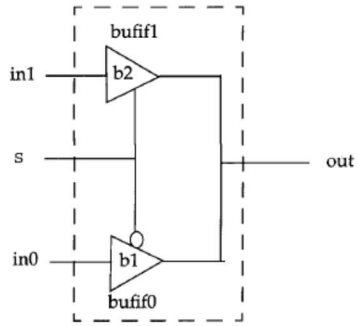
CMR INSTITUTE OF TECHNOLOGY

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

Internal Assesment Test - II

Sub:	Verilog HDL							Code:	15EC53
Date:	08 / 11/ 2017	Duration:	90 mins	Max Marks:	50	Sem:	5th	Branch:	ECE(C & D) and TCE

Answer Any FIVE FULL Questions

		Marks	OBE CO	OBE RBT
1.	Use gate level description of Verilog HDL to design 4 to 1 multiplexer. Write truth table, top-level block, logic expression and logic diagram. Also write the stimulus block for the same.	[10]	CO3	L1
2.	Use dataflow description style of Verilog HDL to design 4-bit adder using i) Ripple carry logic. ii) Carry look ahead logic.	[10]	CO4	L5
3.	Design a 2-to-1 multiplexer using **bufifo** and **bufifl** gates as shown below.  The delay specification for gates b1 and b2 are as follows. Apply stimulus and test the output values.	[10]	CO4	L3
4.	Use behavioral abstraction to write Verilog HDL program to design i) 4 to 1 multiplexer program using CASE statement. ii) 4-bit counter with asynchronous reset.	[5] [5]	CO3	L5
5	Describe the following statements with an example: initial and always. Also design a clock with time period = 40 and a duty cycle of 25% by using the **always** and **initial** statements. The value of clock at time = 0 should be initialized to 0.	[10]	CO4	L1
6	Write Verilog HDL program to simulate traffic signal controller. Also write the stimulus for the same	[10]	CO3	L3
7	a) Using a **while** loop, design a clock generator. Initial value of *clock* is 0. Time period for the clock is 10. b) Using **for** loop, initialize locations 0 to 1023 of a 4-bit register array *cache_var* to 0.	[5] [5]	CO3	L2

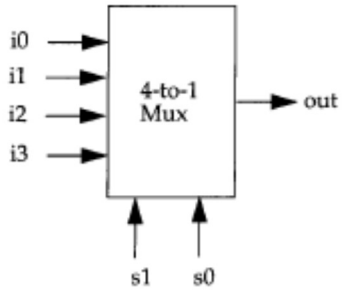
	Min	Typ	Max
Rise	1	2	3
Fall	3	4	5
Turnoff	5	6	7

1. Use gate level description of Verilog HDL to design 4 to 1 multiplexer. Write truth table, top-level block, logic expression and logic diagram. Also write the stimulus block for the same. 10 marks

Truth Table: 1 mark

S1	S0	out
0	0	I0
0	1	I1
1	0	I2
1	1	I3

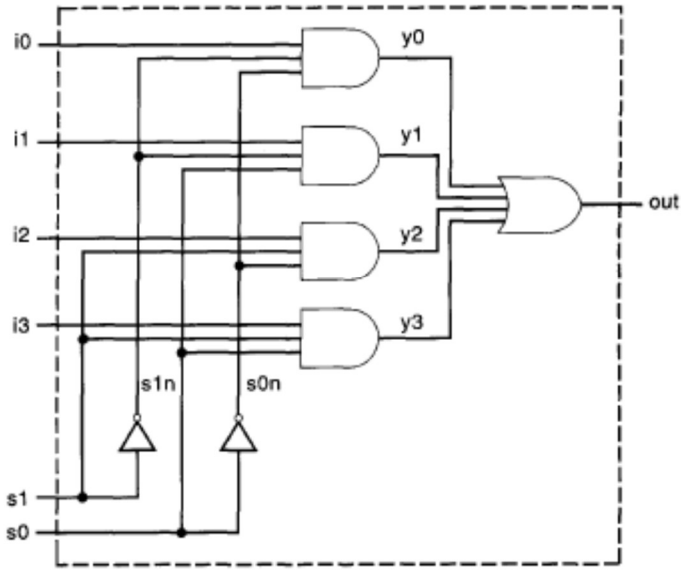
Top level diagram: 1 mark



Logic expression: 1 mark

$$out = \overline{s1}\,.\,\overline{s0}.I0 + \overline{s1}.s0.I0 + s1.\overline{s0}\,.I0 + s1\,.s0.I0$$

