

MODULE AND LEXICAL TOKENS

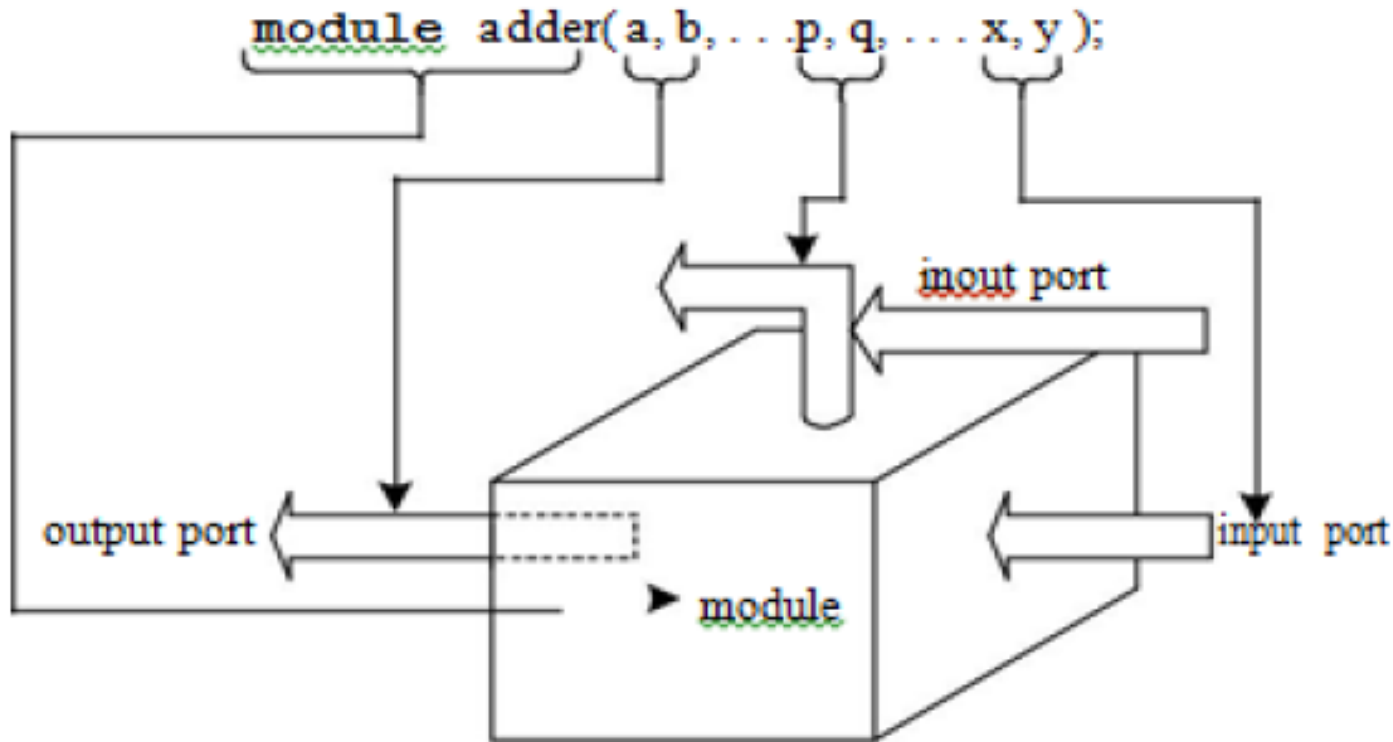


MODULE



- ❑ Any Verilog program begins with a keyword – called a “module.”
- ❑ A module is the name given to any system considering it as a black box with input and output terminals
- ❑ The terminals of the module are referred to as ‘ports’.
- ❑ Whether a module has any of the above ports and how many of each type are present depend solely on the functional nature of the module.

