EITF35 - Introduction to Structured VLSI Design (Fall 2017)

Assignment 3 - Arithmetic Logic Unit (ALU)

Introduction

In this lab assignment, a simple arithmetic logic unit (ALU) will be designed, synthesized, and downloaded to the FPGA board. The ALU takes in user inputs from on-board push buttons and switches, performs selected operations such as addition and subtraction, and shows final results on a 7-segment display. The lab assignment includes design of control-path such as finite state machine(s), data-path such as combinational circuits, and other modules like a 7-segment driver. In order to pass the lab, both preparations and fulfillment of all lab tasks are mandatory.

Lab Preparation

- Go through the manual and try to understand the required functionality and given tasks. Make sure that you have understood what is expected from the lab before you start coding. Consult lab assistants if any functionality or task is unclear to you.
- Tasks 1$\sim$3 should be done before the preparation deadline. However, you should prepare as much as possible before the lab session.
- Go through the files provided in the lab directory and try to figure out how much coding you need to develop to fulfill the tasks.
- Draw ASMD graphs of the different system components.

Deadlines

You have to finish all tasks before **Oct. 27**, in order to get approved on this lab.

- Lab preparation
- Presentation of the completed tasks during the lab

Assignments

Every input of the ALU is an 8-bit digit which can be set by an arbitrary selection of switches (SW0∼SW7) on the FPGA board. The position of every switch represents a binary '0' or '1'. Therefore, the range of each of the ALU inputs can be interpreted as an unsigned integer between 0 and 255, or a signed integer between -128 and 127. To simplify the design, we use two's complement number representation in this lab. The required functionalities of the ALU are:

- After "Reset" (push button CPU RESET), the value shown on the 7-segment display is the unsigned representation of an 8-bit binary value set by on-board switches.
- When the user has set the first data operand (A), a push button BTNC can be used as an "Enter" key to send the value to ALU and buffer it in an internal register.
- While the first ALU operand is saved, the second operand (B) can be set by re-arranging the positions of the switches. The second operand should be saved in a separate register when the "Enter" button is pressed the second time. During the input of operand B, the 7-segment display should show current unsigned value of the on-board switches.
- As soon as the ALU has both its inputs, it can perform the three required operations. The first operation "A+B" should be shown as soon as the "Enter" button is pressed the second time. By pressing the button for another time "A-B" should be performed as the second operation. By pressing the button once more, a modulo operation of "A mod 3" should be displayed. Note that operand B is not used during the modulo operation.
- The ALU output should cyle between "A+B"-> "A-B"-> "A mod 3" -> "A+B"-> ... every time the "Enter" button is pressed. Note that input switches (SW0∼SW7) should be "locked" once the ALU has received both input operands, i.e., toggling the input switches should not affect the result. Because both operands are stored in the internal data registers.
- The push button BTNL is used as a "Sign" key to toggle between unsigned and signed arithmetics for addition and subtraction in ALU. Correspondingly, the displayed ALU results should be updated every time the "Sign" button is pressed. Default ALU operation should be unsigned arithmetics. Use the left most 7-segment module to show "-" when the result is negative or "F" when an overflow has occurred.
- The output of the ALU must be continuously shown on the 7-segment display, unless the "Reset" button (CPU RESET) is pressed, which clears current ALU inputs and starts all over again.

Looking at the required functionalities, a possible implementation could involve a controller, i.e., a finite state machine (FSM), a data-path containing an ALU, a register block to keep the ALU input operands, a binary to binary-coded-decimal converter (Bi/BCD), and a driver for the 7-segment display. An example of a possible implementation is shown in Figure 1.

