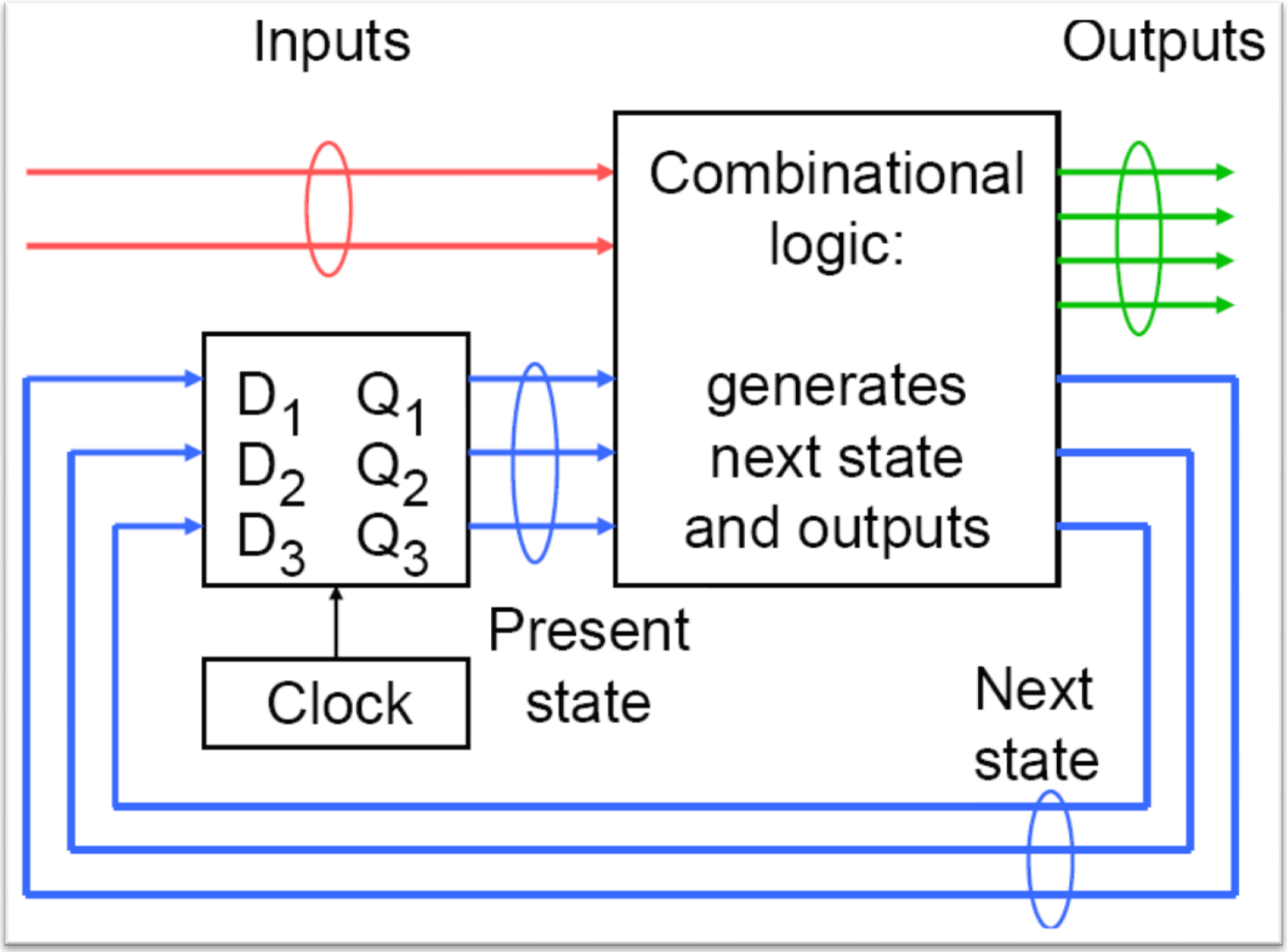
Finite State Machines

Digital Designs ECE 223

Finite-State Machines

Synchronous FSMs	Asynchronous FSMs
Operation is timed by an external clock	Operation is timed by input variable changes
State variables are stored in D-type flip-flops	State variables are stored in gate delays
State variables change simultaneously	State variables change at any time

Synchronous FSM Implementation



Analysis of Synchronous FSMs

The state variables are stored on D-type flip-flops

Present state variables are the outputs of the flip-flops, and are assigned symbols: y_1, y_2, y_3, ...

Next state variables are the inputs of the flip-flops, and are assigned symbols: Y_1, Y_2, Y_3, ...

Inputs to the FSM are assigned symbols: X_1, X_2, X_3, ...

Outputs from the FSM are assigned symbols: Z_1, Z_2, Z_3, ...

Analysis of Synchronous FSMs

To analyse a synchronous FSM:

Determine the next state functions Y_1, Y_2, Y_3, ... of the present state y_1, y_2, y_3, ... and the *inputs* X_1, X_2, X_3, ...

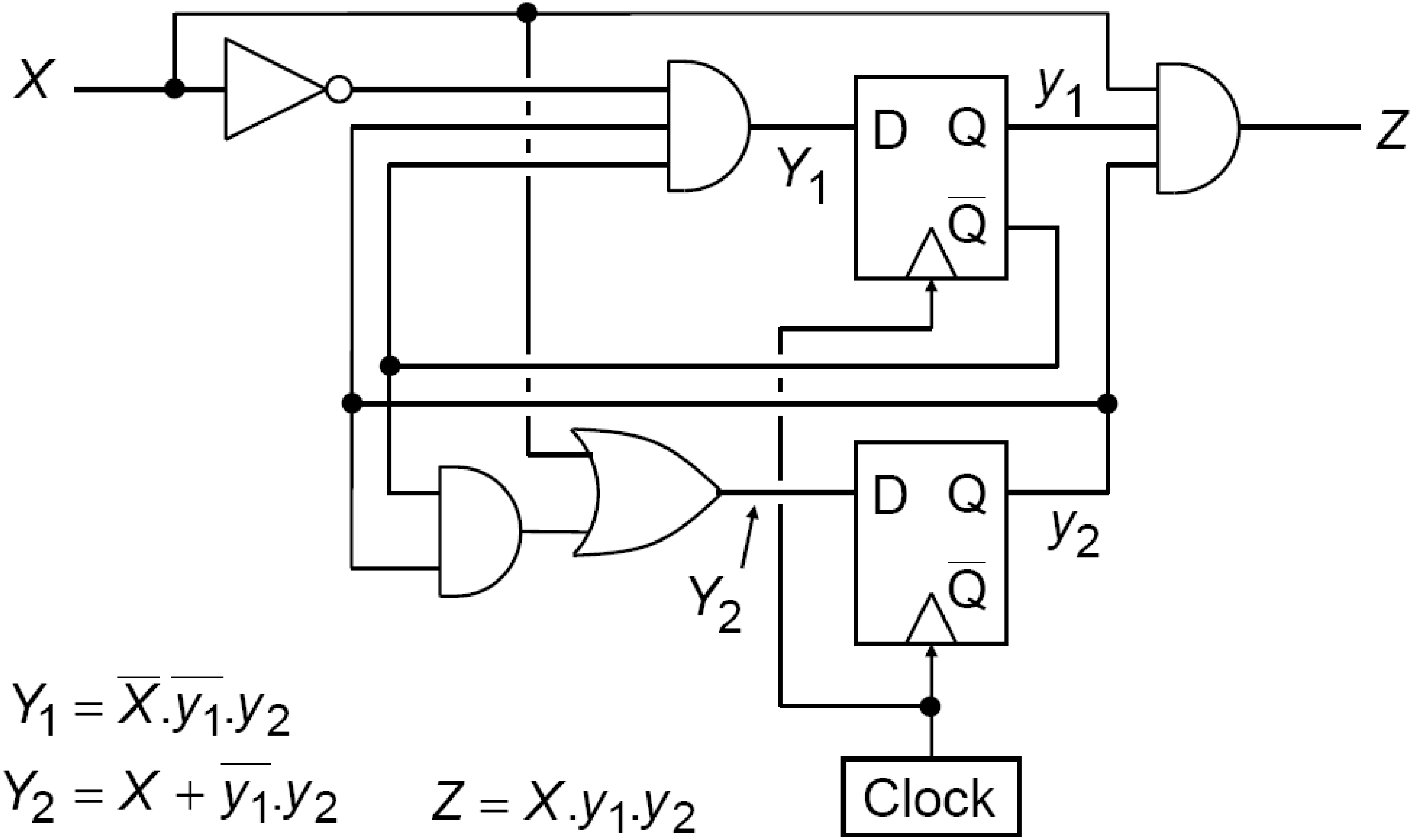
Represent these functions as K-maps

Combine K-maps to produce a Y-map

Assign labels to the state-variable combinations

Finally determine the output functions Z_1, Z_2, Z_3, ... of the present state y_1, y_2, y_3, ... and the *inputs* X_1, X_2, X_3, ...

Analysis of Synchronous FSMs



$$Y_1 = \overline{X}.\overline{y_1}.y_2$$

$$Y_2 = X + \overline{y_1}.y_2 \qquad Z = X.y_1.y_2$$

Analysis of Synchronous FSMs

$Y_1 = \overline{X}.\overline{y_1}.y_2$ $Y_2 = X + \overline{y_1}.y_2$ $Z = X.y_1.y_2$

y_1y_2	X=0	X=1
00	0	0
01	1	0
11	0	0
10	0	0

Y_1

y_1y_2	X=0	X=1
00	0	1
01	1	1
11	0	1
10	0	1

Y_2

y_1y_2	X=0	X=1
00	0	0
01	0	0
11	0	1
10	0	0

Z

Rows are state-variable combinations
Columns are input combinations

Analysis of Synchronous FSMs

y_1y_2	X=0	X=1
00	0	0
01	1	0
11	0	0
10	0	0

Y_1

y_1y_2	X=0	X=1
00	0	1
01	1	1
11	0	1
10	0	1

Y_2

Y-map

y_1y_2	X=0	X=1
00	00	01
01	11	01
11	00	01
10	00	01

Y_1 Y_2

Analysis of Synchronous FSMs

$y_1=0, y_2=0 \rightarrow A$ $\quad$ $y_1=0, y_2=1 \rightarrow B$

$y_1=1, y_2=1 \rightarrow C$ $\quad$ $y_1=1, y_2=0 \rightarrow D$

Y-map

y_1y_2	X=0	X=1
00	00	01
01	11	01
11	00	01
10	00	01

$Y_1\ Y_2$

State Table

Present state	X=0	X=1
A	A	B
B	C	B
C	A	B
D	A	B

Next state

However it would be better practice if each state was given a more meaningful name that could be used by other engineers to understand the method or system used.

