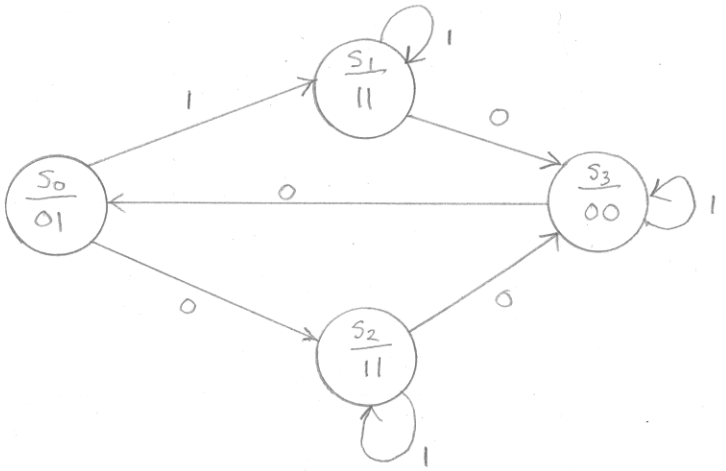
15.1 Elimination of Redundant States

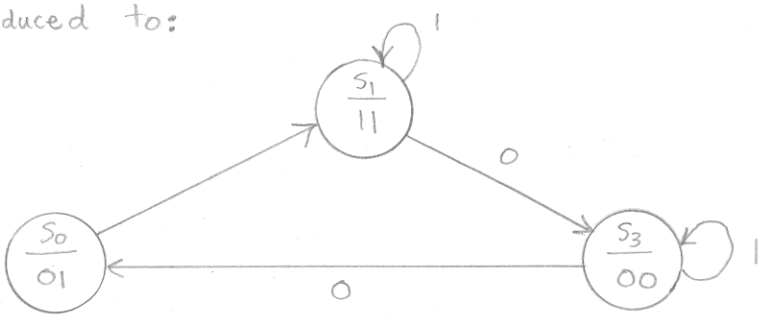
We can simplify state machine design by finding and removing equivalent states.

Ex:



4 state FSM
1 input
2 outputs
Moore machine

Can be reduced to:



S_2 is equivalent to S_1 ($S_2 \equiv S_1$)

Number of states: 4 → 3
Number of required flip-flops: 2 → 2, but input now has a "don't care" condition (transition from S_0 to S_1), so logic can be simpler.

How do we determine if S_2 is equivalent to S_1?

Look at the state table:

Present State	Next State A=0	Next State A=1	Present Output
S_0	S_2	S_1	01
S_1	S_3 (same)	S_1 (?)	11 (Same Outputs)
S_2	S_3 (same)	S_2 (?)	11 (Same Outputs)
S_3	S_0	S_3	00

Check:
Outputs same? Yes
Next states same? Yes –
S_1 can't be distinguished from S_2 since outputs are same!

This "row matching" procedure is not sufficient to find all equivalent states.

Tests for Equivalent States

I. Impossible to tell the states apart from observing inputs and outputs of circuit (e.g., S_1 and S_2 from above).

II. Test Procedure: State p $\stackrel{?}{\equiv}$ State q

1.) (a) Start in State p
(b) Start in state q

2.) Try all possible input sequence combinations

3.) if outputs are always the same, states p and q are equivalent.

This procedure is BAD, because step 2.) requires an infinite number of input samples.

III. (Theorem 15.1)
States p and q are equivalent ($p \equiv q$) iff (if and only if) for every single input X, the outputs are the same and the next states are equivalent

Notation: $\lambda \equiv$ outputs $\quad \lambda(p,X) = \lambda(q,X)$

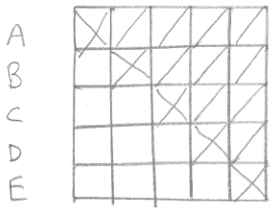
$\delta \equiv$ next states $\quad \delta(p,X) \equiv \delta(q,X)$

15.3 Implication Table

A robust method for finding equivalent states is to use an implication table (chart of pairs between states)

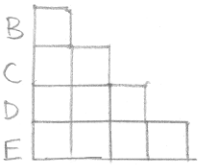
Check each pair for equivalence by checking (1) same outputs and (2) equivalent next states.