Logic Diagram: 2 marks



Gate-level abstraction:

```
module mux4_to_l (out, i0, il, i2, i3, s1, s0);
    output out;
    input i0, il, i2, i3;
        input sl, s0;
        wire sln, s0n;
        wire y0, yl, y2, y3;
        not (s1n, s1) ;
```

```
                not (s0n, S0);
                and (y0, i0, s1n, s0n);
                and (yl, il, s1n, s0);
                and (y2, i2, s1, s0n);
                and (y3, i3, s1, S0);
                or (out, y0, yl, y2, y3 ) ;
endmodule

Stimulus:
module stimulus;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1 mymux (OUTPUT, IN0, IN1, IN2, IN3, S1, S0) ;
        initial
        begin
                IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
                #l $display("IN0= %b, IN1= %b, IN2= %b,
IN3=%b\n",I0,I1,I2,I3);
                S1 = 0; S0 = 0;
                #l $display("Sl = %b, S0 = %b, OUTPUT = %b \n", S1, S0,
OUTPUT);
                S1 = 0; S0 = 1;
                #l $display("Sl = %b, S0 = %b, OUTPUT = %b \n", S1, S0,
OUTPUT);
                / / choose IN2
                S1 = l; S0 = 0;
                #l $display("Sl = %b, S0 = %b, OUTPUT = %b \n", S1, S0,
OUTPUT);
                S1 = l; S0 = 1;
                #l $display("Sl = %b, S0 = %b, OUTPUT = %b \n", S1, S0,
OUTPUT);
        end
endmodule
```

3 marks

2 marks

2. Use dataflow description style of Verilog HDL to design 4-bit adder using
   i. Ripple carry logic.
   ii. Carry look ahead logic.

10 Marks

i. Ripple carry logic:

```
module fulladd4(sum, c_out, a, b, c_in);
output [3:01 sum;
output c_out;
input [3:01 a,b;
input c_in;
wire c0,c1,c2;
  assign s[0]=a[0] ^ b[0] ^ c_in;
  assign s[1]=a[1] ^ b[1] ^ c0;
  assign s[2]=a[2] ^ b[2] ^ c1;
  assign s[3]=a[3] ^ b[3] ^ c2;
  assign c0 = (a[0] & b[0])|(c_in & b[0])|(a[0] & c_in);
```

```
  assign c1 = (a[1] & b[1])|(c0 & b[1])|(a[1] & c0);
  assign c2 = (a[2] & b[2])|(c1 & b[2])|(a[2] & c1);
  assign c_out = (a[3] & b[3])|(c2 & b[3])|(a[3] & c_in);
endmodule
```

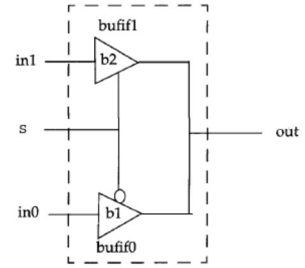
05 marks

ii. Carry lookahead logic :

```
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,p1, p2,p3,g0,g1, g2,g3;
wire c4, c3, c2, c1;

assign p0 = a[0] ^ b[0],
  p1 = a[l] ^ b[1],
  p2 = a[2] ^ b[2],
  p3 = a[3] ^ b[3];

assign g0 = a[0] & b[0],
  g1 = a[l] & 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) | (p2 & p1
               & p0 & c_in));
  assign s[0]=a[0] ^ b[0] ^ c_in;
  assign s[1]=a[1] ^ b[1] ^ c1;
  assign s[2]=a[2] ^ b[2] ^ c2;
  assign s[3]=a[3] ^ b[3] ^ c3;
endmodule
```

05 marks

3. Design a 2-to-1 multiplexer using **bufifo** and **bufifl** gates as shown below.



The delay specification for gates *bl* and b2 are as follows.

	Min	Typ	Max
Rise	1	2	3
Fall	3	4	5
Turnoff	5	6	7

10 Marks

Apply stimulus and test the output values.

