A Large Ion Collider Experiment

ALICE

CRU-HDL coding style and rules

Filippo Costa

ALICE O^2
CERN

A style guide is about consistency

Known when to be inconsistent: sometimes the style guide doesn’t apply … use your best judgment.

DO NOT BREAK COMPATIBILITY TO BE COMPLY WITH CODING STYLE

Don’t hesitate to ask !!!

TO BE CLEAR:

- ***Coding style*** : how the code should look like.
- ***Coding rules*** : rules related to a specific FPGA technology. In this specific case please refer to the **ALTERA QUARTUS HANDBOOK**:
  - *Recommended Design Practices.*
  - *Recommended HDL Coding Style.*

TABULATORS:

Don’t use the so called “**hard tabulators**”, because different editors can have different width assigned to them.
Use the so called “**soft tabulators**”, sequence of spaces (2 or 4).

EMACS hint
If you use emacs to write your code insert in the .emacs file the option

(setq-default indent-tabs-mode nil)

FILE HEADER:

Each file in your design should contain a standardized header.

```
 --==============================================================================
-- file reset_gen.vhd
--------------------------------------------------------------------------------
-- author Filippo Costa (filippo.costa at cern.ch)
--------------------------------------------------------------------------------
-- version 1.0
--
-- details
-- The component generates a reset signal in a TB
-- It is possible to define the reset active high or low
-- and the number of clocks it should be active
--
-- Last changes:
--  1.0 02-12-2015 PC first version
 --==============================================================================

library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

FILE HEADER (DOXYGEN VERSION):

Each file in your design should contain a standardized header.

```
--=============================================================================
--! @file reset_gen.vhd
-------------------------------------------------------------------------------
--! @author Filippo Costa (filippo.costa at cern.ch)
-------------------------------------------------------------------------------
--! @version 1.0\n
--!
--! @details
--! The component generates a reset signal in a TB\n
--! It is possible to define the reset active high or low\n
--! and the number of clocks it should be active\n
--!
--! \n<b>Last changes:</b>\n
--! 1.0 02-12-2015 PC first version \n
--=============================================================================

library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
```

COMMENTS:

Comments that contradict the code are worse than no comments.
Make a priority to update the comment when the code changes.
Each important operation and definition should have a comment above or beside it.

WRITE COMMENTS IN ENGLISH

Example:

```
architecture behavioral of reset_gen is
  signal s_rst : std_logic := '0';       --! internal signal for reset gen
  signal c_cnt : integer := RESET_CYCLE; --! counter for the clock cycle
begin

 --================================================================================
 --! Process used to decrease counter
 --================================================================================
  p_rst : process(clk_i)
  begin
    if rising_edge(clk_i) then
...
```

ENTITY COMMENTS:

Put above the entity declaration comments to describe the behavior of the code and the signals explanation.

Example:

```
--====================================================================================================
-- Entity declaration for double buffer
--
-- Description:
-- The component generates a reset signal in a TB
-- It is possible to define the reset active high or low
-- and the number of clocks it should be active
--
-- Signals Input :
-- clk_i     : input clock 250 MHz
-- reset_i : input reset active high
--
-- Signals Output :
-- data_o : data in output 32 bits
--====================================================================================================
```

COMMENTS:

inline comments can be distracting.

Do not state the obvious:

u_cnt <= u_cnt + 1; -- increase the counter by 1

But they can be useful

constant c_FW_DATE_Y : unsigned := to_unsigned(15, 4); -- year after 2000

NAMING STYLE:

Description	Extension	Example
variable	v_	v_variable
clock	clk_	clk_system_i
alias	a_	a_bit5
constant	c_UPPERCASE	c_CONSTANT
type definition	t_	t_mytype
generic	g_UPPERCASE	g_WIDTH
low active sig	_n	s_reset_n
unit input	_i	clk_i
unit output	_o	led_o
unit bi-dir	_b	data_b

NAMING STYLE:

Description	Extension	Example
internal signal	s_	s_reset
async signal	_a	s_state_a
counter	_c	s_cnt_c
pulse signal	_p	s_done_p

NAMING STYLE:

Description	Extension	Example
process	p_	p_register : process
block	b_	b_monitor : block
component	cmp_	cmp_reset_gen : reset_gen
loop	l_	l_sync_event : for
record	_rec	siu_bus_rec

LABELS:

Large blocks are easier to read if the specific label is used at the beginning (when declared) and at the closing end.

Example :

```
p_reg : process (clk)
begin
…
end process p_reg;
```

IMPORT:

Imports ("*use*" in VHDL) should be grouped in the following orders

1. standard lib.
2. related party lib.
3. user defined lib.

You should put a blank line between each group

Example :

```
library ieee;
use ieee.std_logic_1164.all;

library smartfusion2;
use smartfusion2.all;

library work;
use work.rcu2_constants_pkg.all;
use work.rcu2_addr_map_pkg.all;
```

WHITE SPACES AND INDENTATION:

- avoid ws immediately inside parentheses, brackets or braces.
  - NO: std_logic_vector (15 downto 0) => YES: std_logic_vector(15 downto 0)
  - NO: generic map(DATA => 3) => YES: generic map (DATA => 3)
- use ws in signal assignment:
  - NO: a<=b; => YES: a <= b;
- component instance:

```
cmp_pippo : entity work.pippo
  generic map (
             g_DATA => 3,
             g_CS      => 0
             )
  port map (
             clk_i     => s_clk,
             reset_i => s_rst,
        …..
             );
```

- avoid in general space around assignment to align it with another

```
NO:
s_a    <= s_b;
s_clk <= clk_i;
```

STATEMENTS and line:

avoid to put more than one statement per line.

```
signal s_clk, s_reset, s_flag : std_logic;
```

but

```
signal s_clk : std_logic
signal s_reset : std_logic;
signal s_flag : std_logic;
```

but ok

```
s_a <= ‘1’ when (s_flag = ‘1’) else ‘0’;
```

FILES:

File name should always start with the entity name of the corresponding unit.
(Develop some scripts to check consistency)

Example

reset_gen.vhd

```
entity reset_gen is
…
end entity reset_gen;
```

DIR STRUCTURE:

- **PRJ DIR/**
    - **script/** (tcl file called by prj.tcl to add the different files of the prj)
        - *gbt.tcl*
        - *device.tcl*
        - *custom_vhdl.tcl*
    - **sim/** (all the files needed to simulate components or projects)
        - **src/**
        - **script/**
    - **src/**
        - **constraints/** (sdc file, pinout, time ...)
        - **modules/** (usr logic modules)
            - *top.vhd*
            - **gbt/**
            - **pcie/**
            - **sfp/**
        - **packages/** (all the package definitions)
            - *gbt_package.vhd*
        - **qsys/** (all the QSYS files)
            - **counter_128/**
            - **ep_g3x8_avmm256/**
            - **gbt/**
            - **ttk/**
    - *prj.tcl* (tcl script to generate the full prj)

That Is All