CBCS SCHEME

USN 2 V D 2 1 E C 4 0 3

18EC56

Fifth Semester B.E. Degree Examination, Jan./Feb. 2023

Verilog HDL

Time: 3 hrs. Max. Marks: 100

Note: ***Answer any FIVE full questions, choosing ONE full question from each module.***

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

Module-1

1 a. Explain typical design flow for designing VLSI circuit using the flow chart. **(08 Marks)**

b. i) A 4-bit ripple carry adder (Ripple – Add) contains four 1-bit full adders (FA). Define the module FA. Do not define the internals or the terminal list. Define the module Ripple – Add. Do not define the internals or the terminal list. Instantiate four full adder of the type FA in the module Ripple-Add and call them fa0, fa1, fa2, and fa3.

ii) Define the module IS, using the module/endmodule keywords. Instantiate the modules MEM, Se, Xbar and call the instances mem1, se1 and Xbar 1, respectively. You do not need to define the internals. Assume that the module IS has no terminals. **(06 Marks)**

c. What are the two styles of stimulus applications? Explain each method in brief. **(06 Marks)**

OR

2 a. Explain the trends in HDL. **(04 Marks)**

b. With a hierarchical diagram of a 4-bit ripple carry counter, explain the design hierarchy **(10 Marks)**

c. What is the difference between a module and a module instance? Explain with an example. **(06 Marks)**

Module-2

3 a. Describe different methods of connecting parts to internal signals. **(06 Marks)**

b. Explain $ display, $ monitor, $ finish and $ stop system tasks with examples. **(08 Marks)**

c. What are the basic components of a module? Explain all the components of a verilog module with a neat diagram. **(06 Marks)**

OR

4 a. Declare the following variables in verilog.

i) An 8-bit vector net called a – in

ii) A 16-bit hexadecimal unknown number with all x's

iii) A memory MEM containing 256 words of 64 bits each

iv) A parameter cache-size equal to 512. **(04 Marks)**

b. With example explain different types of lexical conventions. **(08 Marks)**

c. Write verilog description of SR latch. Also write stimulus code. **(08 Marks)**

Module-3

5 a. Write a verilog dataflow description for 4-bit full adder with carry lookahead. **(06 Marks)**

b. What would be the output of the following

a = 4'b1010 , b = 4'b1111

i) a&b (ii) a&&b (iii) &a (iv) a>>1 (v) a>>>1

(vi) $y = \{2\{a\}\}$ (vii) $a \wedge b$ (viii) $z = \{a, b\}$ **(08 Marks)**

c. What re rise, fall and Turn-off delays? How they are specified in verilog? **(06 Marks)**

OR

6 a. A full subtractor has three 1-bit inputs x, y and z (previous borrow) and two 1-bit outputs D (Difference) and B (Borrow) the logic equations are

$D = \overline{X}\overline{Y}Z + \overline{X}Y\overline{Z} + X\overline{Y}\overline{Z} + XYZ$

$B = \overline{X}Y + \overline{X}Z + YZ$ **(06 Marks)**

Write veriolog description using dataflow modeling. Instantiate the subtractor inside a stimulus block and test all possible combinations of inputs X, Y and Z.

b. Discuss the And/or and Not gates with respect to logic symbols, gate instantiation and truth table. **(06 Marks)**

c. Design AND-OR-INVERT (AOI) based 4:1 multiplexer write verilog description for the same and its stimulus. **(08 Marks)**

Module-4

7 a. Explain the following assignment statements and non-blocking assignment statements with relevant examples. **(06 Marks)**

b. Write a veriolog program for 8-to-1 multiplexer using case statement. **(08 Marks)**

c. Give the differences between tasks and functions. **(06 Marks)**

OR

8 a. Explain sequential and parallel blocks with examples. **(06 Marks)**

b. Design a negative edge-triggered D-flipflp (DUFF) with synchronous clear, active high (D-FF clears only at a negative edge of clock when clear is high). Design a clock with a period of 10 units and test the D-flipflop. **(08 Marks)**

c. Write verilog program to call a function called calc-parity which computes the parity of a 32-bit data, [31-0] Data and display odd or even parity message. **(06 Marks)**

Module-5

9 a. Write a note on :

i) Force and release

ii) Defparam statement

iii) time scale

iv) file output. **(08 Marks)**

b. Write a note on verification of gate level netlist. **(04 Marks)**

c. With a neat flow chart explain computer Aided logic synthesis process. **(08 Marks)**

OR

10 a. What is logic synthesis? **(04 Marks)**

b. Interpret the following verilog constructs after logic synthesis.

i) The assign statement

ii) The if-else statement

iii) The case statement

iv) The always statement **(10 Marks)**

c. Write RTL description for magnitude comparator. **(06 Marks)**

* * * * *

KLS Vishwanathrao Deshpande Institute of Technology

Department of Electronics and Communication Engineering

Subject : Verilog HDL Jan/Feb 2023
Scheme and Solution

Subject Code : 18EC56

Sem : 5

Staff : Prof. Shree Gowri S S

Staff - Incharge
(Shree Gowri S S)

HOD
(Dr. Mahendra M Dixit)

Dean academic
(Dr. S. Deshpand

Head of the Department
Dept. of Electronic & Communication Engg.
KLS V.D.I.T., HALIYAL (U.K.)

Shree Gowri SS -

Module - 1

1a. Explain typical design flow for designing VLSI circuit using the flow chart - 08 marks.

