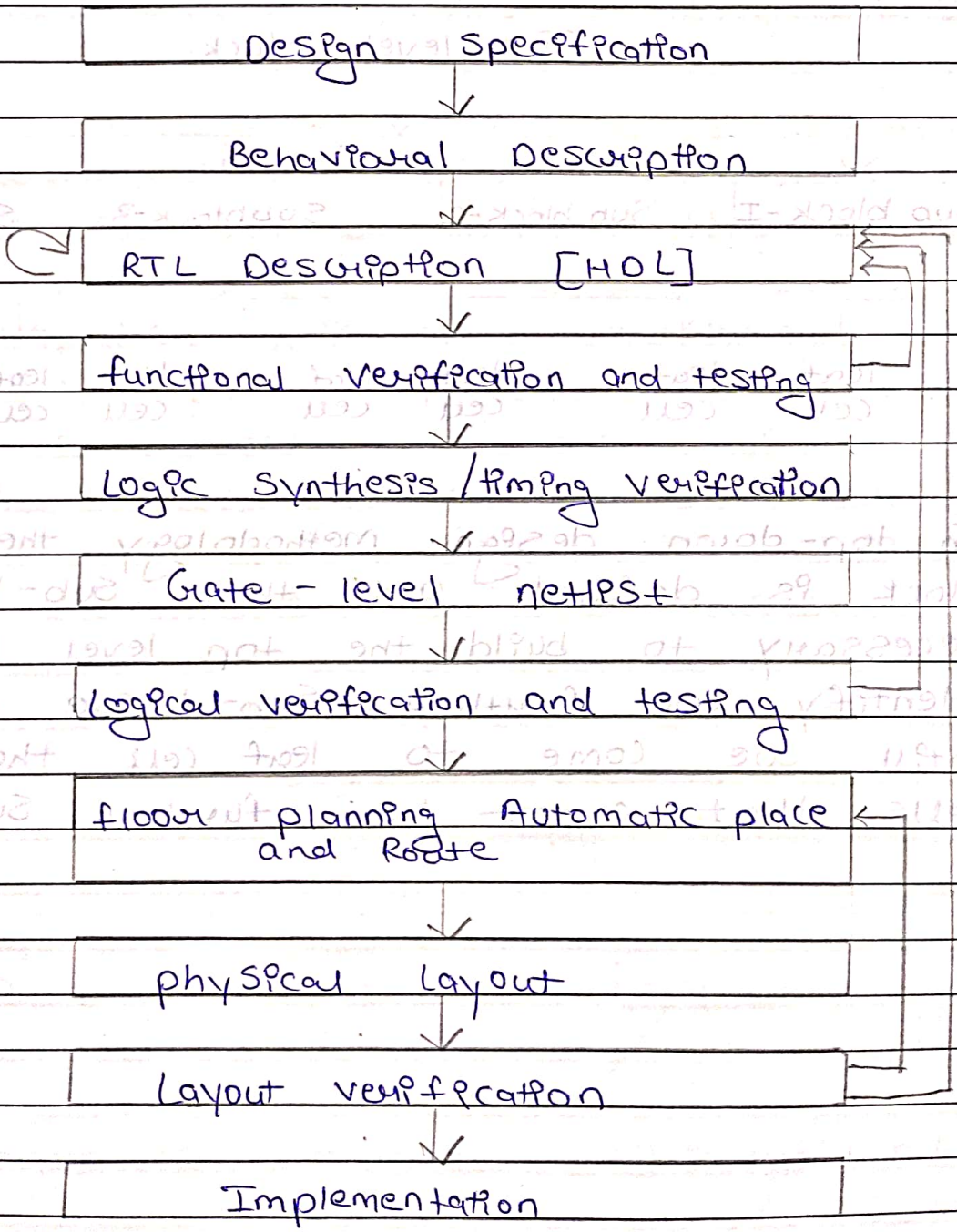
Unit : 1 :-

Digital System Design

* HDL Based Design flow :-

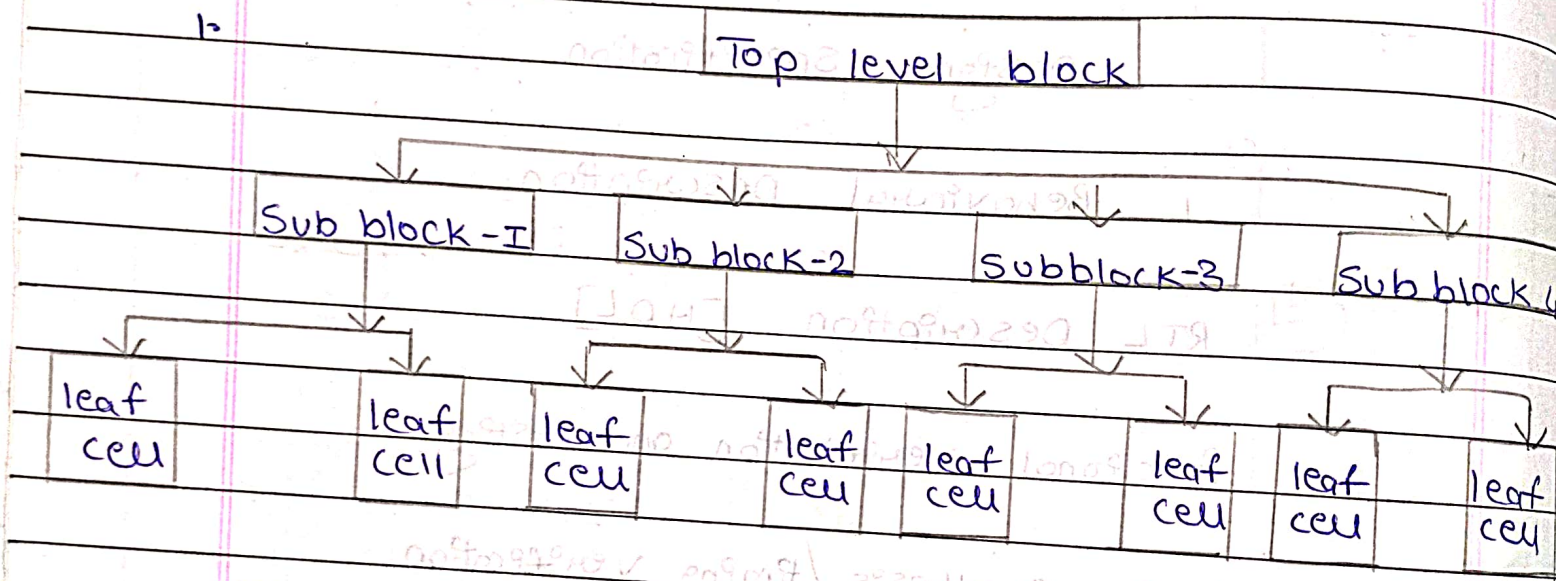


INP

★ Design Methodology :-

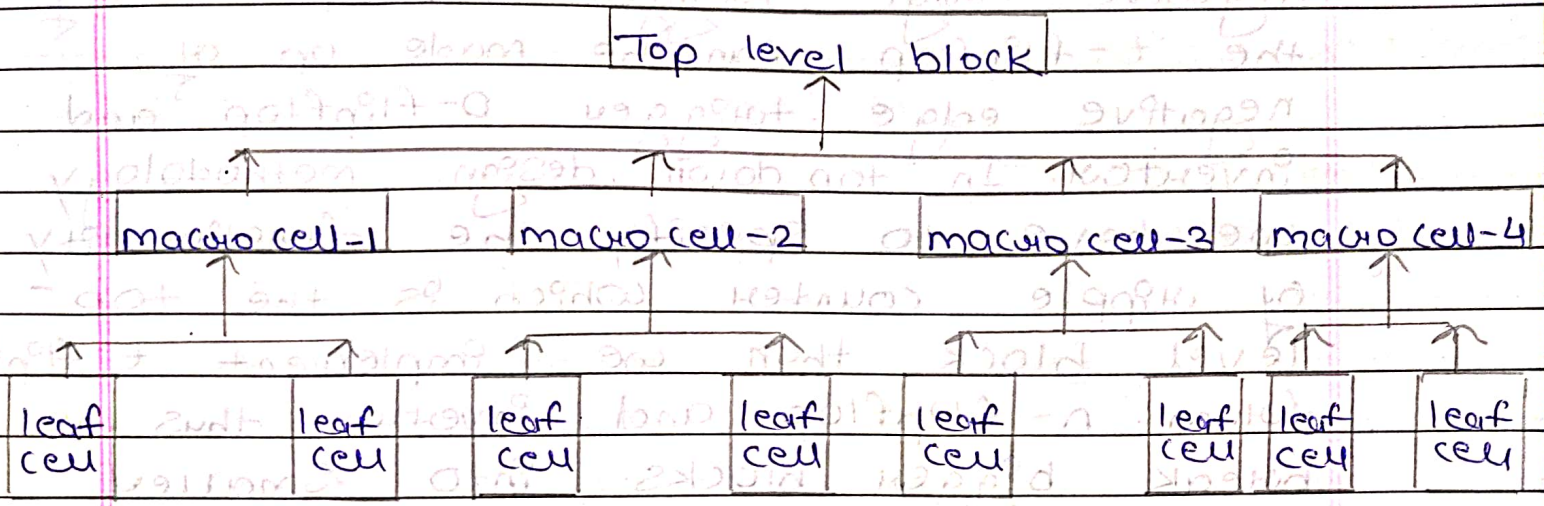
1. Top down design methodology
2. Bottom up design methodology

1.



