LAB #2: Modeling Behavior in VHDL

This homework is due electronically at 5pm on Thursday, September 9, 1999.
NO LATE HOMEWORK WILL BE ACCEPTED.

In this homework, you will complete the following problems using the VHDL simulator within VeriBest’s DesignView. For each problem, turn in your VHDL code and a waveform(s) showing your code working. Be sure to comment your code and annotate your waveforms for full credit.

1. Write and simulate the entity/architecture description of a 3-bit decoder using the conditional signal assignment statement. Test the model with all possible combinations of inputs and plot the decoder output waveforms.

2. Consider the construction of a register file with 8 registers, where each register is 32 bits. Implement the model with two processes. One process reads the register file, while another writes the register file. You can implement the registers as signals declared within the architecture and therefore visible to each process.

3. Implement a 32 bit ALU with support for the following operations: add, sub, and, or, and complement. The ALU should also produce an output signal that is asserted when the ALU output is 0. This signal may be used to implement branch instructions in a processor datapath.

4. Implement and test a 16-bit up-down counter.

1 Arithmetic libraries

For problem 3, you need to create an ALU which supports addition and subtraction. These functions are not built into the standard 1164 library for std_logic_vector’s. To get these functions you will need to use some additional packages. You have two options. You can use ieee.numeric_std package. In this case, you would use the type signed or unsigned instead of std_logic_vector. The numeric_std package includes support for arithmetic operators and type conversion functions for signed and unsigned types (see the handout you got on the first day of class for more info). An example using numeric_std is given below.

You can also use ieee.std_logic_arith. To determine whether you are using signed or unsigned arithmetic, you must include a second package either ieee.std_logic_signed or ieee.std_logic_unsigned. Using this package, arithmetic functions are added for the type std_logic_vector. Again, you can see the handout for more info.

The advantage of the numeric_std is you can specify signed or unsigned arithmetic for each signal or variable. The advantage for std_logic_arith is you can use std_logic_vector type, but whether it is signed or unsigned is fixed based on what package you include.

2 Testbenches

For the register file and ALU above, it will become quite tedious to create tests using the waveform editor. To save time, you will likely want to create a testbench. To do this, you will add an extra process to provide inputs to your design. An example is given below.
3 Example

Below are two examples of a 32 bit adder design. The first uses the `std_logic_arith` package, and the second uses the `numeric_std` package. Notice that the entity's have no ports. That is because the inputs are driven from within. Also, notice that since VHDL is strongly typed, you need type conversion functions to convert integers to `std_logic_vector's` and `signed` vectors (i.e., `conv_std_logic_vector` and `toSigned`). These functions take two parameters: the number and the size of the vector. Finally, you can use wait statements to apply stimulus at different times.

```vhdl
-- A 32 bit adder using std_logic_arith
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
entity alu is
d end alu;

architecture behv of alu is
signal a:std_logic_vector(31 downto 0);
signal b:std_logic_vector(31 downto 0);
signal c:std_logic_vector(31 downto 0);
begin
c <= a + b after 5 ns;
process
begin
a <= conv_std_logic_vector(256,32);
b <= conv_std_logic_vector(128,32);
wait for 10 ns;
a <= conv_std_logic_vector(312,32);
wait for 10 ns;
end process;
end behv;

-- A 32 bit adder using numeric_std
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu is
end alu;

architecture behv of alu is
signal a:signed(31 downto 0);
signal b:signed(31 downto 0);
signal c:signed(31 downto 0);
begint
end behv;
```