***Lab 3 – Implementing a 4-Bit ALU***

## **Review Item Comments Points (max)**

1) Prelabs from each student complete 5(5)

and thoughtful

2) Introduction effectively presents the 5(5)

objectives and purpose of the lab. Methodology gives enough details to allow for replication of procedure.

3) Discussion opens with an effective 5(5)

statement on the goals of the lab, backs up the statement concerning appropriate findings, provides a sufficient and logical explanation for the statement, addresses other issues pertinent to the lab.

4) Results open with an effective statement of 5(5) overall findings, presents visuals clearly

and accurately, presents findings clearly and with sufficient support. You MUST

include screenshots of the test bench results for each part of the lab.

Conclusion convincingly describes what has been learned in the lab.

5) Other: 10(10)

References are included.

Tables and figures are formatted. Grammar and spelling are correct

Comment Blocks for ALL Verilog modules are filled

in with students names and module description/purpose Report is written clearly and to the point.

Overall, the team...

• has successfully demonstrated what the lab was designed to teach

• demonstrates clear and thoughtful scientific inquiry

• has accurately measured and analyzed data for lab findings

Total: 30 (30)

Pre-lab

Implement the truth table for a hexadecimal (0-Fh) to 7-segment display decoder.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Char** | **Inputs** | | | | **Outputs** | | | | | | |
|  | **D3** | **D2** | **D1** | **D0** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| A | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| B | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| C | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| D | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| E | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| F | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |

|  |
| --- |
| EE260 Lab 3  **Implementing a 4-bit Four Function ALU** |

**Submitted by:**

**Kristel & Lianne**

Grade:30



|  |  |
| --- | --- |
| Team Members: | Kristel Zuniga  kzuniga@hawaii.edu  Lianne Rozzelle  liannero@hawaii.edu |
|  |  |
| Date Experiment Performed: | February 6 & 13, 2018 |
| Date of Submission: | February 18, 2018 |
|  |  |
| EE260 Lab Section | 2 |

|  |
| --- |
| **Introduction** |

ALU or Arithmetic Logic Unit is an integrated circuit that performs logical arithmetic operations as well as bitwise operations. In this lab we implemented AND and OR gates, as well as the NOR gate used in the 2’s complement lab to carry out tasks such as addition and subtraction. To present the results of our circuit, we also created a 7-segment display decoder to visually show the output of our operations on an LED display. The ALU was programmed to assume that the inputs were unsigned and that all 4-bits were used to convey the magnitude. However, as a bonus, we edited our Verilog code to indicate whether overflow occurred for unsigned ADD or signed SUBTRACT.

|  |
| --- |
| **Methodology** |

**hex2seg.v:**

Design Entry:

* Create an RTL project in Vivado Design Suite, declare that we will be using the Xilinx Artix 7 FPGA that is on the Basys3 board (XC7A35T-1CPG236C), and call it hex2seg.v.
* Add design sources to specify one input (4-bit D) and two outputs (7-bit segs and 4-bit anodes) and assign the segs to light up the specific cathodes listed in the pre-lab truth table by using nested conditional statements and assign the most right anode to light up on the Basys3 board.
* Run synthesis to make sure there are no errors or warnings.
* Add constraints by creating an .xdc file and specifying FPGA pins that will be used for the SW inputs (D[3:0] to SW3 to SW0) and 7-segment decoder output (segs[6:0] to CA through CG) and LED anode output (anodes[3:0] to AN3 to AN0), respectively.

Generate a test-bench waveform for functional verification:

* Run the simulation for the various combinations and verify the outputs.

Synthesis & Simulation for Synthesis and Timing Verification:

* Run simulation/Run Post-implementation Timing Simulation and verify propagation delay.

Implementation and Downloading:

* Generate downloadable bitstream file, program the FPGA with the bitstream file and test by toggling switches on the Basys3 circuit board, and verify with the 7-segment decoder on the right-most anode on the LED display.

**Repeat synthesis & simulation/implementation and downloading, but implement for the following operations: add4hex.v, and4bits.v, or4bits.v. Combine all previous modules for use in ALU4bit.v.**

**add4hex.v**:

* Add design inputs (A, B), outputs (Sum, anodes) to be 4-bit buses, and output segs7 to be a 7-bit bus. Connect outputs OF\_S and C\_MSB by internal “wire.”
* Add files fulladder.v, fourAdd.v, twosComp4.v from the 2’s complement lab to sources.
* Add constraints for SW inputs (A[3:0] to SW3 to SW0, B[3:0] to SW7 to SW4), LED output (Sum[3:0] to LD3 to LD0), 7-segment decoder output (segs[6:0] to CA through CG), and LED anode output (anodes[3:0] to AN3 to AN0), respectively.

