CPEG324 Lab 2: VHDL Components

By: James David and Ethan Conway

Abstract: The objective of the lab is to design and develop several different VHDL components, including a multiplexer, a 4-bit register, and a 4-bit adder/subtractor. We are then to develop sophisticated schematics of our designs, and test them using a test-bench that we design and develop ourselves.

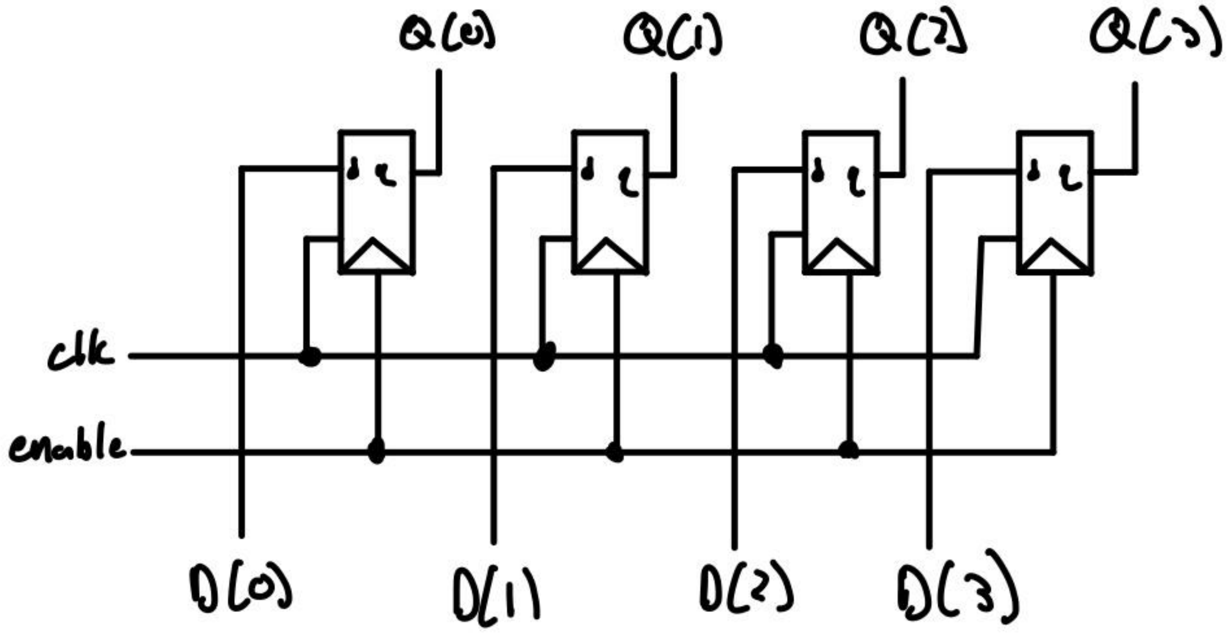
Division of Labor: The labor for this project was mainly broken up into the component and test-bench design. James was responsible for most of the component design and coding for that, and Ethan was responsible for most of the test-bench design and the coding for that. We each helped each other on both parts, and the final lab report was worked on by both members.

Detailed Strategy/Results:

- **Problem #1:**
  - Explanation: For Problem 1, we decided to implement the 4 to 1 mux logic using a with select statement. We use the select bits as an input to decide which of the 4 n-bit vectors to output. To make the Mux accept any length of input, we used the generic parameter. This allows us to select any length for the input and output vectors to be, from 1 bit to any length of bits we need. For the testbench of the mux, we decided to test the mux with 2 different sizes of input, 2 bits, and 4 bits. The testbench of the 4 bit test is provided, and the waveforms of the 2 bit and 4 bit tests are also provided in Appendix II and III.
- **Problem #2a:**
  - Explanation: To implement the 4 bit register, we created a vhdl entity with a 4 bit D input, a 1 bit clock input, a 1 bit enable input, and a 4 bit output. To simulate

checking for a rising edge, we used the rising_edge() function provided in the vhdl libraries. Then, we check for enable being set high. After both a rising edge and enable being set high will the register's value be set to the input. Then it will output what value it currently has. The Schematic below shows how the individual bits would be set. Our testbench tests all values of clock and enable, plus 3 different values of D.

