ALSE's
VHDL Design Rules & Coding Style

version 4.1

http://www.ALSE-FR.com

Introduction

These rules and coding style are the result of more than 23 years of HDL design and teaching experience, hundreds of complex ASIC & FPGA projects, hundreds of thousands of lines of code, and the development of a very rich portfolio of complex IPs.

This list is "reasonable": small and simple enough to be easily remembered, "no-nonense" in that only useful rules have been kept. However it is covering a lot of the usual mistakes and it can significantly enhance the quality of the code when adopted.

Caveat

It is probably useful to remind at this stage that :

- Following the rules is not sufficient *per se*: this Coding Guide is in **not teaching** the **fundamental principles** that any HDL designer must absolutely master to create efficient and reliably working designs.
  It is the purpose of our **Training Courses** !
- The other way around is also true: some of these rules can be bent while a correct, working, design is achieved, if there is a good understanding of the potential issues that may result.
- The *Naming Conventions* proposed here are not absolute rules. You may decide to adopt *other* naming conventions, but it is *not* acceptable to *not* have *any* naming convention enforced !

In summary, this document is only proposing a number of recommendations that, if followed, will reduce the design risks and globally augment the code quality and reliability.

Copyright

Conventions

RTL: Register Transfer Level (almost equivalent to « synthesizable »).

BEH: Behavioral code almost equivalent to « synthesizable »).

SIM: Test benches, test code and ressources for simulation and verification in general.

VHDL 2008

RTL : **In general, DO NOT ADOPT VHDL 2008 constructs for synthesizable code.**
Rationale : As of end 2016, very few FPGA synthesis tools support VHDL2008 completely, and this causes very serious portability concerns.

SIM: VHDL2008 can be attempted in behavioral or test bench code provided you verify first that the simulator(s) that you are going to use properly support(s) the constructs you want to use.

The situation may evolve in the future, however, many FPGA families that will remain in use in the next years will only be supported by tools that will not evolve and therefore not support VHDL2008.

Design Data Organization

O_1) Source **Files Naming** Convention:
Extension = .vhd
Name: same as section (entity, configuration, package) with prefix or suffixes e.g.:
« xxx_TB » for test bench, « xxx_CF » for configuration, « xxx_PK » for package...

O_2) **Only one design unit entity per file** (with the exception of configurations which may be grouped in one file).

O_3) In general, **avoid splitting Entity and Architecture** in different files.

O_4) If one entity has several architectures, there must exist only one **single and unique entity** section.
Multiple architectures are possible in the same file, be sure to place the one used for synthesis in the last position (bottom of file), so the use of a configuration will not be compulsory for synthesis.

O_5) VHDL Configurations: can be useful for simulation (case of multiple architectures with selection at compilation time), but we suggest to avoid them for synthesis if possible.

O_6) RTL: A module (entity) should **not** contain several **different and independent functionalities** except for structural entities (instantiating of sub-modules).

O_7) Use **Scripts** (command line or preferably Tcl) for **synthesis** and for **simulation** tasks.

Cosmetic (yet important) Rules

P_1) Every design file must be properly **documented** in a **standardized header** including at least: actual file name, Title & purpose, Author, Creation Date, Version, simple Functional Description, Specific issues, HDL standard followed, Revisions & ECOs.
If applicable: FPGA target, Speed and Area estimation, tools names and versions used.

P_2) Define only **one** single **port per line**, and **comment every port**.

P_3) The HDL files must **not include any « hard tab »** character (HT) but only **soft spaces**, and must be properly **aligned** and **indented**. Note that a mess can often be cleaned up using Emacs VHDL mode's beautifier.

P_4) Total **line length** should be no more than 132 characters.
First non-space character should appear before column 80.

P_5) Avoid using **accented characters** (this should not happen if only English is used).

P_6) Only **one single executable statement per line**.
This rule is important for readability, simulation, debug, coverage...

P_7) The VHDL code must include **significant and value-adding comments**, in **English**.
Every process or other concurrent statement should be preceded by a clear comment summarizing its purpose.

P_8) It should be easy to **match** the **requirements** and the HDL **code**, **both ways**.

P_9) The **comments** and **header information** must be kept **accurate** and **up-to-date** throughout code changes and design iterations. No comment at all is often preferable to a misleading or incorrect comment.

P_10) There should be **no piece of code commented out**.
Inactive (dead) or incorrect code should be deleted. If absolutely needed, an older version of the architecture can be kept as a reference, for comparison or for non-regression purpose.

Naming Rules

N_1) Adopt **(System)Verilog-friendly Identifiers**, **OS-friendly** names for design units, always restrict yourself to using plain **7-bits Ascii** (avoid accents) and adopt meaningful names (in English).

