ALU Unit

Subject: Pulse and Digital Techniques

Teacher: Frýza Tomáš, doc. Ing., Ph.D.

1. Task

- Solve the problem and make demo application to demonstrate it.
- Application in VHDL for CoolRunner-II CPLD starter board (or your own board).
- Group of two students max.
- Results will be presented during the last computer lab in semester (max. 5 mins). Do not include source files (only); use state diagrams, all team members will present.
- All materials will be uploaded before the presentation via links at GoogleDocs.
- Generated signals could be verified by 7-seg. display and/or logical analyzer.

2. Selected Project

The project that I decide to selected is "5. ALU unit with its own set of arithmetic/logic/binary operations." I selected this project because I think that it could be interesting to program a ALU unit into such a simple board like Coolrunner II.

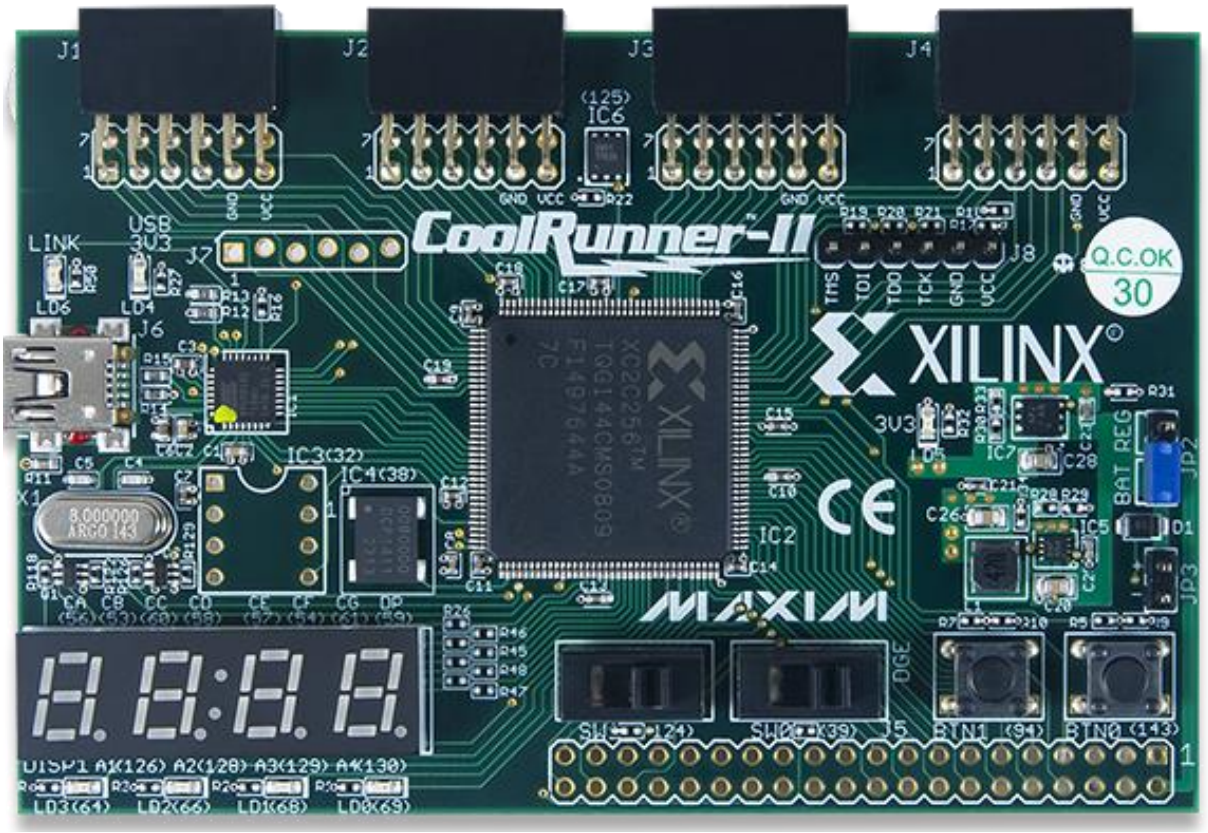
An ALU (Arithmetic Logic Unit) is combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

I inspired in the device 74181, it was the first complete ALU on a single chip. it was used as the arithmetic/logic core in the CPUs of many historically significant minicomputers and other devices. it was used as the arithmetic/logic core in the CPUs of many historically significant minicomputers and other devices. The 4-bit wide ALU can perform all the traditional add / subtract / decrement operations with or without carry, as well as AND / NAND, OR / NOR, XOR, and shift.

