IV-[redacted]

ment of Electronics & Communication Engineering

UNIT-II

LANGUAGE CONSTRUCTS AND CONVENTIONS IN VERILOG

INTRODUCTION:

The constructs & conventions make up a Software language. Any Source file in Verilog (as with any file in any other programming language) 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. Verilog has 7 types of lexical tokens

- Operators
- Keywords
- Identifiers
- white Spaces
- comments
- numbers
- strings.

Keywords

The keyword signifies an activity to be carried Out, initiated, or terminated 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.

Eg: 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

Identifiers

Any program requires blocks of statements, signals, etc., to be identified with an attached nametag. Such nametags are identifiers. It is good practise for us to use identifiers, closely related to the significance of variable, signal, block, etc., concerned.

eg: 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 eg.

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

name aa ← not allowed as an identifier because of the blank ("name" & "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 ← not allowed as an identifier because of the presence of the character "+".

An alternative format makes it is possible to use any of the printable Ascii characters in an identifier. Such identifiers are called "escaped identifiers"; they have to start with the ~~bla~~ backslash (\) character. The character set b/w the first backslash character & the first white space encountered is treated as an identifier.

eg:-

```
\b:c
\control-signal
\& logic
\abc // Here the combination "abc" forms the ~~ch~~ identifier.
```

It is preferable to use the former type of identifiers & avoid the escaped identifiers; they may be reserved for use in files which are available as inputs to the design from other CAD tools.

White space characters

Blanks (\b), Tabs (\t), newlines (\n), & 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 b/w keywords. Keyword & an identifier, b/w two identifiers, b/w identifiers & Operator symbols, & so on. white space characters have significance Only when they appear inside strings.

Comments:

It is a healthy practice to comment a design description liberally - as with any Other program. comments are incorporated in two ways. 1) Single line comment begins with "//" & ends with a newline - for eg.

```
module d-ff (Q, dp, clk); // this is the design description of a D flip-flop.
// Here Q is the o/p
// dp is the i/p & clk is the clock
```

One can incorporate multiline comments also without resorting to "//" at every line. For Such multiline comments "/*" signifies the beginning of a comment & "*/" is end. All lines appearing b/w these two Symbol combinations are together treated as a Single block comment - for eg.

```
module d-ff( Q, dp, clk);
/* this module forms the design description of a d-flip-flop wherein
        Q is the o/p of the flip-flop.
        dp is the data i/p &
        clk the clock i/p */
```

Multiline comments Cannot be nested. For eg, the following comment is not Valid.

/* the following forms the design description of a D flip flop /* which can be modified to form Other types of flip-flops */ with clk & data i/p */.

A Valid Alternative can be as follows:

/* the following forms the design description of a D flip-flop (which can be modified to form other types of flipflops) with clock & data i/p

Numbers:

Frequently numbers needs to be Specified in a design description. logic status of a Signal lines, buses, delay values, & numbers to be loaded in registers are eg. The numbers can be of Integer type or real type.

Integer Numbers.

Integers can be represented in two ways. In the first case it is a decimal number Signed or Unsigned; An Unsigned number is Automatically taken as a positive number. Some examples of Valid number representations of this category are given below.

```
2
25
253
-253
```

The following are invalid since nondecimal representations are not permissible.

```
2a
B8
-2a
-B8
```

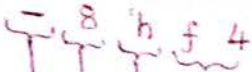
Normally the number is taken as 32 bits wide. Thus all the following numbers are assigned 32 bits of width.

```
2
25
253
-2
-25
-253
```

If a design description has a number Specified in the form given here, the circuit Synthesizer program will assign 32 bits of width to it & to all the related circuits. Hence all such number Specifications - despite their Simplicity - may be Avoided in design descriptions. Number representation in this form may preferably be restricted to test benches.

The alternate form of number representation is more Specific - though elaborate The number can be Specified in binary, Octal, decimal, or hexade

The representation has three tokens with an optional sign preceding it. ③ fig shows typical number representations with the significance of each field.



