KOÇ UNIVERSITY

ELEC 204 Digital System Design
Laboratory Manual

Experiment #3
Design and Implementation of a 4-bit Arithmetic and Logic Unit

---

Department of Electrical & Electronics Engineering
College of Engineering
Koç University

1 Objectives

- To design a 4-bit Arithmetic and Logic Unit (ALU),
- To implement the ALU on FPGA using modular design,
- To experimentally check the operation of the ALU.

This laboratory session 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 with the assignment. However, if you find more efficient or more elegant ways to implement parts of the ALU, go ahead. Just make sure you justify your design and explain it clearly in the laboratory report.

The main objective in this experiment is to get familiar with the elementary logic gates and use of them for implementing hierarchical logic circuit design. You will design your circuit using Xilinx ISE software. Then you will simulate your circuit behavior and deploy your design to the Prometheus board.

2 Equipment & Software

- IBM compatible PC with Windows operating system,
- Xilinx ISE v14.7 & Prometheus software packages,
- Prometheus FPGA board,
- USB cable for programming.

3 Procedure

1. Read the **Background Information** section.
2. Do your preliminary work before coming to lab and upload it to the blackboard web page.
3. Having your designs from the preliminary work, follow the digital design procedure from the Experiment 1 with the following steps.

    - **Design Entry:** Enter gates and blocks to build your design using hardware description language in to the software tool.
    - **Design Simulation:** Test your design in computer simulation. Revise your design if necessary by turning back to the design task.
    - **Synthesize and Map Design:** After a successful simulation, map inputs and outputs of the design to I/O pins of the hardware.
    - **Program Hardware:** Upload the final synthesized bit file to the hardware using a USB cable.

- **Test Hardware:** Verify the hardware functionality for your design. Revise your design if necessary by turning back to the design task.

4 Background Information

4.1 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 (ex. A[3:0] and B[3:0]). The operations performed by an ALU are controlled by a set of function-select inputs. Block diagram of the ALU is shown in Figure 1. In this laboratory session you will design a 4-bit ALU with three function-select inputs: Mode M, Select S_1 and S_0 inputs. The mode input M selects between the Logic (M=0) and Arithmetic (M=1) operation. The functions performed by the ALU are specified in Table 1.

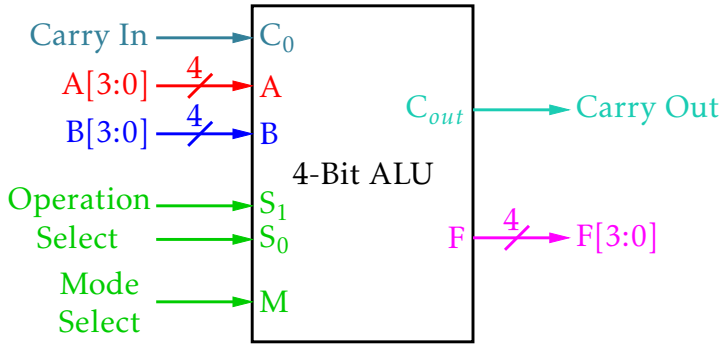

Figure 1: Block diagram of the 4-bit arithmetic and logic unit.

When dealing with arithmetic operations, you need to decide how to represent negative numbers. As is commonly done in digital systems, negative numbers are represented in two's complement. This has a number of advantages over the sign and magnitude representation such as easy addition or subtraction of mixed positive and negative numbers.

4.2 Design Strategies

When designing the ALU we will follow the principle "Divide and Conquer" in order to use a modular design that consists of smaller, more manageable blocks, some of which can be reused. Instead of designing the 4-bit ALU as one circuit you can first design a one-bit ALU, also called a bit-slice. These bit-slices can then be put together to make a 4-bit ALU.

There are different ways to design a bit-slice of the ALU. One method consists of writing the truth table for the one-bit ALU. This table has 6 inputs (M, S_1, S_0, C_i, A_i and B_i) and two outputs F_i and C_i^{out}. This can be done but may be tedious when it has to be done by hand.

