USN | | | | | | | | | | |

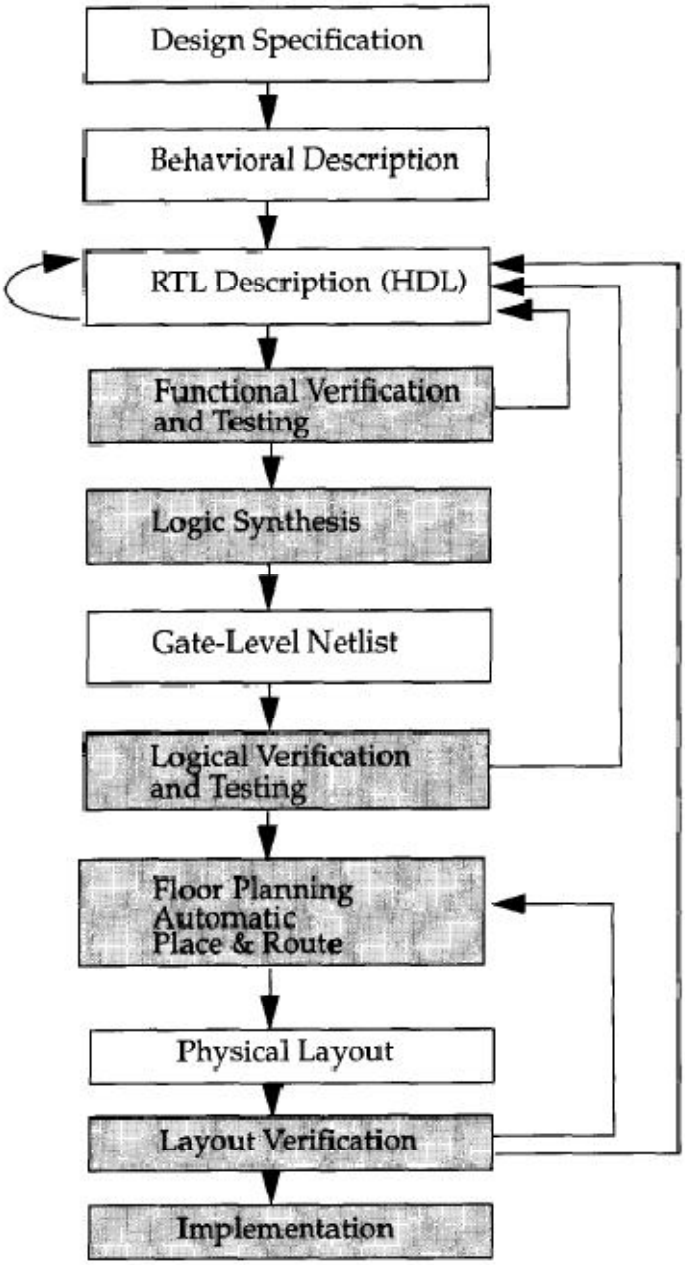
Internal Assessment Test 1 – Sept. 2017

Sub:	Verilog HDL					Sub Code:	15EC53	Branch:	ECE & TCE
Date:	20 / 09 / 2017	Duration:	90 min's	Max Marks:	50	Sem / Sec:	5 / All sec		OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Explain the typical design flow for designing VLSI IC with the flow chart.	[10]	CO1	L4
2	Describe the design of 4-bit Ripple carry adder using top down design methodology. Also write Verilog HDL program for 4-bit ripple carry adder.	[10]	CO2	L2
3	Demonstrate the design of 4-bit ripple carry counter using top down design methodology. Also describe the behavior of D Flip-flop using Verilog HDL.	[10]	CO2	L3
4	Describe the functionality of 4-bit asynchronous counter using Verilog HDL. Also write the stimulus to verify the design.	[10]	CO2	L2
5	Explain the different data types used in Verilog HDL with examples.	[10]	CO2	L4
6	Explain the following with an example a) Module, b) instance, c) System tasks, d) compiler directive, e) simulation	[10]	CO1	L4
7	With block diagram explain the components of Verilog HDL module. Also write Verilog HDL module to design SR Latch with stimulus.	[10]	CO2	L2

1 Explain the typical design flow for designing VLSI IC with the flow chart. [10]

The design flow chart is as shown below



Any design starts by loading the specification first. Specifications describe the functionality, interface and overall architecture of the digital circuits to be designed. Without working on the implementation.

