VHDL CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2016

Revision Sheet

Date	Version	Revision Description	Author
7/19/2016	1.0	Original Release	Derek Lengenfelder
8/14/2016	1.1	Updated appendix and table contents	Matthew Swartz

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 keywords that 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 compliable source code. An embedded comment can co-exist with compliable 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 and on which a breakpoint can be set in a debugging tool. 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:
  - Selection statements (if, ? operator, switch)
  - Iteration statements (for, while, do-while)
  - Empty statements (one or more ";")
  - Jump statements (return, goto, break, continue, exit function)
  - Expression statements (function calls, assignment statements, operations, etc.)
  - Block statements
- 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	Defined in 1.8
Non-executable Lines			
Declaration (Data) Lines	2	One per line	Defined in 1.4
Compiler Directives	3	One per line	Defined in 1.5
Comments			Defined in 1.7
On their own lines	4	Not Included	
Embedded	5	Not Included	
Banners	6	Not included	
Empty comments	7	Not included	
Blank lines	8	Not Included	Defined in 1.6

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 Statements

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 WARNING;	0 1
[label:] report string [severity name] ;	report “Inconsistent data.” severity FAILURE;	0 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;
		0
		1

[label:] procedure-name [(actual parameters)] ;	compute(stuff, A=> a, B => c+d);	1
[label:] next [label] [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
    …statements…
else if condition2 then
    …statements…
else
    …statements…
end if [label] ;
	if a = b then
	    c := a;
	else if b < c then
	    d := b;
	    b := c;
	else
	    do_it;
	end if;
		1
		1
		1
		1
		1
		0
		1
		0

ITERATION Statements

EIS1 – loop Loops

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

[label:] loop
    …statements…
end loop [label] ;
	loop
	    input_something;
	    exit when end_file;
		1
		1
		1

	end loop;	0

EIS2 – for Loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

[label:] for variable in range loop
…statements…
end loop [label] ;
	for I in 1 to 10 loop
	AA(I) := 0;
	end loop;
		1
		1
		0

EIS3 – while Loop

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

[label:] while condition loop
..statements…
end loop [label] ;
	while not end_file loop
	input_something;
	end loop;
		1
		1
		0

CONCURRENT Statements

ECS1 – block statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

label : block [(guard expression)] [is]
[generic clause [generic map aspect ;]]
[port clause [port map aspect ;]]
[block declarative items]
begin
concurrent statements
end block [label] ;
	maybe : block (B'stable(5 ns)) is
	port (A, B, C : inout std_logic);
	port map (A => S1, B => S2, C => outp);
	constant delay: time := 2 ns
	signal temp: std_logic
	begin
	temp <= A xor B after delay;
	C <= temp nor B;
	end block maybe;
		1
		1
		1
		1
		1
		1
		0
		1
		1
		0

ECS2 – process statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

label : process [(sensitivity_list)] [is] [process_declarative_items]
begin
…sequential statements…
end process [label] ;
	printout: process(clk)
	variable my_ln : LINE;
	begin
	if clk='1' then
	write(my_ln, string'("at clk"));
	write(my_ln, counter);
	write(my_ln, string'(" PC="));
	write(my_ln, IF_PC);
	writeline(output, my_ln);
	counter <= counter+1;
	end if;
	end process printout;
		1
		1
		0
		1
		1
		1
		1
		1
		1
		1
		0
		0

ECS3 – generate statement

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

end generate label ;	end generate b3;	0
label: if condition generate	end generate b2;	0
…block declarative items…	end generate band;	0
begin		
…concurrent statements…		
end generate label ;		