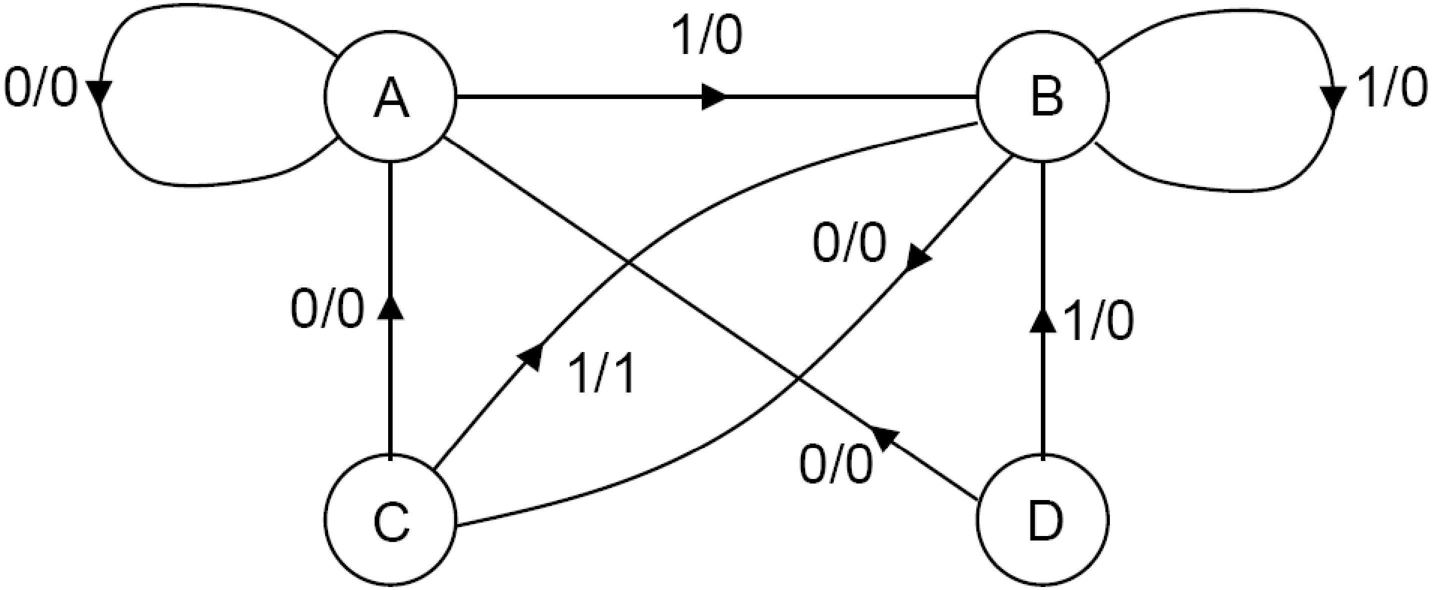
Analysis of Synchronous FSMs

State table with output:

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

Analysis of Synchronous FSMs

State diagram (transitions labelled: *X/Z*):



The "D" state is a dead state, it has no logic combination that leads to it and it can only occur as a first or starting state for the system. These dead states can be removed to simplify the logic used in the final system.

When they are removed they should make no change to the external behaviour of the system.

Design of Synchronous FSMs

1. Understand the problem
2. Obtain formal definition as state diagram or state table
3. Simplify the design by state reduction.
4. Determine the number of state variables.
5. Choose a state assignment.
6. Determine next-state functions
7. Check unused states
8. Determine output functions
9. Implement next-state and output functions

Traffic Lights

A 2-way junction has traffic lights for N-S lanes, E-W lanes and stop/go lights for pedestrians

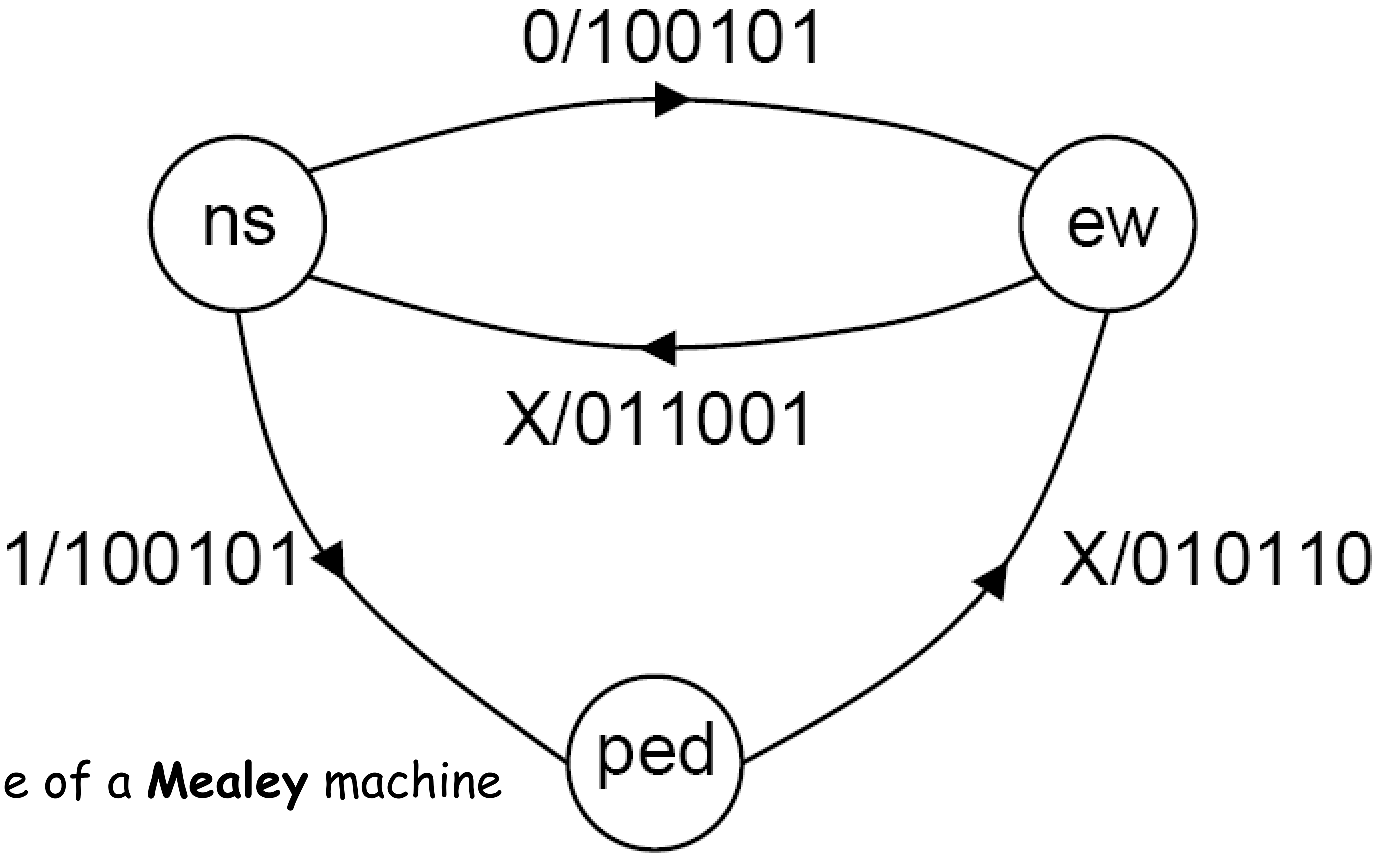
Normally the lights sequence between N-S and E-W

If pedestrian presses request button *p then N-S and E-W go* to red and pedestrian lights to go:

State	NSg	NSr	EWg	EWr	Pgo	Pst
ns	1	0	0	1	0	1
ew	0	1	1	0	0	1
ped	0	1	0	1	1	0

Traffic Lights

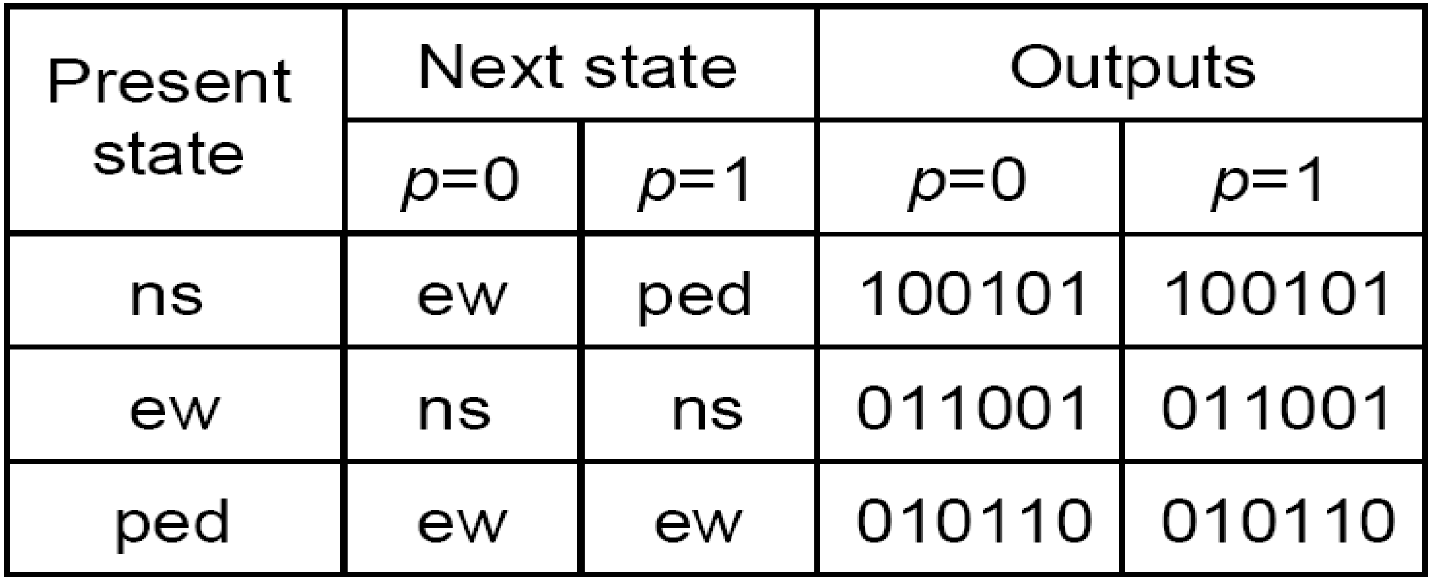
Translations labelled: *p*/NSg NSr Ewg Ewr Pgo Pst



An example of a **Mealey** machine

State Tables

For each present state and input combination the state table shows the next state and the outputs.



Present state	Next state		Outputs	
	p=0	p=1	p=0	p=1
ns	ew	ped	100101	100101
ew	ns	ns	011001	011001
ped	ew	ew	010110	010110

A state table is used to create the logic equations for any given situation. Applying this knowledge to real life situations is a common exam question

Counting Machine

- "Design a clocked synchronous state machine with two inputs, X and Y, and one output Z. The output should be 1 if inputs on X and Y since reset is a multiple of 4, and 0 otherwise.
- There are 4 states

Construct a state table

- □ From word description construct a state table for the problem.

		X Y				
Meaning	S	00	01	11	10	Z
Have zero 1's mod 4	S0	S0	S1	S2	S1	1
Have one 1 mod 4	S1	S1	S2	S3	S2	0
Have two 1's mod 4	S2	S2	S3	S0	S3	0
Have three 1's mod 4	S3	S3	S0	S1	S0	0
		S*				

Do a state assignment

- □ Having state table pick a state assignment
- □ From here we can generate the excitation equations

		X Y				
Q1 Q2	S	00	01	11	10	Z
00	S0	00	01	11	01	1
01	S1	01	11	10	11	0
11	S2	11	10	00	10	0
10	S3	10	00	01	00	0
		Q1* Q2*				

Excitation Equations

- D1 = Q2 X' Y + Q1' X Y + Q1 X' Y' + Q2 X Y'

D1 Map

X Y	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	1	1	0	1
10	1	0	0	0

		X Y				
Q1 Q2	S	00	01	11	10	Z
00	S0	00	01	11	01	1
01	S1	01	11	10	11	0
11	S2	11	10	00	10	0
10	S3	10	00	01	00	0
		Q1* Q2*				

- Z = Q1' Q2'

D2 Map

X Y

	00	01	11	10
00	0	1	1	1
01	1	1	0	1
11	1	0	0	0
10	0	0	1	0

		X Y				
Q1 Q2	S	00	01	11	10	Z
00	S0	00	01	11	01	1
01	S1	01	11	10	11	0
11	S2	11	10	00	10	0
10	S3	10	00	01	00	0
		Q1* Q2*				

D2 = Q1' X' Y + Q1' X Y' + Q2 X' Y' + Q2 X Y

Another Example

- Design a clocked synchronous state machine with one input X and two outputs, UNLK and HINT. The UNLK output should be 1 if and only if X is 0 and the sequence of inputs received on X the preceding seven clock ticks was 0110111. The HINT output should be 1 if and only if the current value of X is the correct one to move the machine close to being in the "unlocked" state (with UNLK = 1).

