Asynchronous sequence circuits

- An asynchronous sequence machine is a sequence circuit without flip-flops
- Asynchronous sequence machines are based on combinational gates with feedback

<u>Upon analysis it is assumed </u>: Only one signal at a time in the gate circuit can change its value at any time

Golden rule

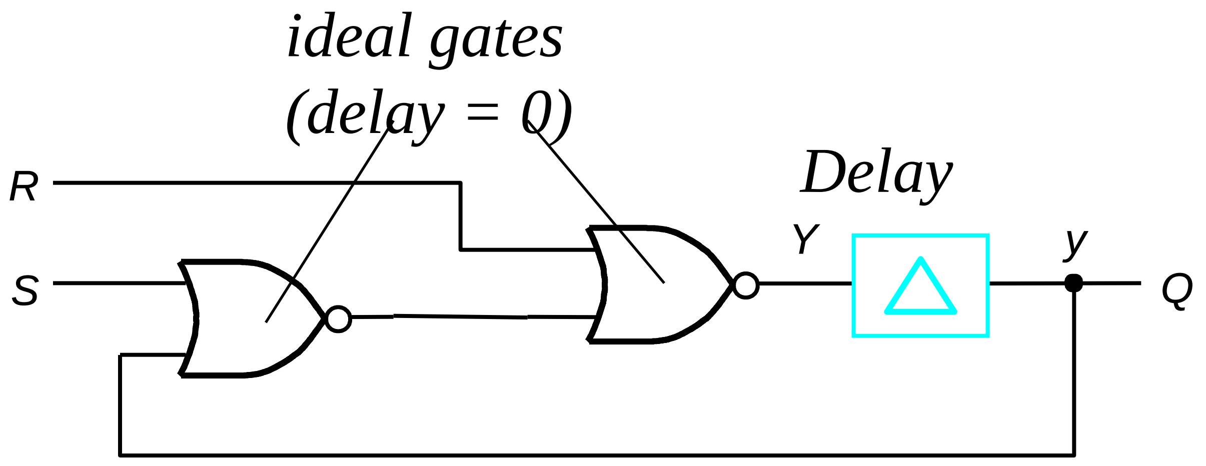
Asynchronous state machine

Asynchronous state machines are used when it is necessary to maintain a state, but when there is no clock available.

- All flip-flops and latches are themselfes asynchronous state machines
- They are useful to synchronize events in situations where metastability is/can be a problem

SR-latch with NOR-gates

To analyze the behavior of an asynchronous circuit one assumes ideal gates and summarizes all the delay to a single block with delay **Δ**.

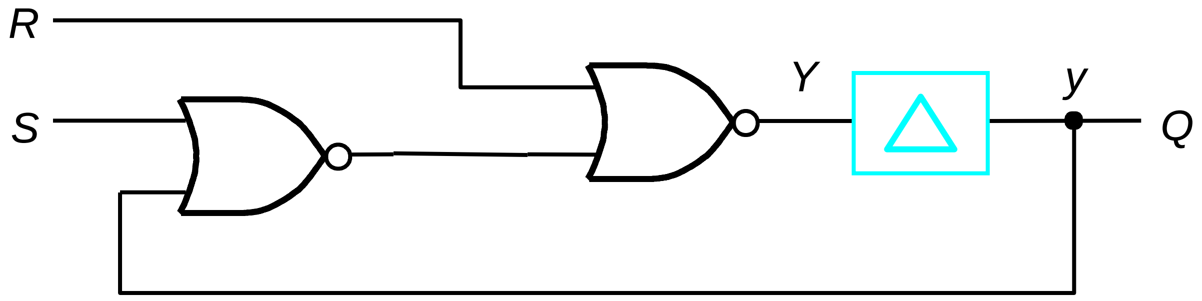


Analysis of sequence circuits

By having a **delay block** we can consider

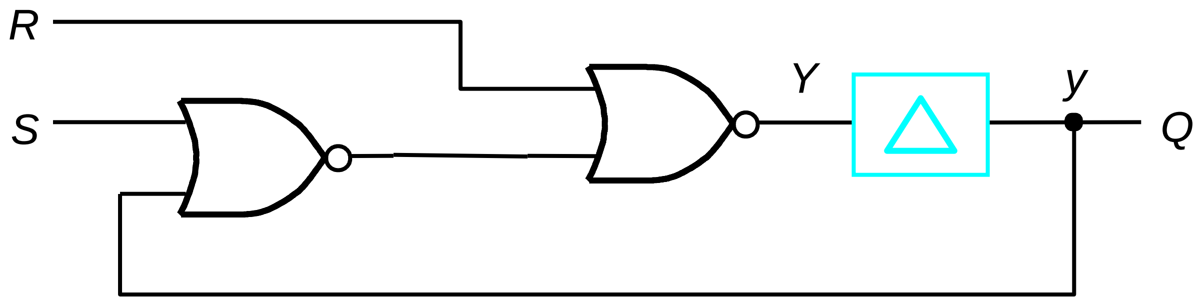
y as the present state

Y as next state



State function

Thus, we can develop a functional relationship of the next state ***Y*** depending on the input signals ***S*** and ***R*** and the current state ***y***



