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

Internal Assessment Test 3 Solution –JAN 2023

Sub:	Verilog HDL					Sub Code:	18EC56	Branch :	ECE
Date:	23/1/23	Duration:	90 min's	Max Marks:	50	Sem/Sec :	5th/A,B,C,D		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	(a)Explain blocking and non blocking statements with an example.	[5]	C01	L2

Blocking Assignments

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.

Example 7-6 Blocking Statements

```
reg x, y, z;
reg [15:0] reg_a, reg_b;
integer count;
//All behavioral statements must be inside an initial or always block
initial
begin
to end
x = 0; y = 1; z = 1; //Scalar assignments
count = 0; //Assignment to integer variables
reg_a = 16'b0; reg_b = reg_a; //initialize vectors
#15 reg_a[2] = 1'b1; //Bit select assignment with delay
#10 reg_b[15:13] = {x, y, z} //Assign result of concatenation
                    // part select of a vector
count = count + 1; //Assignment to an integer (increment)
```

(b) Write behaviorioal verilog description of 4 to 1 Multiplexer [5] CO2, CO3

There are two types of blocks: sequential blocks and parallel blocks.

Sequential blocks

The keywords begin and end are used to group statements into sequential blocks. Sequential blocks have the following characteristics:

- The statements in a sequential block are processed in the order they are specif A statement is executed only after its preceding statement completes executio (except for nonblocking assignments with intra-assignment timing control).
- If delay or event control is specified, it is relative to the simulation time when previous statement in the block completed execution.

2	(a) Explain force, release and assign, deassign statements with an example.	[6]	CO5	L2
	(b) Explain sequential and parallel block with one example.	[4]	CO2	

```
end

//Illustration 2: Sequential blocks with delay.
reg x, y;
reg [1:0] z, w;

initial
begin
        x = 1'b0; //completes at simulation time 0
        #5 y = 1'b1; //completes at simulation time
        #10 z = {x, y}; //completes at simulation t
        #20 w = {y, x}; //completes at simulation t
end
```

There are two types of blocks: sequential blocks and parallel blocks.

Sequential blocks

The keywords begin and end are used to group statements into sequential bloc Sequential blocks have the following characteristics:

- The statements in a sequential block are processed in the order they ar A statement is executed only after its preceding statement completes e (except for nonblocking assignments with intra-assignment timing cor
- If delay or event control is specified, it is relative to the simulation tim previous statement in the block completed execution.

3	Write a design and test bench verilog program for carry look-ahead adder using data flow description.	[10]	CO4	L3
4	(a)Compare task and function with syntax.	[4]	CO4	L3

(b) Write verilog code to calculate parity using task. [6] | CO2, CO3, CO5 |

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

Internal Assessment Test 3 –JAN 2023

Sub:	Verilog HDL					Sub Code:	18EC56	Branch:	ECE
Date:	23/1/23	Duration:	90 min's	Max Marks:	50	Sem/Sec:	5th/A,B,C,D		OBE

Answer any FIVE FULL Questions

		MARKS	CO	RBT
1	(a)Explain blocking and non blocking statements with an example.	[5]	C01	L2
	(b) Write verilog program to design 4 to 1 Multiplexer using logic equation and conditional operator.(Data flow description)	[5]	CO2, CO3	L3
2	(a) Explain force, release and assign, deassign statements with an example.	[6]	CO5	L2
	(b) Explain sequential and parallel block with one example.	[4]	CO2	L2
3	Write a design and test bench verilog program for 4bit full adder with carry look-ahead using data flow description.	[10]	CO4	L3

Example 6-5 4-bit Full Adder with Carry Lookahead

```
module fulladd4(sum, c_out, a, b, c_in);
// Inputs and outputs
output [3:0] sum;
output c_out;
input [3:0] a,b;
input c_in;

// Internal wires
wire p0,g0, p1,g1, p2,g2, p3,g3;
wire c4, c3, c2, c1;

// compute the p for each stage
assign p0 = a[0] ^ b[0],
       p1 = a[1] ^ b[1],
```

4	(a)Compare task and function with syntax.	[4]	CO4	L3

Table 8-1. Tasks and Functions

Functions	**Tasks**
A function can enable another function but not another task.	A task can enable other tasks and functions.
Functions always execute in 0 simulation time.	Tasks may execute in non-zero simulation time.
Functions must not contain any delay, event, or timing control statements.	Tasks may contain delay, event, or timing control statements.
Functions must have at least one input argument. They can have more than one input.	Tasks may have zero or more arguments of type input, output, or inout.
unctions always return a single value. They cannot have output or inout arguments.	Tasks do not return with a value, but can pass multiple values through output and inout arguments.

