Vinod Halaharvi
Assignment#3
Computer Architecture

1)

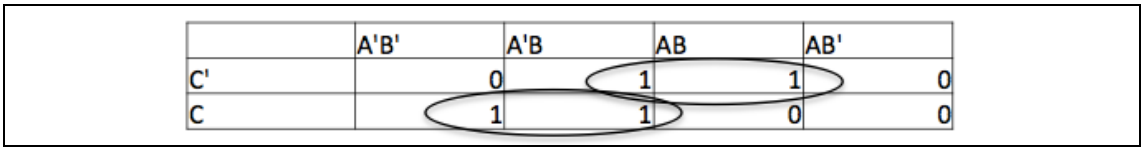
Please see problem code in **prob1** directory.

REFERENCE:

Igor's section vhdl programs
Altera.com website

2) Katz and Borriello 3.18 (Hazard-Free Design)

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

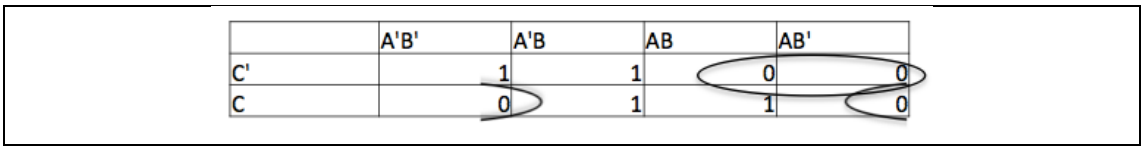


From k map above we have to add another prime implicant (redundant) to avoid type 1 static hazard. We have to add $A'B$.

So, $F_1(A,B,C) = A'B + BC' + A'C$

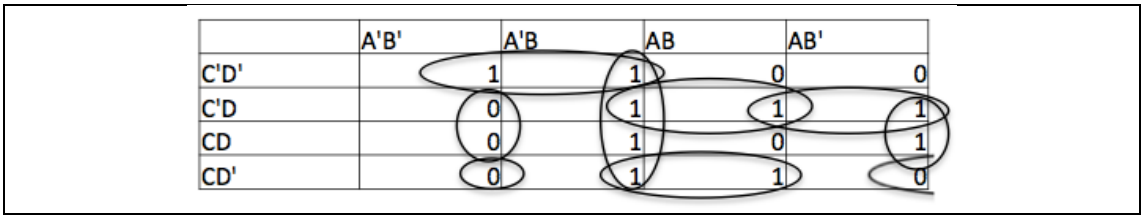
Now we have to see if there are any type of 0 hazards.

$F' = (A'B + BC' + A'C)'$
$= (A + B')(B' + C)(A + C')$



We see that all prime implicants are covered. So state 0 hazard free.

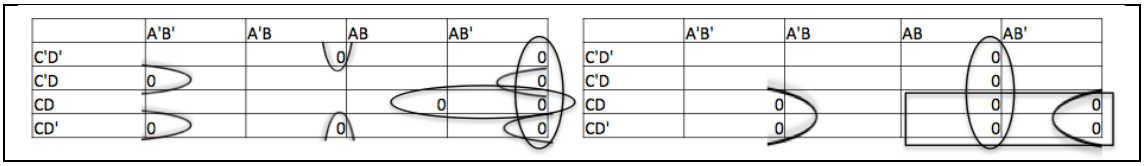
(b) $F(A,B,C,D) = \sum m\,(0,4,5,6,7,9,11,13,14)$



$F = A'C'D' + A'B + BC'D + AC'D + AB'D + BCD'$

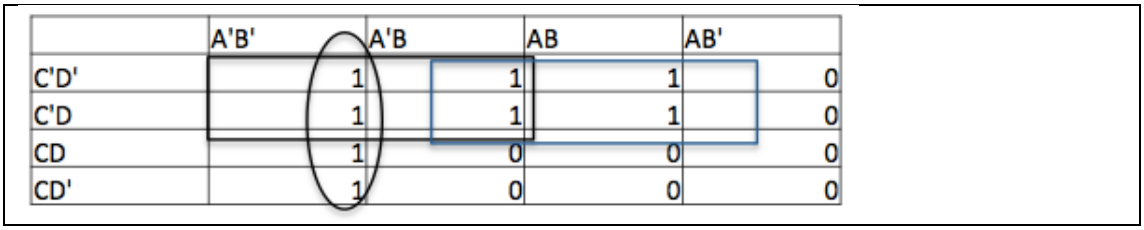
$F' = (A + C + D)(A + B')(B' + C + D')(A' + C + D')(A' + B + D')(B' + C' + D)$

All prime implicants are covered and no state 0 hazard.