$$Y = \overline{R + \overline{(S + y)}}$$

State table

BV uses binary code

From statefunction to truth table

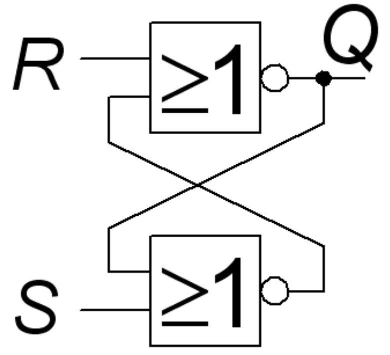
$$Y = \overline{R + (\overline{S + y})}$$

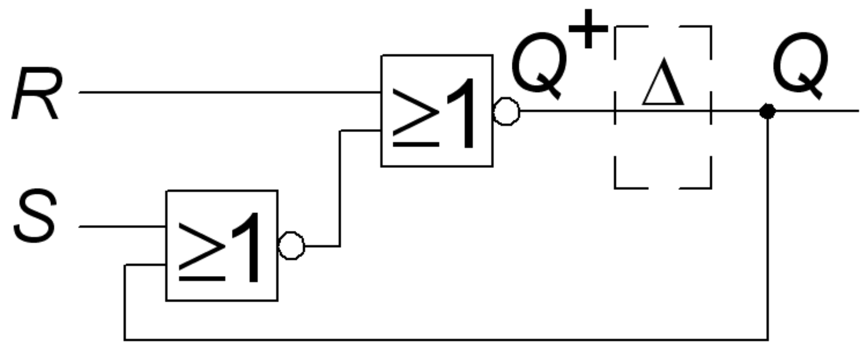
y	S	R	$Y = \overline{R + (\overline{S + y})}$
0	0	0	$0 = \overline{0 + (\overline{0 + 0})}$
0	0	1	$0 = \overline{1 + (\overline{0 + 0})}$
0	1	0	$1 = \overline{1 + (\overline{1 + 0})}$
0	1	1	$0 = \overline{1 + (\overline{1 + 0})}$
1	0	0	$1 = \overline{0 + (\overline{0 + 1})}$
1	0	1	$0 = \overline{1 + (\overline{0 + 1})}$
1	1	0	$1 = \overline{0 + (\overline{1 + 1})}$
1	1	1	$0 = \overline{1 + (\overline{1 + 1})}$

Present state y	*Next state*			
	SR = 00	*01*	*10*	*11*
	Y	*Y*	*Y*	*Y*
0	*0*	*0*	*1*	*0*
1	*1*	*0*	*1*	*0*

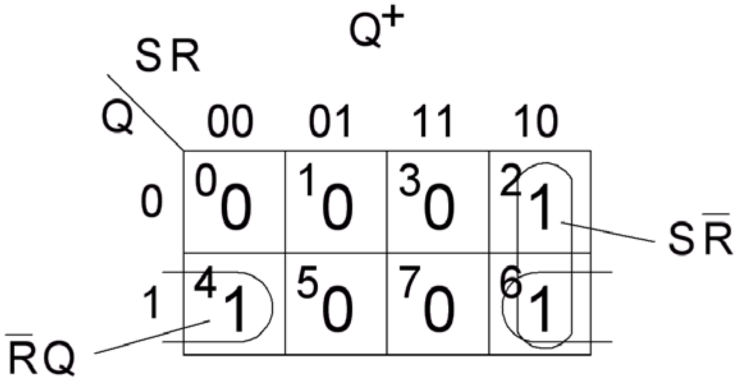
Or, as in the exercise - using the Karnaugh map …

(at exercise, analysis of SR)





$$Q^+ = \overline{R + \overline{S + Q}} = \overline{R} \cdot \overline{\overline{(S + Q)}} = \overline{R} \cdot (S + Q) = S\overline{R} + \overline{R}Q$$



Present state *Q*	Next state Q^+			
	Input signals SR			
	00	01	11	10
0	0	0	0	1
1	1	0	0	1

For binary order

Stable states

Present state *y*	*Next state*			
	SR = 00	*01*	*10*	*11*
	Y	*Y*	*Y*	*Y*
0	*0*	*0*	*1*	*0*
1	*1*	*0*	*1*	*0*

- Since we do not have flip-flops, but only combinational circuits, a state change can result in additional state changes
- A state is
  - stable if $Y(t) = y(t + \Delta)$
  - unstable if $Y(t) \neq y(t + \Delta)$

