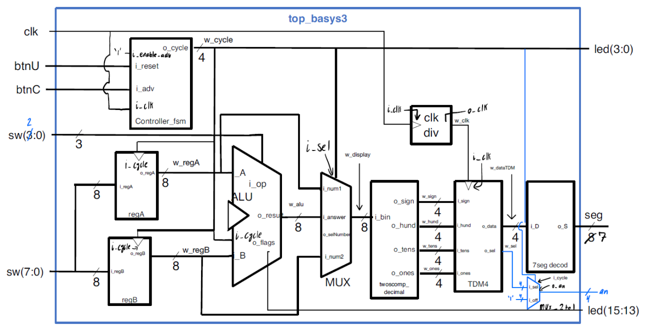

Diagram 3: Completed Top Level Entity

Task B involved implementing addition and subtraction while also displaying the result as an 8-bit number (that also included negative values). Task C included additional logic for bit-wise OR, bit-wise AND, left logical shift, and right logical shift on a 8-bit two's complement operand. When implementing task B and C in code, the group decided to combine the two tasks because once we got the and/or functionality working, it was easy to write two more lines of code and implement the shift. Additionally, we included NAND and NOR functionality in our final design because we did not want to leave two lonely states unused in the MUX.

During the lab, the group encountered multiple problems. Initially, the lab was very confusing because there were a lot of components, and the group did not fully understand the ALU. However, it became clearer as the lab went on and we had discussions with Lt. Col. Trimble about how we could improve upon our prelab to make implementation easier. One improvement was creating flag signals that we passed to LEDs directly from the ALU result. Another issue we had was an error where the carryout LED light would not turn on at the correct time. The group found that the error was caused because the operand "-" was being used to compute subtraction and carry out results. We changed that logic to the addition of the first number and the two's complement of the second number plus 1 which solved the problem.

The group succeeded in learning about new components that were not previously implemented in other labs. Although the group found the ALU to be the most frustrating component to code for, it became a huge milestone once completed. After determining that most errors were caused by the incorrect usage of a "-" operand, most of the logic was easy, and we even implemented the functionality for NOR and NAND for fun. Most individual components were made with the inputs and outputs shown in diagram 2. One difference between diagram 3 and our code is that we decided to implement logic for MUXs in our top-level code rather than create separate components for each MUX. We did this because it was easier to implement logic than create a component.

The CPU in this lab is operated on assembly code. This is readable to humans and represents specific machine instructions and operands. However, it must be translated into machine code which the machine uses to understand and execute. This occurs by converting the assembly instructions into binary code. This can be specifically seen when calling the 6 (+2 for NOR and NAND) operations involved in the lab (such as addition, subtraction, etc.). They were each assigned a 3-bit opcode which dictates their function. The assembly code was able to interact directly with the hardware by fetching the machine code generated and stored in the computer's memory and performing the specified operation.

Conclusion: This lab overall provided more insights on how to code in VHDL. It was particularly challenging to ensure that all components worked individually and as a whole in the top_basys3. For future labs, it would be beneficial if students were given a better understanding of how to code for an ALU component. This could be done by introducing this component in an earlier lab or with more guidance as most issues in this lab were related to this component.

Total hours spent on the lab: 20+