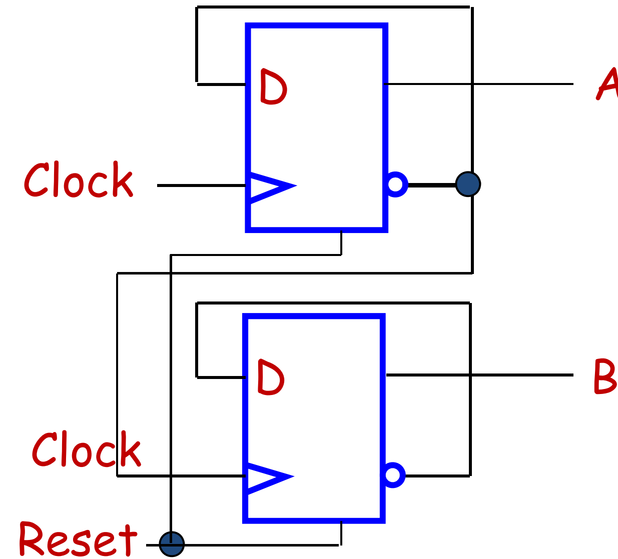
Counters

- ✓ Counters are sequential circuits which "count" through a specific state sequence.
  - They can count up, count down, or count through other fixed sequences.
- ✓ Two distinct types are in common usage:
  - Ripple Counters
    - Clock connected to the flip-flop clock input on the LSB bit flip-flop
    - For all other bits, a flip-flop output is connected to the clock input, thus circuit is not truly synchronous!
    - Output change is delayed more for each bit toward the MSB.
    - Resurgent because of low power consumption
  - Synchronous Counters
    - Clock is directly connected to the flip-flop clock inputs
    - Logic is used to implement the desired state sequencing

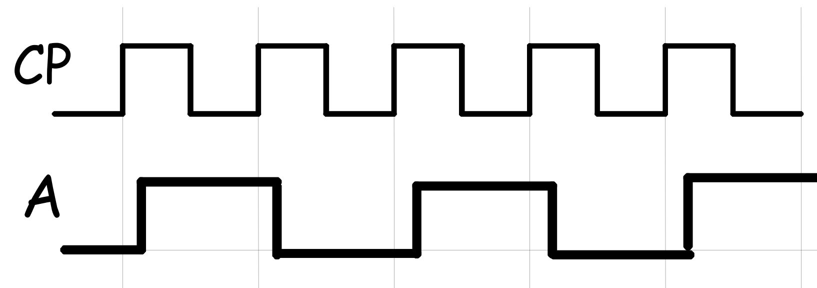
Ripple Counter

✓ How does it work?

- When there is a positive edge on the clock input of A, A complements
- The clock input for flip-flop B is the complemented output of flip-flop A

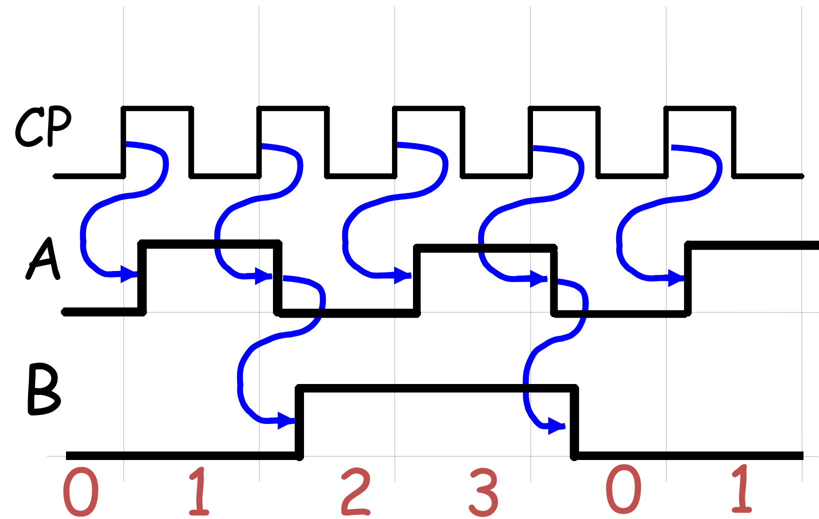


- When flip A changes from 1 to 0, there is a positive edge on the clock input of B causing B to complement



Ripple Counter (continued)

- ✓ The arrows show the cause-effect relation-ship from the prior slide
- ✓ The corresponding sequence of states ⇨



(B,A) = (0,0), (0,1), (1,0), (1,1), (0,0), (0,1), …

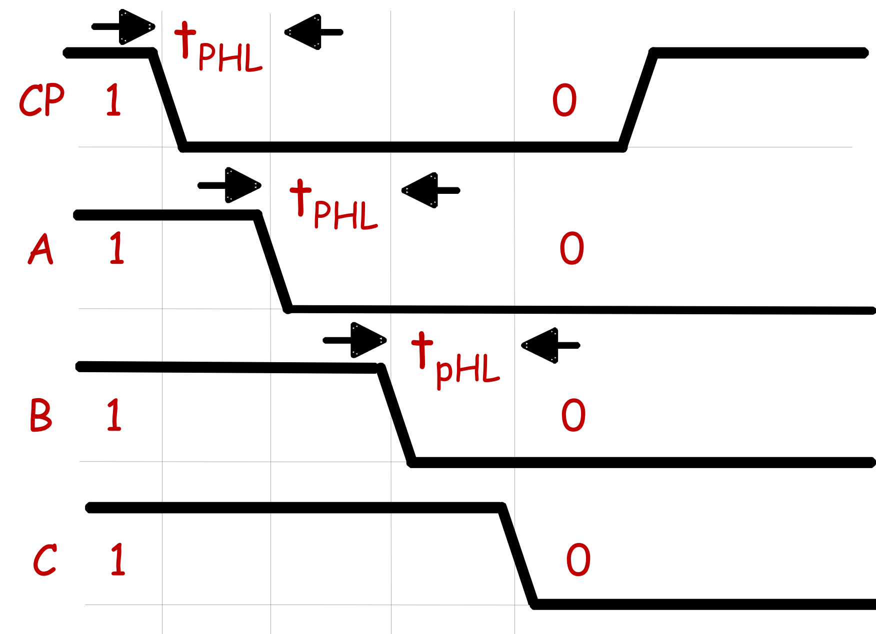
- ✓ Each additional bit, C, D, …behaves like bit B, changing half as frequently as the bit before it.
- ✓ For 3 bits: (C,B,A) = (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1), (0,0,0), …

Ripple Counter (continued)

- ✓ These circuits are called ripple counters because each edge sensitive transition (positive in the example) causes a change in the next flip-flop's **state**.
- ✓ The changes ripple upward through the chain of flip-flops, i. e., each transition occurs after a clock-to-output **delay** from the stage before.

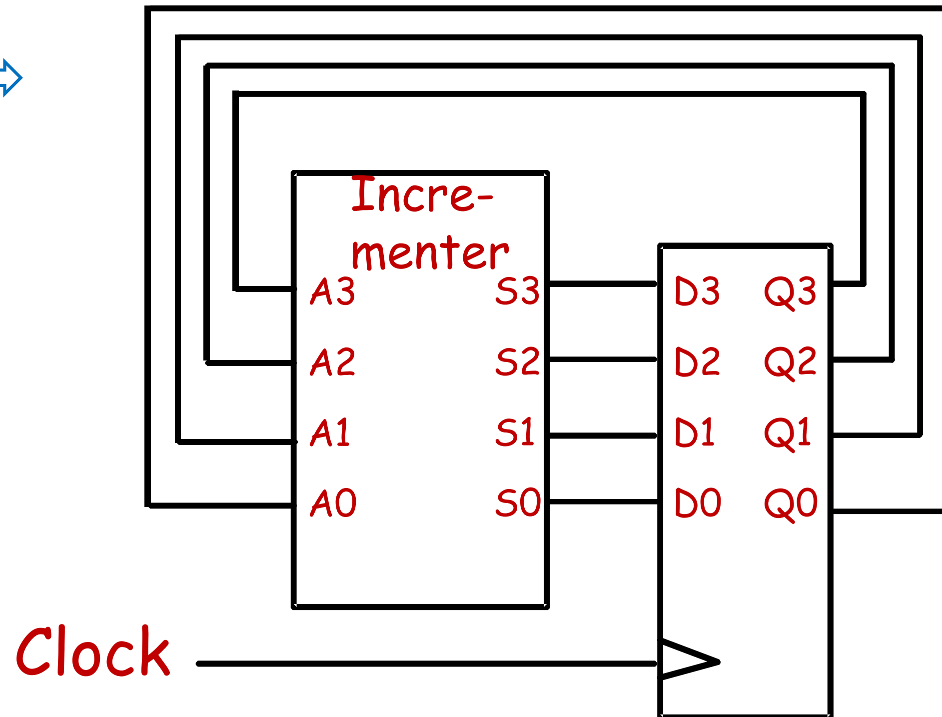
Ripple Counter (continued)

- Starting with $C = B = A = 1$, equivalent to (C,B,A) = 7 base 10, the next clock increments the count to (C,B,A) = 0 base 10. In fine timing detail:
  - The clock to output delay t_{PHL} causes an increasing delay from clock edge for each stage transition.
  - Thus, the count "ripples" from least to most significant bit.
  - For n bits, total worst case delay is $n\ t_{PHL}$.



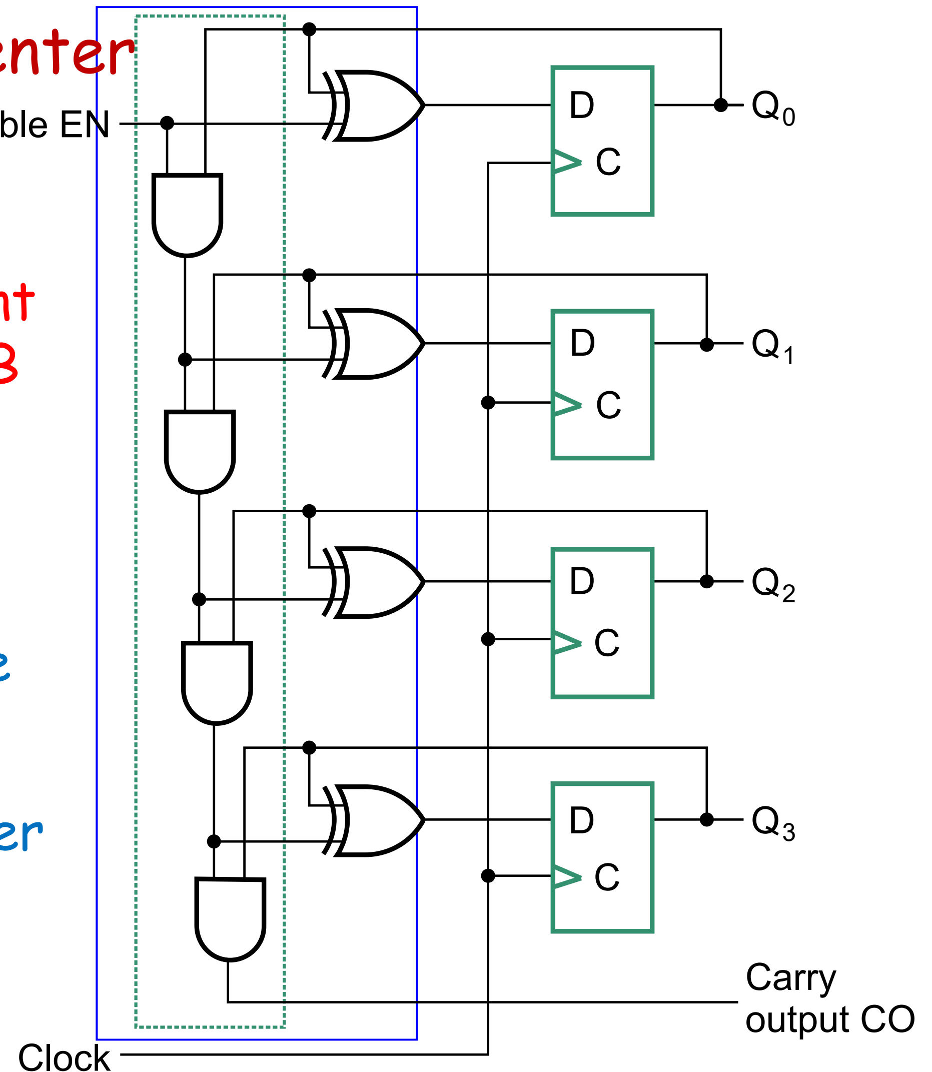
Synchronous Counters

- To eliminate the "ripple" effects, use a common clock for each flip-flop and a combinational circuit to generate the next state.
- For an up-counter, use an incrementer ⇨