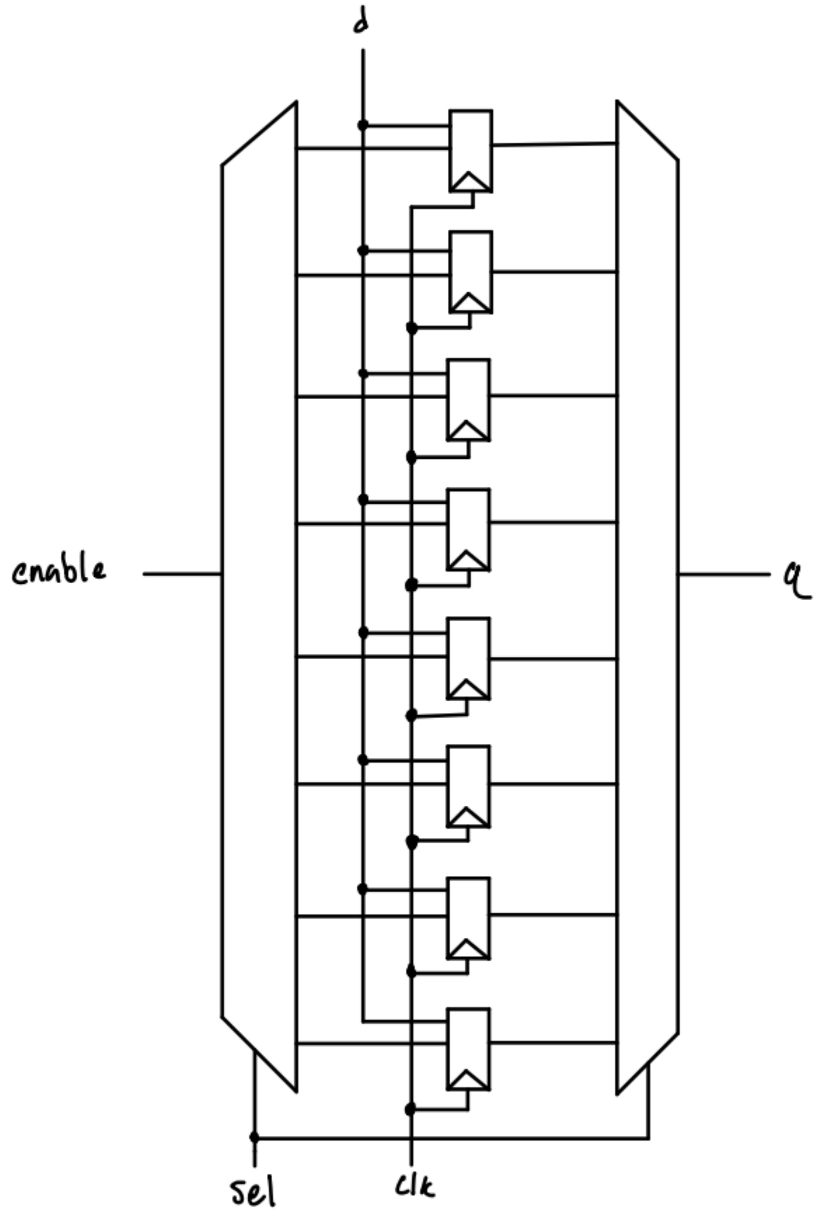
- Circuit schematic:



- **Problem #2b:**
  - Explanation: To implement a vhdl file with 8 of the 4 bit registers from part 2a, we decided to create our circuit first then base our vhdl implementation (reg4n8.vhdl) off of that. After deciding on our plan, we drew up the circuit diagram below. We landed on this design by thinking how the registers would need to be triggered to read and write to. Our circuit works like this: Our inputs

are D, a 4 bit vector, clock, enable, and select, a 3 bit vector. We have D and clock wired to all registers, and wire enable and select to a 1 to 8 demux. Select signifies which register we will read/write to, and enable is what the demux will output to each register. The registers will output then to a 8 to 1 mux, which has a 4 bit input to hold the 4 bit outputs of the registers. The mux has the 8 outputs and select wired to it, then using select it outputs the correct registers contents to Q, a 4 bit output. We created demux and mux components to implement this circuit. The files for all components, the testbench, and the waveforms are found in Appendix II below. Our testbench tests all values of clock and enable, along with reading and writing to all registers.