N_2) **Avoid too short names** (like « i », « n »...) except for very short scope since they tend to be difficult to locate (search for) using a text editor or text searching tools.

N_3) Use **Short identifiers** when the scope is local, and Longer more explicit (English) identifiers for wider scope or for global items.

N_4) Use the « **_t** » suffix for **types** and **sub_types**, like:

```
subtype Byte_t is std_logic_vector (7 downto 0);
```

N_5) Use **Upper and Lower** cases for improved readability e.g. `LocalReadEnable`.
The use of the underscore character "_" for this purpose is also possible.

N_6) Avoid using **1 (one), l (lower case L), I (uppercase i) O (uppercase Oh) and 0 (zero)** in situations that may be visually ambiguous. (like DO vs D0)

N_7) Use meaningful and conventional names for **architecture kinds** like:
"`RTL`", "`Behavioral`", "`Structural`","`Test`", in the appropriate context.

N_8) Adopt a **unique and consistent notation for active low signals** (like **n**Reset or Reset**_n** for example, but not both). All *active low signals must be clearly identifiable*.

N_9) Define **internal** (equivalent) **signals** with a **derived name** when you need to read back output Signals (for example internal signal named **i**OutBus assigned to OutBus output).

N_10) Use instance names derived from the entity names. For example:

```
Fir16x8_i1 Fir16x8 port map ...
```

N_11) ***Avoid*** using ***too long names*** for identifiers, especially for entities when possible.
Some coding habits and automatic code generation tools often break this rule so it can be only a recommendation.

N_12) Be sure to name **Clocks** and **Resets** using an unambiguous name like for example:
Clk, Clock, Reset, Rst, Clk_100M, RST_Clk50M ...

N_13) Do **not** use **extended** identifiers.

Coding Rules

C_1) Use **VHDL** 93 / **2001** standard: VHDL '87 was messy and is definitely obsolete, and you should rad the note about VHDL 2008.

C_2) RTL: Use **exclusively IEEE** libraries: **std_logic_1164** & **numeric_std**..
Do **NOT** use: **std_logic_arith, std_logic_(un)signed, numeric_bit**, and other proprietary libraries...
RTL: **Math_real** should be used with care but is more and more widely supported.

C_3) BEH & SIM: use also **textio** (IEEE) and **std_logic_textio** (Synopsys).
Math_real is also possible and useful, recommended in test benches.

C_4) Except for project-specific constants, using specific **non-standard** packages and libraries should be limited to a strict **minimum** and with great care since this weakens the project's integrity, safety, portability and re-usability.

C_5) Overloading of standard operators for standard types is not allowed (like "+" for std_logic_vectors).

C_6) RTL: Use **exclusively std_logic** (and **std_logic_vector**) types in **ports**.
Avoid all other types like : (un)signed, integers, booleans, reals, multidimensional arrays, records, enumerations... This rule can **not** be bent on the top level !
(all types are indeed okay for BEH and SIM)

C_7) **Unconstrained arrays** can be used in ports (very elegant style) with care since this is an issue for unitary synthesis (always), and for some less capable (old) synthesis tools.

C_8) **Default values** can be used for Entity input ports (avoid in components).

C_9) RTL: **records** are sometimes possible and elegant for inter-entity uni-directional connectivity, but should be **avoided in top level ports**.

C_10) **Vectors directions**:

1. Use **descending** (**downto**) if the vector represents a **number** (or when in doubt...).
2. Use **ascending** (**to**) for the first dimension of a memory array, for example:
   `array (0 to 15) of std_logic_vector (7 downto 0)`
3. **Ascending** is also okay for a numbered collection of items, like LED array etc...
   `LEDS : out std_logic_vector (1 to 8);`

C_11) RTL: **Avoid hard values** and numeric constants, use **attributes** on objects or explicitly **declared constants** instead.

```
~~Y <= (X(7) xor Sign) & X(6 downto 0);~~
Y <= (X(X'high) xor Sign) & X(X'high-1 downto 0);
~~X <= X + 5;~~
constant X_incr : positive := 5; -- X varies by steps of 5 units
X <= X + X_incr;
```

C_12) RTL: **Sequential Logic** with asynchronous reset template:

```
-- <<< Document here what this process does >>>
process (Clk, Rst) -- ONLY Clk & async Rst allowed in the sensitivity list !
begin
  if Rst='1' then
    -- <<< ALL the regs must be initialized here ! >>>
  elsif rising_edge (Clk) then
    if Enable then -- Clock enable
      -- <<< Your code here >>>
    end if; -- do NOT insert code here !
  end if;   -- nor here !
end process;
```

C_13) RTL: In the process above, **every** signal assigned ("<=") inside the "rising_edge" block **must be initialized** in the "**if Rst**" block.