$\boxed{Y = y}$ stable

Exitation table

The asynchronous coded state table is called **Excitation table**

The stable states (those with next state = present state) will be ”encircled”

Present state y	*Next state*			
	$SR = 00$	01	10	11
	Y	Y	Y	Y
0	$\textcircled{0}$	$\textcircled{0}$	1	$\textcircled{0}$
1	$\textcircled{1}$	0	$\textcircled{1}$	0

$$Y = y$$

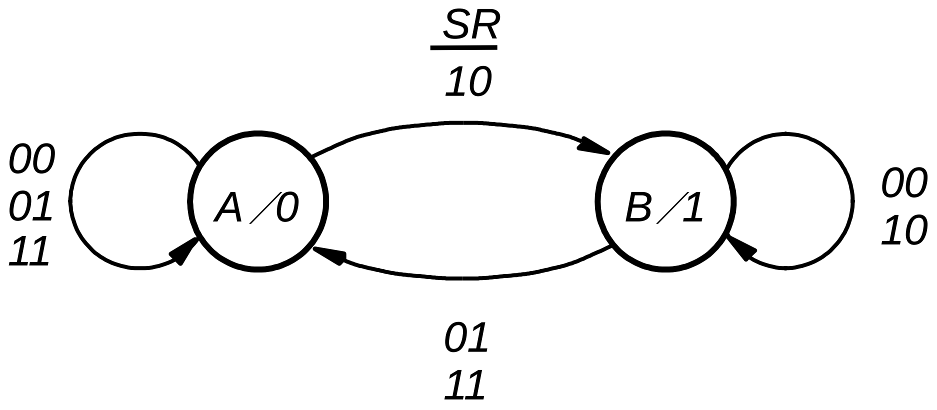
Terminology

When dealing with asynchronous sequential circuits a different terminology is used

- The asynchronous uncoded state table is called **flow table**

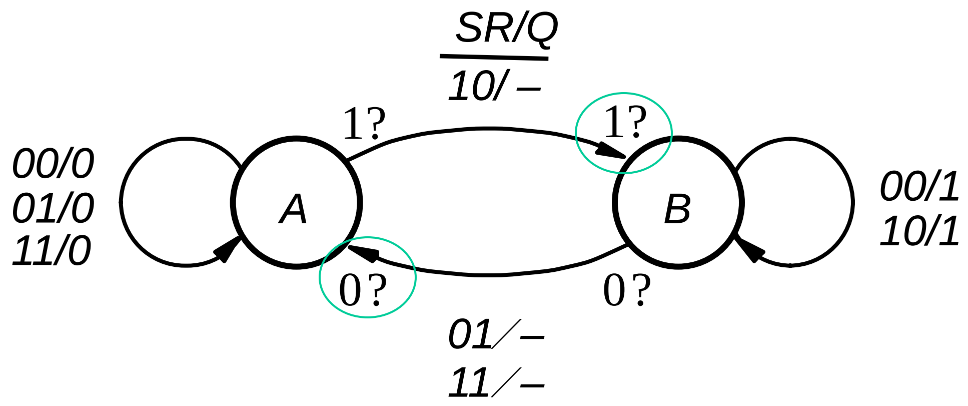
Flowtable and Statediagram (Moore type)

Present state	*Next state*				*Output Q*
	SR = 00	*01*	*10*	*11*	
A	Ⓐ	Ⓐ	*B*	Ⓐ	*0*
B	Ⓑ	*A*	Ⓑ	*A*	*1*



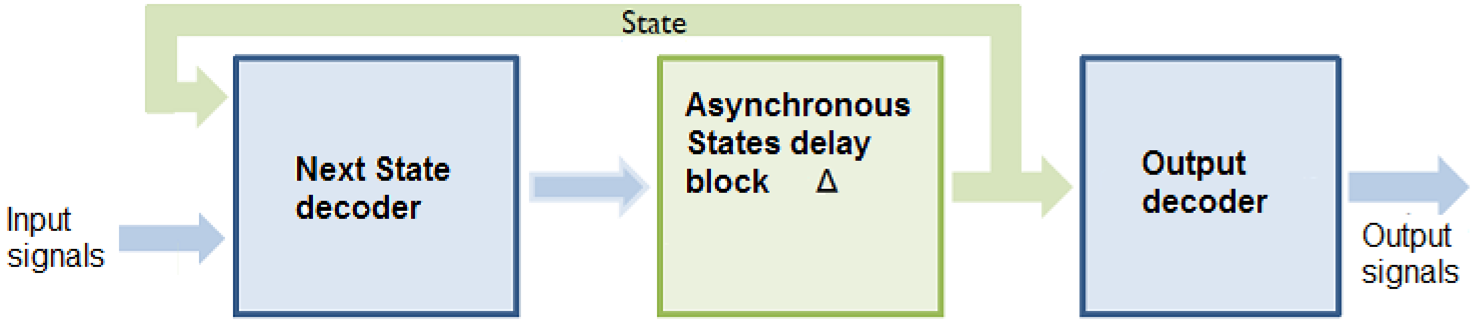
Flowtable and Statediagram (Mealy type)