Behavioral description is created to analyze its design interms of functionality, performance and complia -nce of the to standards, and other high-level issues. Behavioral description can be written using HDL.

This behavioral description is converted manually to an RTL description in HDL using dataflow, that will imple -ment the desired digital circuit. This RTL description is an input to the CAD tool, further process is done by these tools.

Logic synthesis tool converts the RTL description to a gate level netlist. Gate level netlist consists of compon ents and interconnection between them. Netlist will become input to an automatic place and route tool, which creates layout. Layout is verified and then fabricated on chip.

—05 marks)

2 Describe the design of 4-bit Ripple carry adder using top down design methodology. Also write Verilog HDL program for 4-bit ripple carry adder. [10]

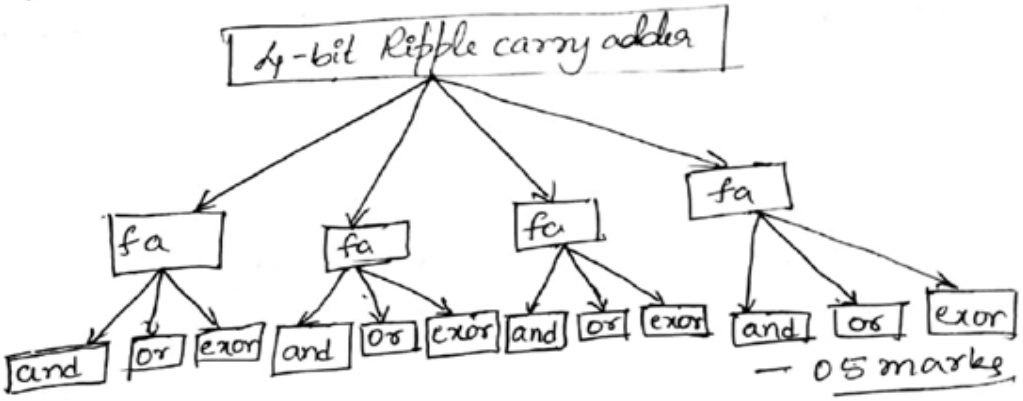
Hence and gate, or gate and exor gate forms the components present in the next level of the hierarchy.

These gates are realized using switches. Hence they cannot be divided further.

Therefore andgate, orgate and exor gates forms the leafcells of the design.

~~the top down~~

The representation of 4-bit ripple carry adder using top down design methodology as.



– 05 marks

Verilog HDL program

Toplevel module definition

~~module RCA4-b (a, b,~~

```
module RCA 4-b (s, cout, a, b, cin);
  input [3:0] a, b;
  input cin;
  output [3:0] s;
  output cout;
  wire c0, c1, c2;
  fa f1 (s[0], c0, a[0], b[0], cin); //instantiation of
  fa f2 (s[1], c1, a[1], b[1], c0); // lower level module
  fa f3 (s[2], c2, a[2], b[2], c1); // fa for full adder
  fa f4 (s[3], cout, a[3], b[3], c2);
endmodule.
```

– 01 mark

```
// Definition of full adder.
module fa (s, co, a, b, cin);
input a, b, cin;
output s, co;
wire y1, y2, y3;
xorg x1 (y1, a, b); // instantiation of leaf cells
[illegible];
andg a1 (y2, a, b);
xorg x2 (s, y1, cin);
andg a2 (y3, cin, y1);
org o1 (co, y2, y3);
endmodule.
```

— 01 mark

```
// Definition of leaf cells.
module andg (y, a, b); // definition of andgate
input a, b;
output y;
assign y = a & b;
endmodule
```

— 01 mark

```
module org (y, a, b); // definition of orgate
input a, b;
output y;
assign y = a | b;
endmodule
```

