Clause 13

Lexical elements

The text of a description consists of one or more design files. The text of a design file is a sequence of lexical elements, each composed of characters; the rules of composition are given in this ~~section~~ clause[1].

13.1 Character set

The only characters allowed in the text of a VHDL description (except within comments—see 13.8)[2] are the graphic characters and format effectors. Each graphic character corresponds to a unique code of the ISO eight-bit coded character set [(ISO 8859-1 : 1987 (E)], and is represented (visually) by a graphical symbol.

basic_graphic_character ::=
    upper_case_letter | digit | special_character | space_character

graphic_character ::=
    basic_graphic_character | lower_case_letter | other_special_character

basic_character ::=
    basic_graphic_character | format_effector

The basic character set is sufficient for writing any description. The characters included in each of the categories of basic graphic characters are defined as follows:

a) Uppercase letters
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Î
   Ï Ð Ñ Ò Ó Ô Õ Ö Ø Ù Ú Û Ü Ý Þ

b) Digits
   0 1 2 3 4 5 6 7 8 9

c) Special characters
   " # & ' () * + , - . / : ; < = > [ ] _ |

d) The space characters
   SPACE[3] NBSP[4]

Format effectors are the ISO (and ASCII) characters called horizontal tabulation, vertical tabulation, carriage return, line feed, and form feed.

1. To conform to IEEE rules.
2. LCS 16.
3. The visual representation of the space is the absence of a graphic symbol. It may be interpreted as a graphic character, a control character, or both.
4. The visual representation of the nonbreaking space is the absence of a graphic symbol. It is used when a line break is to be prevented in the text as presented.

The characters included in each of the remaining categories of graphic characters are defined as follows:

e) Lowercase letters
a b c d e f g h i j k l m n o p q r s t u v w x y z ß à á â ã ä å æ ç è é ê ë í î ï ð ñ ò ó ô
õ ö ø ù ú û ü ý þ ÿ

f) Other special characters
! $ % @ ? \ ^ ` { } ~ ¡ ¢ £ ¥ ¦ § ¨ © ª « ¬ ® ¯ ° ± ² ³ ´ µ ¶ • ¸ ¹ º » ¼ ½ ¾ ¿ × ÷ - (soft hyphen)

Allowable replacements for the special characters vertical line (|), number sign (#), and quotation mark (") are defined in the last ~~clause of this section~~ 13.10[5].

NOTES

1—The font design of graphical symbols (for example, whether they are in italic or bold typeface) is not part of ISO 8859-1:1987.

2—The meanings of the acronyms used in this ~~section~~ clause[6] are as follows: ASCII stands for American Standard Code for Information Interchange, ISO stands for International Organization for Standardization.

3—There are no uppercase equivalents for the characters ß and ÿ.

4—The following names are used when referring to special characters:

Character	Name	Character	Name	Character	Name
"	quotation mark	?	question mark	-	soft hyphen
#	number sign	@	commercial at	®	registered trade mark sign
&	ampersand	[	left square bracket	¯	macron
'	apostrophe, tick	\	backslash, reverse solidus	°	ring above, degree sign
(	left parenthesis	]	right square bracket	±	plus-minus sign
)	right parenthesis	^	circumflex accent	²	superscript two
*	asterisk, multiply	`	grave accent	³	superscript three
+	plus sign	{	left curly bracket	´	acute accent
,	comma	}	right curly bracket	µ	micro sign
-	hyphen, minus sign	~	tilde	¶	pilcrow sign
.	dot, point, period, full stop	¡	inverted exclamation mark	•	middle dot
/	slash, divide, solidus	¢	cent sign	¸	cedilla
:	colon	£	pound sign	¹	superscript one
;	semicolon	¤	currency sign	º	masculine ordinal indicator
<	less-than sign	¥	yen sign	»	right angle quotation mark
=	equals sign	¦	broken bar	¼	vulgar fraction one quarter
>	greater-than sign	§	paragraph sign, section sign	½	vulgar fraction one half
_	underline, low line	¨	diaeresis	¾	vulgar fraction three quarters
|	vertical line, vertical bar	©	copyright sign	¿	inverted question mark
!	exclamation mark	ª	feminine ordinal indicator	×	multiplication sign
$	dollar sign	«	left angle quotation mark	÷	division sign
%	percent sign	¬	not sign		

5. To conform to IEEE rules.
6. To conform to IEEE rules.

The soft hyphen is a graphic character that is imaged by a graphic symbol identical with, or similar to, that representing HYPHEN, for use when a line break has been established within a work.

