

code written by your team members. Inefficient coding styles can adversely impact synthesis and simulation, which can result in slow circuits. Because portions of existing HDL designs are often used in new designs, you should follow coding standards that are understood by the majority of HDL designers. This chapter describes recommended coding styles that you should establish before you begin your designs.

Using Xilinx Naming Conventions

Use the Xilinx naming conventions listed in this section for naming signals, variables, and instances that are translated into nets, buses, and symbols.

- Avoid VHDL keywords (such as entity, architecture, signal, and component), even when coding in Verilog.
- Avoid Verilog keywords (such as module, reg, and wire), even when coding in VHDL. See Annex B of System Verilog Spec version 3.1a.
- A user-generated name should not contain a forward slash (/). The forward slash is generally used to denote a hierarchy separator.
- Names must contain at least one non-numeric character.
- Names must not contain a dollar sign (\$).
- Names must not use less-than (<) or greater-than signs (>). These signs are sometimes used to denote a bus index.
- The following FPGA resource names are reserved. They should not be used to name nets or components.
 - ◆ Device architecture names (such as CLB, IOB, PAD, and Slice)
 - ◆ Dedicated pin names (such as CLK and INIT)
 - ◆ GND and VCC
 - ◆ UNISIM primitive names such as BUFG, DCM, and RAMB16
 - ◆ Do not use pin names such as P1 or A4 for component names

For language-specific naming restrictions, see the language reference manual for Verilog or VHDL. Xilinx does not recommend using escape sequences for illegal characters. In addition, if you plan to import schematics, or to use mixed language synthesis or verification, use the most restrictive character set.

Naming Guidelines for Signals and Instances

Xilinx recommends that you follow the naming conventions set forth below in order to help achieve the goals of:

- Maximum line length
- Coherent and legible code
- Allowance for mixed VHDL and Verilog design
- Consistent HDL code

General

Xilinx recommends that you observe the following general rules:

- Do not use reserved words for signal or instance names.
- Do not exceed 16 characters for the length of signal and instance names, whenever possible.

- Create signal and instance names that reflect their connection or purpose.
- Do not use mixed case for any particular name or keyword. Use either all capitals, or all lowercase.

Recommendations for VHDL and Verilog Capitalization

Xilinx recommends that you observe the following guidelines when naming signals and instances in VHDL and Verilog.

Table 3-1: HDL and Verilog Capitalization

lower case	UPPER CASE	Mixed Case
library names	USER PORTS	Comments
keywords	INSTANCE NAMES	
module names	UNISIM COMPONENT NAMES	
entity names	PARAMETERS	
user component names	GENERICs	
internal signals		

Verilog is case sensitive, so module or instance names can be made unique by changing the capitalization of the letters in the name. However, for compatibility with file names, mixed language support, and other tools, Xilinx recommends that you rely on more than just capitalization to make instances unique.

Matching File Names to Entity and Module Names

Xilinx recommends the following practices in naming your HDL files.

- Make sure that the VHDL or Verilog source code file name matches the designated name of the entity (VHDL) or module (Verilog) specified in your design file. This is less confusing, and generally makes it easier to create a script file for the compilation of your design.
- If your design contains more than one entity or module, put each in a separate file with the appropriate file name. For VHDL designs, Xilinx recommends grouping the entity and the associated architecture into the same file.
- It is a good idea to use the same name as your top-level design file for your synthesis script file with either a .do, .scr, .script, or the appropriate default script file extension for your synthesis tool.

Naming Identifiers

Follow these naming practices to make design code easier to debug and reuse.

- Use concise but meaningful identifier names.
- Use meaningful names for wires, regs, signals, variables, types, and any identifier in the code (example: CONTROL_REGISTER).
- Use underscores to make the identifiers easier to read.

Guidelines for Instantiation of Sub-Modules

Follow these guidelines when instantiating sub-modules.