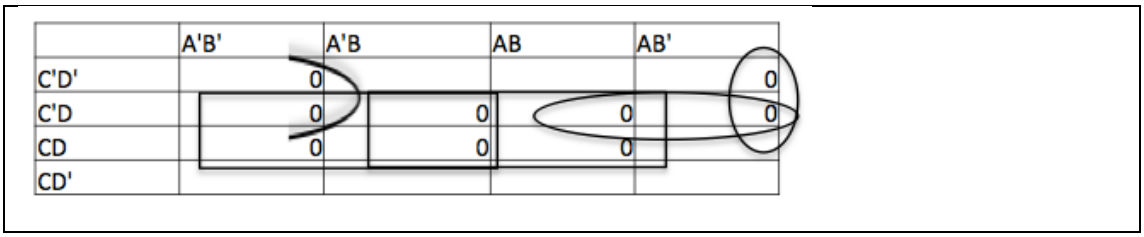


(c)
<< please see above, the image to the right >>
$F = (A + B)(B' + C)(A + C)$

$F' = A'B' + BC' + A'C'$. Which covers all prime implicants and there are no state 0 hazards.



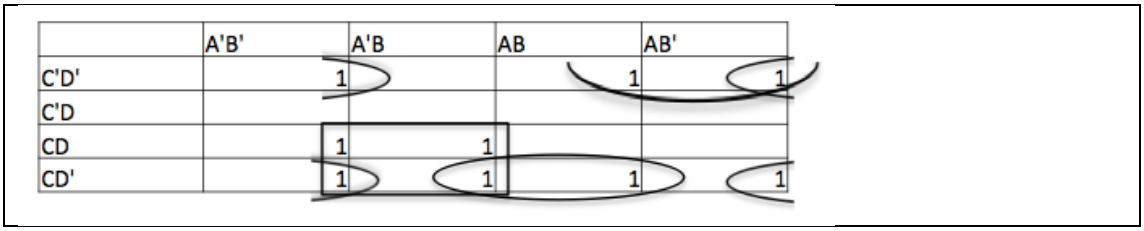
d)



$\pi M\ (0, 1, 3, 5, 7, 8, 9, 13, 15)$

$F = (A + C')(B + D)(A' + D)(A + C' + D)$

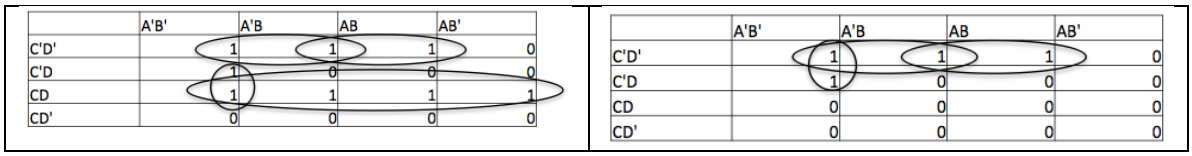
$F' = A'C + B'D' + AD' + A'CD'$



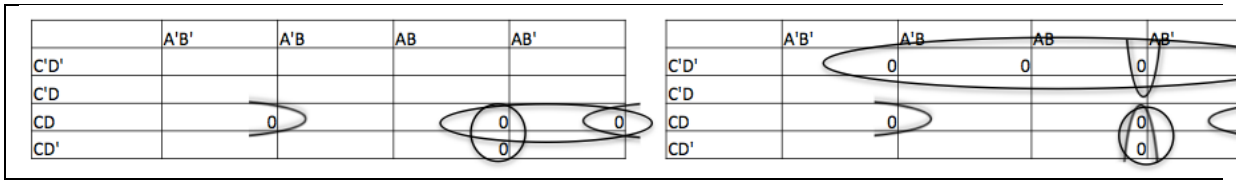
Adding additional implication BCD' to remove type 0 static hazard.

