Objective:

The objective of this lab is to create an 8-bit ALU using behavioral VHDL, whose output is shown on the two 7-segment LEDs. The data inputs to the ALU are connected to the two sets of 8-switch circuits, and the select input is connected to the 3 buttons and a slide switch. In this lab, you will become familiar with the arithmetic VHDL package numeric_std. In addition, you will continue to get experience using test benches to verify the correct functionality of the circuits you specify in VHDL.

Required tools and parts:

Quartus2 software package, ModelSim-Altera Starter Edition, Altera DE0 board.

Pre-lab requirements:

1. Create an 8-bit adder using a behavioral architecture with the ***numeric_std*** package. The entity and architecture must appear in the a file named adder8numeric.vhd and have this exact specification:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity adder8numeric is
  port(input1, input2 : in std_logic_vector(7 downto 0);
      carry_in  : in std_logic;
      sum : out std_logic_vector(7 downto 0);
      carry_out  : out std_logic);
end adder8numeric;
```

   You can use the testbench in Lab2 (adder8_tb.vhd), with appropriate modifications, to simulate adder8numeric.

   Turn in on e-Learning: adder8numeric.vhd and the modified adder8_tb.vhd.

   Printout for your TA (bring to lab): adder8numeric.vhd; representative printout of the simulation output (enough to convince the TA to give you a good grade).

2. Create an 8-bit ALU using a behavioral architecture with the ***numeric_std*** package. The entity and architecture must appear in a file called alu_ns.vhd and have this exact specification:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu_ns is

generic (
   WIDTH : positive := 16
   );
port (
   input1    : in std_logic_vector(WIDTH-1 downto 0);
   input2    : in std_logic_vector(WIDTH-1 downto 0);
   sel       : in std_logic_vector(3 downto 0);
   output    : out std_logic_vector(WIDTH-1 downto 0);
   overflow  : out std_logic
   );
end alu_ns;
```

Note that the width of the ALU is defined by a generic. Therefore, *you must write the architecture to work for any possible width (i.e., don't assume the input is 16 bits)*. The operation of the ALU is described below:

Sel	Output	Overflow
0000	Not input1	'0'
0001	Input1 nor input2	'0'
0010	Input1 xor input2	'0'
0011	Input1 or input2	'0'
0100	Input1 and input2	'0'
0101	input1 + input2	'1' if *input1* + *input2* is bigger than the maximum number that can be written to *output*, '0' otherwise
0110	input1 - input2	'0'
0111	input1*input2 (low half of the mult result; e.g., multiplication of two 8-bit numbers results in a 16-bit number. The output should be the lower 8 bits)	'1' if input1*input2 is bigger than the maximum number that can be written to output *(i.e., >11111111)*, '0' otherwise
1000	0	'0'
1001	0	'0'
1010	Shift input1 left by 1 bit	the original high bit of *input1* before the shift
1011	Shift input1 right by 1 bit	the original low bit of *input1* before the shift
1100	Reverse the bits in input1, write this to output	'0'
1101	Swap the high-half bits of input1 with the low-half bits of input1, write this to output	'0'
1110	0	'0'
1111	0	'0'

Create a VHDL testbench entity (alu_ns_tb). Save the entity in alu_ns_tb.vhd. There is small sample testbench on the lab website, but it is up to you to determine the thoroughness of the testbench. It should test enough cases so you are positive that the architecture is correct. Although the entity must be defined using numeric_std, you can use any package you like for the testbench. Note that the provided sample does not use numeric_std.

Turn in on e-Learning: alu_ns.vhd and alu_ns_tb.vhd. The TAs will grade your VHDL by running it using a testbench that I am providing. Therefore, it is critical you do not change the entity declaration.

Printout for your TA (bring to lab): alu_ns.vhd and alu_ns_tb.vhd; representative printout of the simulation output (enough to convince the TA to give you a good grade).

3. The code of a top-level ALU module (top_level.vhd) is provided for you on the website. All you need to do is to Integrate your code for the ALU (alu_ns.vhd) and the code for two instances of the 7-segment decoder (implemented in Lab 2) with top_level.vhd
   - Note that you will likely need change the name of your 7-segment decoder (to *decoder7seg*) and its PORT definition before you can use it (unless you just happen to define it exactly like the component definition of *decoder7seg*).
   - No simulation is necessary, so you should be very careful and at least compile it to make sure your changes to *decoder7seg* are correct.

Turn in on e-Learning: nothing necessary.

Printout for your TA (bring to lab): nothing necessary.

In-lab procedure:

1. Using Quartus, assign pins to each of the top_level.vhd inputs/outputs such that the signals are connected to the appropriate locations on the board.
   - Assign 8 bits of input 1 to slide switches on the DE0 board
   - Assign 8 bits of input 2 to GPIO headers and connect them to switch circuits built using a DIP switch bank and a SIP resistor pack on a bread board. Use 3.3V and Ground from the GPIO headers to power your switch circuits.
   - Assign 4 bits of the Select input to 3 push button switches and 1 unused slide switch on the DE0 board.
   - Assign the Outputs of the 7-segment decoders to two 7-segment LED displays.
   - Assign the Overflow output to the decimal point (DP) of the most significant 7-segment LED display.
2. Download your design to the board, and test it for different inputs and outputs. Demonstrate for the TA at least one example for each possible select.
3. Be prepared to answer simple questions or to make simple extensions that your TA may request. There is no need to memorize the different packages. If you have done the pre-lab exercises, these questions should not be difficult.