→ this field signifies the value of the number. For binary numbers the characters 0, 1, x, z can be used to form the value.

For octal numbers the numerals 0 to 7, x, z can be used to form the value.

For decimal numbers all the numerals, x, z can be used to form the value.

For hex numbers all the numerals, a, b, c, d, e, f, x, z can be used to form the numbers.

→ this combinations - the single quote character followed by b, o, d or h - specifies the base of the number. The character signifies binary, octal, decimal or hexadecimal base. If this field is absent, the number is taken as a decimal one.

→ If present, the decimal number in this field signifies the bit width of the number. If absent the width is assigned a default value by the compiler.

→ this field (optional) is for the sign bit. It is allowed only with the decimal numbers. If absent, the number is taken as positive. For a number with a negative sign the number is represented in 2's complement form.

Representation of a number in Verilog. One can use capital letters instead small letters in the last two fields.

Observations:

- The characters used to specify the base numbers, the sign or the magnitude can be in either case (Thus A, B, C, D, E or F can be used in place of a, b, c, d, e or f, respectively, to specify the concerned hex digit. x or z can be used in place of x or z value. respectively).

- The single quote character in the base field has to be immediately followed by the character representing the base. Intervening white spaces are not allowed. However such white spaces can precede the magnitude field.
- -Ve numbers are represented in 2's complement form.
- The question mark character – "?" – can be used in place of z. The Underscore character can be used anywhere after the first character. It adds to the readability. It is normally ignored.
- If the number size is smaller than the size specified. The size is made up by padding 0's to the left. However, if the leftmost bit is a x or z, the same is padding to the left.
- Left truncation & right extension can often be confusing. It is preferable to specify the numbers fully.

Real Numbers:

Real numbers can be specified in decimal or scientific notation. The decimal notations has the form.

$-a.b$

where a, b the -Ve sign, & the decimal point have the usual significance. The fields a & b must be present in the number. A number can be specified in scientific notation as.

4.3e2

where 4.3 is the mantissa & 2 the exponent. The decimal equivalent of this number is 430. Other egs

-4.3e2, -4.3e-2, & 4.3e-2.

Strings :

A string is a sequence of characters enclosed within double quotes. A string must be contained on a single line ; that is, it cannot be carried over to two lines with a carriage return. Special characters are specified by preceding them with the "\" character. Verilog treats a string as a sequence of Ascii characters - for eg.

" this is a string "

" this string is one \t with a gap in b/w "

" this is called a \"string\" ":

When a string of Ascii characters as above is an operand in an expressions, it is treated as a binary number. This binary number is formed by replacing each Ascii character by 8 bits - a 0 bit followed by the 7-bit Ascii equivalent - and treating the resulting binary sequence as a single binary number. For eg, the statement (with P defined as a 32-bit vector beforehand).

p = "numb"

assigns the binary value.

0110 1110 0111 0101 0110 1101 0110 0010

to p (0110 1110, 0111 0101, 0110 1101 & 0110 0010 are the 8 bit equivalents of the letters n, u, m, & b respectively.

Different ways of number representations in Verilog.

Representation	Remarks
(1) 9'hza	A hex number of 9 bits. Its value is taken as zzzz 1010.
(2) 'o213	An Octal number of unspecified size having octal value 213.

Logic Values :

Signal lines, logic values appearing on signal lines, etc., can normally take two logic levels :

1 ← Signifies the 1 or high or true level

0 ← Signifies the 0 or low or false leve

Two additional levels are also possible - designated as x and z. Here x represents an Unknown or an Uninitialized Value. This corresponds to the don't care case in logic circuits. z represents/signifies a high impedance state. this is possible when a signal line is tri-stated or left floating.

• when a Variable in an Expression is in the z state, the effect is the same as it having x Value. But when an i/p to a gate is in the z state. It is equivalent to having the x Value.

• If the i/p to a Mos Switches is in the z state, its o/p too remains at the z state.

• In Verilog modeling we are Using four logic Values 0, 1, x, z.

