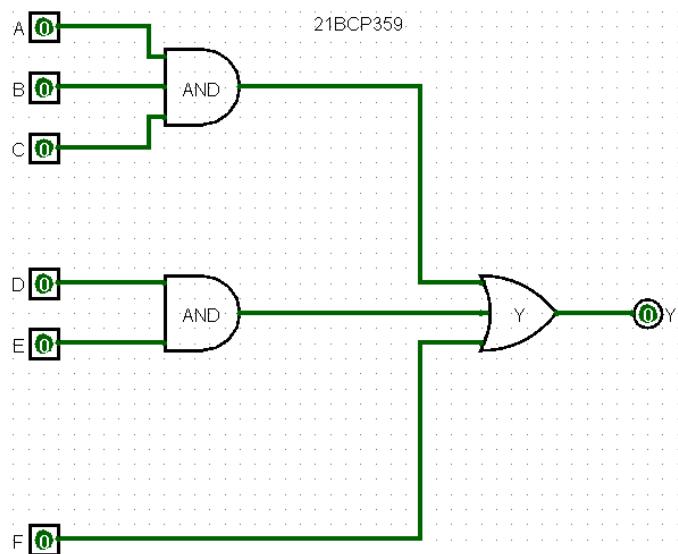


Lab 1: Basic Gate Operations

Question 1: Design the following logic functions using basic gates and test the output using truth table

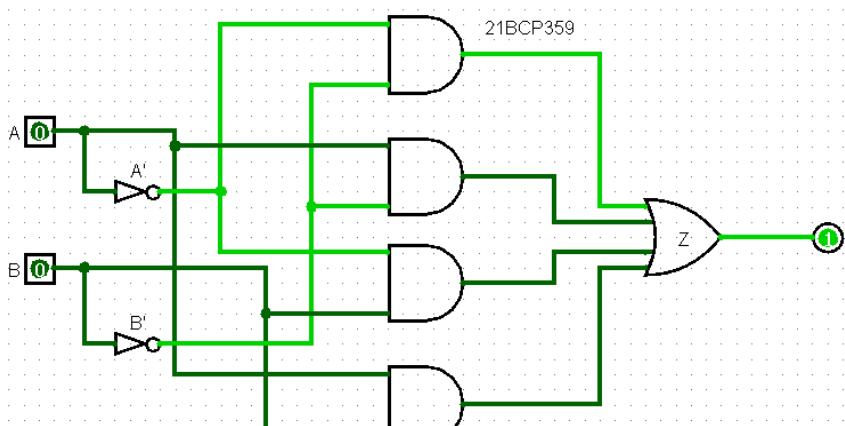
a. $Y = ABC + DE + F$



| A | B | C | D | E | F | Y |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |

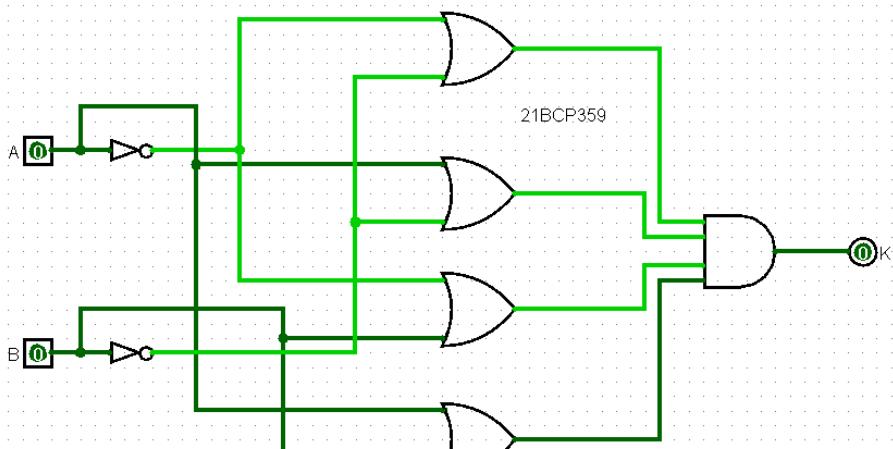
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

b. $Z = A'B' + AB' + A'B + AB$



| A | B | x |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

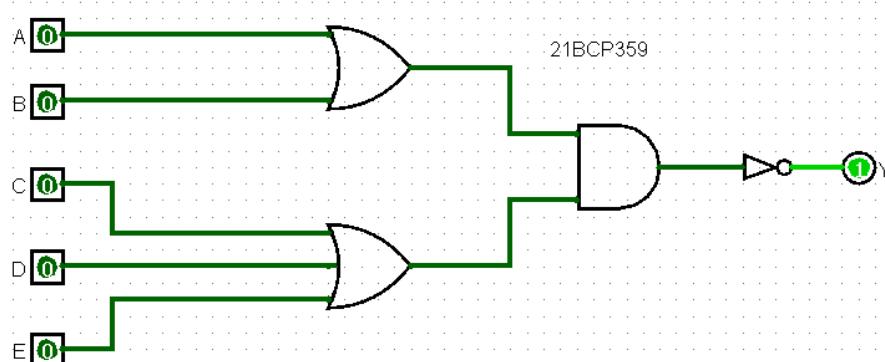
c. $K = (A'+B') (A+B') (A'+B) (A+B)$



| A | B | K |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

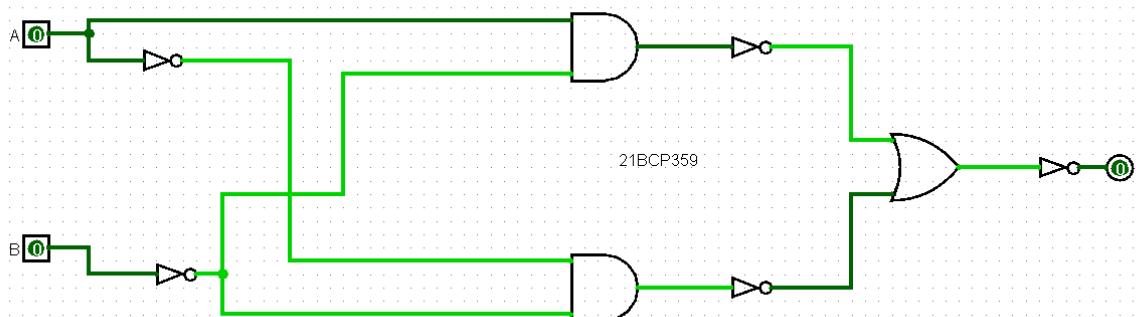
Question 2: Design and verify the functionality using existing library of basic gates in Logisim.

a. $Y = ((A+B).(C+D+E))'$



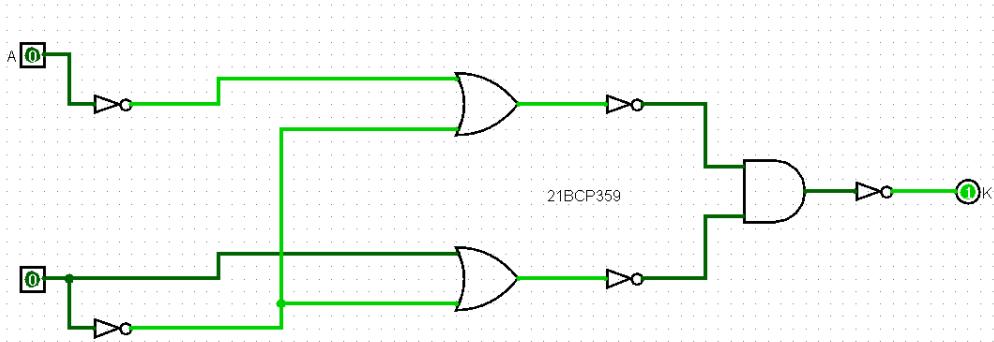
| A | B | C | D | E | Y |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 |

$$\text{b. } F = ((A \cdot B')' + ((A')' \cdot (B')'))'$$



| A | B | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

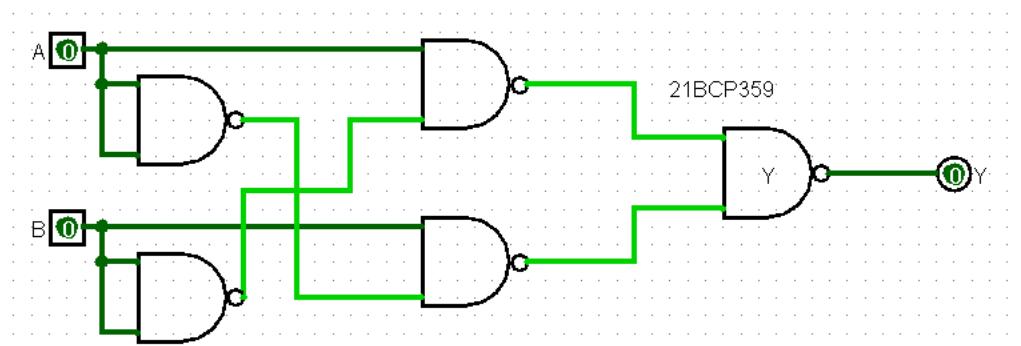
c. $K = (((A)' + B)' \cdot ((A)' + (B)'))'$



| A | a | K |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

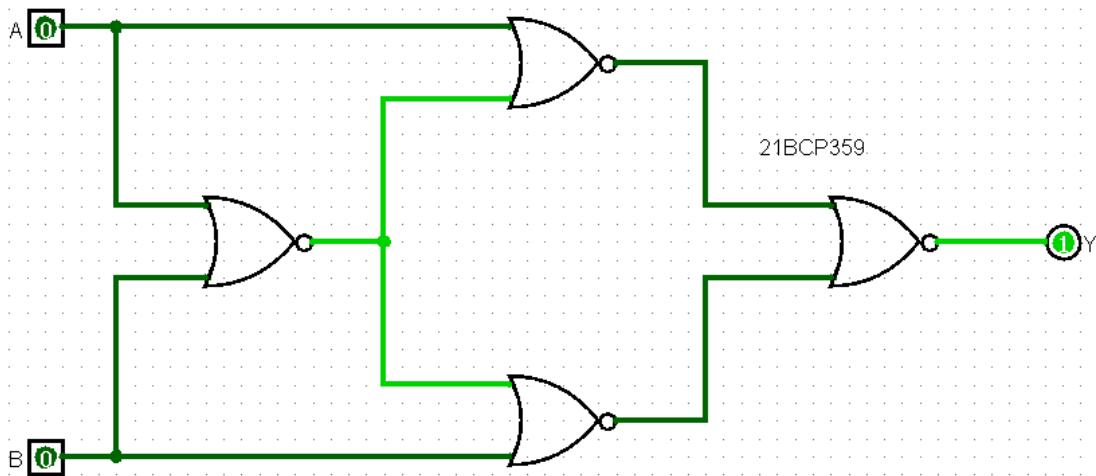
Question 3: For each of the following problem statement, design a circuit, describe a Truth table and verify the outcome using Logisim

a. Implement the functionality of XOR gate using only NAND gates.



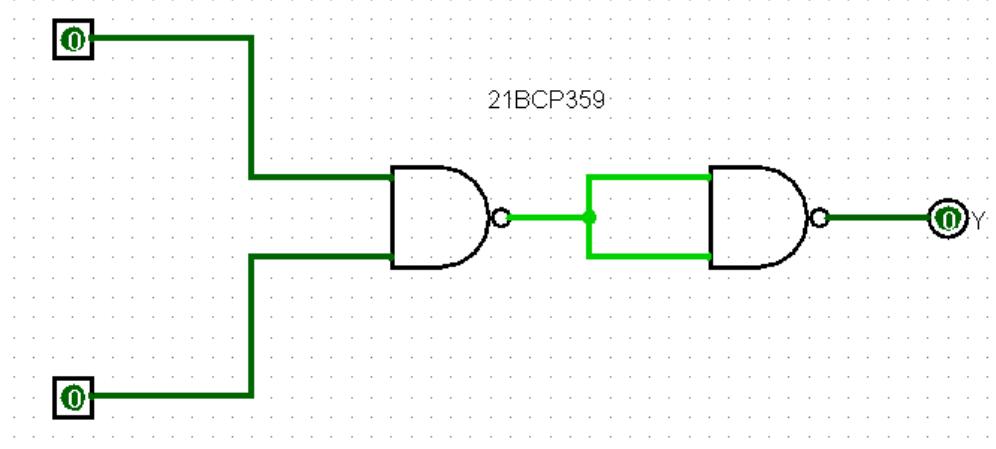
| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

b. Implement the functionality of XNOR gate using only NOR gates.