13.2 Lexical elements, separators, and delimiters

The text of each design unit is a sequence of separate lexical elements. Each lexical element is either a delimiter, an identifier (which may be a reserved word), an abstract literal, a character literal, a string literal, a bit string literal, or a comment.

In some cases an explicit separator is required to separate adjacent lexical elements (namely when, without separation, interpretation as a single lexical element is possible). A separator is either a space character (SPACE or NBSP), a format effector, or the end of a line. A space character (SPACE or NBSP) is a separator except within an extended identifier,[7] a comment, a string literal, or a space character literal.

The end of a line is always a separator. The language does not define what causes the end of a line. However if, for a given implementation, the end of a line is signified by one or more characters, then these characters must be format effectors other than horizontal tabulation. In any case, a sequence of one or more format effectors other than horizontal tabulation must cause at least one end-of-line.

One or more separators are allowed between any two adjacent lexical elements, before the first of each design unit or after the last lexical element of a design file. At least one separator is required between an identifier or an abstract literal and an adjacent identifier or abstract literal.

A delimiter is either one of the following special characters (in the basic character set):

```
& ' ( ) * + , - . / : ; < = > | [ ]
```

or one of the following compound delimiters, each composed of two adjacent special characters:

```
=> ** := /= >= <= <>
```

Each of the special characters listed for single character delimiters is a single delimiter except if this character is used as a character of a compound delimiter or as a character of an extended identifier,[8] a comment, string literal, character literal, or abstract literal.

The remaining forms of lexical elements are described in other ~~clause of this section~~ subclauses of this clause[9].

NOTES

1—Each lexical element must fit on one line, since the end of a line is a separator. The quotation mark, number sign, and underline characters, likewise two adjacent hyphens, are not delimiters, but may form part of other lexical elements.

2—The following names are used when referring to compound delimiters:

Delimiter	Name
=>	arrow
**	double star, exponentiate
:=	variable assignment
/=	inequality (pronounced “not equal”)
>=	greater than or equal
<=	less than or equal; signal assignment
<>	box

7. IR1000.3.1.
8. IR1000.3.1.
9. To conform to IEEE rules.

13.3 Identifiers

Identifiers are used as names and also as reserved words.

identifier ::= basic_identifier | extended_identifier

13.3.1 Basic identifiers

A basic identifier consists only of letters, digits, and underlines.

basic_identifier ::=
letter { [underline] letter_or_digit }

letter_or_digit ::= letter | digit

letter ::= upper_case_letter | lower_case_letter

All characters of a basic identifier are significant, including any underline character inserted between a letter or digit and an adjacent letter or digit. Basic identifiers differing only in the use of corresponding uppercase and lowercase letters are considered the same.

Examples:

COUNT X c_out FFT Decoder

VHSIC X1 PageCount STORE_NEXT_ITEM

NOTE

—No space (SPACE or NBSP) is allowed within a basic identifier since a space is a separator.

13.3.2 Extended identifiers

Extended identifiers may contain any graphic character.

extended_identifier ::=
\ graphic_character { graphic_character } \

If a backslash is to be used as one of the graphic characters of an extended literal, it must be doubled. All characters of an extended identifier are significant (a doubled backslash counting as one character). Extended identifiers differing only in the use of corresponding uppercase and lowercase letters are distinct. Moreover, every extended identifier is distinct from any basic identifier.

Examples:

```
\BUS\      \bus\                    -- Two different identifiers, neither of which is
                                    -- the reserved word bus.

\a\\b\                              -- An identifier containing three characters.

VHDL       \VHDL\       \vhdl\      -- Three distinct identifiers.
```

13.4 Abstract literals

There are two classes of abstract literals: real literals and integer literals. A real literal is an abstract literal that includes a point; an integer literal is an abstract literal without a point. Real literals are the literals of the type *universal_real*. Integer literals are the literals of the type *universal_integer*.

abstract_literal ::= decimal_literal | based_literal

13.4.1 Decimal literals

A decimal literal is an abstract literal expressed in the conventional decimal notation (that is, the base is implicitly ten).

decimal_literal ::= integer [. integer] [exponent]

integer ::= digit { [underline] digit }

exponent ::= E [+] integer | E – integer

An underline character inserted between adjacent digits of a decimal literal does not affect the value of this abstract literal. The letter E of the exponent, if any, can be written either in lowercase or in uppercase, with the same meaning.

