ALSE's
VHDL Design Rules & Coding Style

http://www.ALSE-FR.com

Introduction

These rules and coding style are the result of twelve years of HDL design and teaching experience, tens of complex ASIC & FPGA projects, and hundreds of thousands of lines of code.

I wanted to keep this list small and simple, yet covering as much as possible. I think that 95% of the errors that can be found in existing code could be avoided by just following these rules. However, it is probably useful to remind at this stage that :

- Following the rules is not sufficient *per se* : a good understanding of VHDL and Digital design is still required to create high quality and reliable hardware !
- The other way around is also true : some of these rules can be bent and a correct working design achieved (though I wouldn't recommend this without a strong VHDL expertise and design experience).
- 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 considerably reduce the design risks.

If you do not agree with some rules, or want to suggest adding others, please feel free to contact me : Bert Cuzeau. Technical Manager ALSE - info@alse-fr.com.

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.

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 **a unique entity** section. Multiple architectures are possible in the same file, be sure to put 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 : are very useful and recommended for simulation (case of multiple architectures), but we suggest to avoid them for synthesis if possible. See above.

O_6) RTL : A module (entity) should **not** contain several **different and independent functionalities**.

O_7) Use **Scripts** (command line or Tcl preferably) for **synthesis** and **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 Description, Specific issues, Speed and Area estimates if applicable, tools names and versions used, HDL standard followed, Revisions & ECOs.

P_2) The files must **not include any « hard tab »** (HT) but only **soft spaces**, and must be properly **aligned** and **indented**. In case of mess, use Emacs VHDL mode's beautifier !

P_3) Total **line length** should be less than 132 characters.

P_4) Only **one single executable statement per line**.
This rule is important for simulation and debug.

P_5) The VHDL code must include **significant and value-adding comments**, in **English**.
Every process or continuous assignment should be preceded by a comment summarizing its purpose.

P_6) It should be easy to **match** the **requirements** and the HDL **code**, **both ways**.

P_7) The comments and header information must be kept **accurate** and **up-to-date** throughout code changes and design iterations.

P_8) There should be **no piece of code commented out**.
Inactive or wrong code should be deleted. If needed, an older version of the architecture can be kept as a reference, for comparison or 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 with a text editor.

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

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.

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

N_8) Adopt a **unique notation for active low signals** (like **n**CS or CS**_n** for example, but not both).

N_9) Define **internal signals** with a **derived name** when you need to read back output Signals (for example OutBus**_i** for OutBus).

N_10) Use instance names derived from the entity names. For example :
```
Fir16x8_i Fir16x8 port map ( etc...
```

N_11) Do not use too long names for identifiers, especially for entities.
Try a maximum of 8 characters, and at least no more than 16, **identical with the file name**.

N_12) Name **Clocks** : CLKxxx, **Resets** : RSTxxx...

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

Coding Rules

C_1) Adopt **VHDL** 93 / **2001** standard (VHDL '87 was messy and is definitely obsolete).

C_2) RTL : Use **exclusively IEEE** libraries : **std_logic_1164** & **numeric_std**.
Do **NOT** use Synopsys' : **std_logic_arith, std_logic_(un)signed, numeric_bit**, and other horrendous and proprietary libraries...
Math_real should be avoided if possible in RTL code (but becomes more and more acceptable over the years).

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) 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 reusability.

C_5) RTL + BEH : define only **one single port per line** + **a comment for every port**.

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, SIM)

C_7) **Unconstrained arrays** can be used in ports (very elegant style), but this can cause trouble for unitary synthesis (always), and synthesis (not all synthesizers, at the time of writing, do support unconstrained array in ports).

C_8) **Default values** can be used for input ports.