Sol:- design flow — 03 marks
Explaination — 05 marks

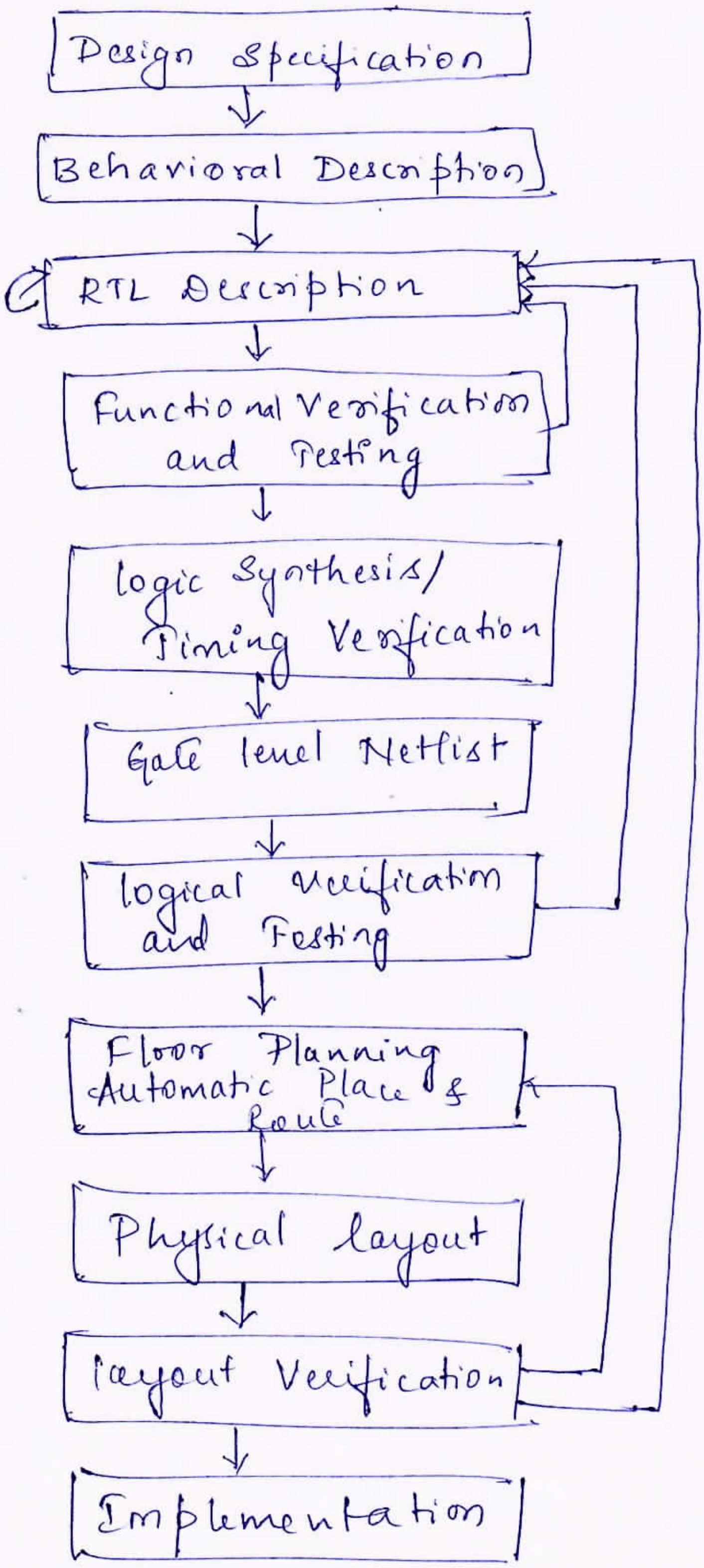

Design Specification
Behavioral Description
RTL Description
Functional Verification and Testing
logic Synthesis/ Timing Verification
Gate level Netlist
logical Verification and Testing
Floor Planning Automatic Place & Route
Physical layout
layout Verification
Implementation

fig : Typical Design flow

The typical design flow is used by designers who use HDLs

In any design, specification are written first, specificatio describe abstractly the functionality, interface & overall architecture of digital circuit to be designed. specifications identify the requirements of the design.

Behavioral Description :- A high-level behavioral description is written to analyze the design in terms of functionality, performance, compliance to standards and other high level issues. Behavioral descriptions are written in HDL.

RTL Description

- The behavioral description undergoes stepwise refinement and is converted to an RTL Description in HDL
- Register Transfer level (RTL) is a design abstraction used to describe the dataflow that will implement the desired digital circuit.

Functional Verification and Testing. Once the RTL design is ready it needs to be verified for functional correctness, with the help of EDA simulators.

Logic Synthesis

- The functionally correct RTL design is converted into hardware schematic. This step is called synthesis.
- Logic synthesis tools convert the RTL description to a gate level netlist

Gate level Netlist :- A gatelevel netlist is a description of the circuit in terms of gate and interconnections between them. Logic synthesis tool ensure that the gate level netlist meet timing, area & power specifications.

Logic Verification & Testing :- The gate level netlist is input to the physical design flow, where automatic place & route is done with the help of EDA tools.

The APR tools will select and place standard cells into rows, define input & output connections

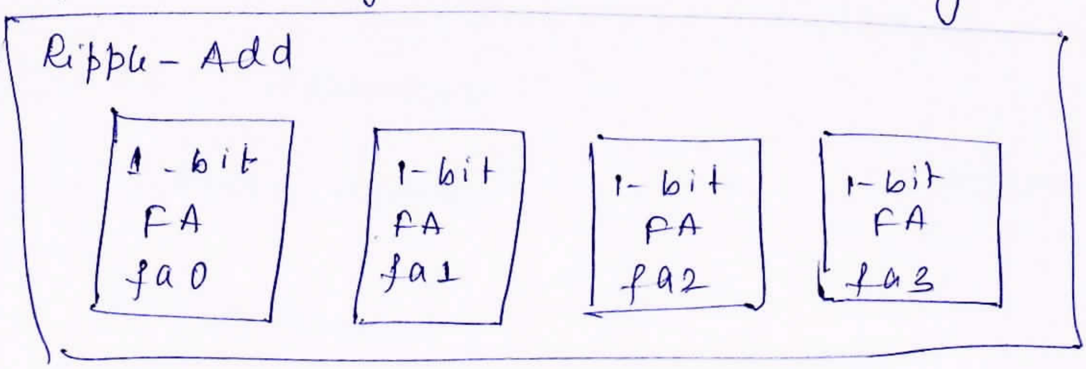
Physical layout Verification :- Once the automatic place & route is done a layout is generated which is then verified.

Implementation :- The verified design is fabricated on a chip.

1b

i) A 4-bit ripple carry adder (Ripple - Add) contains four 1-bit full adders (FA). Define the module FA. Do not define the internal or terminal list. Define the module Rippe - Add. Do not define the internal or terminal list. Instantiate four full-adder of the type FA in the module Ripple Add and call them fa0, fa1, fa2, fa3

Sol:- Scheme -> Defining module and instantiating carries — 3 marks



```
module  FA  ();

 - - .
 -   - .

<Internals>

 -  - .
 -  - .

endmodule

module RippleAdd ();

  - - .
  -  - .

 FA   fa0 ();
 FA   fa1 ();
 FA   fa2 ();
 FA   fa3 ();

 - - .

endmodule
```

ii) Define the module IS, using the module/endmodule keywords. Instantiate the module MEM, Se, Xbar and call the instances mem1, se1, Xbar1, respectively. You do not need to define the internals. Assume the module IS has no ~~internals~~ terminals.

Sol:- Defining the module and instantiating carries - 3marks.

```
module  IS ();

 - - .

MEN   mem1 ();
 Se   se1 ();
Xbar   Xbar1 ();

 - - .

endmodule
```

```
module  MEM();

 - - .
 -

endmodule

module se();

 - - .
 -  -

endmodule

module Xbar();

 - - -
 - - .

endmodule
```

1c what are the two types of stimulus application? Explain each method in brief — 06 Marks.

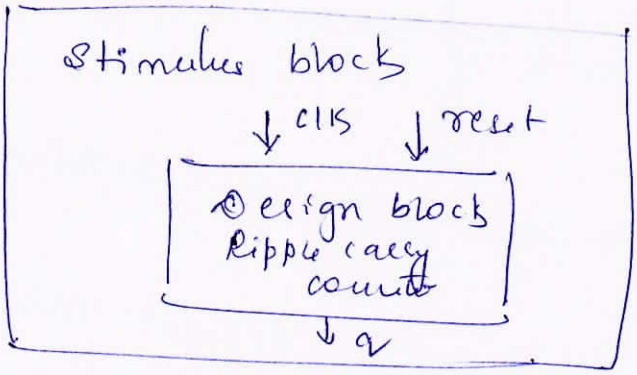
Sol:- Types of stimulus application – 02 Marks
Each method – 02 x 02 marks
04 Marks.

There are two types of stimulus application

1) stimulus block instantiate design block
2) Top-level Dummy block instantiates both design block & Stimulus block.

