CPE462 - VHDL: Simulation and Synthesis - Fall'11
Prof. Nuno Alves (nalves@wne.edu), College of Engineering

Homework Assignment #7

Due Date: Monday, October 10th 2011

Homework policy: If I see something unusual about your work, you can be sure I will ask you about it. If you are unable to explain it to me why you chose some implementation method I will have to consider your entire solution wrong. Now, what is my definition of "unusual work"? Well, things like suspicious parts of code that may have been copied from a website or chunks of your implementation that are very close to one of your colleagues. I really encourage communication amongst yourselves, and I want you guys to have a good time learning while doing these exercises... but please don't blindly copy code. If you really must, at least acknowledge your sources.

1. In class we talked about setting up generic parameters in VHDL. Look at the following parity generator code

```
entity parity_gen is
        generic (n: integer := 7);
        port (input : in bit_vector (n-1 downto 0);
        output : out bit_vector(n downto 0));
end     entity;

architecture myarch of parity_gen is
begin
        process(input)
        variable temp1 : bit;
        variable temp2 : bit_vector (output'range);
        begin
                temp1:='0';
                for i in input'range loop
                        temp1 := temp1 XOR input(i);
                        temp2(i) := input(i);
                end loop;
                temp2(output'high):=temp1;
                output<=temp2;
        end process;
end architecture;
```

a) Provide some code comments which explains exactly what every single line is doing.

b) Change the generic parameter **n** to have the value of 4, and create a test-bench file that will test the circuit. Turn-in a printout of your code and a printout of the wave-forms.

2. The following exercises are based on the following signal declarations:

```
SIGNAL a : BIT := '1';
SIGNAL b : BIT_VECTOR (3 DOWNTO 0) := "1100";
SIGNAL c : BIT_VECTOR (3 DOWNTO 0) := "0010";
SIGNAL d : BIT_VECTOR (7 DOWNTO 0);
SIGNAL e : INTEGER RANGE 0 TO 255;
SIGNAL f : INTEGER RANGE -128 TO 127;
```

a) Practice on operators: fill in the blanks.

```
x1 <= a & c;           ->     x1 <= ________
x2 <= c & b;           ->     x2 <= ________
x3 <= b XOR c;         ->     x3 <= ________
x4 <= a NOR b(3);      ->     x4 <= ________
x5 <= b sll 2;         ->     x5 <= ________
x6 <= b sla 2;         ->     x6 <= ________
x7 <= b rol 2;         ->     x7 <= ________
x8 <= a AND NOT b(0) AND NOT c(1);    ->     x8 <= ________
d <= (5=>'0', OTHERS=>'1');           ->     d <= ________
```

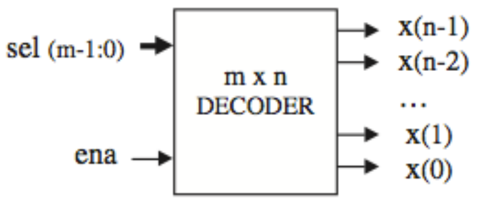
b) Practice in attributes: fill in the blanks.

```
c'LOW              ->     _______
d'HIGH             ->     _______
c'LEFT             ->     _______
d'RIGHT            ->     _______
c'RANGE            ->     _______
d'LENGTH           ->     _______
c'REVERSE_RANGE    ->     _______
```

c) Legal and illegal operations: Verify whether each of the operations below is legal or illegal. Briefly justify your answers.

```
b(0) AND a
a + d(7)
NOT b XNOR c
c + d
e - f
IF (b<c) ...
IF (b>=a) ...
IF (f/=e) ...
IF (e>d) ...
b sra 1
c srl -2
f ror 3
e*3
5**5
f/4
e/3
d <= c
d(6 DOWNTO 3) := b
e <= d
f := 100
```

