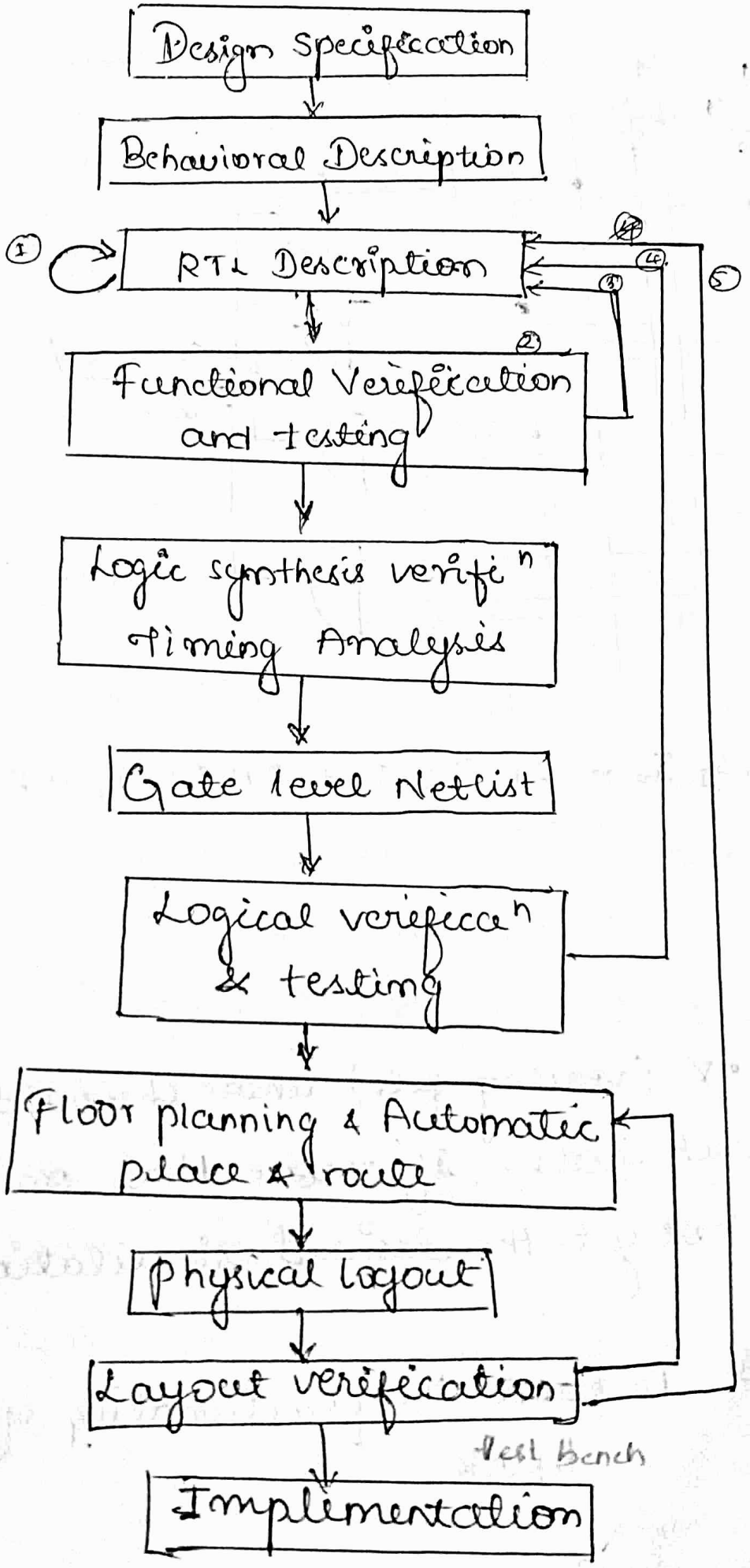
★ Design flow (Imp)



→ The Design flow shown in above figure is used by designers who use HDL's

→ In any design, specifications are written 1st.

→ Specifications: describes abstractly the functionality interface and overall architecture of the digital circuit to be designed.

→ At this point, the architects donot need to think about how they will implement the circuit

→ A Behavioral description is created to analyse the design in terms of functionality, performance, or compliance to standards and other high level issues.

→ Behavioral descriptions are often written with HDL's.

→ New EDA tools have emerged to simulate behavioral descriptions of circuits.

→ These tools combine the powerful concepts from HDL's and object oriented languages such as C++.

→ These tools can be used instead of writing behavioral descriptions in Verilog HDL.

→ The behavioral description is manually converted to an RTL description in HDL.