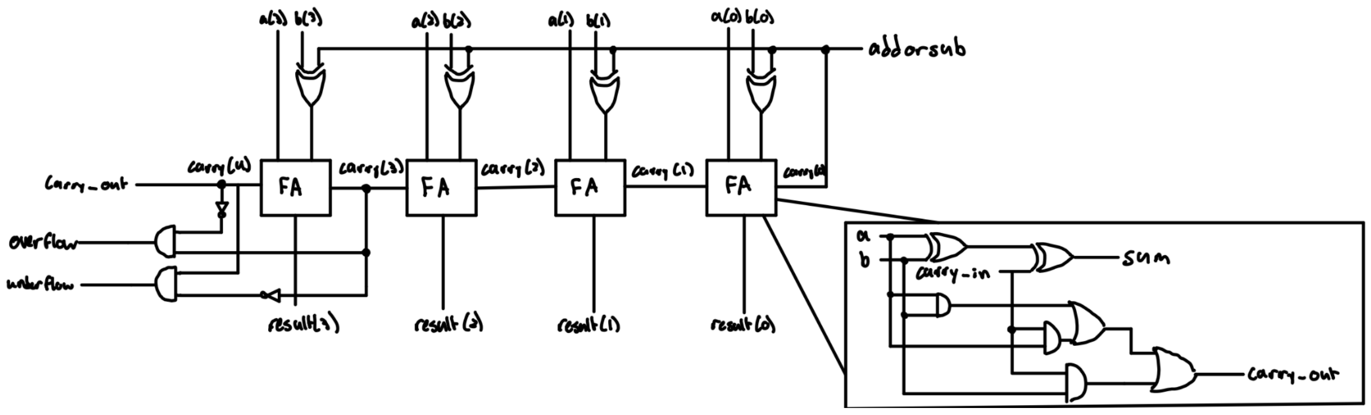
- Circuit schematic:



- **Problem #3:**
  - Explanation: To implement the 4-bit signed adding and subtracting circuit, we started by creating a full adder component, then creating the vhdl logic using 4 full adders wired together. Our inputs are: 2 4-bit signed integers to decide which numbers to add or subtract, and a 1 bit input called addorsub. To decide if the

circuit is computing addition or subtraction, we use the addorsub bit. If the bit is set to 0, then the circuit does addition and nothing about the inputs is changed. If the bit is 1, then the circuit will compute a subtraction operation. The overall operation will convert the second input to two's complement and add the first input and the converted second input. To start the two's complement conversion, the addorsub bit will be fed into a XOR gate with all bits of the second input, which will flip all bits, then be the first carry bit in the first full adder, representing the plus 1 needed for 2's complement conversion. The final carry bit is an output, along with a computation of overflow or underflow using AND and NOT gates. Overflow occurs when the final carry bit is 0 and the second to last carry bit is 1, and Underflow occurs when the final carry bit is 1, and the second to last carry bit is 0. We use NOT gates to flip the 0's needed to check if overflow or underflow has occurred using AND gates. Our testbench tests basic addition, subtraction, two's complement conversion, overflow, and underflow.

- Circuit schematic:



Conclusion: Overall, I believe that we completed this project to the best of our ability. I believe that we successfully designed and developed each of the VHDL components that were assigned to us, and that we also successfully designed and developed circuit schematic designs for each of these components, and tested them thoroughly through the test-benches that we designed and developed ourselves. Given more time and resources, we could potentially implement these components that we designed into more complex circuits, and use them for higher functionality than they are currently used for. The area of this project that gave us the most difficulty was using the 4-bit register and using it to implement a structural VHDL model for an 8 4-bit register file. It was tough to understand what was required of us, and how to fully implement what was needed, but I believe that we did a good job of implementing what was needed in the end.

