UNIT – 1

INTRODUCTION TO VERILOG

INTRODUCTION TO VERILOG:

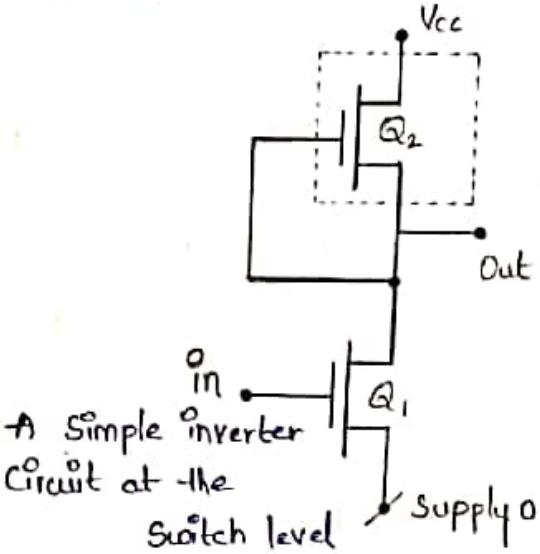
Verilog as an HDL

Verilog aimed at providing as functionally tested and a verified design description for the target FPGA (or) ASIC. The language has a dual function One fulfilling the need for a design description and the other fulfilling the need for verifying the design for functionally and timing Constraints like propagation delay, Critical path delay, slack, setup, and hold times.

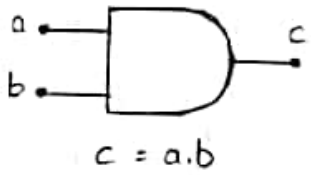
LEVELS OF DESIGN DESCRIPTION

1. **Circuit level:-**

At the Circuit level, a Switch in the basic element with which digital Circuits are built. Switches Can be Combined to form inverters and other gates at the next higher level of abstraction. Verilog has the basic MOS Switches built into its Constructs, which Can be Used to build Circuits like inverters, basic logic gates, simple-1-bit dynamic and static memories.



A Simple inverter Circuit at the Switch level



A Simple AND gate respresented at the gate level.

2. **Gate level :-**

At the next higher level of abstraction, design is Carried Out in terms of basic gates. All the basic gates are avaiable as ready modules called "primitives". Each Such primitive is defined in terms of its inputs and Outputs. primitives Can be incoporated into design descriptions directly.

Scanned with CamScanner

3. Data flow :- Data flow is the next higher level of abstraction. All Possible operations on signals and variables are represented here in terms of assignments. All logic and algebraic operations are accommodated. The assignments define the continuous functioning of the concerned block. At the data flow level, signals are assigned through the data manipulating equations. All such assignments are concurrent in nature. The design descriptions are more compact than those at the gate level.

$$e = \overline{a \cdot b + c \cdot d}$$

An A-O-I gate represented as a data flow type of relationship.

4. Behavioral level :-

Behavioural level constitutes the highest level of design descriptio-n. it is essentially at the system level itself. with the assignment possibilities, looping constructs and conditional branching possible, the design description essentially looks like a "C" program.

```
If (a, b, c or d changes)
Compute e as
        e = ~(a.b + c.d)
```

An A-O-I gate in pseudo code at behavioral level.

CONCURRENCY

In a electric circuit all the units are to be active and functioning concurrently. The voltages and currents in the different elements in the circuit can change simultaneously. In turn the logic levels too can change. Simulation of such a circuit in an HDL calls for concurrency of operation. Verilog simulators are built to simulate concurrency.

SIMULATION AND SYNTHESIS

The design that is specified and entered as described is simulated for functionality and fully debugged. Translation of the debugged design into the corresponding hardware circuit is called "Synthesis" The circuits realized from them are essentially direct translations of functions into circuit elements.

FUNTIONAL VERIFICATION

Testing is an essential ingredient of the VLSI design process as with any hardware circuit. It has two dimensions to it. functional tests and timing tests. Testing (or) functional

www.Jntufastupdates.com

Scanned with CamScanner

Verification is carried out by setting up a "test bench" for the design. The test bench will have the design instantiated in it. it will generate necessary test signals and apply them to the instantiated design. (3)

TEST INPUTS FOR TEST BENCHES:

Any digital systems has to carry out a number of activities in a defined manner. Once a proper design is done, it has to be tested for all its functional aspects. Test inputs can be purely combinational, periodic, numeric sequences, random inputs, conditional inputs (or) combinations of these. As the circuit design proceeds, one develops smaller blocks and groups them together to form bigger circuits units. The process is repeated until the whole system is fully built up. Every stage calls for tests to see whether the subsystem at that layer behaves in the manner expected.

