# VHDL CodeCount™

# Counting Standard

*University of Southern California*

**Center for Systems and Software Engineering**

December , 2012

## Revision Sheet

| Date | Version | Revision Description | Author |
|---|---|---|---|
| 12/14/2012 | 1.0 | Original Release | CSSE |

# Table of Contents

# 1. Definitions

1.1. **SLOC –** Source Lines of Code is a unit used to measure the size of software program. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

1.2. **Physical SLOC –** One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

1.3. **Logical SLOC –** Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

1.4. **Data declaration line or data line –** A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program.

The following table lists the VHDL keywords that denote data declaration lines:

| | | | |
|---|---|---|---|
| type | assert | file | attribute |
| subtype | signal | constant | generic |
| variable | shared | alias | group |
| buffer | linkage | bus | literal |
| new | range | register | record |
| units | | | |

**Table 1 Data Declaration Types**

1.5. **Compiler Directives –** A statement that tells the compiler how to compile a program, but not what to compile.

The following table lists the VHDL directives:

| **Translation Directives** |
|---|
| -- pragma translate_off |
| -- pragma translate_on |
| -- synopsis translate_off |
| -- synopsis translate_on |

**Table 2 Compiler Directives**

1.6. **Blank Line –** A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

1.7. **Comment Line –** A comment is defined as a string of zero or more characters that follow language-specific comment delimiter.

VHDL comment delimiters are "--". A whole comment line may span one line and does not contain any compilable source code. An embedded comment can co-exist with compilable source code on the same physical line. Banners and empty comments are treated as types of comments.

1.8. **Executable Line of code –** A line that contains software instruction executed during runtime. Since VHDL is a declarative programming language, statements that are considered executable consist of everything other than compiler directives, comments and data declaration lines. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Sequential statements (if, loop, wait)
  - Concurrent statements (block, process, select, generate)
  - Empty statements (one or more ";")
  - Design unit statements (entity, architecture, configuration, library and use)
- An executable line of code may not contain the following statements:
  - Compiler directives
  - Data declaration (data) lines
  - Whole line comments, including empty comments and banners
  - Blank lines

# 2. Checklist for source statement counts

| PHYSICAL SLOC COUNTING RULES | | | |
|---|---|---|---|
| MEASUREMENT UNIT | ORDER OF PRECEDENCE | PHYSICAL SLOC | COMMENTS |
| **Executable Lines** | 1 | One per line | |
| **Non-executable Lines** | | | |
| Declaration (Data) lines | 2 | One per line | |
| Compiler Directives | 3 | Once per directive | |
| Comments | | Not Included (NI) | |
| One their own lines | 4 | NI | |
| Embedded | 5 | NI | |
| Banner | 6 | NI | |
| Empty Comments | 7 | NI | |
| Blank Lines | 8 | NI | |

| LOGICAL SLOC COUNTING RULES | | | | |
|---|---|---|---|---|
| NO. | STRUCTURE | ORDER OF PRECEDENCE | LOGICAL SLOC RULES | COMMENTS |
| R01 | Design units | 1 | Count once during definition | Declaration of a design unit should end with keyword “is” as the last word on a line |
| R02 | Concurrent statements | 2 | Count once | |
| R03 | Sequential Statements | 3 | Count once | |
| R04 | Statements ending by a semicolon | 4 | Count once per statement, including empty statement | |

# 3. Examples

## EXECUTABLE LINES

### SEQUENTIAL Statement

**ESS1 – wait, assert, report, next and null statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| [ label: ] **wait** [sensitivity clause] [condition clause] ; | wait until A>B and S1 or S2; | 1 |
| [ label: ] **assert** boolean_condition [ **report** string ] [ **severity** name ] ; | **assert** clk='1' **report** "clock not up" **severity**<br>WARNING; | 0<br>1 |
| [ label: ] **report** string [ **severity** name ] ; | **report** "Inconsistent data." **severity**<br>FAILURE; | 0<br>1 |
| [ label: ] target <= [ delay_mechanism ] waveform ; | sig4 <= **reject** 2 ns sig5 after 3 ns; | 1 |
| [ label: ] target := expression ; | Sig := Sa **and** Sb **or** Sc **nand** Sd **nor** Se **xor** Sf **xnor** Sg; | 1 |
| [ label: ] procedure-name [ ( actual parameters ) ] ; | compute(stuff, A=>a, B=>c+d); | 1 |
| [ label: ] **next** [ label2 ] [ **when** condition ] ; | **next when** A>B; | 1 |
| [ label: ] **null** ; | **null**; | 1 |