— 01 mark

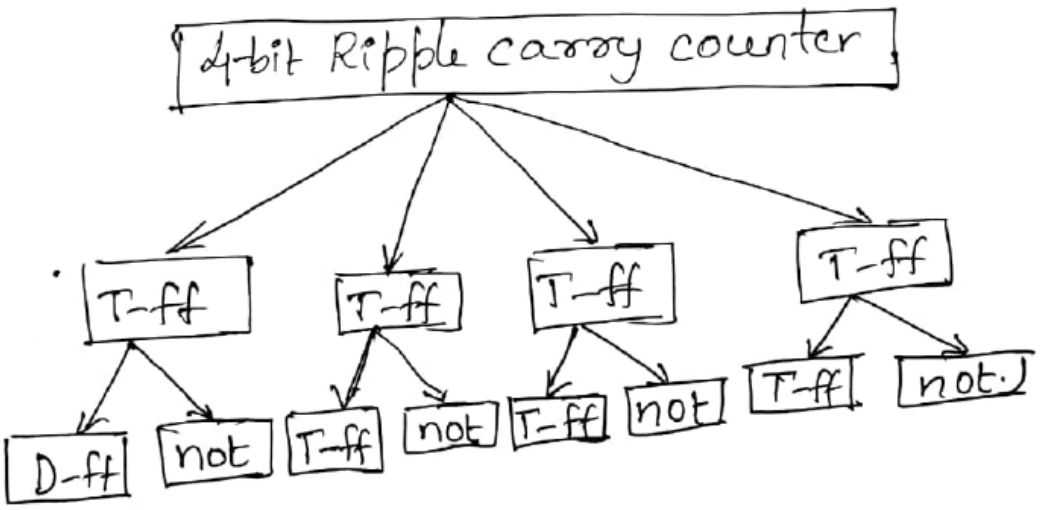
```
module xorg (y, a, b); // definition of exorgate.
input a, b;
output y;
assign y = a ^ b;
endmodule.
```

— 01 mark

3 Demonstrate the design of 4-bit ripple carry counter using top down design methodology. Also describe the behavior of D Flip-flop using Verilog HDL. [10]

Here top level module is 4-bit Ripple carry counter. Ripple carry counter is constructed using T-flipflop. Hence in the next level of hierarchy it consists of T-flipflop. T-flipflop is constructed using D flipflop and an inverter. Hence in the third level hierarchy consists of only D-flipflops and an inverter. D flipflop can be implemented using gates like and and inverter or it may be designed using only switches. If D-ff is implemented using only switches then D-ff is one of the leaf cell. Inverter is implemented using switches. Hence the leaf cells of Ripple carry counter is D-ff and an inverter.

Using top down design methodology the design of Ripple carry counter is as shown below.



—05 marks

The behavior of D-flipflop using verilog HDL

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

— 04 marks

4 Describe the functionality of 4-bit asynchronous counter using Verilog HDL. Also write the stimulus to verify the design. [10]

// Ripple Carry Counter Top Block

```
module ripple-carry-counter(q, clk, reset) ;
output [3:01 q;
input clk, reset;
T-FF tffO(q[O] ,clk, reset)
T-FF tffl(q[l] ,q[O], reset
T-FF tff2 (q[2] ,q[l], reset
T-FF tff3 (q[3l ,q[2], reset
endmodule                                          --2 marks
```

//Flip-flop T-FF

```
module T-FF (q, clk, reset) ;
output q;
input clk, reset;
wire d;
D-FF dff0 (q, d, clk, reset) ;
not nl(d, q) ;//not is a Verilog-provided primitive. case
sensitive
endmodule                                          --2 marks
```

//module D-FF with synchronous reset

```
module D-FF(q, d, clk, reset) ;
output q;
input d, clk, reset;
reg q;
always @(posedge reset or negedge clk)
if (reset)
q = l'bO;
else
q = d;
endmodule                                         --2 marks
```

```
// Stimulus Block
module stimulus;
reg clk;
reg reset;
wire[3:01 q;
ripple-carry-counter rl(q, clk, reset);
initial
clk = l'bO; //set clk to 0
always
#5 clk = -clk; //toggle clk every 5 time units
initial
begin
reset = l'bl;
#l5 reset = l'bO;
#l80 reset = l'bl;
#l0 reset = l'bO;
#20 $finish; //terminate the simulation
end
initial
$monitor ($time, " Output q = %d" , q) ;
endmodule                                          --04 marks
```

5 Explain the different data types used in Verilog HDL with examples. [10]

Datatypes of verilog HDL:

a) value set:

Verilog supports 4 values and eight strength levels. they are

value level	condition in hardware circuits
0	logic zero, false condition.
1	logic one, true condition.
X	Unknown value.
Z	High impedance, floating gate.