Ex: Suppose we have 5 states, A, B, C, D, and E. To check all pairs, there are 5x5 = 25 combinations. How many are unique (i.e., how many do we need to check)?



A-B same as B-A, etc.
A ≡ A for all states

⇒



If two states have different outputs ⇒ NOT equivalent
If two states have the same outputs ⇒ MAYBE equivalent, but you must check next states

Implication Table Procedure:

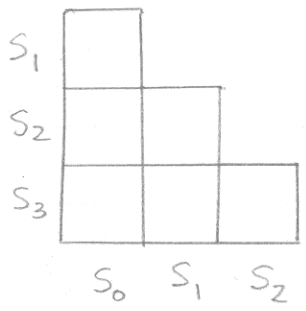
1.) Make chart of boxes for each pair of states

2.) Compare state table rows for each pair.
   a) Different outputs ⇒ put X in Implication Table box.
   b.) Same outputs ⇒ Write implied pair into box.

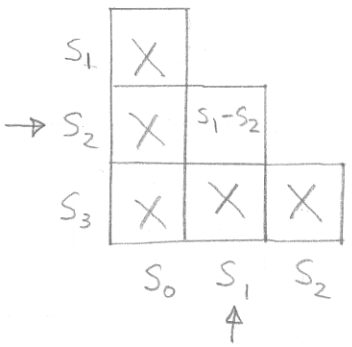
3.) Repeat, until no more new X's are added.

4.) Any box which does NOT have an X indicates a pair of equivalent states.

Ex: state machine discussed above; 4 states → S_0, S_1, S_2, S_3



Iteration 1: Add X's to different outputs, label implied pairs:

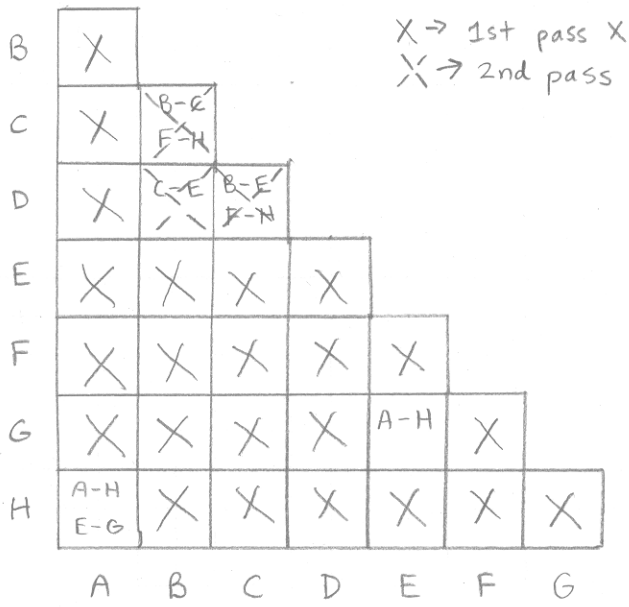


Iteration 2: No new boxes can be crossed off, no new X's added.

Therefore, $S_1 \equiv S_2$ (Note, the indexed pair is equivalent, not the implied pair in the box).

Ex:

Present State	Next State X=0	Next State X=1	Present output X=0	Present output X=1
A	A	E	1	0
B	C	F	0	0
C	B	H	0	0
D	E	F	0	0
E	D	A	0	1
F	B	F	1	1
G	D	H	0	1
H	H	G	1	0



X → 1st pass X's
X → 2nd pass X's

A ≡ H
E ≡ G
} Equivalent states

New state table:

(No G,H states; H replaced with A in Next State, G gone)

Present State	Next State X=0	Next State X=1	Present Output X=0	Present Output X=1
A	A	E	1	0
B	C	F	0	0
C	B	(A)	0	0
D	E	F	0	0
E	D	A	0	1
F	B	F	1	1

Ex: Suppose the FSM specified by the original state table is initially in State B or state G. Find a 3 cycle sequence of inputs to distinguish these initial conditions by the corresponding output sequence...