A logic state can have a "strength" associated with it. It is a quantitative representation of the Internal impedance Value of the corresponding hardware circuit; a change in the Internal Impedance is reflected as a corresponding change in the strength level. whenever the logic Values from two sources are combined, there can be a conflict & the resulting contention has to be resolved.

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 pinouts. To facilitate this, One can Assign strengths to logic levels. Verilog has eight strength levels - four of these are of the driving type, three are of Capacitive type & One of the hi-z type.

When a Signal line is driven Simultaneously from two Sources of different strength levels, the Stronger of the two prevails.

• If a Signal line is driven by two Sources - b at 1 level with Strength "Strong 1" & c at level 0 with Strength "pull 0".

Details of Strengths in Verilog

Strength name	Strength level (signifies inverse of Source Impedance)	Specification Keyword	Abbreviation	element modeled
Supply drive	7	Supply 1 Supply 0	Su1 Su0	power Supply connection
Strong drive	6	Strong 1 Strong 0	St1 St0	Default gate & assign o/p strength
Pull drive	5	pull 1 pull 0	Pu1 Pu0	Gate & assign o/p Strength.
large capacitor	4	large 1 large 0	La1 La0	Size of trireg net capacitor.
Weak drive	3	weak 1 weak 0	We1 We0	Gate & assign o/p strength.
Medium capacitor	2	Medium 1 Medium 0	Me1 Me0	Size of trireg net capacitor.
Small capacitor	1	Small 1 Small 0	Sm1 Sm0	Size of trireg net capacitor.
high Impedance	0	highz 1 highz 0	Hi 1 Hi0	Tri-stated line.

Data types :-

The data handled in Verilog fall into two categories.

(i) Net data type

(ii) Variable data type.

The two types differ in the way they are Used as well as with regard to their respective hardware structure. Data type of each Variable or Signal has to be declared prior to its Use. The same is Valid within the concerned block or mod

Nets:

A net signifies a connection from one circuit's unit to another. Such a net carries the value of the signal it is connected to and transmits to the circuit blocks connected to it. If the driving end of a net is left floating, the net goes to the high impedance state. A net can be specified in different ways.

wire: It represents a simple wire doing an interconnection. Only one o/p is connected to a wire & is driven by that.

tri: It represents a simple signal line as a wire. Unlike the wire, a tri can be driven by more than one signal o/p

Functionally, wire & tri are ~~tri~~ identical. Distinct nomenclatures are provided for the convenience of assigning roles.

Variable Data Type:

A variable is an abstraction for a storage device. It can be declared through the keyword reg and stores the value of a logic level; 0, 1, x, or z. A net or wire connected to a reg takes on the value stored in the reg and can be used as input to other circuit elements. But the o/p of a circuit cannot be connected to a reg. The value stored in a reg is changed through a fresh assignment in the program. time, integers, real, & realtime are the other variable type of data;

Scalars And Vectors:

Entities representing single bits – whether the bit is stored, changed, or transferred – are called "scalars". Often multiple lines carry signals in a cluster-like data bus, address bus, & so on.

Similarly, a group of regs stores a value, which may be assigned, changed, & handled together. The collection here is treated as 'Vector'. Fig the difference b/w a Scalar & a Vector. wr and rd are two scalar nets connecting two circuit blocks circuit 1 & circuit 2. b is a 4-bit wide Vector net connecting the same two blocks. b[0], b[1], b[2], & b[3] are the individual bits of Vector b. They are "part Vectors".

A Vector reg or net is declared at the Outset in a Verilog program & hence treated as Such. The range of a Vector is Specified by a set of 2 digits (or expressions evaluating to a digit) with a colon in between the two. The combination is enclosed within Square brackets.