My first idea was to implement all the operations as I could, but when I started to do the programming, I realized that I will have some limitations because the board only has two switches and two botoms, so at the end I decided to implement add and subtract operations as arithmetic operations, and AND, NAND, OR and XOR as logical.

3. Materials

For this project we only are going to need the board Coolrunner II:



The CoolRunner-II CPLD starter board is a platform for the evaluation and implementation of designs using a high performance, low-power CPLD.
The DataGATE switch allows designers to evaluate a unique power option that permits input signal blocking, stops input switching, and significantly reduces power consumption, extending battery life. With four 12-pin Pmod ports, is posible to add functions such as analog-to-digital conversion, servo motor interface, serial flash, RS232 serial channel, accelerometers, temperature sensors, and more. Check out our Pmod category for a complete list of modules.

Requisites:
The CoolRunner-II Starter Board requires a USB A to mini-B cable for power and configuration (not included).

Connector(s):
Four 12-pin Pmod ports
USB 2.0 port for power, programming, and data transfer

Features:
Xilinx CoolRunner-II CPLD (XC2C256-TQ144)

A 256-macrocell CoolRunner-II CPLD in a TQ-144 package
An 8 MHz fixed-frequency oscillator and a socket for a crystal oscillator
Expansion connectors for 64 I/O signals (32 on the Pmod ports and 32 on the parallel connector)
A one-wire DS28E01Q EEPROM
DataGATE switch
Requires a USB A to mini-B cable for power and configuration (not included)
Four 12-pin Pmod ports

4. VHDL Programming

As we can see we prepare two sources that's makes the project works.

At first, we prepare an Implementation Constraints File that contains all the inputs, outputs and displays that we can use in our board. In our case we are going to declare all of this except the LEDs and the clock, the reason is that in our project we are not going to need any of them.

```
# CoolRunner-II CPLD Starter Board

# Inputs
# Pushbuttons
NET BTN0        LOC = P143;
NET BTN1        LOC = P94;
# Slide Switches
NET SW0         LOC = P39;
NET SW1         LOC = P124;

# Outputs
# LEDs
#NET LED<0>      LOC = P69;
#NET LED<1>      LOC = P68;
#NET LED<2>      LOC = P66;
#NET LED<3>      LOC = P64;

# Seven-Segment Display(s)
NET D_POS<0>    LOC = P126;
NET D_POS<1>    LOC = P128;
NET D_POS<2>    LOC = P129;
NET D_POS<3>    LOC = P130;
NET D_SEG<0>    LOC = P56;     # a
NET D_SEG<1>    LOC = P53;     # b
NET D_SEG<2>    LOC = P60;     # c
NET D_SEG<3>    LOC = P58;     # d
NET D_SEG<4>    LOC = P57;     # e
NET D_SEG<5>    LOC = P54;     # f
NET D_SEG<6>    LOC = P61;     # g
#NET D_SEG<7>    LOC = P59;     # dp

# Clock
#NET clk         LOC = P38;
```

In the other side we are going to prepare our second source that is the VHDL module which contains all the process that we are going to need for our project to work:

In first place we set the libraries that we are going to need for our project as we can see in the image:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;    -- needed for +/- operations
```

Then we describe the inputs and outputs that we are going to use:

```
6  -- Input/output description
7  entity top is
8  port (
9                        BTN0: in std_logic;     -- button 0
10                       BTN1: in std_logic;     -- button 1
11                   SW0: in std_logic;      -- switch 0
12                       SW1: in std_logic;      -- switch 1
13                       D_POS: out std_logic_vector(3 downto 0);    -- display positions
14                       D_SEG: out std_logic_vector(6 downto 0)     -- display segments
15         );
16 end top;
```

To finish we have to design the internal structure of our project so we define the internal signals, in or case variable1, inp_a, inp_b and inp_alu; and all the process.