→ The designer has to describe the dataflow that will implement the desired digital circuit

→ From this point onward the design process is done with the assistance of EDA tool.

→ The Behavioral description is manually converted to an RTL descrip" in HDL.

(refer other page for cont)

→ From Logic synthesis tools converts the RTL description to a gate level net list.

→ A gate level netlist is the descripⁿ of the circuit in terms of gates & connecⁿ b/w them.
The logic synthesis tools ensure that the gate level netlist meets (a) timing (b) area and (c) power specifications.

* The gate level netlist is input to an automatic place and route tool, which creates a layout.
* The layout is verified & then fabricated on a chip.

→ Most digital designers concentrates on optimising the RTL description of the el circuit manually.
* After RTL descripⁿ is frozen. EDA tools are available to assist the designers in further process
* Designing at the RTL level has shrunk the design cycle time from years to a few months. and it is also possible to do many design iterations in a short period of time.

→ Behavioral synthesis tools have emerged recently & can create RTL descriptions from a behavioral or algorithmic descripⁿ of the circuit

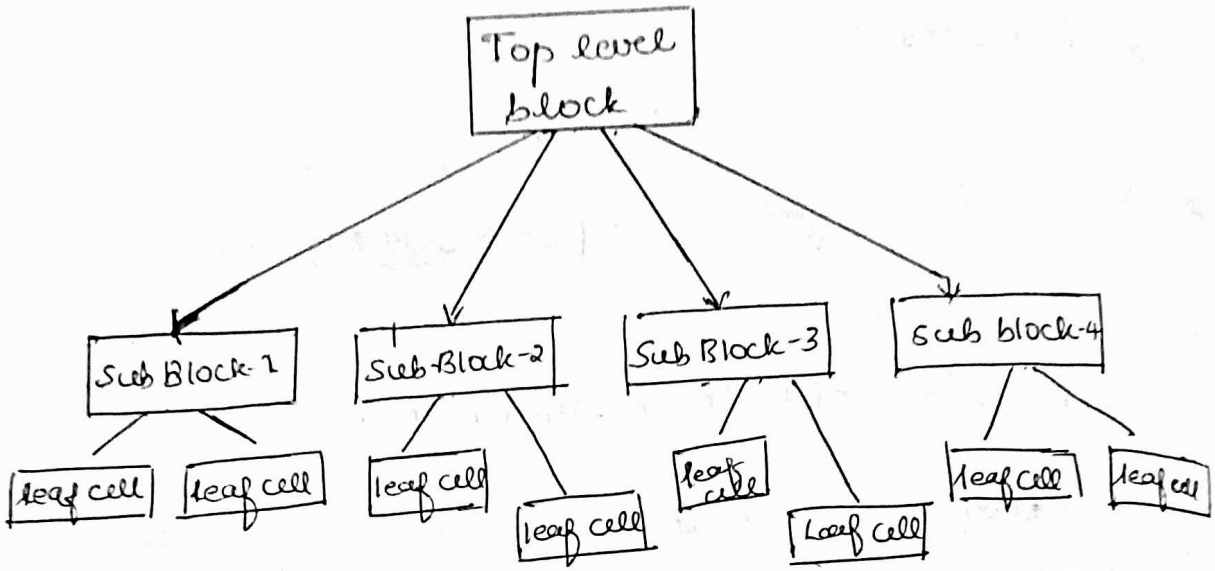
→ As these tools mature, digital circuit design will become similar to high level computer programing

→ Designers will simply implement the algorith at very abstract level.

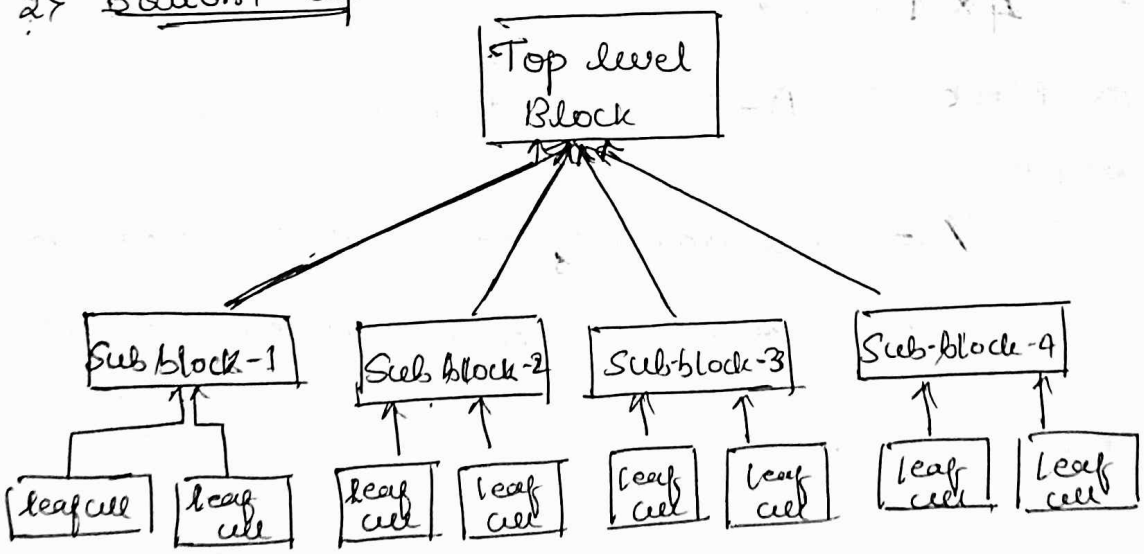
* EDA tools will help designers convert the behavioral description to a final IC chip.