(b) Write verilog code to calculate parity of 32 bit data using task with delay. [6] CO2, CO3, CO5 L3

```
//Define a module that contains the function calc_parity
module parity;
...
reg [31:0] addr;
reg parity;

//Compute new parity whenever address value changes
always @(addr)
begin
        parity = calc_parity(addr); //First invocation of calc_parity
        $display("Parity calculated = %b", calc_parity(addr) );
                                    //Second invocation of calc_parity
end
...
...
//define the parity calculation function
function calc_parity;
input [31:0] address;
begin
        //set the output value appropriately. Use the implicit
        //internal register calc_parity.
        calc_parity = ^address; //Return the xor of all address bits.
end
endfunction
```

5 (a) Explain generate case statement with an example [5] C01, CO2 L2

```
// Port declarations
output [N-1:0] sum;
output co;
input [N-1:0] a0, a1;
input ci;

// Instantiate the appropriate adder based on the width of the b
// This is based on parameter N that can be redefined at
// instantiation time.
generate
case (N)
  //Special cases for 1 and 2 bit adders
  1: adder_1bit adder1(c0, sum, a0, a1, ci); //1-bit implementat
  2: adder_2bit adder2(c0, sum, a0, a1, ci); //2-bit implementat
  // Default is N-bit carry look ahead adder
  default: adder_cla #(N) adder3(c0, sum, a0, a1, ci);
endcase
endgenerate //end of the generate block

endmodule
```

```
// This module generates an N-bit adder

module adder(co, sum, a0, a1, ci);
// Parameter Declaration. This can be redefi
parameter N = 4; // 4-bit bus by default
```

7.8.3 Generate Case

A generate case permits the following Verilog constructs to be conditionally ins into another module based on a select-one-of-many case construct that is determ the time the design is elaborated:

- Modules
- User defined primitives, Gate primitives
- Continuous assignments
- initial and always blocks

(b) Write verilog code to calculate factorial of given number using function.	[5]	CO4	L3

```
module top;

parameter delay=10;
reg [3:0] n;
reg [31:0] n_fac;

initial
begin
  n=4'd5;
  #delay factorial(n_fac,n);
  $display($time, "Factorial of n is %d",n_fac);
end

task automatic factorial;
output [31:0] x_fac;
input [3:0] x;
begin
if(x>=2)
  begin
    factorial(x_fac,x-1);
    x_fac=x_fac*x;
  end
else
 x_fac=1;
end
endtask

endmodule
```

