VHDL CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

August , 2012

Revision Sheet

Date	Version	Revision Description	Author
July 23 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#, Verilog, VHDL) 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.

There are two major kinds of objects used to hold data. They are SIGNAL and VARIABLE. These represent the actual data that is present in hardware. The data types below help in describing the type of data is represented by the signal or variable.

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

bit	bit_vector	integer
boolean	Real	character
std_logic	std_ulogic	string
std_logic_vector	std_logic_vector	time

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

PRAGMA (compiler directive for VHDL) allow to give additional information to the VHDL compiler. Although, it has no influence on the code, it changes the behavior.

1.6.

Translation Stop / Start	Resolution Function	Component Implication
-- pragma translate_off	-- pragma resolution_method wired_and	-- pragma map_to_entity entity_name
-- pragma translate_on	-- pragma resolution_method wired_or	-- pragma return_port_name port_name
-- pragma synthesis_off	-- pragma resolution_method three_state	
-- pragma synthesis_on		

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 are specified by the delimiter, **--** (two dashes), at the beginning of the line. There is no facility for a block comment. If more than one line has to be commented, each line must have the delimiter.

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. An instruction can be stated in a simple or compound form.

1.9. **An executable line of code may contain the following program control statements**

- Selection statements (if, conditional 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

2.0 Checklist for source statement counts

PHYSICAL SLOC COUNTING RULES			
MEASUREMENT UNIT	ORDER OF PRECEDENCE	PHYSICAL SLOC	COMMENTS
Executable Lines	1	One per line	Refer to examples
Non-executable Lines			
Data Declaration	2	One per line	Refer to Data Declaration Section
Library Clause	3	One per line	Refer Definitions Section
Use Clause	4	One per line	none
Compiler Directive	5	One per line	Refer Definitions Section
Comments	6	Not Included	Refer Definitions Section
Blank Lines	7	Not Included	Refer Definitions Section

LOGICAL SLOC COUNTING RULES				
NO.	STRUCTURE	ORDER OF PRECEDENCE	LOGICAL SLOC RULES	COMMENTS
1	Statements ending with semi-colon	1	Count once per statement, including empty statement	none
2	Block statements Begin and end	2	Count the beginning of every block, do not count end	Refer to examples for more details
3	Declaration statements	3	Count the beginning, independent statements ending with semicolon within declaration, do not count end	Refer to types and subtypes, records etc
4	Defining a block in Design units	4	Do not count beginning brace, count ending brace with semicolon [ );] Same rules apply for begin & end	Refer to Design Unit examples

3.0 Examples of logical SLOC counting

SEQUENTIAL STATEMENTS

IF Statement

ESS1 – IF, ELSE, ELSEIF STATEMENTS

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

if <condition> **then**
expression;
end if;
	if reset = '1' then
	value <= 0;
	end if;
		1
		1
		0

if <condition 1> **then**
expression 1;
else
expression 2;
end if;
	if reset = '1' then
	value <= 0;
	else
	value <= input;
	end if;
		1
		1
		0
		1
		0

if <condition 1> **then**
expression 1;
elsif <condition 2> **then**
expression 2;
else
expression 3;
end if;
 	
NOTE: expression can be a group of statements enclosed between "begin" and "end" statements. They are not to be counted. (to be treated as "{" and "}" in c)
	if reset = '1' then
	value <= 0;
	elsif enable = '1' then
	value <= input;
	else
	value <= previous;
	end if;
		1
		1
		1
		1
		0
		1
		0

CASE Statement

ESS2- CASE STATEMENTS

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
	case X is	1
case <expression> **is**	when 0 =>	0
when <choice 1> **=>**	out <= A;	1
statements;	when 1 to 5 =>	0
when <choice 2> **=>**	out <= B;	1
statements;	when others =>	0
when others =>	out <= C;	1
statements;	end case;	0
end case;		

WAIT Statement

ESS3 – WAIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
wait [**on** sensitivity-list] [**until** *boolean*-expression] [**for** *time*-expression] ';'	**WAIT ON** s; -- Wait for value changes on s **WAIT ON** s **UNTIL** s = '1'; -- Wait for a rising edge on s **WAIT UNTIL** s = '1'; -- Wait for a rising edge on s **WAIT**; -- Never passed **WAIT FOR** 10 ns; -- Pass WAIT after 10 ns	1 0 1 0 1 0 1 0 1 0