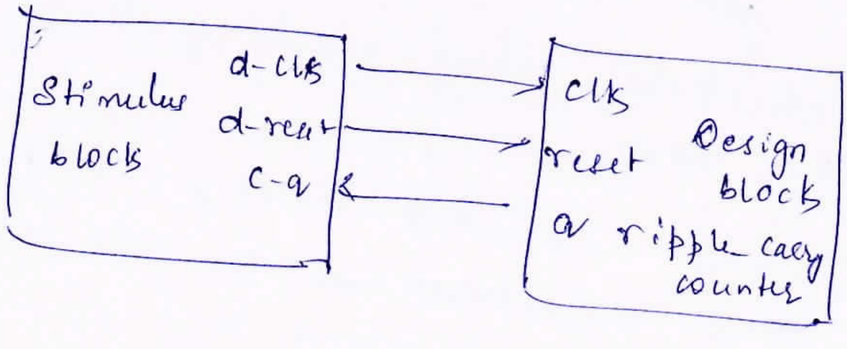
1) first: Stimulus block instantiates design block

In this style stimulus block instantiates design block and directly drives the signal in the design block. stimulus block acts as top level block



2) Second style:- Dummy block instantiates both stimulus and design block

Top-level block



Stimulus block interacts with design blocks through the interface

The stimulus block drives the signal d-clk and d-reset which are connected to clk, reset in the design block i.e d-clk and d-reset are o/p's from stimulus block and acts as output to design block

where q acts as output of design block and acts as input to stimulus block.

OR

2a. Explain the trends in HDL — 04 Marks

Sol:- Trends in HDL (4 points) — 4 marks.

The most popular trend currently is to design in HDL at an RTL level, since logic synthesis tool can create gate level netlists from RTL level design. Verilog HDL is constantly being enhanced to meet the needs of new verification methodologies.

- Formal verification and assertion checking techniques have emerged
- Formal verification applies formal mathematical techniques to verify the correctness of verilog HDL description and to establish equivalency between RTL and gate-level netlist
- Assertion checkers allow checking to be embedded in the RTL code.
- For very high speed & timing - critical circuits like microprocessor, the gate level netlist provided by logic synthesis tool is not optimal

In such cases designers mix gate level description directly into RTL description to achieve optimum result.

Qb with a hierarchical diagram of a 4-bit ripple carry counter, explain the design hierarchy — 10 Mark

Sol:- hierarchical diagram of a 4-bit ripple carry counter – 04 Mark
explaination carries – 06 Mark
10 Marks

Top-level Block = 4-bit Ripple counter

Sub-block = 4- Number of T flip-flop

Cells = 1 ~~D~~ flip-flop & Inverter

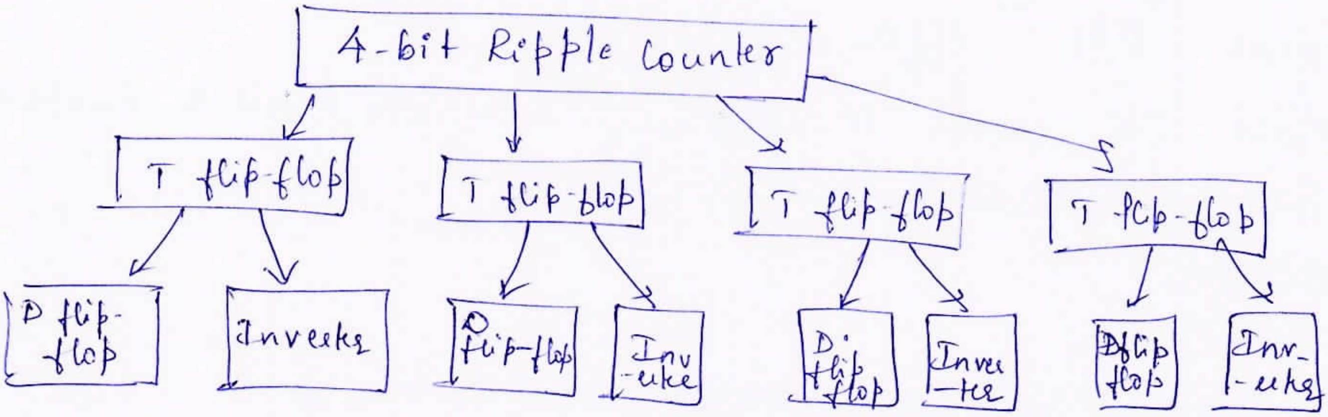
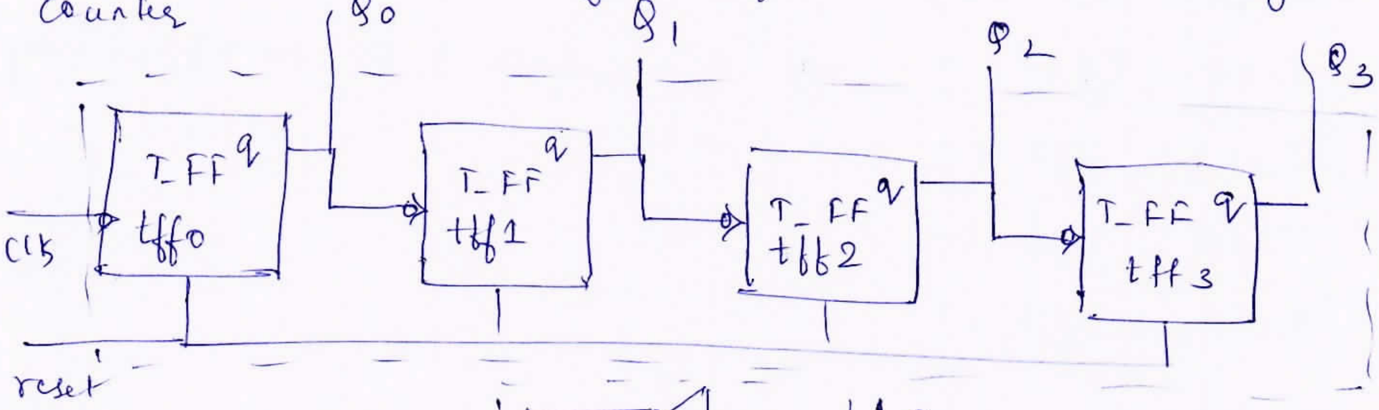
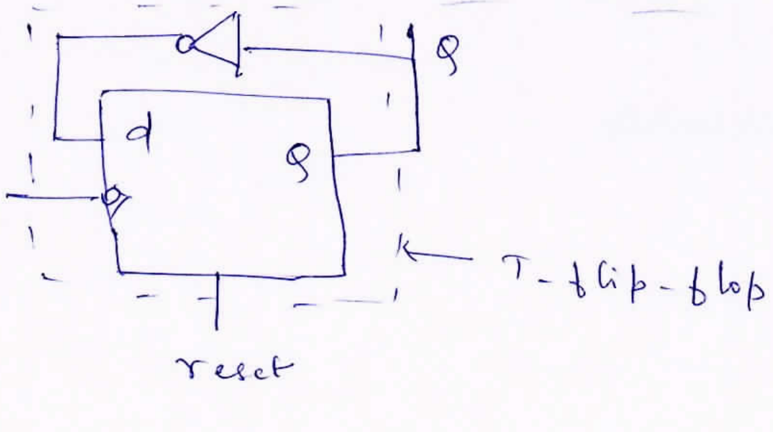


fig: hierarchical diagram of a 4-bit ripple carry Counter



Each T flip-flop



Hierarchical modelling concepts ~~to verilog~~ is related to verilog. verilog provides module, which is the basic building block in verilog. One module calling another module is called instantiation. Object or another module is called instance.