Representation of module as black box and its ports



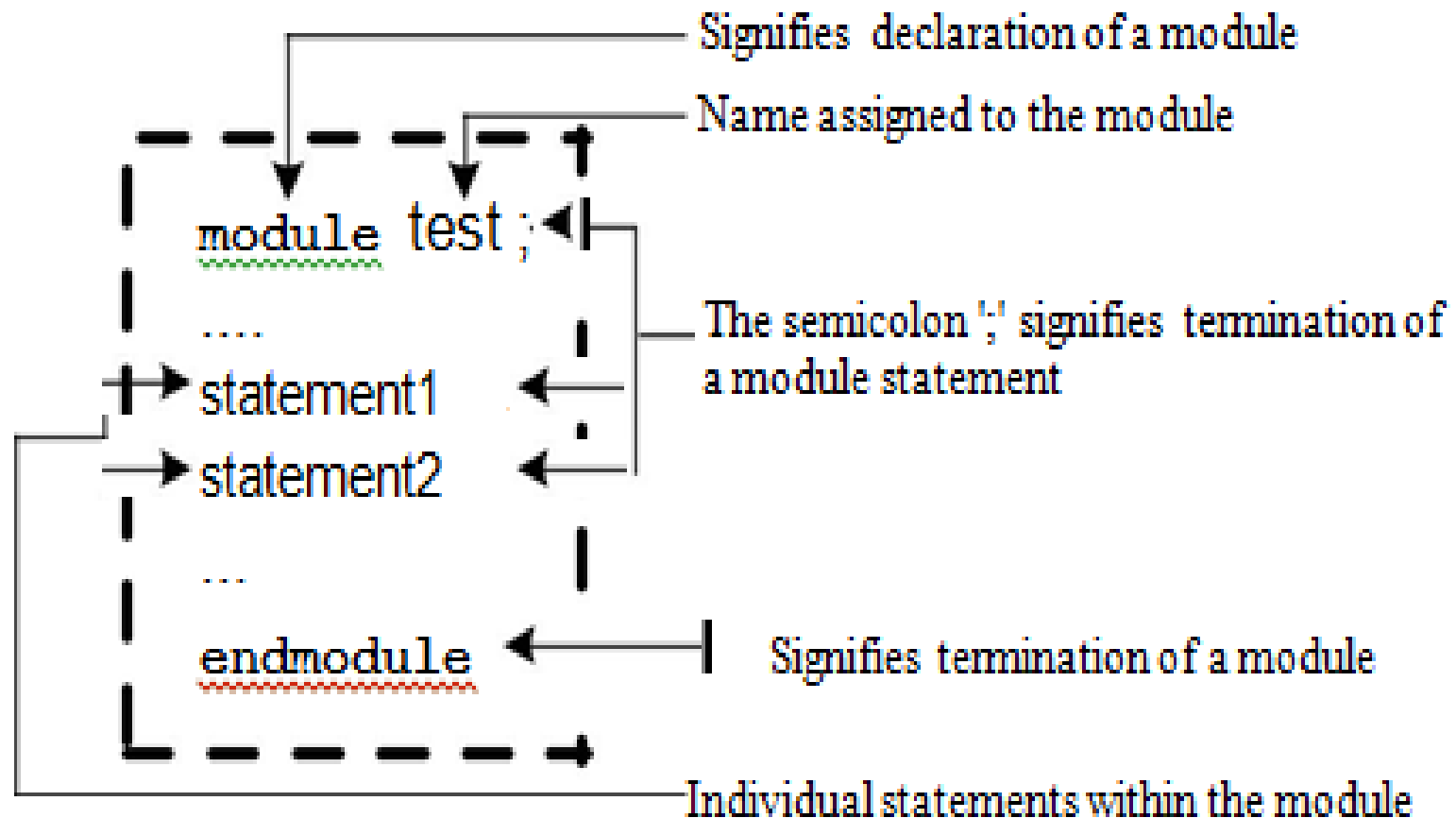
- All the constructs in Verilog are centered on the module. They define ways of building up, accessing, and using modules.


TYPES OF PORTS

The ports attached to a module can be of three types:

- 1. INPUT PORTS:** Input ports through which one gets entry into the module; they signify the input signal terminals of the module.
- 2. OUTPUT PORTS:** Output ports through which one exits the module; these signify the output signal terminals of the module.
- 3. INOUT PORTS:** These represent ports through which one gets entry into the module or exits the module; These are terminals through which signals are input to the module sometimes; at some other times signals are output from the module through these

Typical structure of a MODULE



- 
- ❑ In Verilog any program which forms a design description is a “module.”
 - ❑ Any program written to test a design description is also a “module.” These are often called as “stimulus modules” or “test benches.”
 - ❑ Verilog takes the active statements appearing between the “module” statement and the “endmodule” statement and interprets all of them together as forming the body of the module.
 - ❑ Whenever a module is invoked for testing or for incorporation into a bigger design module, the name of the module (“test” here) is used to identify it for the purpose.

LANGUAGE CONSTRUCTS AND CONVENTIONS IN VERILOG

- ❑ The constructs and conventions make up a software language. A clear understanding and familiarity of these is essential for the mastery of the language.
- ❑ Verilog has its own constructs and conventions [IEEE, Sutherland].
- ❑ In many respects they resemble those of C language [Gottfried].
- ❑ Any source file in Verilog is made up of a number of ASCII characters.
- ❑ The characters are grouped into sets — referred to as “lexical tokens.” A lexical token in Verilog can be a single character or a group of characters.

