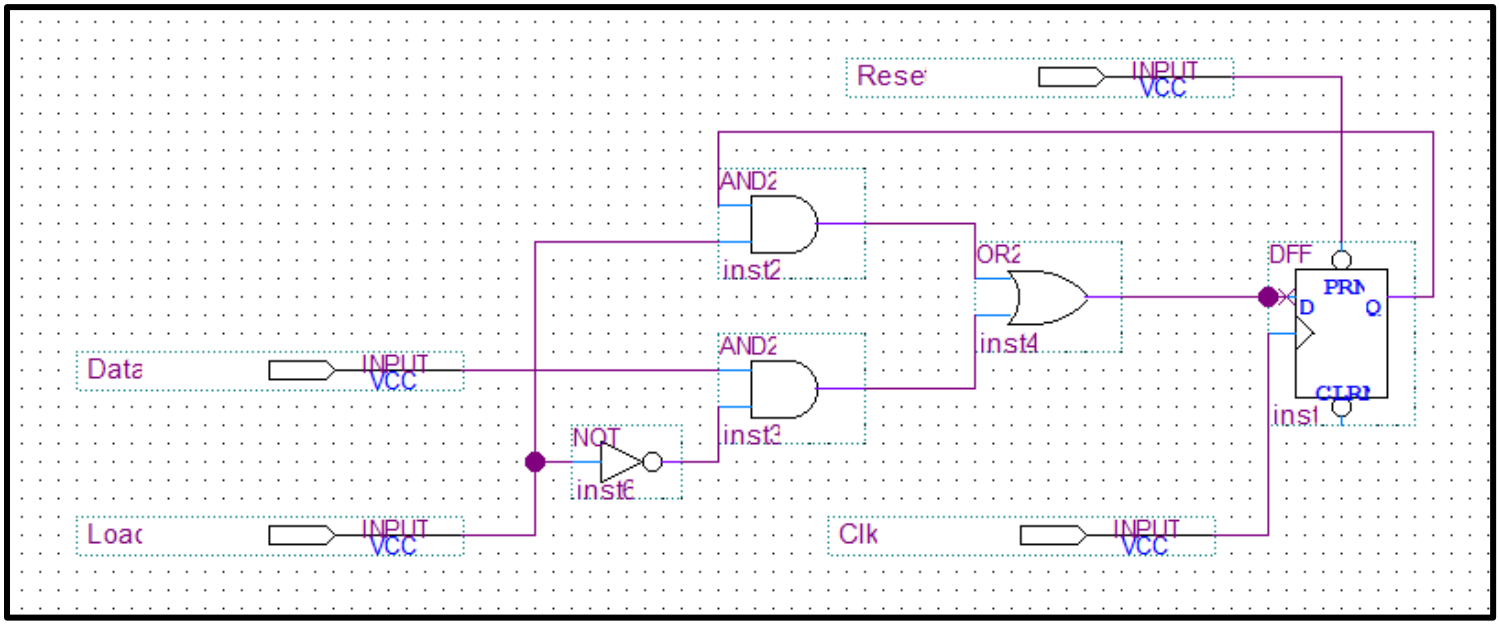
PRE-LAB 4&5

4.2 LAB 4

4.2.2 D FLIP-FLOP WITH ASYNCHRONOUS RESET AND SYNCHRONOUS LOAD

1.



2.