Therefore for the hierarchical model 4-bit ripple carry counter, 4-bit ripple carry counter is the Top-level module or Top-level block.

T flip flop is a subblocks, which is another module in the top-level block (4-bit ripple carry counter)

4 - T flip-flop are instantiated with different instance name like tff0, tff1, tff2 & tff3

where in each T-flip flop module / block instantiates another module called D-FF

~~Example~~
Program :-

```
module ripple-carry-counter (q, clk, reset);
input clk, reset;
output [3:0] q;
T_FF  tff0 ( );  // Instantiate T_FF in ripple-carry-counter
T_FF  tff1 ( );
T_FF  tff2 ( );
T_FF  tff3 ( );
- - :
endmodule
```

```
module T_FF ( );

- -
- -
D-FF dff0 ( );  // instantiate D-FF module in T_FF
- - -
- -
endmodule

module D-FF ( );

.. ..
- -
endmodule
```

2c what is the difference between a module and module instance? explain with an example - 06 Marks.

Sol:- Difference b/wn module and module instance - 04 marks
Example - 02 marks
06 marks

Module provide a template from which we can create actual objects

when module is invoked, verilog creates a unique object from the template

Each object has its own name, variables, parameters and I/O interface

The process of creating objects from a module template is called instantiation and the objects are called as an instances.

Example

```
module ripple-carry-counter (q, clk, reset);
input clk, reset;
output [3:0] q;
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);
endmodule

module T_FF (q, clk, reset);
input  clk, reset;
output  q;
wire  d;
D-FF  dff0 (q, d, clk, reset);
not  n1 (d, q);
endmodule

module  D-FF (q, d, clk, reset);
input  d, clk, reset;
output  q;
reg  q;
always @ (posedge reset or posedge clk)
if (reset)
    q <= 1'b0;
else
    q <= d;
endmodule
```

Module - 2

a. Describe different methods of connecting parts to internal signals — 06 Marks

Sol:- Different methods of making connection between signals — 03 × 2 = 06 Marks

There are two methods of making connection between signals specified in the module instantiation & the ports in module definition.

1) Connecting by ordered list

The signals to be connected must appear in the module instantiation in the same order as the ports in the port list in the module definition.

Ex :-

```
module Top;

reg [3:0] A, B;
reg C_in;
wire [3:0] sum;
wire C-out;
fulladd4 fa-ordered (SUM, C-OUT, A, B, C-in);

- - -
- - -

endmodule

module fulladd4 (sum, c-out, a, b, c_in);
output [3:0] sum;
output c-out;
input [3:0] a, b;
input c-in;

- - -
-
  <module internals>
- -
-
endmodule
```

2) Connecting ports by name :-

for large designs where modules having 50 ports for example, remembering the order of the ports in the module definition is impractical & error-prone. Verilog provides the capability to connect external signals to ports by the port names, rather than by position.

we can specify the port connections in any order as long as the port name in the module definition Correctly matches the external signal.

Ex:-

```
fulladd4  fa-byname( .c-out(C-OUT), .sum(SUM),
                     .b(B), .c-in(C-IN), .a(A));
```

3b. Explain $ display, $ monitor, $ finish and $ stop system tasks with examples — 08 marks.

Sol:- Each system task carries — 02 Marks
— 02 × 4 = 08 Marks.

1> $ display : $ display system task is used for displaying value of variables or strings or expressions.

Syntax:- $display(p1, p2, p3, ... pn);

$ display produces a new line without any arguments.

Ex:- $display("Hello Verilog World");

-- Hello Verilog world.

2> $ monitor :- $ monitor provides a mechanism to monitor a signal when its value changes

Syntax:- $monitor (p1, p2, p3, .. pn);

where p1, p2, p3... pn can be variables, signal names or quoted strings.

$ monitor continuously monitors the values of variables or signals

3) **$ finish** :- $ finish task terminates the simulation

Ex:-
```
initial
  begin
  clk = 0;
  reset = 1;
  #900 $finish;   # terminate the simulation at 900ns
  end
```

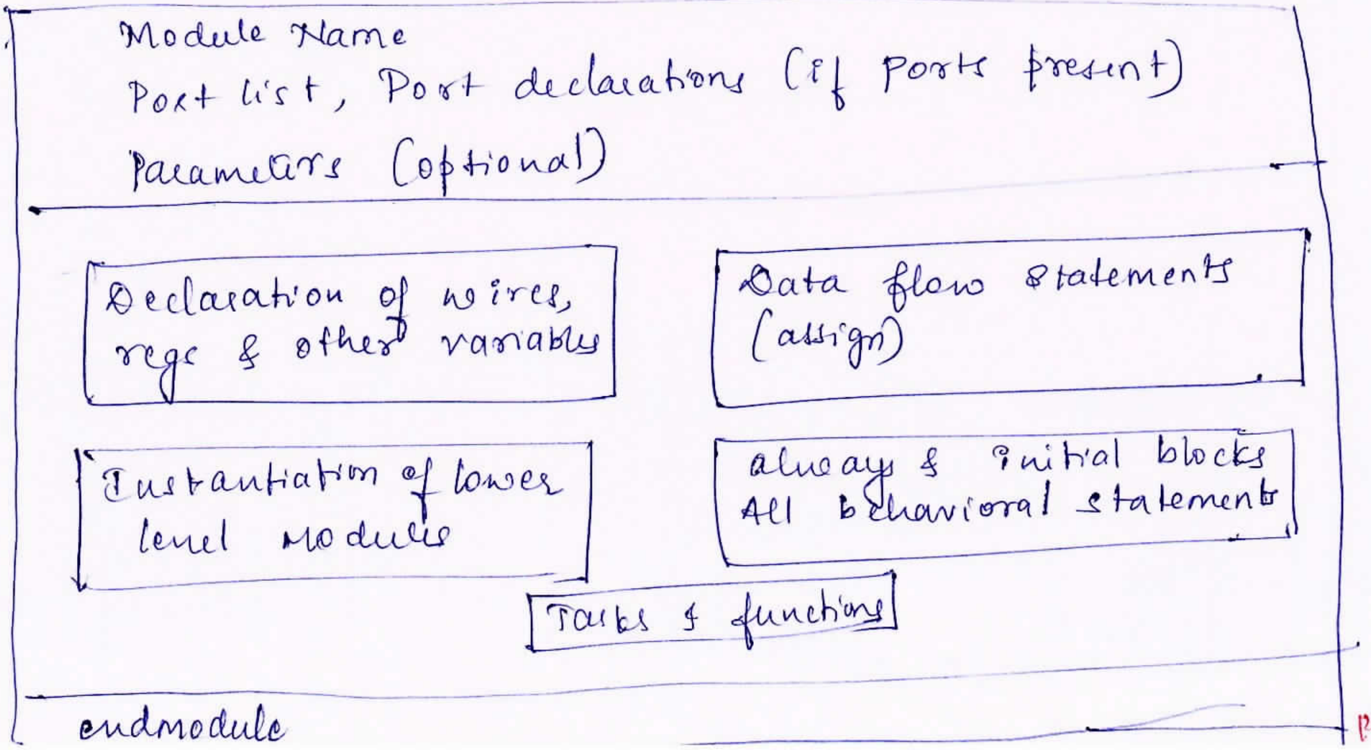
4) **$ stop** :- $ stop task puts the simulation in an interactive mode. The $ stop task is used whenever the designer wants to suspend the simulation & examine the values of the signals in the design.