→ Although EDA tools are available to automate the process and cut design cycle times, the designer is still the person who controls how

1) Top down design

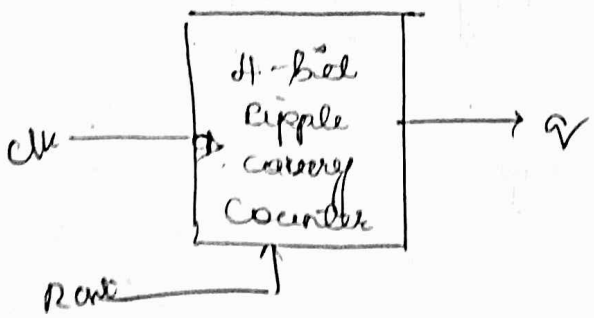


2) Bottom-up

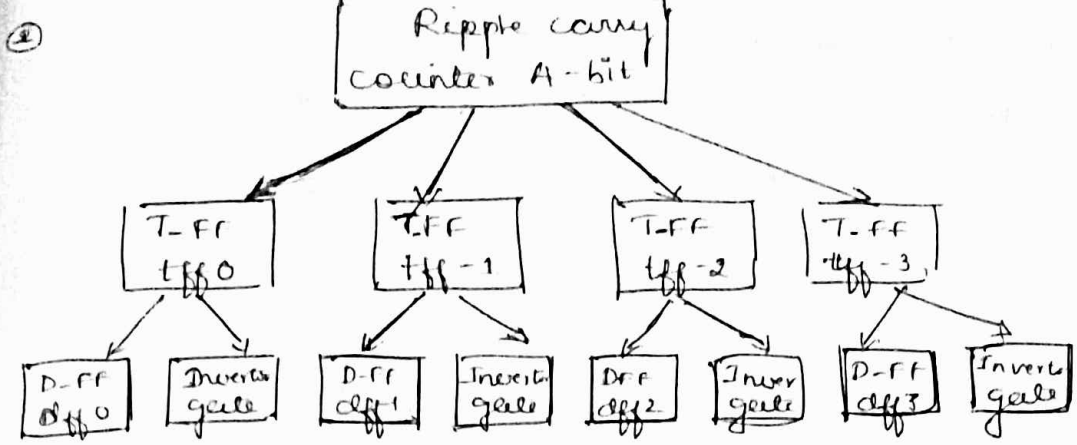


* 4 Bit Ripple carry counter code using methodology (16 M) (VTU)