LOOP, NEXT, EXIT Statement

ESS4 – LOOP,NEXT,EXIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
[*loop*-label ':'] [**while** *boolean*-expression | **for** identifier **in** discrete-range] **loop** { sequential statement } **end loop** [*loop*-label] ';' exit-statement –› [label ':'] **exit** [*loop*-label] [**when** *boolean*-expression] ';'	L1:**for** i **in** 0 to 9 **loop** L2: **for** j **in** opcodes **loop** **for** k **in** 4 downto 2 **loop** **if** k=l **next** =L2; **end loop;** **end loop;**	1 1 1 1 0 0

FUNCTION Statement

ESS5 – Function Declaration & Implementation

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
function (identifier | operator) ['(' interface-list ')'] **return** *type*-name function-body –› function-specification **is** { subprogram-declarative-item } **begin** { sequential-statement } **end** [**function**] [*function*-identifier | operator] ';'	-- declaration **FUNCTION** AnyZeros(**CONSTANT** v : **IN** BIT_VECTOR) **RETURN** BOOLEAN; -- implementation **FUNCTION** AnyZeros(**CONSTANT** v : **IN** BIT_VECTOR) **RETURN** BOOLEAN **IS** **BEGIN** **……** **END FUNCTION** AnyZeros;	1 1 1 … 0

PROCEDURE Statement

ESS6- PROCEDURE

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
procedure identifier ['(' interface-list ')'] **is** { subprogram-declarative-item } **begin** { sequential-statement } **end** [**procedure**] [*procedure*-identifier] ';' (NOTE :interface-list –› **[constant** | **signal** | **variable** | **file]** identifier { ',' identifier } ':')	-- declaration **PROCEDURE** AnyZeros(**CONSTANT** inArray : **IN** BIT_VECTOR; **VARIABLE** result : **OUT** BOOLEAN); -- implementation **PROCEDURE** AnyZeros(**CONSTANT** inArray : **IN** BIT_VECTOR; **VARIABLE** result : **OUT** BOOLEAN) **IS** **BEGIN** …… **END PROCEDURE** Finish;	1 1 1 0 1 1 1 1 … 0

RETURN Statement

ESS7- RETURN

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
[label ':'] **return** [expression] ';'	**RETURN** FALSE; -- Return the value FALSE	1

Variable Assignment Statement

ESS8- Variable Assignment

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
[label ':'] (*variable*-name | *variable*-aggregate) ':=' expression ';'	Variable1 := a **or** b **or** c;	1

Signal Assignment Statement

ESS9 – Signal Assignment

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
[label ':'] (*signal*-name | *signal*-aggregate) '<=' [delay-mechanism] waveform ';'	Signal1 <= a **or** b **or** c;	1

CONCURRENT STATEMENTS

PROCESS Statement

ECS1 – PROCESS

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
process ['(' sensitivity-list ')'] [**is**] {…} **begin** { sequential-statement } **end** [**postponed**] **process** [*process*-label] ';'	**PROCESS** **BEGIN** **…..** **END PROCESS**;	1 1 …. 0

WHEN Statement

ECS2 – WHEN statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
[label ':'] [**postponed**] (*signal*-name | *signal*-aggregate) '<=' [VALUE] **when** *boolean*-expression **else** **…**	Two: s <= '1' **WHEN** sel = "00" **ELSE** **UNAFFECTED WHEN** sel ="11" **ELSE** '0';	1 0 1 0 1

SELECT Statement

ECS3 – SELECT statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
[label ':'] **with** expression **select** (*signal*-name | *signal*-aggregate) '<=' [**guarded**] [delay-mechanism] { waveform **when** choices ',' } waveform **when** choice { | choice } ';'	Choose: **WITH** sel **SELECT** s <= '1' **WHEN** "00", '0' **WHEN** "01";	1 1 1

BLOCK Statement

ECS4 – BLOCK Statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

block-label ':' **block** ['(' *guard*-expression ')'] [**is**]
begin
{
….(concurrent statements)
}
End block block-label**;**
	Block1: **BLOCK**(en = '1')
	BEGIN
	q <= **GUARDED** d **AFTER** t;
	END BLOCK Block1;
		1
		1
		1
		0

DECLARATION OR DATA LINES

OBJECTS DECLARATION Statement

EDS1- OBJECTS

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
Constant constant_name : expression; **Variable** variable_name : value; **Signal** (signal_name, signal_vector_name) : expression;	**Constant** alpha: character := 'a'; **Variable** sum : 0; **Signal** data_bus :bit_vector (0 to 7);	1 1 1

TYPES DECLARATION Statement

ESD2 – TYPES

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
type identifier **is** type-indication ';'	**Type** opcodes **is** (load,store,execute,crash); --(enumeration) **Type** small_int **is range** 0 to 100; **Type** glob **is record** First : interger; Second : big_bus; **End record;**	1 1 1 1 1 0