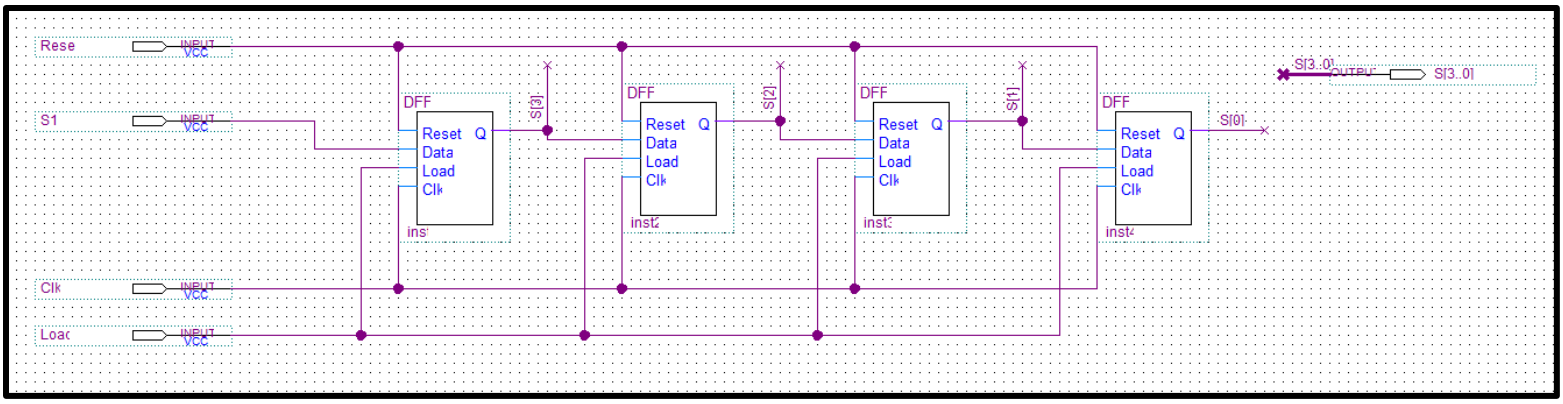
```
module DFF (Q, Data, Reset, Load, Clk);
output Q;
input Data, Reset, Load, Clk;
reg Q;
always @(posedge Clk or negedge Reset)
begin
        case(Reset)
                0: Q <= 0;
                default: Q <= Data;
        endcase
end
endmodule
```

3.

```
module DFF_tb();
 reg Clk, Data, Reset, Load;
 wire Q;
DFF u1(Q, Data, Reset, Load, Clk);
initial
begin
        Clk = 0;
        Load = 0; Data = 0; Reset = 0;
         #10 Load=1; Data=1; Reset=1;
        #10 Load=1; Data=0; Reset=1;
        #10 Load=0; Data=1; Reset=1;
        #10 Load=0; Data=0; Reset=1;
        #10 Load=1; Data=1; Reset=1;
        #10 Load=1; Data=1; Reset=1;
        #10 Load=1; Data=1; Reset=0;
        #10 Load=0; Data=1; Reset=0;
        #10 Load=1; Data=0; Reset=0;
        #10 Load=0; Data=1; Reset=1;
        #10 Load=1; Data=1; Reset=1;
end
always #10 Clk = ~Clk;
endmodule
```

4.2.3 4-BIT SHIFT REGISTER WITH ASYCHRONOUS RESET AND SYNCHRONOUS LOAD

1.



2.

```
module Shift_Reg (S0, Reset, S1, Load, Clk);
 parameter size=4;
 output reg [size-1:0] S0;
  input Reset, S1, Load, Clk;
 always @(posedge Reset)
        S0 <= 4'b0000;
always @(posedge Clk)
 begin
        S0[0] <= S1;
        S0[1] <= S0[0];
        S0[2] <= S0[1];
        S0[3] <= S0[2];
end
endmodule
```

3.

```
module Shift_Reg_tb ();
 parameter size=4;
 reg Reset, S1, Load, Clk;
 wire [size-1:0] S0;
 Shift_Reg u1(S0, Reset, S1, Load, Clk);
 initial
 begin
        Reset=1; Load=0; S1=0; Clk=0;
        #300 Reset=0; S1=1;
         #300 Reset=0; S1=1;
        #300 Reset=1; S1=0;
        #300 Reset=1; S1=1;
        #300 Reset=1; S1=0;
        #300 Reset=0; S1=0;
        #300 Reset=0; S1=1;
 end
 always #300 Clk = ~Clk;
endmodule
```

4.2.4 4-BIT SYNCHRONOUS WRAP-AROUNG UP/DOWN COUNTER WITH SYCNHRONOUS UP/DOWN SELECT AND COUNT-ENABLE AND ASYNCHRONOUS RESET

1.

```
module Counter(Q, E, R, up_down_s, Clk);
parameter size=4;
 output reg [size-1:0] Q;
input E, R, up_down_s, Clk;
always @(posedge R)
        Q <= 0;
        always @(negedge R)
        case(up_down_s)
                1:  Q <= Q+1 ;
                default:  Q <= Q-1;
         endcase
 endmodule
```

2.