**and4bits.v**:

* Add design inputs (A, B) and output (andAB) to be 4-bit buses.
* Add assign statement: andAB = A & B.

**or4bits.v**:

* Add design inputs (A, B) and output (orAB) to be 4-bit buses.
* Add assign statement: orAB = A | B.

**ALU4bit.v:**

* Add design 4-bit inputs (A, B), 2-bit input (op) and 4-bit outputs (Sum, anodes), 1-bit output (negB) and 7-bit output (segs7).
* Instantiate twosComp4, add4hex, and4bits, or4bits, and hex2seg modules to implement the four operation ALU.
* Add constraints for slides switches (op[1:0] to buttons L and C), SW inputs (A[3:0] to SW3 to SW0, B[3:0] to SW7 to SW4), LED output (Sum[3:0] to LD3 to LD0, negB to LD7), 7-segment decoder output (segs[6:0] to CA through CG), and LED anode output (anodes[3:0] to AN3 to AN0), respectively.

**Bonus:**

* Edit ALU4bit.v to show when overflow occurs for unsigned ADD and signed SUBTRACT.
* Add to the constraints file to light up LD15 when overflow occurs.

|  |
| --- |
| **Results** |

Overall, we obtained the expected results from each section of the lab. In the Hex to 7-segment decoder section (hex2seg), it took a 4-bit binary number and converted it to the hexadecimal equivalent, which was displayed on one of the anodes. Figure 1 and 2 shows our test-bench with all the options for the 4-bit binary (0000-1111). In Figure 3, you can see about a 10ns propagation delay for the conversion of the 4-bit to the 7-segment decoder, which is expected because of the nested conditional (if-else) statements.

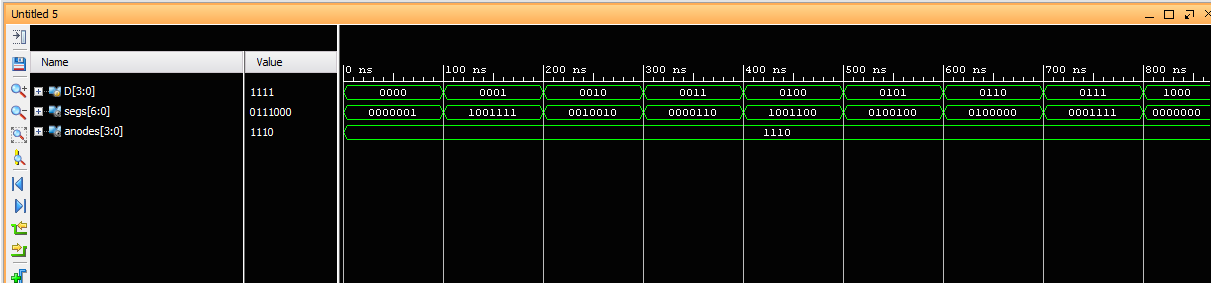


Figure 1. tb\_hex2seg(1)

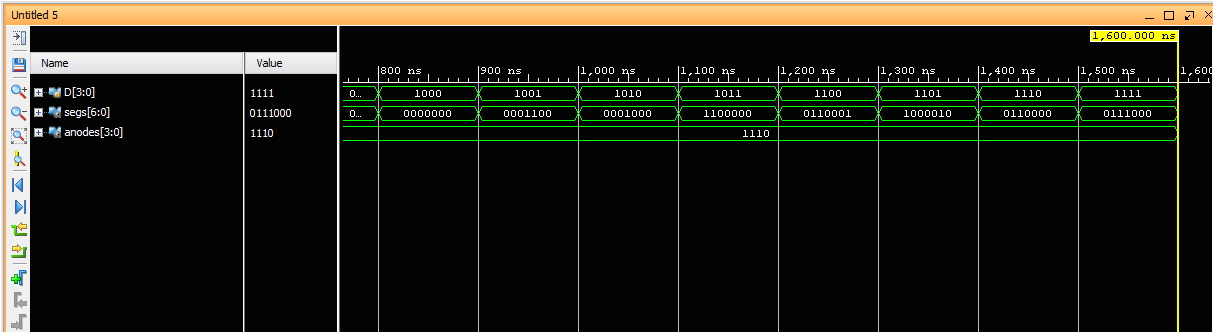


Figure 2. tb\_hex2seg(2)

