Lab 3: 8-bit Behavioral ALU

EEL 4712 – Fall 2023

Objective:

- Create a generic-width Arithmetic Logic Unit (ALU) using behavioral VHDL and learning to use the ***ieee.numeric_std*** package.
- Using a testbench, map the ***generic width*** ALU into an 8-bit ALU to verify its correctness.
- Download your design to the board and demonstrate it for different inputs and outputs.

Required tools and parts:

Intel Quartus Prime, ModelSim-Altera, Terasic DE10-Lite board

Important: For this and the subsequent labs,

- Name your files with your ***Lastname_Firstname_Section#.zip***
- Put your name and section number as a comment at the beginning all VHDL files and other materials that you submit.

Pre-lab Requirements:

Part 1: 8-bit adder using the numeric_std package

Ceate an 8-bit adder using a behavioral architecture with the ***ieee.numeric_std*** package. The entity and architecture must appear in 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 perform a functional simulation of adder8numeric.

Part 1 Pre-lab submission (on Canvas)

Include the following content in `your-UFID/P1` directory (see end of document for more information about submission formatting):

- VHDL code for Part 1: adder8numeric.vhd and the modified adder8_tb.vhd
- Screenshot of functional simulation: showing "Simulation Finished" and 0 assertion errors.

Part 2: Generic-width ALU using the numeric_std package

Design a generic-width ALU using a behavioral architecture with the ***numeric_std*** package.

- **IMPORTANT:** The entity declaration must appear in a file named **alu_ns.vhd** and have the exact specification as follows. Otherwise, it will fail during grading.
- **NOTE:** The ALU uses a generic WIDTH. Therefore, the architecture must work for any possible width (not just 16 bits).

```
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;
```

- Assume all operations of the ALU are with **unsigned** numbers.
- The operation of the ALU is described in the table below:

Sel	Output	Overflow
0000	bitwise-not input1	‘0’
0001	Input1 bitwise-nor input2	‘0’
0010	Input1 bitwise-xor input2	‘0’
0011	Input1 bitwise-or input2	‘0’
0100	Input1 bitwise-and input2	‘0’
0101	input1 + input2	‘1’ if *input1* + *input2* > max value output *(e.g., >11111111 for WIDTH=8)*, ‘0’ otherwise
0110	input1 - input2	‘0’
0111	input1*input2 (low half of product) 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 > max value output *(e.g., >11111111 for WIDTH=8)*, ‘0’ otherwise
1000	Shift input1 left by 1 bit	MSB of *input1* before the shift
1001	Shift input1 right by 1 bit	LSB *input1* before the shift
1010	Reverse the bits in input1, write this to output	‘0’
1011*	Swap the high-half bits of input1 with the low-half bits of input1*	‘0’
1100	0	‘0’
1101	0	‘0’
1110	0	‘0’
1111	0	‘0’

* 1011 Swap: In the case of an odd width, pad with ‘0’ for MSB of result. (For example, 0101100b should become 01000101b)

Create a VHDL testbench entity (alu_ns_tb) and save in a file named alu_ns_tb.vhd. There is a 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 such that you are confident the architecture is correct.

Although for this part (Part 2), the entity must be defined in alu_ns.vhd using numeric_std, you can use any package you like for the testbench. Note that the provided sample testbench uses std_logic_arith to demonstrate different functions.

Important note:

For this lab, you need to compile the alu_ns.vhd (not the alu_ns.vho) file and alu_ns_tb.vhd in ModelSim. If you compile alu_ns.vhd in Quartus (like you did in your previous labs), it will compile the file using the default WIDTH of 16 bits to create the .vho file. So, when you use the .vho file with the given alu_ns_tb.vhd (which set WIDTH to 8) in ModelSim, there is a mismatch.

Part 2 Pre-lab submission (on Canvas)

Include the following content in **`your-UFID/P2`** directory:

- alu_ns.vhd
- alu_ns_tb.vhd
- Screenshot of functional simulation: showing “Simulation Finished” and 0 assertion errors.

Part 3: Top-level module

The code of a top-level module (**top_level.vhd**) is provided for you on the class website. You will:

(1) Create a complete design by integrating top_level.vhd with:
- **alu_ns.vhd** from Part 2
- The code for a 7-segment decoder **from Lab 2**.

(2) Create the following pin assignments:
- Assign the 4 bits of input1 to the leftmost slide switches.
- Assign the 4 bits of input2 to the next four slide switches.
- Assign 4 bits of the Select input to the two rightmost slide switches and the 2 pushbuttons.
- Assign the outputs of the four 7-segment decoders to four 7-segment LED displays.
- Assign the four decimal point (DP) outputs to the corresponding 7-segment LED displays.

(3) Compile top-level.vhd, together with alu_ns.vhd and the 7-segment decode code. Note that the name of your 7-segment decoder must be named ***decoder7seg***.

For Part 3, no simulation is necessary, so you should be very careful in your changes and the code is successfully compiled.

Part 3 Pre-Lab Submission

Include the following content in **`your-UFID/P4`** directory:

- A diagram (e.g., pdf, jpeg format) of the complete design. The diagram should show how the components are connected together, using the entity and signal names in the .vhd files.

***** Important: For full credit,** all prelab materials must be submitted to Canvas by the beginning of your scheduled lab time.*

Pre-Lab submission guidelines

Create the following folder structure:

```
your-UFID/
├── P1/
|    └── part1-files-here
├── P2/
     └── part2-files-here
├── P3/
     └── part3-files-here
```

Replace "your-UFID" with your actual UFID. Compress the folder into

- ***Lastname_Firstname_Section#.zip*** and submit to Canvas.

In-lab Procedure:

- Download your design to the board and test it for different inputs and outputs. Your TA will ask you to demonstrate at least one example for each possible select.
- Be prepared to answer simple questions or to make simple extensions that your TA may request.
    - There is no need to memorize the packages. If you have done the pre-lab exercises, these questions should not be difficult.
- There will be a short lab quiz at the start of lab.

Grade Breakdown

Criteria	Points
----- Pre-Lab -----	**50 pts total**
Part 1: adder8numeric.vhd	5 pts
Modified adder8_tb.vhd	5 pts
Simulation	5 pts
Part 2: alu_ns.vhd	20 pts
alu_ns_tb.vhd	10 pts
Simulation	10 pts
Part 4: Diagram of design	5 pts
----- In Lab -----	**50 pts total**
Demo	30 pts
Quiz	10 pts