Lab 2: 8-bit Behavioral ALU

EEL 4712 – Spring 2011

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 DIP switches, and the select input is connected to the 4 buttons. In this lab, you will become familiar with two arithmetic VHDL packages: numeric_std (recommended) and std_logic_arith. In addition, you will 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, UF-4712 board.

Pre-requisite:

You must be "up-to-speed" with Quartus before coming to lab. Perform Tutorials 1 and 3 (Appendices B and D) in the textbook if necessary. Also, download and read the UF-4712 documents before coming to lab. ***You should know how to map the I/O of the top level VHDL entity onto the corresponding pins on the UF-4712 board.***

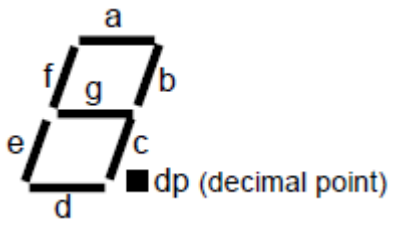
Pre-lab requirements:

1. Design a decoder for the 7 segment display (call it decoder7seg.vhd). The entity must look exactly like this:

```
entity decoder7seg is
  port (
    input  : in  std_logic_vector(3 downto 0);
    output : out std_logic_vector(6 downto 0));
end decoder7seg;
```

Any changes to this entity will cause the test benches used for grading to fail. Also, please use the specified VHDL file name to simplify grading. Create the VHDL architecture to implement the following functionality. Note that the outputs for the LED segments are active low (i.e. a 0 causes the segment to turn on).

Input(i3-i0)	Output (a-g)
0000	0000001
0001	1001111
0010	0010010
0011	0000110
0100	1001100
0101	0100100
0110	0100000
0111	0001111
1000	0000000
1001	0001100
1010	0001000
1011	1100000
1100	0110001
1101	1000010
1110	0110000
1111	0111000



Create a VHDL testbench entity (decoder7seg_tb) for the 7-segment decoder. Save the entity in decoder7seg_tb.vhd. 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. *Test your VHDL with the testbench using ModelSim-Altera Starter Edition. See the tutorial linked off the lab website for an explanation on how to use the tool.*

> Turn in on e-learning: decoder7seg.vhd and decoder7seg_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.

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 should 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	input1+input2	'1' if *input1+input2* is bigger than the maximum number that can be written to *output*, '0' otherwise
0001	input1-input2	'0'
0010	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*, '0' otherwise
0011	Input1 and input2	'0'
0100	Input1 or input2	'0'
0101	Input1 xor input2	'0'
0110	Input1 nor input2	'0'
0111	Not input1	'0'
1000	Shift input1 left by 1 bit	the high bit of *input1* before the shift

1001	Shift input1 right by 1 bit	'0'
1010	Swap the high-half bits of input1 with the low-half bits of input1, write this to output **(clarified)**	'0'
1011	Reverse the bits in input1, write this to output **(clarified)**	'0'
1100	0	'0'
1101	0	'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.

3. Design the same 8-bit ALU using std_logic_arith and std_logic_unsigned (instead of numeric_std). The entity should be saved in alu_sla.vhd, along with a new testbench in alu_sla_tb.vhd. Note that the exact same testbench code can be used. All you have to do is change the name of the alu component that is instantiated. Make sure to use this exact entity specification:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity alu_sla 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_sla;
```

Turn in on e-learning: alu_sla.vhd and alu_sla_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.

4. Integrate your code with the top_level structural entity top_level.vhd (linked off the lab website). Feel free to change the ALU component to use either the numeric_std or std_logic_arith versions. The choice is yours.

> Turn in on e-learning: A graphical illustration of how the provided VHDL connects the components together. Save the illustration in whatever format is convenient (e.g., pdf, jpeg).

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.
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.

Lab report: (In-lab part only)

- If you had any problems with portions of the lab that could not be resolved during lab, please discuss them along with possible justifications and solutions. If you had no problems, this report is not necessary.

> Turn the lab report in on e-learning, if explanation is needed for partial credit. Make sure to turn it in to the "lab" section and not the "pre-lab" section.