Ex:- `#100 $ stop;`

3c what are the basic components of a module? Explain all the components of a verilog module with a neat diagram — 06 Marks.

Sol:- Basic components of verilog module carries - 02 marks
Explaination of all the components with diagram - 04 marks

06 marks.



Module Name
Port list, Port declarations (if Ports present)
Parameters (optional)
Declaration of wires, regs & other variables
Data flow statements (assign)
Instantiation of lower level Modules
always & initial blocks All behavioral statements
Tasks & functions
endmodule

module definition :- begins with keyword module

The module name, port list, port declarations, optional parameters must come in a module definition.

Port list and port declarations are present only if the module has any ports to interact with the external environment.

The five components within a module are

1> variable declarations

2> data flow statements

3> instantiation of lower modules

4> behavioral blocks

5> tasks or functions

These components can be in any order and at any place in the module definition.

endmodule statement must alway come last in a module definition.

OR

4a:- Declare the following variables in verilog - 04 marks

i> an 8-bit vector net called a_in

ii> A 16-bit hexadecimal unknown number with all x's

iii> A memory MEM containing 256 words of 64 bits each.

iv> A parameter cache-size equal to 512.

Sol:- Declaration of variables in verilog 1 X 4 = 04 marks

i> wire [7:0] a_in

ii> 16'hx.

iii> reg [63:0] mem [0:255]

iv> parameter cache-size=512;

4b. with example explain different types of lexical conventions — 08 marks

Sol:- Atleast 04 types of lexical convention — 04×02 = 08 marks

1. whitespace :-

Blank spaces (\b), tabs (\t), newlines (\n) comprise the white space. white space is ignored by verilog except when it separates tokens. white space is not ignored in strings.

2) Comments :- comments can be inserted in the code or program for readability and documentation. There are two ways of writing comments

1) Single or one - line comment :- starts with //

2) Multiple line comment :- start with /* & ends with */

3) Operators :- operators are of three types Unary, binary & ternary.

Ex:- a = ~b; // unary operator

a = b && c; // && is binary operator

a = b ? c : d; // ?: is ternary operator.

4) String :- String is a sequence of characters that are enclosed by double quotes

Ex:- "Hello verilog world" // is a string

4c. Write verilog description of SR latch. Also write stimulus code — 08 marks

Sol:- Verilog description of SR latch — 04 marks
Stimulus code — 04 marks
08 marks

```
module SR_latch (Q, Qbar, Sbar, Rbar);
Output Q, Qbar;
input Sbar, Rbar;
nand n1 (Q, Sbar, Qbar);
nand n2 (Qbar, Rbar, Q);
endmodule

// Stimulus code
module Top;
wire q, qbar;
reg set, reset;
SR_latch m1 (q, qbar, ~set, ~reset);
initial
begin
  $monitor($time, "set = %b; reset = %b, q = %b\n",
                  set, reset, q);

   set = 0; reset = 0;
   #5 reset = 1;
   #5 reset = 0;
   #5 set = 1;
  end
 endmodule
```

Module - 3

5.a. Write a verilog data flow description for 4-bit full adder with carry lookahead — 06 Marks

Sol:- data flow description for 4-bit full adder with carry lookahead — 06 marks

```
module fulladd4 (sum, c-out, a, b, c-in);
output [3:0] sum;
output c-out;
input [3:0] a, b;
input c-in;
wire p0, g0, p1, g1, p2, g2, p3, g3;
wire c4, c3, c2, c1;
assign  p0 = a[0] ^ b[0];
        p1 = a[1] ^ b[1],
        p2 = a[2] ^ b[2],
        p3 = a[3] ^ b[3];
assign  g0 = a[0] & b[0],
        g1 = a[1] & b[1],
        g2 = a[2] & b[2],
        g3 = a[3] & b[3];
assign c1 = g0 | (p0 & c-in),
       c2 = g1 | (p1 & g0) | (p1 & p0 & c-in).
       c3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & c-in),
       c4 = g3 | (p3 & g2) | (p3 & p2 & g1) | (p3 & p2 & p1 & g0) ]
                   (p3 & p2 & p1 & p0 & c-in);
assign sum[0] = p0 ^ c-in;
       sum[1] = p1 ^ c1;
       sum[2] = p2 ^ c2,
       sum[3] = p3 ^ c3;
assign c-out = c4;
endmodule
```

5b What would be the output of the following — 08 Marks

a = 4'b1010 , b = 4'b1111

(i) a & b (ii) a && b (iii) &a (iv) a>>1

(v) a>>>1 (vi) y = {2{a}} (vii) a^b

(viii) z = {a, b}

Sol:- output of each logic operations carries — 1 marks

— 1 × 08 = 08 ma

i> a & b

```
a = 1010
b = 1111
    ----
    1010
```

ii> a && b True

iii> &a

```
a = 1010
    1 & 0 ~~1~~ & 1 & 0
    ←
a = 1'b0
```

iv> a>>1

a = 1010

after right shift a by 1

a = 0101

v> a>>>1

vi> y = {2{a}}

y = 8'b10101010

vii> a ^ b

```
a = 1010
b = 1111
    ----
    0101
```

viii) z = {a, b}

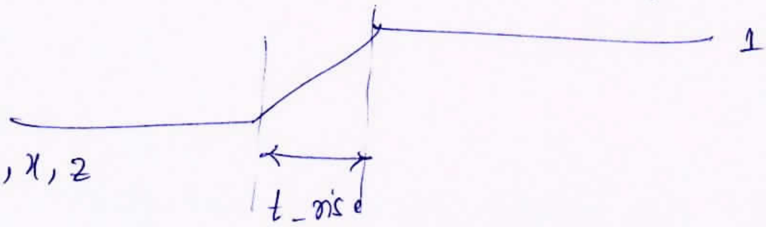
z = 8'b1010 1111

5c. what is rise, fall and Turn-off delays ? How they are specified in verilog Ans:

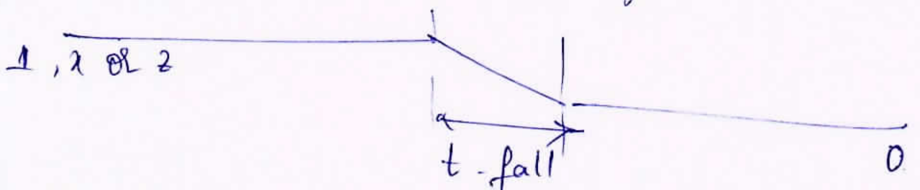
— 06 Marks

Sol:- Delay specification & explaination - 03 x 2 = 06 marks

Rise delay:- The rise delay is associated with a gate output transition to a 1 from another value



fall delay :- The fall delay is associated with a gate output transition to a 0 from another value



Turn-off delay :- The turn-off delay is associated with a gate output transition to the high impedance value (z) from another value

There are three types of delay specification

1) one - delay
2) two - delay
3) three - delay

1) One - delay :- Ex:- and #(delay-time) a1(out, i1, i2);
and #(5) a1(out, i1, i2); // rise = 5
fall = 5
turnoff = 5

2) Two - delay :- Ex:- and #(rise_v, fall_v) a1(out, i1, i2);
and #(2, 3) a1(out, i1, i2); // rise = 2
fall = 3

3) Three - delay Ex:- and #(rise_v, fall_v, turn) a1(out, i1, i2);
and #(2, 3, 4) a1(out, i1, i2, i3);
rise = 2, fall = 3, turn off = 4

OR

Qui.