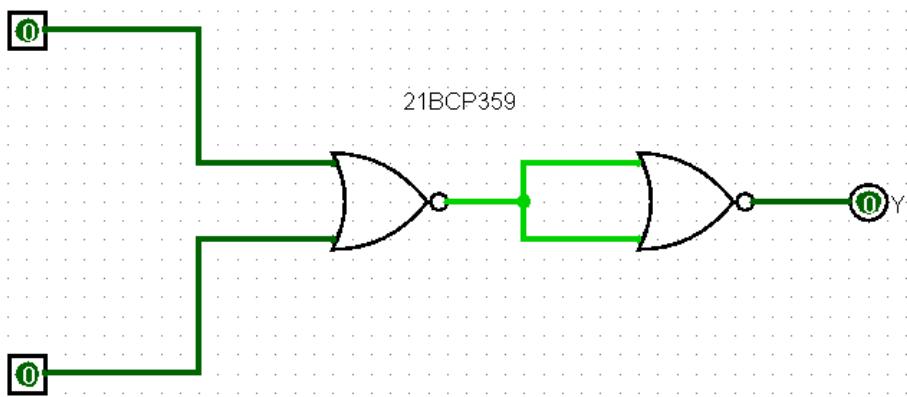
| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

c. Implement the functionality of AND gate using only NAND gates.



| a | b | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

d. Implement the functionality of OR gate using only NOR gates.



| a | b | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

e. Implement the functionality of NOT gate using only NAND gate.



| a | Y |
|---|---|
| 0 | 1 |
| 1 | 0 |

Lab 2: Introduction To Verilog

Question 1: Write a Verilog code to implement AND gate. Write the corresponding Testbench code for the verification of your Verilog code

```
testbench.sv +
```

```

1 // Code your testbench here
2 // or browse Examples
3
4 module tb_and_gate;
5
6 reg A,B;
7 wire Y;
8 and_gate a1 (.a(A), .b(B), .y(Y));
9
10 initial
11 begin
12     A = 0; B =0; #5;
13     A = 0; B =1; #5;
14     A = 1; B =0; #5;
15     A = 1; B =1; #5;
16 end
17
18 initial
19 begin
20     $dumpfile("dump.vcd");
21     $dumpvars(1);
22 end
23
24 endmodule
25

```

```
design.sv +
```

```

1 // Code your design here
2 module and_gate(
3
4     input a,b,
5     output y);
6     assign y = a & b;
7
8 endmodule

```

A timing diagram showing three signals over time. The top signal is labeled 'A' and has a value of 0 at time 0, rising to 1 at time 5, and staying at 1 until time 10. The middle signal is labeled 'B' and has a value of 0 at time 0, rising to 1 at time 5, and staying at 1 until time 10. The bottom signal is labeled 'Y' and has a value of 0 at time 0, rising to 1 at time 5, and staying at 1 until time 10.

Question 2: Write a Verilog code to implement OR gate. Write the corresponding Testbench code for the verification of your Verilog code.

```
testbench.sv +
```

```

1 // Code your testbench here
2 // or browse Examples
3
4 module tb_or_gate;
5
6 reg A,B;
7 wire Y;
8 or_gate a1 (.a(A), .b(B), .y(Y));
9
10 initial
11 begin
12     A = 0; B =0; #5;
13     A = 0; B =1; #5;
14     A = 1; B =0; #5;
15     A = 1; B =1; #5;
16 end
17
18 initial
19 begin
20     $dumpfile("dump.vcd");
21     $dumpvars(1);
22 end
23
24 endmodule
25

```

```
design.sv +
```

```

1 // Code your design here
2 module or_gate(
3
4     input a,b,
5     output y);
6     assign y = a | b;
7
8 endmodule

```

A timing diagram showing three signals over time. The top signal is labeled 'A' and has a value of 1 at time 0, falling to 0 at time 5, and staying at 0 until time 10. The middle signal is labeled 'B' and has a value of 1 at time 0, falling to 0 at time 5, and staying at 0 until time 10. The bottom signal is labeled 'Y' and has a value of 0 at time 0, rising to 1 at time 5, and staying at 1 until time 10.

LAB 2

Question 3: Write a Verilog code to implement XOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```

1 // Code your testbench here
2 // or browse Examples
3
4 module tb_xor_gate;
5
6 reg A,B;
7 wire Y;
8 xor_gate a1 (.a(A), .b(B), .y(Y));
9
10 initial
11 begin
12     A = 0; B =0; #5;
13     A = 0; B =1; #5;
14     A = 1; B =0; #5;
15     A = 1; B =1; #5;
16 end
17
18 initial
19 begin
20     $dumpfile("dump.vcd");
21     $dumpvars(1);
22 end
23
24 endmodule
25

```

design.sv

```

1 // Code your design here
2 module xor_gate(
3
4     input a,b,
5     output y);
6     assign y = ~a&b | a&~b;
7
8 endmodule

```



Question 4: Write a Verilog code to implement NOR gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```

1 // Code your testbench here
2 // or browse Examples
3
4 module tb_nor_gate;
5
6 reg A,B;
7 wire Y;
8 nor_gate a1 (.a(A), .b(B), .y(Y));
9
10 initial
11 begin
12     A = 0; B =0; #5;
13     A = 0; B =1; #5;
14     A = 1; B =0; #5;
15     A = 1; B =1; #5;
16 end
17
18 initial
19 begin
20     $dumpfile("dump.vcd");
21     $dumpvars(1);
22 end
23
24 endmodule
25

```

design.sv

```

1 // Code your design here
2 module nor_gate(
3
4     input a,b,
5     output y);
6     assign y = ~(a | b);
7
8 endmodule

```



Question 5: Write a Verilog code to implement NAND gate. Write the corresponding Testbench code for the verification of your Verilog code.

testbench.sv

```

1 // Code your testbench here
2 // or browse Examples
3
4 module tb_nand_gate;
5
6   reg A,B;
7   wire Y;
8   nand_gate a1 (.a(A), .b(B), .y(Y));
9
10 initial
11 begin
12   A = 0; B =0; #5;
13   A = 0; B =1; #5;
14   A = 1; B =0; #5;
15   A = 1; B =1; #5;
16 end
17
18 initial
19 begin
20   $dumpfile("dump.vcd");
21   $dumpvars(1);
22 end
23
24 endmodule
25

```

design.sv

```

1 // Code your design here
2 module nand_gate(
3
4   input a,b,
5   output y);
6   assign y = ~(a & b);
7
8 endmodule

```



Question 6: Write a Verilog code to implement XNOR gate. Write the corresponding Testbench code for the verification of your Verilog code

testbench.sv

```

1 // Code your testbench here
2 // or browse Examples
3
4 module tb_xnor_gate;
5
6   reg A,B;
7   wire Y;
8   xnor_gate a1 (.a(A), .b(B), .y(Y));
9
10 initial
11 begin
12   A = 0; B =0; #5;
13   A = 0; B =1; #5;
14   A = 1; B =0; #5;
15   A = 1; B =1; #5;
16 end
17
18 initial
19 begin
20   $dumpfile("dump.vcd");
21   $dumpvars(1);
22 end
23
24 endmodule
25

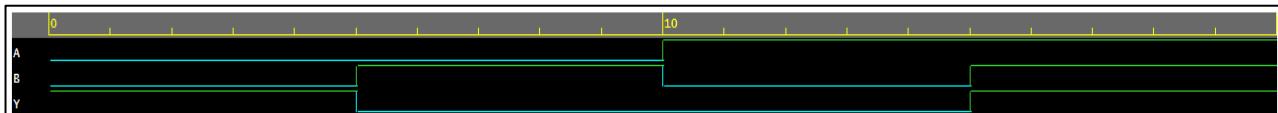
```

design.sv

```

1 // Code your design here
2 module xnor_gate(
3
4   input a,b,
5   output y);
6   assign y = ~(~a&b | a&~b);
7
8 endmodule

```



Lab 3: Implementation of Structural & Behavioral Verilog Code

Question 1: Implement the following expression using the Verilog HDL. Moreover, Verify your circuit against the waveform.

a. $F(A, B, C) = A'BC + AB'C + ABC$

```

testbench.sv
1 module tb_exp;
2   reg K,L,M;
3   wire N;
4   build_exp x1 (.k(K), .l(L), .m(M), .n(N));
5   initial
6     begin
7       K = 1; L = 1; M = 1; #5;
8       K = 1; L = 1; M = 0; #5;
9       K = 1; L = 0; M = 1; #5;
10      K = 1; L = 0; M = 0; #5;
11      K = 0; L = 1; M = 1; #5;
12      K = 0; L = 1; M = 0; #5;
13      K = 0; L = 0; M = 1; #5;
14      K = 0; L = 0; M = 0; #5;
15    end
16  initial
17    begin
18      $dumpfile("dump.vcd");
19      $dumpvars(1);
20    end
21 endmodule

```

```

design.sv
1 module and_gate(input a,b,c, output d);
2   assign d = a&b&c;
3 endmodule
4
5 module not_gate(input e, output f);
6   assign f = ~e;
7 endmodule
8
9 module or_gate(input p,q,r, output s);
10  assign t = p|q|r|s;
11 endmodule
12
13 module build_exp(input k,l,m, output n);
14   wire w,x,y, k_not,l_not,m_not;
15
16   not_gate n1 (.e(k), .f(k_not));
17   not_gate n2 (.e(l), .f(l_not));
18   not_gate n3 (.e(m), .f(m_not));
19
20   and_gate a1 (.a(k_not), .b(l), .c(m), .d(w));
21   and_gate a2 (.a(k), .b(l_not), .c(m), .d(x));
22   and_gate a3 (.a(k), .b(l), .c(m), .d(y));
23
24   or_gate o1 (.p(w), .q(x), .r(y), .s(z));
25 endmodule

```

b. $F(A, B, C, D) = ABCD' + A'BCD + AB'CD' + ABC$

```

testbench.sv
1 module tb_exp;
2   reg K,L,M,N,O;
3   wire O;
4   build_exp x1 (.k(K), .l(L), .m(M), .n(N), .o(O));
5   initial
6     begin
7       K = 1; L = 1; M = 1; N = 1; #5;
8       K = 1; L = 1; M = 1; N = 0; #5;
9       K = 1; L = 1; M = 0; N = 1; #5;
10      K = 1; L = 1; M = 0; N = 0; #5;
11      K = 1; L = 0; M = 1; N = 1; #5;
12      K = 1; L = 0; M = 1; N = 0; #5;
13      K = 1; L = 0; M = 0; N = 1; #5;
14      K = 1; L = 0; M = 0; N = 0; #5;
15      K = 0; L = 1; M = 1; N = 1; #5;
16      K = 0; L = 1; M = 1; N = 0; #5;
17      K = 0; L = 1; M = 0; N = 1; #5;
18      K = 0; L = 1; M = 0; N = 0; #5;
19      K = 0; L = 0; M = 1; N = 1; #5;
20      K = 0; L = 0; M = 1; N = 0; #5;
21      K = 0; L = 0; M = 0; N = 1; #5;
22      K = 0; L = 0; M = 0; N = 0; #5;
23    end
24  initial
25    begin
26      $dumpfile("dump.vcd");
27      $dumpvars(1);
28    end
29 endmodule

```

```

design.sv
1 module and_gate1(input g,h,i, output j);
2   assign j = g&h&i;
3 endmodule
4
5 module and_gate2(input a,b,c,d, output e);
6   assign e = a&b&c&d;
7 endmodule
8
9 module not_gate(input e, output f);
10  assign f = ~e;
11 endmodule
12
13 module or_gate(input p,q,r,s, output t);
14  assign t = p|q|r|s;
15 endmodule
16
17 module build_exp(input k,l,m,n, output o);
18   wire w,x,y,z,out, k_not,l_not,m_not,n_not;
19
20   not_gate n1 (.e(k), .f(k_not));
21   not_gate n2 (.e(l), .f(l_not));
22   not_gate n3 (.e(m), .f(m_not));
23   not_gate n4 (.e(n), .f(n_not));
24
25   and_gate2 a1 (.a(k), .b(l), .c(m), .d(n_not), .e(w));
26   and_gate2 a2 (.a(k), .b(l_not), .c(m), .d(n), .e(x));
27   and_gate2 a3 (.a(k), .b(l_not), .c(m), .d(n_not), .e(y));
28   and_gate1 a4 (.g(k), .h(l), .i(m), .j(z));
29
30   or_gate o1 (.p(w), .q(x), .r(y), .s(z), .t(out));
31 endmodule