Appendix I: Notebooks

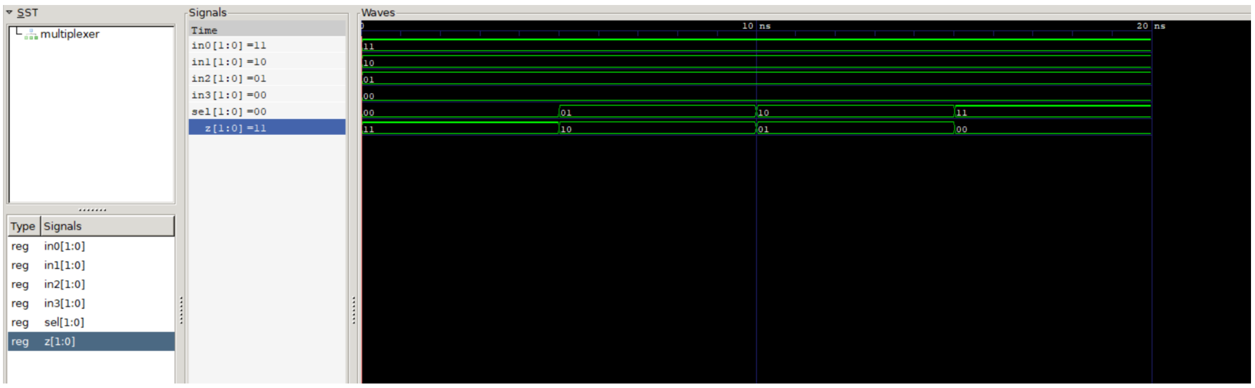
- James David - ~10 total hours spent
- Ethan Conway - ~8 total hours spent

Appendix II: VHDL Files

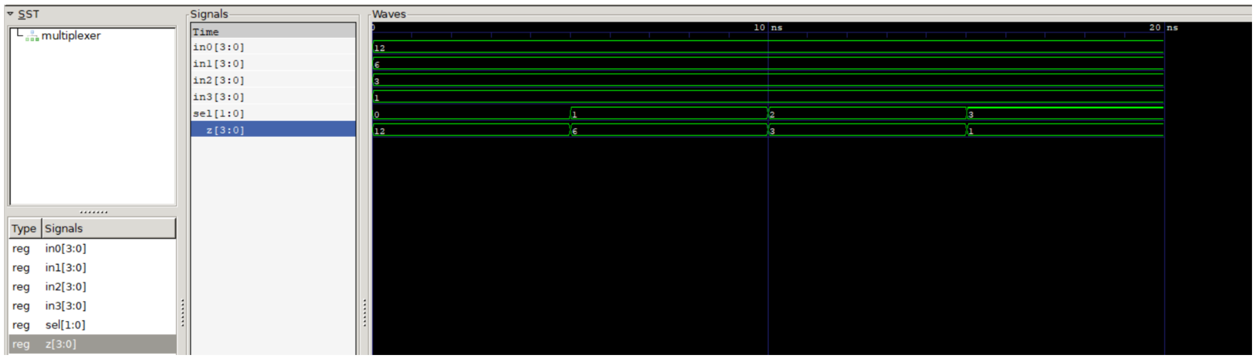
- Problem #1 files
    - gen_mux_tb.vhdl
    - gen_mux.vhdl
    - gen_mux.vcd
- Problem #2a files
    - reg4_tb.vhdl
    - reg4.vhdl
    - reg4.vcd
- Problem #2b files
    - reg4n8_tb.vhdl
    - reg4.vhdl
    - mux_8to1.vhdl
    - reg4n8.vcd
    - reg4n8.vhdl
    - demux_1to8.vhdl
- Problem #3 files
    - full_adder.vhdl
    - add_sub4.vhdl
    - add_sub4_tb.vhdl
    - add_sub4.vcd