Create State Table

- Create a state table from the word description

		X	
Meaning	S	0	1
Got Zip	A	B,01	A,00
Have 0	B	B,00	C,01
Have 01	C	B,00	D,01
Have 011	D	E,01	A,00
Have 0110	E	B,00	F,01
Have 01101	F	B,00	G,01
Have 011011	G	E,00	H,01
Have 0110111	H	B,11	A,00
		S*,UNLK,HINT	

Choose a state assignment

- To get transition/excitation table

	X	
Q1 Q2 Q3	0	1
000	001,01	000,00
001	001,00	010,01
010	001,00	000,00
011	100,01	100,01
100	001,00	101,01
101	001,00	110,01
110	100,00	111,01
111	001,11	000,00
	Q1*Q2*Q3*,UNLK HINT	

Can use Karnaugh Map to get excitation equations

- $D1 = Q1\,Q2'\,X + Q1'\,Q2\,Q3\,X' + Q1\,Q2\,Q3'$
- $D2 = Q2'\,Q3\,X + Q2\,Q3'\,X$
- $D3 = Q1\,Q2'\,Q3' + Q1\,Q3\,X' + Q2'\,X' + Q3'\,Q1'\,X' + Q2\,Q3'\,X$
- $\mathrm{UNLK} = Q1\,Q2\,Q3\,X'$
- $\mathrm{HINT} = Q1'\,Q2'\,Q3'\,X' + Q1\,Q2'\,X + Q2'\,Q3\,X + Q2\,Q3\,X' + Q2\,Q3'\,X$

Unused States

In general not all of the possible combinations of the state variables will be assigned to states.

On power-up the contents of the memory elements (D-type flip-flops) are unpredictable and may correspond to an unused state

It is important that all unused states go, within a few clock cycles, to one of the valid states

It is not necessary that every unused state should go directly to a valid state

Unused States

There are two approaches for dealing with unused states.

The simplest is to force all unused states unconditionally (that is irrespective of the values of the inputs) to valid states

A more efficient approach is to treat the values of the next-state variables for the unused states as “don't care”

In this case the design must be inspected to ensure that unused states do not form a subsidiary loop

State Reduction

It may be possible to reduce the number of states in the state diagram or the state table without affecting the terminal properties of the FSM

This may or may not reduce the number of state variables

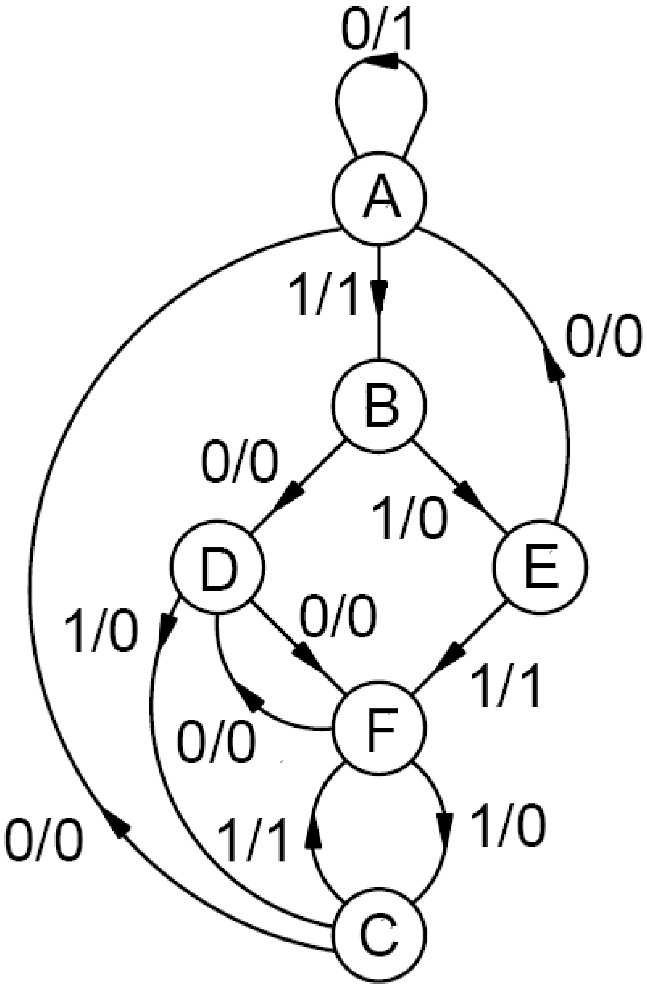
The purpose of state reduction is to reduce the number of states in a FSM definition without altering the input/output relationship

State reduction is performed on the state table, rather than the state diagram

State Reduction

FSM with one input *X*, and one output *Z:*
(transitions are labelled *X/Z)*

State diagrams should never be used for state reduction. In more complex systems it is very easy to miss possible state reductions due to the complexity of the diagrams involved.



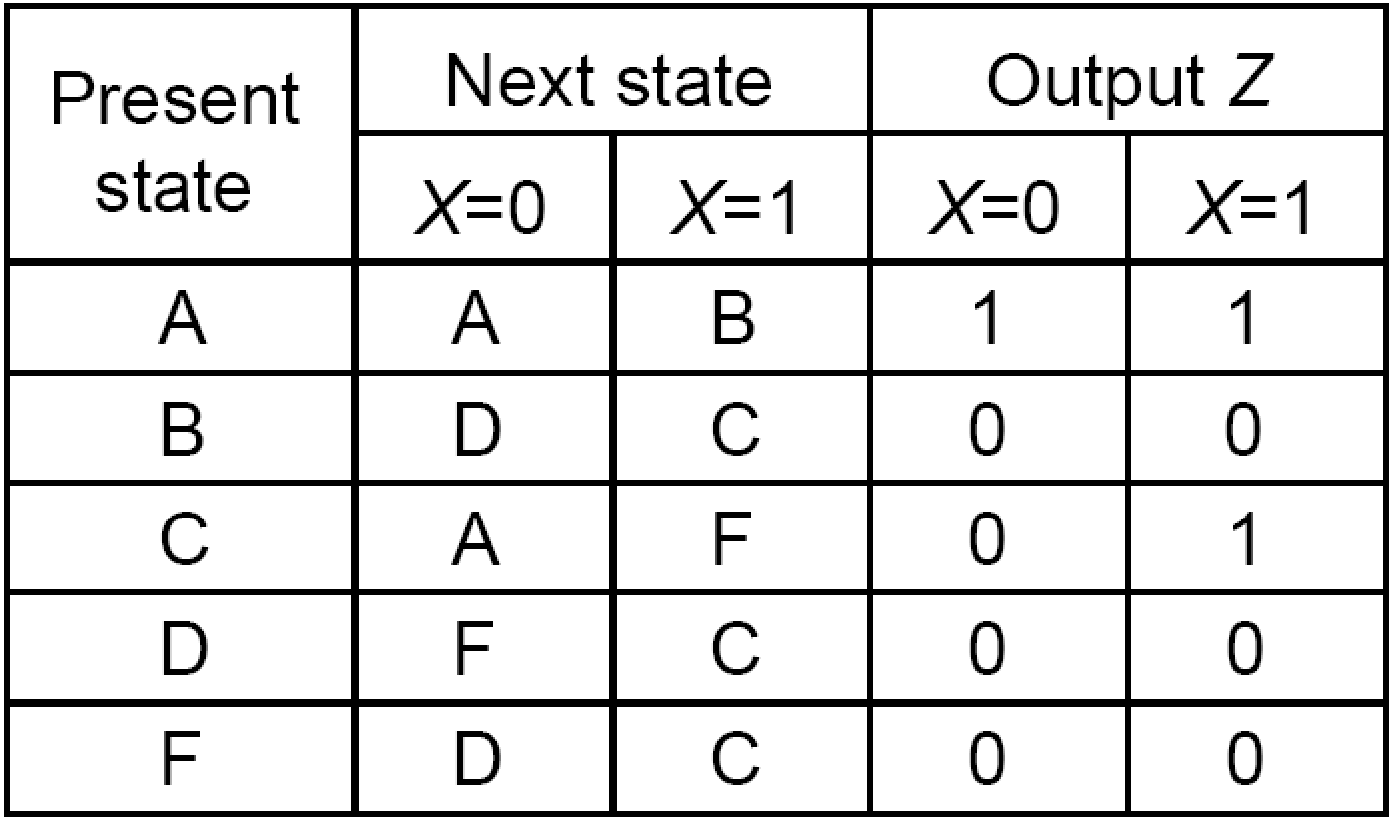
State Reduction

Equivalent state table:

Present state	Next state		Output Z	
	X=0	X=1	X=0	X=1
A	A	B	1	1
B	D	E	0	0
C	A	F	0	1
D	F	C	0	0
E	A	F	0	1
F	D	C	0	0

State Reduction

Reduced state table:

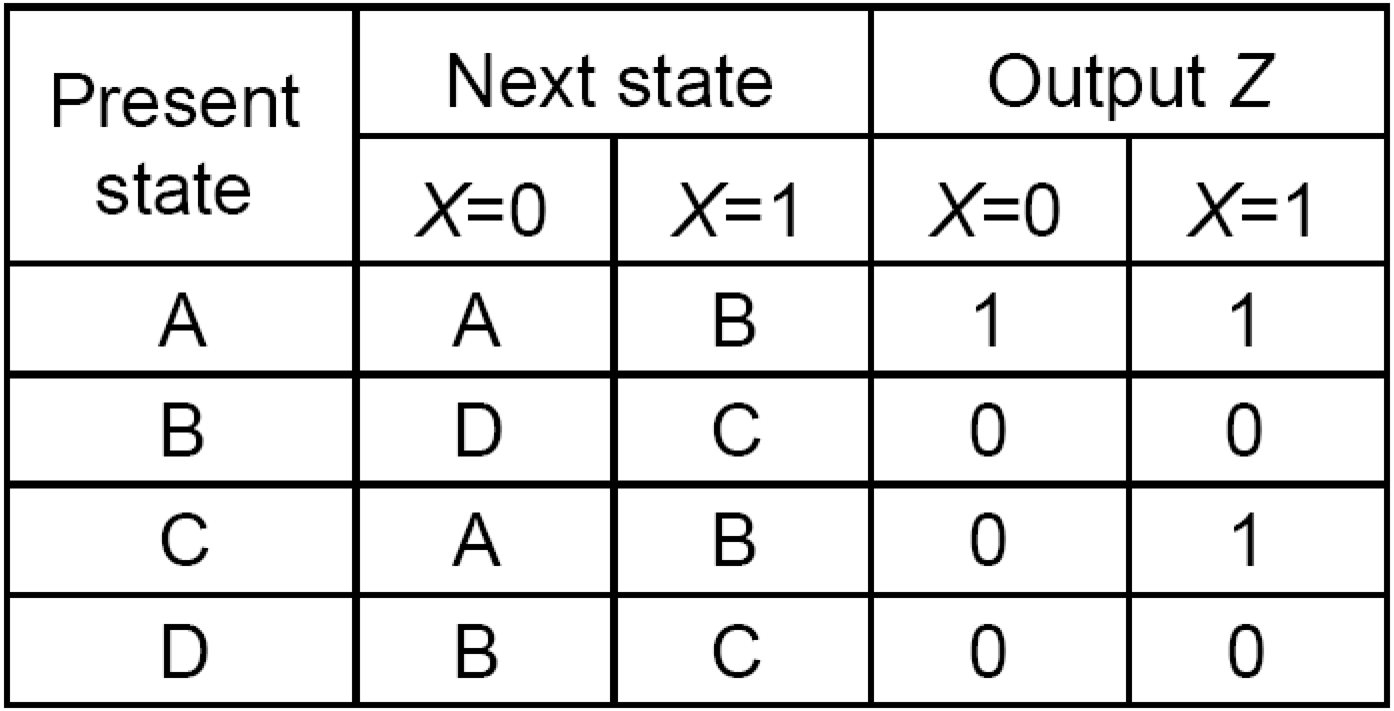


Present state	Next state		Output Z	
	X=0	*X*=1	*X*=0	*X*=1
A	A	B	1	1
B	D	C	0	0
C	A	F	0	1
D	F	C	0	0
F	D	C	0	0

With the removal of E, D and F can be merged.