```

c. $F(A, B) = (A' + B')(A + B')(A' + B)(A + B)$

```
testbench.sv
1 // Code your testbench here
2 // or browse Examples
3 module tb_andgate;
4 reg A, B;
5 wire Y;
6 andgate a1 (.a(A), .b(B), .y(Y));
7 initial
8 begin
9     A=0;B=0;#5;
10    A=0;B=1;#5;
11    A=1;B=0;#5;
12    A=1;B=1;#5;
13 end
14 initial
15 begin
16     $dumpfile("dump.vcd");
17     $dumpvars(1);
18 end
19 endmodule
```

```
design.sv
1 // Code your design here
2 module andgate(
3     input a,b,
4     output y);
5 assign y = (~a | ~b) & (a | ~b) & (~a | b) & (a | b);
6 endmodule
```

d. $(A, B) = ((A \cdot B')' + ((A')' \cdot (B')')')$

```
testbench.sv
1 // Code your testbench here
2 // or browse Examples
3 module tb_andgate;
4 reg A, B;
5 wire Y;
6 andgate a1 (.a(A), .b(B), .y(Y));
7 initial
8 begin
9     A=0;B=0;#5;
10    A=0;B=1;#5;
11    A=1;B=0;#5;
12    A=1;B=1;#5;
13 end
14 initial
15 begin
16     $dumpfile("dump.vcd");
17     $dumpvars(1);
18 end
19 endmodule
```

```
design.sv
1 // Code your design here
2 module andgate(
3     input a,b,
4     output y);
5 assign y = ~((a & ~b) | (~(~a & ~b)));
6 endmodule
```

$$e. \quad (A, B) = (((A)' + B)' \cdot ((A)' + (B')'))'$$

```

testbench.sv
1 // Code your testbench here
2 // or browse Examples
3 module tb_andgate;
4   reg A, B;
5   wire Y;
6   andgate a1 (.a(A),.b(B),.y(Y));
7   initial
8     begin
9       A=0;B=0;#5;
10      A=0;B=1;#5;
11      A=1;B=0;#5;
12      A=1;B=1;#5;
13    end
14  initial
15    begin
16      $dumpfile("dump.vcd");
17      $dumpvars(1);
18    end
19 endmodule
20

design.sv
1 // Code your design here
2 module andgate(
3   input a,b,
4   output y);
5   assign y = ~(~(~a | b)) & (~(~a | ~b));
6 endmodule

```

Question 2: Implement the XOR gate using Behavioral and Structural code of Verilog Hardware Description Language.

```

testbench.sv
1 module tb_xor_str;
2   reg A,B;
3   wire C;
4   build_xor x1 (.m(A), .n(B), .o(C));
5   initial
6     begin
7       A = 0; B = 0; #5;
8       A = 0; B = 1; #5;
9       A = 1; B = 0; #5;
10      A = 1; B = 1; #5;
11    end
12  initial
13    begin
14      $dumpfile("dump.vcd");
15      $dumpvars(1);
16    end
17 endmodule

design.sv
1 // Behavioural Code
2 module not_gate (input e, output f);
3   assign f = ~e;
4 endmodule
5 module and_gate (input a, b, output c);
6   assign c = a & b;
7 endmodule
8 module or_gate (input p, q, output r);
9   assign r = p | q;
10 endmodule
11 module build_xor (input m, n, output o);
12   wire x, y, a_not, b_not;
13   not_gate n1 (.e(m),.f(a_not));
14   not_gate n2 (.e(n),.f(b_not));
15   and_gate a1 (.a(a_not),.b(n),.c(x));
16   and_gate a2 (.a(m),.b(b_not),.c(y));
17   or_gate o1 (.p(x),.q(y),.r(o));
18 endmodule
19
20
21 // Structural Code
22 module Lab4_Build_XOR(a, b, c);
23   input a,b;
24   output c;
25   wire a_not, b_not;
26   wire x,y;
27   not(a_not,a);
28   not(b_not,b);
29   and(x,a_not,b);
30   and(y,a,b_not);
31   or(c,x,y);
32 endmodule
33

```

Question 3: Implement the following expression using the Verilog Hardware Description Language (HDL) (Structural Coding).

a. $F(A, B, C) = (A' + B' + C')(A + B' + C')(A' + B + C')(A' + B')'$

```

testbench.sv
1 // Code your testbench here
2 // or browse Examples
3 module tb_xor;
4   reg A, B, C;
5   wire D;
6   buildxor x1 (.m(A), .n(B), .o(C), .t(D));
7   initial
8     begin
9       A = 0; B = 0; C = 0; #5;
10      A = 0; B = 1; C = 0; #5;
11      A = 1; B = 0; C = 0; #5;
12      A = 1; B = 1; C = 0; #5;
13      A = 0; B = 0; C = 1; #5;
14      A = 0; B = 1; C = 1; #5;
15      A = 1; B = 0; C = 1; #5;
16      A = 1; B = 1; C = 1; #5;
17    end
18  initial
19    begin
20      $dumpfile("dump.vcd");
21      $dumpvars(1);
22    end
23 endmodule
24

design.sv
1 // Code your design here
2 module notgate (input e, output f);
3   assign f = ~e;
4 endmodule
5
6 module andgate (input a, b, c, output d);
7   assign d = a & b & c;
8 endmodule
9
10 module orgate (input p, q, output r);
11   assign r = p | q;
12 endmodule
13
14 module buildxor (input m, n, o, output t);
15   wire w, x, y, z;
16   notgate n1 (.e(m), .f(bnot));
17   notgate n2 (.e(o), .f(cnot));
18   andgate a1 (.a(w), .b(bnot), .c(o), .d(w));
19   andgate a2 (.a(w), .b(bnot), .c(cnot), .d(x));
20   orgate o1 (.p(x), .q(w), .r(t));
21 endmodule
22

```

Question 3: Implement the following expression using universal NAND and NOR gate. Write down Verilog Structural and Behavioral code for that expression.

a. $F(A, B, C) = (AB'C) + (AB'C')$

```

testbench.sv
1 module tb_exp;
2   reg K,L,M;
3   wire N;
4   build_exp x1 (.k(K), .l(L), .m(M), .n(N));
5   initial
6     begin
7       K = 1; L = 1; M = 1; #5;
8       K = 1; L = 1; M = 1; #5;
9       K = 1; L = 1; M = 0; #5;
10      K = 1; L = 1; M = 0; #5;
11      K = 1; L = 0; M = 1; #5;
12      K = 1; L = 0; M = 1; #5;
13      K = 1; L = 0; M = 0; #5;
14      K = 1; L = 0; M = 0; #5;
15      K = 0; L = 1; M = 1; #5;
16      K = 0; L = 1; M = 1; #5;
17      K = 0; L = 1; M = 0; #5;
18      K = 0; L = 0; M = 1; #5;
19      K = 0; L = 0; M = 1; #5;
20      K = 0; L = 0; M = 0; #5;
21      K = 0; L = 0; M = 0; #5;
22      K = 0; L = 0; M = 0; #5;
23    end
24  initial
25    begin
26      $dumpfile("dump.vcd");
27      $dumpvars(1);
28    end
29 endmodule
29

design.sv
1 module build_exp(k,l,m,n);
2   input k,l,m;
3   output n;
4   wire w,x,y,z;
5   wire k_not,l_not,m_not;
6
7   not (k, k_not);
8   not (l, l_not);
9   not (m, m_not);
10
11  or (k_not, l_not, m_not, w);
12  or (k, l_not, m_not, x);
13  or (k_not, l, m_not, y);
14  or (k_not, l_not, l_not, z);
15
16  not (z, z_not);
17
18  and (w, x, y, z_not, out);
19
20 endmodule
20

```

b. $F(A, B) = A'B' + AB' + A'B + AB$

The screenshot shows two code editors side-by-side. The left editor, titled 'testbench.sv', contains a SystemVerilog testbench script. The right editor, titled 'design.sv', contains a SystemVerilog design script. Below the editors is a timing diagram window.

testbench.sv

```
1 // Code your testbench here
2 // or browse Examples
3 module tbxor;
4   reg A, B;
5   wire C;
6   buildxor x1 (.m(A), .n(B), .o(C));
7   initial
8     begin
9       A = 0; B = 0; #5;
10      A = 0; B = 1; #5;
11      A = 1; B = 0; #5;
12      A = 1; B = 1; #5;
13    end
14  initial
15    begin
16      $dumpfile("dump.vcd");
17      $dumpvars(1);
18    end
19 endmodule
20
```

design.sv

```
1 // Code your design here
2 module notgate (input e, output f);
3   assign f = ~e;
4 endmodule
5
6 module andgate (input a, b, output c);
7   assign c = a & b;
8 endmodule
9
10 module orgate (input p, q, r, s, output t);
11   assign t = p | q | r | s;
12 endmodule
13
14 module buildxor (input m, n, output o);
15   wire w, x, y, z, anot, bnot;
16   notgate n1 (.e(m), .f(anot));
17   notgate n2 (.e(n), .f(bnot));
18   andgate a1 (.a(anot), .b(bnot), .c(x));
19   andgate a2 (.a(m), .b(bnot), .c(y));
20   andgate a3 (.a(anot), .b(n), .c(z));
21   andgate a4 (.a(m), .b(n), .c(w));
22   orgate o1 (.p(x), .q(y), .r(z), .s(w), .t(o));
23 endmodule
24
```

Question 4 Write down three modules in a single Verilog code to design AND, OR and NOT gate. Now use those modules to design the XOR gate. Use AND, OR and NOT gate as the instances to implement the XOR gate. Write the corresponding Testbench code for the verification of your XOR gate.

```
testbench.sv
```

```
1 module tb_xor_str;
2   reg A,B;
3   wire C;
4   build_xor x1 (.m(A), .n(B), .o(C));
5   initial
6     begin
7       A = 0; B = 0; #5;
8       A = 0; B = 1; #5;
9       A = 1; B = 0; #5;
10      A = 1; B = 1; #5;
11    end
12  initial
13    begin
14      $dumpfile("dump.vcd");
15      $dumpvars(1);
16    end
17 endmodule
```

```
design.sv
```

```
1 module and_gate(input a, b, output c);
2   assign c = a&b;
3 endmodule
4
5 module not_gate(input e, output f);
6   assign f = ~e;
7 endmodule
8
9 module or_gate(input p,q, output r);
10  assign r = p|q;
11 endmodule
12
13 module build_xor(input m, n, output o);
14   wire x, y, a_not, b_not;
15   not_gate n1 (.e(m),.f(a_not));
16   not_gate n2 (.e(n),.f(b_not));
17   and_gate a1 (.a(a_not),.b(n),.c(x));
18   and_gate a2 (.a(m),.b(b_not),.c(y));
19   or_gate o1 (.p(x),.q(y),.r(o));
20 endmodule
```

Question 5 Consider the following expression: $Y = A' \cdot B' \cdot C' + A' \cdot B \cdot C' + A \cdot B' \cdot C' + A \cdot B' \cdot C$

Design one three input ‘And’ gate module and one four input ‘OR’ gate module using Verilog. Instantiate those two modules to design the above-mentioned expression. Design the corresponding Testbench code for the verification purpose.

| A | B | C | Y |
|---|---|---|---|
| F | F | F | T |
| F | F | T | F |
| F | T | F | T |
| F | T | T | F |
| T | F | F | T |
| T | F | T | T |
| T | T | F | F |
| T | T | T | F |

Reduced Expression: $A' \cdot B' \cdot C'$

```
testbench.sv +
```

```

1 module tb_exp;
2   reg K,L,M;
3   wire N;
4   build_exp x1 (.k(k), .l(l), .m(m), .n(n));
5   initial
6     begin
7       K = 1; L = 1; M = 1; #5;
8       K = 1; L = 1; M = 0; #5;
9       K = 1; L = 0; M = 1; #5;
10      K = 1; L = 0; M = 0; #5;
11      K = 0; L = 1; M = 1; #5;
12      K = 0; L = 1; M = 0; #5;
13      K = 0; L = 0; M = 1; #5;
14      K = 0; L = 0; M = 0; #5;
15   end
16   initial
17   begin
18     $dumpfile("dump.vcd");
19     $dumpvars(1);
20   end
21 endmodule

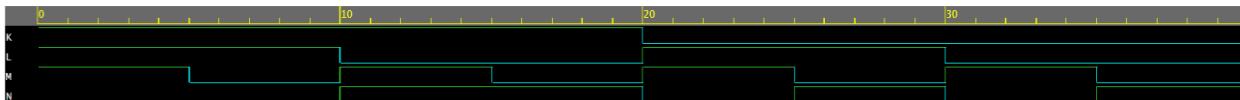
```

```
design.sv +
```

```

1 module and_gate(input a,b,c , output d);
2   assign d = a&b&c;
3 endmodule
4
5 module not_gate(input e, output f);
6   assign f = ~e;
7 endmodule
8
9 module or_gate(input p,q,r,s, output t);
10  assign t = p|q|r|s;
11 endmodule
12
13 module build_exp(input k,l,m, output n);
14   wire w,x,y,z, k_not,l_not,m_not;
15
16   not_gate n1 (.e(k), .f(k_not));
17   not_gate n2 (.e(l), .f(l_not));
18   not_gate n3 (.e(m), .f(m_not));
19
20   and_gate a1 (.a(k_not), .b(l_not), .c(m_not), .d(w));
21   and_gate a2 (.a(k_not), .b(l), .c(m_not), .d(x));
22   and_gate a3 (.a(k), .b(l_not), .c(m_not), .d(y));
23   and_gate a4 (.a(k), .b(l_not), .c(m), .d(z));
24
25   or_gate o1 (.p(w), .q(x), .r(y), .s(z), .t(n));
26 endmodule

```



Lab 4: Design Half Adder & Full Adder Using Half Adder

Question 1: Write Verilog code for half adder. Test using waveform

The screenshot shows a simulation environment with two tabs: "testbench.sv" and "design.sv".

testbench.sv:

```

1 module tbxorgate;
2   reg A, B;
3   wire S, C;
4   xorgate a1 (.a(A), .b(B), .sum(S), .carry(C));
5   initial
6     begin
7       A=0;B=0;#5;
8       A=0;B=1;#5;
9       A=1;B=0;#5;
10      A=1;B=1;#5;
11    end
12    initial
13    begin
14      $dumpfile("dump.vcd");
15      $dumpvars(1);
16    end
17 endmodule

```

design.sv:

```

1 module xorgate(
2   input a,b,
3   output sum, carry);
4   assign sum = ~a & b || a & ~b;
5   assign carry = a & b;
6 endmodule

```

Waveform:

The waveform shows four signals over time: A, B, C, and S. Signal A starts at 0, B starts at 0, and both transition to 1 at time 10. Signal C starts at 0 and transitions to 1 at time 10. Signal S starts at 0 and transitions to 1 at time 10.