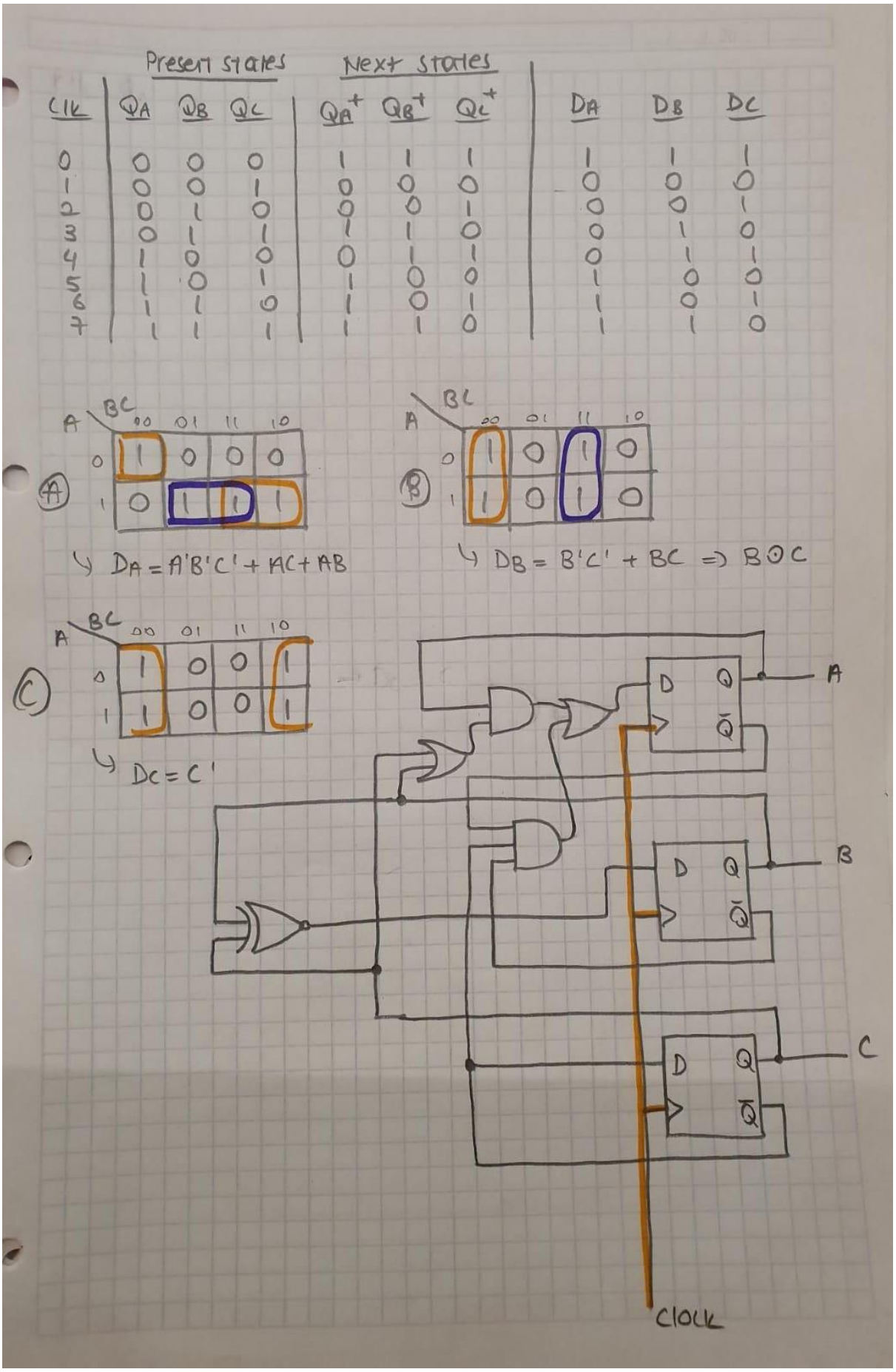
```
module Counter(SeqOut, Q, E, R, up_down_s, Clk);
parameter size=4;
 output reg [size-1:0] Q;
output reg [2*size-1:0] SeqOut;
input E, R, up_down_s, Clk;
always @(posedge R)
        Q <= 0;
always @(negedge R)
        case(up_down_s)
                1:  Q <= Q+1 ;
                default:  Q <= Q-1;
        endcase

always @(*)
 begin
case(Q)
        4'b0000 : SeqOut = 8'b00000001;
        4'b0001 : SeqOut = 8'b01100000;
        4'b0010 : SeqOut = 8'b11011010;
        4'b0011 : SeqOut = 8'b11110010;
        4'b0100 : SeqOut = 8'b01100110;
        4'b0101 : SeqOut = 8'b10110100;
        4'b0110 : SeqOut = 8'b10111110;
        4'b0111 : SeqOut = 8'b11100000;
        4'b1000 : SeqOut = 8'b01100000;
        4'b1001 : SeqOut = 8'b11011010;
        4'b1010 : SeqOut = 8'b11110010;
        4'b1011 : SeqOut = 8'b01100110;
        4'b1100 : SeqOut = 8'b10110100;
        4'b1101 : SeqOut = 8'b10111110;
        4'b1110 : SeqOut = 8'b11100000;
        4'b1111 : SeqOut = 8'b11111110;
endcase
end
endmodule
```