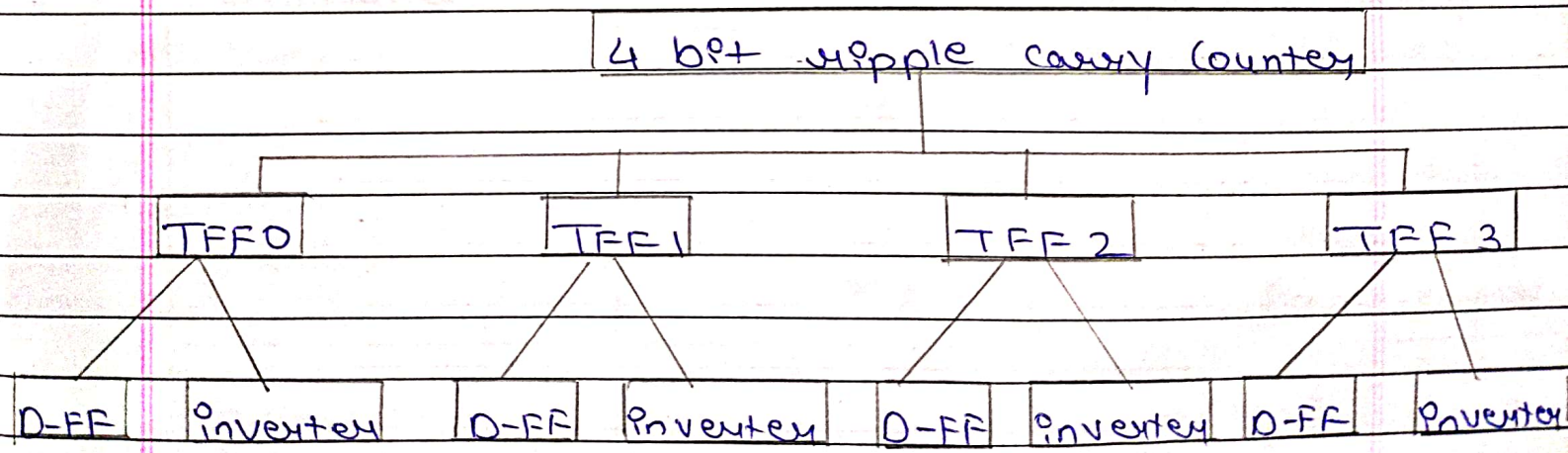
In top-down design methodology the top level block is defined and then Sub-block necessary to build the top level block ar identify. We further Sub-divede the bloc until we come to leaf cell that is the cells that cannot be further Sub-diveded

2. Bottom up design methodology :-



In bottom up design methodology we first identify the building block that are available to us we build bigger cells using these building blocks these cells are than use four higher level blocks until we build the top level block.

example — 4 bit ripple carry counter.