**ESS2 – if, else if, else and nested if statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| [ label: ] **if** condition1 **then** | **if** a=b **then** | 1 |
| sequence-of-statements | c:=a; | 1 |
| **elsif** condition2 **then** | **elsif** b<c **then** | 1 |
| sequence-of-statements | d:=b; | 1 |
| **end if** [ label ] ; | b:=c; | 1 |
| | **else** | 0 |
| | do_it; | 1 |
| | **end if**; | 0 |

**ESS3 – loop statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| [ label: ] **loop** | **loop** | 1 |
| sequence-of-statements | input_something; | 1 |
| **end loop** [ label ] ; | **exit when** end_file; | 1 |
| | **end loop**; | 0 |
| [ label: ] **for** variable **in** range **loop** | **for** I **in** 1 **to** 10 **loop** | 1 |
| sequence-of-statements | AA(I) := 0; | 1 |
| **end loop** [ label ] ; | **end loop**; | 0 |
| [ label: ] **while** condition **loop** | **while not** end_file **loop** | 1 |
| sequence-of-statements | input_something; | 1 |
| **end loop** [ label ] ; | **end loop**; | 0 |

## CONCURRENT Statement

**ECS1 – block and process statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| label : **block** [ ( guard expression ) ] [ **is** ] | maybe : **block** ( B'stable(5 ns) ) **is** | 1 |
| [ generic clause [ generic map aspect ; ] | **port** (A, B, C : **inout** std_logic ); | 1 |
| ] | **port map** ( A => S1, B => S2, C => outp ); | 1 |
| [ port clause [ port map aspect ; ] ] | **constant** delay: time := 2 ns; | 1 |
| [ block declarative items ] | **signal** temp: std_logic; | 1 |
| **begin** | **begin** | 0 |
| concurrent statements | temp <= A **xor** B **after** delay; | 1 |
| **end block** [ label ] ; | C <= temp **nor** B; | 1 |
| | **end block** maybe; | 0 |
| | printout: **process**(clk) | 1 |
| label : **process** [ ( sensitivity_list ) ] [ **is** ] | **variable** my_line : LINE; | 1 |
| [ process_declarative_items ] | **begin** | 0 |
| **begin** | **if** clk='1' **then** | 1 |
| sequential statements | write(my_line, string'("at clock ")); | 1 |
| **end process** [ label ] ; | write(my_line, counter); | 1 |
| | write(my_line, string'(" PC=")); | 1 |
| | write(my_line, IF_PC); | 1 |
| | writeline(output, my_line); | 1 |
| | counter <= counter+1; | 1 |
| | **end if**; | 0 |
| | **end process** printout; | 0 |

**ECS2 – when-else, with-select and port map statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| target <= waveform **when** choice **else** waveform; | sig2 <= not a_sig after 1 ns **when** ctl='1' **else** b_sig; | 1 |
| **with** expression **select** target <= | **with** count/2 **select** my_ctrl <= | 1 |
| waveform **when** choice [, waveform **when** choice ] ; | '1' **when** 1, | 1 |
| | '0' **when** 2, | 1 |
| | 'X' **when others**; | 1 |
| part_name: **entity** library_name.entity_name(architecture_name) | A101: **entity** WORK.gate(circuit) | 0 |
| **port map** ( actual arguments ) ; | **port map** ( in1 => a, in2 => b, out1 => c ); | 1 |
| part_name: component_name | PC_incr : add_32 **port map** (PC, four, zero, | 0 |
| **port map** ( actual arguments ) ; | PC_next, nc1); | 1 |

**ECS3 – generate statement**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| label: **for** variable **in** range **generate** | band : **for** I **in** 1 **to** 10 **generate** | 1 |
| block declarative items | b2 : **for** J **in** 1 **to** 11 **generate** | 1 |
| **begin** | b3 : **if abs**(I-J)<2 **generate** | 1 |
| concurrent statements | part: foo **port map** ( a(I), b(2*J-1), c(I, J) ); | 1 |
| **end generate** label ; | | |
| label: **if** condition **generate** | **end generate** b3; | 0 |
| block declarative items | **end generate** b2; | 0 |
| **begin** | **end generate** band; | 0 |
| concurrent statements | | |
| **end generate** label ; | | |