C_9) RTL : **records** are possible and sometimes recommended for inter-entity 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 : 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 = :

```
-- <<< Explain here what this process does >>>
process (Clk, Rst) -- ONLY Clk & async Rst in the sensitivity list !
begin
  if Rst='1' then
    -- <<< all regs 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 "rising_edge" **must be initialized** in "**if Rst**".

C_14) RTL : Inside "**if Rst**", only **constant values** can be assigned (no asynchronous load). Furthermore, registers powering up at '0' are usually preferable.

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

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

C_17) RTL : Avoid using attributes on types, prefer the **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.

C_19) RTL : Do **not** use "**INOUT**" mode **except at the very top level**.
Define the tri-state drivers only at the top-level of the hierarchy, thus avoiding to rely on "*tri-states bubble-up*". Remember that internal tri-states are strictly forbidden (in most FPGA architectures) or non portable if allowed in the technology.

C_20) RTL : the **tri-state** and **bi-directional** Input/Outputs must be coded in the top level as :
```
ExtBus <= BusOut when Oen='1' else (others=>'Z');
BusIn <= ExtBus;
```

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

C_22) It is possible, maybe recommended to use **direct instanciation** which gets rid of the **component** declaration, but the instance can then **no longer be configured**.

C_23) Instanciation **must** use **named Port Maps**. Ordered port maps should not be allowed. **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 not be very easy to understand.

C_26) RTL : **Global signals and shared variables are not allowed**.
If needed for simulation purpose, exclude them by **synthesis pragmas** (but see below)

C_27) RTL : **beware of proprietary pragmas**. If absolutely required, be sure to document their use and put a note in the header.

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** 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 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 !
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) A **graphical (schematics) top-level**, automatically translated in structural VHDL can be a good idea. But always keep VHDL code as the master document.

C_33) 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 FlipFlops *inputs* etc..).

C_34) BEH & SIM : Use **as many variables as possible** in Test Benches and Behavioral Models !

C_35) RTL : **Never create latches, combinational feedback or asynchronous sequential logic**, whether intended or unintended !

C_36) RTL : You **must not initialize signals at their declaration.**
You must not initialize variables at their declaration in processes.
It is acceptable to initialize variables at their declaration inside functions.

C_37) 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. The trefer functions declared in the architecture (local) rather than in packages (too far) or in the process (too local).

C_38) BEH & SIM : Definitely use procedures as much as possible.

C_39) RTL : Preferably code **combinational logic inside sequential processes**.

C_40) RTL : Avoid **combinational processes** if possible (see above).
In case : **Sensitivity Lists must be complete and accurate** (no missing, no extra).

C_41) BEH & SIM : Avoid processes with **sensitivity list**, use the « **wait** » style instead.

C_42) BEH & SIM : Inside the processes, make sure there is a **`wait`** or equivalent in **every** branch !

C_43) RTL : ***Ressource sharing***. Document (with appropriate comments) the intended operators that synthesis is expected to share. In case of doubt, remove the operator(s) and factor it (them) out « by hand » (like in continuous assignments).

C_44) RTL : Use as few **proprietary** (vendor-specific) **macro-functions** as possible and properly isolate / document them. When possible, inference may be preferable, but this depends on many design- vendors- and tools-specific factors.

C_45) RTL : **One single clock domain per entity** (except on top level and clock domain-resynchronization- crossing entities indeed !).
Signals are not allowed to cross domains without proper resynchronization !

C_46) 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 FlipFlops to resynchronize and thus take care of possible metastability issue.
Synchronous I/Os must be handled specifically (timing constraints etc...).

C_47) RTL : All the **Entity's outputs should be registered**.
Combinational outputs can also create combinational feedbacks through the hierarchy.

C_48) 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 ».

C_49) RTL, BEH, SIM : Avoid active-low signals *inside* the design.
The internal logic should be active-high.

C_50) RTL : **Finite State Machines Coding Style**.
Please contact ALSE (info@alse-fr.com) for our recommendations and Application Note.

C_51) RTL : **Isolate unavoidable technology-specific code** (internal tri-states, memory inference / instanciation, primitives instanciations like PLLs, etc...

C_52) RTL : **Complex** entities or **technology-specific** *instances* must have a b**ehavioral model**, for simulation (or faster simulation).

C_53) RTL, SIM, BEH :
Place **documented assertions** (VHDL, OVL, or PSL) where appropriate.
Refer to **ABV methodology**.

C_54) 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).

Design Flow

F_1) Use **Scripts** (command line or **Tcl** preferably) for **synthesis** and **simulation** tasks.
Use GUIs when investigating, or other early phases.
A finalized design should never rely on any GUI.

F_2) Test Benches must be **self-checking and regressionnable.**
They should avoid using not-portable run-time « simulator » commands (like `force`, `unforce`, etc...).

F_3) Test benches, auxiliary files (vectors, expected results, memory contents, behavioral models, etc...) should be **versioned, properly documented and archived**.

F_4) 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_5) Every entity must be **white-box tested** (unitary) with (at least) one Test bench.

F_6) Every entity must be **unitary synthesized** (very simple entities may violate this rule).

F_7) The design must be simulatable at **post-layout timing level**.

F_8) The exact **Type** and **Version** (including Services Pack info) of all the **Tools** used must be properly documented. On some key projects, it may be desirable to maintain the availability of all the tools used throughout the life of the product.

-=oOo=-

Note : these rules are intended to be used by the delegates who followed our **VHDL Training Courses** (where all the proper concepts are taught and the rules are explained).
If you are interested by these Training Courses, please contact us at :

info@alse-fr.com

Advanced users may decide that some rules could be bent. Our experience suggests that this is possible, but also that it should be done in a very controlled and documented way. An example is the initialization of signals at their declaration, which is supported by a few synthesis tools and some FPGA families, but definitely not all (so we still recommend keeping our rule).