To resolve conflicts between drivers of different strengths in digital circuits values 0 & 1 can have strengths levels. as listed below.

Strength level	Type	Degree.
Supply	driving	strongest
Strong	driving.	
pull	driving.	
large	storage.	
weak	driving	
medium	storage.	
small	storage.	
highz	high impedance	weakest.

— 02 marks

b) Nets: Nets represents connections between hardware elements. Nets are declared as wires.

Default value of wire 'z'.

ex: wire a; - 01 marks

c) Registers: Register represents data storage elements. Register retain value until another value is placed onto them. They are declared using keyword 'reg'.

ex: reg initial;

Default value of reg data type is 'x'. - 01 mark

d) Vectors:

Nets or reg datatypes can be declared as vectors. If bitwidth is more than one bit they are vectors.

ex: wire [7:0] b;
reg [3:0] a; 01 mark

e) Integer:

Integer is general purpose register data type used for manipulating quantities.

Integers are declared using keyword Integer

ex: Integer n; 01 marks

They are of type 'reg'.

f) Real:

Real number constants and real register data types declared using keyword 'real'.

They can be specified using decimal notation or in a scientific notation.

ex: real pi = 3.14;
real a = 3e4;

— 01 mark

g) Arrays: Collection of more than one elements of same datatype.

Arrays are allowed in verilog for reg, integer, time, and vectored register data types.

Arrays are not for real types.

ex: integer count [0:7]; // array of 8 elements (integer
reg bool [31:0]; // array of 32 elements each elemen
// is a binary digit.

— 01 marks

~~Memory: Models of memory elements in the given digital circuit sequences models~~

h) Memory: It is a register file, RAM or ROMs.

Some design requires these memories. It can be modeled as shown in the following example.

reg [7:0] mumbyte [0:1023];
// In the above example a memory is modeled of size 8x1k or 1kBytes.

— 01 marks

i) parameters:

Verilog allows constants to be defined in a module by the keyword 'parameter'. They cannot be used as a variables.

— 01 mark

ex: parameter port_id = 5;

6 Explain the following with an example
a) Module, b) instance, c) System tasks, d) compiler directive, e) simulation [10]

a) Module: module is a basic building block in verilog.
A module can be an element or a collection of lower level design blocks. Elements are grouped into modules to provide common functionality that is used at many places in the design.

```
module <module_name>(<module_terminal_list>);
...
<module internals>
...
endmodule.
```

ex:
```
module tff(q, clk);
  input clk;
  output q;
endmodule.
```

ex:
```
module tff (q, clk, res);
  input clk, res;
  output q;
endmodule.
```
— 02 marks

b) Instance:
Process of creating objects from a module template is called instantiation. and the object name is called instances.

ex:
```
module RCC (q, clk, res);
  output [3:0] q;
  input clk, res;
  tff t0(q[0], clk, res); // tff – module template name
                          // t0 – instance name
```
— 02 marks

c) System tasks:

Verilog provides standard system tasks to do certain routine operations. They appear in the form $<keyword>.

ex: $display // displays a message.

$monitor // monitors the change in values of variables

```
$display ("hello. verilog world");
$monitor ($time, "value of a=%b, b=%b", a, b);
```

– 02 marks

d) Compiler directive:

They are used to define macro texts, global constants, ~~another to #define in c language~~.

ex: `define ~~stmt~~ in verilog is similar #define in c language.

~~ex~~ `define pi 3.14 ← macro value.

