OBJECTIVES

- Understand the use of multiplexers as data selectors.
- Design an arithmetic logic unit (ALU) circuit by combining SSI and MSI components.
- Augment the ALU with registers (a sequential datapath) and additional MUX components (for steering logic) to create an RALU.

INTRODUCTION

The objective of this lab is to design a simple arithmetic logic unit (ALU) and then to expand and augment the simple ALU. The augmentation of the ALU with registers (creating an RALU) and the addition of a few multiplexers will result in the creation of a system that can be augmented with a controller to form a simple central processing unit (CPU). In section 4 of the RALU part of this lab, you will act as the controller for the RALU. In Lab 6, a controller circuit will replace you in the creation of a very simple 4-bit CPU. The 4-bit CPU in Lab 6 will be expanded into an 8-bit CPU, called the Gator CPU (G-CPU or GCPU) in Lab 7.

LAB STRUCTURE

In this lab, you will design the core functionality of a simple CPU by creating two ALUs. In § 1, you will design a purely combinational ALU with multiple functions. In § 2, you will add registers to this ALU to create the core functionality of the GCPU.

REQUIRED MATERIALS

- Your entire lab kit (including your DAD)
- UF's DAD *Waveforms Tutorial*
- *DE10-Lite Pins*
- Useful Quartus Components:
  - In “others | maxplus2” library
    - 74153: Two 4-input MUX’s (recommended)
    - 74151: 8-input MUX (recommended)
    - 74283: 4-bit Adder (recommended)
    - 7474: Dual D-FF (not recommended)
    - 74175: Quad D-FF (not recommended)
  - In “primitives | storage” library
    - dff (recommended)
  - In “primitives | other” library
    - vcc, gnd
- Document on website: *Explanation_of_Table 4*

SUPPLEMENTAL MATERIALS

- *PLD on Breadboard Programming WARNING!*
- *DE10-Lite Schematic*
- *DE10-Lite Manual*

PRE-LAB PROCEDURE

1. ARITHMETIC LOGIC UNIT (ALU)

In this part of the lab, you will design a 4-bit ALU (**Lab4_ALU**). The ALU has two 4-bit inputs (A_{3-0} and B_{3-0}), two function-select inputs (S_{1-0}), and a carry input (C_{in}). The ALU has a 4-bit function output (F_{3-0}) and a carry output (C_{out}) as described Table 1, shown in Figure 1 and Figure 2.

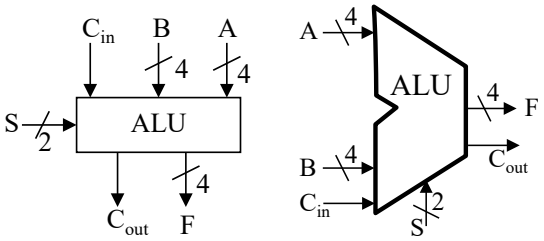

Figure 1: ALU functional block diagrams (with all signals active-high).

S1:S0	Action	Equation
00	complement of A	F = /A
01	bit-wise OR	F= A or B
10	Sum (w/ Cin), Cout	F = A + B + Cin , Cout
11	bit-wise AND	F = A and B

Table 1. ALU functions.

You must design **independent** circuits for each of these actions. The function-select inputs, S_{1-0}, determine which F_{3-0} is used for the ALU output (see Table 1 and Figure 2). The carry output (C_{out}) should be set **only** when an addition causes a carry output **and** the SUM function is selected**.** In all other operations, the carry output should be false.

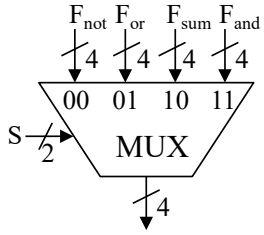

Figure 2: Function selection of ALU.

It is often helpful to create extra output signals so that your circuit design is easier to debug (in simulation and in hardware). I suggest adding outputs for each of the functional blocks (F_{not}, F_{or}, F_{sum}, and F_{and}). Using a functional compilation and simulation will also execute much faster in Quartus. A functional compilation and simulation is independent of the chip chosen. I usually start all my designs (when possible) by functional compiling my circuits before I even attempt to fit the design onto a particular chip (full compilation). After you have thoroughly tested your circuit through simulation, you can remove the unneeded outputs if it is necessary to make the design fit in PLD (which for us is the DE10-Lite's PLD). Note that when you perform a functional compilation and functional simulation, there is no limit to the amount of output signals you can have; but once you do a full simulation, you are limited by the size of the PLD that is being programmed. Functional compilation is also much faster than a full compilation (and the same is true for a function simulation over a timing simulation). You will **NOT** demo the design in this section with hardware, but you will simulate it.

