LAB 4: ALU and CPU

OBJECTIVES

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 several multiplexers will result in the creation of a simple central processing unit (CPU). The CPU will be a building block used in a future lab.

MATERIALS

- Your entire lab kit (including your DAD/NAD)
- 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

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, two function-select inputs, and a carry input. The ALU has a 4-bit function output and a carry output as described Table 1, shown in Figure 1 and Figure 2.

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

Table 1. ALU functions.

Independent circuits must be designed for each of these actions. The function-select inputs (S) determine which F is used for the ALU output (see Table 1 and Figure 2). **The carry output should be set only when an addition causes a carry output.** In all other operations, the carry output should be false.

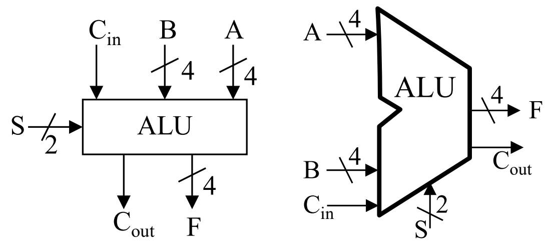

Figure 1: ALU functional block diagrams (with all signals active-high).

DEBUG OUTPUTS

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_{and}, F_{or}, and F_{sum}). 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 PLD. But you will <u>**NOT**</u> program this design into your PLD.

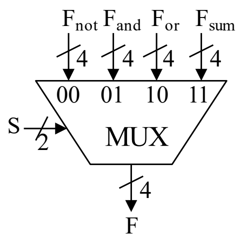

Figure 2: Function selection of ALU.

PRE-LAB REQUIREMENTS (ALU)

1. Design the ALU in Quartus (**`Lab4_ALU`**). Simulate the circuit identifying that it works by **annotating** your simulation. (Please do **NOT** make a simulation that includes **ALL** possible input combinations.) 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 MUX's in your design.
2. 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.)
3. You do <u>**NOT**</u> need to build this circuit.

PRE-LAB QUESTIONS (ALU)

1. If you were to make a complete (un-abbreviated) truth table for your ALU, how many rows would it need? (The answer should tell you why I did not want a simulation including ALL input combinations.)
2. What changes would be necessary to the design already made if you wanted to add two more functions, F=A NOR B and F=A XOR B, where XOR is exclusive-or?

CPU WITH RALU

A block diagram of the CPU you will design is shown in Figure 3.

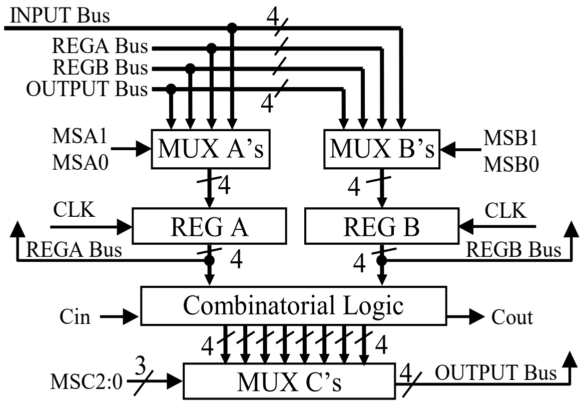

Figure 3: A simple CPU.

1. 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.

2. 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 busses are connected to REG A and REG B as described in Table 2.

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

MSA1/ MSB1	MSA0/ MSB0	Bus Selected as Input to REGA/REGB
0	0	INPUT Bus
0	1	REGA Bus
1	0	REGB Bus
1	1	OUTPUT Bus

The outputs of REGA and REGB are thus fed back to MUX A and MUX B inputs as well as to a combinatorial logic block. The signals REGA and REGB are system outputs.

The combinatorial logic block is used to perform data complement, addition, ORing, ANDing, and shifts.

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

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

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

COMPILATION AND SIMULATION

In this lab, your signals REGA and REGB are required outputs of the circuit, since these are the actual system outputs. It is often helpful to create extra output signals (as discussed below, called **debug outputs**) so that your circuit design is easier to debug (in simulation and in hardware). For example, you might want to output the signals that come out of MUX A and MUX B. With these extra outputs, your design might not fit in your PLD. In this case, you have two choices, the best of which is to use a functional compilation and simulation (as described in the Quartus tutorial). 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). Note that with your MAX 10 part, Quartus does **NOT** have a non-functional compilation and simulation (called a full compilation and timing simulation) available! An alternative is to choose a bigger device (like the 10M50SCE144I7G device in the same MAX 10 family as our 10M02SCU169C8G PLD).