Synchronous Counters (continued)

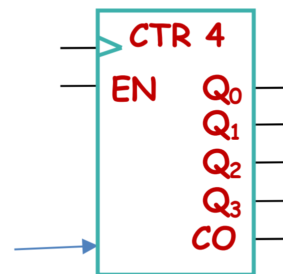
- ✓ Internal details ⇨ Incrementer
- ✓ Internal Logic
  - XOR complements each bit
  - AND chain causes complement of a bit if all bits toward LSB from it equal 1
- ✓ Count Enable
  - Forces all outputs of AND chain to 0 to "hold" the state
- ✓ Carry Out
  - Added as part of incrementer
  - Connect to Count Enable of additional 4-bit counters to form larger counters



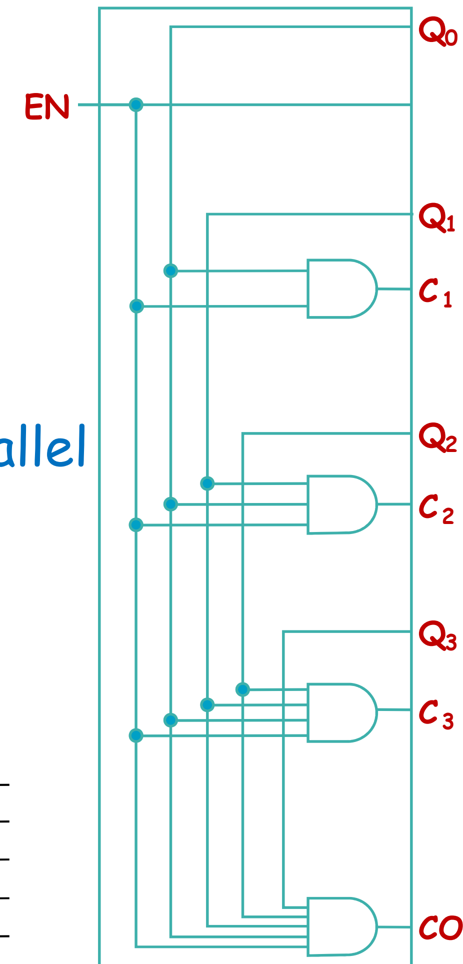
(a) Logic Diagram-Serial Gating

Synchronous Counters (continued)

- ✓ Carry chain
  - series of AND gates through which the carry "ripples"
  - Yields long path delays
  - Called *serial gating*
- ✓ Replace AND carry chain with ANDs ⇨ in parallel
  - Reduces path delays
  - Called *parallel gating*
  - Like carry lookahead
  - Lookahead can be used on COs and ENs to prevent long paths in large counters
- ✓ Symbol for Synchronous Counter



Symbol



Logic Diagram-Parallel Gating

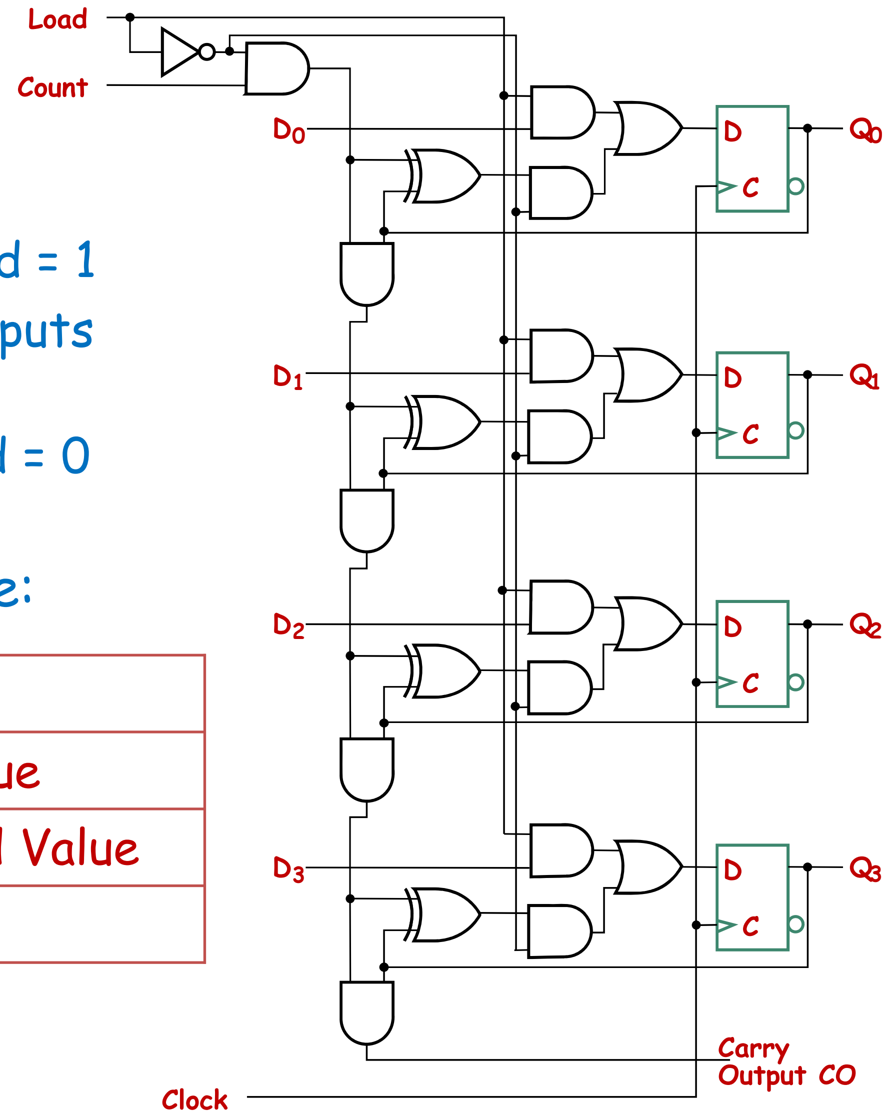
Other Counters

- Counters:
  - Down Counter - counts downward instead of upward
  - Up-Down Counter - counts up or down depending on value a control input such as Up/$\overline{\text{Down}}$
  - Parallel Load Counter - Has parallel load of values available depending on control input such as Load
- Divide-by-n (Modulo n) Counter
  - Count is remainder of division by *n; n* may not be a power of 2
  - Count is arbitrary sequence of *n* states specifically designed state-by-state
  - Includes modulo 10 which is the BCD counter

Counter with Parallel Load

- ✓ Add path for input data
  - enabled for Load = 1
- ✓ Add logic to:
  - disable count logic for Load = 1
  - disable feedback from outputs for Load = 1
  - enable count logic for Load = 0 and Count = 1
- ✓ The resulting function table:

Load	Count	Action
0	0	Hold Stored Value
0	1	Count Up Stored Value
1	X	Load D



Design Example: Synchronous BCD

- ✓ Use the sequential logic model to design a synchronous BCD counter with D flip-flops
- ✓ Input combinations 1010 through 1111 are don't cares

Current State				Next State			
Q8	Q4	Q2	Q1	Q8	Q4	Q2	Q1
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

Synchronous BCD (continued)

- ✓ Use K-Maps to two-level optimize the next state equations:

$$D1 = \overline{Q1}$$
$$D2 = \overline{Q8}\,\overline{Q2}Q1 + Q2\overline{Q1}$$
$$D4 = \overline{Q4}Q2Q1 + Q4\overline{Q2} + Q4\overline{Q1}$$
$$D8 = Q8\overline{Q1} + Q4Q2Q1$$

- ✓ The logic diagram can be draw from these equations
  - An asynchronous or synchronous reset should be added
- ✓ What happens if the counter is perturbed by a power disturbance or other interference and it enters a state other than 0000 through 1001?

Synchronous BCD (continued)

- ✓ Find the actual values of the six next states for the don't care combinations from the equations
- ✓ Find the overall state diagram to assess behavior for the don't care states (states in decimal)

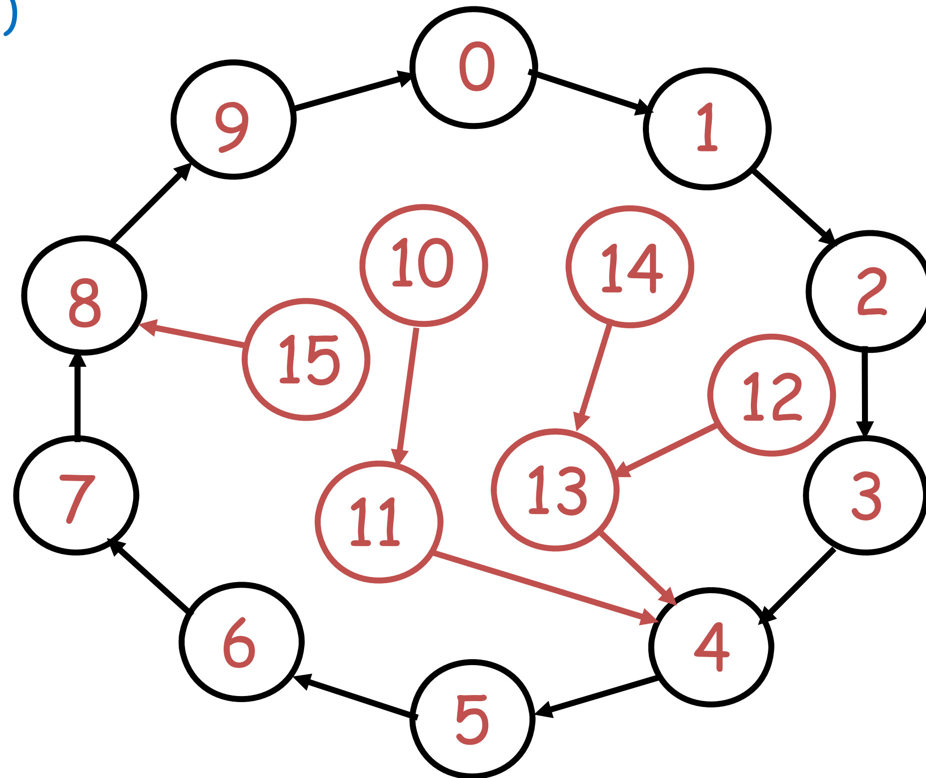
$D1 = \overline{Q1}$

$D2 = \overline{Q8}\,\overline{Q2}Q1 + Q2\overline{Q1}$

$D4 = \overline{Q4}Q2Q1 + Q4\overline{Q2} + Q4\overline{Q1}$

$D8 = Q8\overline{Q1} + Q4Q2Q1$

Present State	Next State
Q8 Q4 Q2 Q1	Q8 Q4 Q2 Q1
1 0 1 0	1 0 1 1
1 0 1 1	0 1 0 0
1 1 0 0	1 1 0 1
1 1 0 1	0 1 0 0
1 1 1 0	1 1 0 1
1 1 1 1	1 0 0 0