ECS4 – when-else, with-select and port map statements

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

EXPRESSION Statements

EES1 – entity, architecture, configuration, package, procedure, function statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
entity identifier is	entity Latch is	0
generic (generic_variable_declarations)	port (Din: in Word;	0
port	Dout: out Word;	0
(input_and_output_variable_declarations) ;	Load: in Bit;	0
[other declarations]	Clk: in Bit);	1
begin	constant Setup: Time := 12 ns;	1
[statements]	constant PulseWidth: Time := 50 ns;	1
end entity identifier ;	use WORK.TimingMonitors.all;	1
	begin	0
architecture identifier of entity_name is	assert Clk='1' or Clk'Delayed'Stable(PulseWidth);	1
[declarations]	CheckTiming(Setup, Din, Load, Clk);	1
begin	end entity Latch;	1
[statements]		0
end architecture identifier;	architecture circuits of add4c is	0
	signal c : std_logic_vector(3 downto 0);	1
configuration identifier of entity_name is	component fadd	0
[declarations]	port(a : in std_logic;	0
[block configuration]	b : in std_logic;	0
end architecture identifier ;	cin : in std_logic;	0
	s : out std_logic;	0
package identifier is	cout : out std_logic);	1
[declarations, see allowed list below]	end component fadd;	1
end package identifier ;	begin -- circuits of add4c	0
	a0: fadd port map(a(0), b(0), cin , sum(0), c(0));	1
package body identifier is	a1: fadd port map(a(1), b(1), c(0), sum(1), c(1));	1
[declarations, see allowed list below]	a2: fadd port map(a(2), b(2), c(1), sum(2), c(2));	1
end package body identifier ;	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	0
procedure identifier [(formal parameter list)] ;	((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
procedure identifier [(formal parameter list)] is	((a(0) and b(0)) or ((a(0) or b(0)) and cin))))))	0

[declarations, see allowed list below]	after 1 ns;	1
begin	end architecture circuits;	1
…sequential statement(s) …		0
end procedure identifier ;	configuration add32_test_config of add32_test is	0
	for circuits	1
function identifier [(parameter list)]	for all: add32	1
return a_type is	use entity WORK.add32(circuits);	1
[declarations, see allowed list below]	for circuits	1
begin	for all: add4c	1
…sequential statement(s)…	use entity WORK.add4c(circuits);	1
return some_value;	for circuits	1
end function identifier ;	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;	1
		0
	package body my_pkg is	0
	procedure s_inc(A : inout small) is	0
	begin	0
	A := A+1;	1
	end procedure s_inc;	1
	function s_dec(B : small) return small is	0
	begin	0
	return B-1;	0
	end function s_dec;	1
	end package body my_pkg;	1
		1
	procedure print_header is	0
	use STD.textio.all;	1
	variable my_line : line;	1
	begin	0
	write (my_line, string'("A B C"));	1
	writeline (output, my_line);	1
	end procedure print_header ;	1
		0
	function random return float is	0
	variable X : float;	1
	begin	0
	return X;	1
	end function random ;	1

EES2 – library and use statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
	library ieee ;	1
	use ieee.std_logic_1164.all;	1
library library_name ;	use ieee.std_logic_textio.all;	1
use library_name.unit_name.all;	use ieee.std_logic_arith.all;	1
	use ieee.numeric_std.all;	1
	use ieee.numeric_bit.all;	1
	use WORK.my_pkg.s_inc;	1

DECLARATION OR DATA LINES

DDL1 – type, subtype, variable, constant, file, shared variable, alias, attribute, disconnect and group statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
type <indentifier>;	type node;	1
		0
type <identifier> is	type my_bits is range 31 downto 0;	1
<scalar_type_definition>;		0
	type stuff is	0

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

DDL2 – component type

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

COMPILER DIRECTIVES

CDL1 – pragma type

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

4. Complexity

Complexity measures the occurrences of different keywords in code baseline. Below table identifies the categories and their respective keywords that are counted as part of the complexity metrics.

CALCULATIONS	CONDITIONALS	LOGIC	PRE-PROCESSOR	ASSIGNMENT
+	IF	=	--PRAGMA	:=
-	ELSE IF	<	--SYNOPSIS	
*	ELSE	>		
/	FOR	/=		
MOD	LOOP	>=		
ABS	WHILE	<=		
**		&		
		AND		
		OR		
		XOR		
		IN		
		NAND		
		XNOR		
		SLA		