An exponent indicates the power of ten by which the value of the decimal literal without the exponent is to be multiplied to obtain the value of the decimal literal with the exponent. An exponent for an integer literal must not have a minus sign.

Examples:

```
12          0       1E6         123_456         -- Integer literals

12.0        0.0     0.456       3.14159_26      -- Real literals

1.34E–12    1.0E+6  6.023E+24                   -- Real literals with exponents
```

NOTE

—Leading zeros are allowed. No space (SPACE or NBSP) is allowed in an abstract literal, not even between constituents of the exponent, since a space is a separator. A zero exponent is allowed for an integer literal.

13.4.2 Based literals

A based literal is an abstract literal expressed in a form that specifies the base explicitly. The base must be at least two and at most sixteen.

based_literal ::=
    base # based_integer [. based_integer] # [exponent]

base ::= integer

based_integer ::=
    extended_digit { [underline] extended_digit }

extended_digit ::= digit | letter

An underline character inserted between adjacent digits of a based literal does not affect the value of this abstract literal. The base and the exponent, if any, are in decimal notation. The only letters allowed as extended digits are

the letters A through F for the digits ten through fifteen. A letter in a based literal (either an extended digit or the letter E of an exponent) can be written either in lowercase or in uppercase, with the same meaning.

The conventional meaning of based notation is assumed; in particular the value of each extended digit of a based literal must be less than the base. An exponent indicates the power of the base by which the value of the based literal without the exponent is to be multiplied to obtain the value of the based literal with the exponent. An exponent for a based integer literal must not have a minus sign.

Examples:

```
-- Integer literals of value 255:
    2#1111_1111#        16#FF#              016#0FF#

-- Integer literals of value 224:
    16#E#E1             2#1110_0000#

-- Real literals of value 4095.0:
    16#F.FF#E+2         2#1.1111_1111_111#E11
```

13.5 Character literals

A character literal is formed by enclosing one of the 191 graphic characters (including the space and nonbreaking space characters) between two apostrophe characters. A character literal has a value that belongs to a character type.

character_literal ::= ' graphic_character '

Examples:

```
'A'   '*'   '"'   ' '
```

13.6 String literals

A string literal is formed by a sequence of graphic characters (possibly none) enclosed between two quotation marks used as string brackets.

string_literal ::= ~~" { graphic_character } "~~ " { graphic_character } "[10]

A string literal has a value that is a sequence of character values corresponding to the graphic characters of the string literal apart from the quotation mark itself. If a quotation-mark value is to be represented in the sequence of character values, then a pair of adjacent quotation marks must be written at the corresponding place within the string literal. (This means that a string literal that includes two adjacent quotation marks is never interpreted as two adjacent string literals.)

The length of a string literal is the number of character values in the sequence represented. (Each doubled quotation mark is counted as a single character.)

Examples:

```
"Setup time is too short"       -- An error message.

""                              -- An empty string literal.

" "     "A"     """"            -- Three string literals of length 1.
```

10. Boyer.

"Characters such as $, %, and } are allowed in string literals."

NOTE

—A string literal must fit on one line, since it is a lexical element (see 13.2). Longer sequences of graphic character values can be obtained by concatenation of string literals. The concatenation operation may also be used to obtain string literals containing nongraphic character values. The predefined type CHARACTER in package STANDARD specifies the enumeration literals denoting both graphic and nongraphic characters. Examples of such uses of concatenation are

"FIRST PART OF A SEQUENCE OF CHARACTERS " &
"THAT CONTINUES ON THE NEXT LINE"

"Sequence that includes the" & ACK & "control character"

13.7 Bit string literals

A bit string literal is formed by a sequence of extended digits (possibly none) enclosed between two quotations used as bit string brackets, preceded by a base specifier.

bit_string_literal ::= base_specifier " [bit_value] "

bit_value ::= extended_digit { [underline] extended_digit }

base_specifier ::= B | O | X

An underline character inserted between adjacent digits of a bit string literal does not affect the value of this literal. The only letters allowed as extended digits are the letters A through F for the digits ten through fifteen. A letter in a bit string literal (either an extended digit or the base specifier) can be written either in lowercase or in uppercase, with the same meaning.

If the base specifier is 'B', the extended digits in the bit value are restricted to 0 and 1. If the base specifier is 'O', the extended digits in the bit value are restricted to legal digits in the octal number system, i.e., the digits 0 through 7. If the base specifier is 'X', the extended digits are all digits together with the letters A through F.