State Reduction

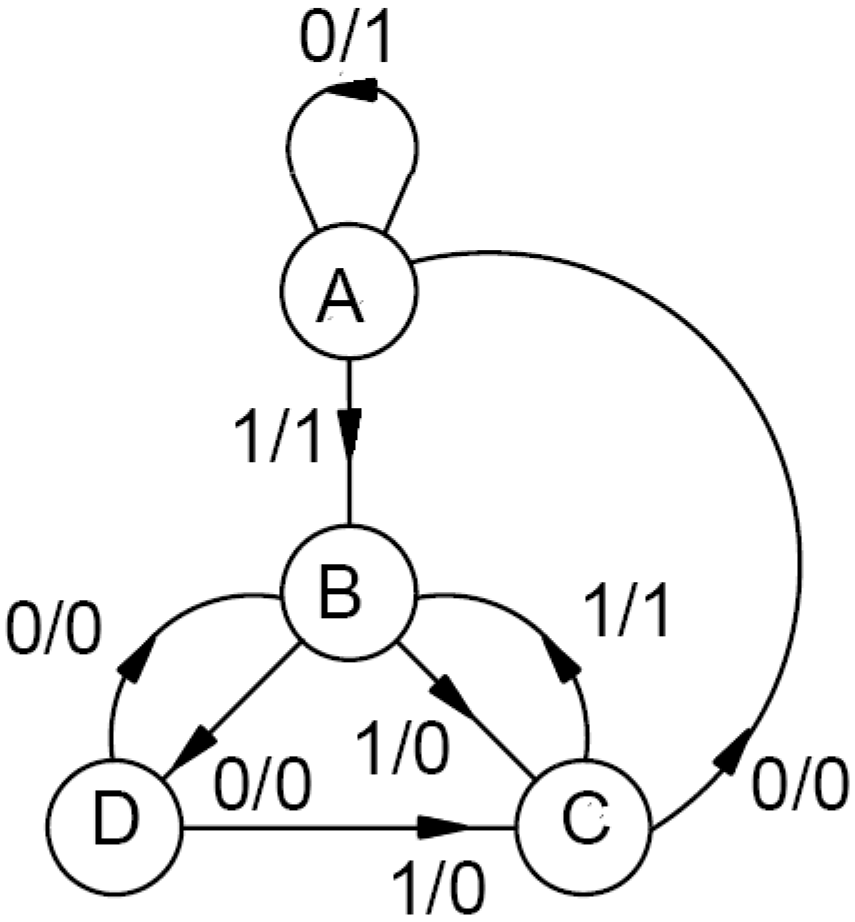
Reduced state table:

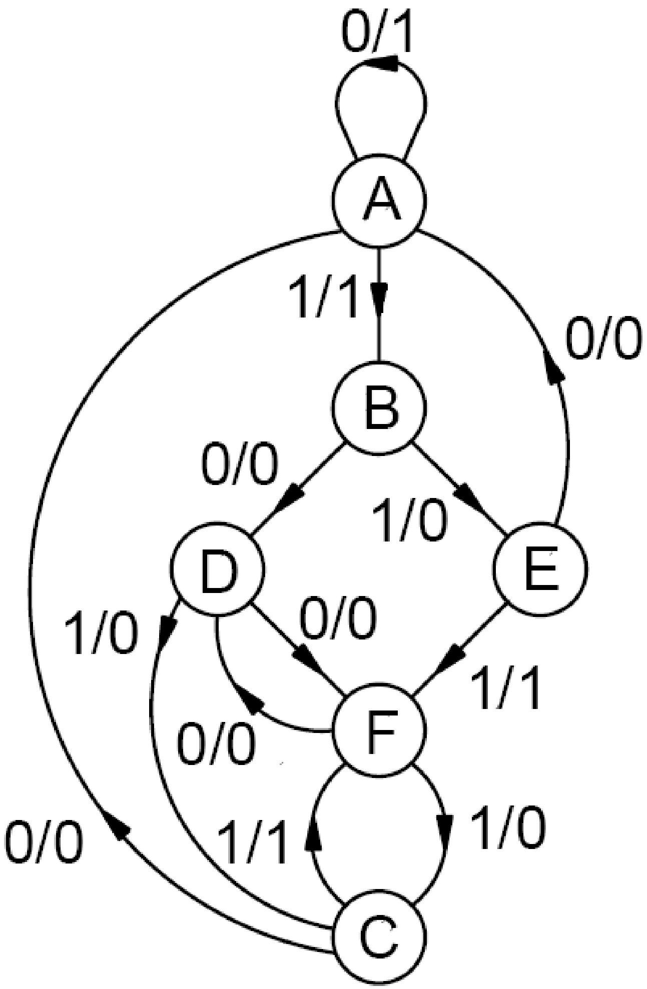


Present state	Next state		Output *Z*	
	X=0	*X*=1	*X*=0	*X*=1
A	A	B	1	1
B	D	C	0	0
C	A	B	0	1
D	B	C	0	0

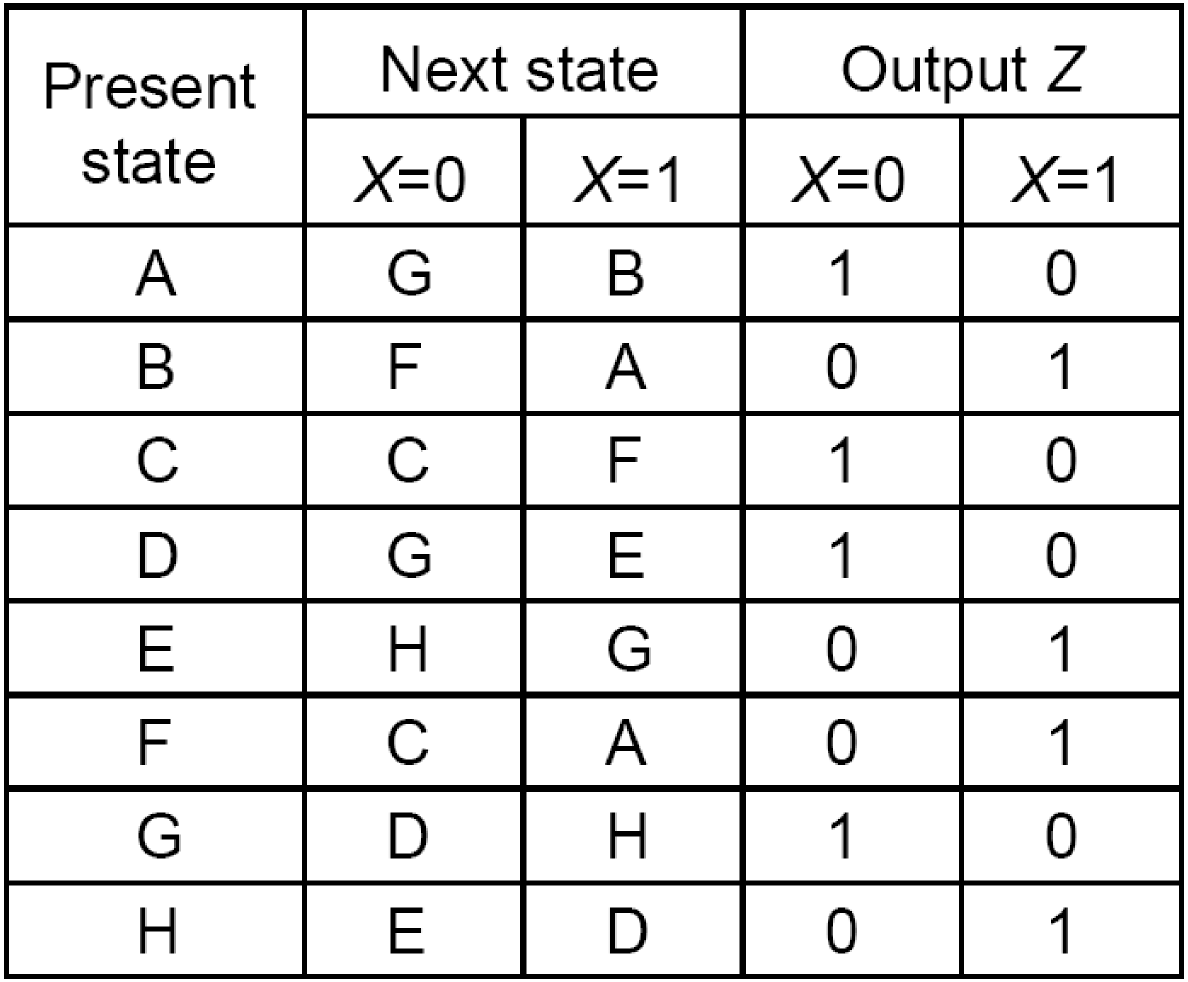
State Reduction

State diagram of reduced FSM:





Implication Charts



Present state	Next state		Output *Z*	
	X=0	*X*=1	*X*=0	*X*=1
A	G	B	1	0
B	F	A	0	1
C	C	F	1	0
D	G	E	1	0
E	H	G	0	1
F	C	A	0	1
G	D	H	1	0
H	E	D	0	1

Implication Charts

1. Draw

Present state	Next state		Output Z	
	X=0	X=1	X=0	X=1
A	G	B	1	0
B	F	A	0	1
C	C	F	1	0
D	G	E	1	0
E	H	G	0	1
F	C	A	0	1
G	D	H	1	0
H	E	D	0	1

B							
C							
D							
E							
F							
G							
H							
	A	B	C	D	E	F	G

Implication Charts

2. Where states do not have outputs, place an x

Present state	Next state		Output Z	
	X=0	X=1	X=0	X=1
A	G	B	1	0
B	F	A	0	1
C	C	F	1	0
D	G	E	1	0
E	H	G	0	1
F	C	A	0	1
G	D	H	1	0
H	E	D	0	1

B	X						
C		X					
D		X					
E	X		X	X			
F	X		X	X			
G		X			X	X	
H	X		X	X			X
	A	B	C	D	E	F	G

Implication Charts

3. Where states have the same output, put equalities in the box

A	G	B	1	0
C	C	F	1	0

	A	B	C	D	E	F	G
B	✗						
C	C ≡ G B ≡ F	✗					
D	B ≡ E	✗	C ≡ G E ≡ F				
E	✗	F ≡ H A ≡ G	✗	✗			
F	✗	C ≡ F	✗	✗	C ≡ H A ≡ G		
G	D ≡ G B ≡ H	✗	C ≡ D F ≡ H	E ≡ H	✗	✗	
H	✗	E ≡ F A ≡ D	✗	✗	D ≡ G	C ≡ E A ≡ D	✗