3.

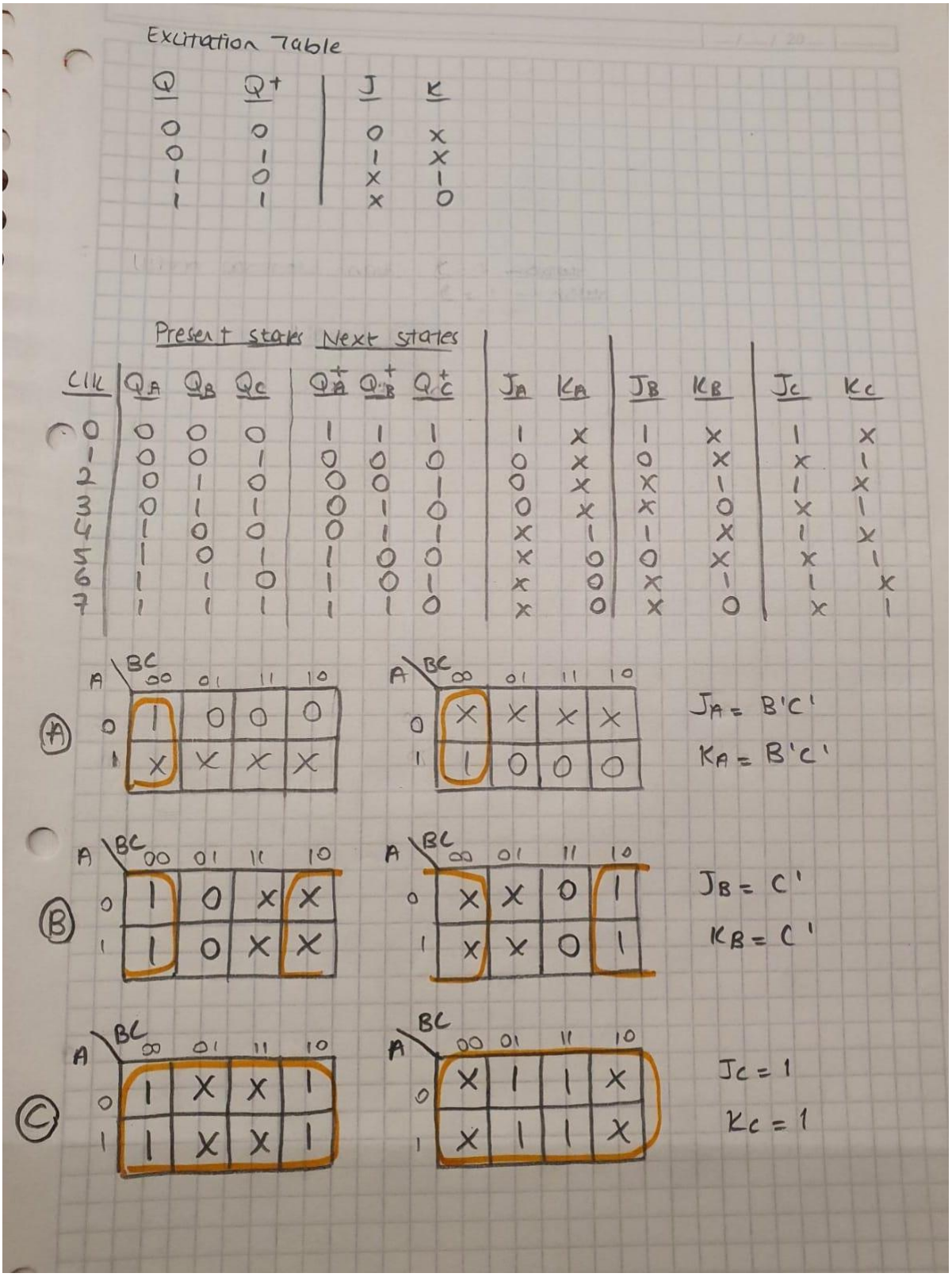
```
module Counter_tb();
parameter size=4;
reg E, R, up_down_s, Clk;
wire [size-1:0] Q;
wire [2*size-1:0] SeqOut;

 Counter u1(SeqOut, Q, E, R, up_down_s, Clk);

 initial
begin
        R=0; Clk=0; up_down_s=0; E=0;
        #200 R=1; up_down_s=0; E=0;
        #200 R=1; up_down_s=1; E=1;
        #200 R=1; up_down_s=1; E=0;
        #200 R=0; up_down_s=0; E=1;
        #200 R=0; up_down_s=1; E=1;
        #200 R=1; up_down_s=0; E=0;
        #200 R=1; up_down_s=1; E=1;
        #200 R=1; up_down_s=1; E=1;
        #200 R=0; up_down_s=1; E=1;
        #200 R=0; up_down_s=1; E=0;
        #200 R=0; up_down_s=0; E=0;
        #200 R=0; up_down_s=1; E=1;
        #200 R=0; up_down_s=1; E=1;
        #200 R=0; up_down_s=1; E=1;
        #200 R=0; up_down_s=1; E=1;
        #200 R=0; up_down_s=0; E=1;
        #200 R=0; up_down_s=0; E=1;
        #200 R=0; up_down_s=0; E=1;
        #200 R=0; up_down_s=1; E=0;
end
always #200 Clk = ~Clk;
endmodule
```