Appendix III: Testing

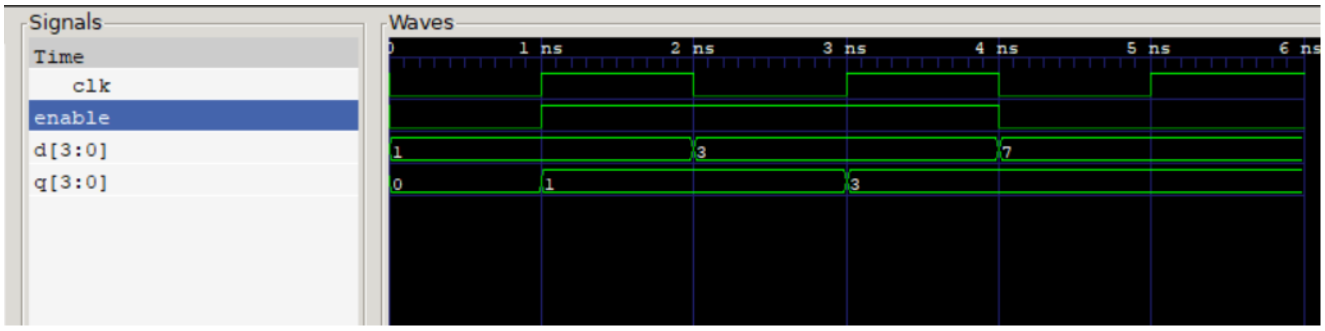
- Problem #1 Waveform images
    - 4-to-1 2-bit input Mux



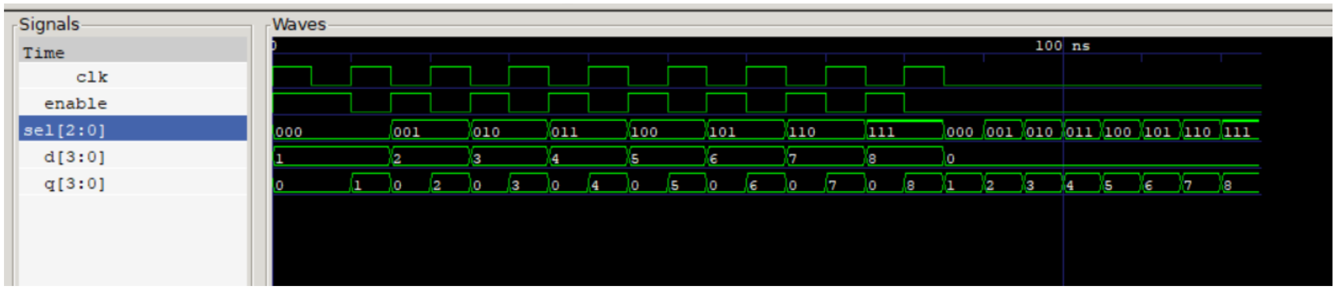
    - 4-to-1 4-bit input Mux



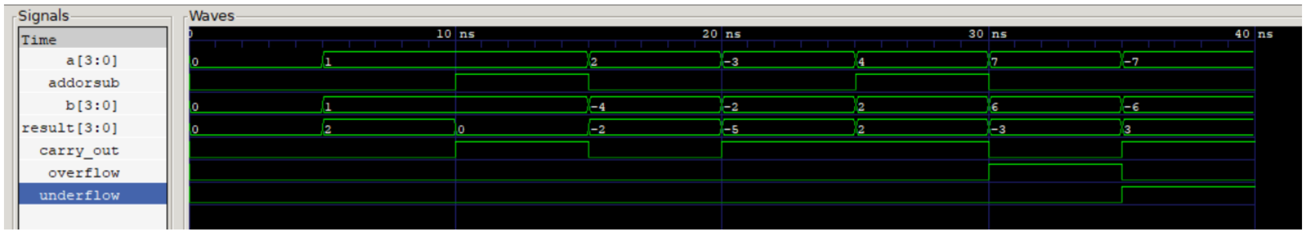
- Problem #2a Waveform:



- Problem #2b Waveform:



- Problem #3 Waveform


Signals
Waves
Time
a[3:0]
addorsub
b[3:0]
result[3:0]
carry_out
overflow
underflow
0
10 ns
20 ns
30 ns
40 ns
0 1 2 -3 4 7 -7
0 1 -4 -2 2 6 -6
0 2 0 -2 -5 2 -3 3