Question 3. Write Verilog code for full adder. Test using waveform.

The screenshot shows a simulation environment with two tabs: "testbench.sv" and "design.sv".

testbench.sv:

```

1 module tbxorgate;
2   reg A, B, C;
3   wire S, CA;
4   xorgate a1 (.a(A), .b(B), .c(C), .sum(S), .carry(CA));
5   initial
6     begin
7       A=0;B=0;C=0;#5;
8       A=0;B=1;C=0;#5;
9       A=1;B=0;C=0;#5;
10      A=1;B=1;C=0;#5;
11      A=0;B=0;C=1;#5;
12      A=0;B=1;C=1;#5;
13      A=1;B=0;C=1;#5;
14      A=1;B=1;C=1;#5;
15    end
16    initial
17    begin
18      $dumpfile("dump.vcd");
19      $dumpvars(1);
20    end
21 endmodule

```

design.sv:

```

1 module xorgate(
2   input a,b,c,
3   output sum, carry);
4   assign sum = (~(~a & b || a & ~b) & c) | ((~a & b || a & ~b) & ~c);
5   assign carry = (a & b) | (b & c) | (c & a);
6 endmodule

```

SV/Verilog Design:

Waveform:

The waveform shows five signals over time: A, B, C, CA, and S. Signals A, B, and C all transition from 0 to 1 at time 10. Signal CA starts at 0 and transitions to 1 at time 20. Signal S starts at 0 and transitions to 1 at time 20.

Question 4. Write Verilog code for full adder using module instantiation of half adder. Test using waveform.

design.sv +

```

1 module halfadder (input a, b, output c, s);
2   assign c = a & b;
3   assign s = (~a & b) | (a & ~b);
4 endmodule
5
6 module orgate (input i, j, output k);
7   assign k = i | j;
8 endmodule
9
10 module fulladder(input m, n, o, output p , q);
11   wire sum1, carry1, carry2;
12   halfadder h1 (.a(m),.b(n),.s(sum1));
13   halfadder h2 (.a(m),.b(n),.c(carry1));
14   halfadder h3 (.a(sum1),.b(o),.s(p));
15   halfadder h4 (.a(sum1),.b(o),.c(carry2));
16   orgate or1 (.i(carry1),.j(carry2),.k(q));
17
18 endmodule
19

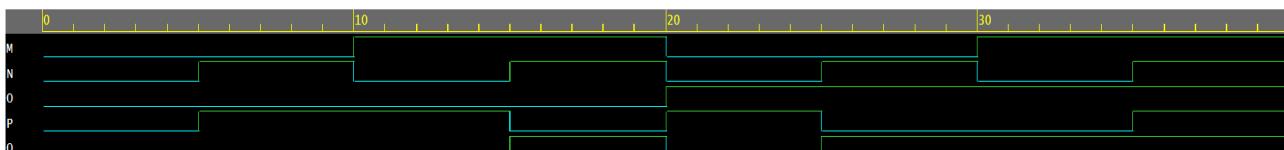
```

testbench.sv +

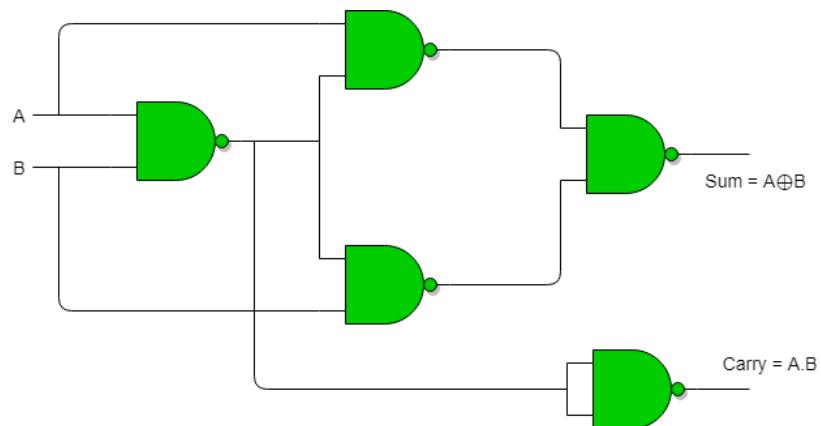
```

1 module fulladder_tb;
2   reg M, N, O;
3   wire P, Q;
4   fulladder x1 (.m(M), .n(N), .o(O), .p(P), .q(Q));
5   initial
6     begin
7       M = 0; N = 0; O = 0; #5;
8       M = 0; N = 1; O = 0; #5;
9       M = 1; N = 0; O = 0; #5;
10      M = 1; N = 1; O = 0; #5;
11      M = 0; N = 0; O = 1; #5;
12      M = 0; N = 1; O = 1; #5;
13      M = 1; N = 0; O = 1; #5;
14      M = 1; N = 1; O = 1; #5;
15    end
16  initial
17    begin
18      $dumpfile("dump.vcd");
19      $dumpvars(1);
20    end
21  endmodule
22

```



Question 5. Write Verilog code for half adder using only NAND gate. Test using waveform



testbench.sv

```

1 module tb_ha_nand;
2   reg a,b;
3   wire sum,carry;
4
5   ha_nand ha1(a,b,sum,carry);
6
7
8   initial
9     begin
10    a = 0; b = 0; #5;
11    a = 0; b = 1; #5;
12    a = 1; b = 0; #5;
13    a = 1; b = 1; #5;
14  end
15
16   initial
17   begin
18     $dumpfile("dump.vcd");
19     $dumpvars(1);
20   end
21 endmodule

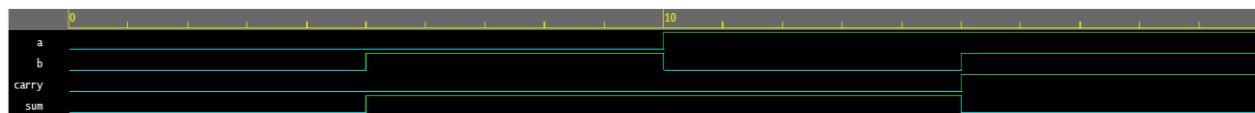
```

design.sv

```

1 module ha_nand(input a,b, output sum,carry);
2   wire temp1,temp2,temp3;
3   assign temp1 = ~(a & b);
4
5   assign temp2 = ~ (a & temp1);
6   assign temp3 = ~ (b & temp1);
7
8   assign sum = ~ (temp2 & temp3);
9
10  assign carry = ~ (temp1 & temp1);
11 endmodule

```



LAB 5: Design Half Subtractor, Full Subtractor Using Half Subtractors

Question 1: Write a Verilog code to design a Half subtractor and test it using the Waveform

design.sv
+

SV/Verilog Testbench

```

1 module HalfSubtractor(a,b,difference,borrow);
2   input a,b;
3   output difference,borrow;
4   xor(difference,a,b);
5   assign borrow=(~a&b);
6 endmodule

```


testbench.sv
+

```

1 module TestModule;
2   // Inputs
3   reg a;
4   reg b;
5
6   // Outputs
7   wire difference;
8   wire borrow;
9
10 HalfSubtractor uut (.a(a),.b(b), .difference(difference),
11 .borrow(borrow));
12 initial begin
13   // Initialize Inputs
14   a = 0;
15   b = 0;
16   // Wait 100 ns for global reset to finish
17   #100;
18   a = 1;
19   b = 0;
20 end
21 initial begin
22   $dumpfile("dump.vcd");
23   $dumpvars(1);
24 end
25 endmodule

```

Question 2: Write a Verilog code to design a Full subtractor using Half subtractors and test it using the Waveform.

testbench.sv
+

```

1 module top;
2   reg a, b, bin;
3   wire d, bout;
4
5   full_subtractor instantiation(.A(a), .B(b), .Bin(bin), .D(d), .Bout(bout));
6
7   initial
8     begin
9       $dumpfile("xyz.vcd");
10      $dumpvars;
11      a=0;
12      b=0;
13      bin=0;
14      #100 $finish;
15    end
16    always #40 a=~a;
17    always #20 b=~b;
18    always #10 bin=~bin;
19    always @(a or b or bin)
20      $monitor("At TIME(in ns)=%t, A=%d B=%d Bin=%d D = %d Bout = %d", $time, a, b, bin, d, bout);
21 endmodule

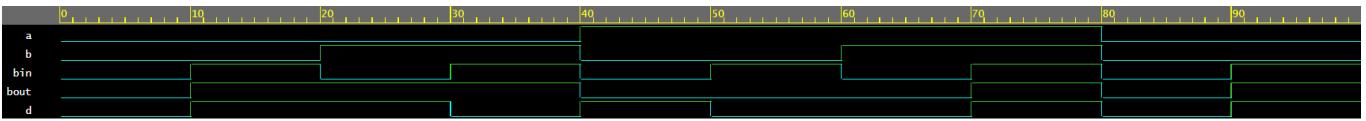
```

```
design.sv 
```

```

1 module or_gate(a0, b0, c0);
2   input a0, b0;
3   output c0;
4   assign c0 = a0 | b0;
5 endmodule
6
7 module xor_gate(a1, b1, c1);
8   input a1, b1;
9   output c1;
10  assign c1 = a1 ^ b1;
11 endmodule
12
13 module and_gate(a2, b2, c2);
14   input a2, b2;
15   output c2;
16   assign c2 = a2 & b2;
17 endmodule
18
19 module not_gate(a3, b3);
20   input a3;
21   output b3;
22   assign b3 = ~ a3;
23 endmodule
24
25 module half_subtractor(a4, b4, c4, d4);
26   input a4, b4;
27   output c4, d4;
28   wire x;
29   xor_gate u1(a4, b4, c4);
30   and_gate u2(x, b4, d4);
31   not_gate u3(a4, x);
32 endmodule
33
34 module full_subtractor(A, B, Bin, D, Bout);
35   input A, B, Bin;
36   output D, Bout;
37   wire p, q, r;
38   half_subtractor u4(A, B, p, q);
39   half_subtractor u5(p, Bin, D, r);
40   or_gate u6(q, r, Bout);
41 endmodule

```



Question 3: Write the Verilog code (either structural or behavioural) for a 4-bit ripple carry adder. The block diagram of a 4-bit ripple carry adder has been given in Figure 1 for your reference.

```
design.sv 
```

```

1 module FA (a,b,c,s,cr);
2   input a,b,c;
3   output s,cr;
4   wire s1,c1,c2;
5
6   xor(s1, a, b);
7   xor(s, s1, c);
8   and(c1, a, b);
9   and(c2, s1, c);
10  or(cr, c1, c2);
11 endmodule
12
13 module RCA(x,y,z,sum,cout);
14   input [3:0]x,y;
15   input z;
16   output [3:0]sum;
17   output cout;
18   wire w1,w2,w3;
19
20  FA f0(x[0], y[0], z, sum[0], w1);
21  FA f1(x[1], y[1], w1, sum[1], w2);
22  FA f2(x[2], y[2], w2, sum[2], w3);
23  FA f3(x[3], y[3], w3, sum[3], cout);
24 endmodule

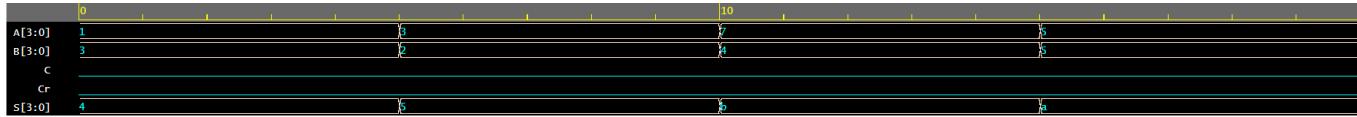
```

testbench.sv

```

1 module tb_RCA;
2   reg [3:0]A,B;
3   reg C;
4   wire [3:0]S;
5   wire Cr;
6
7   RCA rca1(.x(A), .y(B), .z(C), .sum(S), .cout(Cr));
8   initial
9   begin
10     A = 1; B = 3; C = 0; #5;
11     A = 3; B = 2; C = 0; #5;
12     A = 7; B = 4; C = 0; #5;
13     A = 5; B = 5; C = 0; #5;
14   end
15   initial begin
16     $dumpfile("dump.vcd");
17     $dumpvars(1);
18   end
19 endmodule

```



Question 4: Write the Verilog code to construct a 4-bit adder-subtractor using. The logic diagram of a 4-bit adder-subtractor using a ripple carry adder has been given in Figure 2 for your reference.