Time	Input (x)	Circuit 1 State	Circuit 1 Output	Circuit 2 State	Circuit 2 Output
T_0	1	B	0	G(E)	1 ← Distinguished in first cycle!
T_1	1	F	1	H(A)	0
T_2	1	F	1	G(E) ↑ equivalent states	1

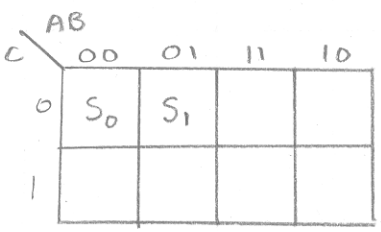
15.8 Guidelines for Efficient State Assignment

When designing an FSM, we want to look for <u>efficient</u> state assignments.

<u>Ex</u>: We have 5 states; we want to use the minimum number of flip-flops (3). How do we assign the states?

One approach: place 1's on the flip-flop input maps in adjacent boxes to help minimize terms.

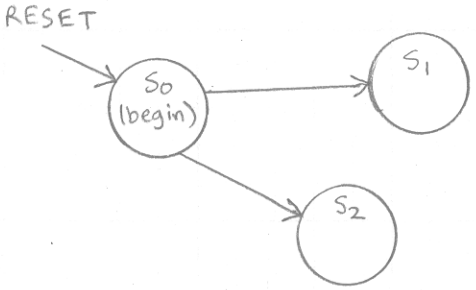
Plot next states in a Karnaugh Map to help make <u>adjacent</u> state assignments. <u>Adjacent states</u> differ in their assignments by exactly <u>one</u> bit.



State	Flip-Flops
S_0 →	ABC = 000
S_1 →	ABC = 001

} adjacent assignments

Beneficial to put the "Reset" state at 000 (we can use the clear input of the flip-flops to initialize the FSM).

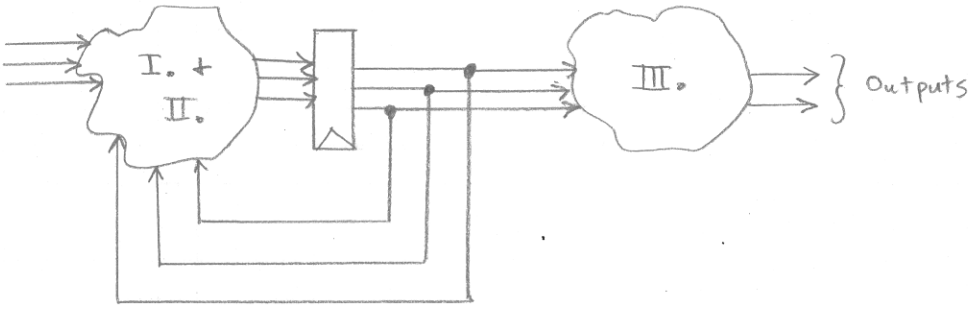


$S_0 \rightarrow 000$

How we assign states will affect the complexity of combinational logic. Some guidelines ...

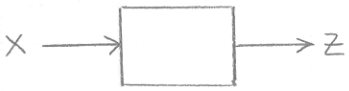
Guideline I. Different present states
Same input ⇒ Same next state } Try to group together.

Guideline II. Different next states of one present state } Try to group together.