Synchronous BCD (continued)

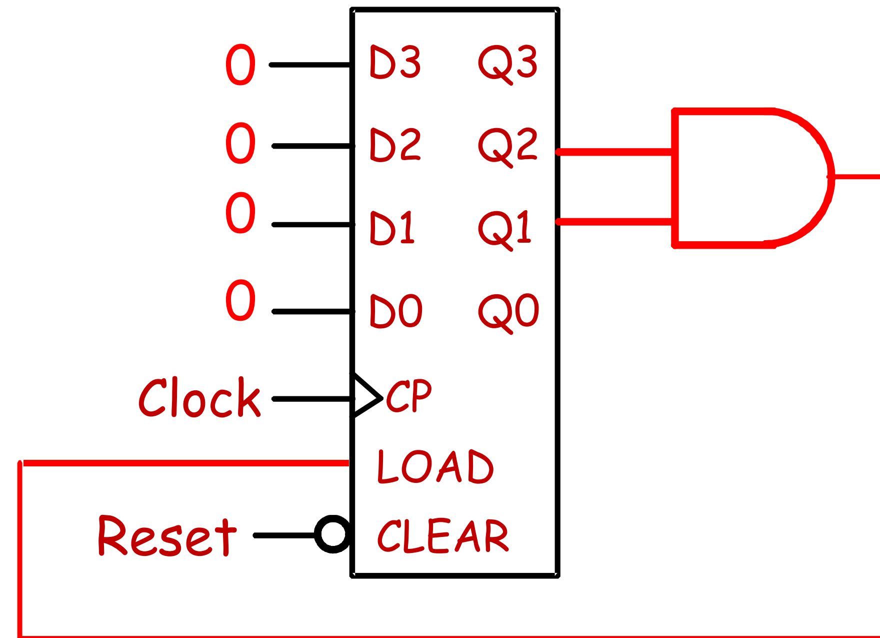
- ✓ For the BCD counter design, if an invalid state is entered, return to a valid state occurs within two clock cycles
- ✓ Is this adequate? If not:
  - Is a signal needed that indicates that an invalid state has been entered? What is the equation for such a signal?
  - Does the design need to be modified to return from an invalid state to a valid state in one clock cycle?
  - Does the design need to be modified to return from a invalid state to a specific state (such as 0)?
- ✓ The action to be taken depends on:
  - the application of the circuit
  - design group policy

Counting Modulo N

- The following techniques use an *n*-bit binary counter with asynchronous or synchronous clear and/or parallel load:
  - Detect a terminal count of N in a Modulo-N count sequence to asynchronously Clear the count to 0 or asynchronously Load in value 0
  - Detect a terminal count of N - 1 in a Modulo-N count sequence to Clear the count synchronously to 0
  - Detect a terminal count of N - 1 in a Modulo-N count sequence to synchronously Load in value 0
  - Detect a terminal count and use Load to preset a count of the terminal count value minus (N - 1)
- Alternatively, custom design a modulo N counter as done for BCD

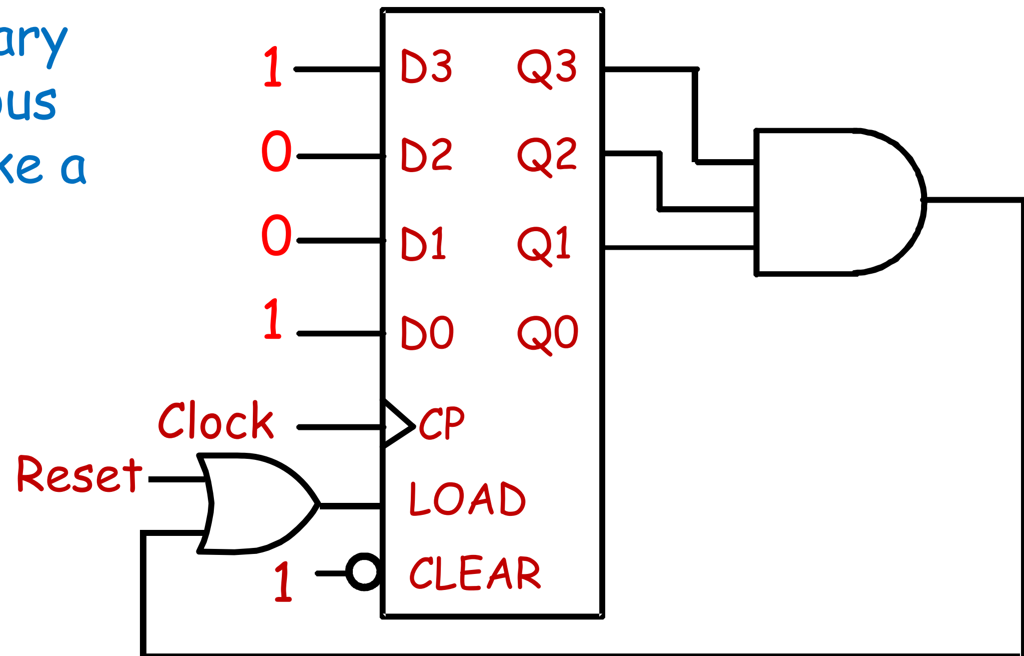
Counting Modulo 7: Synchronously Load on Terminal Count of 6

- A synchronous 4-bit binary counter with a synchronous load and an asynchronous clear is used to make a Modulo 7 counter
- Use the Load feature to detect the count "6" and load in "zero". This gives a count of 0, 1, 2, 3, 4, 5, 6, 0, 1, 2, 3, 4, 5, 6, 0, ...
- Using don't cares for states above 0110



Counting Modulo 6: Synchronously Preset 9 on Reset and Load 9 on Terminal Count 14

- A synchronous, 4-bit binary counter with a synchronous Load is to be used to make a Modulo 6 counter.



- Use the Load feature to preset the count to 9 on Reset and detection of count 14.

- This gives a count of 9, 10, 11, 12, 13, 14, 9, 10, 11, 12, 13, 14, 9, …
- If the terminal count is 15 detection is usually built in as Carry Out (CO)

Example 4: Design a modulo-8 up-counter which counts in the way specified below, use J-K FF

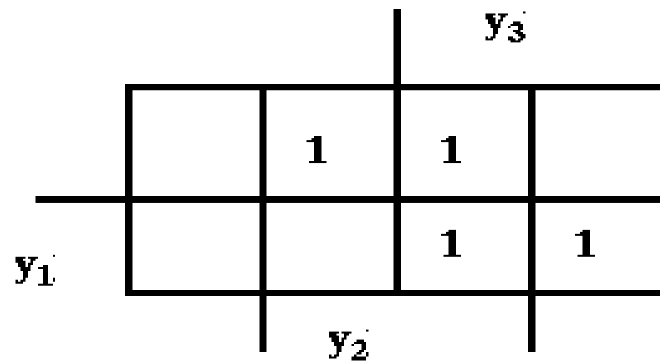
Decimal	Gray
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

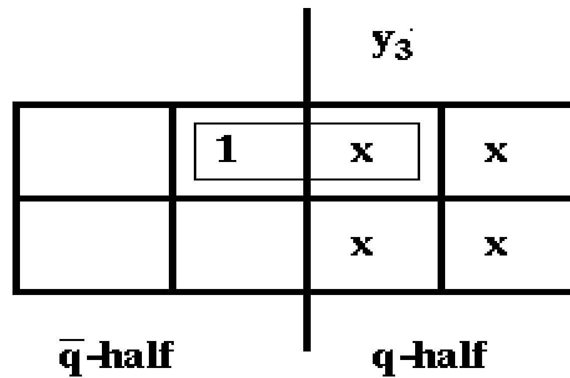
Example 4: TRUTH TABLE

present state			next state		
y_3	y_2	y_1	y_{3+}	y_{2+}	y_{1+}
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