$F' = A'C + B'D' + AD' + A'CD' + BCD'$
No more static hazards.

(e)



$F = CDE' + A'B'C' + BC'D' + A'C'D' + A'B'DE'$



$F' = (C' + D' + E)(A + B + C)(B' + C + D)(A + C + D) + (A + B + D' + E)$

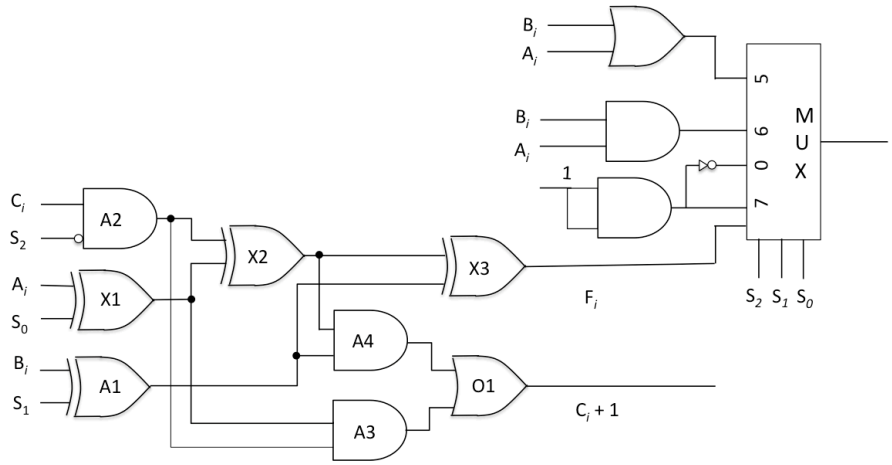
No type 0 state hazard

3) Katz and Borriello 5.12 (ALU Design)

ALU Design

S_2	S_1	S_0	*ALU Operation*
0	0	0	$F_i = 0$
0	0	1	$F_i = B$ minus A
0	1	0	$F_i = A$ minus B
0	1	1	$F_i = A$ plus B
1	0	0	$F_i = A$ XOR B
1	0	1	$F_i = A$ OR B
1	1	0	$F_i = A$ AND B
1	1	1	$F_i = 1$

4 – Multilevel ALU bit



REFERENCE: Katz and Borriello, page 249

From the above figure we have as follows,

When S2 = 0, S1 = 0 and S0 = 0 and when S2 = 1, S1 = 1 and S0 = 1 we simple apply this input to the mux to select 0 and 1 respectively. It's easy to build a MUX using AND gates, we can select any input and copy that input to the output. Similarly S2 = 1, S1 = 0 and S0 =1 we can select Ai OR Bi and when S2 =1, S1 = 1 and S0 = 0 we can select Ai AND Bi. For all other inputs of S2, S1, S0 we select Fi. Now we see how Fi gets the right value consistent with the table in this problem set.

When S2 = 0, S1 = 0 and S0 = 1: Since S2 = 0, Ci is asserted at the output gate of A2, Since S0 is 1, ~Ai is asserted at output of Gate of X1 and hence ~Ai XOR 1 is asserted at the output of Gate X2 when Ci = 1. Since S1 = 0, B is asserted at the output of A1. So B and (~Ai XOR 1) is asserted at the input of X3. We know (B XOR ~Ai XOR 1) gives us Bi minus Ai at the output of X3.

When S2 = 0, S1 = 1 and S0 = 0: Since S2 = 0, Ci is asserted at the output gate of A2, Since S0 is 0, Ai is asserted at output of Gate of X1 and hence Ai XOR 1 is asserted at the output of Gate X2 when Ci = 1. Since S1 = 1, ~B is asserted at the output of A1. So ~B and (Ai XOR 1) is asserted at the input of X3. We know (~Bi XOR Ai XOR 1) gives us Ai minus Bi at the output of X3.