↑ textmacro

```
real t = `pi;
```

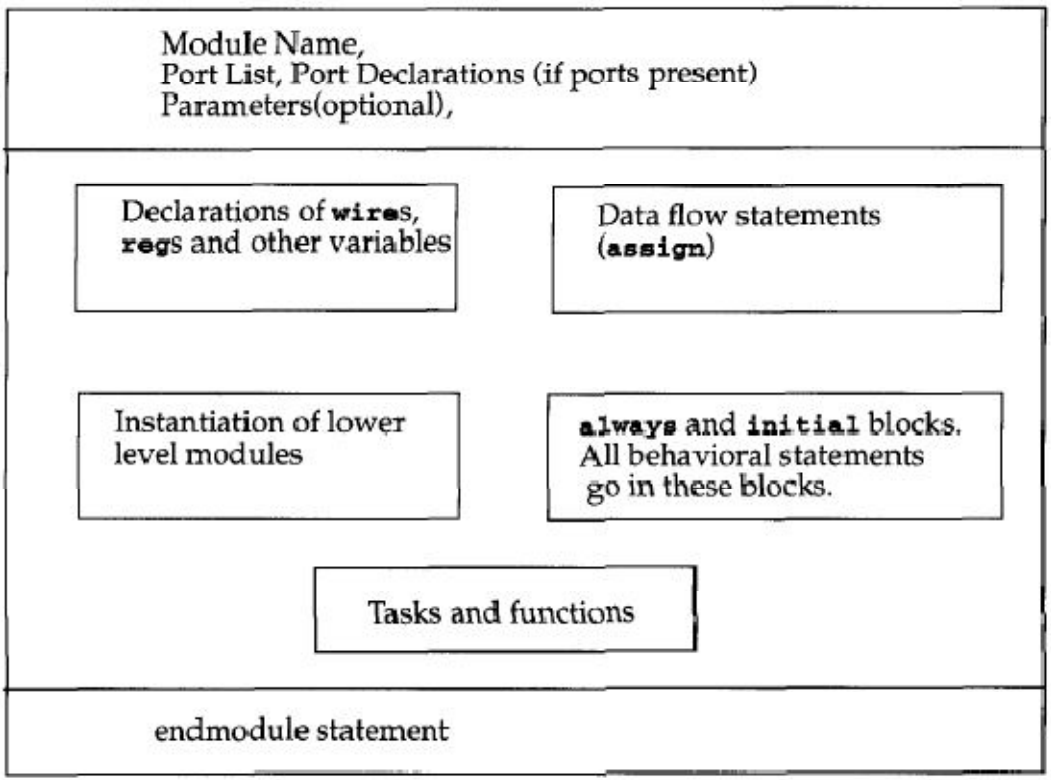
`include in verilog is similar to #include in c language contents of one verilog module is inserted in another verilog module.

02 marks

e) Simulation : Simulation is the execution of a model in the software environment. Simulation is the process verifying the functional correctness of a digital design that is modeled using a HDL. The test bench is used to simulate design by specifying the inputs to the system. 02 marks.

7 With block diagram explain the components of Verilog HDL module. Also write Verilog HDL module to design SR Latch with stimulus. [10]

The components of Verilog HDL program is as shown below.



--02 Marks

The stimulus block has module name, variable declarations, lower level module instantiation and behavioral statements and finally endmodule. It also contain one system task. Here data flow statements, ~~exp~~ port list, ~~and~~ port declaration and parameter declaration are missing.

```
module Top;
wire q, qbar;
reg set, reset;
SR-latch ml(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                                    -- 03 Marks
```

The design block has module name, port listing, port declarations, primitive gate instantiation, and endmodule. There other components such as parameters, variable declaration, dataflow, behavioral statements, system tasks are missing

```
module SR-latch (Q, Qbar, Sbar, Rbar) ;
output Q, Qbar;
input Sbar, Rbar;
nand nl (Q, Sbar, Qbar) ;
nand n2 ( Qbar , Rbar , Q) ;
endmodule                                    --03 marks
```

From this it is evident that a verilog HDL contains module name and endmodule compulsorily. Module body may contain mix and match of different types of description statements. It may contain port, port declarations, and parameters, system tasks. etc.

02 Marks