ESKİŞEHİR TECHNICAL UNIVERSITY

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

EEM 334 - Digital Systems II

LAB 4 – COMBINATIONAL LOGIC CIRCUIT – ALU DESIGN

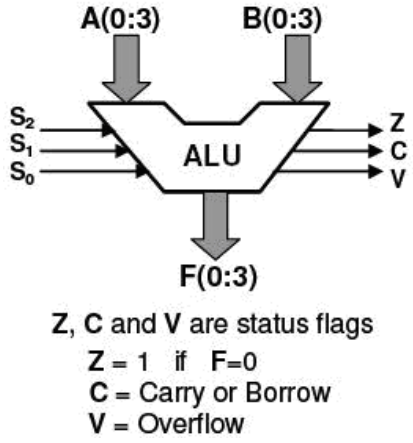
1. PURPOSE

After implementing 4-bit adder on Nexys4 board in previous laboratory, now you are responsible for making an ALU circuit whose block diagram and truth table are given below. Your ALU design must take two 4-bit operands and it must perform Addition, Subtraction, Increment, Decrement, AND, OR, XOR and NOT operations.

2. BACKGROUND

An Arithmetic and Logic Unit (ALU) is a combinational circuit that performs logical and arithmetic operations on a pair of n-bit operands (in our case, A[3:0] and B[3:0]). Unless otherwise stated, you can assume that the inputs A and B are signed, two's complement numbers 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 4-bit ALU with 3 operation-select inputs, S[2:0]. Logical operations take place on the bits that comprise a value (known as bitwise operations), while arithmetic operations treat inputs and outputs as two's complement integers.

The block diagram and the truth table for the ALU are shown below. You must write the VHDL code for this simple 4-bit ALU according to the following functionality.



S_2	S_1	S_0	Function (F)
0	0	0	A+B
0	0	1	A-B
0	1	0	A-1
0	1	1	A+1
1	0	0	A $\wedge$ B
1	0	1	A $\vee$ B
1	1	0	NOT A
1	1	1	A $\oplus$ B

Figure 1: 4-bit ALU block and truth table

3. PROCEDURE

In this lab, you will design 8 modules individually (one of them 4-bit full adder is already designed). Each operation must be designed on different VHDL modules and they can be used on "ALU_top" module which is written on following pages.

Your module names must be like as shown below:

1. four_bit_adder.VHD
2. four_bit_subtract.VHD
3. four_bit_and.VHD
4. four_bit_or.VHD
5. four_bit_xor.VHD
6. increment.VHD
7. decrement.VHD
8. NotA.VHD

In addition, you should complete "ALU_top" module, which is given in the next page to combine your eight components.

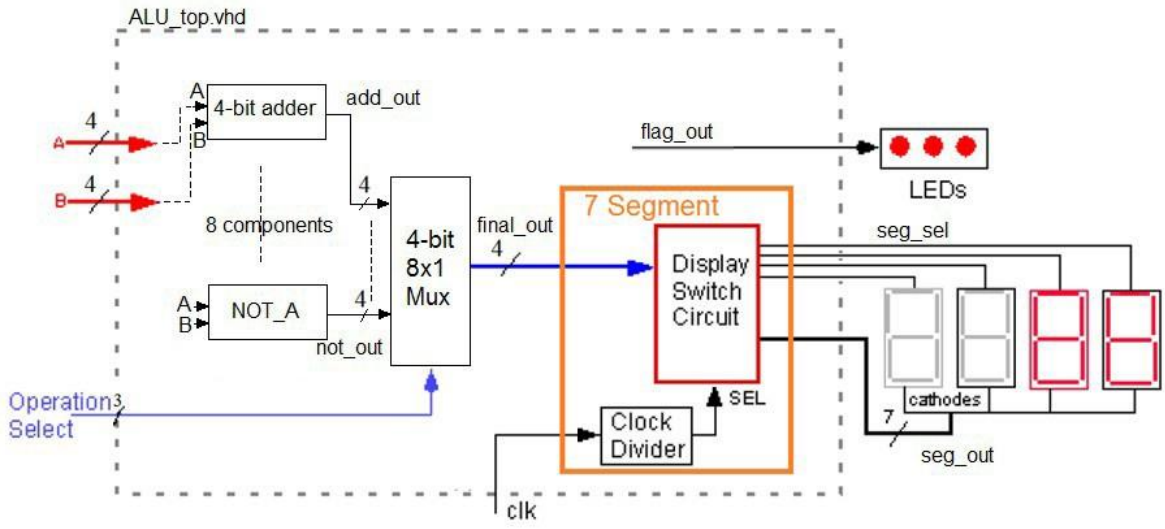


Figure 2 ALU_top module

The lab assistants will allow you to enter into the lab session ONLY IF you bring a pre-lab report that shows you designed each of 8 modules, tested them individually by using Xilinx ISE simulator, and verified each of them to be correct.

To summarize your jobs;

1. Design each of 8 modules and test them individually by using Xilinx ISE simulator.

2. Instantiate all components in ALU_top.

3. Choose the proper result for "final_out" with a multiplexer according to "op_sel" select signal.

4. Create a UCF file and connect your inputs to switches, and flags to LEDs and 7-segment outputs to 7-segment displays on the board.