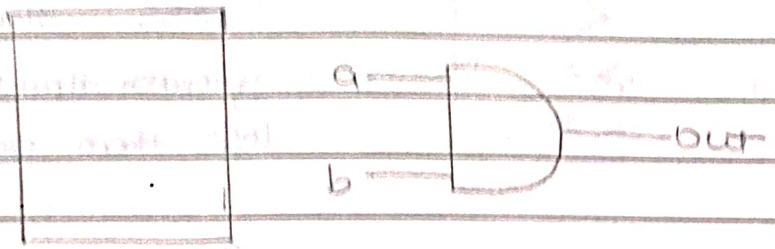


The ripple carry counter is made up of negative edge trigger flipflop. each of the t-flipflop can be made up of negative edge trigger D-flipflop and invertor. In top down design methodology we have to specify the functionality of ripple counter which is the top level block than we implement t-ff from D-flipflop and invertor. thus we break bigger blocks into smaller building blocks until we decide that we cannot break the blocks further.

A bottom up methodology ~~flows closed~~ in opposite direction that is we combine small building blocks and build bigger blocks that is D-flipflop can be build from transistor or gates.

PAGE NO:
DATE: / /

* Module :-



out = a·b

// Basic module

```
module <module name>(<module terminal list>);
.....
module internals
endmodule
```

```
module    and-gate (out, a, b);
output    out;
input     a, b;
assign    out = a && b;
end module
```

* Operator types :-

Operator types	operator symbol	operational performed
Arithmatic	*	multiply
	/	Divide
	+	add
	-	Subtract
	%	Modulus
	**	Power exponent
Logical	!	logical negation
	&&	logical and
	||	logical OR

Relational	>	greater than
	<	less than
	>=	greater than or equ
	<=	less than or equ
Equality	==	Equality
	!=	in equality
	===	Case equality
	!==	Case in equality
Bitwise	~	Bitwise negation
	&	bitwise and
	|	bitwise OR
	^	bitwise xor
	~^	bitwise xnor
Reduction	&	reduction and
	~&	reduction nand
	~|	reduction nor
Shift	>>	right shift
	<<	left shift

PAGE NO.:
DATE : / /

* level of abstraction in verilog :-

1. Behavioral level or Algorithmic level :-
This is the highest level of abstraction provided by verilog. module can be implemented in terms of desired design algorithm without concern for hardware implementation details.

ex :- 2:1 mux

```
if select = 1 then
    out = i1;
else
    out = i0;
```

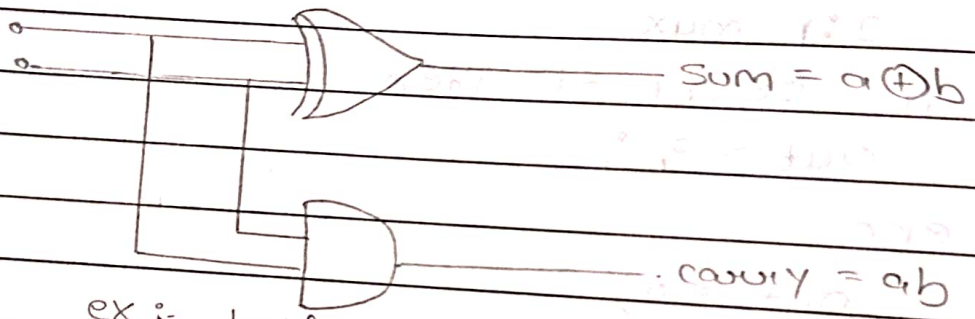
2. Data flow level :-
At this level the module is design by specifying data flow. this shows how data flows between hardware register and how the data is processed in the design

ex :- Z = not Y

Z = 9V O/P

3. Gate level:-
The module is implemented in terms of logic gates and interconnection between these logic gates. Design at these level is similar to describing a design in terms of gate level logic diagram.

