Homework 10

Solutions

1. $Q1* = Q1' + Q2$

   $Q2* = X.Q2'$

   $Z = Q1 + Q2'$

Q1	**Q2**	**Q1*Q2***		**Z**	
		X =0	**X =1**	**X=0**	**X=1**
0	0	10	11	1	1
0	1	10	10	0	0
1	0	00	01	1	1
1	1	10	10	1	1

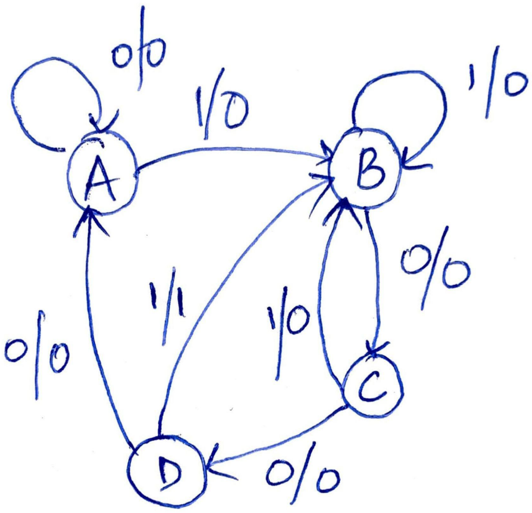
Present state	**Input X = 0**	**Input X = 1**
	Next state, Output	
A	C,1	D,1
B	C,0	C,0
C	A,1	B,1
D	C,1	C,1

2. Overlapping sequence 1001

   State A - 3 or more 1s
   State B - 1 or more consecutive 1s
   State C - 10

State D - 100

Present state	Input X = 0	Input X = 1
	Next state, Output	
A	A, 0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	B,1

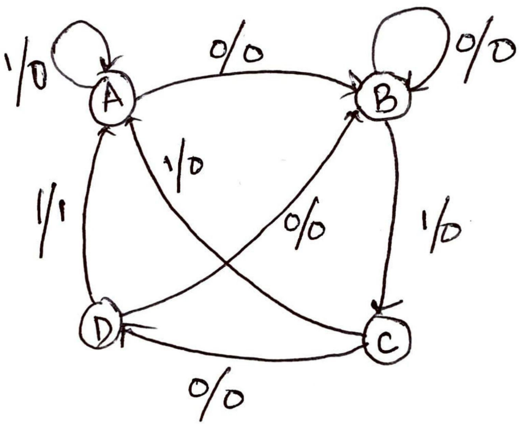


3. State A - 0101, 2 or more 1s

   State B - 1 or more 0s

   State C - 01

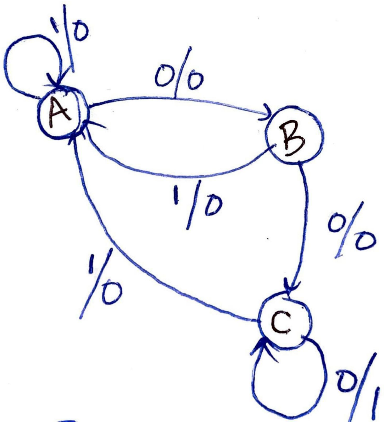
   State D - 010



4. State A - 1or more 1s

   State B - 0

   State C - 2 or more 0s



5.

Present state	**Input X = 0**	**Input X = 1**
	Next state, Output	
A	B, 0	C,1
B	D,0	F,1

C	F,0	E,0
D	B,0	G,1
E	F,0	C,0
F	E,0	D,0
G	F,0	G,0

P0 = (ABCDEFG)

To perform state minimization, we first separate the states based on the outputs for a 1 bit input.

For X=0, all outputs are same.

For X=1, outputs are 1 for ABD and 0 for CEFG.

Hence, P1 = (ABD)(CEFG)

P2:

X=0 : ABD => BDB

CEFG => FFEF

X=1: ABD => CFG

CEFG => ECDG

Since ECG and D are in separate set of states, we can separate F from CEG

Thus P2 = (ABD)(CEG)(F)

P3:

X=0 : ABD => BDB

CEG => FFF

F => E

X=1 : ABD => CFG

CEG => ECG

F => D

Since CG and F are in separate set of states, we can separate B from AD.

