SANTA CLARA UNIVERSITY	ELEN 21L Fall 2022

Laboratory #6: Mini-calculator with a small 4-bit Arithmetic Logic Unit (ALU)

For lab sections Monday-Friday October 31- November 4, 2022

I. OBJECTIVES

· Design and test a 4-bit arithmetic logic unit (ALU) with adder/subtractor functions.
· Use a MUX hierarchy in the design of the ALU.
· Use your ALU in a mini-calculator with one memory location.

PROBLEM STATEMENT

The mini-calculator will use a small ALU to perform arithmetic operations on two 4-bit values which are set using switches. The ALU operations described below are implemented with an Adder/Subtractor component. A pushbutton input allows the current arithmetic result to be saved. An upgraded mini-calculator allows the saved value to be used in place of B as one of the operands.

- The small ALU that you will design will use the 4-bit adder myadder4 to do several possible operations. The circuit will take two 4-bit input values from switches, **a3 a2 a1 a0** and **b3 b2 b1 b0**, to be the operands.
- Four switches, **p3, p2, p1**, and **p0** will specify the arithmetic operation. These four bits can be interpreted as an **operation code**. The operations are specified in the table below. (Note that p3 = 0 for all table entries below. A new feature for **p3**=1 will be added in the lab.)
- The output will be a 4-bit value **r3 r2 r1 r0** and three single bit outputs: a carry out, an arithmetic overflow, and a zero result bit.

Op-Select Input	Output R	Notes
p3 p2 p1 p0	**r3 r2 r1 r0**	**Operation**
0 0 0 0	R = A + B	Add
0 0 0 1	R = A + B + 1	Add and Increment
0 0 1 0	R = A – B - 1	Subtract and Decrement
0 0 1 1	R = A – B	Subtract
0 1 0 0	R = A	Transfer A
0 1 0 1	R = A + 1	Increment A
0 1 1 0	R = A – 1	Decrement A
0 1 1 1	R = A	Transfer A (duplicate – note that this can be done in 2 ways)

II. PRE-LAB

1. Be sure you understand 2's complement representation of signed numbers. See Appendix A. The 4-bit adder can be used for adding or subtracting since (-B) represented in 2's complement form can be added to A to get A-B = A+(-B).

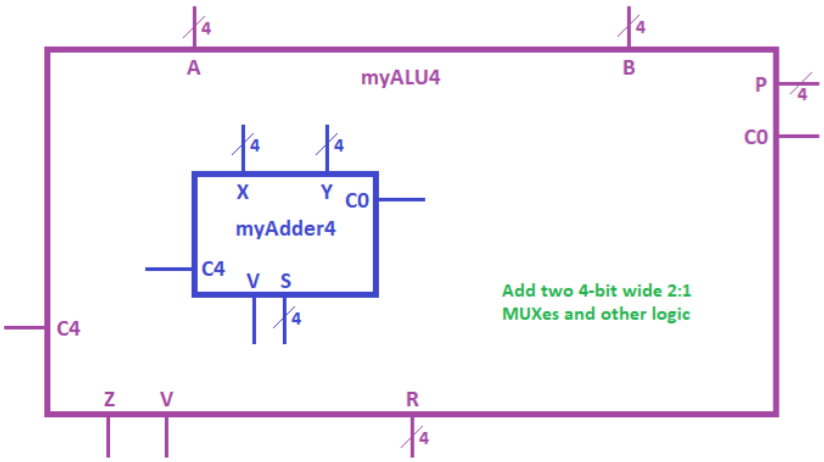
   The bit operations for addition are the **same for both** unsigned addition and signed addition using 2's complement representation. This is why 2's complement representation is so widely used. The bit pattern result is the same for both, but **the interpretation of the value of the bits will be different for signed and unsigned numbers**. (See Appendix A).

   Fill in the table below for your adder **Y** inputs and carry-in input for each of the four operations listed. The operations are specified by **p1** and **p0** when **p3** = **p2** = 0. The first entry has already been completed for you.