6 Explain neat flow chart explain logic systhesis from RTL to gates. [10 CO6 L2

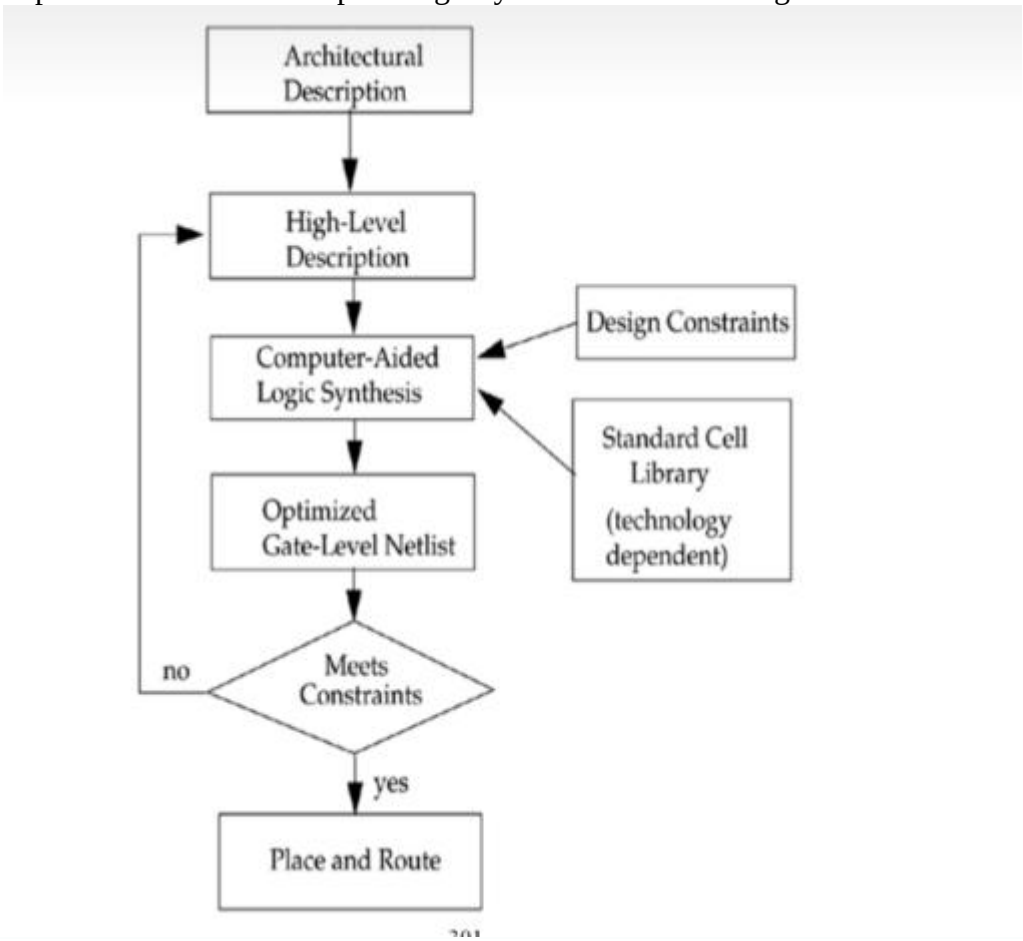
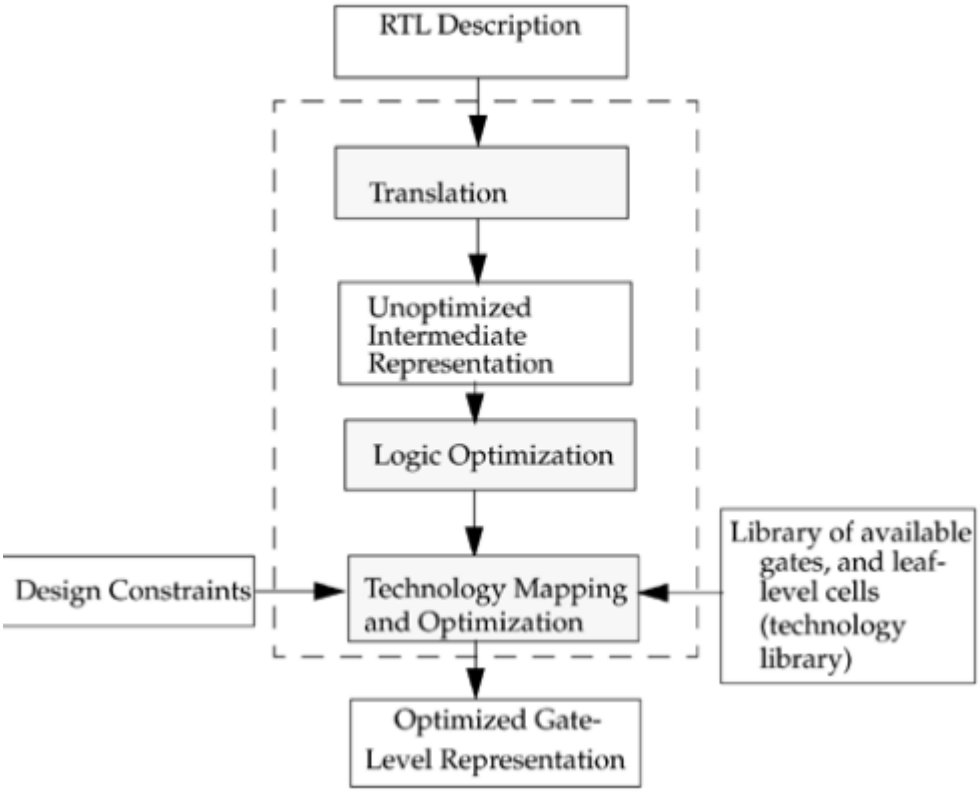


Figure 14-4. Logic Synthesis Flow from RTL to Gates



7	Explain overriding parameters with examples.	[10]	CO2, CO3	L3

9.2 Overriding Parameters

Parameters can be defined in a module definition, as was discussed earlier in Section 3.2.8, Parameters. However, during compilation of Verilog modules, parameter values can be altered separately for each module instance. This allows us to pass a distinct set of parameter values to each module during compilation regardless of predefined parameter values.

There are two ways to override parameter values: through the defparam statement or through module instance parameter value assignment.

9.2.1 defparam Statement

Parameter values can be changed in any module instance in the design with the keyword defparam. The hierarchical name of the module instance can be used to override