4.3 LAB 5

4.3.1 3-BIT DOWN COUNTER USING D-TYPE FLIP-FLOP



4.3.2 3-BIT DOWN COUNTER USING JK-TYPE FLIP-FLOP



Excitation Table

Q	Q^+	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

	Present states			Next states									
CLK	Q_A	Q_B	Q_C	Q_A^+	Q_B^+	Q_C^+	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	1	1	1	x	1	x	1	x
1	0	0	1	0	0	0	0	x	0	x	x	1
2	0	1	0	0	0	1	0	x	x	1	1	x
3	0	1	1	0	1	0	0	x	x	0	x	1
4	1	0	0	0	1	1	x	1	1	x	1	x
5	1	0	1	1	0	0	x	1	0	x	x	1
6	1	1	0	1	0	1	x	0	x	1	1	x
7	1	1	1	1	1	0	x	0	x	0	x	1

(A)

A \ BC	00	01	11	10
0	1	0	0	0
1	x	x	x	x

A \ BC	00	01	11	10
0	x	x	x	x
1	1	0	0	0

$J_A = B'C'$

$K_A = B'C'$

(B)

A \ BC	00	01	11	10
0	1	0	x	x
1	1	0	x	x

A \ BC	00	01	11	10
0	x	x	0	1
1	x	x	0	1

$J_B = C'$

$K_B = C'$

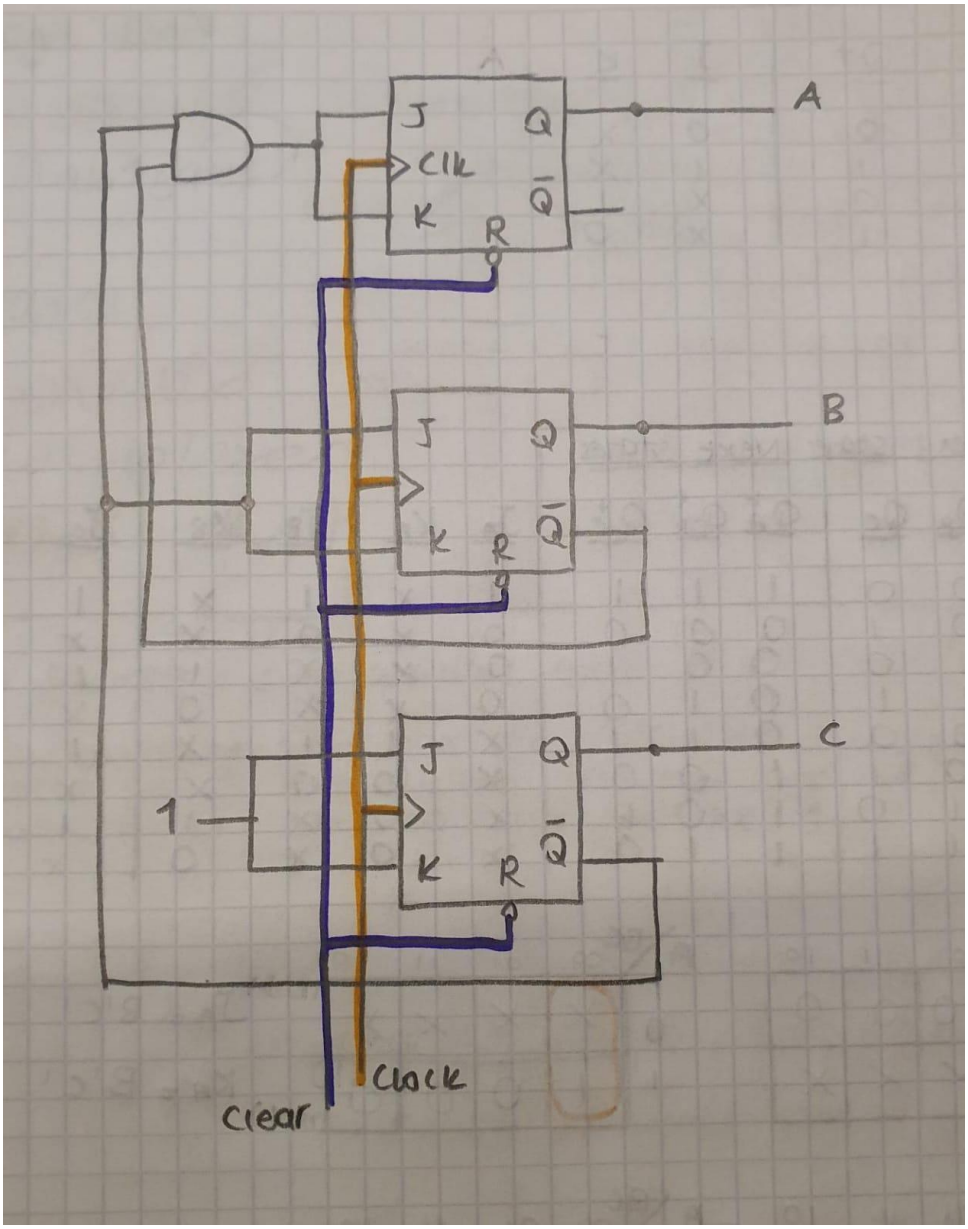
(C)