5. Show your design burned into FPGA to the lab assistants.

You can also make your own complete design without using "ALU_top" as a template. This template is to help you while you are designing this simple ALU.

Note That: *You must be ready to implement the task which will be given to you during your lab sessions.*

ALU_top

```
libraryIEEE;
use   IEEE.STD_LOGIC_1164.ALL;

entity ALU_top is
    Port ( clk : in STD_LOGIC;
        in1 : in STD_LOGIC_VECTOR (3 downto 0);              --Firstoperand
        in2 : in STD_LOGIC_VECTOR (3 downto 0);              --Secondoperand
        op_sel : in STD_LOGIC_VECTOR (2 downto 0);           --Operationsellect
        flag_out : out STD_LOGIC_VECTOR (2 downto 0);        -- You should connect them to leds to see changes
        seg_out : out  STD_LOGIC_VECTOR (7 downto 0);
        seg_sel : out  STD_LOGIC_VECTOR (7 downto 0));
endALU_top;

architecture Behavioral of ALU_top is

component four_bit_adder                                     --Componentdeclerations
          Port ( in1 : in STD_LOGIC_VECTOR (3 downto 0);
                 in2 : in STD_LOGIC_VECTOR (3 downto 0);
                 sout : out STD_LOGIC_VECTOR (3 downto 0);
                 cout : out STD_LOGIC);
endcomponent;

component four_bit_subtract
          Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
                 a2 : in STD_LOGIC_VECTOR (3 downto 0);
                 out1: out STD_LOGIC_VECTOR (3 downto 0));
endcomponent;
componentdecrement
          Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
                 out1: out STD_LOGIC_VECTOR (3 downto 0));
endcomponent;

componentincrement
          Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
                 carry : out STD_LOGIC;
                 out1 : out STD_LOGIC_VECTOR (3 downto 0));
endcomponent;

componentfour_bit_and
          Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
                 a2 : in STD_LOGIC_VECTOR (3 downto 0);
                 out1: out STD_LOGIC_VECTOR (3 downto 0));
endcomponent;

componentfour_bit_or
          Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
                 a2 : in STD_LOGIC_VECTOR (3 downto 0);
                 out1: out STD_LOGIC_VECTOR (3 downto 0));
endcomponent;
componentNotA
          Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
                 out1: out STD_LOGIC_VECTOR (3 downto 0));
endcomponent;
```

```
component four_bit_xor
        Port ( a1 : in STD_LOGIC_VECTOR (3 downto 0);
               a2 : in STD_LOGIC_VECTOR (3 downto 0);
               out1 : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component seven_four
    Port ( in1 : in STD_LOGIC_VECTOR (3 downto 0);
           in2 : in STD_LOGIC_VECTOR (3 downto 0);
           in3 : in STD_LOGIC_VECTOR (3 downto 0);
           in4 : in STD_LOGIC_VECTOR (3 downto 0);
           clk : in STD_LOGIC;
           dp : out STD_LOGIC;
           sel : out STD_LOGIC_VECTOR (3 downto 0);
           segment : out STD_LOGIC_VECTOR (6 downto 0));
end component;
--Signal declarations before begin block of architecture
signal add_out, sub_out, and_out, or_out : STD_LOGIC_VECTOR(3 downto 0);
signal xor_out, inc_out, dec_out, not_out, final_out, carry_show : STD_LOGIC_VECTOR(3 downto 0);
signal add_carry, inc_carry:  STD_LOGIC;
signal Z, V, C : STD_LOGIC;     -- Flag bits
signal dp : STD_LOGIC;
signal sel_out_7 : STD_LOGIC_VECTOR(6 downto 0);
signal seg_sel_4 : STD_LOGIC_VECTOR(3 downto 0);
signal temp1, temp3 : STD_LOGIC;
signal temp2: STD_LOGIC_VECTOR(4 downto 0);
begin
        carry_show<= "000" & C;                 --To show carry bit on seven segment (concatenate)
        A0 : four_bit_adder port map();         -- You should complete the component port maps
        A1 : four_bit_subtract port map();      -- You should also design these modules
        A2 : four_bit_and port map();
        A3 : four_bit_or port map();
        A4 : four_bit_xor port map();
        A5 : increment port map();
        A6 : decrement port map();
        A7 : Not A port map();
        A8 : seven_four port map (final_out, carry_show, "0000", "0000", clk, dp, seg_sel_4, seg_out_7);
-- Seven segment related part
        seg_out <= (seg_out_7 & dp);
        seg_sel <= "1111" & seg_sel_4;

-- Finding the flag_out output
temp1 <= '1' when in2>in1 else
         '0';
temp3 <= '1' when in1 = "0000" else
         '0';
V <= temp1 when op_sel = "001" else
     temp3 when op_sel = "010" else
     '0';
C <= add_carry when op_sel = "000" else
     inc_carry when op_sel = "011" else
     '0';
temp2 <= final_out & C;
Z <= '1' when temp2 = "00000" else
     '0';
flag_out <= Z & C & V; -- concatenation flags.
--
--Your 4-bit 8x1 multiplexer code will be here
--
--
end Behavioral;
```