Present state	Next state				Output, Q			
	SR = 00	01	10	11	00	01	10	11
A	Ⓐ	Ⓐ	B	Ⓐ	0	0	–	0
B	Ⓑ	A	Ⓑ	A	1	–	1	–



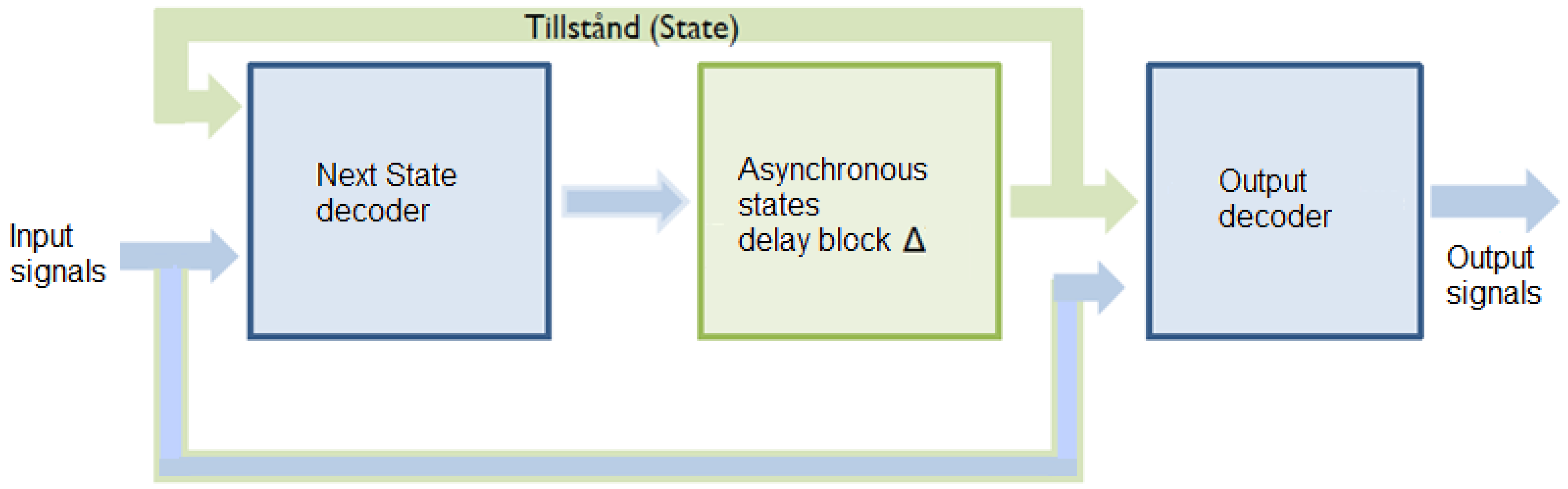
Don’t care (‘-’) has been selected for the output decoder. It does not matter if the output is changed before or after the state transition (= simpler gate array).

Asynchronous Moore compatible



- Asynchronous sequential circuits have similar structure as synchronous sequential circuits
- Instead of flip-flops one have a "delay block"

Asynchronous Mealy compatible



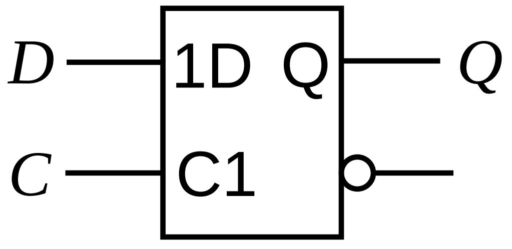
- Asynchronous sequential circuits have similar structure as synchronous sequential circuits
- Instead of flip-flops one have a "delay block"

Analysis of asynchronous circuits

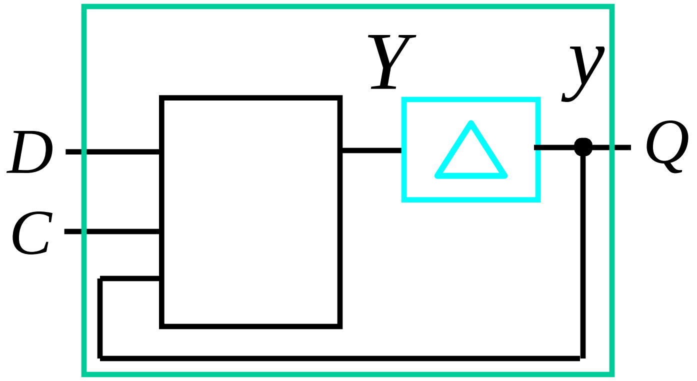
The analysis is done in the following steps :