A \ BC	00	01	11	10
0	1	x	x	1
1	1	x	x	1

A \ BC	00	01	11	10
0	x	1	1	x
1	x	1	1	x

$J_C = 1$

$K_C = 1$



4.3.3 4-BIT SYNCHRONOUS PARALLEL LOAD SHIFT REGISTER WITH COUNTER AND ASYNCHRONOUS RESET

1.

```
module Count_Ones (Data, en, rst, clk, load, Q, Ones);
  parameter size=4;
  output reg [2*size-1:0] Q;
  output reg [size-1:0] Ones;
  input [2*size-1:0] Data;
  input en, rst, clk, load;
  always @(posedge rst)
  begin
    Q <= 0;
    Ones <= 0;
  end
```

```
  integer i;
  always @(negedge rst)
  begin
   case(en)
    0: begin
        Q <= 0;
        Ones <= 0;
       end
    default:
     case(load)
      1: begin
          Q <= Data;
          Ones <= 0;
         end
      default: begin
               Q <= Data;
               begin
                 Ones=0 ;
                 for (i=0 ; i<2*size ; i=i+1)
                   Ones = Ones + Data[i];
               end
              end
      endcase
    endcase
   end
endmodule
```

2.

```
module Count_Ones (Data, en, rst, clk, load, Q, Ones, SeqOut);
 parameter size=4;
 output reg [2*size-1:0] Q, SeqOut;
 output reg [size-1:0] Ones;
 input [2*size-1:0] Data;
 input en, rst, clk, load;
 always @(posedge rst)
 begin
          Q <= 0;
          Ones <= 0;
```

```
        end
        integer i;
        always @(negedge rst)
        begin
                case(en)
                0: begin
                        Q <= 0;
                        Ones <= 0;
                end
                default:
                case(load)
                1: begin
                        Q <= Data;
                        Ones <= 0;
                end
                default: begin
                Q <= Data;
            begin
                Ones=0 ;
                for (i=0 ; i<2*size ; i=i+1)
                        Ones = Ones + Data[i];
             end
            end
       endcase
    endcase
  end
always @(*)
 begin
  case(Ones)
    4'b0000 : SeqOut = 8'b00000001;
    4'b0001 : SeqOut = 8'b01100000;
    4'b0010 : SeqOut = 8'b11011010;
    4'b0011 : SeqOut = 8'b11110010;
    4'b0100 : SeqOut = 8'b01100110;
    4'b0101 : SeqOut = 8'b10110100;
    4'b0110 : SeqOut = 8'b10111110;
    4'b0111 : SeqOut = 8'b11100000;
    4'b1000 : SeqOut = 8'b01100000;
```

```
    4'b1001 : SeqOut = 8'b11011010;
    4'b1010 : SeqOut = 8'b11110010;
    4'b1011 : SeqOut = 8'b01100110;
    4'b1100 : SeqOut = 8'b10110100;
    4'b1101 : SeqOut = 8'b10111110;
    4'b1110 : SeqOut = 8'b11100000;
    4'b1111 : SeqOut = 8'b11111110;
  endcase
 end
endmodule
```

3.

```
module Count_Ones_tb();
parameter size=4;
 reg [2*size-1:0] Data;
reg en, rst, clk, load;
wire [2*size-1:0] Q, SeqOut;
wire [size-1:0] Ones;
Count_Ones u1(Data, en, rst, clk, load, Q, Ones, SeqOut);
initial
begin
        Data=0; en=0; rst=1; clk=0; load=0;
        #200 Data=00110101; rst=0; load=1; en=1;
        #200 Data=00110101; rst=0; load=1; en=0;
        #200 Data=00111010; rst=0; load=0; en=1;
        #1000 Data=00110111; rst=0; load=1; en=1;
        #200 Data=00100000; rst=0; load=0; en=1;
        #200 Data=11110101; rst=0; load=0; en=1;
        #200 Data=00010111; rst=0; load=0; en=0;
        #200 Data=01110101; rst=0; load=1; en=0;
        #200 Data=00110001; rst=1; load=0; en=1;
        #1000 Data=00110000; rst=0; load=0; en=1;
        #200 Data=00010000; rst=0; load=0; en=1;
        #200 Data=11110111; rst=0; load=0; en=1;
        #200 Data=00111111; rst=1; load=0; en=0;
        #200 Data=00110101; rst=0; load=0; en=1;
        #1000 Data=00111101; rst=1; load=0; en=1;
        #200 Data=10110001; rst=0; load=0; en=1;
```

```
                #200 Data=00110000; rst=0; load=0; en=1;
                #200 Data=00100101; rst=0; load=0; en=1;
                #200 Data=10001101; rst=0; load=1; en=1;
        end
  always #20 clk = ~clk;
endmodule
```