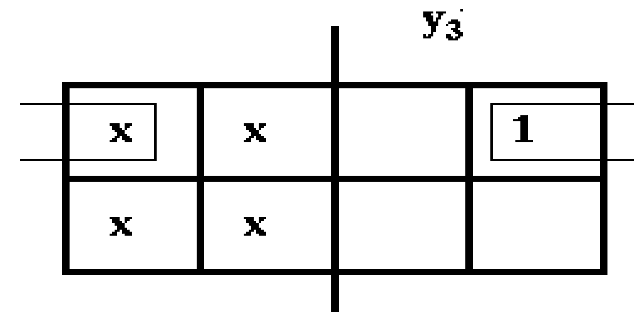
Example 4: Gray code counter

Y3





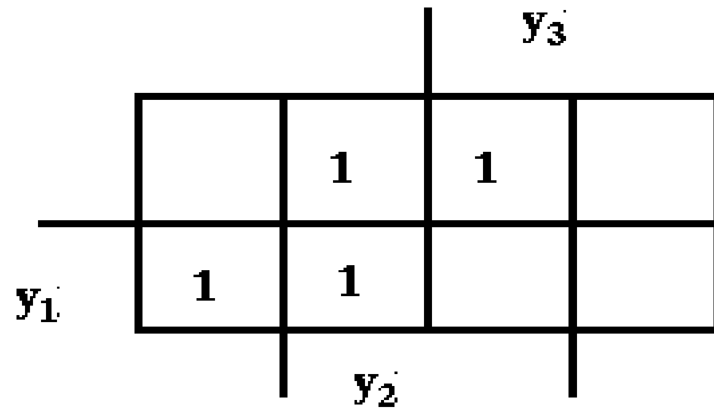
$J_{y3} = y_2\bar{y}_1$

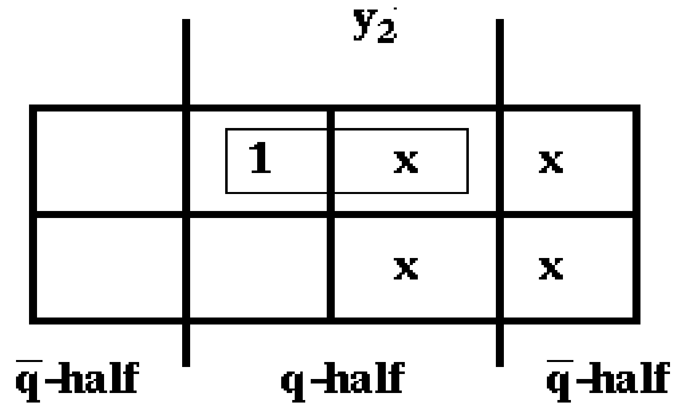


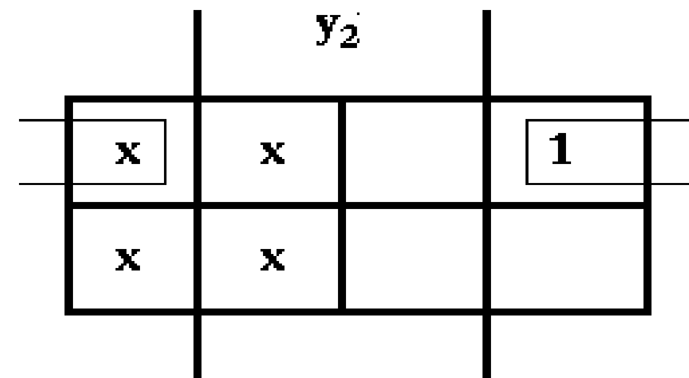
$K_{y3} = \bar{y}_2 y_1$

Example 4: Gray code counter

Y2





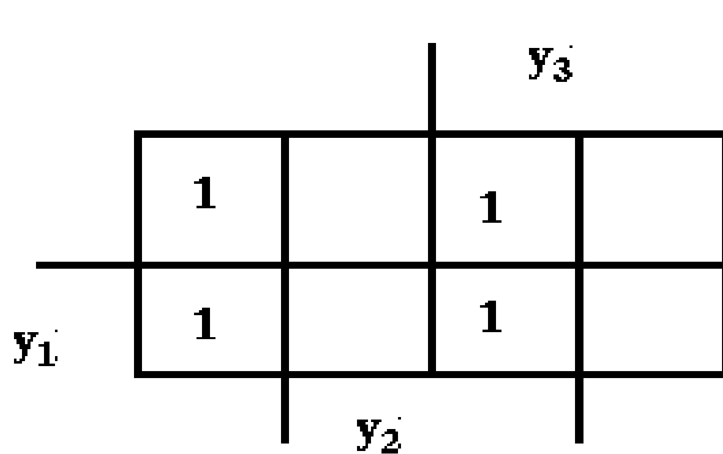


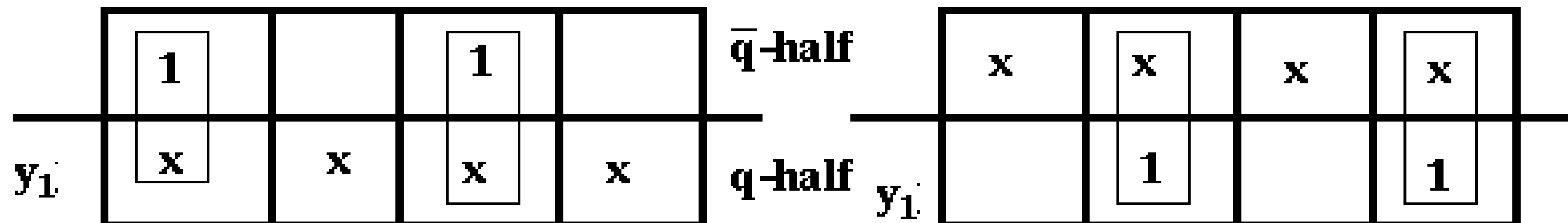
$J_{y2} = \bar{y}_3 y_1$

$K_{y2} = y_3 y_1$

Example 4: Gray code counter

Y1





$J_{y1} = \bar{y}_3\bar{y}_2 + y_3 y_2$

$K_{y1} = \bar{y}_2 y_3 + y_2 \bar{y}_3$

Objective: Eliminate redundant states

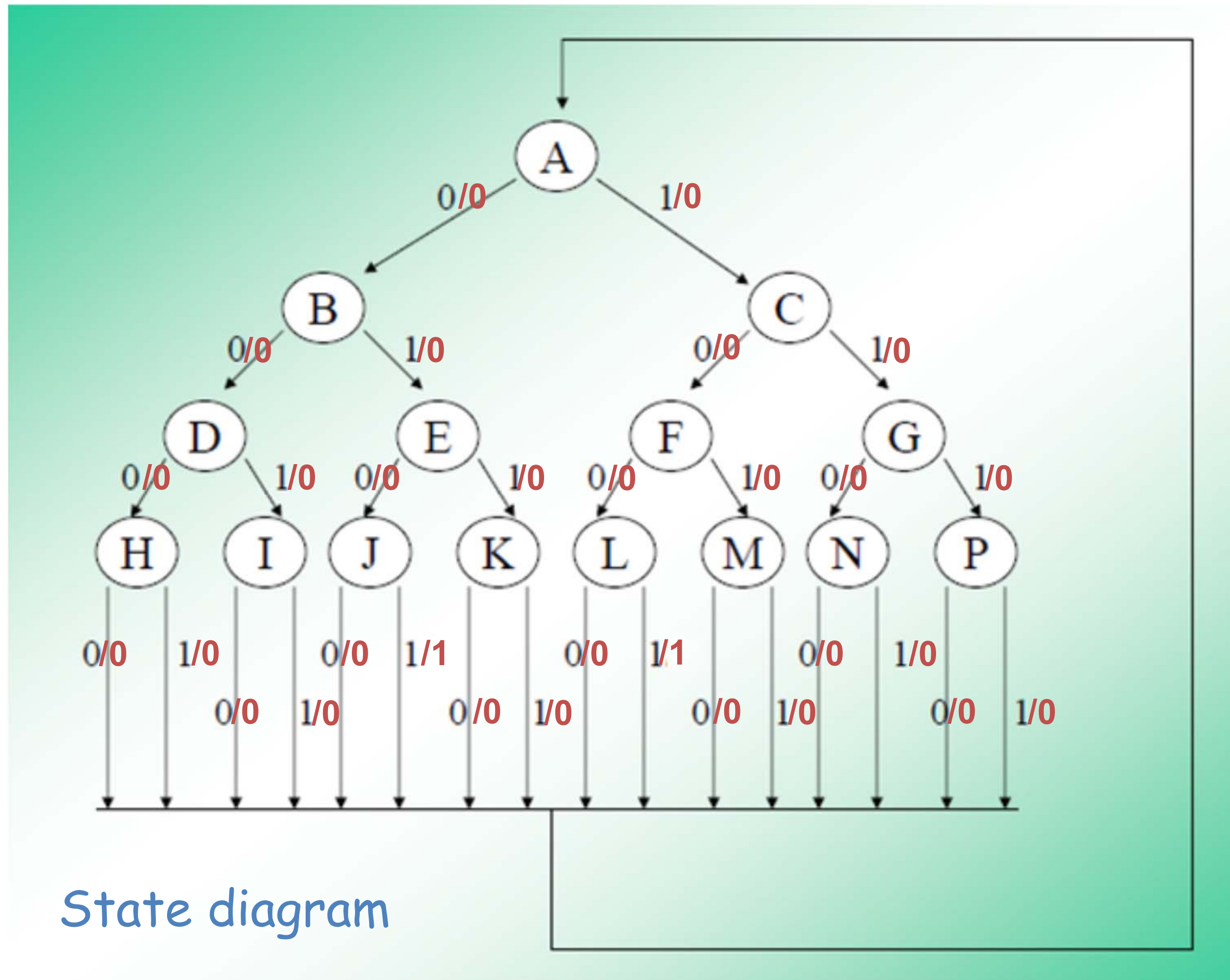
- ✓ Reduce the number of states in the state table to the minimum.
  - Remove redundant states
  - Use don't cares effectively
- ✓ Reduction to the minimum number of states reduces
  - The number of F/Fs needed
  - Reduces the number of next states that has to be generated ⇨ Reduced logic.

An example circuit

- A sequential circuit has one input X and one output Z.
- The circuit looks at the groups of four consecutive inputs and sets Z=1 if the input sequence 0101 or 1001 occurs.
- The circuit returns to the reset state after four inputs.
- Design the Mealy machine.

Elimination of Redundant States

- ✓ When first setting up the state table, we will not be overly concerned with inclusion of extra states, and when the table is complete, we will eliminate any redundant states.



State diagram

State table

- ✓ Set up a table for all the possible input combinations

- ✓ For the two sequences when the last bit is a 1 return to reset with Z=1.

Input Sequence	Present State	Next State X = 0	Next State X = 1	Present Output X = 0	Present Output X = 1
reset	A	B	C	0	0
0	B	D	E	0	0
1	C	F	G	0	0
00	D	H	I	0	0
01	E	J	K	0	0
10	F	L	M	0	0
11	G	N	P	0	0
000	H	A	A	0	0
001	I	A	A	0	0
010	J	A	A	0	1
011	K	A	A	0	0
100	L	A	A	0	1
101	M	A	A	0	0
110	N	A	A	0	0
111	P	A	A	0	0

Note on state table generation

- ✓ When generated by looking at all combinations of inputs the state table is far from minimal.
- ✓ First step is to remove redundant states.
  - There are states that you cannot tell apart
    - Such as H and I – both have A with Z=0 as output.
    - State H is equivalent to State I and state I can be removed from the table.
    - Examining table shows states K, M, N and P are also the same – they can be deleted.
    - States J and L are also equivalent.

Input Sequence	Present State	Next State X = 0	Next State X = 1	Present Output X = 0	Present Output X = 1
reset	A	B	C	0	0
0	B	D	E	0	0
1	C	F	G	0	0
00	D	H	I	0	0
01	E	J	K	0	0
10	F	L	M	0	0
11	G	N	P	0	0
000	H	A	A	0	0
001	I	A	A	0	0
010	J	A	A	0	1
011	K	A	A	0	0
100	L	A	A	0	1
101	M	A	A	0	0
110	N	A	A	0	0
111	P	A	A	0	0

Reduction continued

- ✓ Having made these reductions move up to the D E F G section where the next state entries have been changed.
- ✓ Note that State D and State G are equivalent.
- ✓ State E is equivalent to F.
- ✓ The result in a reduced state table.