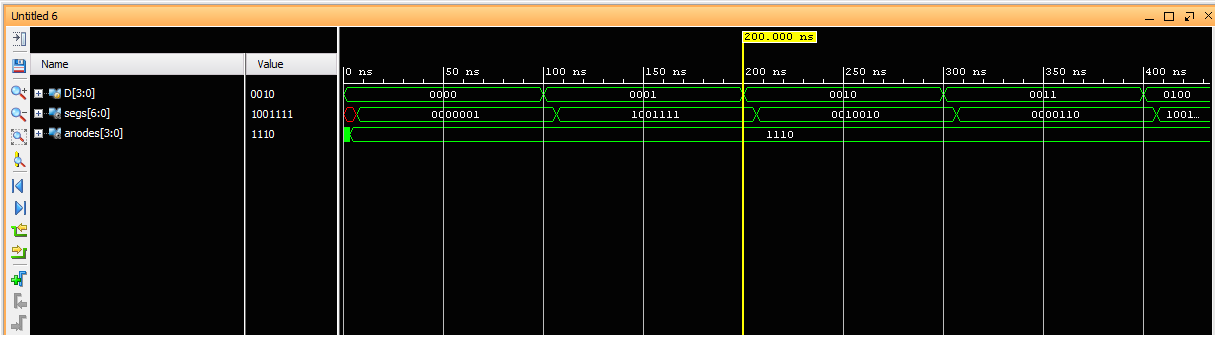


Figure 3. timing\_hex2seg

In the 4-bit adder to hex display section (add4hex), we used the fullAdder and fourAdd modules from Lab 2 to calculate the sum, which was displayed on the 7-segment display using the hex2seg decoder module. Figure 4 shows our test-bench with various inputs for A and B and the resulting sum. In Figure 5, you can see that the propagation delay for the sum was slightly longer than the delay for the 7-segment decoder, which is to be expected because of the boolean logic required to calculate the result.

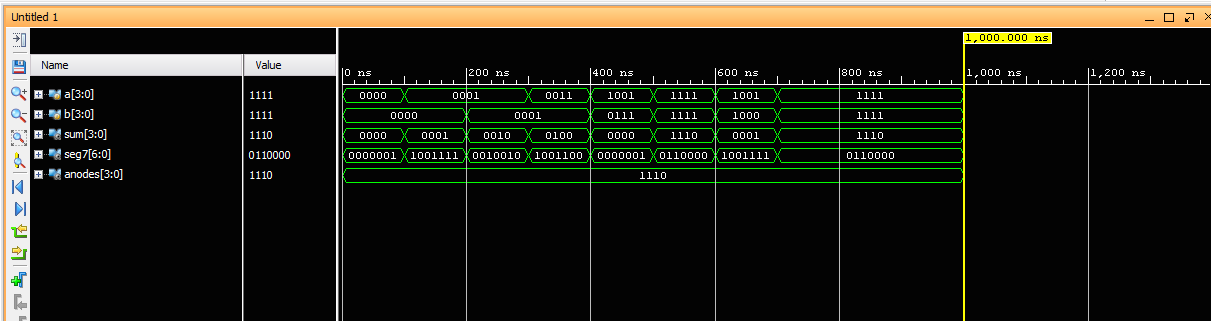


Figure 4. tb\_add4hex

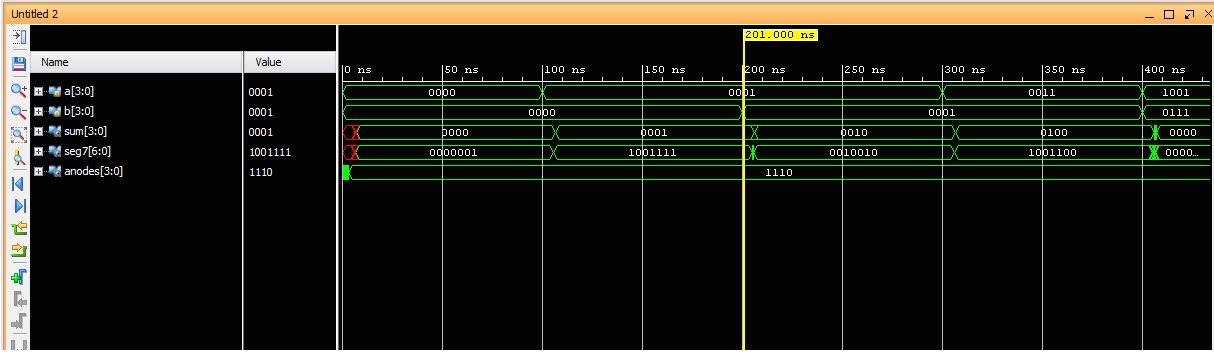


Figure 5. Timing\_add4hex

Figure 6 is an extra test-bench we performed to make sure that when we took the 2’s complement of B, the resulting sum would be correct (Sum = A + -B). Figure 7 is similar to the timing delay for the add4hex module where the sum took longer to calculate than the conversion of the 4-bit binary to 7-segment decoder.