When S2 = 0, S1 = 1 and S0 = 1: Since S2 = 0, Ci is asserted at the output gate of A2, Since S0 is 1, ~Ai is asserted at output of Gate of X1 and hence ~A XOR 1 is asserted at the output of Gate X2 when Ci = 1. Since S1 = 1, ~B is asserted at the output of A1. So ~B and (~A XOR 1) is asserted at the input of X3. We know (~B XOR ~A XOR 1) gives us A plus B at the output of X3.

When S2 = 1, S1 = 0 and S0 = 1: Since S2 = 1, 0 asserted at the output gate of A2 regardless of the value of Ci, Since S0 is 0, Ai is asserted at output of Gate of X1 and

hence Ai XOR 0 is asserted at the output of Gate X2 which is simply Ai. Since S1 = 0, B is asserted at the output of A1. So Bi and Ai is asserted at the input of X3. So we get Ai XOR Bi at the output of Gate X3.

For all the above case we have Ai.Ci OR Bi(Ai XOR Ci) asserted at gave O1, which is the correct form of carry. So our circuit works correctly for the table shown above.

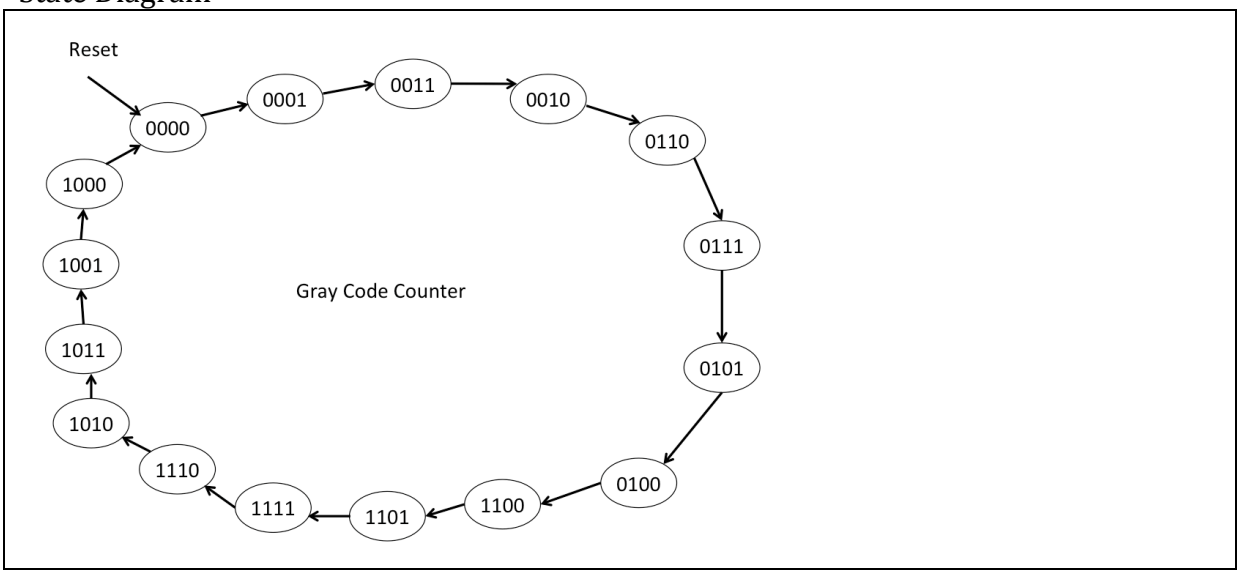
4) Katz and Borriello 7.4 (Counter Design)

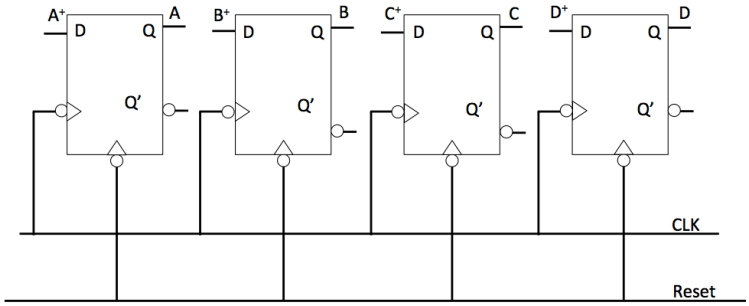
a)