1) Replace the feedbacks in the circuit with delay element Δ_i. Input signal to delay-element forms the next state Y_i, while the output signal y_i represents the present state.
2) Find out the next-state and output expressions
3) Set up the corresponding excitationstable
4) Create a flow table by replacing the encoded states by symbolic states
5) Draw a state diagram if needed

First: D-latch state function



$$C = follow/\overline{latch}$$

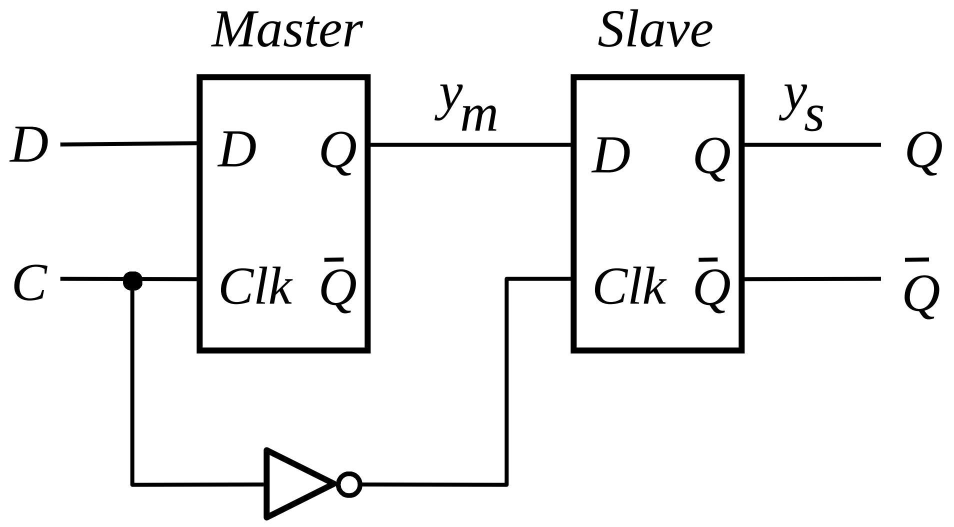


D-latch statefunction. Functional relationship between the current state *y* and next state *Y*

$$Y = \underset{\substack{\uparrow \\ follow}}{D \cdot C} + \underset{\substack{\uparrow \\ \overline{latch}}}{y \cdot \overline{C}}$$

Exemple: Master-Slave-flip-flop

Master-slave D flip-flop is constructed from two asynchronous D-latches.



State expression:

$$Y_m = D \cdot C + y_m \cdot \overline{C}$$

$$Y_s = y_m \cdot \overline{C} + y_s \cdot C$$

Exitationstable

From the expressions one can directly derive the excitation table (if you can keep it all in your head?)

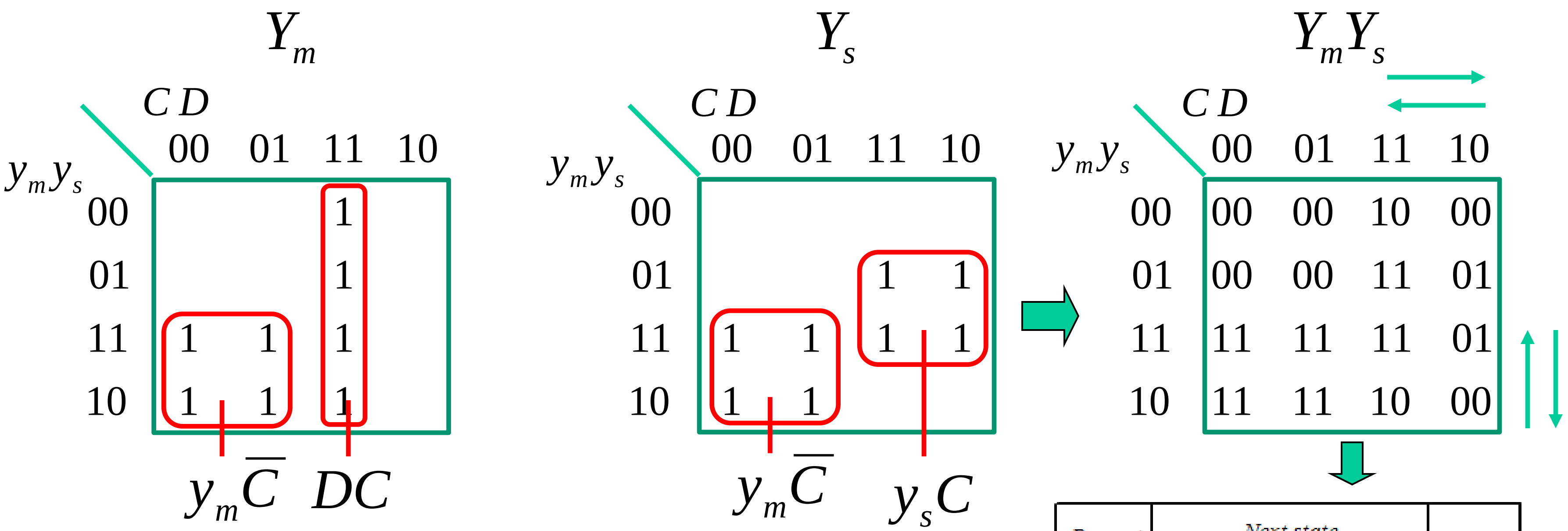
$$Y_m = D \cdot C + y_m \cdot \overline{C}$$