1. Design the ALU in Quartus (**Lab4_ALU**). There is no need to design an adder; just use the 74'283 described in the Materials section of this document. Similarly, you can use MUXes provided by Quartus in your design.
2. Simulate the circuit identifying that it works by **annotating** your simulation. Please do **NOT** make a simulation that includes **ALL** possible input combinations. Only include enough test cases to show that every operation of your ALU works as expected.
3. Each of the above items, including circuit schematic and annotated Quartus simulation results should be part of the Canvas-submitted lab document. (Of course, you should also submit your archive file.)
4. You do **NOT** need to build this circuit.

2. REGISTERED ALU

A block diagram of the RALU you will design is shown in Figure 3. By adding a controller (in Lab 6), we will turn the RALU into a simple CPU.

The device has one 4-bit wide INPUT bus and one 4-bit wide OUTPUT bus. These buses are used to bring data to and from the ALU. The OUTPUT bus is fed back for possible re-entry to the system.

REG A and REG B are 4-bit wide registers (i.e., four D Flip-Flops) that are used to hold data originating from MUX A and MUX B. MUX A and B (each containing four 4-input multiplexers) are used to connect a particular bus to REG A and REG B, respectively. The buses are connected to REG A and REG B as described in Table 2.

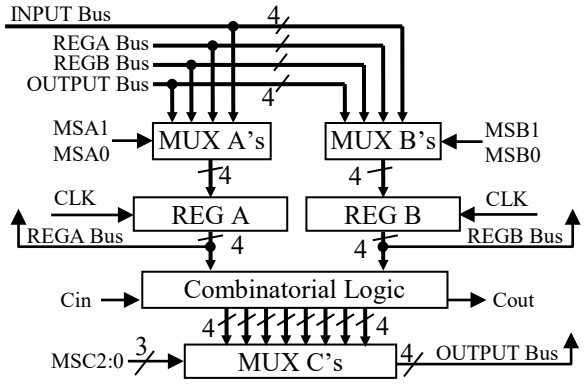

Figure 3: A simple RALU.

Table 2: Input source MUXs for Registers A and B.

MSA/ MSB1	MSA/ MSB0	Bus Selected as Input to REGA/REGB
0	0	INPUT Bus
0	1	REGA Bus
1	0	REGB Bus
1	1	OUTPUT Bus

Table 3: ALU function selection MUX (for MUX C).

MSC2:0	Action
000	REGA Bus to OUTPUT Bus
001	REGB Bus to OUTPUT Bus
010	complement of REGA Bus to OUTPUT Bus
011	bit wise AND REGA/REGB Bus to OUTPUT Bus
100	bit wise OR REGA/REGB Bus to OUTPUT Bus
101	sum of REGA Bus & REGB Bus to OUTPUT Bus
110	shift REGA Bus left one bit to OUTPUT Bus
111	shift REGA Bus right one bit to OUTPUT Bus **without** sign extension

The outputs of REGA and REGB are thus fed back to MUX A and MUX B inputs as well as to a combinational logic block. The combinational logic block is used to perform the following operations on the data in REGA and REGB: complement, ANDing, ORing, addition, and shifts (both left and right). The signals **REGA** and **REGB** are **system outputs.**

MUX C consists of four 8-input multiplexers. They are used to select a particular operation to put onto the output bus. The three select lines MSC2:0 function as shown in Table 3 (where the most significant bit is on the left).

The carry output, Cout, should come directly from the adder circuit (with no additional circuit necessary).

1. Draw a complete and detailed **functional block diagram** (expanding on the block diagram given above), labeling all inputs and outputs and internal signals. All signals are active-high, so in **this case**, activation can be left out.

2. Design the required circuit (using project name **`Lab4_RALU`**) in Quartus. The signal names should match those of your functional block diagram. In addition to the output bus, the outputs of registers A and B (REGA and REGB) should be outputs of your design.

   As in the previous section, it is often helpful to create extra output signals so that your circuit design is easier to debug (in simulation and in hardware). For example, you could add the signals that come out of MUX A and MUX B as outputs as well as each of the eight functions that come out of the *Combinational Logic* block of Figure 3. After you have thoroughly tested your circuit through simulation, you can remove the unneeded outputs if it is necessary to make the design fit in your DE10-Lite's PLD.