Next-Sate table

Present State	Next State
0000	0001
0001	0011
0011	0010
0010	0110
0110	0111
0111	0101
0101	0100
0100	1100
1100	1101
1101	1111
1111	1110
1110	1010
1010	1011
1011	1001
1001	1000
1000	0000

State Diagram





Please see below for the K-Map and the simplified next state output values derived from the current state values

	A'B'	A'B	AB	AB'
C'D'	0	1	1	0
C'D	0	0	1	1
CD	0	0	1	1
CD'	0	0	1	1

$A^+ = AB + AC + AD + B'C'D'$

	A'B'	A'B	AB	AB'
C'D'	0	1	1	0
C'D	0	1	1	0
CD	0	1	1	0
CD'	1	1	0	0

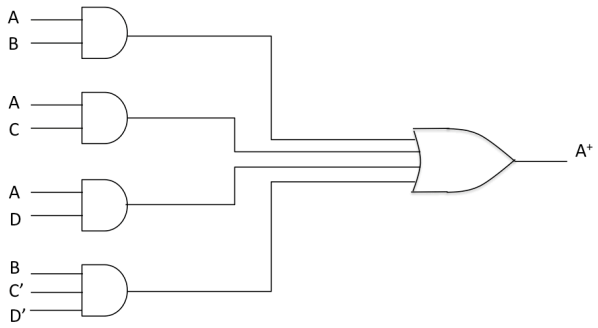
$B^+ = A'B + BC' + BD + A'CD'$

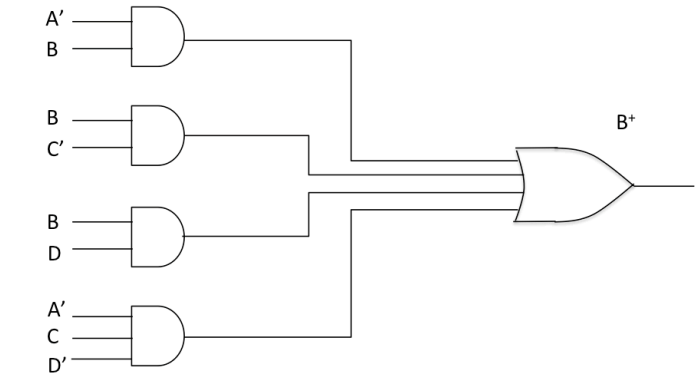
	A'B'	A'B	AB	AB'
C'D'	0	0	0	0
C'D	1	0	1	0
CD	1	0	1	0
CD'	1	1	1	1

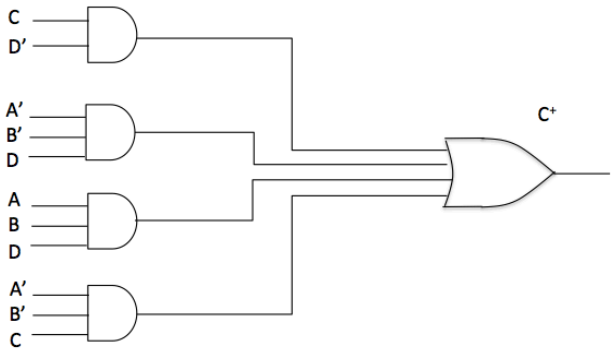
$C^+ = CD' + A'B'C + ABD + A'B'D$

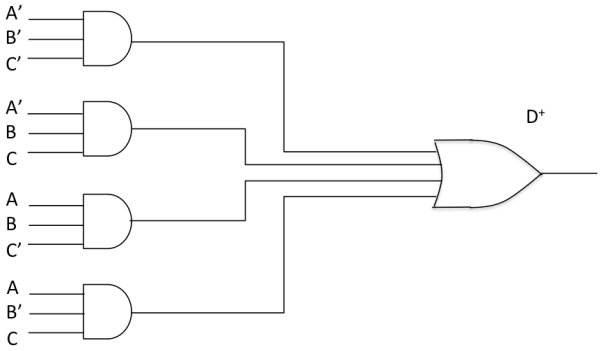
	A'B'	A'B	AB	AB'
C'D'	1	0	1	0
C'D	1	0	1	0
CD	0	1	0	1
CD'	0	1	0	1