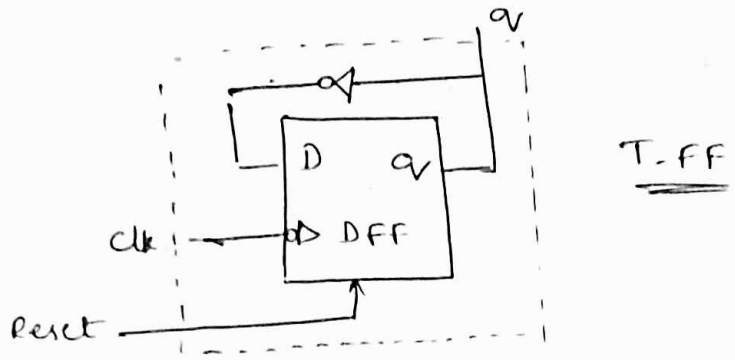
①



②

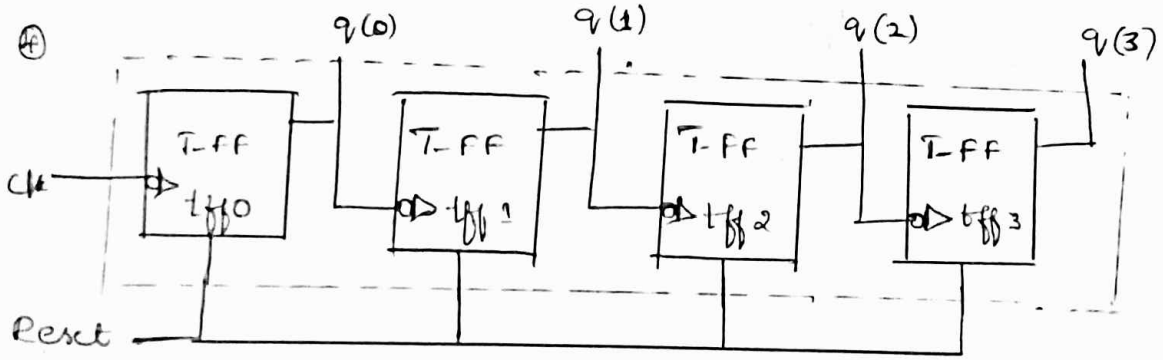


③



T-FF

④



⑤ Code

```
module ripple_carry_counter (q, clk, reset);
output [3:0] q;
input clk, reset;
T_FF tff0 (q[0], clk, reset);
T_FF tff1 (q[1], q[0], reset);
T_FF tff2 (q[2], q[1], reset);
T_FF tff3 (q[3], q[2], reset);
end module
```

instance

instantiation

```
module TFF (q, clk, reset);
  output q;
  input clk, reset;
  wire d;
DFF dff0 (q, d, clk, reset);
    not n1 (d, q); → gate level modeling
end module

module D_FF (q, d, clk, reset);   ← instance
  output q;
  input d, clk, reset;

always @(negedge clk or posedge reset)
begin
    if (reset)
      q <= 1'b0;
    else
      q <= d;
  end
endmodule
```

* Emergence of HDL's

1> Fortan, Pascal & C were used.

2> Later HDL's came into picture.

3> Verilog HDL - 1983 at Gateway design automation

4> DARPA ..

5> Logic synthesis → RTL (Register Transfer logic)

6> FPGA's → latest language

1> For a long-time programming languages such as Fortan, pascal & C were being used to describe computer programs that were sequencial in nature

2> Similarly in the DSD field designers felt the need for a standard language to describe digit circuits

3> Thus Hardware Descripⁿ languages (HDL's) came into existance.

3> Hardware languages such as Verilog HDL & VHDL became popular.

4> Verilog HDL originated in 1983 at Gateway Design automation. & later VHDL was

developed under contract from DARPA.

→ Verelog HDL & VHDL simulators were being used to simulate large digital circuits & quickly gain acceptance from designers

5) Even though HDL's were popular for logic verifications, designers had to manually translate the HDL based design into a schematic circuit with interconnection b/w the gates.

6) The advent of logic synthesis in the late 1980's change the design methodology radically.

→ Thus the designer had to specify how the data flows b/w registers & how the design processes the data.

→ Logic synthesis pushed the HDL's into the fore front of digital design.

7) ~~As~~ Designers no longer had to manually place gates to built digital circuits.

→ They could describe complex ckt's at an abstr-act level in terms of functionality & data flow by designing those circuits in HDL's.

8) Logic synthesis tools could implement the specified functionality in terms of gates & gate interconnections.

9) HDL's also began to be used for system level design.

→ HDL's were used for simulation of system boards, interconnect busses, FPGA's (Field Programmable Gate Array's) and PAL's (Program-mable Array Logic)

10) ~~PAL's~~ A common approach is to design each IC chip using an HDL & then verify

system functionality via simulation.

→ Verilog is an accepted IEEE standard. In 1995 the original IEEE-1364-1995 was approved & the latest is 1364-2001

→ Verilog HDL standards made significant improvements to the original standards.

* Trends in Verilog HDL's

→ The most popular trend currently is to design in HDL at an RTL level, because logic synthesis tools can create gate-level netlists from RTL level design. Today, RTL design continues to be very popular.

→ Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies

→ Then formal verification & assertion checking techniques have emerged.

→ Formal verification applies formal mathematical technique to verify the correctness of verilog HDL descriptions & to establish equivalently between RTL and gate level netlist.

→ New verification languages have also gained rapid acceptance. These languages combine the parallelism & hardware constructs from HDL's with object oriented nature of C++.

→ For a very high-speed & timing-critical ASIC microprocessors, the gate level netlist provided

by logic synthesis tools is not optimal. In such cases, designers often mix-gate level descriptions to be directly into the RTL description to achieve optimum results.