design.sv

```

1 module FA (a,b,c,s,cr);
2   input a,b,c;
3   output s,cr;
4   wire s1,c1,c2;
5
6   xor(s1, a, b);
7   xor(s, s1, c);
8   and(c1, a, b);
9   and(c2, s1, c);
10  or(cr, c1, c2);
11 endmodule
12
13 module FBAS(x,y,c0,m,sum,cout);
14   input [3:0]x,y;
15   input c0,m;
16   output [3:0]sum;
17   output cout;
18
19   FA f0(x[0], y[0]^m, c0, sum[0], c1);
20   FA f1(x[1], y[1]^m, c1, sum[1], c2);
21   FA f2(x[2], y[2]^m, c2, sum[2], c3);
22   FA f3(x[3], y[3]^m, c3, sum[3], cout);
23 endmodule

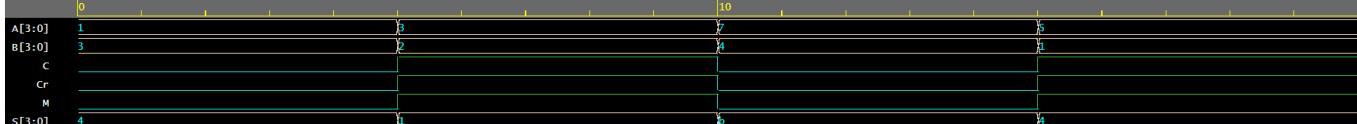
```

testbench.sv

```

1 module tb_FBAS;
2   reg [3:0]A,B;
3   reg C,M;
4   wire [3:0]S;
5   wire Cr;
6
7   FBAS fbas1(.x(A), .y(B), .c0(C), .m(M), .sum(S), .cout(Cr));
8   initial
9   begin
10     A = 1; B = 3; C = 0; M = 0; #5; // Adder
11     A = 3; B = 2; C = 1; M = 1; #5; // Subtractor
12     A = 7; B = 4; C = 0; M = 0; #5; // Adder
13     A = 5; B = 1; C = 1; M = 1; #5; // Subtractor
14   end
15   initial begin
16     $dumpfile("dump.vcd");
17     $dumpvars(1);
18   end
19 endmodule

```



Question 5: Write the Verilog code for a 4-bit carry look-ahead adder (CLA). The block diagram of the same has been given in Figure 3.

design.sv

```

1 module CLA_Adder(a,b,cin,sum,cout);
2   input [3:0] a,b;
3   input cin;
4   output [3:0] sum;
5   output cout;
6   wire p0,p1,p2,p3,g0,g1,g2,g3,c1,c2,c3,c4;
7
8   assign p0=(a[0]^b[0]),
9     p1=(a[1]^b[1]),
10    p2=(a[2]^b[2]),
11    p3=(a[3]^b[3]);
12
13  assign g0=(a[0]&b[0]),
14    g1=(a[1] & b[1]),
15    g2=(a[2] & b[2]),
16    g3=(a[3] & b[3]);
17
18  assign c0=cin,
19    c1=g0|(p0 & cin),
20    c2=g1|(p1 & g0)|(p1 & p0 & cin),
21    c3=g2|(p2 & g1)|(p2 & p1 & g0)|(p1 & p1 & p0 & cin),
22    c4=g3|(p3 & g2)|(p3 & p2 & g1)|(p3 & p2 & p1 & g0)|(p3 & p2 & p1 & p0 & cin);
23
24  assign sum[0]=p0^c0,
25    sum[1]=p1^c1,
26    sum[2]=p2^c2,
27    sum[3]=p3^c3;
28  assign cout=c4;
29 endmodule

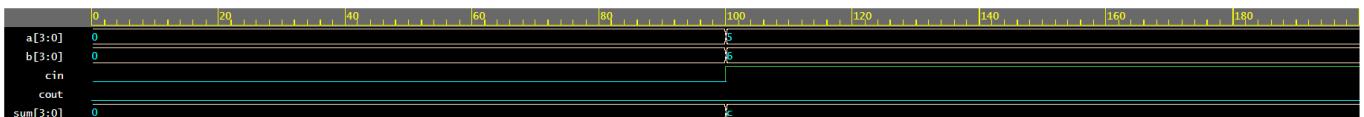
```

testbench.sv

```

1 module TestModule;
2   // Inputs
3   reg [3:0] a;
4   reg [3:0] b;
5   reg cin;
6
7   // Outputs
8   wire [3:0] sum;
9   wire cout;
10
11  // Instantiate the Unit Under Test
12  CLA_Adder uut (
13    .a(a),
14    .b(b),
15    .cin(cin),
16    .sum(sum),
17    .cout(cout)
18  );
19  initial begin
20    // Initialize Inputs
21    a = 0;
22    b = 0;
23    cin = 0;
24    // Wait 100 ns for global reset to finish
25    #100;
26    a = 5;
27    b = 6;
28    cin = 1;
29    // Wait 100 ns for global reset to finish
30    #100;
31  end
32  initial begin
33    $dumpfile("dump.vcd");
34    $dumpvars(1);
35  end
36 endmodule

```



LAB 6: Designing of Multiplexer, Demultiplexer, Decoder, Encoder

Question 1: Write a Verilog code to design a 4:1 Multiplexer and verify the same.

design.sv +

```

1 module mux_4to1_case ( input [3:0] a,
2                         input [3:0] b,
3                         input [3:0] c,
4                         input [3:0] d,
5                         input [1:0] sel,
6                         output reg [3:0] out);
7   always @ (a or b or c or d or sel) begin
8     case (sel)
9       2'b00 : out <= a;
10      2'b01 : out <= b;
11      2'b10 : out <= c;
12      2'b11 : out <= d;
13    endcase
14  end
15 endmodule

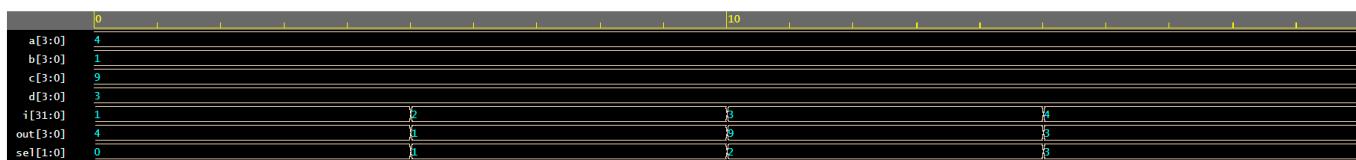
```

testbench.sv +

```

1 module tb_4to1_mux;
2
3   reg [3:0] a;
4   reg [3:0] b;
5   reg [3:0] c;
6   reg [3:0] d;
7   wire [3:0] out;
8   reg [1:0] sel;
9   integer i;
10
11   mux_4to1_case mux0 (.a (a),
12                       .b (b),
13                       .c (c),
14                       .d (d),
15                       .sel (sel),
16                       .out (out));
17
18   initial
19   begin
20     $monitor ("[%0t] sel=%0h a=%0h b=%0h c=%0h d=%0h
21     out=%0h", $time, sel, a, b, c, d, out);
22     sel <= 0;
23     a <= $random;
24     b <= $random;
25     c <= $random;
26     d <= $random;
27     $dumpfile("dump.vcd");
28     $dumpvars(1);
29     for (i = 1; i < 4; i=i+1) begin
30       #5 sel <= i;
31     end
32     #5 $finish;
33   end
34 endmodule

```



Question 2: Write a Verilog code to design an 8:1 Multiplexer and verify the same.

design.sv

```

1 module Multiplexer(d0,d1,d2,d3,d4,d5,d6,d7,sel,out);
2   input d0,d1,d2,d3,d4,d5,d6,d7;
3   input [2:0] sel;
4   output reg out;
5   always@(sel)
6     begin
7       case(sel)
8         3'b000:out=d0;
9         3'b001:out=d1;
10        3'b010:out=d2;
11        3'b011:out=d3;
12        3'b100:out=d4;
13        3'b101:out=d5;
14        3'b110:out=d6;
15        3'b111:out=d7;
16      endcase
17    end
18 endmodule

```

testbench.sv

```

1 module TestModule;
2
3 reg d0, d1, d2, d3, d4, d5, d6, d7;
4 reg [2:0] sel;
5 wire out;
6
7 Multiplexer uut(.d0(d0),.d1(d1),.d2(d2),.d3(d3),.d4(d4),.d5(d5),.d6(d6),.d7(d7),.sel(sel),.out(out));
8
9 initial
10 begin
11   d0 = 0;
12   d1 = 0;
13   d2 = 0;
14   d3 = 0;
15
16   d4 = 0;
17   d5 = 0;
18   d6 = 0;
19   d7 = 0;
20   sel = 0;
21   #100;
22   d0 = 0;
23   d1 = 0;
24   d2 = 0;
25   d3 = 0;
26   d4 = 1;
27   d5 = 1;
28   d6 = 0;
29   d7 = 1;
30   sel = 5;
31   #100;
32 end
33 initial
34 begin
35   $dumpfile("dump.vcd");
36   $dumpvars(1);
37 end
38 endmodule

```



Question 3: Write a Verilog code to design a 1:4 Demultiplexer and verify the same.

design.sv



```

1 module Demultiplexer1to4case (output reg [3:0] Y,
2                                 input [1:0] A,
3                                 input din);
4   always @(Y, A) begin
5     case (A)
6       2'b00 : begin Y[0] = din; Y[3:1] = 0; end
7       2'b01 : begin Y[1] = din; Y[0] = 0; end
8       2'b10 : begin Y[2] = din; Y[1:0] = 0; end
9       2'b11 : begin Y[3] = din; Y[2:0] = 0; end
10    endcase
11  end
12 endmodule

```

testbench.sv



```

1 module Demultiplexer1to4case_tb;
2 wire [3:0] Y;
3 reg [1:0] A;
4 reg din;
5 Demultiplexer1to4case I0 (Y, A, din);
6 initial begin
7   din = 1;
8   A = 2'b00;
9 #1 A = 2'b01;
10 #1 A = 2'b10;
11 #1 A = 2'b11;
12 end
13 initial begin
14   $monitor("%t| Din = %d| A[1] = %d| A[0] = %d| Y[0] = %d| Y[1] = %d| Y[2] = %d| Y[3] = %d",
15           $time, din, A[1], A[0], Y[0], Y[1], Y[2], Y[3]);
16   $dumpfile("dump.vcd");
17   $dumpvars(1);
18 end
19 endmodule

```



Question 4: Write a Verilog code to design a 1:8 Demultiplexer and verify the same.

design.sv



```

1 module Demultiplexer(in,s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7);
2   input in,s0,s1,s2;
3   output d0,d1,d2,d3,d4,d5,d6,d7;
4   assign d0=(in & ~s2 & ~s1 &~s0),
5         d1=(in & ~s2 & ~s1 &s0),
6         d2=(in & ~s2 & s1 &~s0),
7         d3=(in & ~s2 & s1 &s0),
8         d4=(in & s2 & ~s1 &~s0),
9         d5=(in & s2 & ~s1 &s0),
10        d6=(in & s2 & s1 &~s0),
11        d7=(in & s2 & s1 &s0);
12 endmodule

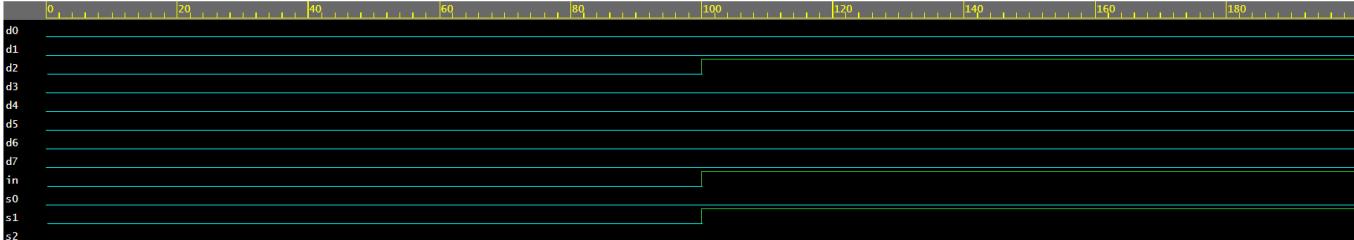
```

testbench.sv +

```

1 module TestModule;
2   reg in;
3   reg s0;
4   reg s1;
5   reg s2;
6
7   wire d0;
8   wire d1;
9   wire d2;
10  wire d3;
11  wire d4;
12  wire d5;
13  wire d6;
14  wire d7;
15
16  Demultiplexer d(.in(in), .s0(s0), .s1(s1), .s2(s2), .d0(d0), .d1(d1),
17 .d2(d2), .d3(d3), .d4(d4), .d5(d5), .d6(d6), .d7(d7));
18 initial
19 begin
20   in = 0;
21   s0 = 0;
22   s1 = 0;
23   s2 = 0;
24   #100;
25   in = 1;
26   s0 = 0;
27   s1 = 1;
28   s2 = 0;
29   #100;
30 end
31 initial
32 begin
33   $dumpfile("dump.vcd");
34   $dumpvars(1);
35 end
endmodule