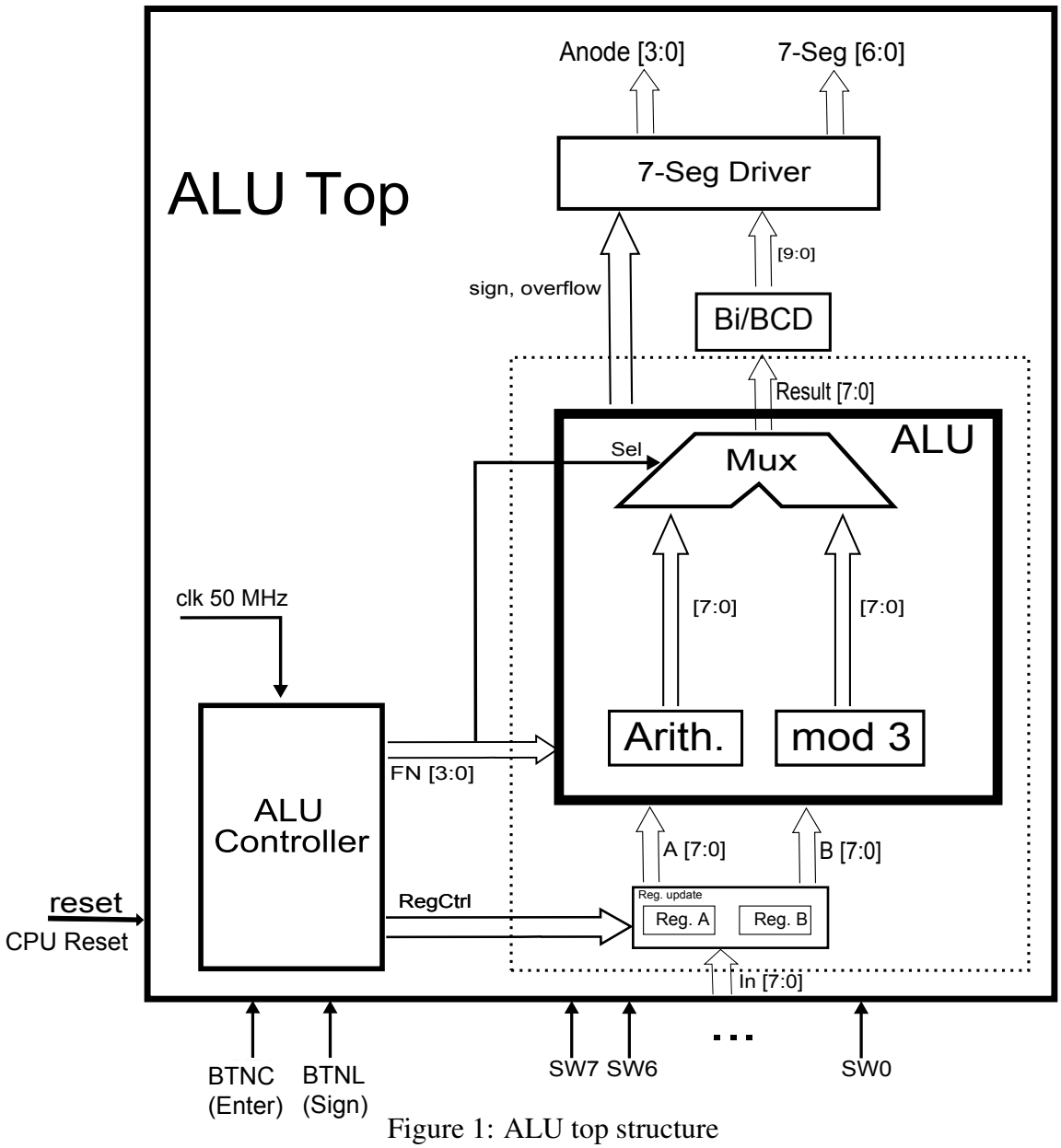

Figure 1: ALU top structure

Task 1 Based on input signals "Reset" (CPU RESET), "Sign" (BTNL), and "Enter" (BTNC), propose a FSM of the ALU controller that provides control signals to the ALU to achieve the required functionalities. Draw a reasonably detailed finite state machine (FSM) on a piece of paper for your solution. Determine what control signals should be applied to the ALU in each state. The FSM can be either of type Moore or Mealy.