$D^+ = A'B'C' + A'BC + ABC' + AB'C$









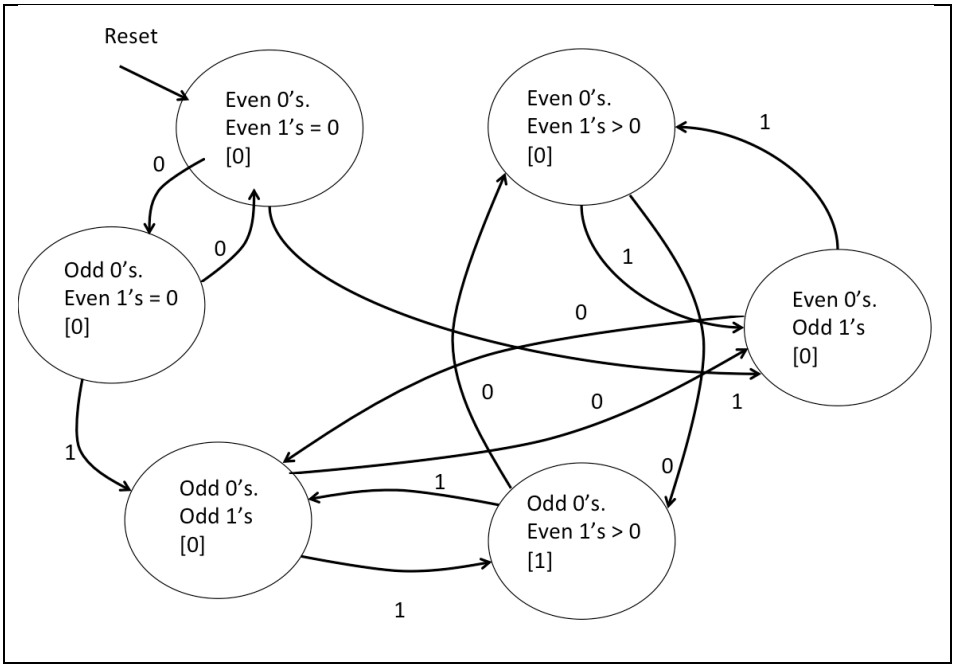
Yes, this counter is self-starting. Counters will all possible states are occupied are always self-starting so, yes!

5) Katz and Borriello 7.27 (Word Problem)

Moore Machine

Abbreviations
Even 0's, Even 1's = 0
Odd 0's, Even 1's = 0
Even 0's, Even 1's > 0
Even 0's, Odd 1's > 0
Odd 0's, Even 1's > 0
Odd 0's, Odd 1's > 0

Please see below for the state transition diagram:



6) Katz and Borriello 8.2 (State Reduction)

We will use row-matching algorithm here. Even though row-matching algorithm does not always produce the most efficient FSM, it is lot simpler to work with than implication chart. So we use this method. Please see for the row-matching table below

Present State	NextState		Output	
	X = 0	X = 1	X = 0	X = 1
A	B	C	0	0
B	A	C	0	0
C	D	C	1	0
D	D	E	1	0
E	A	F	0	0
F	B	G	0	0
G	A	E	0	0

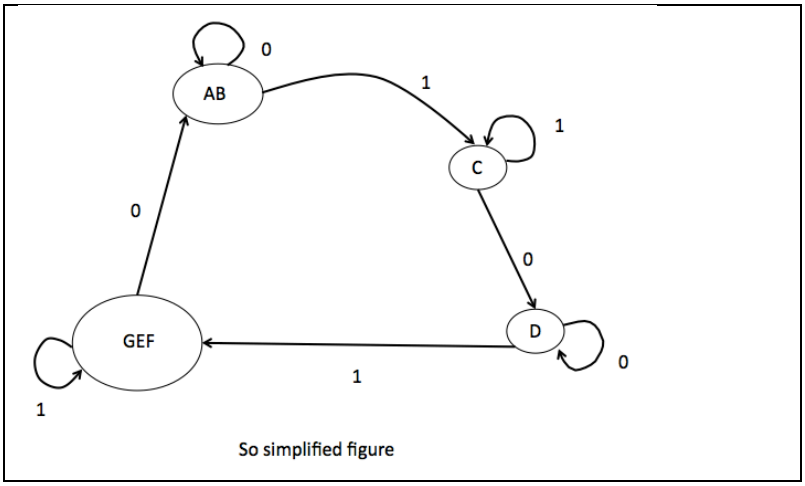
For both states A and B, the next state and output values are the same, so states A and B are equivalent and we can merge them into a single state.
Similarly in table 2 we have state E and F equivalent and in table 3 we have state G and EF equivalent. So we can merge state G, E and F in to a single state.