```



Question 5: Write a Verilog code to design a 3:8 Decoder and verify the same.

design.sv +

```

1 module encoder(Do, Din);
2   input [7:0]Din;
3   output [2:0]Do;
4
5   or(Do[0], Din[4], Din[5], Din[6], Din[7]);
6   or(Do[1], Din[2], Din[3], Din[6], Din[7]);
7   or(Do[2], Din[1], Din[3], Din[5], Din[7]);
8 endmodule
9

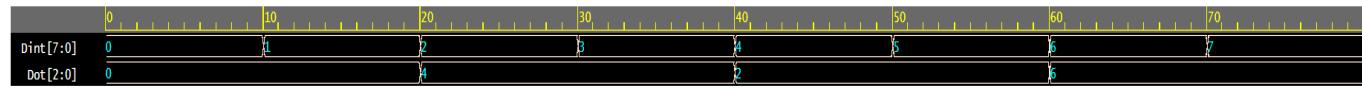
```

testbench.sv

```

1 module encoder_tb_v;
2   reg [7:0] Dint;
3   wire [2:0] Dot;
4
5   encoder uut(.Do(Dot),.Din(Dint));
6   initial
7     begin
8       Dint = 8'b00000000; #10;
9       Dint = 8'b00000001; #10;
10      Dint = 8'b00000010; #10;
11      Dint = 8'b00000011; #10;
12      Dint = 8'b00000100; #10;
13      Dint = 8'b00000101; #10;
14      Dint = 8'b00000110; #10;
15      Dint = 8'b00000111; #10;
16    end
17   initial
18   begin
19     $dumpfile("dump.vcd");
20     $dumpvars(1);
21   end
22 endmodule

```



Question 6: Write a Verilog code to design a 8:3 Encoder and verify the same.

design.sv

```

1 module encoder83(din, do0, do1, do2);
2   input [7:0]din;
3   output do0, do1, do2;
4   or(do0[0], din[4], din[5], din[6], din[7]);
5   or(do1[1], din[2], din[3], din[6], din[7]);
6   or(do2[2], din[1], din[3], din[5], din[7]);
7 endmodule

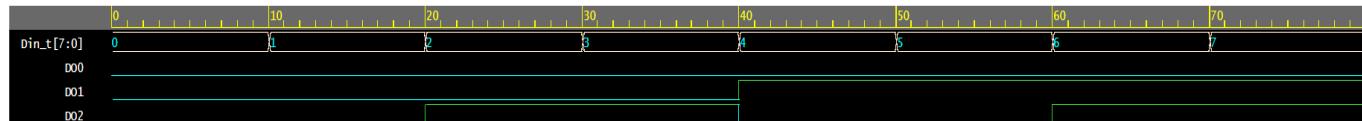
```

testbench.sv

```

1 module encoder83_tb;                                     SV/Verilog Test Bench
2   reg [7:0] Din_t;
3   wire DO0, DO1, DO2;
4   encoder83 e1.(din(Din_t), .do0(DO0), .do1(DO1), .do2(DO2));
5   initial
6     begin
7       Din_t = 8'b00000000; #10;
8       Din_t = 8'b00000001; #10;
9       Din_t = 8'b00000010; #10;
10      Din_t = 8'b00000011; #10;
11      Din_t = 8'b00000100; #10;
12      Din_t = 8'b00000101; #10;
13      Din_t = 8'b00000110; #10;
14      Din_t = 8'b00000111; #10;
15    end
16   initial
17   begin
18     $dumpfile("dump.vcd");
19     $dumpvars(1);
20   end
21 endmodule

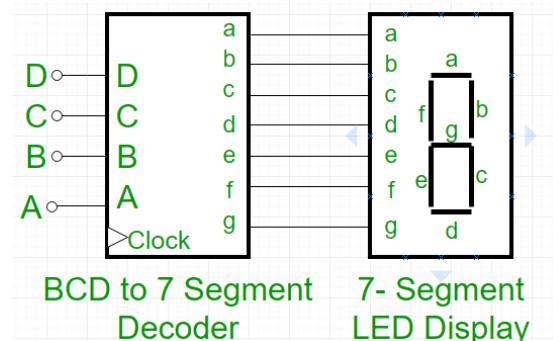
```



LAB 7: Application of Decoder, Recap of Logisim

Question 1: Write a Verilog code to implement BCD to seven segment display Decoder. Prepare the Truth Table and circuits and verify the same using Test Bench.

| Decimal Number | INPUT Lines | | | | OUTPUT Lines | | | | | | |
|----------------|-------------|---|---|---|--------------|---|---|---|---|---|---|
| | A | B | C | D | a | b | c | d | e | f | g |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |



```
testbench.sv +
```

```

1 module tb_segment7;
2
3   reg [3:0] bcd;
4   wire [6:0] seg;
5   integer i;
6
7   segment7 uut (
8     .bcd(bcd),
9     .seg(seg)
10 );
11
12 initial begin
13   for(i = 0;i < 16;i = i+1)
14   begin
15     bcd = i;
16     #10;
17   end
18 end
19 initial begin
20   $dumpfile("dump.vcd");
21   $dumpvars(1);
22 end
23
24 endmodule

```

```
design.sv +
```

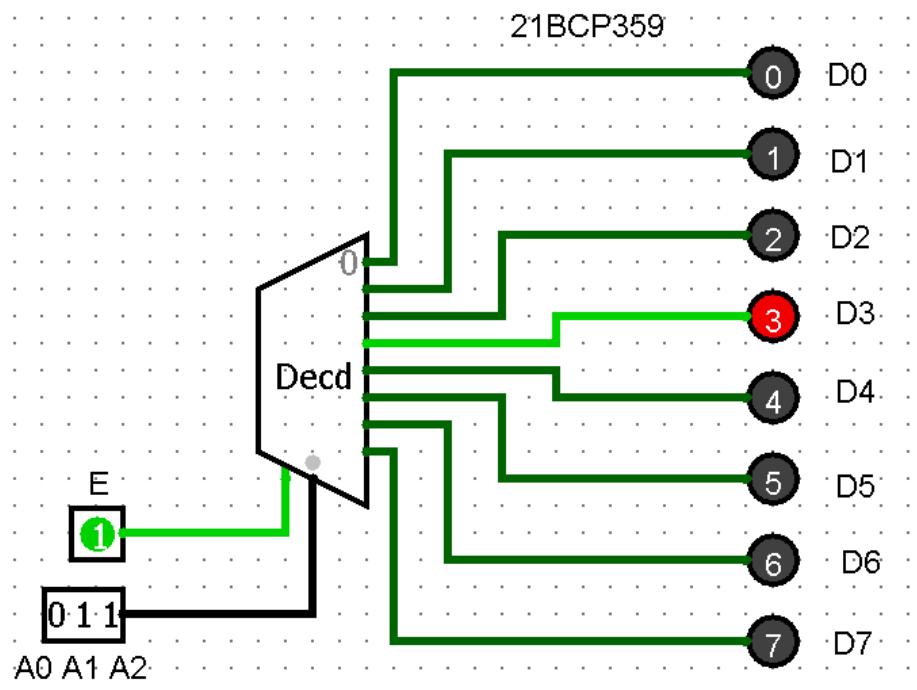
```

1 module segment7(
2   bcd,
3   seg
4 );
5
6   input [3:0] bcd;
7   output [6:0] seg;
8   reg [6:0] seg;
9
10 always @(bcd)
11 begin
12   case (bcd)
13     0 : seg = 7'b0000001;
14     1 : seg = 7'b1001111;
15     2 : seg = 7'b0010010;
16     3 : seg = 7'b0000110;
17     4 : seg = 7'b1001100;
18     5 : seg = 7'b0100100;
19     6 : seg = 7'b0100000;
20     7 : seg = 7'b0000111;
21     8 : seg = 7'b0000000;
22     9 : seg = 7'b0000100;
23   endcase
24 end
25
26 endmodule

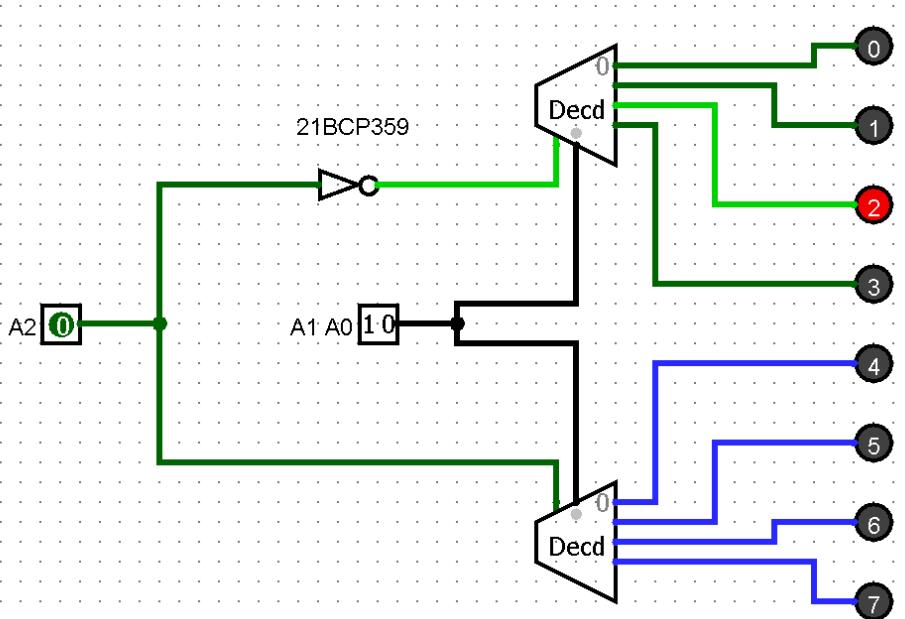
```

| | | | | | | | | | | | | | | | | |
|----------|---|----|----|---|----|----|----|---|---|---|---|---|---|---|---|---|
| bcd[3:0] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| i[3:0] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
| seg[6:0] | 1 | 4f | 12 | 6 | 4c | 24 | 20 | f | 0 | 4 | | | | | | |

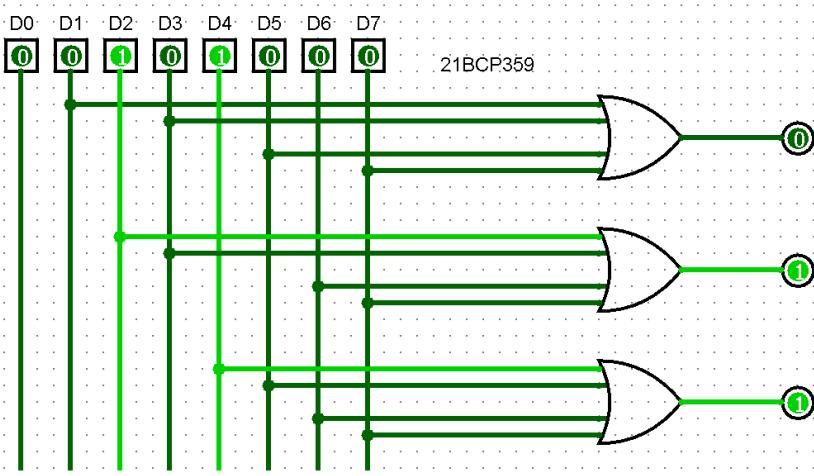
Question 2: Design a 3:8 Decoder using Logisim.



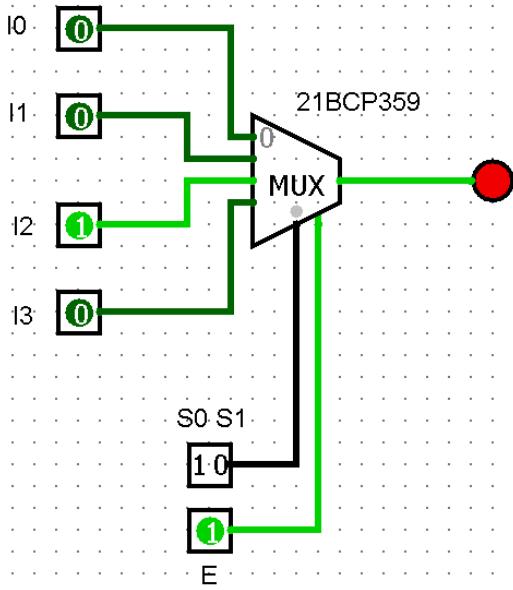
Question 3. Design a 3:8 Decoder using two 2:4 Decoder using Logisim.



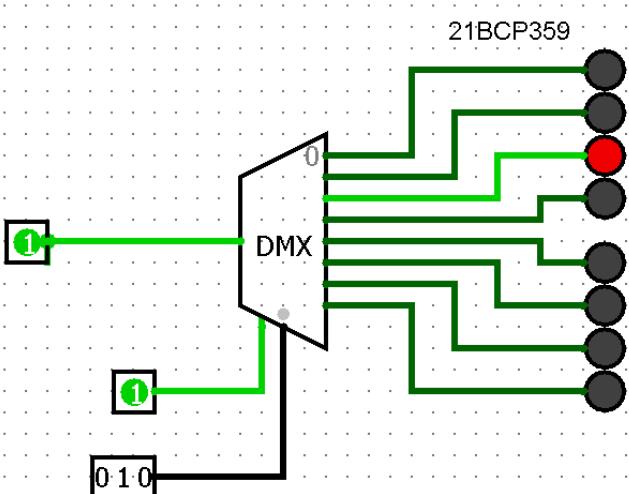
Question 4. Design an 8:3 Priority Encoder using Logisim.