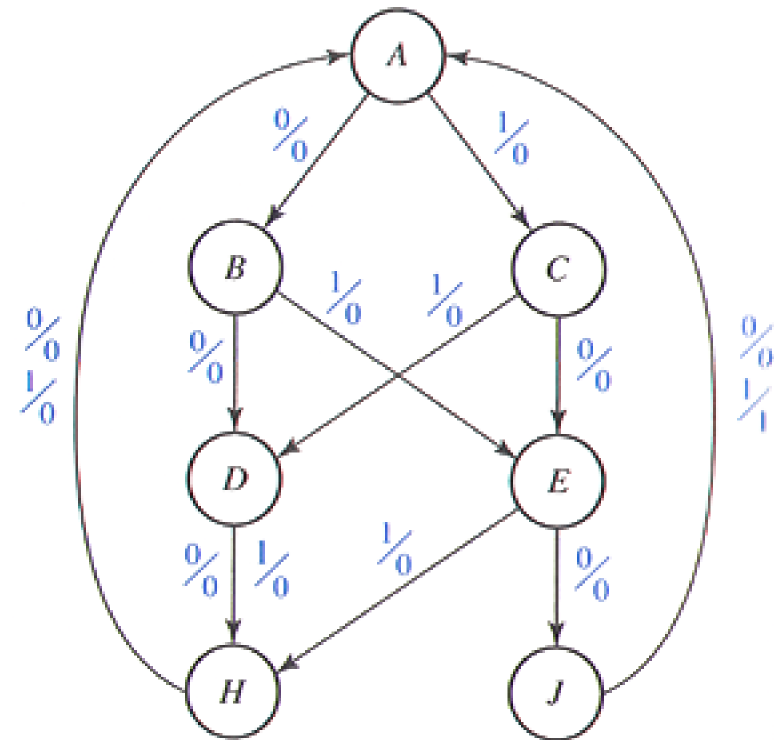
Present State	Next State $X=0$	Next State $X=1$	Present Output $X=0$	Present Output $X=1$
A	B	C	0	0
B	D	E	0	0
C	~~F~~ E	~~G~~ D	0	0
D	H	~~I~~ H	0	0
E	J	~~K~~ H	0	0
~~F~~	~~L J~~	~~M H~~	~~0~~	~~0~~
~~G~~	~~N H~~	~~R H~~	~~0~~	~~0~~
H	A	A	0	0
~~I~~	~~A~~	~~A~~	~~0~~	~~0~~
J	A	A	0	1
~~K~~	~~A~~	~~A~~	~~0~~	~~0~~
~~L~~	~~A~~	~~A~~	~~0~~	~~1~~
~~M~~	~~A~~	~~A~~	~~0~~	~~0~~
~~N~~	~~A~~	~~A~~	~~0~~	~~0~~
~~P~~	~~A~~	~~A~~	~~0~~	~~0~~

The result

- ✓ Reduced state table and graph

Present State	Next State X = 0	Next State X = 1	Output X = 0	Output X = 1
A	B	C	0	0
B	D	E	0	0
C	E	D	0	0
D	H	H	0	0
E	J	H	0	0
H	A	A	0	0
J	A	A	0	1

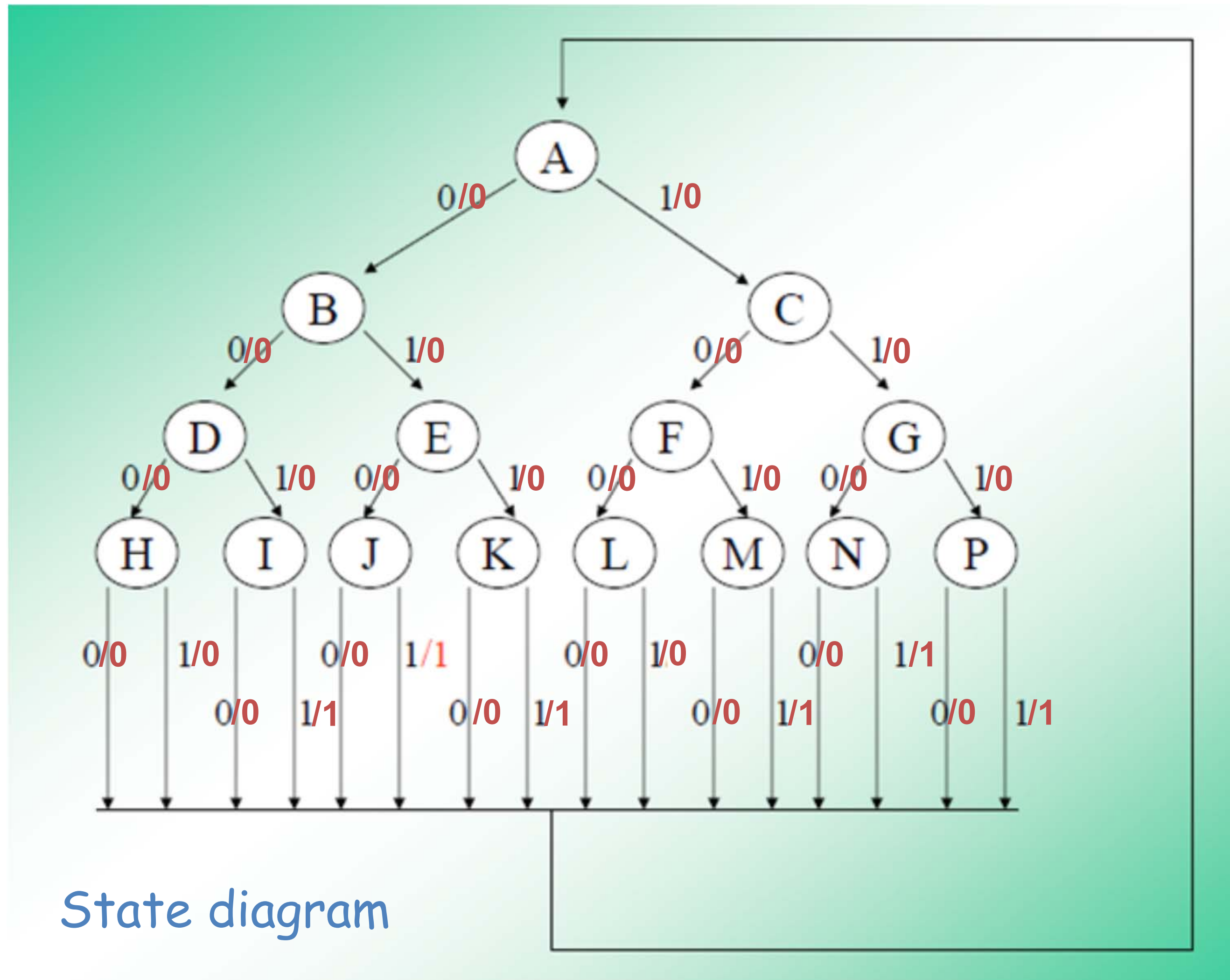
(a)



- ✓ Original – 15 states – reduced 7 states

Elimination of Redundant States

- ✓ Design a binary checker that has in input a sequence of BCD numbers and for every four bits (LSB order) has output 0 if the number is 0≤N≤9 and 1 if 10≤N≤15

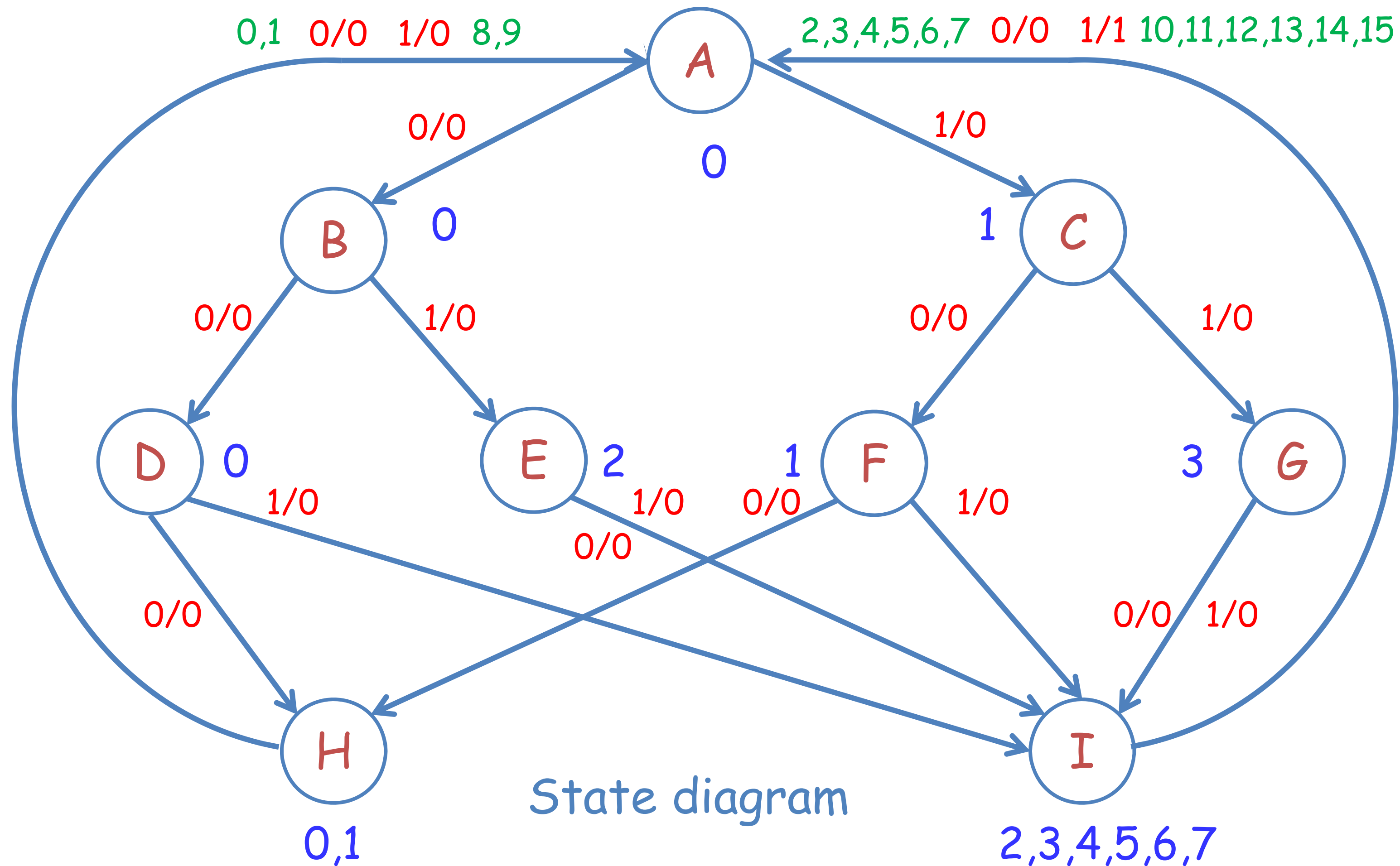


Elimination of Redundant States

- ✓ Design a binary checker that has in input a sequence of BCD numbers and for every four bits (LSB order) has output 0 if the number is 0≤N≤9 and 1 if 10≤N≤15

Input Sequence	Present State	Next State $X = 0$	Next State $X = 1$	Present Output $X = 0$	Present Output $X = 1$
reset	A	B	C	0	0
0	B	D	E	0	0
1	C	F	G	0	0
00	D	H	I	0	0
01	E	J	K	0	0
10	F	L	M	0	0
11	G	N	P	0	0
000	H	A	A	0	0
001	I	A	A	0	1
010	J	A	A	0	1
011	K	A	A	0	1
100	L	A	A	0	0
101	M	A	A	0	1
110	N	A	A	0	1
111	P	A	A	0	1