Present State	NextState		Output	
	X = 0	X = 1	X = 0	X = 1
AB	AB	C	0	0
C	D	C	1	0
D	D	E	1	0
E	AB	F	0	0
F	AB	G	0	0
G	AB	E	0	0

Present State	NextState		Output	
	X = 0	X = 1	X = 0	X = 1
AB	AB	C	0	0
C	D	C	1	0
D	D	EF	1	0
EF	AB	G	0	0
G	AB	EF	0	0

Present State	NextState		Output	
	X = 0	X = 1	X = 0	X = 1
AB	AB	C	0	0
C	D	C	1	0
D	D	GEF	1	0
GEF	AB	GEF	0	0

The reduce FSM is as shown below. This is equivalent to the FSM we started with

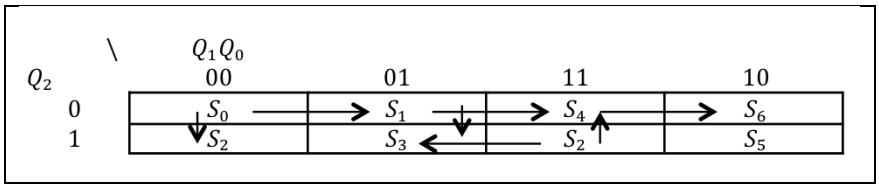


So simplified figure

---

7) Katz and Borriello 8.7 (State Assignment)

(a)
Since we have 7 states we can use three bits to encode the states. Looking at the state diagram in the problem set, we have come up with the following transitions that produces the minimum bit changes

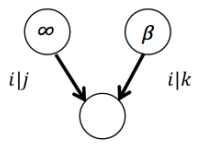


Transition		Assignment bit changes
$S_0 \rightarrow S_1$	1	
$S_0 \rightarrow S_2$	1	
$S_1 \rightarrow S_3$	1	
$S_1 \rightarrow S_4$	1	Total bit changes = 13
$S_2 \rightarrow S_3$	1	
$S_2 \rightarrow S_4$	1	
$S_3 \rightarrow S_6$	2	
$S_3 \rightarrow S_5$	2	
$S_4 \rightarrow S_6$	1	
$S_6 \rightarrow S_1$	1	
$S_5 \rightarrow S_0$	2	

Based on the K-Map above we have the following state encodings.
S0 -> 000
S1 -> 001
S2 -> 100
S3 -> 101
S4 -> 011
S5 -> 110
S6 -> 010

(b) Now we use state assignment guidelines for the same problem. We have the following state priorities,

Highest: $\{S_1, S_2\}$, $\{S_3, S_4\}, \{S_5, S_6\}$
Based on



Medium: $\{S_1, S_2\}, \{S_3, S_4\}$ x 2, $\{S_5, S_6\}$

Lowest:

$0|0: \{S_0, S_5, S_6, S_4\}$

$1|0: \{S_0, S_1, S_5, S_2, S_4\}$

$0|1: \{S_1, S_3, S_6\}$

$1|1: \{S_3, S_6\}$

$S_0 \rightarrow 000$ $S_4 \rightarrow 111$
$S_1 \rightarrow 001$ $S_5 \rightarrow 010$
$S_2 \rightarrow 101$ $S_6 \rightarrow 110$
$S_3 \rightarrow 011$

Please see below:

Q_2 \ Q_1Q_0	00	01	11	10
0	S_0	S_1	S_3	S_5
1		S_2	S_4	S_6

8 Final Program in High-level Language

Please see folder prob8 for the programming problem assignment. Please read **README.txt** to see how to run the program.