CPE462 - VHDL: Simulation and Synthesis - Fall'11

Prof. Nuno Alves (nalves@wne.edu), College of Engineering

Homework Assignment #5

Due Date: Monday, October 3rd 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.

This homework set is out of 100 points. Q1-Q4 are all worth 25 points. The extra credit question is also worth 25 points.

1. In no more than two lines describe what is the difference between these two codes:

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

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix := ( "01000", "10010", "11001");
begin

end architecture;
```

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

entity test is
end entity;

architecture myarch of test is
type matrix is array (2 downto 0) of std_logic_vector(4 downto 0);
signal x: matrix;
begin

        x(0) <= "11001";
        x(1) <= "10010";
        x(2) <= "01000";

end architecture;
```

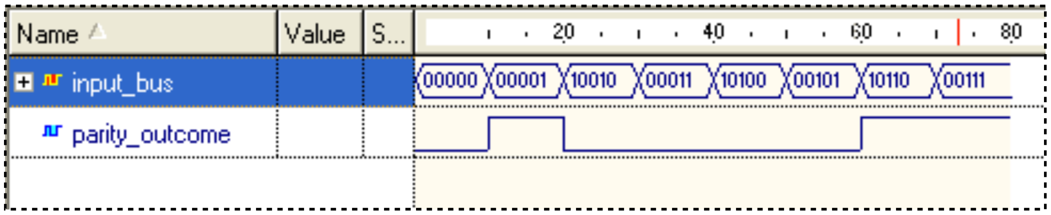
2. Why is this assignment illegal?

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

entity test is
        port (signal x: in bit);
end entity;

architecture myarch of test is
begin

        x <= ‘0’;

end architecture;
```

3. A parity bit is a bit that is added to ensure that the number of bits with the value “one” in a set of bits is even or odd. These Parity bits are used as the simplest form of error detecting code. It works like this; if the number of “ones” in a string of bits is even, then parity bit will be 0. Of course, if the number of “ones” in a string of bits is odd, then parity bit will be 1. I challenge you to write a program that will find the parity bit for a 4-bit bus of type std_logic_vector. Use the test-bench available on the course website and the following entity to guide you on your implementation. Turn in a copy of your code and a printout of the waveform. You may **NOT** modify the test-bench.