Task 2 Read the paper under the Lab 3 links (i.e. Modulo3.pdf) and focus on Fig. 1 to understand the proposed architecture of unsigned modulo 3 operation. Then, modify this architecture to be able to calculate both **signed** and **unsigned** modulo 3. Try to minimize your additional logics to the architecture in Fig. 1.

Task 3 Come up with a hardware-friendly algorithm to convert a binary value to BCD format.

1 ALU Block

The ALU block is a simple combinational logic circuit that performs the following functions according to the commands from the controller.

FN[3:0]	Operation
0000	Input A
0001	Input B
0010	Unsigned (A + B)
0011	Unsigned (A - B)
0100	Unsigned (A) mod 3
1010	Signed (A + B)
1011	Signed (A - B)
1100	Signed (A) mod 3

Table 1: ALU block - functions and the corresponding input control signals

The ALU performs unsigned & signed addition, subtraction, and modulo operation on a pair of 8-bit binary inputs. Besides the 8-bit output, ALU uses signals “sign” and “overflow” to indicate the sign and arithmetic status during computation.

The modulo operation is performed as “x mod n = x - n·floor(x/n)”, where n is fixed to 3 in this lab. For example, 7 mod 3 = 7 - 3·floor(7/3) = 7 - 3·floor(2.33) = 7 - 3·2 = 7 - 6 = 1.The result of "mode 3" should be positive and don't implement it only using add/sub.

Task 4 Write VHDL code for the ALU block to perform the required functions as given in Table 1. For this purpose, you can use the file “ALU.vhd” in the lab directory and complete it with your code.

When you have written the functional VHDL code for ALU block you need to make sure that your ALU works as expected. In order to verify the correct functionality of your ALU entity a testbench “tb_ALU” is provided in the lab directory.

Task 5 Add “ALU.vhd” and “tb_ALU.vhd” to your project in Vivado/Questasim. Compile the files and start a simulation. In the wave window, right click on a, b and result signals and choose *Radix -> Decimal*. Verify your ALU results.

2 ALU Controller

According to the required functionalities, a controller, i.e. a finite state machine is necessary. It manages the behaviour of the ALU with respect to user commands input from push buttons. As an example, the ALU behaviour, according to the “Enter” signal sequence is shown in Figure 2. In the figure, “1” states that the button is pressed and “0” indicates when it is released.

Task 6 Design the ALU controller in HDL. Don’t integrate it into your design unless you have fully verified its functionalities.

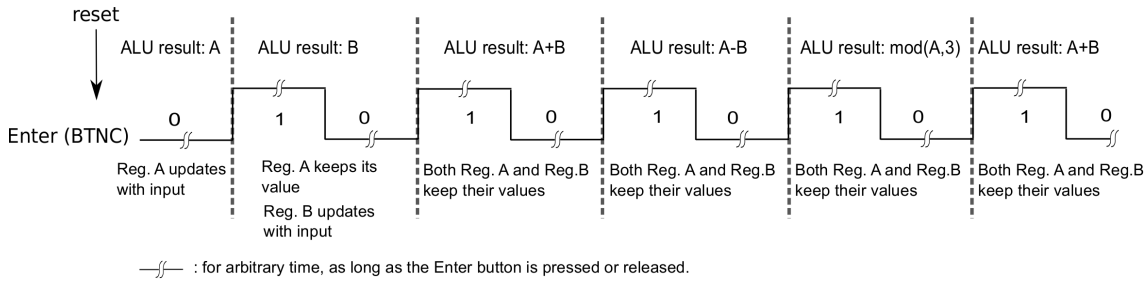

Figure 2: Functionality of ALU based on the input commands

3 Register Update

Finally, when the whole design is downloaded into the FPGA, the input for operands A and B are provided with 8 switches on the board (SW0 - SW7). Thus, only one register can be updated at any time by the binary value set with the switches. You can use the controller you have designed and tested previously to control when each register should be updated. A possible RTL structure describing the needed sequential and combinatorial processes is shown in Figure 3.