$$Y_s = y_m \cdot \overline{C} + y_s \cdot C$$

⇨

Present state	*Next state*				*Output*
	CD = 00	*01*	*10*	*11*	
$y_m\, y_s$	$Y_m\, Y_s$				Q
00	$\textcircled{00}$	$\textcircled{00}$	$\textcircled{00}$	*10*	*0*
01	*00*	*00*	$\textcircled{01}$	*11*	*1*
10	*11*	*11*	*00*	$\textcircled{10}$	*0*
11	$\textcircled{11}$	$\textcircled{11}$	*01*	$\textcircled{11}$	*1*

or with help from K-map …



$$Y_m = D\cdot C + y_m\cdot\overline{C} \qquad Y_s = y_m\cdot\overline{C} + y_s\cdot C$$

Present state	Next state				Output
	CD = 00	01	10	11	
$y_m\ y_s$	$Y_m\ Y_s$				Q
00	00	00	00	10	0
01	00	00	01	11	1
10	11	11	00	10	0
11	11	11	01	11	1

Change rows and colums to get the binary order as in BV

Flow table

We define four states S1, S2, S3, S4, which gives us the flow table

Present state	Next state				Output
	CD = 00	01	10	11	
$y_m\ y_s$	$Y_m\ Y_s$				Q
00	00	00	00	10	0
01	00	00	01	11	1
10	11	11	00	10	0
11	11	11	01	11	1

Present state	Nextstate				Output Q
	CD = 00	01	10	11	
S1	S1	S1	S1	S3	0
S2	S1	S1	S2	S4	1
S3	S4	S4	S1	S3	0
S4	S4	S4	S2	S4	1

Flow table

***Remember**: Only one input can be changed at a time*

- *Thus, some transitions will never be able to happen!*

Present state	Next state				Output Q
	CD = 00	01	10	11	
S1	$\textcircled{S1}$	$\textcircled{S1}$	$\textcircled{S1}$	S3	0
S2	S1	S1	$\textcircled{S2}$	S4	1
S3	S4	S4	S1	$\textcircled{S3}$	0
S4	$\textcircled{S4}$	$\textcircled{S4}$	S2	$\textcircled{S4}$	1

Flowtable – impossible transitions

Present state	Nextstate CD = 00	01	10	11	Output Q
S1	$\textcircled{S1}$	$\textcircled{S1}$	$\textcircled{S1}$	S3	0
S2	S1	S1	$\textcircled{S2}$	S4	1
S3	~~S4~~	S4	S1	$\textcircled{S3}$	0
S4	$\textcircled{S4}$	$\textcircled{S4}$	S2	$\textcircled{S4}$	1

State S3

Only stable state for **S3** is when input is 11

Only one input at a time can change → possible changes are 11 → 01, 11 → 10

- Theese combinations leaves S3!
- Input 00 in S3 is *not* possible!
- Input 00 is therefore **don’t care**!

Flowtable – impossible transitions

Present state	Nextstate				Output Q
	CD = 00	01	10	11	
S1	(S1)	(S1)	(S1)	S3	0
S2	S1	–	(S2)	S4	1
S3	–	S4	S1	(S3)	0
S4	(S4)	(S4)	S2	(S4)	1