```
entity parity_encoder is
        port (input_bus   : in std_logic_vector(3 downto 0);
        output_bus : out std_logic_vector(4 downto 0)
        );
end entity;
```

4. Now create another program that that will tell if the parity of a 4bit bus is correct or not. Use the test-bench available on the course website and from it determine the input/output names + sizes of your implementation. To help you out, I am including the output waveform. Turn in a copy of your code. You may **NOT** modify the test-bench.



Extra Credit. One of the biggest problems that face modern circuits is transient errors. Generally speaking, a transient error is caused when an external source causes a momentary localized energy deviation at the electron level. In binary terms, a transient error may cause a specific logic zero to temporarily become a logic one, and vice-versa. There are many sources of transient errors; for example alpha-particles that come from the sun. It is imperative that we ensure that our circuit operates as it is expected; this means we must incorporate extra functionality in the circuit to ensure its correctness.

Hamming Code - When data is transmitted from one location to another there is always the possibility that an error may occur. There are a number of reliable codes that can be used to encode data so that the error can be detected and corrected. In this homework you will explore a simple error detection-correction technique called a Hamming Code. A Hamming Code can be use to detect and correct one-bit change in an encoded code word. This approach can be useful as a change in a single bit is more probable than a change in two bits or more bits.

General algorithm - The following general algorithm generates a single-error correcting (SEC) code for any number of bits.

a) Number the bits starting from 1: bit 1, 2, 3, 4, 5, etc.

b) Write the bit numbers in binary. 1, 10, 11, 100, 101, etc.

c) All bit positions that are powers of two (have only one 1 bit in the binary form of their position) are parity bits.

d) All other bit positions, with two or more 1 bits in the binary form of their position, are data bits.

e) Each data bit is included in a unique set of 2 or more parity bits, as determined by the binary form of its bit position.

I. Parity bit 1 covers all bit positions which have the least significant bit set: bit 1 (the parity bit itself), 3, 5, 7, 9, etc.

II. Parity bit 2 covers all bit positions which have the second least significant bit set: bit 2 (the parity bit itself), 3, 6, 7, 10, 11, etc.

III. Parity bit 4 covers all bit positions which have the third least significant bit set: bits 4–7, 12–15, 20–23, etc.

IV. Parity bit 8 covers all bit positions which have the fourth least significant bit set: bits 8–15, 24–31, 40–47, etc.

In general each parity bit covers all bits where the binary AND of the parity position and the bit position is non-zero. This general rule can be shown visually:

Bit position		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Encoded data bits		**p1**	**p2**	**d1**	**p4**	**d2**	**d3**	**d4**	**p8**	**d5**	**d6**	**d7**	**d8**	**d9**	**d10**	**d11**	**p16**
Parity bit coverage	p1	X		X		X		X		X		X		X		X	
	p2		X	X			X	X			X	X			X	X	
	p4				X	X	X	X					X	X	X	X	
	p8								X	X	X	X	X	X	X	X	
	p16																X

Example: This is how you encode a number with Hamming Code

A byte of data: 1001-1010

Create the data word, leaving spaces for the parity bits: _ _ 1 _ 0 0 1 _ 1 0 1 0

Calculate the parity for each parity bit (a ? represents the bit position being set):

* Position 1 checks data bits d1, d3, d5, d7, d9, d11; so d1 XOR d3 XOR d5 XOR d7 XOR d9 XOR d11 is 0

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	1	p4	0	0	1	p8	1	0	1	0
Parity bit coverage	p1	0		X		X		X		X		X	
	p2		?	X			X	X			X	X	
	p4				?	X	X	X					X
	p8								?	X	X	X	X
	p16												

* Position 2 checks bits d2, d3, d6, d7, d10, d11

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	1	p4	0	0	1	p8	1	0	1	0
Parity bit coverage	p1	0		X		X		X		X		X	
	p2		1	X			X	X			X	X	
	p4				?	X	X	X					X
	p8								?	X	X	X	X
	p16												

* Position 4 checks bits 4,5,6,7,12:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	1	p4	0	0	1	p8	1	0	1	0
Parity bit coverage	p1	0		X		X		X		X		X	
	p2		1	X			X	X			X	X	
	p4				1	X	X	X					X
	p8								?	X	X	X	X
	p16												

* Position 8 checks bits 8,9,10,11,12:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	1	p4	0	0	1	p8	1	0	1	0
Parity bit coverage	p1	0		X		X		X		X		X	
	p2		1	X			X	X			X	X	
	p4				1	X	X	X					X
	p8								0	X	X	X	X
	p16												

Hamming encoding of 1001-1010 is 0111-0010-1010.

Finding and fixing a bad bit - The above example created a code word of 0111-0010-1010. Suppose the word that was received was 0111-0010-1110 instead. Then the receiver could calculate which bit was wrong and correct it. The method is to verify each check bit. Write down all the incorrect parity bits. Doing so, you will discover that parity bits 2 and 8 are incorrect. It is not an accident that 2 + 8 = 10, and that bit position 10 is the location of the bad bit. In general, check each parity bit, and add the positions that are wrong, this will give you the location of the bad bit.

Three part question

(a) Test if the following code words are correct, assuming they were encoded using Hamming Codes. If one string incorrect, indicate what the correct code word should have been. Also, indicate what the original data was.

1. 0101-0110-0011
2. 1111-1000-1100
3. 0000-1000-1010

(b) Create a VHDL circuit that will create a 12-bit Hamming-Code encoding from 8-bits of data. Use the test-bench provided in the course website to figure out the input/output datatypes & signal names, and to test the circuit. Turn in a print-out of the code, as well as the output wave-form.

(c) Create a VHDL circuit that will take a 12-bit Hamming encoded string as an input and it will output the corrected 12-bit sequence. You can assume that only one error will show up in your input sequence. Turn in a print-out of the code, as well as the output wave-form.