Digital electronics 1

VHDL Guidelines

Tomas Fryza
Dept. of Radio Electronics, Brno University of Technology, Czechia

2019/20L

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

Contents

List of tables

1 General information

1. Do not start writing any VHDL code until the specifications have been completely understood.
2. Use version control software, such as Git, SVN, Mercurial, etc.
3. A project should consist of one top-level module and several sub-level modules. The top-level module should only contain the instantiations of the sub-level modules.
4. All file names must be lowercase and in the form: `entity_name.vhd` for source file containing the VHDL module, `tb_entity_name.vhd` for testbenches and `package_name_pkg.vhd` for packages.
5. Use only IEEE standard packages `ieee.std_logic_1164.all` and `ieee.numeric_std.all`. Do not use the Synposys' libraries `std_logic_arith`, `std_logic_(un)signed`, `numeric_bit`, or any other no IEEE library.
6. Indent the code. It helps to read and understand the code. Do not use *hard tabulators* because differences in text editors make the positioning of the tabs unpredictable. Thus, preferably use a sequence of 2–4 spaces.
7. Every VHDL file starts with a standard header where important information about the file content is stored.
8. Comment the general functionality of each section/sub-section.
9. The comments written in the code must include valuable and significant information. Avoid VHDL syntax or semantics comments in your projects.
10. Use English comments only. Be sure to update the comments if the code changes. Thing of yourself reading the code after a decade.
11. The line length of the code should not exceed 80 characters. Use <enter> to split long lines and indent the next line to show continuity from the previous line.
12. Use blank lines to make code more readable.

2 Name style rules

1. VHDL elements should have names containing only the letters A–Z, a–z, underscore _, and digits 0–9. Use underscore for improving names readability.
2. Keep the signal name's consistency through all the hierarchy of the design.
3. There are different prefixes and suffixes. They should be used to clearly identify the type of the user defined name.

Table 2.1: Prefixes and suffixes

Description	**Extension**	**Example**
entity input signals	`<lower_case_id>_i`	`data_i`
entity output signals	`<lower_case_id>_o`	`led_o`
clock	`clk_<group_id>`	`clk`, `clk_cnt`
reset	`reset_<group_id>`, `rst_<group_id>`	`reset`, `reset_cnt`
low active signal	`<lower_case_id>_n`	`reset_n`
generics	`g_<UPPER_CASE_ID>`	`g_DATA_WIDTH`
constant	`c_<UPPER_CASE_ID>`	`c_CLK_1MHZ_PERIOD`
internal signal	`s_<lower_case_id>_from_to`	`s_data_ha0_ha1`
type definition	`t_<lower_case_id>`	`t_mytype`
states of FSM	`<UPPER_CASE_ID>`	`IDLE`, `WAIT_FOR_DATA`

Table 2.2: Label prefixes

Description	**Extension**	**Example**
process	`p_<lower_case_id>:`	`p_clk_ena:`
loop	`l_<lower_case_id>:`	`l_sync_event:`
generate	`gen_<component_name_id>:`	`gen_adders:`

4. Some recommended abbreviations of often used identifiers:

`clk`	clock
`rst` or `reset`	synchronous reset
`arst` or `areset`	asynchronous reset
`ena` or `enable`	enable
`ce`	clock enable
`cnt`	counter
`addr`	address
`src`	source
`dst`	destination

3 Entity

1. Write an I/O port declaration per line of code, along with a comment about the port.
2. Use only port modes `in` and `out`.
3. Use only `std_logic` and `std_logic_vector` data types for I/O ports.
4. Use descending ordering (high `downto` low) if the vector represents a bus. Least Significant Bit (LSB) should be the right-most bit in the vector and should be 0 (not 1).
5. Size of the vector should be parametrized.
6. Align the colons and port types in the entity port.

7. Whenever is possible, use only active high signals. When using active low signals, name the signal according the rule explained above.

8. Use one entity + architecture per file.

4 Architecture

1. Use meaningful and conventional names for the architecture. In the appropriated context use `dataflow`, `behavioral`, `structural`, `rtl`, `testbench`.

2. Statements in architecture are not sequential.

3. Input ports can only be read, they cannot be assigned. Output ports can only be assigned, they cannot be read. Use internal signals instead.

4. Conditions are written inside parenthesis. There is a space outside parenthesis, but not inside. There is a space after a comma ”`,`”, but not before. There is a space on both sides of a colon ”`:`”.

5. Check the code for latch generation. For long `if-then-else` or `case` statements assign values to signals by default to avoid missing any assignment.

6. Always use named signal mapping in instantiations, never ordered mapping. The names signal mapping works even if the declaration order of ports in entity changes. Align the `=>` sign in generic and port maps.

7. Use `for generate` statement for repetitive instantiations of the same component.

8. Always read through all the warnings generated by the synthesis tool. Get ride off the ones that can be solved.

9. Utilize the RTL viewer to have a glace of the synthesized logic.

5 Synchronous process

1. Every process should be preceded by a comment about the functionality of the code.

2. Label every process.

3. Processes statements are sequential.

4. Use one clock signal and one edge.

5. Use `rising_edge`, resp. `falling_edge` functions instead of the `clk’event and clk=’1’` method.

6. Never use `rising_edge` or `falling_edge` functions to detect the edge of a non-clock signal.

7. Preferably use the synchronous sets and resets.

8. Sensitivity list of a synchronous process has one signal (`clk`) when reset is synchronous and two signals (`clk`, `areset`) when asynchronous reset is used.

9. Signals that are assigned inside synchronous process, will become D-flip flops at synthesis.

10. Never give initial value to signal at declarative part because initial values are ignored by most synthesis tools (but causes a warning). The functionality of the simulated design may not match the functionality of the synthesized design.

6 Combinatorial (asynchronous) process

1. An asynchronous process must have all input signals in the sensitivity list. (Input signals are on the right side of assignment or in conditions: `if`, `for`, `case`.)
2. If-clauses must be complete. Cover all cases, e.g. with `else` branch. Every signal is assigned in every `if` branch. Otherwise, you will get latches. Always check synthesis report.
3. Same signal cannot be on both sides of assignment in combinatorial process. That would create combinatorial loop, i.e. malfunction.

7 Finite State Machine (FSM)

1. Before coding the FSM, read through full the specifications and then draw either the State Diagram or the Algorithmic State Machine (ASM).
2. Declare an enumerated type that holds each state of the FSM. Use meaningful name for each state, such as `IDLE`, `WAIT_FOR_DATA`, `START_TX`. Do not use `S0`, `S1`, `S3`.
3. The FSM code must have a synchronous or asynchronous reset, to be correctly inferred.
4. When describing an FSM, use two processes: one combinational process for the next state logic and the other, a sequential process to describe the current state logic.

8 Testbench

1. Each entity requires at least one associated testbench.
2. Code coverage should be as high as possible. Verify correct functionality with different generic values.
3. Do use as many asserts as possible to make the testbench auto-checkable.
4. Do use `wait` statements in processes and do not use sensitivity lists at all.
5. Use separate processes for clock generation, reset generation, and data generation.
6. When testing an FSM, verify the behave when it is forced to reach no specified states.
7. Preferably the testbench should be written by a different designer than the one who created the device under test.

References

[1] Cristian Sisterna. *Design and Coding Style Guidelines for Synthesizable VHDL-FPGA Code.*

[2] AMIS BRNO. *VHDL/Verilog Coding Guidelines.*

[3] Patrick Loschmidt, et al. *Guidelines for VHDL Coding.*

[4] Xilinx. *Coding Style Guidelines.*

[5] Tampere University of Technology. *VHDL Coding Rules.*