Thus P3 = (AD)(B)(CEG)(F)

P4:

X=0 : AD => BB

B => D

CEG => FFF

F => E

X=1 : AD => CG

B => F

CEG => ECG

F => D

P4 = (AD)(B)(CEG)(F)

Since P3=P4, no further separation can be done,

From P4, it is evident that A, D are equivalent states. And C,E,G are equivalent states.

We can therefore remove D,E,G from the state table.

Thus after minimization the state table looks like:

Present state	**Input X = 0**	**Input X = 1**
	Next state, Output	
A	B, 0	C,1
B	A,0	F,1
C	F,0	C,0
F	C,0	A,0

6.

Present state	**Input X = 0**	**Input X = 1**
	Next state, Output	
A	A, 0	B,0
B	C,0	B,0
C	D,0	B,0
D	A,0	E,0
E	C,0	B,1

State minimization:

P0 = (ABCDE)

P1:

X=0, all outputs are 0.

X=1, outputs are 0 for ABCD and 1 for E

P1 = (ABCD)(E)

P2:

X=0: ABCD => ACDA

E => C

X=1: ABCD => BBBE

E => B

P2 = (ABC)(D)(E)

P3:

X=0: ABC => ACD

D => A

E => C

X=1: ABC => BBB

D => E

E => B

P3 = (AB)(C)(D)(E)

P4:

X=0: AB => AC

C => D

D => A

E => C

X=1: AB => BB

C => B

D => E

E => B

P4 = (A)(B)(C)(D)(E)

No minimization is possible.

State assignment:

Present state Q0Q1Q2	**Input X = 0**	**Input X = 1**
	Next state, Output Q0*Q1*Q2*,Z	
000	000, 0	001,0
001	010,0	001,0
010	011,0	001,0
011	000,0	100,0
100	010,0	001,1

Q2X	00	01	11	10
Q0Q1 00				
01			1	
11	X	X	X	X

10			X	X

Q0* = Q1Q2X

Q2X	00	01	11	10
Q0Q1 00				1
01	1			
11	X	X	X	X
10	1		X	X

Q1* = Q1Q2’X’ + Q1’Q2X’ + Q0Q2’X’

Q2X	00	01	11	10
Q0Q1 00		1	1	
01	1	1		
11	X	X	X	X
10		1	X	X

Q2* = Q2’X + Q1Q2’ + Q0’Q1’X

Q2X	00	01	11	10
Q0Q1 00				
01				
11	X	X	X	X
10		1	X	X

Z = Q0X

7. .

Q0Q1Q2	Q0*Q1*Q2* D0D1D2	
	UP=1	UP=0
000	001	111
001	010	000
010	011	001
011	100	010
100	101	011
101	110	100
110	111	101
111	000	110

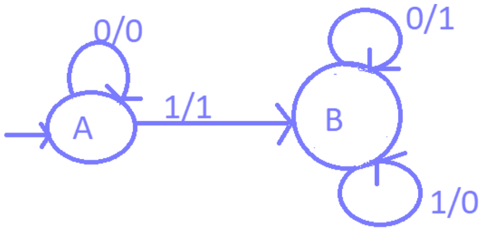
8. Total no. of unique states required = (70+5+75)/5 = 30

No. of flip flops required = n

$2^n >= 30$

Thus, n = 5

9.



We know the regular method to find 2’s complement. Another technique that is useful to draw a state machine is described below:

Start from the right. Do not change the no.s until you reach a 1. Leave the first 1 as such. Complement every no. to the left of the 1.

10. In Moore model the output depends only on the present state.

In Mealy model the output depends on both the present state and the inputs.

```
module FSM(CLOCK,XZ);
input CLOCK,X;
output reg Z;
reg [2:0] Sreg,Snext;
parameter [2:0] S0=3’b000,


S1=3’b001,
S2=3’b010,
S3=3’b011


/* create state memory
always @(posedge CLOCK)

Sreg <= Snext;

/* create next-state logic */

always @(X,Sreg)
begin
case(Sreg)
S0: if(X==0) Snext=S0;
else Snext=S1;
S1: if(X==0) Snext=S2;
else Snext=S0;
S2: if(X==0) Snext=S3;
else Snext=S2;
S3: if(X==0) Snext=S1;
Snext=S0;
default: Snext=S0;
endcase
end

/* create output logic */
always @(Sreg)
case(Sreg
S0,S1,S2: Z=0;
S3: Z=1;
default: Z=0;
endcase
endmodule
```