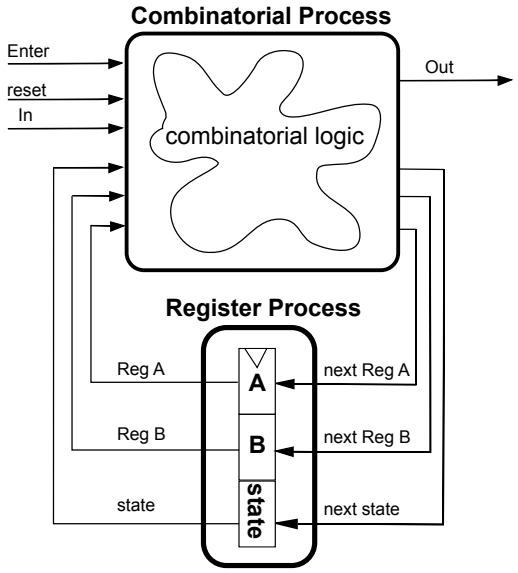

Figure 3: The RTL structure describing the processes needed

Task 7 Design an entity to provide suitable inputs for the register update in this lab based on the expected functionality described in Figure 2.

4 Binary to BCD Converter

BCD or binary-coded-decimal is a method of using four binary digits to represent the decimal digits 0 to 9. For instance, “249” with binary representation “11111001” can be represented by ”10 0100 1001” in BCD format. Converting a binary value to BCD is very common, in order to represent each digit separately on a 7-segment display.

Task 8 Write a binary to BCD entity VHDL code and a proper testbench to verify its functionality. Compile the converter and simulate it using Vivado/QuestaSim.

5 Synthesizing the whole ALU structure

Now that you have written and tested different parts of the whole ALU structure using Vivado/QuestaSim, you are ready to implement a prototype circuit. This is often done using an FPGA technology. To accomplish this, we use Xilinx Nexys-4 FPGA boards available in the lab. To finalize your design before synthesizing and downloading to the FPGA, you need to fulfill two remaining tasks:

1 You need to integrate all the codes you have developed so far together with the 7-segment driver you have designed in lab 2 into a project.

2 Mapping the ports of the top structure of your design to the physical ports on FPGA.

Task 9 Add all the required VHDL files you have developed and verified so far in a project in Vivado. To connect the separate components of your design, add a new VHDL source file into the project and develop the VHDL code for the ALU top structure in case you have not done it so far. Use the interface model shown in Figure 4 for the top structure.

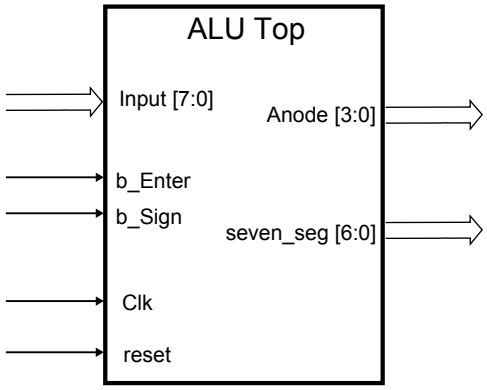

Figure 4: ALU_top block

Task 10 Use the configuration definition for the physical implementation that was used for Lab 2 and add the additional connections required for this lab using by clicking on the Xilinx design constraints file.

Task 11 Synthesize and download the whole design in FPGA using Xilinx Vivado tool. Verify that your design works as expected. If the 7-segment display starts blinking choose a shorter multiplexing time between the segments in your code.

6 Some Questions

- What are the advantages of using 2's complement numbers over using a dedicated bit for the sign?

- Why is it necessary use a de-bouncer for the push buttons?

- What is the advantage of performing per module verification before all modules are assembled together?