Lab session #2

Preliminary remarks & suggestions

The quality of a design and its description are two independent factors. We can express the initial design by a textual VHDL program. We realize the design automatically by synthesis software. Using VHDL and synthesis software does not lead automatically to either a good or a bad design. VHDL description and synthesis software, however, can shield tedious implementation details and greatly simplify the realization process. They allow us to have more time to explore and investigate alternative design ideas.
Derivation of an efficient, synthesizable VHDL description requires two major tasks:

- Research to find an efficient design
- Develop VHDL code that accurately describes the design

For a problem in digital system development, there is seldom a single unique solution. A large number of possible designs exist. The resulting implementations differ in size and performance and their quality may vary significantly. There is no simple, mechanical way to derive an efficient design. It frequently relies on a designer's experience, insight and understanding of the problem.

The next step is to derive VHDL code that describes the design accurately. Although the VHDL textual code cannot precisely specify the final structural implementation (it's up to the synthesis tool), it describes the "big picture" that establishes the basic skeleton of the circuit. For a complex design, it is useful to draw a rough schematic sketch to help in locating the key components and identifying the critical path (path with the longest delay). In addition to faithfully describing the intended design, good VHDL code should be clear and compact, and can be easily "scaled." Scalability concerns the amount of code modification needed when the signal width of a circuit changes.

***RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability.* By Pong P. Chu Copyright @ *2006* John Wiley & Sons, Inc. 163**

Guidelines for VHDL formatting

- Include an information header for each file.
- Be consistent with the use of case.
- Use proper spaces, blank lines and indentations to make the code clear.
- Add necessary comments.
- Use symbolic constant names to replace hard literals in VHDL code.
- Use meaningful names for the identifiers.
- Use a suffix to indicate a signal's special property, such as n for the active-low signal.
- Keep the line width within 72 characters so that the code can be displayed and printed properly by various editors and printers without wrapping.

Sample information header

```
-- Author: yourname
-- File : vfilename.hd
-- Design units:
-- entity entityname
-- function: summarize what the function does
-- input: list of inputs
-- output: list of outputs
-- architecture architecturename
-- Library/package:ieee.std-logic-1164: to use std-logic
-- Synthesis and verification (optional):
-- Synthesis software: . . .
--  Options/script: . . .
--  Target technology: . . .
--  Testbench: testbenchname.vhd
--  Revision history
--  Version
--  Date:
--  Comments:
```

Activity #1: 8:1 multiplexer

Design a 8:1 multiplexer with 8 data inputs, 3 selection inputs and 1 output according to the following description styles:

1. dataflow using conditional signal assignments (WHEN ELSE statements)
2. dataflow using selected signal assignments (WITH SELECT WHEN statements)
3. behavioural using IF THEN ELSIF statements
4. behavioural using the CASE WHEN statement
5. structural: create a behavioural description of 4:1 multiplexer and a structural description of a 2:1 multiplexer, instantiate and interconnect them to create a structural description of the 8:1 multiplexer.

Create a testbench to simulate the circuit on relevant inputs.
The testbench should include a configuration statement to choose among different architectures.

Activity #2: 3:8 decoder with enable

A binary decoder is an n to 2^n decoder, which has an n-bit input and a 2^n-bit output. Each bit of the output represents an input combination. Based on the value of the input, the circuit activates the corresponding output bit.
Design a 3:8 decoder with 3 selection inputs, 1 enable input and 8 outputs. If the enable signal is 0, the outputs are set to the all-0 pattern. Else the device works as a decoder. Design the circuit according to the following description styles:

1. dataflow using conditional signal assignments (WHEN ELSE statements)
2. dataflow using selected signal assignments (WITH SELECT WHEN statements)
3. behavioural using IF THEN ELSIF statements
4. behavioural using the CASE WHEN statement

Create a testbench to exhaustively simulate the circuit.

Activity #3: 4:2 priority encoder

A priority encoder checks the input requests and generates the code of the request with highest priority. For example, in a 4:2 priority encoder there are four input requests, req3, req2, req1 and req0. The outputs include a 2-bit signal, named code, which is the binary code of the highest-priority request, and a 1-bit signal, named active, which indicates whether there is at least an active request. The req3 request has the highest priority. When it is asserted, the other three requests are ignored and the code signal becomes "11". If req3 is not asserted, the second highest request, req2, is examined. If it is asserted, the code signal becomes "10". The process repeats until all the requests are checked. The code signal returns "00" when only req0 is asserted or no request is asserted. The active signal is used to distinguish the two conditions (active is ‘1’ if req0 is asserted, otherwise it is ‘0’). Design the circuit according to the following description styles:

1. dataflow using conditional signal assignments (WHEN ELSE statements)
2. dataflow using selected signal assignments (WITH SELECT WHEN statements)
3. behavioural using IF THEN ELSIF statements
4. behavioural using the CASE WHEN statement

Create a testbench to exhaustively simulate the circuit.

Activity #4: synthesis results

After synthesis completes, you can view the reports, and open, analyze, and use the synthesized design. The Reports window contains a list of reports provided by various synthesis and implementation tools in the Vivado IDE.

The Synthesis Report:

- Is an ASCII text file
- Is a hybrid between a report and a log
- Contains information about the synthesis run

During synthesis, the Synthesis Report allows you to:

- Control the progress of the synthesis
- Review what has occurred so far

After synthesis, the Synthesis Report allows you to:

- Determine whether the HDL description has been processed according to expectations
- Determine whether device resources utilization and optimization levels are likely to meet design goals once the synthesized netlist has been run through the implementation chain

Using the features provided by the Xilinx Vivado, first synthesize the circuits designed in activities 1, 2 and 3, then compute the device utilization and the estimated maximal working frequency.

Compare the implementations using the following table:

Circuit	**Inputs [#]**	**Outputs [#]**	**BELs [#]**	**Maximum Working Frequency [MHz]**
8:1 multiplexer				
dataflow (WHEN ELSE)				
dataflow (WITH SELECT)				
behavioural (IF THEN ELSIF)				
behavioural (CASE WHEN)				
Structural				
3:8 decoder with enable				
dataflow (WHEN ELSE)				
dataflow (WITH SELECT)				
behavioural (IF THEN ELSIF)				
behavioural (CASE WHEN)				
4:2 priority encoder				

dataflow (WHEN ELSE)				
dataflow (WITH SELECT)				
behavioural (IF THEN ELSIF)				
behavioural (CASE WHEN)				

The information about number of inputs, outputs and BELs can be found in the report utilization.
How can you calculate the maximum working frequency? Try it yourself.