

**Lab5**  
**Design and Implementation of an ALU**

**Purpose**

The purpose of this lab is:

1. To design an 8-bit ALU
2. To learn to use the Core Generator with pre-defined IP (Intellectual Property) blocks.
3. To implement the ALU on a FPGA
4. To experimentally check the operation of the ALU both in simulation and on the FPGA

This lab is different from the other assignments in the sense that it is a *design* project which gives you more freedom to come up with your own solutions. The following write-up serves as a guideline to help you design the lab. Because your choice of design may vary dramatically from others', make sure you explain your design clearly in the lab report.

You will have two weeks to perform this lab. In the first week, it is recommended that you have finished all the individual functions in the ALU, and in the second put them together into one coherent ALU and add the components for testing it on the board.

**Lab Background**

**a) Problem Statement**

An Arithmetic and Logic Unit (ALU) is a combinational circuit that performs logic and arithmetic micro-operations on a pair of n-bit operands (in our case, A[7:0] and B[7:0]). You can assume that the inputs A and B are signed (if they are negative, they are in two's complement) when they are presented to the input of the ALU. The operations performed by an ALU are controlled by a set of *operation-select* inputs. In this lab you will design an 8-bit ALU with 4 operation-select inputs, S0, S1, S2 and S3. Logical operations take place on the bits that comprise a value, while arithmetic operations treat inputs and outputs as 2's complement integers. Errors must be detected by the ALU, specifically division by zero; if any occur, raise the ERR output, which should otherwise be zero. If an addition occurs that overflows or a multiplication resulting in a value that can not be shortened to 8 bits, raise the Overflow output. The functions performed by the ALU are specified in Table 1 and 2.

Table 1: Logical Functions				
S3	S2	S1	S0	O[7:0]
0	0	0	0	A
0	0	0	1	A AND B
0	0	1	0	A OR B
0	0	1	1	A XOR B
0	1	0	0	NOT A
0	1	0	1	NOT (A AND B)
0	1	1	0	NOT (A OR B)
0	1	1	1	NOT (A XOR B)

Table 2: Arithmetic Functions				
S3	S2	S1	S0	O[7:0]
1	0	0	0	A+1
1	0	0	1	A-1
1	0	1	0	A+B
1	0	1	1	A-B
1	1	0	0	A/B
1	1	0	1	A%B (remainder)
1	1	1	0	A*B (only low 8 bits used)
1	1	1	1	A>>B (logical shift)

The logical operations are bitwise, meaning that for S="0001", O(1) is equal to A(1) AND B(1). This of course means that they ignore what sign A and B have. For arithmetic operations, the ALU should truncate at 8 bits.

We may ignore the carry-out in additions and subtractions. Overflow is defined as when we create a value too great for 8 bits to hold, and so serves as a warning to anyone using our ALU. This can occur only in addition/subtraction and multiplication. ERR is defined only for division by zero.

The logical shift operates on the value by shifting its bits left or right, filling zeroes in after. For A = "11011011" and B = "11111111", the output should be "10110110", as A is shifted right by -1 bits (i.e. 1 bit to the left), and a zero is filled in on the right. The difference between a logical shift and an arithmetic shift is that the logical treats A as unsigned and B as signed. The choice of logical shift is to make it simpler, but if you design an arithmetic shifter, be sure to make note of it in your lab report and tell the TA to whom you demo.

## **b. Designing the Macros**

There are many strategies to perform this lab, and you may choose any combination of them. You may use schematics, VHDL, Xilinx library symbols such as adders or multiplexers. While it is possible to do the entire lab using only logic gates and VHDL, for at least multiplication it is recommended to use Xilinx's IP Core Generator, which will essentially create custom components for us to use in the design. There are also a variety of symbols that Xilinx provides by default which can be used, such as 'adsu8', an adder/subtractor. If you are unsure of how something works, you may wish to consult the Xilinx library guide (<http://toolbox.xilinx.com/docsan/xilinx82/books/docs/lib/lib.pdf>). Note that the multiplier design used in the previous lab will not handle negative numbers correctly, and so will need to be modified for use in this lab if you do not use a Xilinx multiplier, in addition to being made to handle a larger number of bits. We will supply you with code for a divider to integrate into the rest of your ALU as a macro. The dividers generated by Xilinx Core Generator cannot be simulated in the ISE environment, so you should not use them.

To use the IP core generated multiplier, add a new source to the project with the type “IP (Coregen & Architecture Wizard)”. Name it e.g. “Multiplier”. Then select the core “Multiplier v9.0” from the list under “Math Functions”. After you click Finish, the CORE Generator interface shows up for you to customize the block. Browse through the pages and choose appropriate bit lengths for the inputs. After you are done, the core will be generated and you can find it in the library. It will be a sequential circuit with two 8-bit inputs, one 15-bit output, and a clock input. There are data sheets available for each core, which can be accessed through the “View Data Sheet” link after the core is selected from the list.