6a A full subtractor has three 1-bit inputs, x, y and z and two 1-bit outputs D and B the logic equations are

$$D = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$

$$B = \bar{x}y + \bar{x}z + yz$$

write verilog description using dataflow modeling. Instantiate a full subtractor inside stimulus block and test all possible combinations of inputs X, Y and Z — 06 Marks

Sol:- Verilog description for full subtractor using dataflow modeling carries - 03 marks

test bench - 03 marks

06 marks

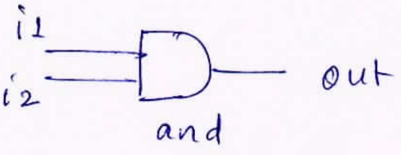
```
module Subtractor (x, y, z, D, B);
input x, y, z;
output D, B;
assign D = ((~x & ~y & z) | (~x & y ~z) | (x & ~y & ~z) |
                                (x & y & z));
assign B = ((~x & y) | (~x & z) | (y & z));
endmodule

module testBench;
 wire d, b;
 reg  X, Y, Z;
 Subtractor  sub (X, Y, Z, d, b);
 initial
  begin
    X = 0; Y = 0; Z = 0;
 #100 X = 0; Y = 0; Z = 1;
  #100  X = 0; Y = 1; Z = 0;
 #100  X = 0;  Y = 1; Z = 1;
 #100  X = 1;  Y = 0; Z = 0;
 # 101  X = 1;  Y = 0; Z = 1;
 # 110  X = 1;  Y = 1; Z = 0;
 # 111  X = 1;  Y = 1;  Z = 1;
end  endmodule
```

6b. Discuss And/or and Not gates with respect to logic Symbols, gate instantiation & truth table — 06 Marks

Sol:- each logic gate carries — 02 Marks
02 × 3 ~~AA~~ = 06 marks

AND gate

Symbol :-



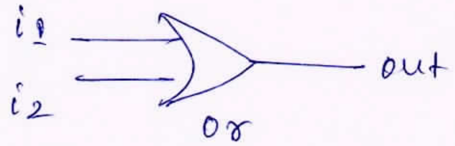
Gate instantiation and a1 (out, i1, i2);

Truth table :

and	0	1	X	Z
0	0	0	0	0
1	0	1	X	Z
X	0	X	X	X
Z	0	Z	X	X

OR gate :-

Symbol :-



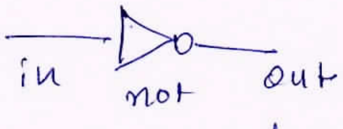
Gate instantiation :- or or1 (out, i1, i2);

Truth table

or	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

Not gate

Symbol :-



Gate instantiation
not n1 (out, in);

Truth table :-

not	in	out
	0	1
	1	0
	X	X
	Z	X

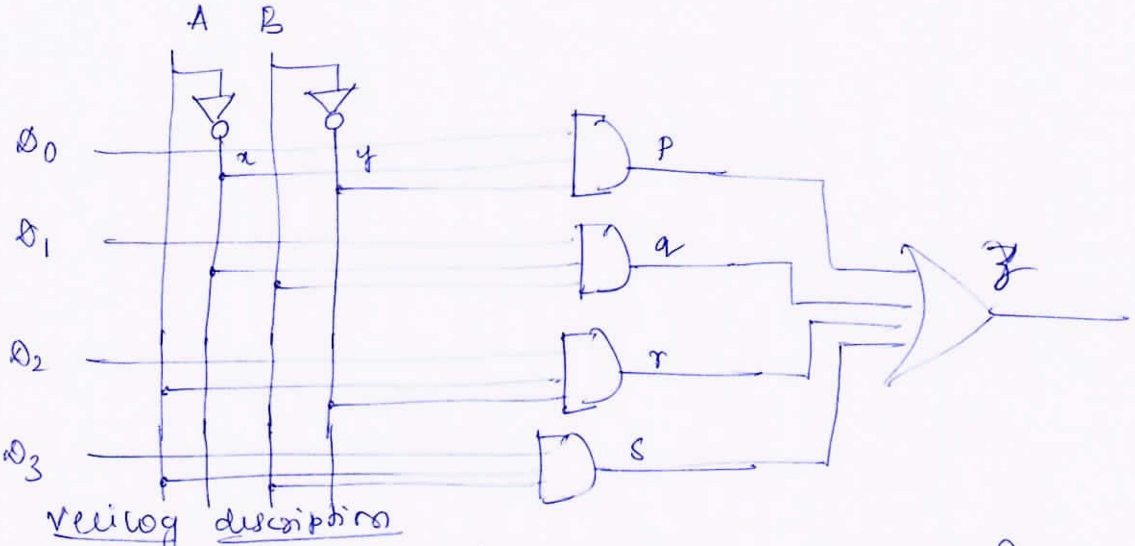
6c Design AND - OR - Invert based 4:1 multiplexer write verilog description for the same & its stimulus - 08 marks

Sd:- Designing of AND - OR - Invert based 4:1 mux - 02 marks
verilog description for 4:1 mux — 03 marks
Stimulus for 4:1 mux — 03 marks
08 marks

4:1 Mux

A	B	D_0	D_1	D_2	D_3	Z
0	0	1	0	1	0	$D_0 = 1$
0	1	1	0	1	0	$D_1 = 0$
1	0	1	0	1	0	$D_2 = 1$
1	1	1	0	1	0	$D_3 = 0$

$$Z = D_0 \bar{A} \bar{B} + D_1 \bar{A} B + D_2 A \bar{B} + D_3 A B$$



verilog description

```
module   mux4to1 ( A, B, D0, D1, D2, D3, Y);
input   A, B, D0, D1, D2, D3;
output   Z;
wire   x, y, P, q, r, S; →   not (x, A);
                             not (y, B);
and   a1 (P, D0, x, y);
and   a2 (q, D1, x, B);
and   a3 (r, D2, A, y);
and   a4 (S, D3, A, B);
y   o1 (z, P, q, r, s);
endmodule
```

Stimulus code

```
module tb;
  wire z;
  reg  a, b, d0, d1, d2, d3;
  mux4to1  MUX [a, b, d0, d1, d2, d3]);
  initial
   begin
   a = 1'b0;
   b = 1'b0;
   d0 = 1'b0;
   d1 = 1'b0;
   d2 = 1'b1;
   d3 = 1'b0;
 #100;
    ...
    - -
   -  -
   end
   endmodule
```

Module - 4

7a Explain the following assignment statements and non-blocking assignment statements with relevant examples — 06 Marks

Sol:- assignment statement with ex – 03 Marks
non-blocking assignment statement with ex - 03 Marks
06 Marks

Blocking assignment statements are executed in the order they are specified in a sequential block.
A blocking assignment will not block execution of statements that follow in a parallel block.
The = operator is used to specify blocking assignments

Ex:-

```
 reg x, y, z;
 reg [15:0] reg_a, reg_b;
 integer count;
 initial
 begin x = 0; y = 1; z = 1;
     count = 0;
     reg_a = 16'b0; reg_b = reg_a;
    #15 reg_a [2] = 1'b1;
    #10 reg-b [15:13] = {x, y, z}
    count = count +1;
end
```

Non blocking assignments

Non blocking assignments allow scheduling of assignments without blocking execution of the statement that follows in a sequential block A <= operator is used to specify non-blocking assignments.