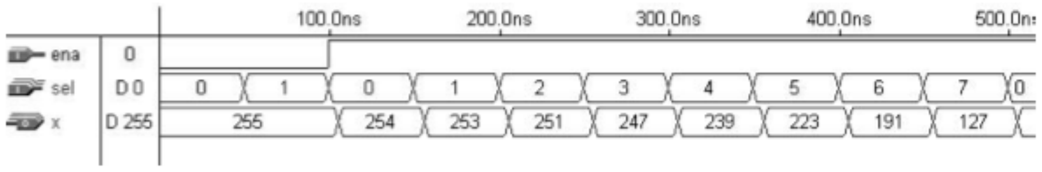
3. The following figure shows the top-level diagram of a generic m-by-n decoder. The circuit has two inputs, sel (m bits) and ena (single bit), and one output, x (n bits). We assume that n is a power of two, so m=log2(n). If ena='0', then all bits of x should be high; otherwise, the output bit selected by sel should be low, as illustrated in the truth table.



ena	sel	x
0	00	1111
1	00	1110
	01	1101
	10	1011
	11	0111

The ARCHITECTURE below is totally generic, for the only changes needed to operate with different values of m and n are in the ENTITY (through sel, line 7, and x, line 8, respectively). In this example, we have used m=3 and n=8. However, though this works fine, the use of GENERIC would have made it clearer that m and n are indeed generic parameters.

```
---------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
ENTITY decoder IS
   PORT ( ena : IN STD_LOGIC;
          sel : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
          x : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END decoder;
---------------------------------------------
ARCHITECTURE generic_decoder OF decoder IS
BEGIN
   PROCESS (ena, sel)
      VARIABLE temp1 : STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
      VARIABLE temp2 : INTEGER RANGE 0 TO x'HIGH;
   BEGIN
      temp1 := (OTHERS => '1');
      temp2 := 0;
      IF (ena='1') THEN
         FOR i IN sel'RANGE LOOP  -- sel range is 2 downto 0
            IF (sel(i)='1') THEN  -- Bin-to-Integer conversion
               temp2:=2*temp2+1;
            ELSE
               temp2 := 2*temp2;
            END IF;
         END LOOP;
         temp1(temp2):='0';
      END IF;
      x <= temp1;
   END PROCESS;
END generic_decoder;
---------------------------------------------
```



The functionality of the encoder above can be verified in the simulation results. As can be seen, all outputs are high, that is, x="11111111" (decimal 255), when ena='0'. After ena has been asserted, only one output bit (that selected by sel) is turned low. For example, when sel="000" (decimal 0), x="11111110" (decimal 254); when sel = "001" (decimal 1), x="11111101" (decimal 253); when sel="010" (decimal 2), x="11111011" (decimal 251); and so on.

Questions:

(a) Create an **identical** replication of the waveform presented above in Active HDL.

(b) In order for this design to operate with another vector size, two values must be changed: the range of sel (line 7) and the range of x (line 8). We want now to transform this design in a truly generic one. In order to do so, introduce a GENERIC statement in the ENTITY, specifying the number of bits of sel (say, n=3), then replace the upper range limits of sel and x by an attribute which is a function of n. Crete a test-bench and simulate your circuit in order to verify its functionality. Turn in a printout of the modified code, the test-bench and the output waveform.

(c) In the original design, a binary-to-integer conversion was implemented (lines 20–26). This conversion could be avoided if sel had been declared as an INTEGER. Modify the code, declaring sel as an INTEGER. The code should remain truly generic, so the range of sel must be specified in terms of n. Crete a test-bench and simulate your circuit in order to verify its functionality. Turn in a printout of the modified code, the test-bench and the output waveform.

4. List all operators, attributes and generics in the following code:

```
entity parity_det is
        generic (n: integer := 7);
        port (input : in bit_vector (n downto 0);
        output : out bit);
end entity;

architecture myarch of parity_det is
begin
        process(input)
        variable temp : bit;
        begin
                temp:='0';
                for i in input'range loop
                        temp := temp XOR input(i);
                end loop;
                output<=temp;
        end process;
end architecture;
```