Verilog CodeCount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

January , 2013

Revision Sheet

Date	Version	Revision Description	Author
1/7/2013	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 Verilog keywords that denote data declaration lines:

Simple Data Types	Net Data Types	Register DataTypes	Input Data Types
event	supply0	reg	input
function	supply1		output
integer	tri		inout
module	tri0		
parameter	tri1		
real	triand		
realtime	trior		
task	trireg		
time	wand		
	wire		
	wor		

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 Verilog directives:

`define	`include	`ifdef	`else
`endif	`timescale		

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.

Verilog comment delimiters are "//" and "/*". 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 and on which a breakpoint can be set in a debugging tool. 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	
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	Module/ Function/Task Declarations	1	Count once during definition	Declare then assignment statements are counted as declaration statements
R02	Assignment Statements	2	Count once	
R03	Block Statements	3	Count once	
R04	Statements ending by a semicolon	4	Count once per statement, including empty statement	
R05	Compiler Directive	5	Count once per directive	

3. Examples

EXECUTABLE LINES

ASSIGNMENT Statements

EAS1 – assign statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
assign wire_variable = expression;	**assign** b = c&d;	1

BLOCK Statements

EBS1 – always statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
always @(event_1 **or** event_2 **or** ...)	**always** @(posedge c)	1
begin	**begin**	0
... statements ...	a <= b;	1
end	b <= a;	1
	end	0

EBS2 – initial statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
initial	**initial**	0
begin	**begin**	0
... statements ...	clr = 0; *// variables initialized at*	1
end	clk = 1; *// beginning of the simulation*	1
	end	0

EBS3 – if…else statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
if (expression)	**if** (alu_func == 2'b00)	1
begin	aluout = a + b;	1
... statements ...	**else if** (alu_func == 2'b01)	1
end	aluout = a - b;	1
else if (expression)	**else if** (alu_func == 2'b10)	1
begin	aluout = a & b;	1
... statements ...	**else** *// alu_func == 2'b11*	0
end	aluout = a | b;	1
...more else if blocks ...	**if** (a == b) *// This if with no else will generate*	1
else	**begin** *// a latch for x and ot. This is so they*	0
begin	x = 1; *// will hold their old value if (a != b).*	1
... statements ...	ot = 4'b1111;	1
end	**end**	0

EBS4 – case statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
case (expression)	**case** (state)	1
case_choice1:	state0: **begin**	1
begin	**if** (start) nxt_st = state1;	2
... statements ...	**else** nxt_st = state0;	1
end	**end**	0
case_choice2:	state1: **begin**	1
begin	nxt_st = state2;	1
... statements ...	**end**	0
end	state2: **begin**	1
... more case choices blocks ...	**if** (skip3) nxt_st = state0;	1
default:	**else** nxt_st = state3;	1
begin	**end**	0
... statements ...	state3: **begin**	1
end	**if** (wait3) nxt_st = state3;	1
endcase	**else** nxt_st = state0;	1
	end	0
	default: nxt_st = state0;	1
	endcase	0

EBS5 – while statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
while (*expression*)	**while** (!overflow) **begin**	1
begin	@(**posedge** clk);	1
... statements ...	a = a + 1;	1
end	**end**	0

EBS6 – repeat statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
always @(event_1 **or** event_2 **or** ...)	**always** @(posedge c)	1
begin	**begin**	0
... statements ...	a <= b;	1
end	b <= a;	1
	end	0

EBS7 – for statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
always @(event_1 **or** event_2 **or** ...)	**always** @(posedge c)	1
begin	**begin**	0
... statements ...	a <= b;	1
end	b <= a;	1
	end	0

EBS8 – forever statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

always @(event_1 **or** event_2 **or** ...)
begin
... statements ...
end
	always @(posedge c)
	begin
	a <= b;
	b <= a;
	end
		1
		0
		1
		1
		0

DECLARATION OR DATA LINES

DDS1 – wire and supply statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

wire [msb:lsb] wire_variable_list;	**wire** c; // *simple wire*
	wand d;
	assign d = a; // *value of d is the logical AND of*
	assign d = b; // *a and b*
	wire [9:0] A; // *a cable (vector) of 10 wires*.
		1
		1
		1
		1
		1

supply0 logic_0_wires;
supply1 logic_1_wires;
	supply0 my_gnd; // *equivalent to a wire assigned 0*
	supply1 a, b;
		1
		1

DDS2 – reg statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

reg [msb:lsb] reg_variable_list;	**reg** a; // *single 1-bit register variable*
	reg [7:0] tom; // *an 8-bit vector; a bank of 8 registers.*
	reg [5:0] b, c; // *two 6-bit variables*
		1
		1
		1

DDS3 – input/output statements

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

DDS4 – simple data types 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

DDS5 – module 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

DDS6 – function 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

DDS7 – task 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

CDL1 – directive types

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
`timescale time_unit / time_precision	**`timescale** 1 ns /100 ps	1
	// time unit = 1ns; precision = 1/10ns;	0
`define macro_name text_string	**`define** add_lsb a[7:0] + b[7:0]	1
`include file_name	**`include** “dclr.v”	1
`ifdef macro	**`ifdef** FIRST	1
…statements…	**$display**("First code is compiled");	1
`else	**`else**	1
…statements…	**`ifdef** SECOND	1
`endif	**$display**("Second code is compiled");	1
	`else	1
	$display("Default code is compiled");	1
	`endif	1
	`endif	1