Elimination of Redundant States



State diagram

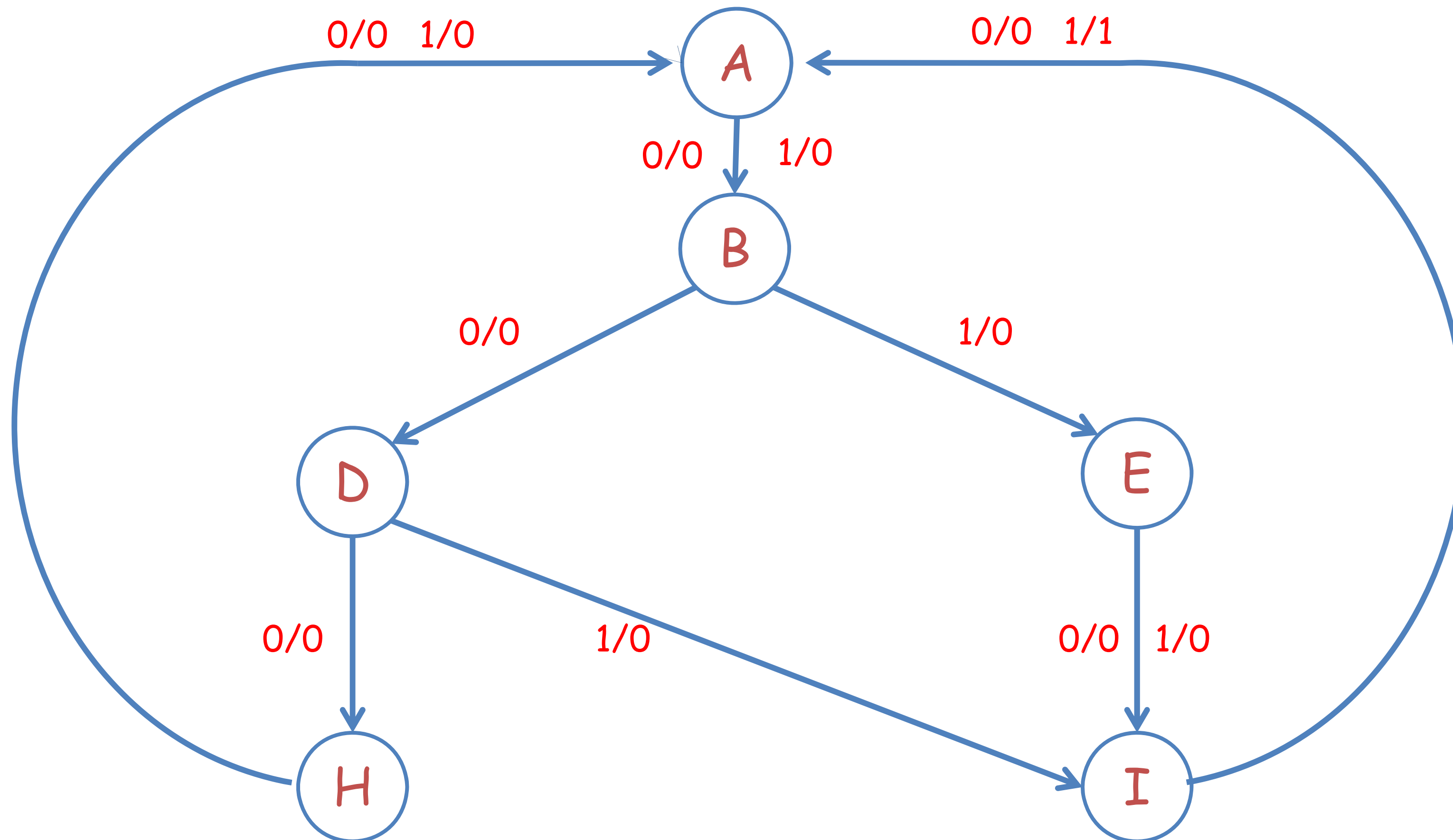
Elimination of Redundant States

	Input Sequence	Present State	Next State $X = 0$	Next State $X = 1$	Present Output $X = 0$	Present Output $X = 1$
0	reset	A	B	C	0	0
0	0	B	D	E	0	0
1	1	C	F	G	0	0
0	00	D	H	I	0	0
2	01	E	I	I	0	0
1	10	F	H	I	0	0
3	11	G	I	I	0	0
0,1	000 100	H	A	A	0	0
2,3,4,5,6,7	001 101 010 110 011 111	I	A	A	0	1

{D,F}, {E,G}, {B,C}

State table

Elimination of Redundant States



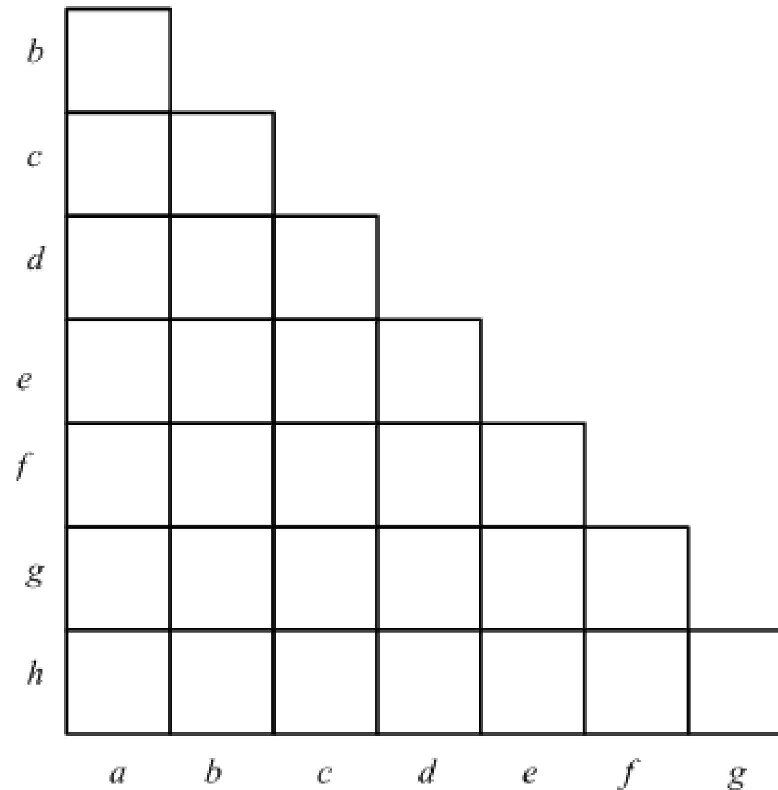
State diagram

Equivalence

- ✓ Two states are equivalent is there is no way of telling them apart through observation of the circuit inputs and outputs.
- ✓ Formal definition:
  - Let N_1 and N_2 be sequential circuits (not necessarily different). Let $\underline{X}$ represent a sequence of inputs of arbitrary length. Then state p in N_1 is equivalent to state q in N_2 iff $\lambda_1(p,\underline{X}) = \lambda_2(q,\underline{X})$ for every possible input sequence $\underline{X}$.
- ✓ The definition is not practical to apply in practice. Theorem:
  - Two states p and q of a sequential circuit are equivalent iff for every single input X, the outputs are the same and the next states are equivalent, that is, $\lambda(p,\underline{X})=\lambda(q,\underline{X})$ and $\delta(p,\underline{X})=\delta(q,\underline{X})$ where $\lambda(p,\underline{X})$ is the output given present state p and input X, and $\delta(p,\underline{X})$ is the next state given the present state p and input X.
- ✓ So the outputs have to be the same and the next states equivalent.

Implication Tables

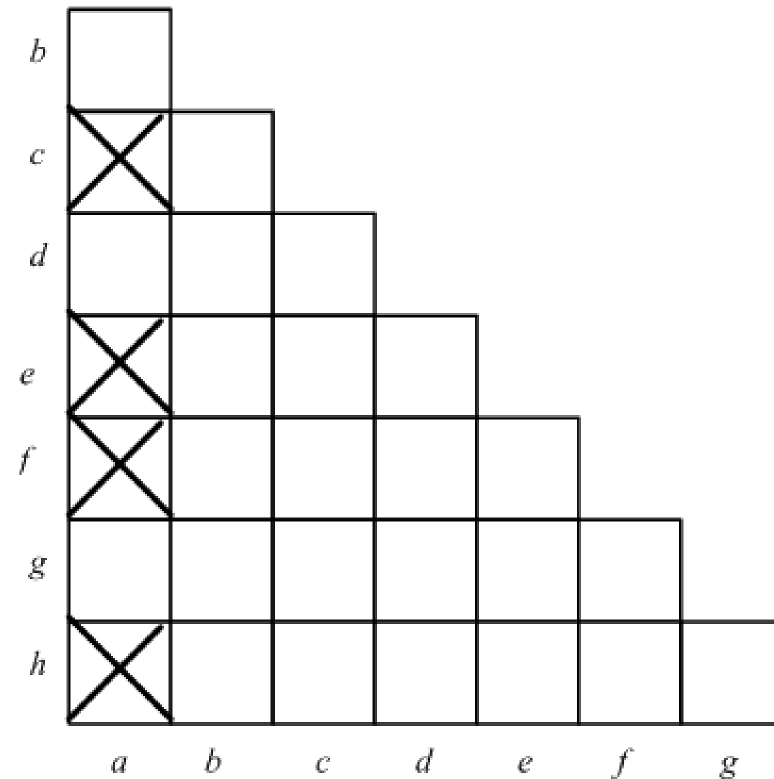
- A procedure for finding all the equivalent states in a state table.
- Use an implication table – a chart that has a square for each pair of states.



Step 1

- ✓ Use a X in the square to eliminate output incompatible states.
- ✓ 1st output of a differes from c, e, f, and h

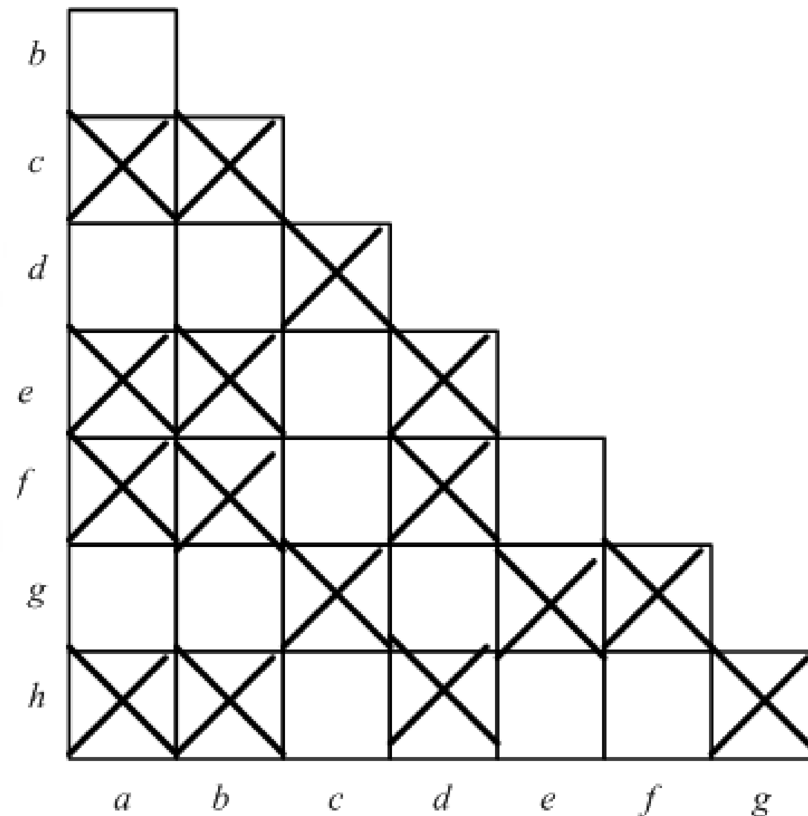
Present State	Next State $X = 0$	1	Present Output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