Design:

```
module bufnot(
   input in1, in0, s,
   output out
   );
bufif1 #(1:2:3,3:4:5,5:6:7) b2(out,in1,s);
bufif0 #(1:2:3,3:4:5,5:6:7) b1(out,in0,s);
endmodule
```

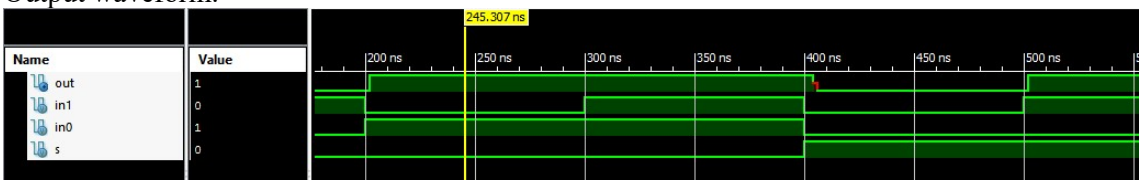
05 marks

Stimulus:

```
module bufnot_tb;
reg in1, in0, s;
wire out;
bufnot uut (.in1(in1), .in0(in0), .s(s), .out(out));
      initial begin
                in1 = 0; in0 = 0; s = 0;#100;
                 in1 = 1;in0 = 0; s = 0;#100;
                in1 = 0; in0 = 1; s = 0;#100;
                in1 = 1; in0 = 1; s = 0;#100;
               in1 = 0; in0 = 0; s = 1;#100;
                in1 = 1; in0 = 0; s = 1;#100;
                in1 = 0; in0 = 1; s = 1;#100;
               in1 = 1; in0 = 1; s = 1;#100;
      end
endmodule
```

05 marks

Output waveform:



4. Use behavioral abstraction to write Verilog HDL program to design 10 marks

i) 4 to 1 multiplexer program using CASE statement.

```
module mux4_to_1 (out, i0, il, i2, i3, s1, s0);
      output out;
      input i0, il, i2, i3;
            input sl, s0;
```

```
always @(s1 or S0 or i0 or il or i2 or i3)
   begin
         case ({s1, S0})
               2'b00: out = iO0
               2'b01: out = i1;
               2'b10: out = i2;
               2'b11: out = i3;
               default: out = 1'bx;
         endcase
   end
endmodule
```

05 marks

ii) 4-bit counter with asynchronous reset.

```
module counter(Q , clock, clear);
output [3:0] Q;
input clock, clear;
reg [3:0] Q;
always @(posedge clear or negedge clock)
begin
if (clear)
   Q = 4'd0;
else
   Q = (Q + 1) % 16;
end
endmodule
```

05 marks

5. Describe the following statements with an example: initial and always. Also design a clock with time period = 40 and a duty cycle of 25% by using the **always** and **initial** statements. The value of clock at time = 0 should be initialized to 0. 10 marks

"initial":

All statements inside an i n i t i a l statement constitute an "initial" block. An "initial" block starts at time 0, executes exactly once during a simulation, and then does not execute again. If there are multiple "initial" blocks, each block starts to execute concurrently at time 0. Each block finishes execution independently of other blocks. Multiple behavioral statements must be grouped, typically using the keywords 'begin' and 'end'.

Example:

```
module stimulus;
reg x, y, a, b, m;
initial
     m = 1'b0;
initial
begin
     #5 a = 1'b1;
     #25 b = 1'b0;
end
initial
begin
#10 x = 1'b0;
#25 y = 1'bl;
end
initial
```

```
#50 $finish;
endmodule
```

04 marks

"always":
All behavioral statements inside an '**always'** statement constitute an '**always'** block. The '**always'** statement starts at time '0' and executes the statements in the '**always'** block continuously in a looping fashion. This statement is used to model a block of activity that is repeated continuously in a digital circuit.
Example:

```
module clock_gen;
reg clock;
initial
clock = l'b0;
always
#l0 clock = ~clock;
initial
#l000 $finish;
endmodule
```

04 marks

Clock with time period = 40 and a duty cycle of 25% by using the **always** and **initial** statements. The value of clock at time = 0 should be initialized to 0:

```
module clock_gen;
reg clock;
initial
clock = 1'b0;
always
begin
    #30 clock = ~clock;
    #10 clock = ~clock;
end
endmodule
```