```
18 -- Internal structure description
19 architecture Behavioral of top is
20     -- internal signal definitions
21     signal variable1: std_logic_vector(3 downto 0) :="0000";
22     signal inp_a: std_logic_vector(3 downto 0) :="0000";
23     signal inp_b: std_logic_vector(3 downto 0) :="0000";
24     signal inp_alu: std_logic_vector(3 downto 0) :="0000";
```

As we can see we are using "if" functions to express the follow functioning:

If we switch to 0 the SW1 we can change the values of our both inputs A and B:

Turning of the SW0 we select the input A and turning on we select the input B, then if we press the BTN1 we are going to sum up one number to our input and if we press the BTN0 we are going to reset its values to 0.

In the case we turn on the SW1 we pass to do the operations:

So if we turn to 1 the SW0 we are doing arithmetics operations, pressing the BTN0 we are going to substract inp_b to inp_a and if we don´t pulse anything we are going to sum inp_b to inp_a.

Then if we turn to 0 the SW0 we pass to do logic operations, pressing the BTN0 we are doing an OR operation to the inputs, if we press BTN0 and BTN1 at the same time we are doing an NAND operation, if we only press BTN1 we obtain the AND operation and if we don´t pulse anything we are doing a XOR operation.

```
27  begin
28      process(SW1,SW0,BTN0,BTN1)
29      begin
30          if SW1='0' then
31              if SW0='0' then
32                  if BTN0='0' then
33                      inp_a <= "0000";
34                  end if;
35                  if BTN1='0' then
36                      inp_a <= inp_a + 1;
37                  end if;
38              else
39                  if BTN0 = '0' then
40                      inp_b <= "0000";
41                  end if;
42                  if BTN1 = '0' then
43                      inp_b <= inp_b + 1;
44                  end if;
45              end if;
46          else
47              if SW0='1' then
48                  if BTN0 = '0' then
49                      inp_alu <= inp_a - inp_b;
50                  else
51                      inp_alu <= inp_a  + inp_b;
52                  end if;
53              else
54                  if BTN0='0' then
55                      inp_alu <= inp_a or inp_b;
56                      if BTN1='0' then
57                          inp_alu <= inp_a nand inp_b;
58                      end if;
59                  end if;
60                  if BTN1='0' then
61                      inp_alu <= inp_a and inp_b;
62                  else
63                      inp_alu <= inp_a xor inp_b;
64                  end if;
65              end if;
66          end if;
67      end process;
```

To finish we add a process who simplifies the programming and we set the values that we want to be displayed into the 7 segments display.

```
69      process(SW1,SW0)
70      begin
71          if SW1 = '0' then
72              if SW0 = '0' then
73                  variable1 <= inp_a;
74              else
75                  variable1 <= inp_b;
76              end if;
77          else
78              variable1 <= inp_alu;
79          end if;
80      end process;
81
82
83  with variable1 select
84                  D_SEG <= "1111001" when "0001"
85                   "0100100" when "0010",
86                   "0110000" when "0011",
87                   "0011001" when "0100",
88                   "0010010" when "0101",
89                   "0000010" when "0110",
90                   "1111000" when "0111",
91                   "0000000" when "1000",
92                   "0011000" when "1001",
93                   "0001000" when "1010",
94                   "0000011" when "1011",
95                   "1000110" when "1100",
96                   "0100001" when "1101",
97                   "0000110" when "1110",
98                   "1000000" when others;
99
100     D_POS <= "1110";
101 end Behavioral;
```

5. References

https://gitlab.com/tomas.fryza/ict-digital

Textbook of Pulse and Digital Techniques

Lectures from Pulse and Digital Techniques

https://en.wikipedia.org/wiki/74181

https://www.xilinx.com/products/silicon-devices/cpld/coolrunner-ii.html