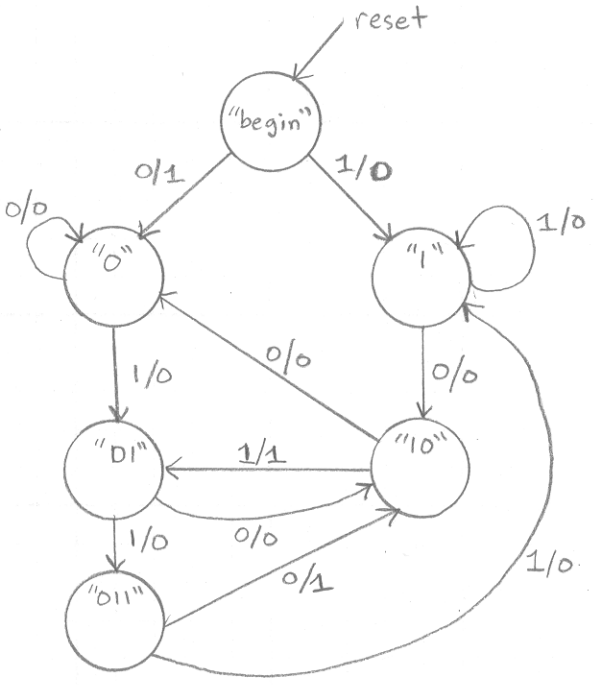
Guideline III. Different present states
Same outputs for all input combinations } Try to group together

Ex: Putting it all together : a sequence detector, Mealy implementation



Z = 1 iff X = ... 0110 or X = ... 101

Example operation → X = 0 1 0 1 1 0 1
Z = 0 0 0 1 0 1 1



State Table:

	Present State	Next State X=0	Next State X=1	Output Z X=0	Output Z X=1
S_0	"begin"	S_1	S_4	0	0
S_1	"0"	S_1	S_2	0	0
S_2	"01"	S_5	S_3	0	0
S_3	"011"	S_5	S_4	1	0
S_4	"1"	S_5	S_4	0	0
S_5	"10"	S_1	S_2	0	1

Determine number of flip-flops and encoding of states:
Minimum binary encoding, 6 states → 3 FFs

Use guidelines for state encoding:

Guideline I: Group present states with same next state for a given input.

(S_0, S_1, S_5) → next state S_1 for $X=0$
(S_2, S_3, S_4) → next state S_5 for $X=0$

(S_0, S_3, S_4) → next state S_4 for $X=1$

(S_1, S_5) → next state S_2 for $X=1$

Guideline II: Next states of present state grouped together

Next states of $S_0 = (S_1, S_4)$

Next states of $S_1, S_5 = (S_1, S_2) \times 2$

Next states of $S_2 = (S_5, S_3)$

Next states of $S_3, S_4 = (S_5, S_4) \times 2$

Guideline III: Group present states with same output for a given input

(S_0, S_1, S_2, S_4) → $Z=0$ always

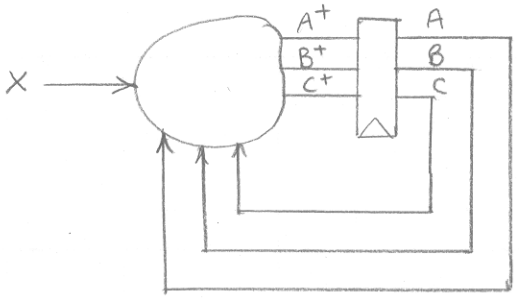
$(S_0, S_1, S_2, S_4, S_5)$ → $Z=0$ for $X=0$

$(S_0, S_1, S_2, S_3, S_4)$ → $Z=0$ for $X=1$

Make state mapping using Guideline I. and assigning $S_0 = 000$. (S_0 ↑ Reset state)
Flip-flop outputs labeled A, B, C

C \ AB	00	01	11	10
0	S_0		S_1	S_5
1	S_3		S_2	S_4

Now design combinational logic for next state logic:



	Present State A B C	Next State X=0 A+ B+ C+	Next State X=1 A+ B+ C+	Output Z X=0	Output Z X=1
S_0	0 0 0	1 1 0	1 0 1	0	0
S_3	0 0 1	1 0 0	1 0 1	1	0
X	0 1 0	X X X	X X X	X	X
X	0 1 1	X X X	X X X	X	X
S_5	1 0 0	1 1 0	1 1 1	0	1
S_4	1 0 1	1 0 0	1 0 1	0	0
S_1	1 1 0	1 1 0	1 1 1	0	0
S_2	1 1 1	1 0 0	0 0 1	0	0