Implication Charts

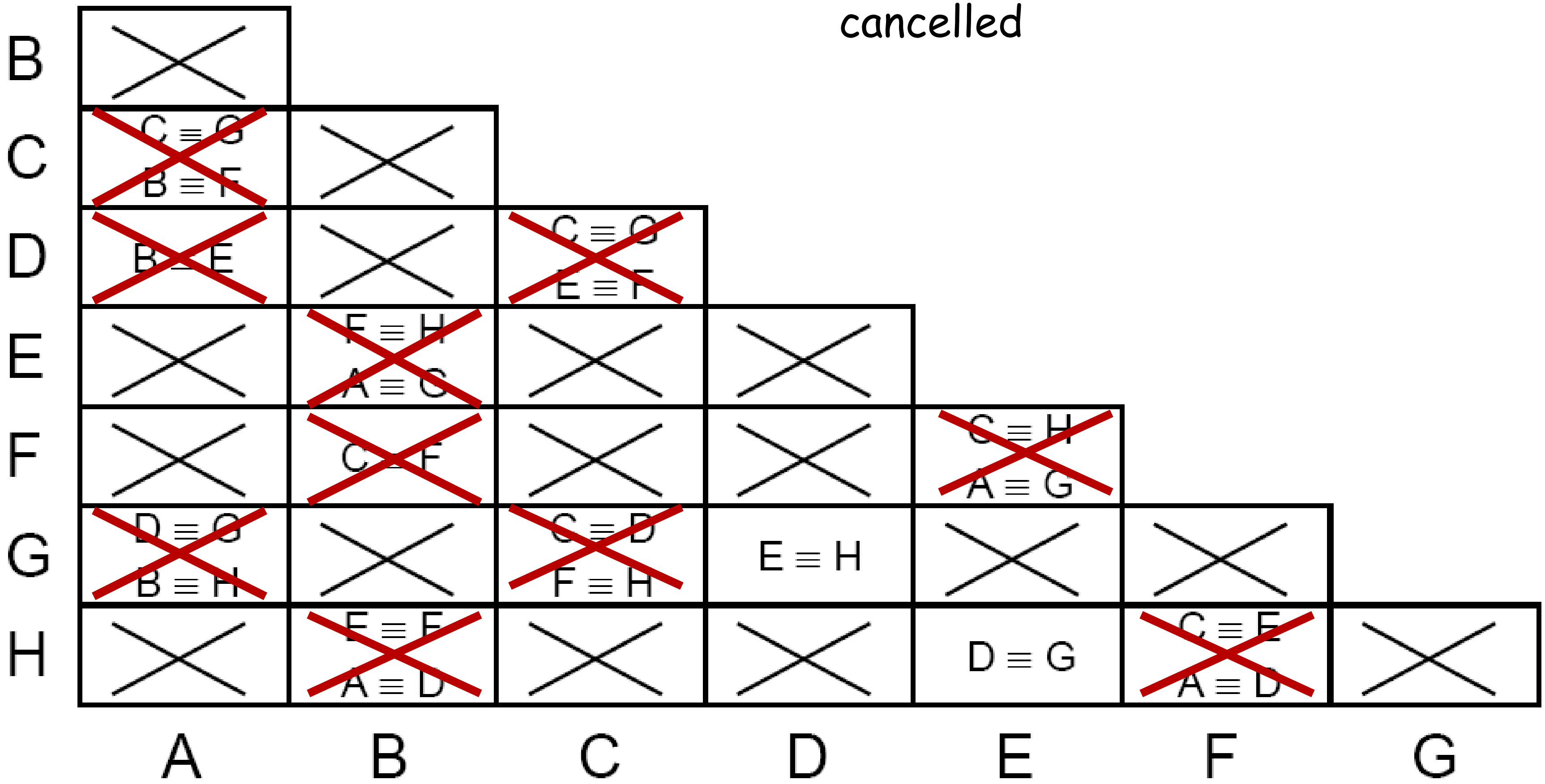
4. Where equivalents are false, cross out the box

(e.g. B,F has C≡F as a false equivalent as C,F has no contents)

	A	B	C	D	E	F	G
B	✗						
C	C ≡ G B ≡ F	✗					
D	B ≡ E	✗	C ≡ G E ≡ F				
E	✗	F ≡ H A ≡ G	✗	✗			
F	✗	~~C ≡ F~~	✗	✗	C ≡ H A ≡ G		
G	D ≡ G B ≡ H	✗	C ≡ D F ≡ H	E ≡ H	✗	✗	
H	✗	E ≡ F A ≡ D	✗	✗	D ≡ G	C ≡ E A ≡ D	✗

Implication Charts

5. Repeat until no more can be cancelled



B	✕						
C	~~$C \equiv G$ $B \equiv F$~~	✕					
D	~~$B \equiv E$~~	✕	~~$C \equiv G$ $E \equiv F$~~				
E	✕	~~$F \equiv H$ $A \equiv G$~~	✕	✕			
F	✕	~~$C \equiv F$~~	✕	✕	~~$C \equiv H$ $A \equiv G$~~		
G	~~$D \equiv G$ $B \equiv H$~~	✕	~~$C \equiv D$ $F \equiv H$~~	$E \equiv H$	✕	✕	
H	✕	~~$E \equiv F$ $A \equiv D$~~	✕	✕	$D \equiv G$	~~$C \equiv E$ $A \equiv D$~~	✕
	A	B	C	D	E	F	G

Implication Charts

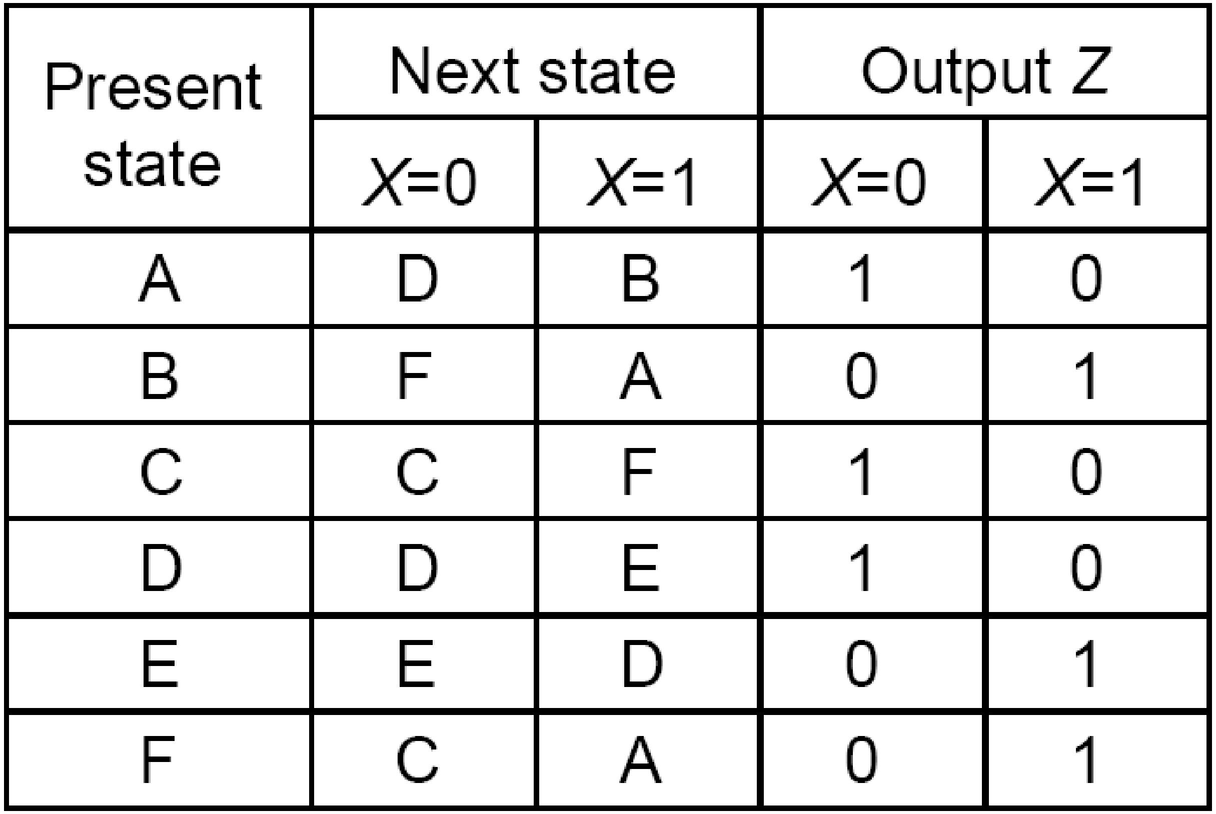
6. What is left can be replaced with equivalents

(e.g. H with E, G with D)

B	✕						
C	✕	✕					
D	✕	✕	✕				
E	✕	✕	✕	✕			
F	✕	✕	✕	✕	✕		
G	✕	✕	✕	E ≡ H	✕	✕	
H	✕	✕	✕	✕	D ≡ G	✕	✕
	A	B	C	D	E	F	G

Implication Charts

7. Draw reduced state table

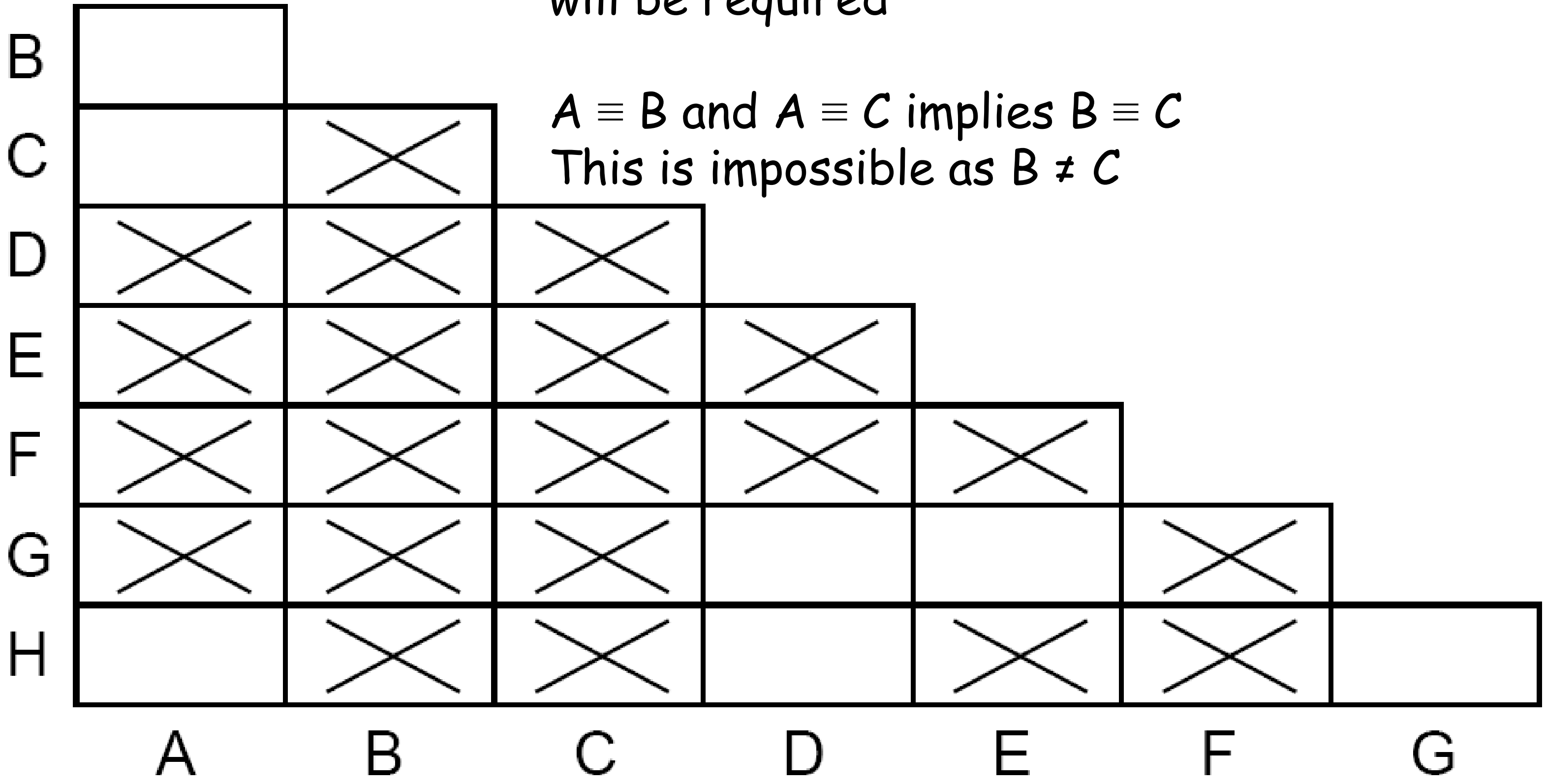


Present state	Next state		Output Z	
	X=0	X=1	X=0	X=1
A	D	B	1	0
B	F	A	0	1
C	C	F	1	0
D	D	E	1	0
E	E	D	0	1
F	C	A	0	1

Merger Diagrams

Sometimes implication charts show many possible equivalences. If this occurs a Merger diagram will be required