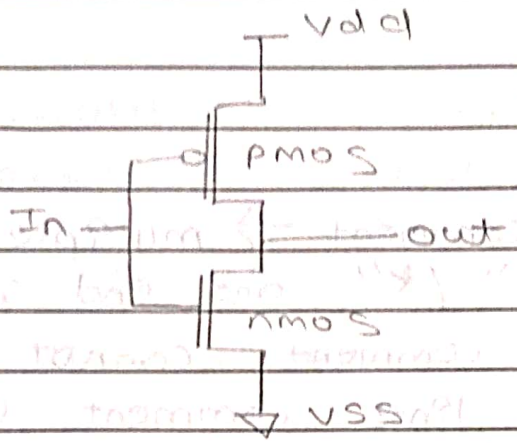
ex:- and gate, nand gate.



ex:- half adder gate level.

4. Switch level:-
This is the lowest level of abstraction provided by Verilog. A module can be implemented in terms of switches, nodes and interconnection between switches. Design at this level requires knowledge of switches.

ex:- CMOS Inverter



* Basic constructs and convention in verilog / lexical conventions :-

lexical conventions include comments, white space, numbers, strings, identifiers

1. White Space :-
   Blank Space (\b)
   tabs (\t)
   newlines (\n)

   White space is ignore by verilog but white space is not ignore in string

2. Comments :-
   Comments can be inserted in codes for readability and documentation. there are two ways to right comments.
   i One line comments — one line comments start with "//" verilog skips that point to the

end of the line.

ii. Multiple line comment → multiple line comment start with "/*" and end with "*/" multiple line comment cannot be nested however one line comment can be used in multiple line comment.

ex :- a = ~b; // ~ is unary operator
// b is operand

a = b %%C;
// this is one line comment

/* This is multiple line ----
---------- */

/* This is /* an illegal */ comment */

/* This is // a legal comment */

3. Operators :-
operators are of three types are unary, binary, ternary.

Unary operator precede an operand
Binary operator appear between two operand.

Ternary operators have two separate operators that separate three operand.

ex:- a = ~b; // ~ is unary operator
// b is operand

a = b && c; // && is binary operator
// b, c are operands

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

4. Number specification :-
There are two types of number specification in verilog
i) sized
ii) unsized

i. Sized number → sized number are represented in <size>'<base format><number>

Sized is return only in decimal and specifices the number of bits in the number.

eg :- 4'b 1111
12'b 111
0001 0001 0001

ex :- 12'habc // This is 12 bit hex n
10101011 1100 (12 bit)

16'h 255 // 16 bit decimal numbe
// 0000 0000 1111 1111
128 64 32 16 8 4 2 1

12'h dec // 12 bit hex
// 1101 1110 1100

4'b 1111
3'b 101 // 8 bit binary no 101

⁂ Unsized numbers → Numbers that are specified without the <base format> specification are decimal numbers by default. numbers that are written with <size> specification have default number bits that are simulator and machine specific that is must be atleast 32 bits

Ex :- 4567 // 32 bit decimal no. by de

234567 // this is 32 bit decimal nu

'hC3 // this is 32 bit hexadecim number C3

'021 // This is 32 bit octal no 21

5 X or Z value :-

Verilog has 2 symbols for unknown and high impedance values these values are very important for modeling real circuit. An unknown value is represented by 'x' and high impedance value is denoted by 'z'.

ex :- 12'h 13x // this is 12 bit hexadecimal number four least significant bits are x

0001 0011 xxxx

6'hx // 6 bit hex number

xxxxxx

9'o13x // 9 bit octal number

xxx 001 011

4'b01x1 // 4 bit binary number.

