Verilog Code Count™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

December , 2016

Revision Sheet

Date	Version	Revision Description	Author
7/15/2016	1.0	Original Release	Derek Lengenfelder
8/13/2016	1.1	Updated tables and appendix	Matthew Swartz
Date	Version No.	Click here to Description.	Name

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:

event	function	genvar	inout	input	integer
localparam	output	parameter	reg	specparam	supply0
supply1	task	time	tri	tri0	tri1
triand	trior	trireg	wand	wire	wor
bit	byte	shortint	int	longint	logic

Table 1 Verilog Data Keywords

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 keywords that denote data declaration lines:

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

Table 2 Verilog Compiler Directive

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	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 (NI)	
Embedded	5	NI	
Banners	6	NI	
Empty comments	7	NI	
Blank lines	8	NI	Defined in 1.6

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 statements	
R05	Compiler Directive	5	Count once per directive	

3. Examples

EXECUTABLE LINES

ASSIGNMENT Statement

EAS1 - assignment statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

assign wire_variable = expression;	module and(a,b,y);
	
	input a,b;
	
	output y;
	
	asssign y = a & b;
	
	endmodule
		1
		0
		1
		0
		1
		0
		1
		0
		0

BLOCK Statement

EBS1 – always statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

always @(event_1 or event_2 or …)
begin
…statements…
end
	module ao2_gate(
	input a, b, c, d,
	output logic y);
	
	logic tmp1, tmp2;
	
	always @(a,b,c,d) begin
	tmp1 = a & b;
	tmp2 = c & d;
	y = tmp1 | tmp2;
	end
	endmodule
		0
		0
		1
		0
		1
		0
		0
		1
		1
		1
		0
		0

EBS2 – initial statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

initial
begin
…statements…
end
	module initial_example();
	  reg clk,reset,enable,data;
	
	  initial begin
	  clk = 0;
	  reset = 0;
	  enable = 0;
	  data = 0;
	  end
	endmodule
		1
		1
		0
		0
		1
		1
		1
		1
		0
		0

EBS3 – if…else statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

if (<expression>)
begin
…statements…
end
else if (<expression>)
begin
…statements…
end
…more else if blocks…
else
begin
…statements…
end
	if (alu_func == 2'b00)
	  aluout = a + b;
	else if (alu_func == 2'b01)
	  aluout = a - b;
	else if (alu_func == 2'b10)
	  aluout = a &b;
	else
	  aluout = a | b;
	
	if (a == b)
	begin
	  x = 1;
	  ot = 4'b1111;
	end
		1
		1
		1
		1
		1
		1
		0
		1
		0
		1
		0
		1
		1
		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 nst_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, forever statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
repeat (<range>) begin	repeat (8) begin	1
<statements>	#1 temp = data[15];	1
end	data = data << 1;	1
	data[0] = temp;	1
forever (<range>) begin	end	0
<statements>		0
end	forever begin	1
	#5 clk = ! clk;	1
	end	0
EBS7 – for statement		
GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
for (<range>) begin	for (i=0; i<256; i=i+1)	1
<statements>	begin	0
end	$display("%g %h",i,ram[i]);	1
	ram[i] <= 0;	1
	$display("%g %h",i,ram[i]);	1

	end	0

DECLARATION OR DATA LINES

DDS1 – wire and supply statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

wire [msb:lsb] wire_variable_list;
 	
supply0 logic_0_wires;
supply1 logic_1_wires;
	wire c; // simple wire
	wand d;
	assign d = a; // value of d is logical
	assign d = b; // AND of a and b
	// a cable (vector) of 10 wires
	wire [9:0] A;
	
	// equiv. to wire assigned 0
	supply0 my_gnd;
	supply1 a, b;
		1
		1
		1
		1
		0
		1
		0
		0
		1
		1

DDS2 – reg statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

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

DDS3 – input/output statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT

component <component_name>
[is]
[generic (variable_declarations);]
port
(<input_and_output_var_decl.'s>);
end component
<component_name>;
	component reg32 is
	generic (setup_time : time := 50
	ps; pulse_width : time := 100 ps);
	port (input : in
	std_logic_vector(32 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
		0
		0
		0
		1
		0

DDS4 – simple data types, module, function, task statement

GENERAL EXAMPLE	SPECIFIC EXAMPLE	SLOC COUNT
<data_type> <variable_name>;	module data_types();	1

	bit data_1bit; byte data_8bit; shortint data_16bit; int data_32bit; longint data_64bit; integer data_integer; endmodule	1 1 1 1 1 1 0 0

COMPILER DIRECTIVES

CDL1 – directive type

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

Table 3 - Complexity Keywords List

Calculations	Conditionals	Logic	Pre-processor	Assignment
+	case	!	`define	=
-	if	~	`include	=>
*	else if	&	`ifdef	
/	else	|	`else	
%	for	^	`endif	
	forever	~&	`timescale	
	repeat	~|		
	while	~^		
		<<		
		>>		
		<		
		>		
		>=		
		<=		
		==		
		!=		
		===		
		^~		
		&&		
		||		