A ≡ B and A ≡ C implies B ≡ C
This is impossible as B ≠ C



	A	B	C	D	E	F	G
B							
C		X					
D	X	X	X				
E	X	X	X	X			
F	X	X	X	X	X		
G	X	X	X			X	
H		X	X		X	X	

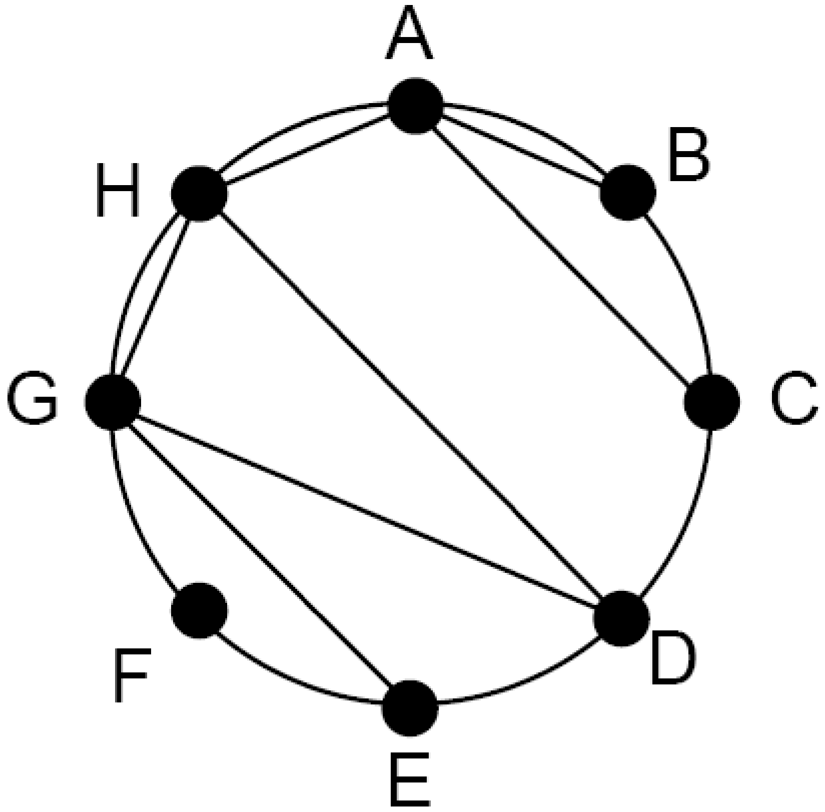
A≡B A≡C A≡H D≡G D≡H E≡G G≡H

Merger Diagrams

- Lines are placed on the merger diagram with regards to all possible equivalences

- Polygons formed by these lines with all their sides displayed are to be found.

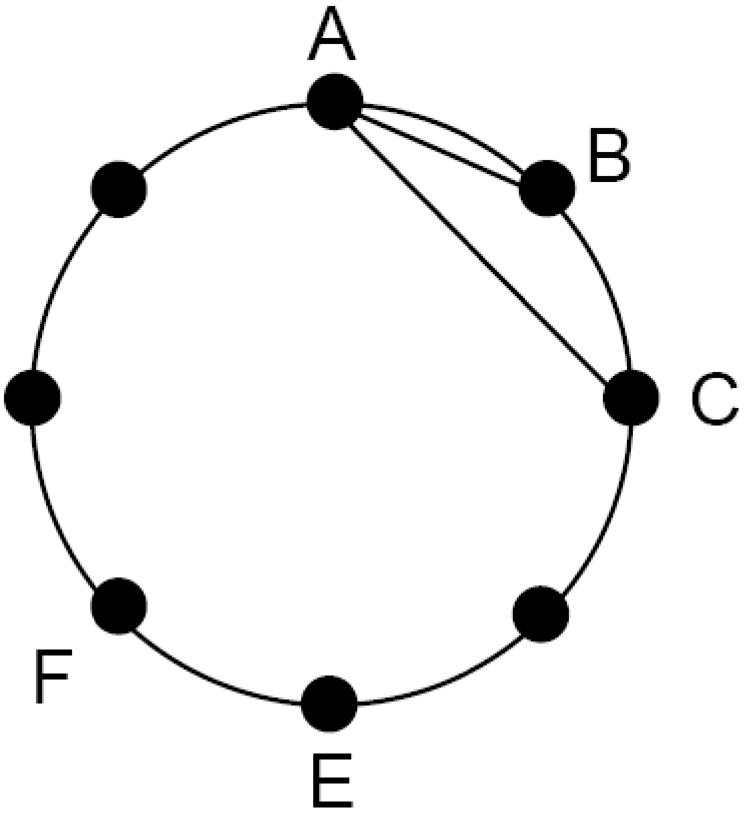
- The triangle GDH determines that G≡D≡H must be true



G≡H≡D
A ≡ B
A ≡ C
B ≠ C

Merger Diagrams

- We are left with an arbitrary decision between A≡B and A≡C



$A \equiv B$
$A \equiv C$
$B \neq C$

Asynchronous Sequential Logic

Asynchronous sequential systems have no clock; internal states change when there is a change in the input variables

Memory is effectively provided by the finite time taken by a signal to propagate through the gates

Asynchronous sequential systems are used where a fast response to input changes, without having to wait for a clock transition, is necessary

Asynchronous sequential systems are also used where the introduction of extra frequency components related to the clock must be avoided

Asynchronous processors are being considered for very fast applications such as computers.

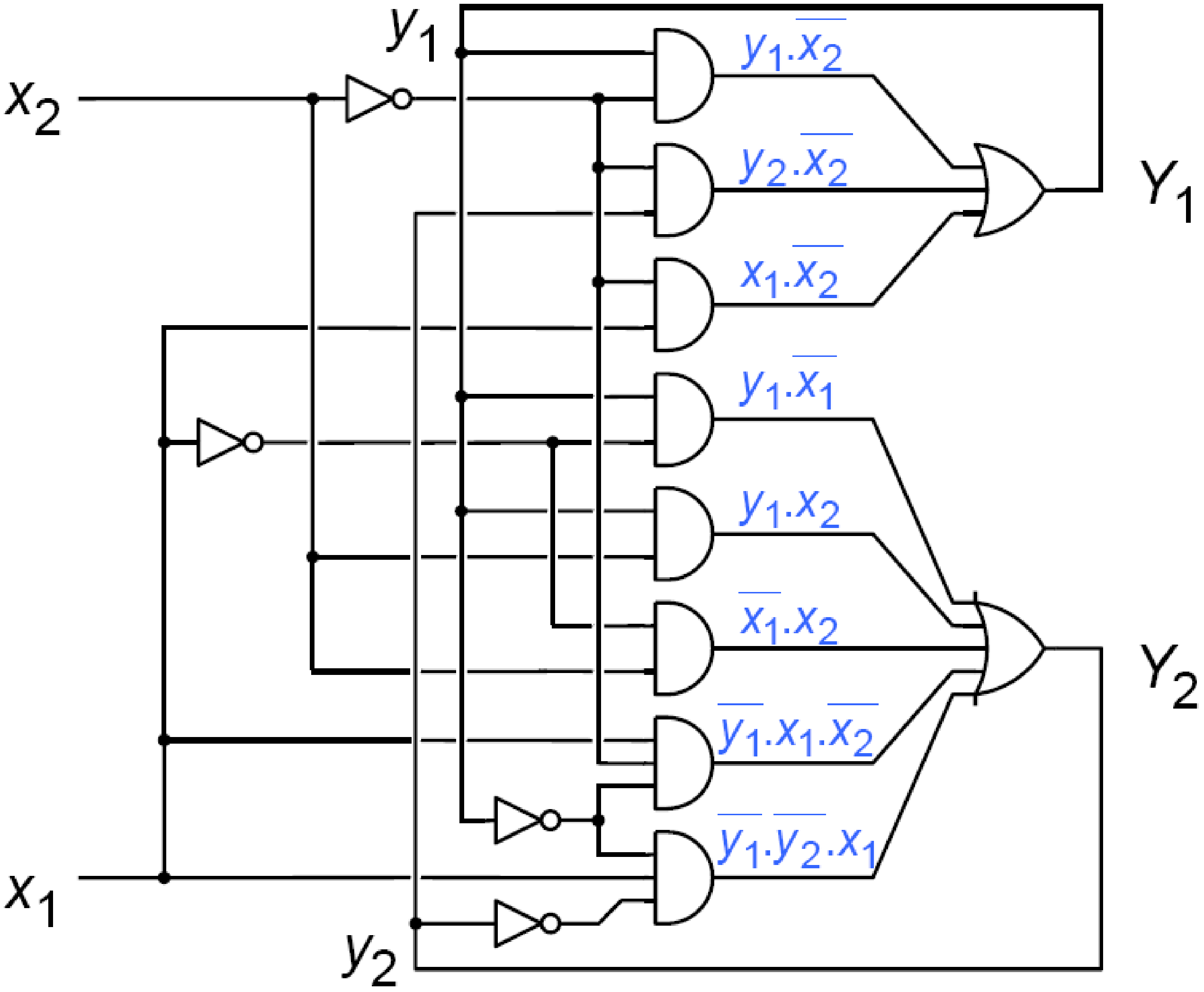
Fundamental Mode

Asynchronous FSMs operating under the restriction that:

1. only one input variable may change at a time
2. that the time between input changes is greater than the time required to reach a steady state

are said to operate in the fundamental mode

Analysis of Asynchronous FSMs



Analysis of Asynchronous FSMs

The next-state functions for this system are given by:

$$Y_1 = y_1.\overline{x_2} + y_2.\overline{x_2} + x_1.\overline{x_2}$$

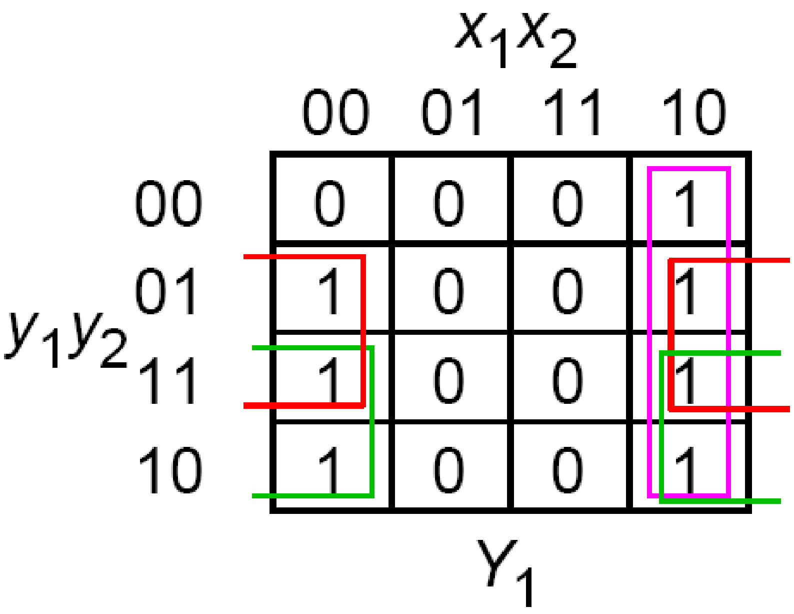
$$Y_2 = y_1.\overline{x_1} + y_1.x_2 + \overline{x_1}.x_2 + \overline{y_1}.x_1.\overline{x_2} + \overline{y_1}.\overline{y_2}.x_1$$

These can be represented on K-maps

It is convenient to associate the present state variables y_i with the rows of the K-map and the input variables x_i with the columns of the K-map

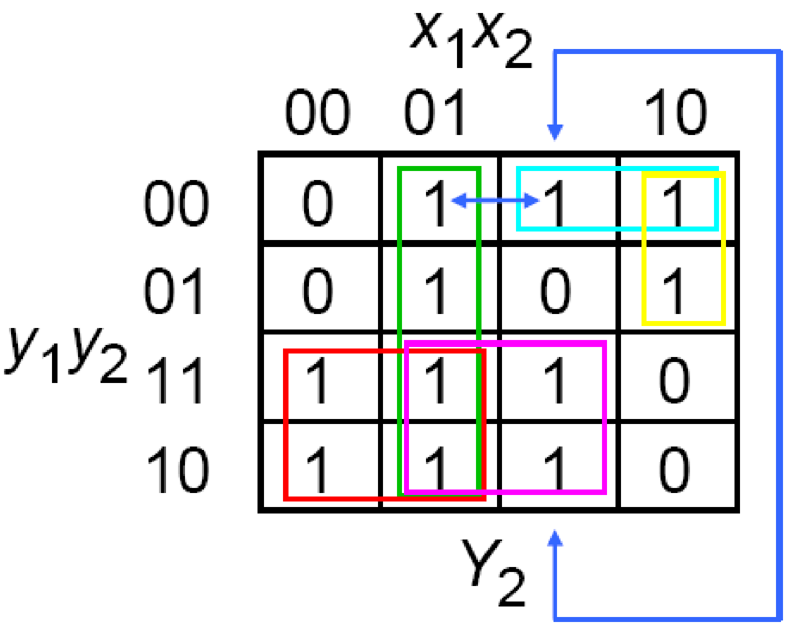
Analysis of Asynchronous FSMs

K-maps of next-state functions:



	00	01	11	10
00	0	0	0	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	1

Hazards!