3. Simulate each of the simple functions (in Table 3) implemented directly with MUX C using the Quartus simulator. Include each of these simulations in your lab document. (As usual, all pre-lab material must be submitted through Canvas prior to the start of your lab. Be sure to also submit your archive files.)

4. A **control word** (in this case the word is 7-bits) can now be defined as the bit pattern:

   MSA1:0, MSB1:0, MSC2:0

   By setting the appropriate control word bit pattern, it is now possible to load data into the system and perform all the ALU functions mentioned previously.

   Derive and test control words (i.e., write **and simulate** programs) to do each of the following complex functions. Create (and turn in) a table (see Table 4 and *Explanation of Table 4*) with columns labeled MSA, MSB, MSC, etc. and showing the sequences of steps necessary to implement each of these complex functions. Describe what is occurring for each line in the column labeled Description. Include these tables in your lab document. You must include the simulation outputs for each these complex functions in your lab report. As always, add appropriate annotations. At a minimum, describe in your annotation what is happening at each active clock edge.

   a. Load A and B with known values, bit wise AND the registers and then store the result in A. Preserve the contents of register B during this operation.
   b. Load A and B with known values, bit wise OR the registers and then store the result in B. Preserve the contents of register A during this operation.
   c. Load A with a known value, complement A and then store the result in B. Preserve the contents of register A during this operation.
   d. Load A and B with known values, sum them and then store the result in A. Preserve B during this operation.
   e. Load A with a value, shift it right one bit and then store it in B.
   f. Load A with a value, shift it left one bit and then store it in B.
   g. Now write "a program" (i.e., the sequence of steps necessary) to ADD 3 and $C, AND $5 to the result, multiply the result by 4, OR $A to the result, complement this new result, and then finally divide it by 2. (Don't worry if the result makes no sense; just perform the required operations. This "program" should work independently of the input values, i.e., the "program" should remain unchanged even if the 3 and $C inputs are changed to $E and 7 and the 5 and $A were changed to $B and 4. This is not the case for the divide and multiply values.)

5. Download your design to your DE10-Lite with **nothing else connected to the board** (except power and ground).

6. Use your **DAD** for inputs (including CLK) and the HEX displays on the DE10-Lite for outputs (i.e., the three buses RegA, RegB, and Output). Use the DP (decimal point) of the Output bus HEX display for Cout. Use the static I/O of the DAD for the inputs.

7. Once you are convinced that your RALU works correctly, **you must make a DAD custom pattern** (see the bottom of page 4 in the *Waveforms Tutorial*) with a single **control word**, along with the Input bus, Cin, and CLK, for each line of Table 4. You **MUST** use a custom pattern at a slow enough frequency (e.g., 0.5 Hz) to see all the numbers on the 7-segment displays on the DE10-Lite. Note that when the static I/O is open and running, if the signals are set for buttons or switches, the static I/O will override the pattern generator. (If the DAD had more digital I/O pins, we could use it for both the inputs and outputs.) Verify that your circuit design functions as specified in the Pre-Lab Requirements, specifically part **g**.
   a. Save the Waveforms workspace (which will include the control word custom pattern file) to **`Lab4_4g.dwf3work`** and submit it to Canvas.
   b. You will demo your lab to your PI using this custom pattern file.

8. You must adhere to the Lab Rules & Policies document for every lab. Re-read, if necessary. Documents must be submitted through Canvas for every lab. All pre-lab material is to be submitted **BEFORE** the beginning of your lab.

PRE-LAB PROCEDURE SUMMARY

1. Design a simple ALU using SSI and MSI combinational logic § 1.
2. Design a more complex RALU that includes sequential components in § 2.
3. Write control word sequences to create programs for your RALU in § 2.

IN-LAB PROCEDURE

1. Complete the lab quiz utilizing the RALU you created in § 2.
2. Demonstrate the correct function of program 4g from § 2.

PRE-LAB QUESTIONS (RALU)

1) Draw the single simple device that can be added to your circuit design to "remember" the last carry output. Specify the inputs and outputs for this device.
2) Will a divide by two work for all 4-bit 2's complement numbers? Explain.
3) Describe how you can take the 2's complement of a number, i.e., if A is loaded with a number, get the 2's complement of A into B.
4) Describe how you subtract with your RALU. Hint: See the previous question.
5) Suppose you're not allowed to use a flip-flop that has an asynchronous CLR or SET, how can you add a function that clears the contents of either A or B?

Table 4: Sample table for Pre-lab Requirement 4.

MSA	MSB	MSC	Input	Cin	RegA	RegB	Output	RegA+	RegB+	Output+	Cout+	Description

Note that the + means “after the clock edge”