ELEX 2117 VHDL Coding Guidelines

Version 1. These are subject to change and apply to labs and exams starting with Lab 5.

Requirements

Your design will be considered **incorrect**, even if it behaves correctly, if you do not follow the guidelines in this section.

Synchronous design

Your design may use only one clock signal unless otherwise specified. The same signal must appear in all arguments to the `rising_edge()` function and only that signal may be listed in **Compilation Report** > **Fitter** > **Resource Section** > **Control Signals** as a Clock.

Explanation Modern hardware and design tools assume synchronous (one clock) design. Using "computed" clocks leads to inefficient designs that are difficult to design and verify.

No process statements

You may not use `process` statements for synthesis. Registers must be defined by conditional assignments with a `rising_edge(`*clock*`)` condition.

Explanation Statements in processes are executed sequentially. This often leads students to believe that the synthesized hardware carries out the operations described by the process. Processes can also include variable assignments whose semantics differ from those of signal assignments.

Not using processes allows students to start using HDLs for synthesis before learning about these topics. Concurrent assignments are easy for students to map into the hardware, the code tends to be less verbose and concurrent assignments are sufficient to describe any synthesizable design.

Explicit register input signals

Registers must be described as:

signal `<=` *signal*`_next when rising_edge(`*clock*`);`

Explanation This simplifies the description in most cases and makes it possible to detect events (changes of state) because both the current and the next values of a register are available.

Only IEEE 1164 types

You may only use `std_logic`, `std_logic_vector`, `unsigned`, enumerated types and arrays of these.

Explanation Although `bit`, `bit_vector` and `integer` types are built in to VHDL, they are rarely used for synthesis. Using the IEEE standard types avoids the need to switch to different types when additional values (`Z`, `X`, `-`, `U`) are needed.

File-level comments

As a minimum, each source file must include, near the start of the file, comments that include: the file name, a line describing the purpose of the file, the author's name and the date.

Explanation These file-level comments allow you to identify the purpose and source of the code without having to read the contents.

Recommendations

It is also strongly recommended you follow the following guidelines.

Consistent signal naming helps avoid confusion. The following are widely-recognized conventions:

- append `_next` to the register output name to derive the name of the register input signal
- append `_n` to active-low signals
- append `_t` to type names
- append `_out` to names of output ports that are copies of internal signals

- append `_in` to names of input ports that have a corresponding internal signal (e.g. a synchronized or debounced version)

A consistent indentation style throughout the file helps spot errors. An `end` should be indented the same as its corresponding `begin`. Beginners often ignore this recommendation and then have problems spotting trivial errors.

Additional comments next to port and signal declarations and for non-obvious portions of your design are a good idea. These comments should explain why you're doing something rather than repeating what is obvious from the code. They will help you and others understand the purpose of each section of code.

Use enumerated types for states to simplify your code and help the synthesizer optimize your design.