Op-Select Input	Adder 4-bit X inputs	Adder 4-bit Y inputs	Adder Carry-in	Operation	Notes
p3 p2 p1 p0	**a3 a2 a1 a0**	**b3 b2 b1 b0**			
0 0 0 0	a3 a2 a1 a0	b3 b2 b1 b0	0	R=A+B	Add
0 0 0 1	a3 a2 a1 a0			R=A+B+1	Add and Increment
0 0 1 0	a3 a2 a1 a0			R=A-B-1	Sub. and Decrement
0 0 1 1	a3 a2 a1 a0			R=A-B	Subtract

   The *myadder4* component from the previous lab (Lab 5) will be used to perform the ALU's addition. The ALU **A** input will be connected to the adder **X** input, and the adder **S** output will be the ALU **R** output. The adder Y input and carry-in to the adder will depend on which operation is selected, as defined by the table above.

2. Use a 4-bit adder symbol, a 4-bit wide 2:1 MUX, and other logic components as needed to design this first part of the arithmetic unit. Based on your table, use a 4-bit wide 2:1 MUX to produce the **Y** adder input with **p1** as the select. Add logic for the adder carry in input. Draw the schematic for your design.



3. Modify your design to include the next four operations shown in the problem statement table when **p2**=1. A second 4-bit wide 2:1 MUX with **p2** as the select can be used to create the inputs to the MUX used in the previous step. Draw a schematic for your design.

4. Write a hierarchical Verilog module *myALU4* which uses *myadder4*. The ALU inputs will be P, A, B, and C0. The outputs will be R, C4, V, and Z. It will perform the eight operations described in the problem statement.
    a. Add logic to make the carry input to the adder.
    b. Add logic to make the Z output which will be 1 only when all four bits of R are zero. This is minterm 0 with respect to the 4 bits of R.
    c. Add a case statement to create the Y adder input values. This is effectively using a 4:1 MUX. Note that the case statement must be inside an always block. What is the controlling expression for the case statement?

5. Consider how you would efficiently and strategically start your test of your ALU circuit after you implement it. Assume that the 4-bit adder has been tested and is known to function correctly. You will want to test the data path to be sure that the four-bit values of the input switches, the MUXes, the adder inputs and outputs, and the seven segment displays are all consistently ordered and connected. You also want to check that the control logic is working correctly. What does **p0** control? What does **p1** control? What does **p2** control? Note that **p3** is not connected to anything yet. It will be added during the lab.

6. Write the first four steps of your test plan. Indicate what values you would set for the inputs and what you would expect to see at the outputs of each step. Indicate what you would have verified if you see the expected output. There are many possible testing strategies.

Submit the logic expressions, the schematic circuit diagrams and Verilog source code for your prelab and include your test plan and answers to questions.

III. LAB PROCEDURE

Part 1 – Design and test the small ALU

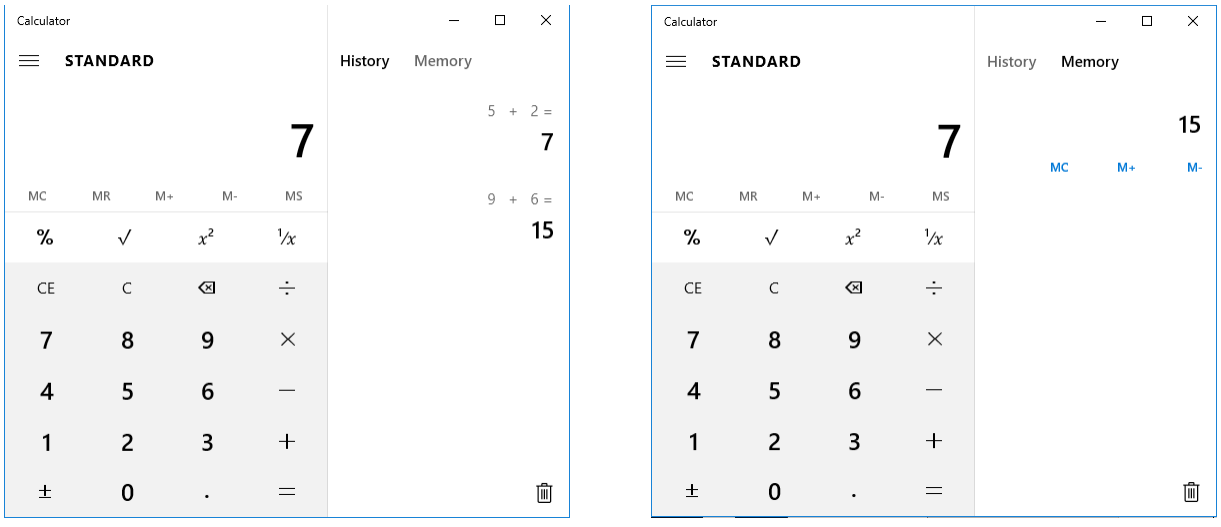
1. Compile the Verilog for *myALU4*. For the ALU to operate correctly, the newly designed control of the adder Y input must be tested. Simulate the operation of the multiplexer function on the least significant bit only. The simulator output will be Y[0] and the inputs will be A[0], B[0], P[2], and P[1]. The adder is presumed to be working because it was previously tested.