SUBTYPES DECLARATION Statement

ESD3 – SUBTYPES

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
subtype identifier **is** subtype-indication ';'	**Subtype** shorter **is** integer **range** 0 **to** 7;	1

ARRAYS & RECORDS DECLARATION Statement

ESD4- Declare Arrays & Records

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
--ARRAY **type** identifier **is array** '(' *type*-name **range** '<>' { ',' *type*-name **range** '<>' } ')' **of** *element*-subtype-indiciation ';'	--ARRAY **TYPE** ArrayType **IS ARRAY**(4 **DOWNTO** 0) **OF** BIT;	1
--RECORD **type** identifier **is record** element-declaration { …. } **end record** [*record-type-name*-identifier] ';'	**TYPE** Clock **IS RECORD** { Hour : INTEGER **RANGE** 0 **TO** 23; Min : INTEGER **RANGE** 0 **TO** 59; Sec : INTEGER **RANGE** 0 **TO** 59; **}** **END RECORD** Clock;	1 0 1 1 1 0 0

DESIGN UNITS

ENTITY Unit

EDU1 – ENTITY UNIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
entity <entity name> **is** **port** (<port mappings>); **end entity** <entity name>; NOTE: The entity keyword after end can be omitted	entity adder is port (a : in bit; b : in bit; s : out bit; c : out bit); end entity adder;	1 1 0 1 1 1 1 1 0

ARCHITECTURE Unit

EDU2- ARCHITECTURE UNIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

architecture <architecture name> **of**
<entity name> **is**
begin
statements;
.
.
.
end
end architecture <architecture name>;
 	
NOTE: The architecture keyword after end can be omitted
	architecture half_adder of adder is
	begin
	s <= a xor b;
	c <= a and b;
	end
	end architecture half_adder;
		1
		1
		1
		1
		0
		0

COMPONENT Unit

EDU3- COMPONENT UNIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
component <component name> **is** **port** (<port mappings>); **end component** <component name>; NOTE: The component keyword after end can be omitted	component flip_flop is port (d : in bit; q : out bit; clk : in bit;); end component flip_flop;	1 1 0 1 1 1 1 0

CONFIGURATION Unit

EDU4- CONFIGURATION UNIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

configuration <config name> **of** <entity name> **is**
for <architecture name>
<use clause>
<port mappings>
end for
end configuration <config name>
 	
NOTE: The component keyword after end can be omitted
	configuration config_adder of adder is
	for full_adder
	use entity work.adder(half_adder);
	end for;
	end config_adder;
		1
		
		1
		0
		0

PACKAGE Unit

EDU5- PACKAGE UNIT

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
package <package name> **is** <package declarations> **end package** <package name>	package basic_gate is component and_gate is port (a : in bit; b : in bit; c : out bit;); end component and_gate; end package basic_gate;	1 1 1 0 1 1 1 1 0 0

package body <package name> **is**
<package body definitions>
end package body <package name>
	package body basic_gate is
	entity and2 is
	port
	(
	in1, in2: in std_logic;
	out1 : out std_logic
	);
	end and2;
	architecture RTL of and2 is
	begin
	out1 <= in1 and in2;
	end RTL;
	end package body basic_gate;
		1
		1
		1
		0
		1
		1
		1
		0
		1
		0
		1
		0
		0

NOTE: The package and package body keyword after end can be omitted		

3.1 Notes on Special Character Processing

Reserved words in VHDL				
abs	disconnect	is	out	sli
access	downto	label	package	sra
after	else	library	port	srl
alias	elsif	linkage	postponed	subtype
all	end	literal	procedure	then
and	entity	loop	process	to
architecture	exit	map	pure	transport
array	file	mod	range	type
assert	for	nand	record	unaffected
attribute	function	new	register	units
begin	generate	next	reject	until
block	generic	nor	return	use
body	group	not	rol	variable
buffer	guarded	null	ror	wait
bus	if	of	select	when
case	impure	on	severity	while
component	in	open	signal	with
configuration	inertial	or	shared	xnor
constant	inout	others	sla	xor

VHDL OPERATOR	FUNCTIONALITY
**	exponentiation
abs	absolute value
not	complement
*	multiplication
/	division
mod	modulo
rem	remainder
+	unary plus
-	unary minus
+	addition
-	subtraction
&	concatenation
sll	shift left logical
srl	shift right logical
sla	shift left arithmetic
sra	shift right arithmetic
rol	rotate left
ror	rotate right

OPERATOR	FUNCTIONALITY
=	test for equality
/=	test for inequality
<	test for less than
<=	test for less than or equal
>	test for greater than
>=	test for greater than or equal
and	logical and
or	logical or
nand	logical complement of and
nor	logical complement of or
xor	logical exclusive or
xnor	logical complement of exclusive or