- Design A^+ Logic:

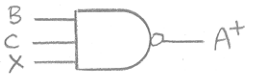
CX \ AB	00	01	11	10
00	1	X	1	1
01	1	X	1	1
11	1	X	0	1
10	1	X	1	1

Use minterms and simplify with algebra:

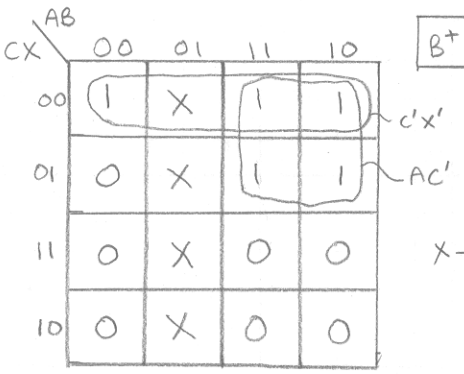
$A^{+\prime} = BCX$

$A^+ = (BCX)' = B' + C' + X'$

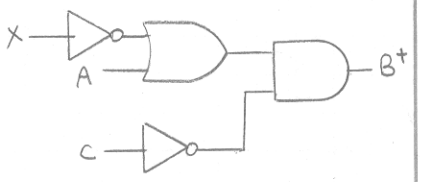
$\boxed{A^+ = B' + C' + X'}$



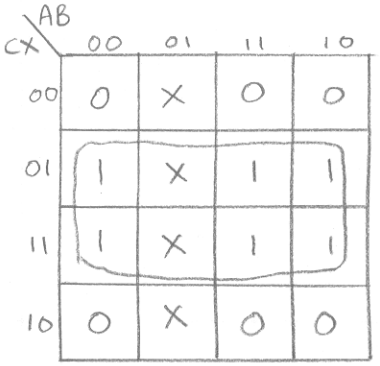
• Design B^+ Logic:



$B^+ = c'x' + Ac' = c'(x'+A)$



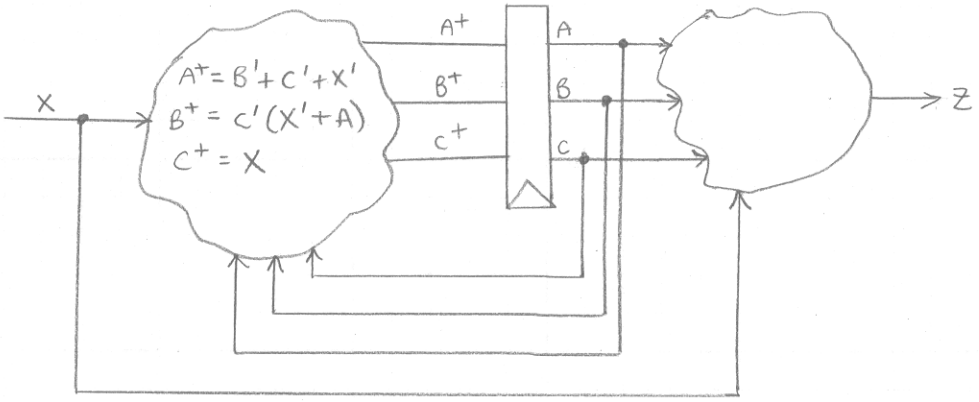
• Design C^+ Logic:

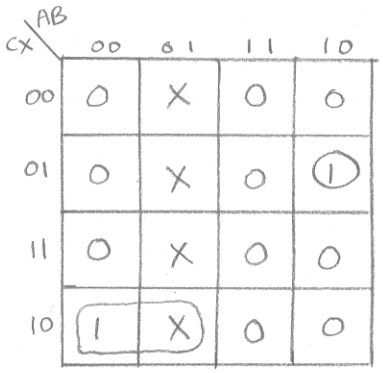


$C^+ = X$

X —— C^+

• Design (Mealy) Output combinational Logic for Z:





$Z = AB'C'X + A'CX$