Step 1 continued

- ✓ Continue to remove output incompatible states

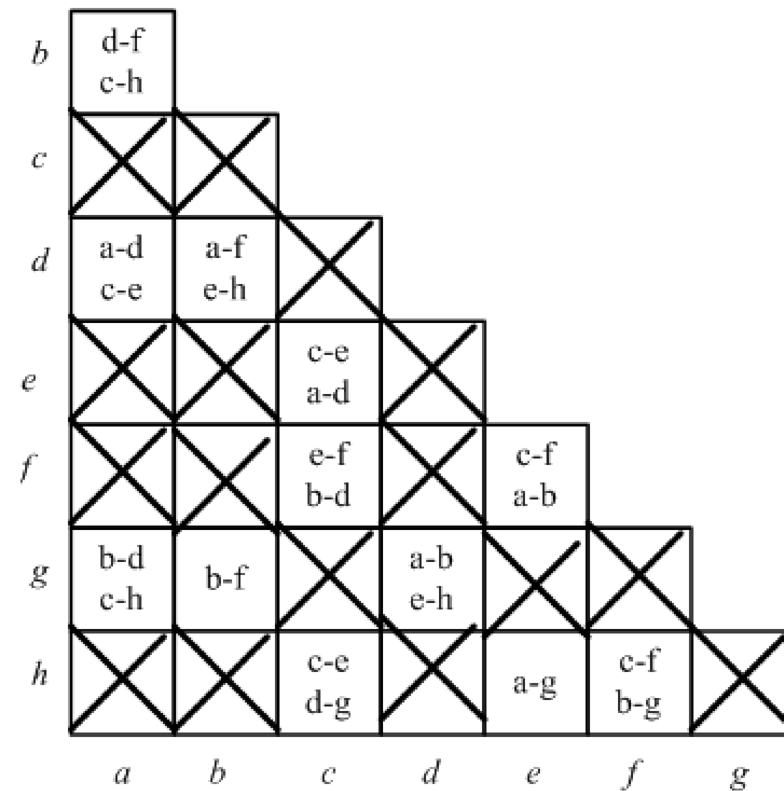
Present State	Next State $X = 0$	1	Present Output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



Now what?

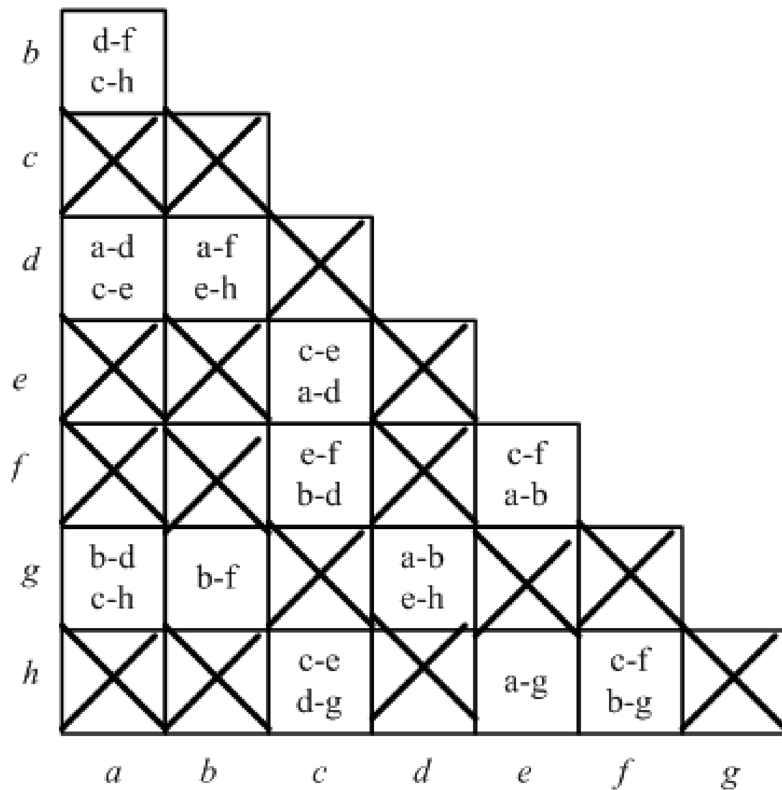
- ✓ Implied pair are now entered into each non X square.
- ✓ Here a≡b iff d≡f and c≡h

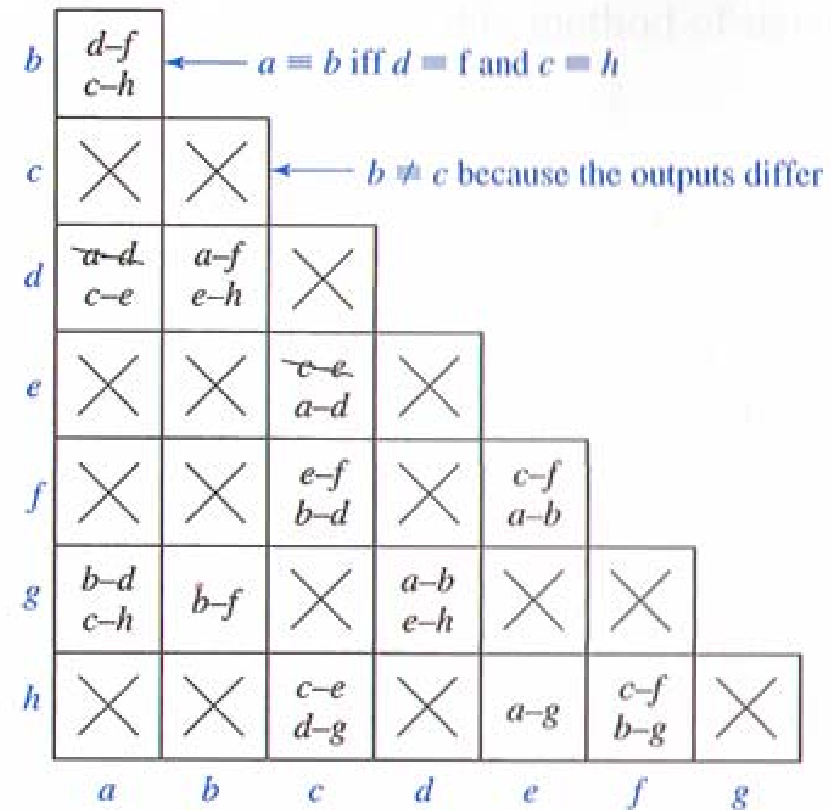
Present State	Next State X = 0	1	Present Output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1



Self redundant pairs

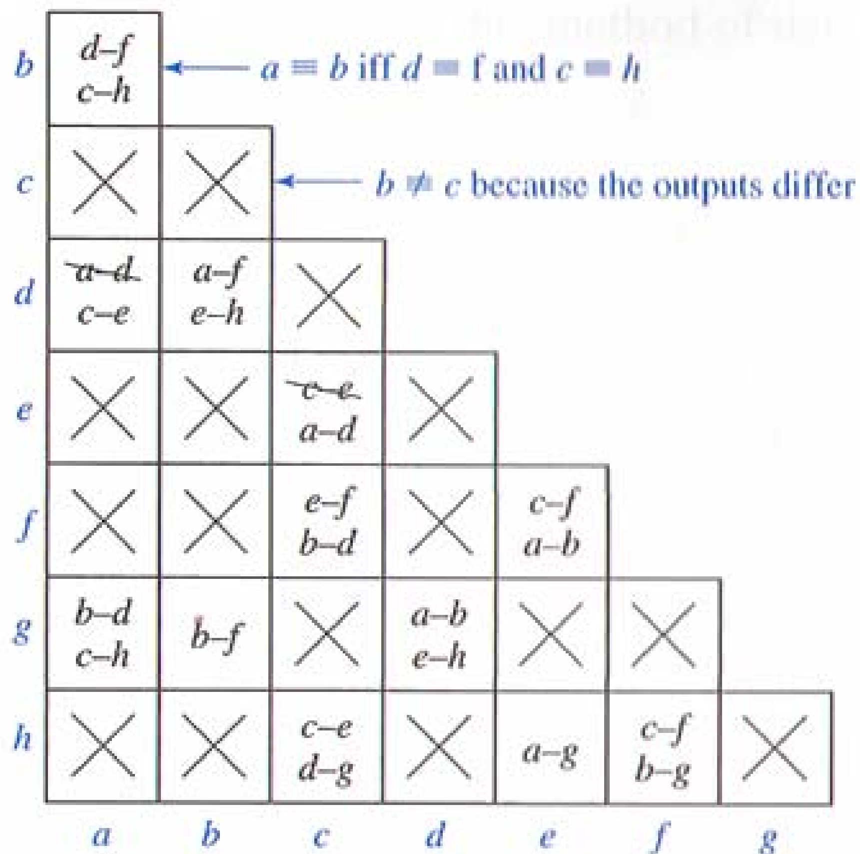
✓ Self redundant pairs are removed, i.e., in square a-d it contains a-d.