Such testing calls for two types of observations.

study of signals within a small unit until when test inputs are given to the whole unit. Isolation of a small element and doing local test to facilate debugging.

Constructs for modelling Timing delays:

Any basic gate has propagation delays and transmission delays associated with it. As the elements in the circuit increase in number, the type and variety of such delay increase rapidly, often one reaches a stage where the expected function is not realized thanks to an unduly large time delay.

Verilog has constructs for modelling the following delays:

- ↳ Gate delay
- ↳ Net delay
- ↳ path delay
- ↳ pin-to-pin delay.

A design can be tested for setup time, hold time, clock-width time specifications etc.. Such constructs (or) delay models are akin to the finite delay time, rise time, fall time, path (or) propagation delays etc.. associated with real digital circuits (or) systems. The use of such constructs in the design helps simulate realistic conditions in a DC

Scanned with CamScanner

PROGRAMMING LANGUAGE INTERFACE (PLI)

PLI provides an active interface to a Compiled Verilog module. The interface adds a new dimension to working with Verilog routines from a C platform.

The key functions of the interface are as follows:

One can read data from a file and pass it to a Verilog module as input. Such data can be test vectors (or) other input data from a file and pass it to a Verilog module. Similarly, Variables in Verilog modules can be used as accessed and their values written to Output devices.

Delay values, logic values, etc.. within a module can be accessed and altered. Blocks written in C language can be linked to Verilog modules.

SYSTEM TASKS

-A number of systems tasks are avaiable in Verilog. Through used in a design description, they are not-part of it. Some tasks facilate control and flow of the testing process. Reading data from specified files into a module and writing back into files are also possible through other tasks. Timescale can be changed prior to simulation with the help of specific tasks for the purpose.

-A set of system functions add to the flexibility of test benches: They are of 3 categories:

↳ functions that keep track of the progress of simulati -on time.

↳ functions to convert data (or) values of variables from one format to another.

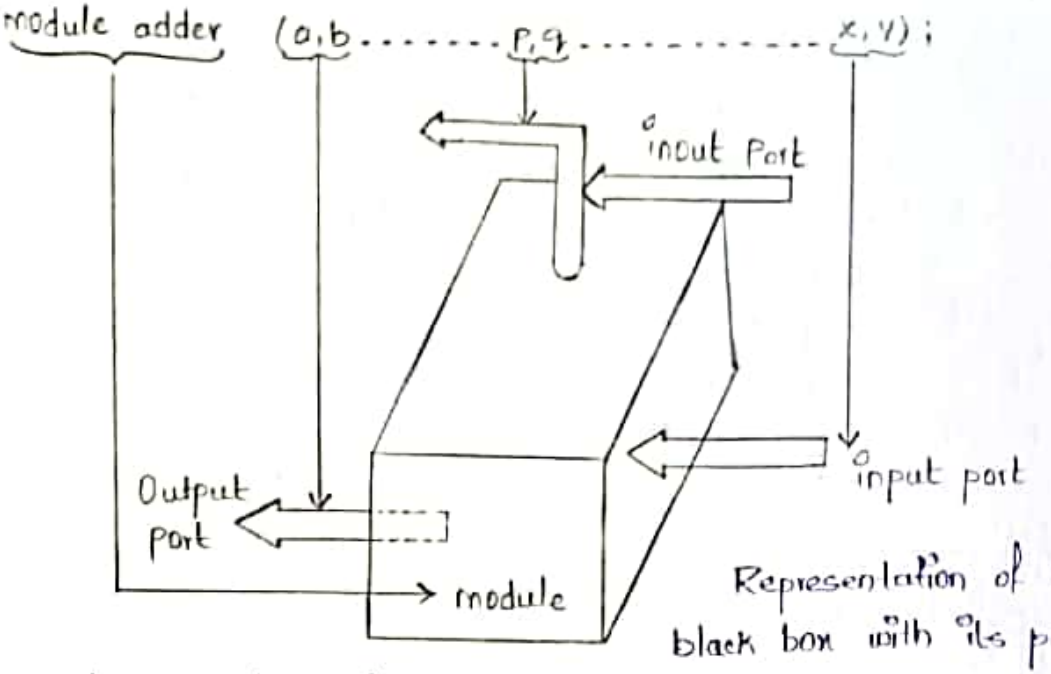
↳ functions to generate random numbers with specific distributions.

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'.