→ Another technique that is used for system-level design is a mixed bottom up methodology where the designers use either existing Verilog HDL modules, basic building blocks or vender supplied core blocks to quickly bring up their system simulation.

This is done to reduce development costs and compress design schedules.

Module - 2

Lexical conventions, datatypes

Lexical conventions

→ The basic lexical conventions used by Verilog HDL are similar to those in the C prog. lang.

→ Verilog contains a stream of tokens.

→ Tokens can be comments, delimiters, numbers, strings, identifiers and keywords.

→ Verilog HDL is a case sensitive language.

→ All keywords are in lowercase.

* Whitespace
* Comments
* Operators
* Number specification
  1) (a) sized & (b) unsized
  2) X or Z values
  3) Negative number
  4) Underscore characters & question marks

} Lexical conventions

* Strings
* Identifiers & keywords
* Escaped identifiers.

} LC

1) **Whitespace** :- Blank spaces (\b), tabs (\t) and newlines (\n) comprise whitespace.

→ Whitespace is ignored by Verilog except when

it separates tokens.

→ Whitespace is not ignored in strings.

2) 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.

Ex: a = b&&c ; // This is one-line com,

/* This is a multiple line
comment */

/* this is/ * an illegal * /comment */

/* this is // a legal comment */

3) Operators

→ 3 types → unary, binary & ternary

→ Unary operators precede a operand.

→ Binary operators appear b/w two operands

* Ternary operators have two seperate operators that seperate three operands.

Ex: 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 & d are operands.

Number specefication

→ 2 types → sized & unsized

→ Sized :- syntax :- <size>'<base format><number>

→ <size> is written only in decimal & specifies the no. of bits in number

→ Legal base formats are decimal ('d or 'D') hexadecimal ('h or 'H'), binary ('b or 'B') & octal ('o or 'O')

→ The no. 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 no. specifican.

Ex: 'H'b1111 // This is 4 bit binary number
12'habc // 12 bit hexadecimal no.
16'd255 // 16 bit decimal number.

Unsized Numbers

→ without a <base format>
<specifier> are decimal by default.

→ Numbers are written without <size>
Specifier have a default no. of bits that is simulator & machine-specific (must be at least 32) Ex:- 23456 //

This is a 32 bit dec no. by default.

* 'hC3 // 32 bit hexadecimal no.
* 'O21 // 32 bit octal no.

→ X or Z values

→ Verilog has two symbols for unknown & high impedence values.

→ These values are very imp for modeling real circuits. An unknown value is denoted by an X.

→ A high impedence value is denoted by z.

Ex: 12'h13X // 12 bit hex no.; & least significant bits unknown.

* 6'hx // This is 6 bit hex number.
* 32'bz // 32 bit high impedence number

Negative numbers

- Negative numbers can be specified by putting a minus sign before the size for a constantnumber.
- 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. Eg:-

-6'd3 // 8-bit negative number stored as 2's complement of 3

-6'sd3 // Used for performing si

gned integer math

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.
- The ? is used to enhance readability in the casex and casez statements where the high impedance value is a don't care condition. (? has a different meaning in the context of user-defined primitives).

- Eg:-

12'b1111_0000_1010 // Use of underline characters for readability.

4'b10?? // Equivalent of a 4'b10zz.

5.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.

Eg:-"Hello Verilog World" // is a string

"a / b" // is a string

Identifiers and Keywords

- Keywords are special identifiers reserved to define the language constructs.
- Keywords are in lowercase.
- A list of all keywords in Verilog is contained in **Appendix C**, List of Keywords, System Tasks, and Compiler Directives.
- **Appendix C. List of Keywords, System Tasks, and Compiler Directives:-**

• Section C.1. Keywords

• Section C.2. System Tasks and Functions

• Section C.3. Compiler Directives

- Identifiers are names given to objects so that they can be referenced in the design.
- Identifiers are made up of alphanumeric characters, the underscore (_), or the dollar sign($).
- Identifiers are case sensitive. Identifiers start with an alphabetic character or an underscore.
- They cannot start with a digit or a $ sign (The $ sign as the first character is reserved for system tasks). Eg:-

reg value; // reg is a keyword; value is an identifier

input clk; // input is a keyword, clk is an identifier

Escaped Identifiers

- Escaped identifiers begin with the backslash (\ ) character and end with whitespace (space, tab, or newline).
- All characters between backslash and whitespace are processed literally.
- Any printable ASCII character can be included in escaped identifiers.
- Neither the backslash nor the terminating whitespace is considered to be a part of the identifier. Eg:-

\a+b-c

my_name