Ex:-

```
reg x, y, z;
reg [15:0] reg-a, reg-b;
integer count;
initial
begin x = 0; y = 1; z = 1;
    count = 0;
    reg-a = 16'b0; reg-b = reg-a;
    reg-a [2] <= 15 1'b1;
    reg-b [15:13] <= #10 {x, y, z}
    Count <= count +1;
end
```

16. Write verilog program for 8-to-1 multiplexer using case statement -08 marks

Sol:- verilog code for 8:1 Mux using case statement carries - 08 Marks

```
module mux8to1 ( s0, s1, s2, d, y);
input s0, s1, s2;
input [7:0] d;
output y;
input en;
always @ ( d, s0, s1, s2)
begin
 if (en = 1)
 case { s0, s1, s2}
 3'd0: y = d[0];
 3'd1; y = d[1];
 3'd2; y = d[2];
```

```
     3'd3: y = d[3];
     3'd4; y = d[4];
     3'd5: y = d[5];
     3'd6 : y = d[6];
     3'd7 : y = d[7];
    default : y = 1'bx;
   endcase
   else
    y = 1'bz;
  end
endmodule
```

7c Give the difference between taske and functions - 06 Marks

Sol:- listing out the differences b/wn task & function atleast 6 - carries - 06 marks

Functions	Tasks
1. A function can enable another function but not another task	1. A task can enable other tasks & functions
2. Function always execute in 0 simulation time	2. Tasks may execute in non-zero simulation time
3. Functions must not contain any delay, event or timing control statements	3. Tasks may contain delay, event or Timing control statements
4. Functions must have atleast one input argument. They have more than one input	4. Tasks may have zero or more arguments of type input, output or inout
5. Function always return a single value. They cannot have inout or output arguments	5. Taske do not return with a value, but can pass multiple values through output & inout arguments
6. Functions can have input arguments	6. Taske can have input, output & inout arguments

OR

8 a. Explain sequential and Parallel blocke with an example — 06 Marks

Sol:- Sequential block —— 03 Marks
Parallel block - 03 Marks
06 Marks

Sequential blocks: The keywords begin and end to group statements into sequential blocks Statements in a sequential blocks are processed in the order they are specified. If delay or event control is specified, it is relative to the simulation time when the previous statement in the block completed execution.

Ex:-
```
reg x, y;
reg [1:0] z, w;
initial
begin x = 1'b0;
      y = 1'b1;
      z = {x, y};
      w = {y, x};
end
```

Parallel blocks:- Parallel blocke, specified by keywords fork and join, statements in a parallel block are executed concurrently. Ordedeing of statements is controlled by the delay or event control assigned to each statment. If delay or event control is specified, it is relative to the time the block was entered.

Ex:-
```
reg x, y;
reg [1:0] z, w;
initial
fork
   x = 1'b0;

  #5 y = 1'b1;
  #10 z = {x, y};
  #20 w = {y, x};
join
```

8b. Design a negative edge-triggered D flip-flop (DUFF) (ui with synchronous clear, active high (DFF clears only at a negative edge of clock when clear is high). Design a clock with a period of 10 units & test the D-flip-flop — 08 Marks.

Sol:- Designing of a negative edge-triggered D flip-flop - 04 mar
Designing of a clock with a period of 10 units & testing — 04 mark
08 mark

Design blocks

```
module  DFF ( d, clk, clr, q);
input  d, clk, clr;
output  q;
 reg   q;
always @ (negedge clk)
begin
 if (clr)
   q = 1'd0;
   else
    q = d;
 end
 endmodule
```

Stimulus block

```
module test ;
reg  clk, clr, d;
wire  q;
d-ff  cut ( .clk (clk), .q(q), .clr(clr), .d(d));
initial
     clk = 1'b0;
always @ (clk)
   forever
     #5 clk = ~clk;
endmodule
```

8c Write verilog program to call function called calc-parity which computes the parity of 32-bit data [31-0] Data & display odd or even parity - 06 Marks

Sol:- verilog code for a calc-parity — 06 marks

```
module parity;
 reg [31 : 0] addr;
 reg parity;
 always @ (addr)
 begin
     parity = calc - parity (addr);
     $display ("parity calculated = %b", calc-parity (addr));
 end

function calc-parity;
input [31:0] address;
begin
    calc- parity = ^address;
end
endfunction
—
—
endmodule
```

Module 5

9a. Write a note on — 08 Marks

i> force & release

ii> Defparam statement

iii> time scale

iv> file output

Sol:- i> force & release — 02 Marks

force and release are used to express the second form of the procedural continuous assignments. They can be used to override assignments on both registers and nets. force and release statements are typically used in interactive debugging process, where certain registers or nets are forced to a value & the effect on other registers & nets is noted.

ii> Deffparam statement —— 02 marks

parameters value can be changed in any module instance in the design with the keyword deffparam. The hierarchical name of the module instance can be used to override parameter values. Multiple deffparam can appear in a module. Any parameter can be overridden with the deffparam statement.

iii> time - scale —— 02 marks

In a single simulation, delay values in one module need to be defined by using a certain time unit eg. 1.μs and delay values in another module need to be defined by using differents time unit eg. 100 ns. Therefore verilog HDL allows the reference time unit for modules to be specified with the ~time scale compiler directive

Syntax:- ~timescale <reference_time-unit>/<time-precision>

iv> file output —— 02 marks

file can be opened with the system task $fopen

```
$fopen ("name-of-file>");
<file_handle> = $fopen ("name-of-file");
```

$fopen returns a 32-bit value called multichannel file descriptor note on verification of gate level netlist -04 m

Sol:- note on verification of gate level netlist - 04 marks.

The optimized gate-level netlist produced by the logic synthesis tool must be verified for functionality. Also, the synthesis tool may not always be able to meet both timing & area requirements if they are too stringent.

Functional verification

Identical stimulus is run with the original RTL and synthesized gate-level descriptions of the design. The output is compared to find any mismatches