02 marks

6. Write Verilog HDL program to simulate traffic signal controller. Also write the stimulus for the same. 10 Marks

Traffic signal controller design:

```
`define TRUE 1'b1
`define FALSE 1'b0
`define RED 2'd0
`define YELLOW 2'd1
`define GREEN 2'd2

`define S0 3'dO
`define S1 3'd1
`define S2 3'd2
`define S3 3'd3
`define S4 3'd4
`define Y2RDELAY 3
`define RZGDELAY 2
```

```
module sig_control(hwy, cntry, X, clock, clear);
output [1:0] hwy, cntry;
reg [1:0] hwy, cntry;
input X;
input clock, clear;
reg [2:0] state;
reg [2:0] next_state;
initial
begin
	state = `S0;
	next_state = `S0;
	hwy = `GREEN;
	cntry = `RED;
end
always @(posedge clock)
state = next_state;
always @(state)
begin
	case (state)
		`S0 : begin
			hwy = `GREEN;
			cntry = `RED;
		end
		`S1 : begin
			hwy = `YELLOW;
			cntry = `RED;
		end
		`S2 : begin
			hwy = `RED;
			cntry = `RED;
		end
		`S3 : begin
			hwy = `RED;
			cntry = `GREEN;
		end
		`S4 : begin
			hwy = `RED;
			cntry = `YELLOW;
		end
	endcase
end
always @(state or clear or X)
	begin
		if (clear)
			next_state = `SO;
		else
			case (state)
				`S0: if ( X)
					next_state = `Sl;
				else
```

```
                                    next_state = `SO;
                        `S1: begin
                                repeat(`Y2RDELAY) @(posedge clock) ;
                                next_state = `S2;
                                end
                        `S2: begin
                                repeat ( `R2GDELAY) @ (posedge clock)
                                next_state = `S3;
                                end
                        `S3: if( X)
                                    next_state = `S3;
                                else
                                    next_state = `S4;
                        `S4: begin
                                repeat(`Y2RDELAY) @(posedge clock) ;
                                next_state = `SO;
                                end
                            default: next_state = `SO;
                    endcase
    end
endmodule
```

06 marks

Stimulus:

```
module stimulus;
wire [1:0] MAIN_SIG, CNTRY_SIG;
reg CAR_ON_CNTRY_RD;
reg CLOCK, CLEAR;
sig_control SC(MAIN_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD, CLOCK,
CLEAR);
initial
$monitor($time, " Main Sig = %b, Country Sig = %b, Car-on-cntry =
%b",MAIN_SIG, CNTRY_SIG, CAR_ON_CNTRY_RD) ;
initial
begin
CLOCK = `FALSE;
forever #5 CLOCK = -CLOCK;
end
initial
begin
    CLEAR = `TRUE;
    repeat (5) @(negedge CLOCK)
    CLEAR = `FALSE;
end
initial
begin
    CAR_ON_CNTRY_RD = 'FALSE;
    #200 CAR_ON_CNTRY_RD = 'TRUE;
    #100 CAR_ON_CNTRY_RD = 'FALSE;
    #200 CAR_ON_CNTRY_RD = 'TRUE;
    #100 CAR_ON_CNTRY_RD = 'FALSE;
```

```
    #200 CAR_ON_CNTRY_RD = 'TRUE;
    #100 CAR_ON_CNTRY_RD = 'FALSE;
    #100 $stop;
end
endmodule
```

04 marks

7. a) Using a **while** loop, design a clock generator. Initial value of *clock* is 0. Time period for the clock is 10. 05 Marks

```
module clock_gen;
reg clock;
initial
        begin
                clock = 1'b0;
                while(!clock)
                begin
                        #5 clock = 1'b1;
                        #5 clock = 1'b0;
                end
        end
endmodule
```

05 marks

b) Using the **for** loop, initialize locations 0 to 1023 of a 4-bit register array *cache_var* to 0. 05 Marks

```
`define loc 1024
integer i;
initial
begin
for(i = 0; I < `loc ; i= i+1)
   cache_var[i] = 4'b0000;
end
```

05 marks