## DESIGN UNIT Statement

**EDUS1 – entity, architecture, configuration, package, procedure and function statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| **entity** identifier **is** | **entity** Latch **is** | 1 |
| **generic** ( generic_variable_declarations ) | **port** ( Din: **in** Word; | 0 |
| **port** (input_and_output_variable_declarations ) ; | Dout: **out** Word; | 0 |
| [ other declarations] | Load: **in** Bit; | 0 |
| **begin** | Clk: **in** Bit); | 1 |
| [ statements ] | **constant** Setup: Time := 12 ns; | 1 |
| **end entity** identifier ; | **constant** PulseWidth: Time := 50 ns; | 1 |
| | **use** WORK.TimingMonitors.**all;** | 1 |
| | **begin** | 0 |
| | **assert** Clk='1' **or** | 1 |
| | Clk'Delayed'Stable(PulseWidth); | 1 |
| | CheckTiming(Setup, Din, Load, Clk); | 1 |
| | **end entity** Latch; | 0 |

| | | |
|---|---|---|
| **architecture** identifier **of** entity_name **is** | **architecture** circuits **of** add4c **is** | 1 |
| [ declarations ] | **signal** c : std_logic_vector(3 **downto** 0); | 1 |
| **begin** | **component** fadd | 1 |
| [statements] | **port**(a : **in** std_logic; | 0 |
| **end architecture** identifier; | b : **in** std_logic; | 0 |
| | cin : **in** std_logic; | 0 |
| | s : **out** std_logic; | 0 |
| | cout : **out** std_logic); | 1 |
| | **end component** fadd; | 0 |
| | **begin** -- circuits of add4c | 0 |
| | a0: fadd **port map**(a(0), b(0), cin , sum(0), c(0)); | 1 |
| | a1: fadd **port map**(a(1), b(1), c(0), sum(1), c(1)); | 1 |
| | a2: fadd **port map**(a(2), b(2), c(1), sum(2), c(2)); | 1 |
| | a3: fadd **port map**(a(3), b(3), c(2), sum(3), c(3)); | 1 |
| | cout <= (a(3) **and** b(3)) **or** ((a(3) **or** b(3)) **and** | 1 |
| | ((a(2) **and** b(2)) **or** ((a(2) **or** b(2)) **and** | 0 |
| | ((a(1) **and** b(1)) **or** ((a(1) **or** b(1)) **and** | 0 |
| | ((a(0) **and** b(0)) **or** ((a(0) **or** b(0)) **and** | 0 |
| | cin))))))) | 0 |
| | **after** 1 ns; | 1 |
| | **end architecture** circuits; | 0 |
| **configuration** identifier **of** entity_name **is** | **configuration** add32_test_config **of** add32_test | 0 |
| [ declarations] | **is** | 1 |
| [ block configuration] | **for** circuits | 1 |
| **end architecture** identifier ; | **for all**: add32 | 1 |
| | **use entity** WORK.add32(circuits); | 1 |
| | **for** circuits | 1 |
| | **for all**: add4c | 1 |
| | **use entity** WORK.add4c(circuits); | 1 |
| | **for** circuits | 1 |
| | **for all**: fadd | 1 |
| | **use entity** WORK.fadd(circuits); | 1 |
| | **end for**; | 0 |
| | **end for**; | 0 |
| | **end for**; | 0 |
| | **end for**; | 0 |
| | **end for**; | 0 |
| | **end for**; | 0 |
| | **end configuration** add32_test_config; | 0 |
| **package** identifier **is** | **package** my_pkg **is** | 1 |
| [ declarations, see allowed list below ] | **type** small **is range** 0 **to** 4096; | 1 |
| **end package** identifier ; | **procedure** s_inc(A : **inout** small); | 1 |
| | **function** s_dec(B : small) **return** small; | 1 |
| | **end package** my_pkg; | 0 |

| | | |
|---|---|---|
| **package body** identifier **is**<br>[ declarations, see allowed list below ]<br>**end package body** identifier ; | **package body** my_pkg **is**<br>**procedure** s_inc(A : **inout** small) **is**<br>**begin**<br>A := A+1;<br>**end procedure** s_inc;<br>**function** s_dec(B : small) **return** small is<br>**begin**<br>**return** B-1;<br>**end function** s_dec;<br>**end package body** my_pkg; | 1<br>1<br>0<br>1<br>0<br>1<br>0<br>1<br>0<br>0 |
| **procedure** identifier [ ( formal parameter list ) ] ; | **procedure** build ( A : **in constant** integer;<br>B : **inout signal** bit_vector;<br>C : **out variable** real;<br>D : **file** ) ; | 0<br>0<br>0<br>1 |
| **procedure** identifier [ ( formal parameter list ) ] **is**<br>[ declarations, see allowed list below ]<br>**begin**<br>sequential statement(s)<br>**end procedure** identifier ; | **procedure** print_header **is**<br>**use** STD.textio.all;<br>**variable** my_line : line;<br>**begin**<br>write ( my_line, string'("A    B    C"));<br>writeline ( output, my_line );<br>**end procedure** print_header ; | 1<br>1<br>1<br>0<br>1<br>1<br>0 |
| **function** identifier [ (parameter list ) ]<br>**return** a_type **is**<br>[ declarations, see allowed list below ]<br>**begin**<br>sequential statement(s)<br>return some_value;<br>**end function** identifier ; | **function** random **return** float **is**<br>variable X : float;<br>**begin**<br>**return** X;<br>**end function** random ; | 1<br>1<br>0<br>1<br>0 |