2. Check that the arithmetic operation is correct using the test plan you created for your pre-lab. Try various inputs and ensure that all operations are being executed correctly in the ALU.

3. Verify and demonstrate that the circuit functions correctly for all eight operations. Remember that some operations may result in overflow, and in that case the 4-bit result will not show the correct result that would need 5-bit representation.

4. When your test plan has been successful, demonstrate the operations to the TA. The TA may also ask you to demonstrate some input combinations that were not part of your test plan.

Part 2 – Add a memory save to the mini-calculator



Windows 10 Calculator showing History and Memory

5. From the Quartus II library components, add a 4-bit wide 2:1 MUX component.
   a. Go to Symbol Tool > mega-functions > gates and select "busmux"
   b. Once the symbol is on the schematic and selected, edit the properties for the symbol
   c. and set the "WIDTH" parameter to 4.

6. From the Quartus II library components, add one *8dff* component to your schematic to make one memory location to save the results of an arithmetic operation. This part will store 8 bits, but we will use only four bits. Go to libraries → other → maxplus2 and select *8dff*.
    a. Connect a signal called **Save** to the CLK input of the 8dff.
    b. Connect the CLRN and PRN inputs to a logic 1 level, Vcc. The clear and preset inputs will not be used in your circuit.
    c. Connect the adder output **s3 s2 s1 s0** to the *8dff* inputs D1 to D4.
    d. Make a bus for 8dff outputs Q1 to Q4 named **m3 m2 m1 m0.**
7. Disconnect the input to your ALU circuit that came directly from your **B** inputs, and connect it to the MUX output.

8. Use input **p3** to control whether the **B** inputs or the memory **M** outputs will be the **B** inputs to your 8 operation ALU by connecting **p3** to the MUX select.

9. If **p3** = 0, your ALU should operate as it did in part 1. If **p3** = 1, the operations will use A and M rather than A and B.

Testing

10. Verify that the ALU works as it did in Part 1 when **p3** is 0.

11. Test the **Save** signal, which should cause your circuit to save the adder output. You should see that the output of the register, **M,** should not change during your testing. It should only change when you cause **Save** to assert. (Make sure to de-assert **Save** after a short interval, so that you can use it to capture another result.)

12. Test the operation of your mini-calculator. You should be able to do the operations from Part 1, save a result, and then use that saved result in another operation.

13. Show a sequence of operations using memory that will make your ALU output count by 2's. As you save each result, you should see the sequence 2, 4, 6 on the output of the register. What is the next output and what other indications do we expect?

14. If we save one more result, what register output do we see?

IV. REPORT

1. Include: Introduction, procedure, results, conclusions and references.

2. Include schematic, Verilog code, test plan, and results for each operation.

3. Describe how negative results appear in your simulation waveforms.

4. List three pairs of A, B inputs that will cause the Z output to be 1 when the operation is P = 0 0 0 0 and C0 is 0.

5. List three pairs of A, B inputs that will cause the Z output to be 1 when the operation is P = 0 0 1 1 and C0 is 0.

7. If the B input values are zero, describe a sequence of operations using memory that you could use to get a mini-calculator output of –A.

8. How would you connect two 4-bit ALUs to make an 8-bit ALU?

Appendix A
Table of 4-bit Unsigned Numbers and 4-bit Two's Complement Signed Numbers

4-bit Binary Number	4-bit Un-signed Value	4-bit 2's Complement Signed Value
0000	0	0
0001	1	+1
0010	2	+2
0011	3	+3
0100	4	+4
0101	5	+5
0110	6	+6
0111	7	+7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	11	-4
1101	13	-3
1110	14	-2
1111	15	-1