Question 5. Design a 4:1 Multiplexer using Logisim.

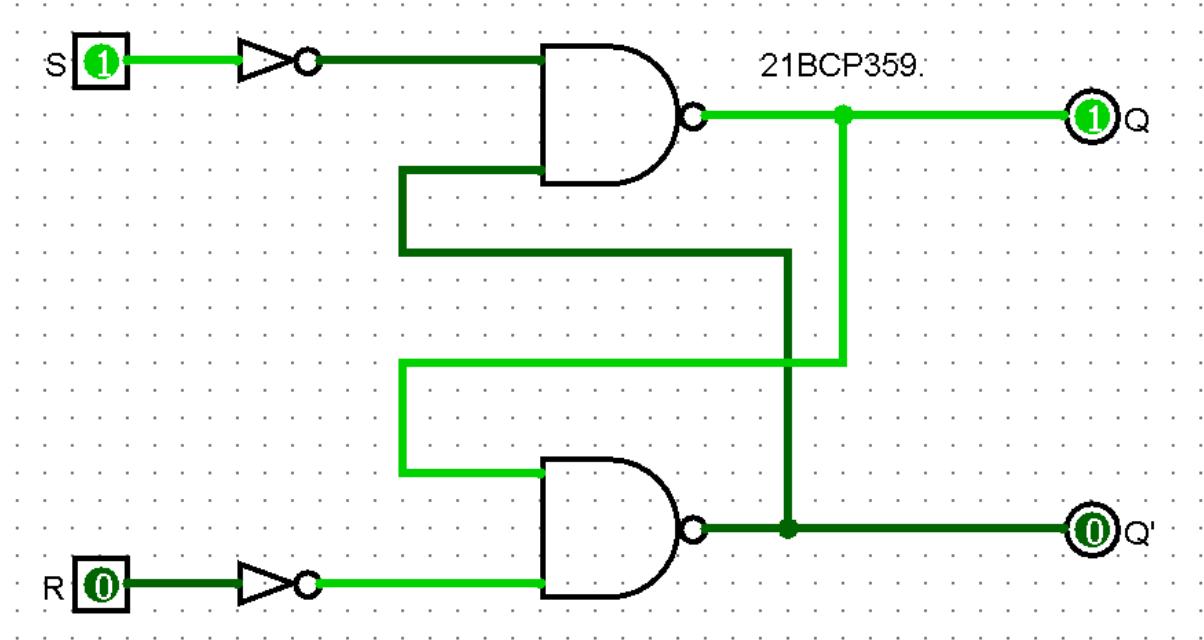


Question 6. Design a 1:8 Demultiplexer using Logisim



LAB 8: Sequential Circuits: Latch And SR Flip Flop

Question 1: Design an SR flip-flop using Logisim.

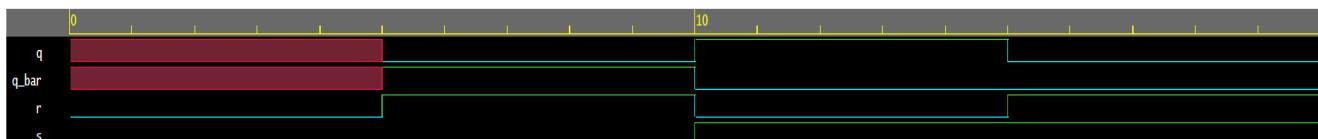


Question 2: Write a module for NOR gate and develop a structural Verilog code for the S-R latch using the NOR gate module. Validate the code via a suitable Testbench code.

Expression: $Q_{n+1} = E \cdot D + E' \cdot Q_n$

```
testbench.sv +  
1 module nor_latch_tb;  
2   reg s,r; //input  
3   wire q,q_bar; //output  
4   nor_latch x(.s(s),.r(r),.q(q),.q_bar(q_bar));  
5   initial  
6     begin  
7       s<=1'b0; r<=1'b0; #5;  
8       s<=1'b0; r<=1'b1; #5;  
9       s<=1'b1; r<=1'b0; #5;  
10      s<=1'b1; r<=1'b1; #5;  
11    end  
12  initial  
13    begin  
14      $dumpfile("dump.vcd");  
15      $dumpvars(1);  
16    end  
17 endmodule
```

```
design.sv +  
1 // SR LATCH  
2 module nor_latch(input s,r, output q,q_bar);  
3   nor(q,r,q_bar);  
4   nor(q_bar,q,s);  
5 endmodule
```



Question 3: Write a Verilog module for NAND gate and utilize it to develop a structural Verilog module. Validate it using suitable Test bench.

Expression: $Q_{n+1} = S + Q_n R'$

```

testbench.sv + design.sv +

```

```

1 module SRflipflop;
2 reg S, R, CLK;
3 wire Q, QBAR;
4 SRflipflop f1 (.s(S), .r(R), .clk(CLK), .q(Q), .qbar(QBAR));
5 initial
6 begin
7   S = 1; R = 0; #10;
8   S = 0; R = 1; #10;
9   S = 0; R = 0; #10;
10  S = 1; R = 1; #10;
11 end
12 initial
13 begin
14   $dumpfile("dump.vcd");
15   $dumpvars(1);
16 end
17 endmodule
18
19

```

```

1 module SRflipflop(q, qbar, s, r, clk);
2
3 input s, r, clk;
4 output q, qbar;
5
6 assign q = clk ? (s + ((~r) & q)) : q;
7 assign qbar = ~q;
8 endmodule
9

```

Q 4. Develop a similar behavioral code and test bench for S-R flip flop using if-else condition as per question 3.

```

testbench.sv + design.sv +

```

```

1 // 21BCP294 -Ayush Thakor
2 // SR -Flip Flop behavioral code
3 module tb_srff_ifelse;
4 reg S,R, CLK;
5 wire Q, QBAR;
6 srff_ifelse dut(S,R,CLK,Q,QBAR);
7 initial
8 begin
9 S= 0; R= 0; CLK=0; #1;
10 $display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
11 S= 0; R= 1; CLK=1; #1;
12 $display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
13 S= 0; R= 0; CLK=0; #1;
14 $display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
15 S= 1; R= 0; CLK=1; #1;
16 $display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
17 S= 0; R= 0; CLK=0; #1;
18 $display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
19 S= 1; R= 1; CLK=1; #1;
20 $display ("CLK = %b, S= %b, R = %b , Q = %b , QBAR = %b", CLK,S,R, Q, QBAR);
21 $finish;
22 end
23 initial
24 begin
25 $dumpfile("dump.vcd");
26 $dumpvars (1);
27 end

```

```

1 // 21BCP294 -Ayush Thakor
2 module srff_ifelse(S, R, CLK, Q, QBAR);
3 input S, R, CLK;
4 output reg Q, QBAR;
5 always@ (CLK)
6 begin
7 if(S==1 & R==0)
8 begin
9 Q = 1;
10 QBAR = 0;
11 end
12 else if(S==0 & R==1)
13 begin
14 Q = 0;
15 QBAR = 1;
16 end
17 else if(S==0 & R==0)
18 begin
19 Q = Q;
20 QBAR = QBAR;
21 end
22 else if(S==1 & R==1)
23 begin
24 Q = 1;
25 QBAR = 1;
26 end
27 end
28 endmodule
29
30

```



$CLK = 0, S = 0, R = 0, Q = x, QBAR = x$
 $CLK = 1, S = 0, R = 1, Q = 0, QBAR = 1$
 $CLK = 0, S = 0, R = 0, Q = 0, QBAR = 1$
 $CLK = 1, S = 1, R = 0, Q = 1, QBAR = 0$
 $CLK = 0, S = 0, R = 0, Q = 1, QBAR = 0$
 $CLK = 1, S = 1, R = 1, Q = 1, QBAR = 1$

Question 5: Develop a similar behavioural code and test bench for S-R flip flop using if-else condition as per question 3.

```

testbench.sv + design.sv + [+] + [x]

1 module SRflipfloptest;
2   reg S, R, CLK;
3   wire Q, QBAR;
4   SRflipflop f1 (.s(S), .r(R), .clk(CLK), .q(Q), .qbar(QBAR));
5   initial
6     begin
7       S = 1; R = 0; #10;
8       S = 0; R = 1; #10;
9       S = 0; R = 0; #10;
10      S = 1; R = 1; #10;
11    end
12  initial
13    begin
14      $dumpfile("dump.vcd");
15      $dumpvars(1);
16    end
17 endmodule
18
19
1 module SRflipflop(s, r, clk, q, qbar);
2   input clk, s, r;
3   output reg q, qbar;
4   always@(posedge clk)
5     begin
6       if(s == 1)
7         begin
8           q <= 1;
9           qbar <= 0;
10        end
11       else if(r == 1)
12         begin
13           q <= 0;
14           qbar <= 1;
15         end
16       else if(s == 0 & r == 0)
17         begin
18           q <= q;
19           qbar = qbar;
20         end
21     end
22 endmodule

```

CLK Q QBAR R S

LAB 9: Flip Flops (JK, D, T)

Question 1: Write a Verilog code to implement J-K flip flop and validate the code via a suitable Test bench code.

```
design.sv +
```

```

1 module jk_ff ( input j, input k, input clk, output q);
2   reg q;
3   always @ (posedge clk)
4     case ({j,k})
5       2'b00 : q <= q;
6       2'b01 : q <= 0;
7       2'b10 : q <= 1;
8       2'b11 : q <= ~q;
9     endcase
10 endmodule

```

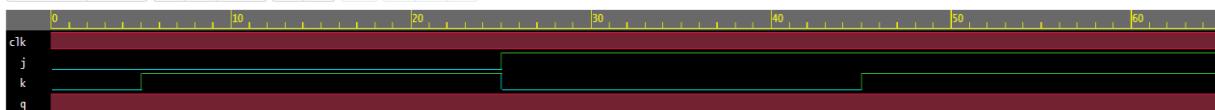


```
testbench.sv +
```

```

1 module tb_jk;
2   reg j;
3   reg k;
4   reg clk;
5   always #5 clk = ~clk;
6   jk_ff jk0 (.j(j), .k(k), .clk(clk), .q(q));
7
8   initial
9     begin
10       j <= 0;
11       k <= 0; #5
12
13       j <= 0;
14       k <= 1; #20
15
16       j <= 1;
17       k <= 0; #20
18
19       j <= 1;
20       k <= 1; #20
21       $finish;
22   end
23   initial
24     begin
25       $dumpfile("dump.vcd");
26       $dumpvars(1);
27     end
28
29   initial
30     $monitor ("j=%0d k=%0d q=%0d", j, k, q);
31 endmodule

```



```
[2022-11-14 09:32:17 EST] iverilog '-Wall' design.sv testbench.sv && unbuffer vvp a.out
```

```
testbench.sv:6: warning: implicit definition of wire logic tb_jk.q.
```

```
VCD info: dumpfile dump.vcd opened for output.
```

```
j=0 k=0 q=x
```

```
j=0 k=1 q=x
```

```
j=1 k=0 q=x
```

```
j=1 k=1 q=x
```

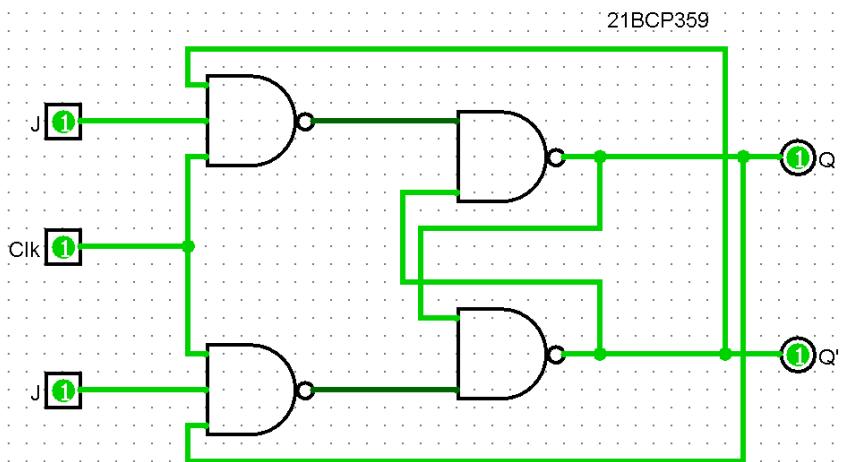
```
Finding VCD file...
```

```
./dump.vcd
```

```
[2022-11-14 09:32:17 EST] Opening EPWave...
```

```
Done
```

Question 2: Design a J-K flip flop in Logisim and validate the circuit.