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

Scanned with CamScanner



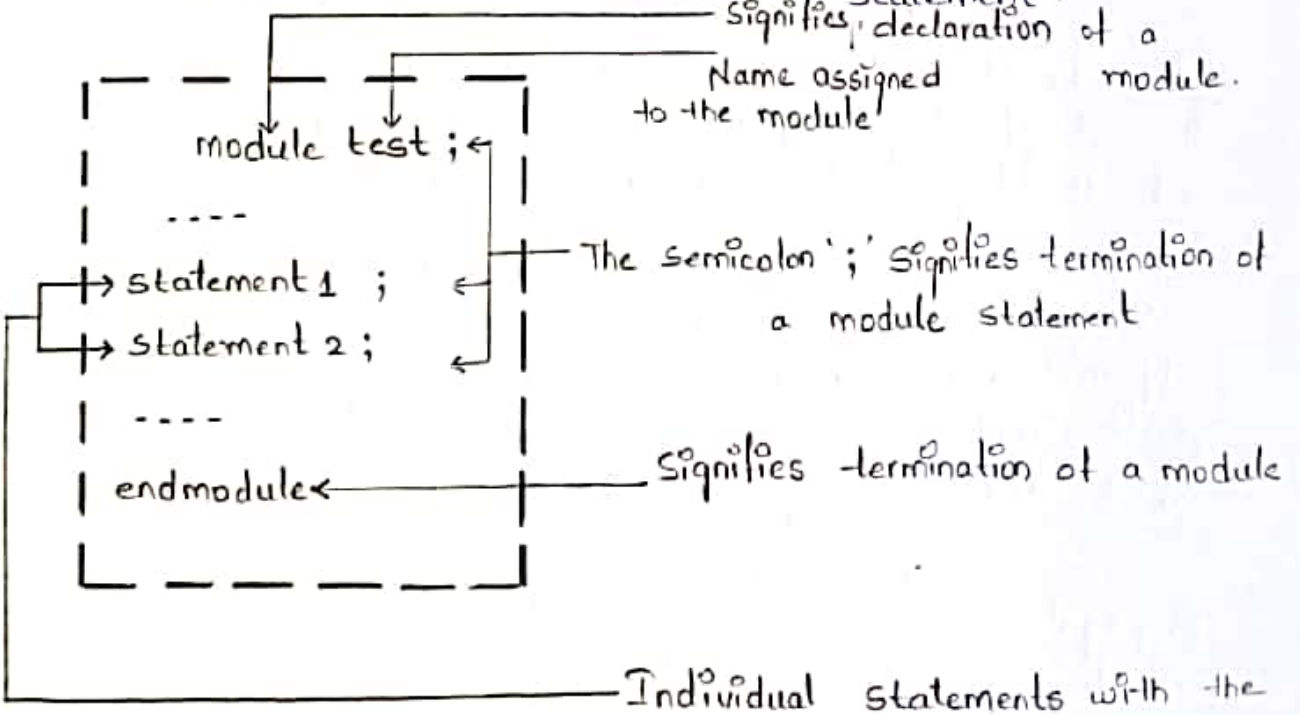
Representation of a module as black box with its ports.

input - ports through which one gets entry into the module; they signify the input signal terminals of the module.

Output - ports through which one exists the module; these signify the Output terminals of the module. inout ports:

These are the representing ports through which one gets entry into the module (or exists 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.

Verilog takes the active statements appearing between the "module" statement and the "endmodule" statement.



Structure of a typical Simulation module.

Scanned with CamScanner

SIMULATION AND SYNTHESIS TOOLS

A variety of software tools related to VLSI design is avaiable. we discuss here two of them directly relevant to us - Modelsim and leonardo spectrum of mentor Graphics. Modelsim has been used to simulate the designs. Simulation results presented for the variety of examples discussed in the book have been obtained using it. Leonardo spectrum of mentor has been used to obtain the synthesized circuits presented. we would like to draw the attention of the readers to the following in the context:

* Only the essential aspects of the tools are presented - those essential for the progress of the book
* for more details of the tools and the variety of facilities they offer, one can refer to the respective user manuals and the help menus.
* Tools from other sources are similar in essentials. Any of them can be used.

TEST BENCHES -

Any digital circuit that has been designed and wired goes through a testing process before being declared as ready for use. Testing involves studying circuit behaviour under simulated conditions for the following:

Check and ensure that all functions are carried out as desired. It is the test for the static behaviour of the circuit. A set of logic input values are applied at selected points, and the logic values at another set of points observed.