An alternative way is to split the ALU into two modules: one logic and one arithmetic module. Designing each module separately will be easier than designing a bit-slice as one

	M	S_1	S_0	C_0	Function	Operation
Logic Unit	0	0	0	x	A_iB_i	AND
	0	0	1	x	$A_i + B_i$	OR
	0	1	0	x	$A_i \oplus B_i$	XOR
	0	1	1	x	$\overline{A_i \oplus B_i}$	XNOR
Arithmetic Unit	1	0	0	0	A	Transfer A
	1	0	0	1	A + 1	Increment A by 1
	1	0	1	0	A + B	Sum A and B
	1	0	1	1	A + B + 1	Increment the sum of A and B by 1
	1	1	0	0	$A + \overline{B}$	A plus one's complement of B
	1	1	0	1	A - B	Subtract B from A (i.e. $\overline{B}$ + A + 1)
	1	1	1	0	$\overline{A} + B$	B plus one's complement of A
	1	1	1	1	B - A	Subtract A from B (i.e. $\overline{A}$ + B + 1)

Table 1: Functions of the arithmetic and logic unit.

unit. A possible block diagram of the ALU is shown in Figure 2. It consists of three modules: 2-to-1 multiplexer, a logic unit and an arithmetic unit.

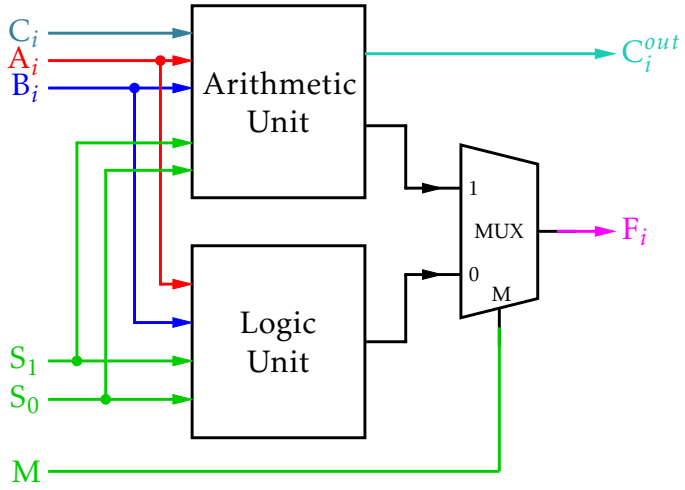

Figure 2: Block diagram of the bit-slice arithmetic and logic unit.

4.3 Displaying the Results

In order the easily see the output of the ALU you will display the results on the seven-segment display and the LEDs (LD).

The result of the logic and arithmetic operations will be displayed on the LEDs and seven-segment display respectively.

Since you are working with a 4-bit representation for 2's complement numbers, the maximum positive number is +7 and the most negative number is −8. Thus a single seven-segment display can be used to show the magnitude of the number. Use a LED display for the "−" sign.

A possible pin-out of the resultant signals is shown in Figure 3. In logic mode, use LEDs LED3:LED2:LED1:LED0 (pins P3, P6, P13, P16) to display the output F[3:0]. In arithmetic mode, use seven-segment display (SEVSEG0) to show the output F[3:0] and LED7 to show the carry out.

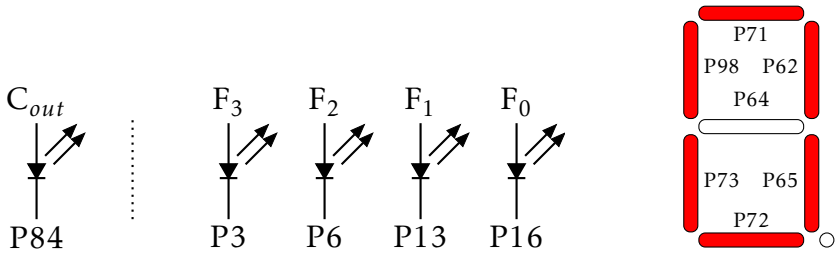

Figure 3: Pin-out of the LEDs.