State S2

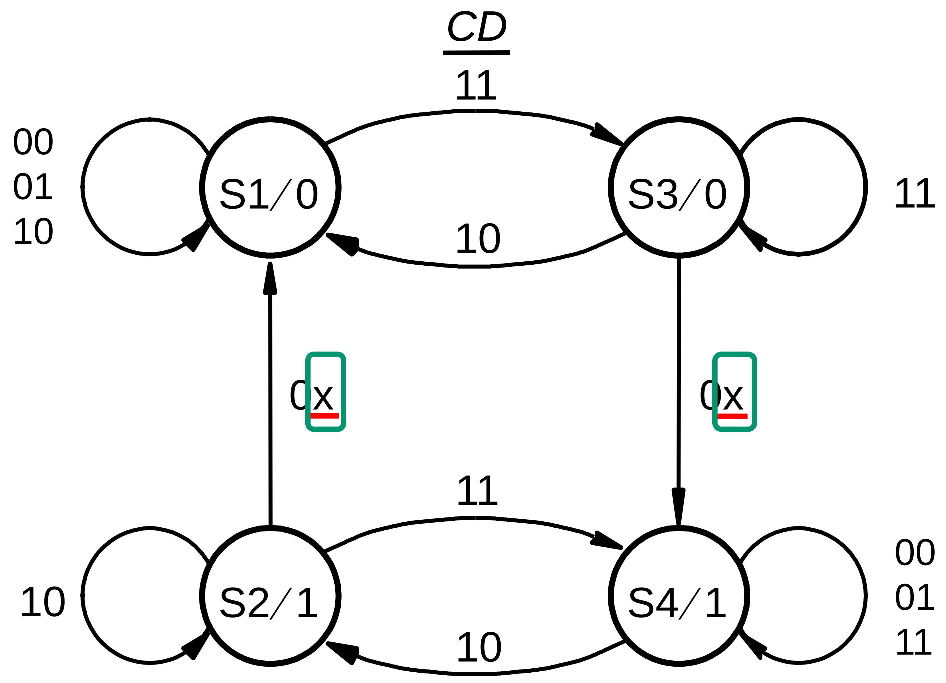
Only stable state for **S2** is when input is 10

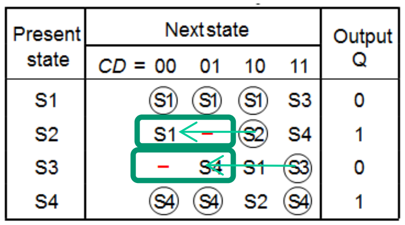
Only one input at a time can change → possible changes are 10 → 11, 10 → 00

- Theese combinations leave S2!
- Input 01 in S2 is *not* possible!
- Input 01 is therefore **don’t care**!

D-flip-flop state diagram

Don't care is here denoted by x





Present state	Next state				Output
	CD = 00	01	10	11	Q
S1	S1	S1	S1	S3	0
S2	S1	–	S2	S4	1
S3	–	S4	S1	S3	0
S4	S4	S4	S2	S4	1

Don't care can be used to simplify the circuit (the next state decoder).

Synthesis of asynchronous circuits

The synthesis is carried out in the following steps :

1) Create a **state diagram** acording to the functional description

2) Create a flow table and reduce the number of states if possible

3) Assign codes to the states and create the **excitationstable**

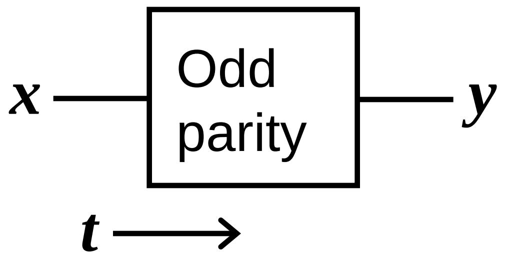
4) Develop expressions (transfer functions) for next state and outputs