A bit string literal has a value that is a string literal consisting of the character literals '0' and '1'. If the base specifier is 'B', the value of the bit string literal is the sequence given explicitly by the bit value itself after any underlines have been removed.

If the base specifier is 'O' (respectively 'X'), the value of the bit string literal is the sequence obtained by replacing each extended digit in the bit_value by a sequence consisting of the three (respectively four) values representing that extended digit taken from the character literals '0' and '1'; as in the case of the base specifier 'B', underlines are first removed. Each extended digit is replaced according to this table:

Extended digit	Replacement when the base specifier is 'O'	Replacement when the base specifier is 'X'
0	000	0000
1	001	0001
2	010	0010
3	011	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111
8	(illegal)	1000

9	(illegal)	1001
A	(illegal)	1010
B	(illegal)	1011
C	(illegal)	1100
D	(illegal)	1101
E	(illegal)	1110
F	(illegal)	1111

The *length* of a bit string literal is the length of its string literal value.

Examples:

```
B"1111_1111_1111"--  Equivalent to the string literal "111111111111"
X"FFF"               --  Equivalent to B"1111_1111_1111"
O"777"               --  Equivalent to B"111_111_111"
X"777"               --  Equivalent to B"0111_0111_0111"

constant c1: STRING := B"1111_1111_1111";

constant c2: BIT_VECTOR := X"FFF";

type MVL is ('X', '0', '1', 'Z');
type MVL_VECTOR is array (NATURAL range <>) of MVL;
constant c3: MVL_VECTOR := O"777";

assert    c1'LENGTH = 12 and
          c2'LENGTH = 12 and
          c3 = "111111111";
```

13.8 Comments

A comment starts with two adjacent hyphens and extends up to the end of the line. A comment can appear on any line of a VHDL description and may contain any character except the format effectors vertical tab, carriage return line feed, and form feed[11]. The presence or absence of comments has no influence on whether a description is legal or illegal. Furthermore, comments do not influence the execution of a simulation module; their sole purpose is to enlighten the human reader.

Examples:

```
-- The last sentence above echoes the Algol 68 report.

end; -- Processing of LINE is complete

-- A long comment may be split onto
-- two or more consecutive lines.

---------- The first two hyphens start the comment.
```

NOTES

1[12]—Horizontal tabulation can be used in comments, after the double hyphen, and is equivalent to one or more spaces (SPACE characters) (see 13.2).

11. LCS 16.
12. LCS 16.

2—Comments may contain characters that, according to 13.1, are non-printing characters. Implementations may interpret the characters of a comment as members of ISO 8859–1, or of any other character set; for example, an implementation may interpret multiple consecutive characters within a comment as single characters of a multi-byte character set.[13]

13.9 Reserved words

The identifiers listed below are called *reserved words* and are reserved for significance in the language. For readability of this manual, the reserved words appear in lowercase boldface.

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

A reserved word must not be used as an explicitly declared identifier.

NOTES

1—Reserved words differing only in the use of corresponding uppercase and lowercase letters are considered as the same (see 13.3.1). The reserved word **range** is also used as the name of a predefined attribute.

2—An extended identifier whose sequence of characters inside the leading and trailing backslashes is identical to a reserved word is not a reserved word. For example, \next\ is a legal (extended) identifier and is not the reserved word **next**.

13.10 Allowable replacements of characters

The following replacements are allowed for the vertical line, number sign, and quotation mark basic characters:

— A vertical line (|) can be replaced by an exclamation mark (!) where used as a delimiter.

— The number sign (#) of a based literal can be replaced by colons (:), provided that the replacement is done for both occurrences.—The quotation marks (") used as string brackets at both ends of a string literal can be replaced by percent signs (%), provided that the enclosed sequence of characters contains no quotation marks, and provided that both string brackets are replaced. Any percent sign within the sequence of characters must then be doubled, and each such doubled percent sign is interpreted as a single percent sign value. The same replacement is allowed for a bit string literal, provided that both bit string brackets are replaced.

These replacements do not change the meaning of the description.

NOTES

1—It is recommended that use of the replacements for the vertical line, number sign, and quotation marks be restricted to cases

13. LCS 16.

where the corresponding graphical symbols are not available. Note that the vertical line appears as a broken line on some equipment; replacement is not recommended in this case.

2—The rules given for identifiers and abstract literals are such that lowercase and uppercase letters can be used indifferently; these lexical elements can thus be written using only characters of the basic character set.

3—The use of these characters as replacement characters may be removed from a future version of the language. See Annex F.[14]

14. LCS 25.