LEXICAL TOKENS

□ A module comprises a number of “lexical tokens” arranged according to some predefined order. The possible tokens are of seven categories:

- ❖ Keywords
- ❖ Identifiers
- ❖ White spaces
- ❖ Comments
- ❖ Operators
- ❖ Numbers
- ❖ Strings

1. KEYWORDS



- ❑ The keywords define the language constructs.
- ❑ A keyword signifies an activity to be carried out, initiated, or terminated. As such, a programmer cannot use a keyword for any purpose other than that it is intended for.
- ❑ All keywords in Verilog are in small letters and require to be used as such (since Verilog is a case-sensitive language).
- ❑ All keywords appear in the text in New Courier Bold-type letters.

Example:

module --	signifies the beginning of a module definition.
endmodule --	signifies the end of a module definition.
begin --	signifies the beginning of a block of statements
end --	signifies the end of a block of statements.
if --	signifies a conditional activity to be checked
while --	signifies a conditional activity to be carried out.

Case Sensitivity

- ❑ Verilog is a case-sensitive language like C.
- ❑ Thus sense, Sense, SENSE, sENse,... etc., are all related as different entities / quantities in Verilog.

2. Identifiers

- Any program requires blocks of statements, signals, etc., to be identified with an attached nametag. Such nametags are identifiers.
- It is good practice for us to use identifiers, closely related to the significance of variable, signal, block, etc., concerned. This eases understanding and debugging of any program.
- e.g., *clock, enable, gate_1, . . .*
- There are some restrictions in assigning identifier names.
- All characters of the alphabet or an underscore can be used as the first character.
- Subsequent characters can be of alphanumeric type, or the underscore (_), or the dollar (\$) sign.

□ For example,

name, _name. Name, name1, name_\$, ... -- all these are allowed as identifiers

- *name aa* -- not allowed as an identifier because of the blank space between (“name” and “aa” are interpreted as two different identifiers)
- *\$name* -- not allowed as an identifier because of the presence of “\$” as the first character.
- *1_name* -- not allowed as an identifier, since the numeral “1” is the first character
- *@name* -- not allowed as an identifier because of the presence of the character “@”.
- *A+b m* --not allowed as an identifier because of the presence of the character “+”.

3. White Space Characters

- ❑ **Blanks (\b), tabs (\t), newlines (\n),** and formfeed form the white space characters in Verilog.
- ❑ In any design description the white space characters are included to improve readability.
- ❑ Functionally, they separate legal tokens.
- ❑ They are introduced between keywords, keyword and an identifier, between two identifiers, between identifiers and operator symbols, and so on.
- ❑ White space characters have significance only when they appear inside strings.

4. Comments

- Comments can be inserted in the code for readability and documentation.
- There are two ways to write comments.
- A one-line comment starts with **"//"**. Verilog skips from that point to the end of line.
- A multiple-line comment starts **with "/*" and ends with "*/"**.
- Multiple-line comments cannot be nested. However, one-line comments can be embedded in multiple-line comments.

Examples:

1. `a = b && c; // This is a one-line comment`
2. `/* This is a multiple line
comment */`
3. `/* This is /* an illegal */ comment */`
4. `/* This is //a legal comment */`

5. Operators

- Operators are of three types: **UNARY, BINARY, TERNARY**.
- Unary operators precede the operand.
- Binary operators appear between two operands.
- Ternary operators have two separate operators that separate three operands.

`a = ~ b;` // `~` is a unary operator. `b` is the operand

`a = b && c;` // `&&` is a binary operator. `b` and `c` are operands

`a = b ? c : d;` // `?:` is a ternary operator. `b`, `c` and `d` are operands

6. Numbers

- There are two types of number specification in Verilog:

SIZED & UNSIZED

1. Sized numbers

- Sized numbers are represented as

<size> '<base format> <number>

4'b1111 // This is a 4-bit binary number

12'habc // This is a 12-bit hexadecimal number

16'd255 // This is a 16-bit decimal number.

- ❖ <size> is written only in decimal and specifies the number of bits in the number. Legal base formats are decimal ('d or 'D), hexadecimal ('h or 'H), binary ('b or 'B) and octal ('o or 'O).

- ❖ The number is specified as consecutive digits from 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f.
- ❖ Only a subset of these digits is legal for a particular base.
- ❖ **Uppercase letters are legal for number specification.**

2. Unsized numbers

- ❑ Numbers that are specified without a <base format> specification are decimal numbers by default.
- ❑ Numbers that are written without a <size> specification have a default number of bits that is simulator- and machine-specific (must be at least 32).

23456 // This is a 32-bit 'hc3 // This is a 32-bit 'o21 // This is a 32-bit

decimal number by default hexadecimal number octal number

Number specification example

<size> '<base format> <number>

Allowed only
with decimal
number

If present, the decimal
number in this field
signifies the bit width of
number.

If absent, the width is
assigned a default value by
compiler.

This field signifies value of the number.