The arithmetic IP cores are typically clocked and pipelined, meaning that they will only start doing something on a clock edge and that they can be designed to do only part of an operation per clock (such that you can feed it several calculations to do). Since the demo circuit we are building is not so interested in maximizing speed or minimizing power usage, we can simply connect these core's clock inputs to the clock used for the display. Then we can treat them as combinational circuits, since the clock is so fast that the outputs respond to inputs almost immediately.

For the shifter, one possible way to design it is called a ‘barrel shifter’, which consists of smaller shifters, shifting 1, 2, or 4 bits. The larger shift would consist of choosing which smaller shifters are to be active. This is not the only way for you to design it, only a hint.

## **c. Demonstrating the ALU**

In order to easily see the output of the ALU for the demo, you will display the results on 4 seven-segment displays and the LEDs.

1. Take as input two 4-bit 2's complement values from the board's switches, using the push buttons to select which operation to perform. The switches will not be able to

test all possible situations, but at least most of them. You will need to sign-extend the 4-bit input values to the full 8-bits.

2. Produce an 8-bit value from the ALU, on the right three 7-segment displays with the left blank for non-negative values and showing a '-' for negative. Use also the 8 LED's on the extension board to display  $O[7:0]$ , and the main board LEDs to display ERR and Overflow.

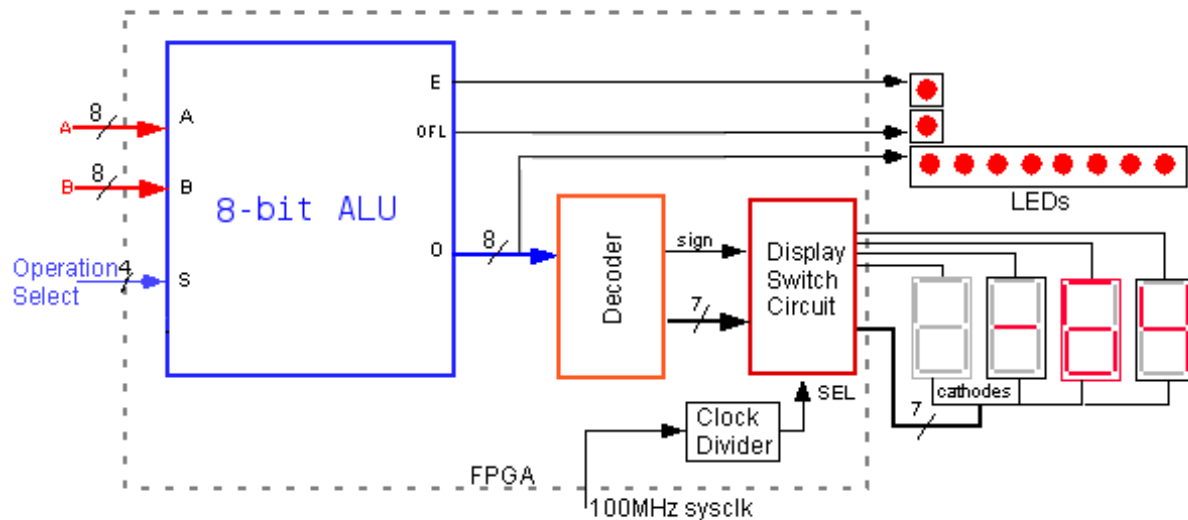


Figure 1: Overall system, including the ALU and display units.

### In-Lab Assignment:

Your task is to design and implement the ALU using the Xilinx ISE tools and the prototyping board. You will create a project with a top-level schematic that contains the 8-bit ALU. This top-level file will have several macros which you need to create using the Schematic editor, Xilinx CORE Generator, or VHDL. Macros can have macros embedded in it.

*As this project is more complicated than earlier projects, it will be important that you be very systematic during the design. Each macro should be simulated and errors corrected before you proceed. Failing to do so will make it difficult for you to debug the system. It is also important that you plan your work for each week.*

- Design the components that implement each of the logical and arithmetic functions of the ALU. Try to reuse the same components for multiple functions, if possible.
- Simulate each component and verify that they work correctly.  
(You are expected to complete at least the above tasks in the first week.)
- Combine the macros into an ALU to be placed in the top level schematic.
- Make the decoder and the display switch circuit, or reuse from a previous lab.

- Make the sign-extenders that convert the switch inputs to the two 8-bit value for A[7:0] and B[7:0].
- After simulation, implement the ALU and test it on the board.
- Give a demo to the lab instructor and convince him/her that your ALU works properly.

### **Pre-lab**

Before coming to the lab, prepare an approximately 1-page block diagram of your ALU's interior, so that you will be prepared to create it in lab. There are many design decisions to be made; give some serious thought to what you think will be easiest to design that will meet the requirements.

### **Hand-in (at the start of the next lab)**

You have to hand in a report that contains the following (See [guidelines](#) for reports):

1. Course title, Lab number, Lab title, Your names, and date
2. Section on the Pre-lab explaining the design of each block and giving the answers to each task.
3. Brief description of the experiment including the goals and theory
4. Circuit schematics (screenshots, include your names)
5. Waveform simulations (screenshots)
6. Discussion of the results indicating proper function.
7. Brief description of any honors points you have attempted and results.
8. Conclusion

Updated Oct 18, 2007