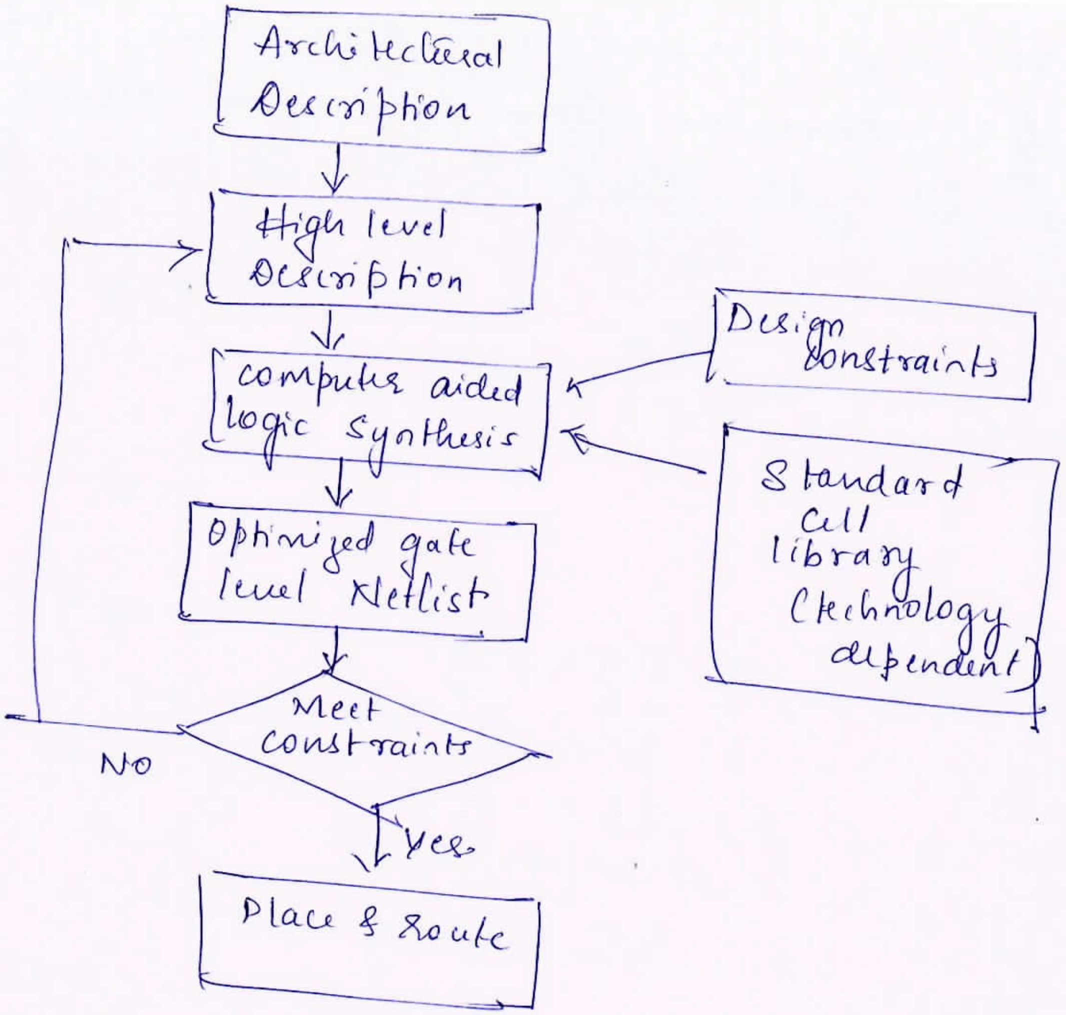
Timing verification :- The gate level netlist is typically checked for timing by use of timing simulation or by a static timing verifier. If any timing constraints are violted, the designer must either redesign part of the RTL or make trade-offs in design constraints for logic synthesis. The entire flow is iterated until timing requirements are met.

1c With a neat flow chart explain computer Aided logic Synthesis process — 08 marks.

Sol:- flow chart carries - 03 marks

Explaination - 05 marks

08 marks.


Architectural Description
High level Description
computer aided logic synthesis
Design constraints
Standard cell library (technology dependent)
Optimized gate level Netlist
Meet constraints
NO
Yes
Place & Route

The advent of computer aided logic synthesis tool has automated the process of converting the high level description to logic gates. Instead of trying to perform logic synthesis in their minds, designers can now concentrate on the architectural trade offs, high level description of the design, accurate design constraints & optimization of cells in the standard cell library. These are fed to the the computer-aided logic synthesis tool, which performs several iterations internally & generates optimized gate level descriptions. Also instead of drawing the high level description on a screen or a piece of paper, designers describe the high level design in terms of HDLs. Verilog HDL has become one of the popular HDLs for writing of high-level descriptions.

automated logic synthesis has significantly reduced the time for conversion from high-level design representations to gates. This allowed designers to spend more time on designing at higher level of representation, because less time is required for converting the design to gates.

10

OR

a. what is logic synthesis — 04 Marks

Sol:- explaination of logic synthesis — 04 marks

Logic synthesis is the process of conversting a high level description of the design into an optimized gate level representation, given a standard cell library & certain design constraints.

A standard cell library can have simple cells, such as basic logic gates like and, or & not gates or macro cells such as adders, mux and flip-flops. A standard cell library is also known as technology library.

Logic synthesis always exited.

10

b Interpret the following verilog constructs after logic synthesis — 10 Marks

i) the assign statement

ii) if - else statement

iii) case statement

iv) always statement

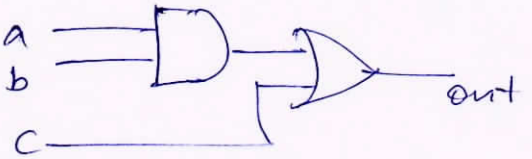
:- each interpretation carries — 2.5 marks

2.5×4 = 10 marks

i> assign statement

```
assign out = (a & b) | c ;
```

The above assign statement is translated to gate level representation.



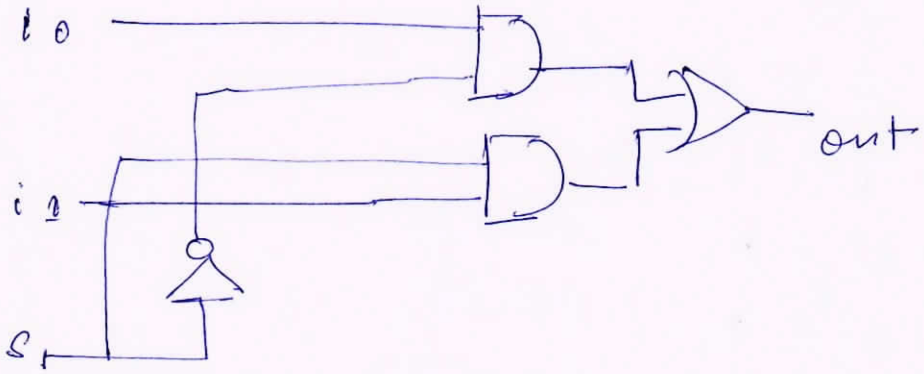
ii> if - else statement

if - else statements translate to multiplexers where the control signal is the signal or variable in the if clause.

```
if (s)
    out = i1 ;
else
    out = i0 ;
```

iii> case statement

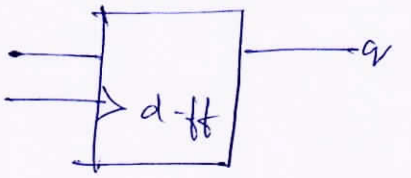
```
case (s)
1'b0 : out = i0;
1'b1 : out = i1;
endcase
```



iv) always statement

always Statement can be used to infer sequential. and combinational logic. For sequential logic, the always statement must be controlled by the change in the value of a clock signal called clk

```
always @ (posedge clk)
        q<=d;
```



||ly

```
always @ (clk or d)
   if (clk)
      q<=d;
```

verilog description create - level sensitive latch.

10c. Write RTL description for magnitude comparator — 06 Marks

Sol:- RTL description for magnitude comparator — 6 marks

```
module magnitude-comparator (A-gtB A_lt_B, A-eq-B,
                                         A, B);
output A-gt-B, A-lt-B, A-eq-B;
input  [3:0] A, B;
assign  A-gt-B  =  (A>B);
assign  A-lt-B  =  (A<B);
assign  A-eq-B  =  (A==B);
endmodule
```