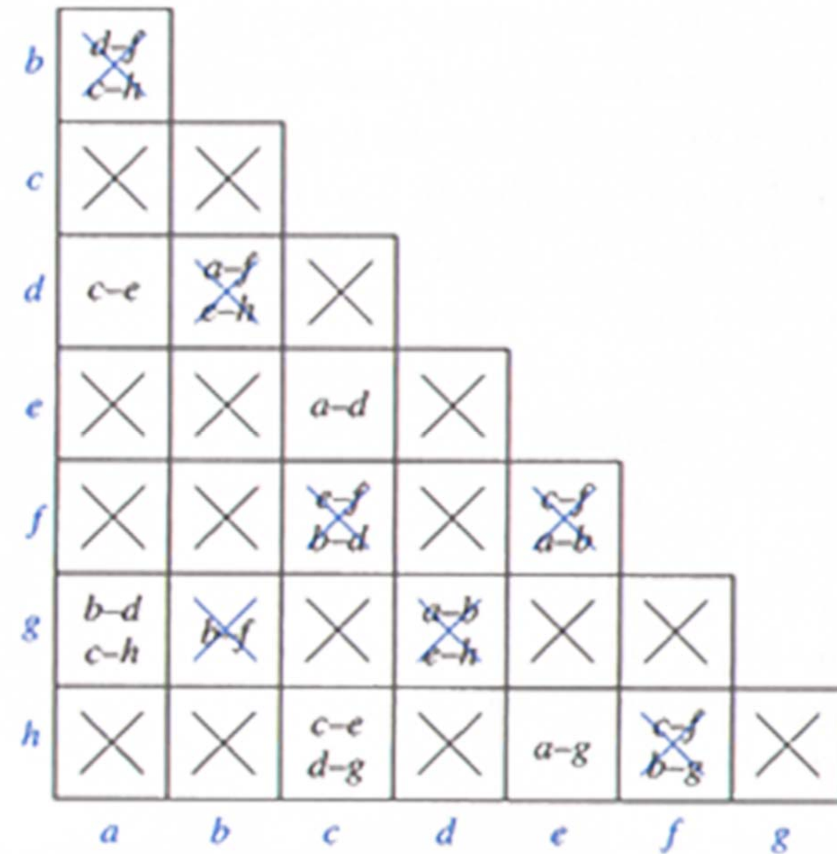




Next pass

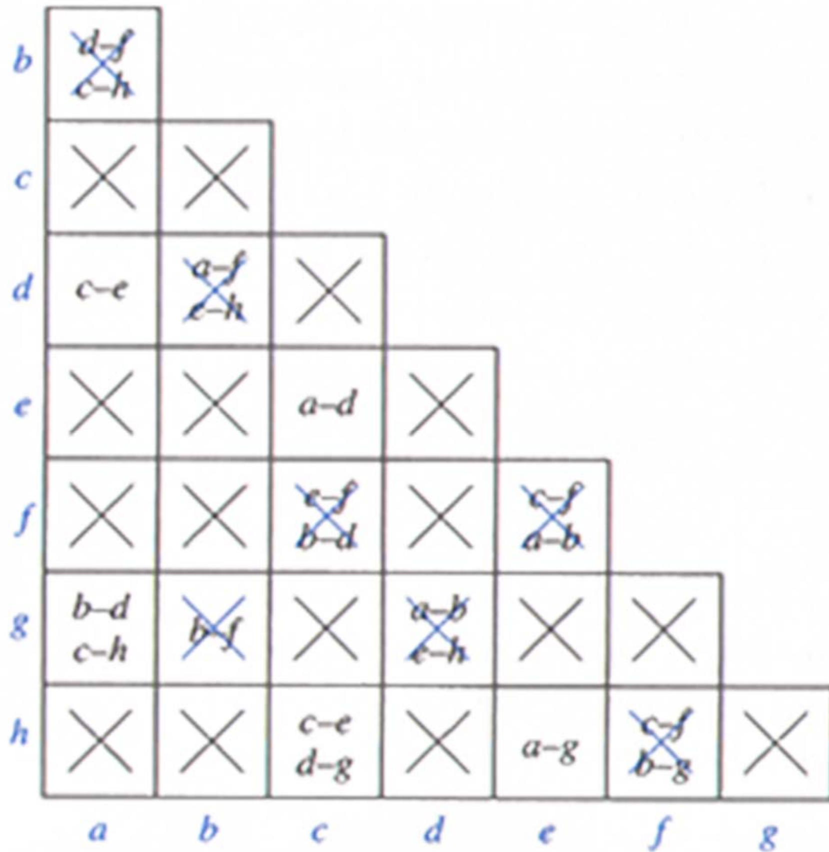
- ✓ X all squares with implied pairs that are not compatible.
- ✓ Such as in a-b have d-f which has an X in it.
- ✓ Run through the chart until no further X's are found.

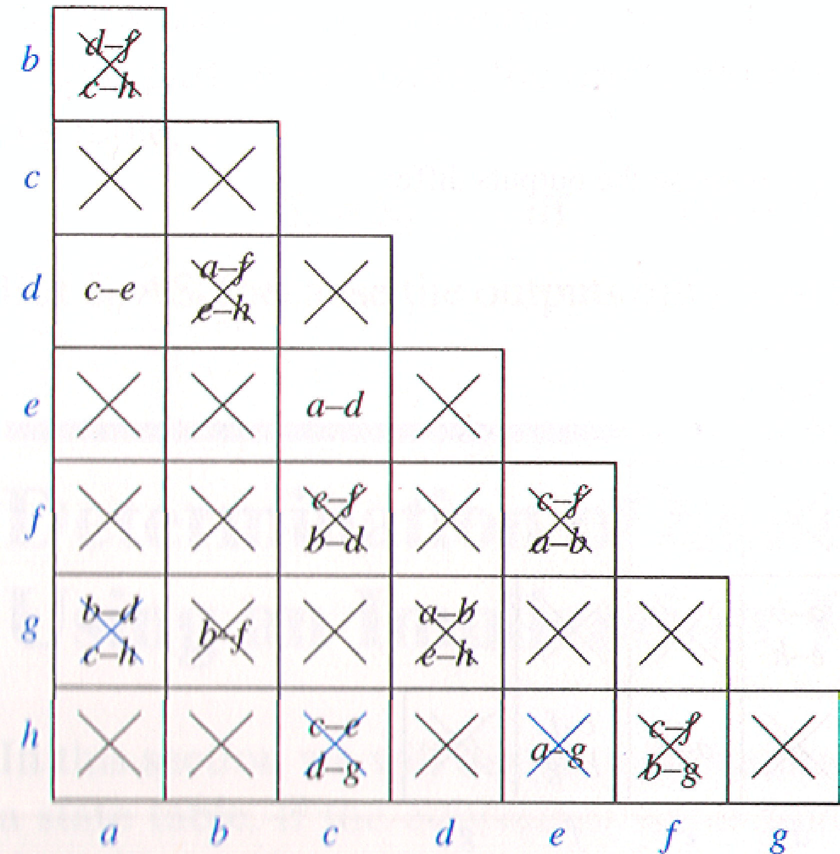




Final step

- ✓ Note that a-d is not X and is equivalent if c≡e, and the same for is c-e: is not X and is equivalent if a≡d. We can conclude that a≡d., i.e. and c≡e.





Reduced table

- ✓ Removing equivalent states.

Present State	Next State $X = 0$	1	Present Output
a	d	c	0
b	f	h	0
c	e	d	1
d	a	e	0
e	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

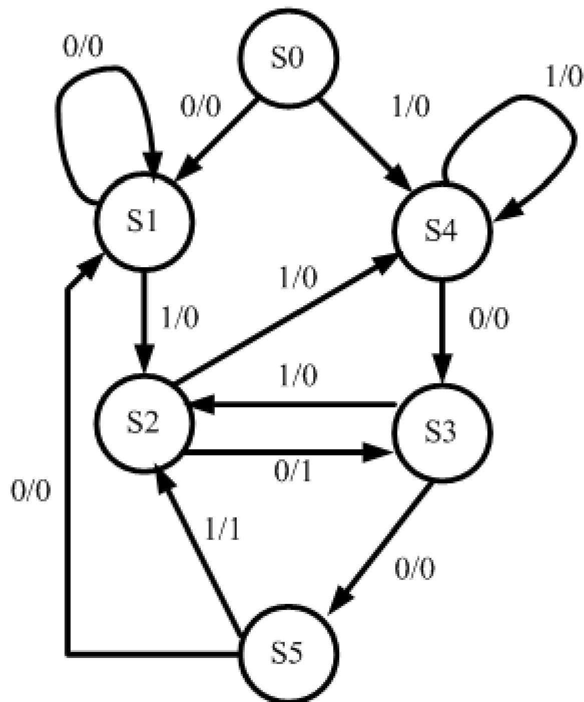
Present State	Next State $X = 0$	1	Present Output
a	a	c	0
b	f	h	0
c	c	a	1
f	f	b	1
g	b	h	0
h	c	g	1

Summary of method

- ✓ Construct a chart with a square for each pair of states.
- ✓ Compare each pair of rows in the state table. X a square if the outputs are different. If the output is the same enter the implied pairs. Remove redundant pairs. If the implied pair is the same place a check mark as $i \equiv j$.
- ✓ Go through the implied pairs and X the square when an implied pair is incompatible.
- ✓ Repeat until no more Xs are added.
- ✓ For any remaining squares not Xed, $i \equiv j$.

Another example

- ✓ Consider the state diagram:



Present State	NEXT STATE X=0	NEXT STATE X=1	OUTPUT X=0	OUTPUT X=1
S0	S1	S4	0	0
S1	S1	S2	0	0
S2	S3	S4	1	0
S3	S5	S2	0	0
S4	S3	S4	0	0
S5	S1	S2	0	1

Set up Implication Chart

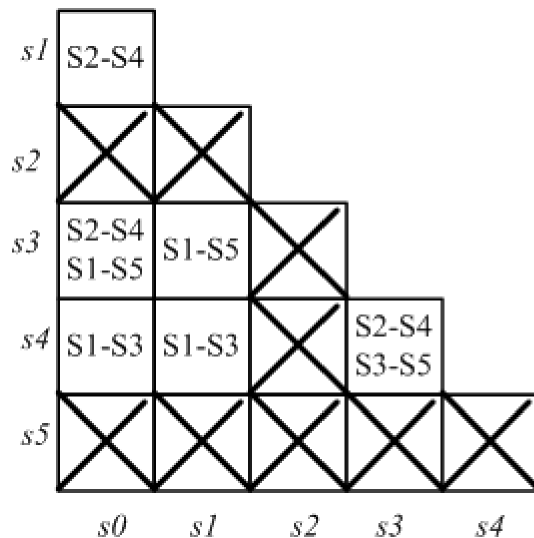
- ✓ Remove output incompatible states
- ✓ and indicate implied pairs

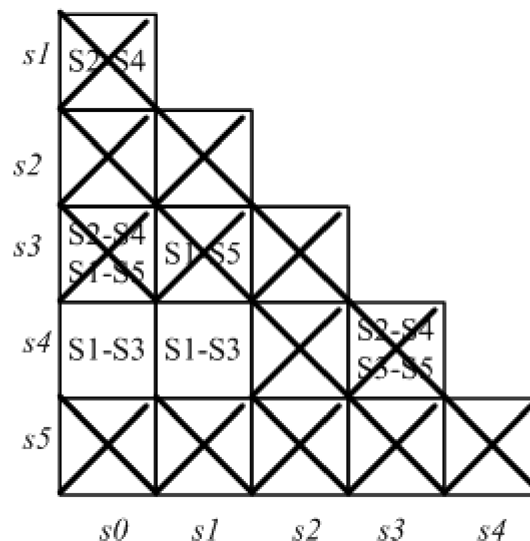
Present State	NEXT STATE X=0	NEXT STATE X=1	OUTPUT X=0	OUTPUT X=1
S0	S1	S4	0	0
S1	S1	S2	0	0
S2	S3	S4	1	0
S3	S5	S2	0	0
S4	S3	S4	0	0
S5	S1	S2	0	1

Check implied pairs and X

1st pass

2nd pass





s1
S2-S4
s2
s3
S2-S4
S1-S5
S1-S5
s4
S1-S3
S1-S3
S2-S4
S3-S5
s5
s0 s1 s2 s3 s4

- ✓ In this case the state table is minimal as no state reduction can be done.

Implication Table (another example)

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	*d*	*b*	0	0
b	*e*	*a*	0	0
c	*g*	*f*	0	1
d	*a*	*d*	1	0
e	*a*	*d*	1	0
f	*c*	*b*	0	0
g	*a*	*e*	1	0

- ✓ Its clear that (e,d) are equivalent. And this leads (a,b) and (e,g) to be equivalent too.
- ✓ Finally we have [(a,b) , c , (e,d,g) , f] so four states.
- ✓ So the original flow table can be reduced to:

b	*d*,*e* ✓					
c	×	×				
d	×	×	×			
e	×	×	×	✓		
f	*c*,*d* ×	*c*,*e* × *a*,*b*	×	×	×	
g	×	×	×	*d*,*e* ✓	*d*,*e* ✓	×
	a	*b*	*c*	*d*	*e*	*f*

Implication Table

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	*d*	*b*	0	0
b	*e*	*a*	0	0
c	*g*	*f*	0	1
d	*a*	*d*	1	0
e	*a*	*d*	1	0
f	*c*	*b*	0	0
g	*a*	*e*	1	0

	a	*b*	*c*	*d*	*e*	*f*
b	*d*, *e* ✓					
c	×	×				
d	×	×	×			
e	×	×	×	✓		
f	*c*, *d* ×	*c*, *e* *a*, *b* ×	×	×	×	
g	×	×	×	*d*, *e* ✓	*d*, *e* ✓	×

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	*d*	*a*	0	0
c	*d*	*f*	0	1
d	*a*	*d*	1	0
f	*c*	*a*	0	0