C_14) RTL: Inside "**if Rst**", only **constant values** can be assigned (no asynchronous load). Except in rare situations, registers powering up at '1' are not an issue.

C_15) RTL: use "**rising_edge**" exclusively (give up obsolete `clock'event and clk='1'`).

C_16) RTL: "**wait**" must **not** be used.

C_17) RTL: Avoid using attributes on *types*, always prefer **attributes on objects** (signals or

variables).

C_18) RTL: Do **not** use "**BUFFER**" mode in ports.
Use « **out** » mode + internal signals with proper naming convention (see N9) and suitable type. If different type, convert in the continuous assignment.
VHDL 2008 is another potential solution (when its use is tolerated).

C_19) RTL: Avoid using "**INOUT**" mode **except at the very top level**. In FPGA flows, it is usually tolerated to rely on "*tri-states bubble-up*", but internal multiple drivers are not allowed.

C_20) RTL: the **tri-state** and **bi-directional** Input/Outputs must be coded in the top level as:
`ExtBus <= BusOut when OE='1' else (others=>'Z');`
A register (signal assigned in a clocked process) should never receive 'Z'.

C_21) **Input ports** can be left **unconnected** (**open**) at instantiation provided they are assigned **default values** at declaration.

C_22) It is generally recommended to use **direct instantiation** which gets rid of the `component` declaration, but the instance can **no longer be configured** through a configuration statement.

C_23) Instantiations **must** use **named Port Maps** (vs Ordered port maps).
Generic maps can be ordered if the number of generics does not exceed two (2).

C_24) Do **not** leave ports **unconnected by omission**: use "`open`".

C_25) RTL: **Avoid recursive code !**
This can work with some tools, and it will likely improve over the years, but it's taking chances with tools and it may be hard to maintain and understand.

C_26) RTL: **Global signals and shared variables are not allowed** except in very specific and well-justified situations.
If needed for simulation purpose, exclude them by **synthesis pragmas** (but see below).
Note: some smart Synthesis tools allow both with some restrictions !

C_27) RTL: **beware of proprietary pragmas**.
If absolutely required, be sure to document their use and put a clear note in the header.
Their correct interpretation must then be verified.

C_28) RTL: inside an entity's synthesizable architecture, the authorized **types** are: `std_logic, std_logic_vector, signed, unsigned`..
The use of `integer range` and `Boolean` is possible but requires care and caution since these scalar types are implicitly initialized at creation and do not support 'X' et 'U'.
Their correct (hardware) initialization must therefore be verified by some other means.
Note that **Enumerated types** have the same behavior and must be treated with even greater care (encoding issues).

C_29) RTL: **forbidden** types = `integer, bit, std_ulogic, real, time`, ...
(though syntactically okay, sometimes accepted by synthesis tools, these types **must not be used for RTL descriptions**).

C_30) RTL: do **not** declare **user-defined types** except for enumerations and arrays.
If necessary, declare **subtypes** which retain compatibility with the original type.

C_31) RTL: **avoid** using VHDL 93 **rotate and shift operators**. Prefer slices & concatenations.

C_32) Do **NOT multiply a Signed/Unsigned by an Integer** (due to a numeric_std bug).

C_33) Normally it is recommended to **avoid graphical tools** (schematics & FSM).
In any case, keep and treat the (translated) **VHDL code** as the master document.

C_34) RTL: Use **as few variables as possible** in synthesizable code, and never when you may use signals instead. Use variables for their specific behavior (factoring, intermediate results, re-using the Flip-Flops *inputs* etc..).

C_35) BEH & SIM: Use **as many variables as possible** in Test Benches and Behavioral Models !

C_36) RTL: **Never create latches, combinational feedback or asynchronous sequential logic**, whether intended or unintended !

C_37) RTL: You **must not initialize signals at their declaration.**
You must not initialize variables at their declaration in processes.
It is perfectly acceptable to initialize variables at their declaration *inside functions*.

C_38) RTL: **Avoid using procedures in RTL code**. This is technically possible, but there is no real advantage for doing so, and the code may become more difficult to understand. Using **functions** in RTL is possible. Prefer functions declared in the architecture (local) rather than in packages (too far) or in the process (too local).

C_39) BEH & SIM: Definitely use procedures as much as possible.

C_40) RTL: Preferably code **combinational logic inside sequential processes.** (Code the combinational logic together with the FF that follows).

C_41) RTL: Avoid **combinational processes** if possible (see above). In case: **Sensitivity Lists must be complete and accurate** (no missing, no extra).

C_42) RTL: do **not** code processes using the **wait** statement (no sensitivity list).

C_43) BEH & SIM: Avoid processes with **sensitivity list**, use the « **wait** » style instead.