	00	01		10
00	0	1	1	1
01	0	1	0	1
11	1	1	1	0
10	1	1	1	0

Static hazards in K-map for Y_2 will cause unpredictable behaviour – errors propagating through the system may cause it to crash. Even if crashes are not noted in the system, it is bad practice to leave hazards present.

Analysis of Asynchronous FSMs

Combining K-maps gives Y-map:

y_1y_2 \ x_1x_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

Entries where next state equals present state (Y_1=y_1, Y_2=y_2) correspond to stable states and have been marked: ○

Analysis of Asynchronous FSMs

Present state 11, inputs 00. The Y-map indicates that the next state is 11, which is the same as the present state.

Thus the system is stable

Present state 11, inputs change from 00 to 01. The Y-map indicates that the next state is 01; this is unit distance from 11 so there is no ambiguity. The present state then becomes 01 and the Y-map indicates that the next state is now 01; the system has reached a stable state

11 → 01 → 01 (stable)

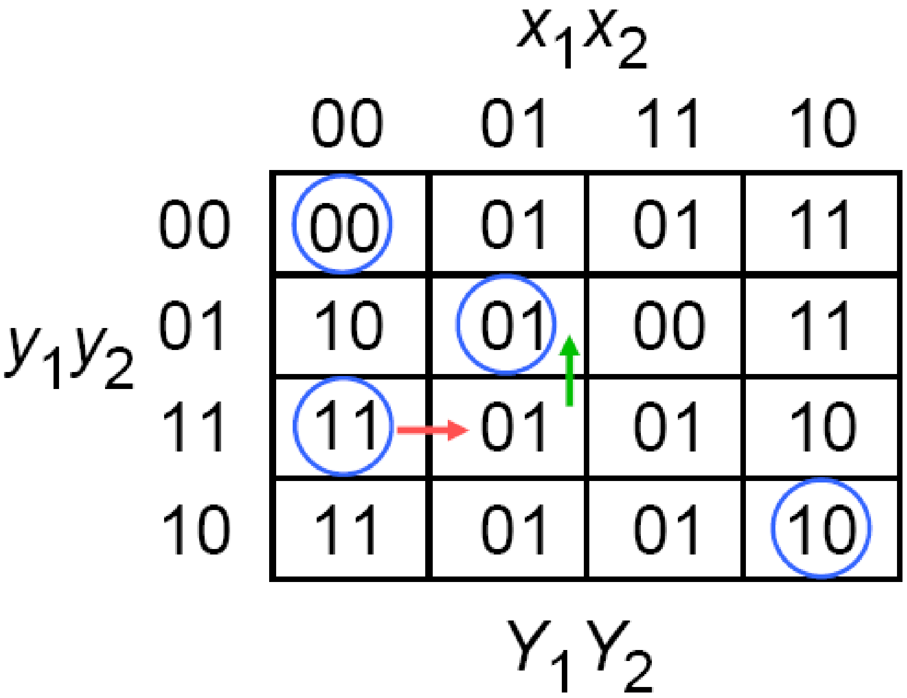
x_1x_2

y_1y_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

Analysis of Asynchronous FSMs

Present state 11, inputs change from 00 to 01



	x_1x_2			
y_1y_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

11 → 01 → 01 (stable)

Analysis of Asynchronous FSMs

Present state 01, inputs change from 01 to 11

The Y-map now indicates a next state of 00; this is unit distance from 01 so there is no ambiguity

When the present state has become 00 the next state is 01. Thus the system will oscillate between 00 and 01

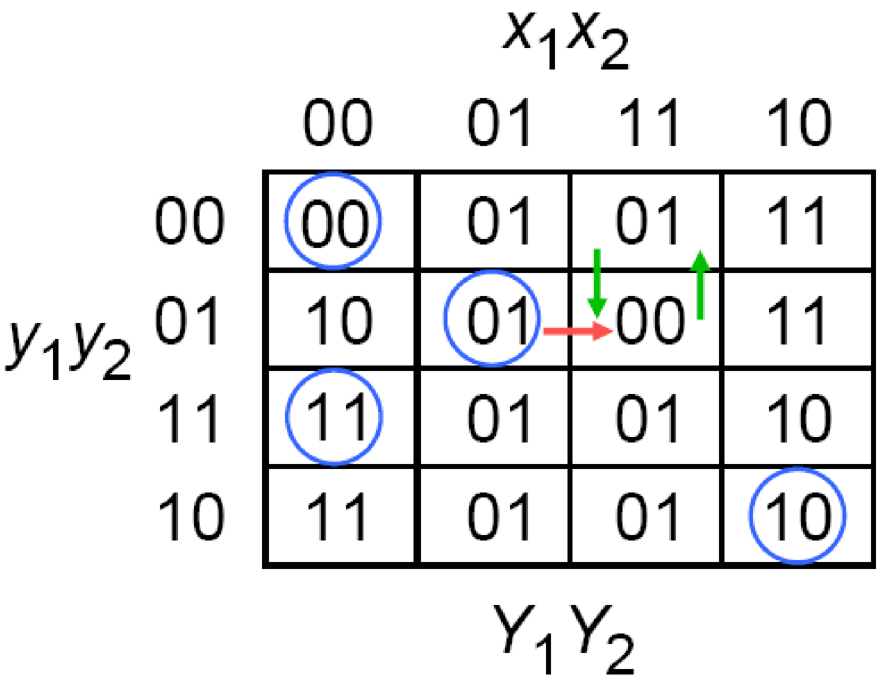
01 → 00 → 01 → 00 → 01 … (unstable)

	x_1x_2			
y_1y_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

Analysis of Asynchronous FSMs

Present state 01, inputs change from 01 to 11



x_1x_2

y_1y_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

01 → 00 → 01 → 00 → 01 … (unstable)

Analysis of Asynchronous FSMs

Present state 01, inputs change from 01 to 00

The Y-map indicates a next state of 10; in this case both state variables change. There are three possibilities: y_1 may change before y_2 giving state 11; y_2 may change before y_1 giving state 00; and the variables may change effectively simultaneously giving state 10

01 → 11 → 11 (stable)
→ 00 → 00 (stable)
→ 10 → 11 → 11 (stable)

Critical race!

Analysis of Asynchronous FSMs

Present state 01, inputs change from 01 to 00

x_1x_2

y_1y_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

01 → 11 → 11 (stable)
→ 00 → 00 (stable)
→ 10 → 11 → 11 (stable)

Critical race!

Analysis of Asynchronous FSMs

Present state 00, inputs change from 00 to 10

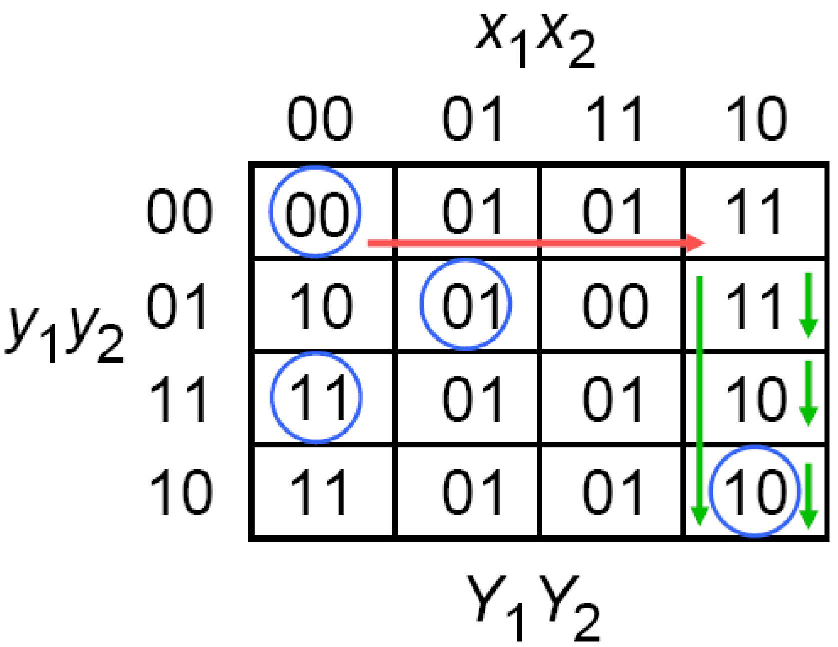
The Y-map indicates a next state of 11; in this case both state variables change. There are three possibilities: y_1 may change before y_2 giving state 10; y_2 may change before y_1 giving state 01; and the variables may change effectively simultaneously giving state 11.

$00 \rightarrow 10 \rightarrow 10$ (stable)
$\rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 10$ (stable)
$\rightarrow 11 \rightarrow 10 \rightarrow 10$ (stable)

Non-critical race!

Analysis of Asynchronous FSMs

Present state 00, inputs change from 00 to 10



x_1x_2

y_1y_2	00	01	11	10
00	00	01	01	11
01	10	01	00	11
11	11	01	01	10
10	11	01	01	10

Y_1Y_2

00 → 10 → 10 (stable)

→ 01 → 11 → 10 → 10 (stable)

→ 11 → 10 → 10 (stable)

Non-critical race!

Flow Table

Using state identifiers: A = 00, B = 01, C = 11, D = 10:

Present state	Next state			
	$X_1=0$ $X_2=0$	$X_1=0$ $X_2=1$	$X_1=1$ $X_2=1$	$X_1=1$ $X_2=0$
A	Ⓐ	B	B	C
B	D	Ⓑ	A	C
C	Ⓒ	B	B	D
D	C	B	B	Ⓓ

A circle around individual states dictates a totally stable state (this will be the same from now on in the notes).