Xilinx recommends that you always use named association to prevent incorrect connections for the ports of instantiated components. Never combine positional and named association in the same statement as illustrated in the following examples.

Table 3-2: Incorrect and Correct VHDL and Verilog Coding Examples

	VHDL	Verilog
Incorrect	<pre>CLK_1: BUFG port map (I=>CLOCK_IN, CLOCK_OUT);</pre>	<pre>BUFG CLK_1 (.I(CLOCK_IN), CLOCK_OUT);</pre>
Correct	<pre>CLK_1: BUFG port map(I=>CLOCK_IN, O=>CLOCK_OUT);</pre>	<pre>BUFG CLK_1 (.I(CLOCK_IN), .O(CLOCK_OUT));</pre>

Xilinx also recommends using one port mapping per line to improve readability, provide space for a comment, and allow for easier modification.

VHDL Coding Example

```
-- FDCPE: Single Data Rate D Flip-Flop with Asynchronous Clear, Set
and
-- Clock Enable (posedge clk). All families.
-- Xilinx HDL Language Template

FDCPE_inst : FDCPE
generic map (
  INIT => '0') -- Initial value of register ('0' or '1')
port map (
  Q => Q,    -- Data output
  C => C,    -- Clock input
  CE => CE,  -- Clock enable input
  CLR => CLR, -- Asynchronous clear input
  D => D,    -- Data input
  PRE => PRE -- Asynchronous set input
);

-- End of FDCPE_inst instantiation
```

Verilog Coding Example

```
// FDCPE: Single Data Rate D Flip-Flop with Asynchronous Clear, Set and
// Clock Enable (posedge clk). All families.
// Xilinx HDL Language Template

FDCPE #(
  .INIT(1'b0) // Initial value of register (1'b0 or 1'b1)
) FDCPE_inst (
  .Q(Q),    // Data output
  .C(C),    // Clock input
  .CE(CE),  // Clock enable input
```

```

        .CLR(CLR), // Asynchronous clear input
        .D(D),    // Data input
        .PRE(PRE) // Asynchronous set input
    );

    // End of FDCPE_inst instantiation

```

Recommended Length of Line

Xilinx recommends that the length of a line of VHDL or Verilog code not exceed 80 characters. If a line needs to exceed this limit, break it with the continuation character, and align the subsequent lines with the preceding code. Choose signal and instance names carefully in order to not break this limit.

Try not to make too many nests in the code, such as nested **if** and **case** statements. If you have too many **if** statements inside of other **if** statements, it can make the line length too long, as well as inhibit optimization. By following this guideline, code is generally more readable and more portable, and can be more easily formatted for printing.

Using a Common File Header

Xilinx recommends that you use a common file header surrounded by comments at the beginning of each file. A common file header:

- Allows for better documentation of the design and code
- Improves code revision tracking
- Enhances reuse

The contents of the header depend on personal and company standards. Following is an example file header in VHDL.

```

-----
-- Copyright (c) 1996-2003 Xilinx, Inc.
-- All Rights Reserved
-----
--
--  _____
-- /  /\ /  /  Company: Xilinx
-- /___/ \ /  Design Name: MY_CPU
-- \ \ \ /  Filename: my_cpu.vhd
-- \ \   Version: 1.1.1
-- / /   Date Last Modified: Fri Sep 24 2004
-- /___/ /\  Date Created: Tue Sep 21 2004
-- \ \ / \
-- \___\/\___\
--
--Device: XC3S1000-5FG676
--Software Used: ISE 8.1i
--Libraries used: UNISIM
--Purpose: CPU design
--Reference:
-- CPU specification found at: http://www.mycpu.com/docs
--Revision History:
-- Rev 1.1.0 - First created, joe_engineer, Tue Sep 21 2004.
-- Rev 1.1.1 - Ran changed architecture name from CPU_FINAL
--            john_engineer, Fri Sep 24 2004.

```