**EDUS2 – library and use statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| **library** library_name ;<br>**use** library_name.unit_name.**all** ; | **library** ieee ;<br>**use** ieee.std_logic_1164.**all**;<br>**use** ieee.std_logic_textio.**all**;<br>**use** ieee.std_logic_arith.**all**;<br>**use** ieee.numeric_std.**all**;<br>**use** ieee.numeric_bit.**all**;<br>**use** WORK.my_pkg.s_inc; | 1<br>1<br>1<br>1<br>1<br>1<br>1 |

## Declaration or Data Lines

**DDS1 – type, subtype, variable, constant, file, shared variable, alias, attribute, disconnect and group statements**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| **type** <identifier>; | **type** node; | 1 |
| **type** <identifier> **is** <scalar_type_definition>; | **type** my_bits **is range** 31 downto 0; | 1 |
|  | **type** stuff **is** | 1 |
| **type** <identifier> **is** <composite_type_definition>; | **record** | 0 |
|  | I : integer; | 1 |
|  | X : real; | 1 |
|  | day : integer range 1 to 31; | 1 |
|  | name : string(1 to 48); | 1 |
|  | prob : matrix(1 to 3, 1 to 3); | 1 |
|  | **end record**; | 0 |
| **variable** <identifier> : <subtype_indication> [:=expression]; | **variable** item : node := root.all; | 1 |
| **subtype** <identifier> **is** <subtype_indication>; | **subtype** small_int **is** integer range 0 to 10; | 1 |
|  | **constant** N, N5 : integer := 5; | 1 |
| **constant** <identifier> : <subtype_indication> := <constant expression>; | **signal** my_word : word := X”01234567”; | 1 |
| **signal** <identifier> : <subtype_indication> [ signal_kind] [ :=expression]; | **shared variable** status : status_type := stop; | 1 |
| **shared variable** <identifier> : < subtype_indication> [:=expression]; | **file** my_file : text **open** write_mode **is** "file5.dat"; | 1 |
| **file** identifier : <subtype_indication> [ file_open_information ]; | **alias** mantissa:std_logic_vector(23 downto 0) **is** my_real(8 to 31); | 0<br>1 |
| **alias** <new_name> **is** <existing_name_of_same_type >; | **alias** "<" **is** my_compare [ my_type, my_type, **return** boolean ] ; | 0<br>1 |
| **alias** new_name [ : subtype_indication ] : **is** [ signature ]; | **attribute** enum_encoding **of** my_state : **type is** "001 010 011 100 111"; | 0<br>1 |
| **attribute** identifier : type_mark ; |  |  |
| **group** <identifier> **is** ( <entity_class_list> ) ; | **group** my_stuff **is** ( **label** <> ) ; | 1 |
| **disconnect** <signal_name> : type_mark **after** <time_expression> ; | **disconnect** my_sig : std_logic **after** 3 ns; | 1 |

**DSS2 – component statement**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| **component** <component_name> [**is**] | **component** reg32 **is** | 0 |
| [**generic** ( variable_declarations> ) ; ] | **generic** ( setup_time : time := 50 ps; | 0 |
| **port** ( | pulse_width : time := 100 ps ); | 1 |
| <input_and_output_variable_declarations> | **port** ( input : **in** std_logic_vector(31 downto 0); | 0 |
| ) ; | output: **out** std_logic_vector(31 downto 0); | 0 |
| **end component** <component_name> ; | Load : **in** std_logic_vector; | 0 |
|  | Clk : **in** std_logic_vector ); | 1 |
|  | **end component** reg32; | 0 |

## COMPILER DIRECTIVES

**CDP1 – pragma statement**

| GENERAL EXAMPLE | SPECIFIC EXAMPLE | SLOC COUNT |
|---|---|---|
| -- **pragma** <directive statement> | -- pragma translate_off | 1 |