To form values:

For binary : 0,1,x,z can be used

For Octal: 0 to 7, x, z can be used

For Hexa : all numerals,a,b,c,d,e,f,x,z

For decimal: all numerals, x,z

<base format> can be b or B, d or
D, o or O and h or H.

Numbers without <base format>
are decimal by default.

Number Specification -Example

Sized numbers :

4'b1111 // This is a 4-bit binary number

12'habc // This is a 12-bit hexadecimal number

16'd255 // This is a 16-bit decimal number.

Unsize numbers :

23456 // This is a 32-bit decimal number by default

'hc3 // This is a 32-bit hexadecimal number

'o21 // This is a 32-bit octal number


Unknown and High Impedance Values (X or Z values)

- Verilog has two symbols for unknown and high impedance values. These values are very important for modeling real circuits.
- **An unknown value is denoted by an x.**
- **A high impedance value is denoted by z.**

12'h13x // This is a 12-bit hex number; 4 least significant bits unknown

6'hx // This is a 6-bit hex number

32'bz // This is a 32-bit high impedance number

- 
- An x or z sets
 - 4 bits for a number in the hexadecimal base,
 - 3 bits for a number in the octal base, and
 - 1 bit for a number in the binary base.
 - If the most significant bit of a number is 0, x, or z, the number is automatically extended to fill the most significant bits, respectively, with 0, x, or z.
 - This makes it easy to assign x or z to whole vector.
 - If the most significant digit is 1, then it is also zero extended.

Negative numbers

- ❑ Negative numbers can be specified by putting a minus sign before the size for a constant number.
 - ❑ Size constants are always positive.
 - ❑ It is illegal to have a minus sign between <base format> and <number>.
 - ❑ An optional signed specifier can be added for signed arithmetic.
1. **-6'd3** // 8-bit negative number stored as 2's complement of 3
 2. **-6'sd3** // Used for performing signed integer math
 3. **4'd-2** // Illegal specification

Underscore Characters And Question Marks

- ❑ An underscore character "_" is allowed anywhere in a number except the first character.
- ❑ Underscore characters are allowed only to improve readability of numbers and are ignored by Verilog.
- ❑ A question mark "?" is the Verilog HDL alternative for z in the context of numbers.
- `12'b1111_0000_1010` // Use of underline characters for readability
- `4'b10??` // Equivalent of a `4'b10zz`

Number Specification-Example

5'037	5-bit octal
4'D2	4-bit decimal
9'b11011x01	x signifies the concerned bit to be of unknown value.
9'o12z	equivalent to 001 010 zzz
7'Hx	7-bit x (x extended), i.e.... xxxxxxx
4'hz	4-bit z (z extended), i.e... zzzz
4'd-4	Not legal
-4'd7	Its value in 2's complement form is 7.
8 'h 2A	Spaces allowed between size & ' character & between <i>base</i> and <i>value</i>
3' b001	Not legal: no space allowed between ' and base b
10'b10	Padded with 0 to the left, 0000000010
11'hb0	equivalent value is 000 1011 0000.
5'hza	A 5-bit hex number. Its value is taken as z 1010.
3'b1001_0011	is same as 3'b011

7. Strings

- A string is a sequence of characters that are enclosed by double quotes.
- The restriction on a string is that it must be contained on a single line, that is, without a carriage return. It cannot be on multiple lines.
- Strings are treated as a sequence of one-byte ASCII values.
 - **"Hello Verilog World"** // is a string
 - **"a / b"** // is a string

Value Set or Logic Values

- Verilog supports four values and eight strengths to model the functionality of real hardware.
- The four value levels are listed in Table below.

Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
Z	High impedance, floating state


Strengths

- ❑ The logic levels are also associated with strengths.
- ❑ In many digital circuits, multiple assignments are often combined to reduce silicon area or to reduce pin-outs.
- ❑ To facilitate this, one can assign strengths to logic levels.
- ❑ Verilog has eight strength levels
 - 4 of these are of the driving type,
 - 3 are of capacitive type and
 - 1 of the hi-Z type.
- ❑ In addition to logic values, strength levels are often used to resolve conflicts between drivers of different strengths in digital circuits.

□ Value levels 0 and 1 can have the strength levels listed in Table below

Strength Level	Type
supply	Driving
strong	Driving
pull	Driving
large	Storage
weak	Driving
medium	Storage
small	Storage
highz	High impedance

Degree
strongest
↑
weakest

- 
- If two signals of unequal strengths are driven on a wire, the stronger signal prevails.
 - For example, if two signals of strength `strong1` and `weak0` contend, the result is resolved as a `strong1`.
 - If two signals of equal strengths are driven on a wire, the result is unknown.
 - If two signals of strength `strong1` and `strong0` conflict, the result is an `x`.
 - Strength levels are particularly useful for accurate modeling of signal contention, MOS devices, dynamic MOS, and other low-level devices.