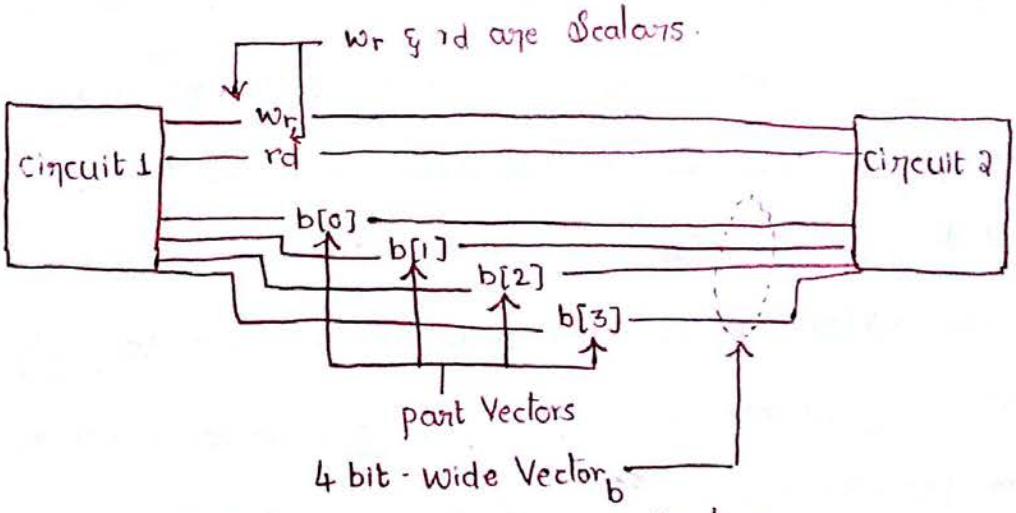


Illustration of Scalars & Vectors

Eg:-

```
wire [3:0] a; /* a is a four bit Vector of net type, the bits are designated as
                 a[3], a[2], a[1] & a[0]. */
reg [2:0] b: /* b is a three bit Vector of reg type; the bits are designated as
                b[2], b[1] & b[0]. */
reg [4:2] c; /* c is a three bit Vector of reg type; the bits are designated as
                c[4], c[3] & c[2]. */
wire [-2:2] d; /* d is a 5 bit Vector with individual bits designated as d[-2],
                  d[-1], d[0], d[1] & d[2]. */
```

Whenever a range is not specified for a net or a reg, the same is treated as a scalar - a single bit quantity. In the range specification of a vector the most significant bit & the least significant bit can be assigned specific integer values. These can also be expressions evaluating to integer constants - +ve or -ve.

Normally vectors - nets or regs - are treated as unsigned quantities. They have to be specifically declared as "signed" if so desired.

Eg:-

```
wire signed [4:0] num; //num is a vector in the range -16 to +15.
reg signed [3:0] num-1; //num-1 is a vector in the range -8 to +7.
```

Parameters:

In some designs, certain parameter values are not committed at the outset. Proportionality constants, frequency-scaling levels, number of taps in digital filters, etc., are typical examples. There are also situations where the size of the design is left open and decided at a later stage. Bus width, µp depth, & memory size are such quantities which may be committed later. All such constants can be declared as parameters at the outset in a Verilog module, & values can be assigned to them; for example,

```
parameter word_size = 16;
parameter word-size = 16, mem_size = 256;
```

Such parameter assignments are made at compiler time. The parameter values cannot be changed (normally) at runtime. However, a parameter that has been assigned a value in a module definition can have its value changed at runtime — in some other design (i.e., instantiated) or when it is tested. These are carried out through a "defparameter" statement. The

done as part of parameter declaration can have the appropriate constant on the right-hand side of the assignment statement, as was the case above. The assignment can also have algebraic expressions on the right hand side. such expressions can involve constants & other parameters declared already;

```
parameter word_size = 16, factor = word_size/2;
```

Operators:

Verilog has a number of operators akin to the c language. These are of three types:

1. Unary; the Unary operator is associated with a single operand. The operator precedes the operand - for example, ~a.
2. Binary; the binary operator is associated with two operands. The operator appears between the two operands - for example, a & b.
3. Ternary; the ternary operator is associated with three operands. The two operators together constitute a ternary operation. The two operators separate the three operands - for example

```
a ? b : c // Here the operators "?" and ":" together define an operation.
```