32'bz // 32 bit high impedance number.

6. Negative Numbers :-
Negative number can be specified by putting a minus sign before size of a constant number.

ex :- -6'b3 // this is 6 bit negative no. store as 2's complement of 3

4'd-2 // Illegal specification

-8'd3 // negative 8 bit no stored at 2's complement of 3

7. Underscore characters and Question mark :-
An underscore character is allowed anywhere in a number except the first character. Underscore character are allowed only to into readability of numbers and ignored by verilog.

A question mark is a verilog alternative of z

ex :- 12'b 1111_0000_1010 // 12bit binary no

4'b10?? // equivalent of 4'b10zz

8. Strings :-
A string is a sequence of character that are enclosed by double quotes. the restriction in string is that it must be written in single line. "Hello verilog world"

ex :- "Hello verilog world" // this is a with 19 characters

"a/b" // 3 character.

"Hello everyone" // 14 character

9. Identifiers and Keywords :-
Keywords are special identifiers reserved to define language constructs. Keywords are in lowercase.

ex :- module, input, output, endmodule, wire, assign, always.

Identifiers are the name given to object Identifiers are made up of characters, dollar, underscore sign. Identifiers are case sensitive

ex :- reg values; // reg is a keyword and values is identifier

input clk ; // input is keyword clk is identifier

~~escape~~ ~~ede~~

10. Escape Identifier :-
Escape Identifier begin with back slash character [\] and ends with white space [space, tabs, newline] All characters between back slash and white space are proce

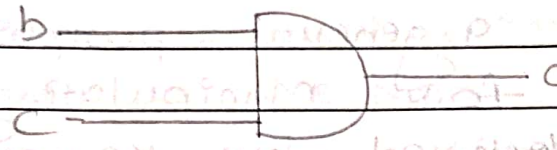
ex :- \a+b-c

** my_name **

* Data types :-
The data handle in verilog is divided in two types.
1. Net datatype
2. Variable datatype.

1. Net datatype :-
Net represents connection between hardw element in case of real circuit nets have values continously driven by the outp of the devices that they are connected nets are declared by keyword wire.

ex :-




```
wire a ;  // declare net a
wire b, c ;  // declare net b, c
wire d = 1'b0 ;  // declare net d
```

Net is not a keyword but a class of data type.

2. Variable datatype :-
A variable is an abstraction for a storage device it can be declared with a keyword REG. Variable data types are register, integer, real, time.

i. Register datatype :-
Register represent data storage element Register contains values until another value is placed on them

ex :-
```
reg reset ;
initial
begin
reset = 1'b1 ;  // Initial reset to 1
#100 reset = 1'b0 ;  // after 100 times
end                  until reset value is
                     change to zero
```

PAGE NO.:
DATE : / /

ii Integer :-
An Integer is a general purpose register data type use for manipulating quantities integer are declared by keyword integer. The default width for an integer is forced machine specific that is 32 bit. Registers declared as datatype REG store values as unsign quantities whereas integer store value as sign quantities.

Ex :- integer counter ; // General purpose variable used as counter.
initial
count = -1; // negative value is stored in counter

iii. Real :-

ex :- real delta ; // define real variable
initial
begin
delta = 4e10 ; // delta = 4×10^{10}
delta = 2.13 ;
end
integer i ;
initial
i = delta ; // i = 2

Real number constant and real register data type are declared with the keyword real. They can be specified in decimal notation that is 3.14 or scientific notation 3e6, which is 3×10^6. Real number cannot have a range declaration and there default value is zero. When a real value is assign to integer the real number is rounded or to the nearest integer.

iv. Time :-

Verilog simulation is done with respect to simulation time. A time variable is declared with keyword time, the width of time register datatype is atleast 64 bits. the system function dollar ($) time is used to get the current simulation time.

ex :- time save_sim_time ;
initial
save_sim_time = $ time ;

-x-