Check and ensure that all the functional sequences are carried out as desired. It is one of the tests for the dynamic behaviour of the circuit. It may call for the generation of specific input sequences with respect to time, applying them to the circuit and observing selecting outputs.

check for the timing behaviour. One tests for the propagation and other types of delays here. A variety of tests may have to be carried out. It may involve observation of variations in the signals at selected points, measuring the time delay between specified events, measuring pulse widths, and so on.

Scanned with CamScanner

LANGUAGE CONSTRUCTS AND CONVENTIONS:

INTRODUCTION

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. In many respects they resemble those of c language. we discuss the Contructs and Conventions essential to the progress of the book.

Case Sensitivity

Verilog is a Case-Sensitive language like C. Thus Sense, Sense, SENSE, SENse, ---- etc---, are all treated as different entities/ quantities in Verilog.

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 intended for. All Keywords in Verilog are in small letters and require to be used as such. All Keywords appear in the text in New Courier Bold-type letters.

Examples

module signifies the beginning of a module defination.
endmodule signifies the end of a module defination.
begin signifies the beginning of a block of statement.
end signifies the end of a block of statement.
if signifies Conditional activity to be checked.
while signifies a Conditional activity to be Carried out.

IDENTIFIERS

Any program requires blocks of elements/ statement, signals etc., to the 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.

eg: clock, enable, gate-1, ----

Scanned with CamScanner

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.

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

name aa not allowed as an identifier because of the blank ("name" and "aa" are interpreted as two different identifiers)

$name not allowed as an identifiers because of the presence of "$" as the first character.

I_name not allowed as an identifiers, since the presense of "$" (or) "I" 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 "+"

WHITE SPACE CHARACTERS

Blanks(\b), tabs(\t), newlines(\n), and form feed form the white space characters in Verilog, In any design description the white space characters are included to improve readability. functionality, they seperate legal tokens. They are introduced between keywords, keyword and an identifiers, between two identifers, between identifiers and the Operator symbols, and 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 d ways. A single line Comment begins with "//" and ends with a new line - for example

```
module d_ff (Q,dp, clk) ; // this is the design description liberall
-y of a D- filp flop.
    // here Q is the output
// dp is the input and clk is the clock.
```


Scanned with CamScanner

One can incorporate multiline Comments also without resorting to "//" at every line. for such multiline Comments "/*" signifies the beginning of a Comment and "*/" its end. All lines appearing between these two symbol Combinations are together treated as a single block Comment - for example module d_ff(Q, dp, clk);

/* This module forms the design description of a d-flip flop where in Q is the Output of the flip-flop.
dp is the data input and
clk the clock input */

NUMBERS

Frequently numbers need to be specified in a design description - logic status of signal lines, buses, delay values, and numbers to be loaded in registers are examples. The numbers can be of integer type (or) real type

1. Integer Numbers
2. Real Numbers.

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 example

"this is a string"
"This string is one it with a gap in between".
"This is called a \" string \"".

when a string of ASCII characters as above is an operand in an expression, it is treated as a bit 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 example, the statement p = "numb"

assigns the binary Value

Scanned with CamScanner

0110 1110 0111 0101 0110 1101 0110 0010

top (0110 1110, 0111 0101, 0110 1101 and 0110 0010 are the 8-bit equivalents of the letters n, u, m, and b, respectively).

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 level.

Two additional levels are also possible - designed as x and z. Here x represents as 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. The following are note worthy here:

when a variable in an expression is the z state, the effect is the same as it having x value. But when an input to a gate is in the z state, it is equivalent to having the x value.

The MOS switches discussed in chapter 10 form an exception to the above. If the input to a MOS switches in the z state its output too remains at the z state.

with a few exceptions all data types in Verilog can take on all the 4 logic values (or) levels. The event is an exception to this It cannot store any value. The trireg cannot take on the z value.

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 and one of the hi-z type.

when a signal line is driven by two sources - b at 1 level with strength "strong1" and c at level 0 with strength "pull0" - a will take the value 1.

Scanned with CamScanner

If a signal line a is driven by two sources - b at 1 level with strength "pull1" and c at level 0 with strength "strong0", a will take the value 0.

If a signal line a is driven by two sources - b at 1 level with strength "strong1" and c at level 0 with strength "strong0," a will take the value x.

If a signal line a is driven by the two sources - b at 1 level with strength "weak1" and c at level 0 with strength weak signifies a gate / assigned output drive with a high source impedance, despite this, due to the higher strength level, the large signal prevails.

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 structures. Data type of each variable or signal has to be declared prior to its use. The same is valid within the concerned block (or) module.