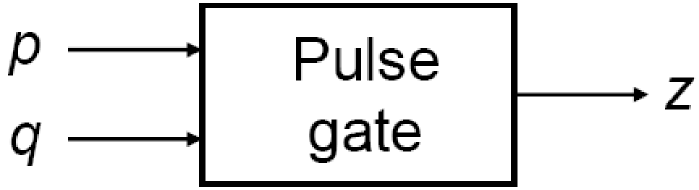
Design of Asynchronous FSMs

1. Construct a primitive flow table
2. Simplify the flow table using the state reduction technique
3. Assign state representations to give transition table
4. Derive the logic from the transition table

A primitive flow table contains *total states*

Total states are states that are stable with one, and only one, combination of input variables

Design Example: A Pulse Gate



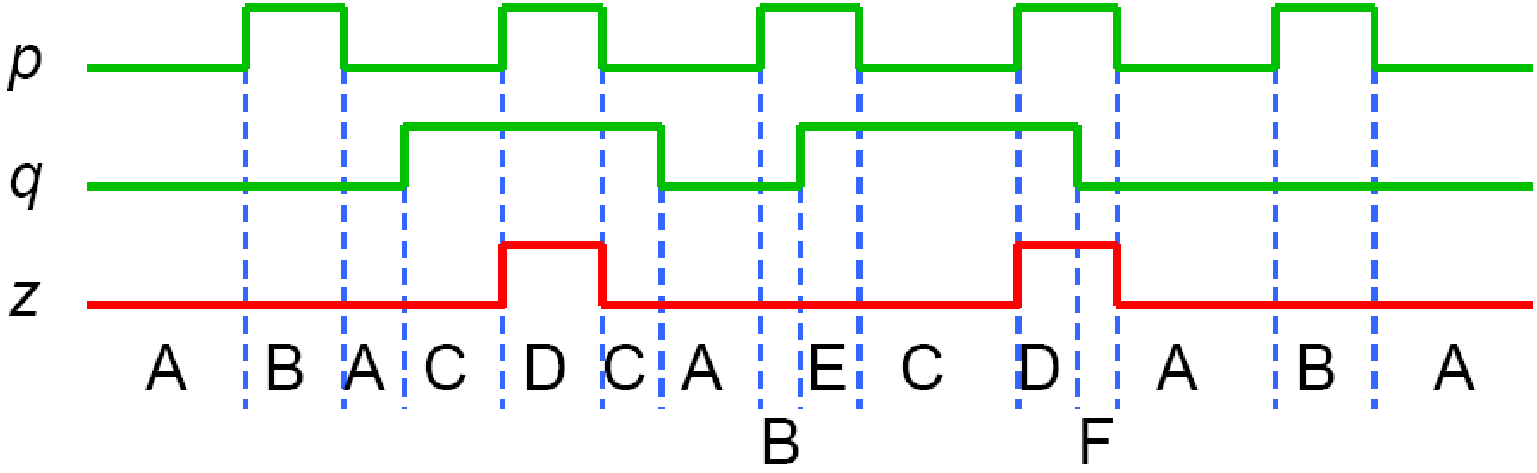
The pulse gate accepts a sequence of pulses on its input *p*

These pulses are gated by an asynchronous input *q*

If *q*=0 then the pulses on the input *p* do not reach the output *z*;
If *q*=1 then the pulses on *p* are transmitted to the output *z*

Only complete pulses must be output

Primitive Flow Table of Pulse Gate

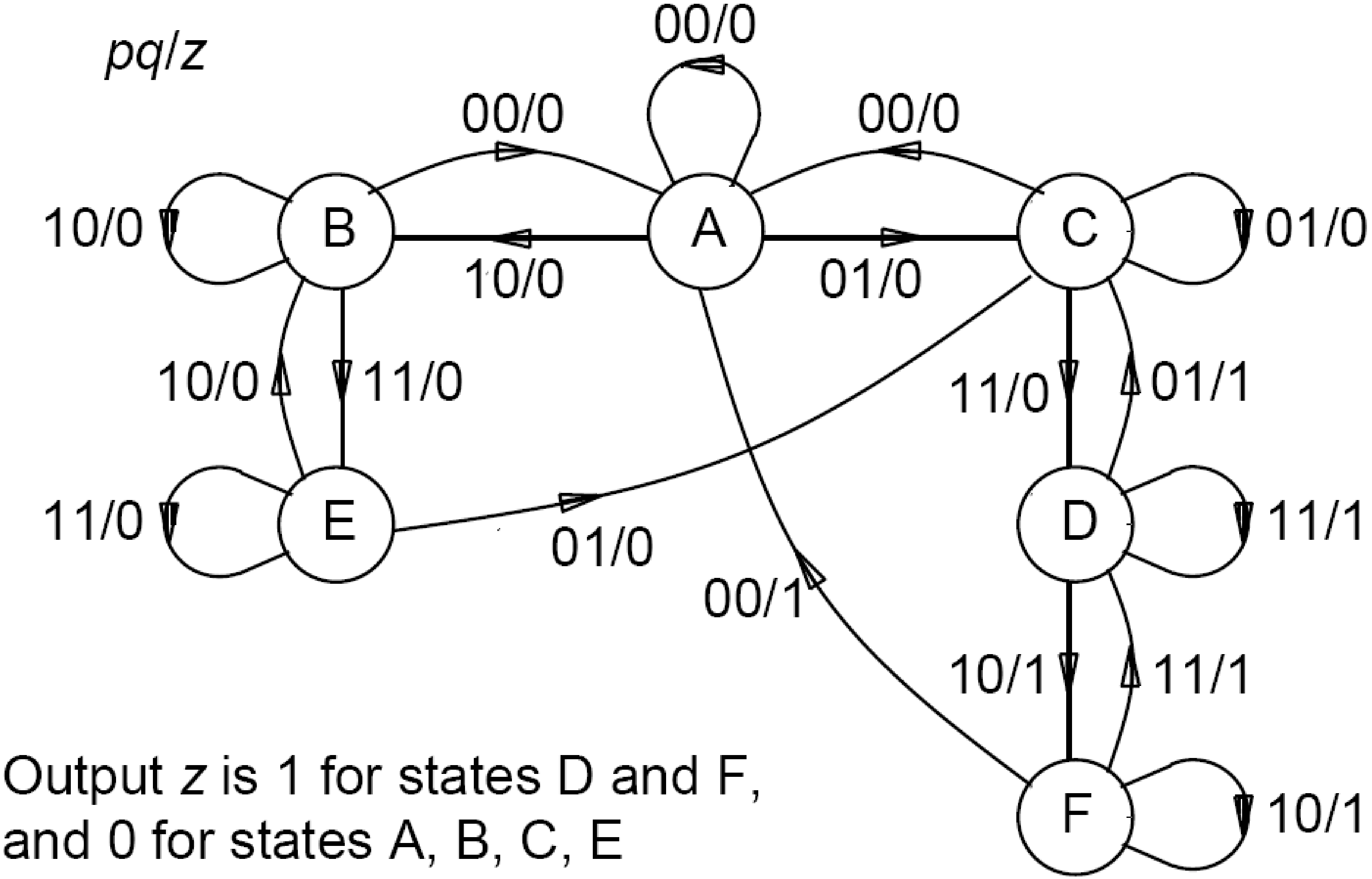


There are 6 total states in this example

Consider the total state A: this is stable as long as *p*=0 and *q*=0.

If *p* goes to 1 then the state changes to B; if *q* goes to 1 then the state changes to C

State Diagram of Pulse gate



Output *z* is 1 for states D and F,
and 0 for states A, B, C, E

Primitive Flow Table of Pulse Gate

Present state	Next state				Output z
	p=0 q=0	p=0 q=1	p=1 q=1	p=1 q=0	
A	Ⓐ	C	-	B	0
B	A	-	E	Ⓑ	0
C	A	Ⓒ	D	-	0
D	-	C	Ⓓ	F	1
E	-	C	Ⓔ	B	0
F	A	-	D	Ⓕ	1

Flow Table State Reduction

The procedure of simplifying the flow table is similar to state reduction for synchronous systems

Equivalent states are identified and combined

Two states are equivalent if they give the same output and go to the same next states for all input combinations

Undefined entries (-) in the primitive flow table are considered to match any of the states

Flow Table State Reduction

Initially the table contains only crosses where states are non-equivalent because their outputs differ

Present state	Next state				Output
	p=0 q=0	p=0 q=1	p=1 q=1	p=1 q=0	z
A	A	C	-	B	0
B	A	-	E	B	0
C	A	C	D	-	0
D	-	C	D	F	1
E	-	C	E	B	0
F	A	-	D	F	1

B					
C					
D	✕	✕	✕		
E				✕	
F	✕	✕	✕		✕
	A	B	C	D	E

Flow Table State Reduction

Then implications of states being equivalent are entered in the table

B					
C		D ≡ E			
D	×	×	×		
E			D ≡ E	×	
F	×	×	×		×
	A	B	C	D	E

Present state	Next state				Output z
	p=0 q=0	p=0 q=1	p=1 q=1	p=1 q=0	
A	Ⓐ	C	-	B	0
B	A	-	E	Ⓑ	0
C	A	Ⓒ	D	-	0
D	-	C	Ⓓ	F	1
E	-	C	Ⓔ	B	0
F	A	-	D	Ⓕ	1

Flow Table State Reduction

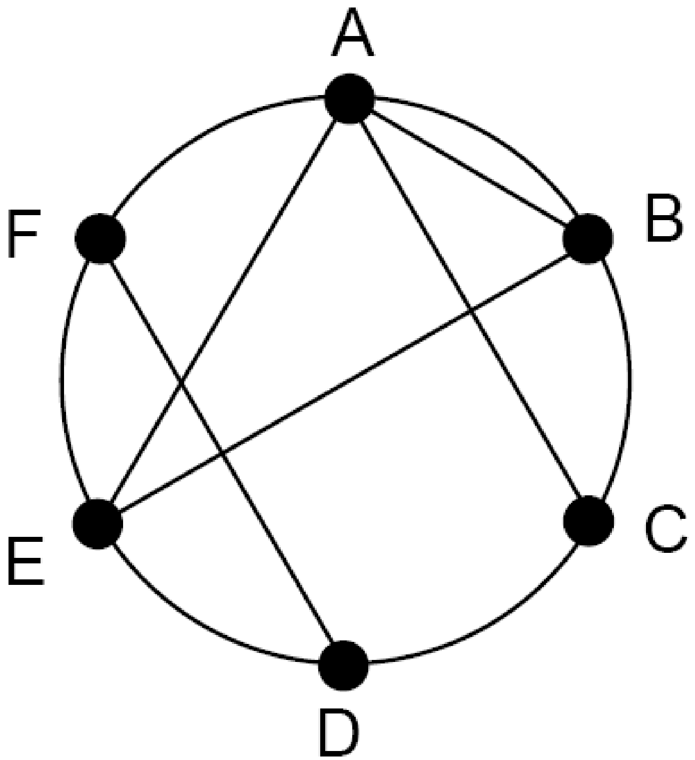
It is not possible for D≡E because this combination is marked with a cross. Thus B≠C and C≠E

B					
C		✗			
D	✗	✗	✗		
E			✗	✗	
F	✗	✗	✗		✗
	A	B	C	D	E

Flow Table State Reduction

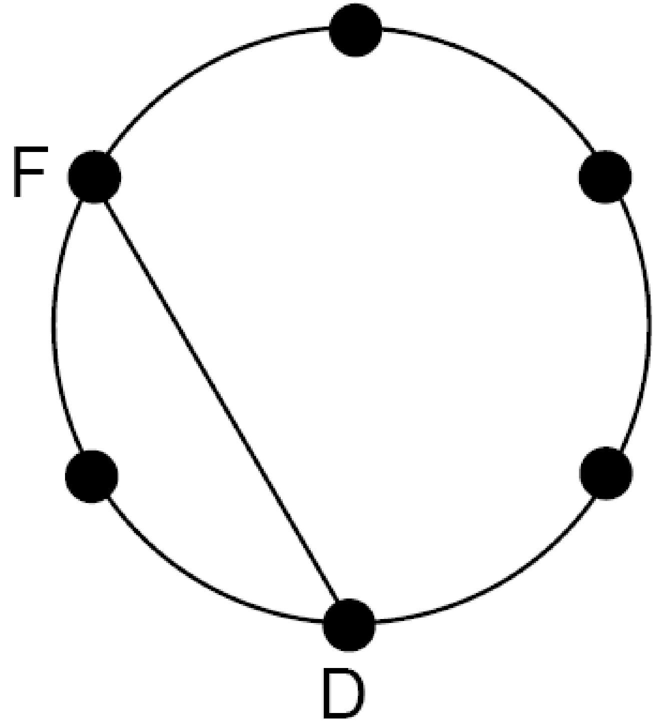
It is possible for some, but not necessarily all, of the following pairs to be equivalent:

A≡B   A≡C   A≡E   B≡E   D≡F



Flow Table State Reduction

A, B and E form a set of three equivalent states; removing these states gives:



F and D form another set of equivalent states, leaving C without equivalent. Thus:

$$A \equiv B \equiv E \qquad D \equiv F$$

Flow Table State Reduction

Merging equivalent states: A≡B≡E D≡F on the primitive flow table gives:

Present state	Next state				Output z
	p=0 q=0	p=0 q=1	p=1 q=1	p=1 q=0	
A	Ⓐ	C	Ⓐ	Ⓐ	0
C	A	Ⓒ	D	-	0
D	A	C	Ⓓ	Ⓓ	1