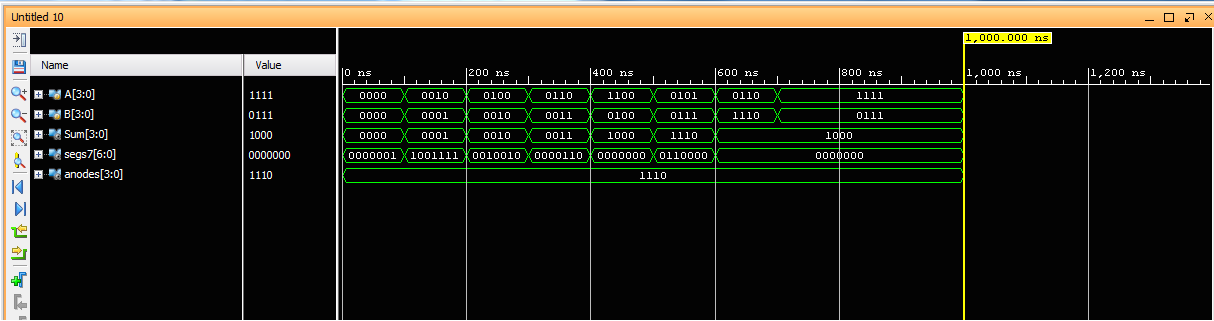


Figure 6. tb\_sub4hex

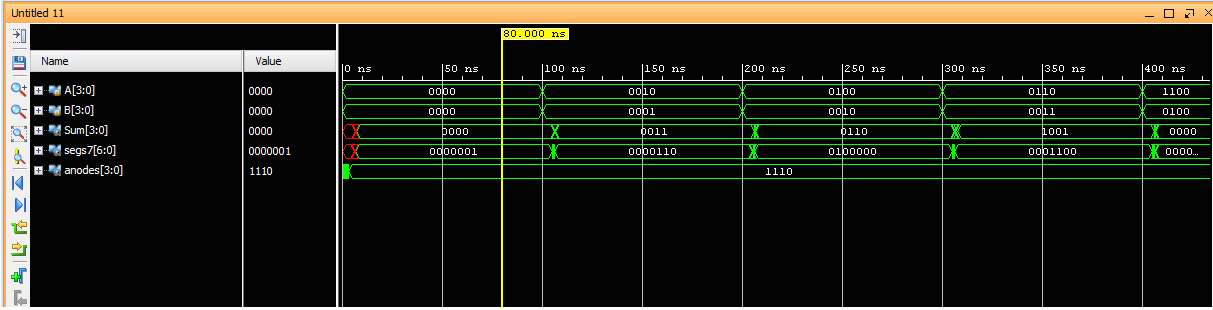


Figure 7. timing\_sub4hex

|  |
| --- |
| **Discussion** |

The 7-segment display on the Basys3 board runs on an “active low” signal, which means that the cathode for the 7-segment display will illuminate when the input is “0.” By completing the truth table in the pre-lab, we were able to quickly implement the display in the hex2seg module. The add4hex module was just a combination of the Lab 2 modules and then using the hex2seg module to display the output. The bitwise AND and OR modules were composed of one line assign statements, which were easy to implement individually.

The propagation delays we recorded for the hex2seg module were expected because of the nested conditional statements. In class, we learned that there is an order priority when using nested if-else statements. Each statement condition is checked before moving on to the next, which increases run time. Just as we found in Lab 2, the 4-bit adder doesn’t react instantaneously and the dataflow and propagation is affected by the distance it takes to get information from the input to the final output.

By using a bottom-up design approach, we were able to make sure that each element was working properly before putting it all together in the ALU and didn’t run into any issues when running the synthesis and simulations.

|  |
| --- |
| **Conclusion** |

In this lab, we gained more knowledge on how to use Verilog HDL to implement a 4-bit ALU and how to access more ports on the Basys3 board by adding more constraints for use in choosing operations (sliding switches) and displaying output (7-segment display). By combining the fourAdd, fullAdder, and twosComp4 modules that were created in Lab 2 with the hex2seg, add4hex, and4bits, or4bits modules from this lab, we were able to implement a 4-bit ALU that could perform addition, subtraction, bitwise AND, and bitwise OR operations.

|  |
| --- |
| **References** |

* Ee260\_2018\_spring\_materials\_laboratory\_03\_description.docx
* \*.v and \*.xdc files on GitHub: [ee260-2018-spring](https://github.com/ee260-2018-spring)/[ee260-lab-3-moretacocat](https://github.com/ee260-2018-spring/ee260-lab-3-moretacocat)