C_44) BEH & SIM: Inside the processes, make sure there is a **`wait`** or equivalent in **every** branch ! (else simulation will likely hang)

C_45) RTL: ***Ressource sharing***. Document (with appropriate comments) the operators that synthesis is expected to share. In case of doubt, remove the operator(s) and factor it (them) out « by hand » (using a concurrent assignment for example).

C_46) RTL: Use as few **proprietary** (vendor-specific) **hardware macro-functions** as possible. When possible, inference is usually preferable, but this depends on many design- vendors- and tools-specific factors.
Isolate and document unavoidable technology-specific code (memory inference / instantiation, primitives instantiations like PLLs, etc...).

C_47) RTL: **One single clock domain per entity** (except on top level and clock domain resynchronization or clock crossing modules indeed !).

C_48) Signals can not cross clock domains without proper resynchronization !
Clock domain crossing is a very complex topic. It *cannot be verified* so it must be CORRECT BY DESIGN, and should therefore be reserved to very experienced designers.

C_49) RTL: All **a**synchronous **input signals should be re-synchronized**, preferably at the top-level. **Do not insert logic between the I/O and the input Flip-Flop !**
You may decide to use two Flip-Flops to resynchronize and thus take care of metastability. In any case, you must beware of re-convergence issues between multiple signals. Synchronous I/Os must be handled specifically (timing constraints etc...).

C_50) RTL: All the **Entity's outputs should be registered** by default and properly documented otherwise -and checked since they can create combinational feedback loops through the hierarchy-.

C_51) RTL: At the **top level**, combinational outputs should not be allowed.
In general, all the device's outputs should be direct Flip-Flop outputs.
Avoid relying on « not gate push back ». Do not insert logic between the FF and the I/O.

C_52) RTL, BEH, SIM: Avoid active-low signals *inside* the design.
The internal logic should be active-high.

C_53) RTL: **Finite State Machines Coding Style** (a topic rich in urban legends, false ideas and misconceptions and misuse). By experience, we know FSMs are rarely coded correctly.
Use resynchronized Mealy style (aka "**one-process**"). Some essential precautions must be taken when designing FSMs but certainly not the use of "safe" implementation options available in some Synthesis tools (typically useless if not counter-productive).
Please contact ALSE (info@alse-fr.com) for our detailed FSM coding recommendations.

C_54) RTL: **Complex** entities or **technology-specific** *instances* must have a **behavioral model**, for simulation (or for faster simulation).

C_55) RTL, SIM, BEH: Place **documented ASSERTIONS** (VHDL, OVL, or PSL) where appropriate. Refer to **ABV methodology**.

C_56) SIM: At the end of the simulation, the simulator should stop due to **events starvation**, to avoid useless simulation runs (beyond stimulus range). The clock must therefore be stopped (as well as other stimuli not depending on clock).
In some complex situations, this may become too difficult and the simulation will have to be run (script) for a specific amount of time, or end using a *severity failure report*.

Design Flow

F_1) A **Design Documentation** must be created and maintained.
It must provide a description of : - the overall system, - the high-level functions performed by the FPGA, - the I/Os, - the hierarchical architecture, - a reference to the Board's schematics, - all the design entities, - the Unitary test benches, - the Top-level test benche(es), - the Verification methodology, - the Synthesis, P&R and bitstream generation, - the tests performed on the real system, - other useful topics.
It must also include the exact type and version (including Services Pack info) of all the Tools. On some key projects, it may be desirable to maintain the availability of all the tools used, throughout the entire life of the product.

F_2) Every entity must be **white-box tested** (unitary) with (at least) one Test bench.

F_3) Every entity must be **unitary synthesized** (very simple entities may violate this rule).

F_4) Test Benches must be **self-checking and regressionnable.**
They should avoid using not-portable run-time « simulator » commands (like `force`, `unforce`, etc...).

F_5) Test benches, auxiliary files (vectors, expected results, memory contents, behavioral models, etc...) should be **versioned, properly documented and archived**.

F_6) Every step involved in producing the final object (usually the bitstream) must be automated through **documented, versioned and archived scripts**.
The steps must be documented since the scripts are tool-specific and version-specific.

F_7) Use **Scripts** (command line or **Tcl** preferably) for **synthesis** and **simulation** tasks.
Use GUIs when investigating, or other during the early development phases.
A finalized design should never rely on any GUI.

-=oOo=-

Note: these rules are intended to be used by the delegates who followed our **Training Courses** (where all the proper concepts are taught and the rules are explained).

If you are interested by these Training Courses, or to request an Authorization to use this document for any other purpose than personal use, please contact us:

A.L.S.E
8 passage Barrault
75013 – PARIS – France
Tel +33 1 84 16 32 32
E-Mail : info@alse-fr.com