| Clock | J | K | Q_{n+1} | State |
|-------|---|---|-------------|--------|
| 0 | x | x | Q_n | |
| 1 | 0 | 0 | Q_n | Hold |
| 1 | 0 | 1 | 0 | Reset |
| 1 | 1 | 1 | 1 | Set |
| 1 | 1 | 1 | \bar{Q}_n | Toggle |

Question 3: Write a Verilog code to implement D flip flop and validate the code via a suitable Test bench code.

```
design.sv +
```

```

1 // Design
2 // D flip-flop
3 module dff (clk, reset,
4   d, q, qb);
5   input  clk;
6   input  reset;
7   input  d;
8   output q;
9   output qb;
10
11  reg   q;
12
13  assign qb = ~q;
14
15  always @(posedge clk or posedge reset)
16  begin
17    if (reset) begin
18      // Asynchronous reset when reset goes high
19      q <= 1'b0;
20    end else begin
21      // Assign D to Q on positive clock edge
22      q <= d;
23    end
24  end
25 endmodule

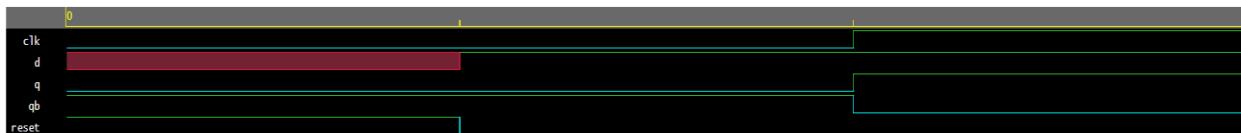
```

```
testbench.sv +
```

```

1 // Testbench
2 module test;
3
4 reg clk;
5 reg reset;
6 reg d;
7 wire q;
8 wire qb;
9
10 // Instantiate design under test
11 dff DFF(.clk(clk), .reset(reset),
12           .d(d), .q(q), .qb(qb));
13
14 initial begin
15   // Dump waves
16   $dumpfile("dump.vcd");
17   $dumpvars(1);
18
19   $display("Reset flop.");
20   clk = 0;
21   reset = 1;
22   d = 1'bx;
23   display;
24
25   $display("Release reset.");
26   d = 1;
27   reset = 0;
28   display;
29
30   $display("Toggle clk.");
31   clk = 1;
32   display;
33 end
34
35 task display;
36   #1 $display("d:%0h, q:%0h, qb:%0h",
37             d, q, qb);
38 endtask
39
40 endmodule

```



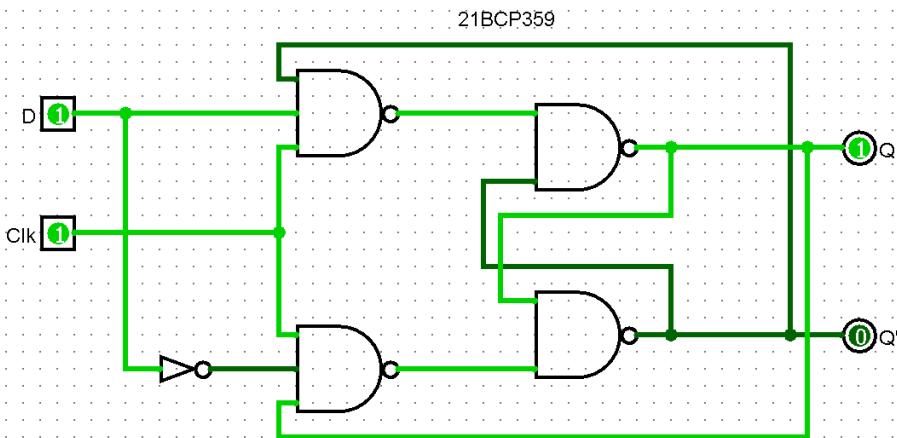
```
[2022-11-14 09:37:27 EST] iverilog '-Wall' design.sv testbench.sv && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
```

Reset flop.
d:x, q:0, qb:1
Release reset.
d:1, q:0, qb:1
Toggle clk.
d:1, q:1, qb:0
Finding VCD file...
./dump.vcd

```
[2022-11-14 09:37:28 EST] Opening EPWave...
```

```
Done
```

Question 4: Design a D flip flop in Logisim and validate the circuit.



| S | R | Q | Q' |
|---|---|----------|----------|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | ∞ | ∞ |

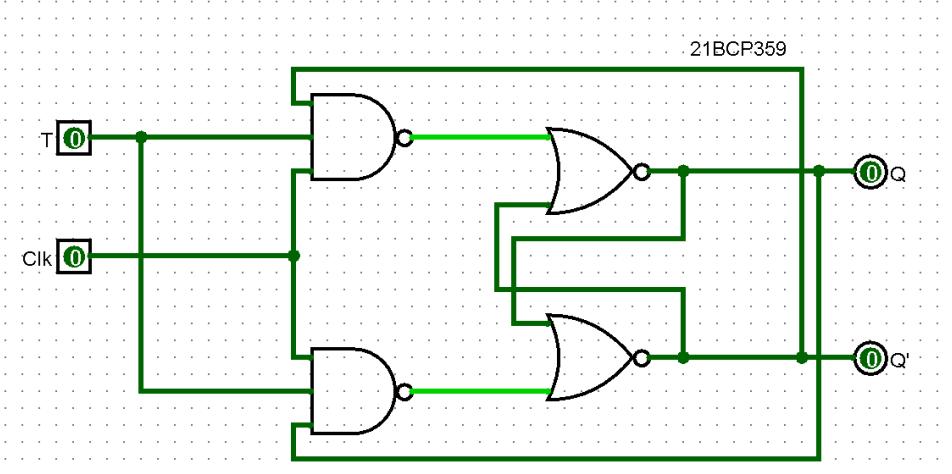
Question 5: Write a Verilog code to implement T flip flop and validate the code via a suitable Test bench code.

```
testbench.sv + 
1 module tb;
2   reg clk;
3   reg rstn;
4   reg t;
5
6   tff u0 (.clk(clk),
7             .rstn(rstn),
8             .t(t),
9             .q(q));
10
11  always #5 clk = ~clk;
12
13 initial
14 begin
15   {rstn, clk, t} <= 0;
16
17   $monitor ("T=%0t rstn=%0b t=%0d q=%0d", $time, rstn, t, q);
18   repeat(2) @ (posedge clk);
19   rstn <= 1;
20 end
21 initial
22 begin
23   $dumpfile("dump.vcd");
24   $dumpvars(1);
25 end
26 endmodule
```

```
design.sv + 
1 module tff ( input clk, input rstn, input t, output reg q);
2
3   always @ (posedge clk) begin
4     if (!rstn)
5       q <= 0;
6     else
7       if (t)
8         q <= ~q;
9       else
10        q <= q;
11   end
12 endmodule
```

```
[2022-11-15 00:51:05 EST] iverilog '-Wall' design.sv testbench.sv && unbuffer vvp a.out
testbench.sv:9: warning: implicit definition of wire logic tb.q.
VCD info: dumpfile dump.vcd opened for output.
T=0 rstn=0 t=0 q=x
T=5 rstn=0 t=0 q=0
T=15 rstn=1 t=0 q=0
```

Question 6: Design a T flip flop in Logisim and validate the circuit.



| Truth table | | | |
|--------------|---|------------|------------|
| CLK | T | Q_{next} | Comment |
| Rising edge | 0 | Q | Hold state |
| Falling edge | 0 | Q | Hold state |
| Rising edge | 1 | \bar{Q} | Toggle |
| Falling edge | 1 | Q | No change |

Q_{next} - "after the clock transition" output

Q - the current output

Lab 10: Flip Flops Conversion

Question 1: Modify the S-R flip flop created in last lab into JK flip flop. Verify the functionality of J-K flip flop using suitable Testbench.

Conversion Table

| J-K Inputs | | Outputs | | S-R Inputs | |
|------------|---|----------------|------------------|------------|---|
| J | K | Q _p | Q _{p+1} | S | R |
| 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 1 | 1 | X | 0 |
| 0 | 1 | 0 | 0 | 0 | X |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | X | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |

Conversion Expression: S = J'Q and R = KQ

testbench.sv
design.sv

```

testbench.sv
1 module FLOP();
2   reg clk,J,K,rst;
3   wire Q;
4
5   flop f_1(.clk(clk),.J(J),.K(K),.rst(rst),.Q(Q));
6
7 initial begin
8   clk=0;
9   J=0;
10  K=0;
11  rst=1;
12 end
13
14 initial forever #10 clk=~clk;
15 initial forever #20 J=~J;
16 initial forever #40 K=~K;
17 initial forever #160 rst=~rst;
18
19 initial begin
20   #800 $finish;
21 end
22
23 initial begin
24   $dumpfile("flop.vcd");
25   $dumpvars;
26 end
27 endmodule

```

```

design.sv
1 module flop(input clk,J,K,rst, output Q);
2   reg ff1,ff2;
3
4   always @(rst) begin
5     ff1 <= 1'b0;
6     ff2 <= 1'b0;
7   end
8
9   always @ (posedge clk) begin
10    if (!rst) begin
11      if (J==1 && K==0) begin
12        ff1 <= 1'b1;
13      end
14      else if (J==0 && K==1) begin
15        ff1 <= 1'b0;
16      end
17      else if (J==1 && K==1) begin
18        ff1 <= ~ff1;
19      end
20      else begin
21        ff1 <= ff1;
22      end
23    end
24    assign Q=ff1;
25  endmodule

```

Question 2: Develop the behavioural Verilog module of D flip-flop, converting it from a J-K flip flop. You may utilize if-else statements to develop your Verilog code. Verify the functionality via a suitable test bench code.

Conversion Table

| D Input | Outputs | | J-K Inputs | |
|---------|----------------|------------------|------------|---|
| | Q _p | Q _{p+1} | J | K |
| 0 | 0 | 0 | 0 | X |
| 0 | 1 | 0 | X | 1 |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | X | 0 |

Conversion Expression: $J = D$ and $K = D'$

testbench.sv

```

1 module tb_jkff
2
3 reg clk=0;
4 rreg d=0;
5 reg reset=1;
6 wire q, qnot;
7
8 jkff dut(reset, clk, d, q, qnot);
9
10 initial
11 begin
12   $dumpfile("dump.vcd");
13   $dumpvars(1);
14   d = 1'b0;
15   reset 1'b0;
16   #5
17   #25
18   $finish;
19 end
20 always #1 clk = ~clk;
21 endmodule

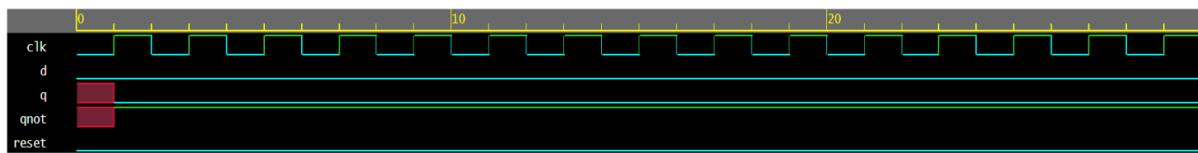
```

design.sv

```

1 module jkff (input reset, input clk, input
d, output reg q, output qnot);
2
3 wire j,k;
4 assign qnot = ~q;
5 assign j=d;
6 assign k = ~d;
7 always @(posedge clk)
8 if(reset) q<=1'b0; else
9 case({d})
10   2'b0 : q<=1'b0;
11   2'b1 : q<=1'b1;
12 endcase
endmodule

```



Question 3: Develop a behavioural Verilog module for JK flip flop using case statements. Modify it to act like a T flip flop. Validate the modification via a suitable test bench.

Conversion Expression: $J = T$ and $K = T'$

testbench.sv

```

1 module TFLOP();
2   reg clk,T,rst;
3   wire Q;
4
5   TFF tff_1(.clk(clk),.T(T),.rst(rst),.Q(Q));
6
7   initial begin
8     clk=0;
9     T=0;
10    rst=1;
11  end
12
13  initial forever #10 clk=~clk;
14  initial forever #20 T=~T;
15  initial forever #160 rst=~rst;
16
17  initial begin
18    #800 $finish;
19  end
20
21  initial begin
22    $dumpfile("TFLOP.vcd");
23    $dumpvars;
24  end
25 endmodule

```

design.sv

```

1 module flop(input clk,J,K,rst, output Q);
2   reg ff1,ff2;
3
4   always @(rst) begin
5     ff1 <= 1'b0;
6     ff2 <= 1'b0;
7   end
8
9   always @(posedge clk) begin
10    if (rst) begin
11      if (J==1 && K==0) begin
12        ff1 <= 1'b1;
13      end
14      else if (J==0 && K==1) begin
15        ff1 <= 1'b0;
16      end
17      else if (J==1 && K==1) begin
18        ff1 <= ff1;
19      end
20      else begin
21        ff1 <= ff1;
22      end
23    end
24    assign Q=ff1;
25  endmodule
26
27 module TFF(input T,clk,rst,output Q);
28   flop TFF_1(.J(T),.K(~T),.clk(Clk),.rst(rst),.Q(Q));
29 endmodule
30
31 endmodule

```

LAB 10