After you have thoroughly tested your circuit through functional or full simulation, you can remove the unneeded outputs, if necessary, to make the design fit into your PLD.

Functional Compilation and Simulation

When designing something this big, it is a good idea to keep a lot of data available for simulation. Then, when you know that your design simulates correctly, you can remove as many outputs as needed to get the design to fit into your particular device. Extra signals can be output and used in your simulation. I call these extra outputs **debug outputs.**

A **functional** compilation and simulation has (effectively) no limit on the number of inputs, outputs and internal elements available. For a functional compilation, just select the arrow next to "Analysis & Synthesis" in the "Compile Design" task. Note that with your MAX 10 part, Quartus does **NOT** have a non-functional compilation and simulation (called a full compilation and timing simulation) available!

See the section titled "E. Functional and Timing Simulation" in your *Quartus Tutorial* for more information on how to obtain a functional simulation in Quartus.

CONTROL WORD

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 in the Objectives.

PRE-LAB REQUIREMENTS (CPU WITH RALU)

1) Draw a complete and detailed functional block diagram (expanding on the one given above), labeling all inputs and outputs and internal signals.
2) Design the required circuit (all call it `Lab4_CPU`) using Quartus' schematic entry (.bdf file). The signal names should match those of your functional block diagram. In addition to the output bus, the outputs of registers A and B should be outputs of your design.
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) 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) with columns labeled CLK, 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 save the simulation outputs for these complex functions and include them in your Quartus archive file. Save each simulation in a separate file. Annotate each simulation. At a minimum, describe in your annotation what is happening at each active clock edge.
   a) Load register A with data, complement the A data and then store it into B. Preserve the data in A during this operation.
   b) 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.
   c) 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.
   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 left one bit and then store it in B.
   f) Now write a program to ADD 3 and 7, OR the result with 4, divide the result by 4, complement this new result, AND the result with 3, and then finally multiply it by 2. (Don't worry if the result makes no sense; just perform the required operations. This "program" should work, independent of the inputs, i.e., the "program" should remain unchanged even if the 3 and 7 inputs are changed to $B and 6.)
5) Use the debounced switch (made in the previous lab) for your clock input. If you removed this design from your breadboard, re-build it in a corner of your breadboard and leave it there for the remaining labs this semester.
6) Download your design to your PLD PCB with **nothing else connected to the board** (except power and ground).
7) Use your **DAD** for inputs and you DIP LEDs (and/or 7-segment displays) for outputs (i.e., Cout and the three busses RegA, RegB, and Output). For testing at home, you can use the static I/O of the DAD for the inputs. I suggest that you start with DAD/NAD Static I/O but use a debounced circuit for the clock input so that you can control when the signals change. But once you are convinced that it works okay, then **you must make** a DAD **custom pattern** (see the bottom of page 4 in the *Waveforms Tutorial*) with a single **control word** for each line of Table 4 to replace your debounced circuit clock and DAD/NAD Static I/O. You **MUST** use a custom pattern at a slow enough frequency (e.g., 0.5 Hz) to see all the number on the LEDs on the breadboard (and/or 7-segment displays on the PLD PCB). 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/NAD 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 f.
   a) Save and submit the control word custom pattern file (**`Lab4_4f.dwf3work`**) to Canvas.
   b) You will demo your lab to your TA using this custom patter file.
8) You must adhere to the Lab Rules and Policies document for every lab. Re-read, if necessary. Documents must be submitted through Canvas and on paper for every lab. All pre-lab material is to be submitted **BEFORE** the beginning of your lab.

IN-LAB REQUIREMENTS (CPU with RALU)

Demonstrate that part 4f works properly on CPU design on your breadboard. Use the DAD/NAD pattern generator file **`Lab4_4f.dwf3work`**.

PRE-LAB QUESTIONS (CPU WITH 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.

LAB 4: ALU and CPU

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 CPU. Hint: See above 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?

LAB 4: ALU and CPU

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”