5 Preliminary Work

Do the following tasks beforehand. Write the answers to all questions in your preliminary report prior to coming to the laboratory.

Question 5.1 *Design the multiplexer*. You should design the multiplexer with gates and write the design down in your preliminary report.

Question 5.2 *Design of the logic unit*. Here you have couple of choices to design this unit:

a) Write truth table, derive the K-map and give the minimum gate implementation

b) Use a 4-to-1 multiplexer and gates.

As part of the preliminary work, you can choose any of the two methods. Briefly justify why you chose a particular design method. Explain the design procedure and give the logic diagram. In case you use a multiplexer, you need also to give its schematic.

Question 5.3 *Design the arithmetic unit*. Again, here you have different choices to design and implement the arithmetic unit. A particularly attractive method is one that makes use of previously designed modules, such as your full-adder. The arithmetic unit performs basically additions on a set of inputs. By choosing the proper inputs, one can perform a range of operations. Also, design a circuit that detects over- or underflow.

Question 5.4 *Design the decoder for displaying results*. The decoder should be designed in such a way that when the logic mode (M=0) is selected, only the LEDs are active and when the arithmetic mode (M=1) is selected only the seven-segment displays are active.

Question 5.5 *Hardware implementation preparation*. Answer the following questions.
a) Either sketch or extract the RTL schematics of your program.
b) What clock frequency do you use for the 7 segment multiplexer.
c) Describe the input and output ports of your program. Print out your ucf file.

6 Experimental Work

6.1 Task 1: Bit-Slice

Your task is to design and implement the bit-slice (this could be a macro containing several sub macros) of the ALU using the Xilinx ISE software and the Prometheus FPGA board. Write a VHDL code to operate with one bit. Make sure the bit-slice is working properly and demonstrate it in the laboratory.

6.2 Task 2: 4-Bit ALU

Implement the 4-bit ALU using the bit-slice ALU. Combine consecutive bit-slices of ALU into a single circuit by connecting carry-out C_i^{out} of the previous bit-slice to the carry-in C_{i+1} of the next bit-slice. The sample VHDL codes given in Listings 1,2 demonstrates how a 2-input XOR circuit can be used as a component (module) to construct a 4-input XOR gate. You can use this example to construct your 4-bit ALU circuit.

Note that logic results should be displayed on the LEDs and arithmetic results should be displayed on the 7-segment display as defined in section 4.3. For this purpose, you can use the sample VHDL code under the resources section of the lab web-page.

Simulate the full 4-bit ALU and verify that all logic and arithmetic operations function properly.

7 Assessment

Provide the lab report as described in the *Lab Report Guidelines* document, which is available online on the blackboard webpage. Do not forget to include a clear description of your design approach and the overall schematic with simulation waveforms.

Listing 1: VHDL code for the 2-input XOR gate.

```
--.-- File: "TwoInXOR.vhd"
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity TwoInXOR is
port (X,Y : in STD_LOGIC;
        S : out STD_LOGIC);
end TwoInXOR;

architecture Behavioral of TwoInXOR is
begin
 S <= X xor Y;
end Behavioral;
```

Listing 2: VHDL code for the 4-input XOR gate.

```
--.-- File: "FourInXOR.vhd"
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FourInXOR is
   port (A,B,C,D : in STD_LOGIC; --.-- Inputs
           S : out STD_LOGIC; --.-- Outputs
end FourInXOR;

architecture Behavioral of FourInXOR is
--.-- Use two input XOR component from "TwoInXOR.vhd" file
component TwoInXOR
   port (X,Y : in STD_LOGIC;
           S : out STD_LOGIC);
end component;

--.-- Intermediate signal "Z"
signal Z: STD_LOGIC_VECTOR(2 downto 1);
begin
   --.--Create three copies of the two input XOR
   XOR0: TwoInXOR port map (A,B,Z(1));
   XOR1: TwoInXOR port map (C,D,Z(2));
   XOR2: TwoInXOR port map (Z(1),Z(2),S);
end Behavioral;
```