Nets

A net signifies a connection from one circuit unit to another. Such a net carries the value of the signal it is connecting 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 output is connected to a wire and is driven by that.

tri: It represents a simple signals line a wire. Unlike the wire, a tri can be driven by more than one signal outputs.

functionally, wire and tri are identical. Distinct nomenclatures are provided for the convenience of assigning rules.

Scanned with CamScanner

Variable Data type -

A Variable is an abstraction for a storage devices. It Can be declared through the keyword reg and stores the value of a logic level: 0, 1, X and Z. A net or wire Connected to a reg takes on the Values stores in the reg and Can be used as input to other Circuit elements. But the output 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, integer, real, and realtime are the other Variable types of data; these are dealt with later.

SCALARS AND VECTORS

Enitities 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, and so on. Similarly, a group of regs stores a Value, which may be assigned, changed and handled together. The collection here is two scalar nets Connecting two Circuits, treated as a "Vector". Below figure illustrates the difference between a scalar and a Vector. wr and rd are two scalar nets Connecting two Circuit blocks Circuit 1 and Circuit 2. b is a 4-bit-wide vector net Connecting the same two blocks. b[0], b[1], b[2], and 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 and hence treated as such. The range of a Vector is specified by a set of 2 digits with a colon in between the two. The Combination is enclosed within square brackets.

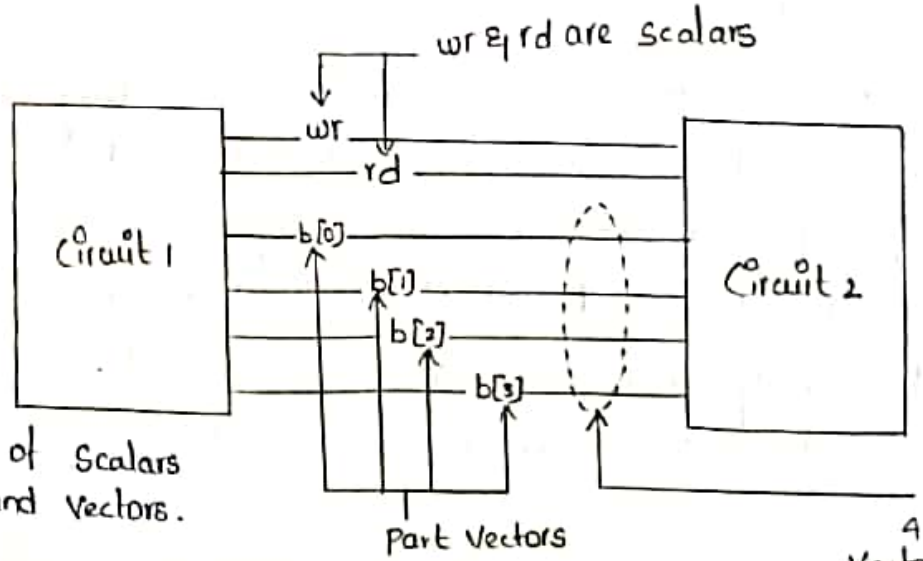


Illustration of Scalars and Vectors.

Scanned with CamScanner

```
wire [3:0] a; /* a is a four bit vector of net type; the
bit are designated as a[3], a[2], a[1] and a[0], */
reg [2:0] b; /* b is a three bit vector of reg type; the bits are
designated as b[2], b[1], and b[0]. */
reg [4:2] c; /* c is a three bit vector of reg type; the bits are
designated as c[4], c[3] and 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] and 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 and the least significant bit can be assigned specific integer values. These can also be expressions evaluating to integer constants - positive or negative. Normally vectors - nets or regs - are treated as unsigned quantities. They have to be specifically declared as "signed" if so desired.

Examples

```
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 committed at the outset. Proportionally constants, frequency-scaling levels, number of taps in digits 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, LIFO depth, and memory size as such quantities which may be committed that later. All such constants can be declared as parameters at the outset in a verilog module, and 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 computer time. The parameter values cannot be changed a runtime. However, a parameter that has

been assigned a value in a module definition can have its value changed at runtime - that is, when the module is used in a module defination have its value changed at runtime - that is, when the module is used at runtime in some other design (or) when it is tested. Such modifications are carried out through a "department/ "defparameter" statement. The parameter assignment 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 algebric expressions on the right hand side. Such expression can involve constants and other parameters declared already; for example


```
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 seperate the three operands - for example a ? b : c // Here the operators "?" and ":" together define an operation.

Scanned with CamScanner