5) Design a circuit that implements the above expressions

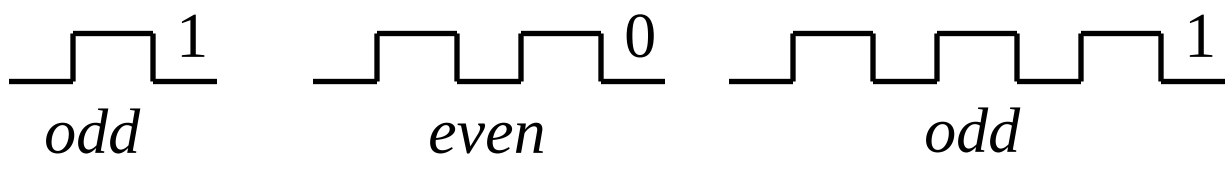
Exemple: serial paritety circuit

Input ***x*** Output ***y***

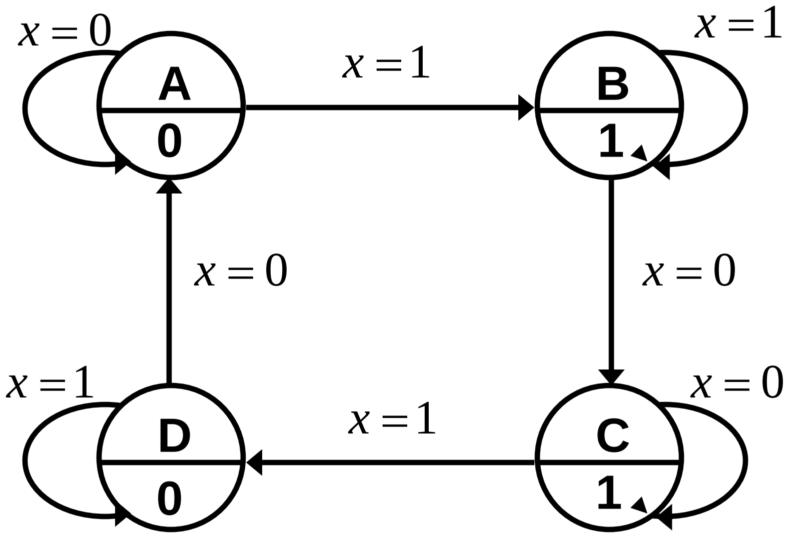
y = 1 if the number of pulses at input ***x*** is an odd number.

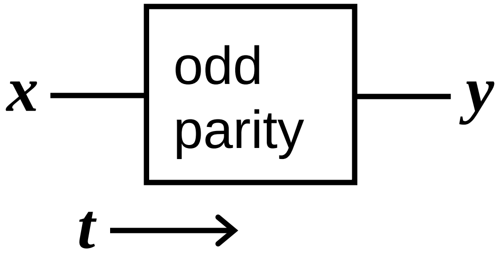


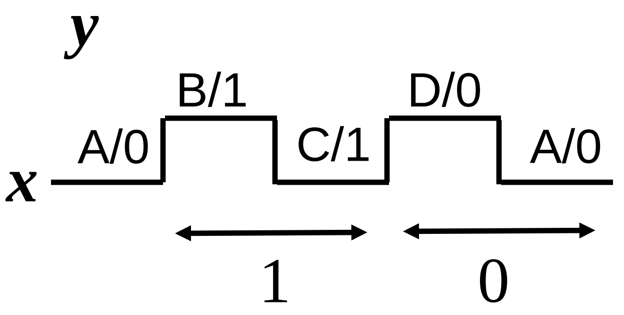
In other words, an "every other time" circuit …



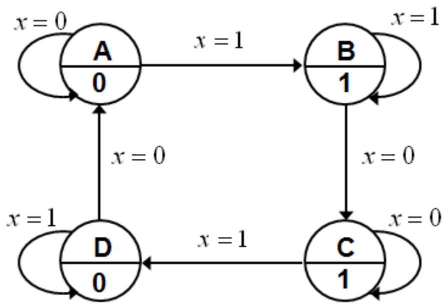
Create state diagram

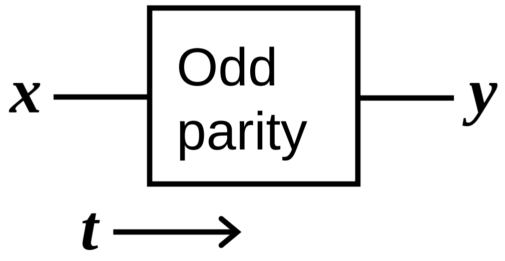






Create state table





Pres state	Next State		Q
	X=0	1	
A	A	B	0
B	C	B	1
C	C	D	1
D	A	D	0

What is a good state encoding?

00, 01, 10, 11 - binary code?

Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	10	01	1
10	10	11	1
11	00	11	0

Bad encoding (HD=2!)

- *Suppose*

$X = 1 \;\; Y_2Y_1 = 11$

- *Then*

$X \rightarrow 0 \rightarrow Y_2Y_1 = 00?$

$11 \rightarrow 10!$

$11 \rightarrow 01 \rightarrow 10! \quad ? \rightarrow 00$

We will never reach 00?

What is a good state encoding?

00, 01, 11, 10 – gray code

- *Suppose*

$X = 1 \quad Y_2Y_1 = 10$

- *Then*

$X \rightarrow 0 \rightarrow Y_2Y_1 = 00$

$10 \rightarrow 00$

Pres state	Next State		Q
	$X=0 \leftarrow$	1	
y_2y_1	Y_2Y_1		
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0

Good encoding (HD=1)

State encoding

Richard Hamming

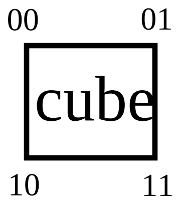
- In asynchronous sequential circuits,
  it is impossible to guarantee that the two state variables changes values simultaneously
  - Thus, a transition 00 → 11 could result in
    - A transition 00 → 01 → ???
    - A transition 00 → 10 → ???
- To ensure the function all state transitions **<u>MUST</u>** have the ***Hamming distans* 1**
  - The Hamming distans is the number of bits that differs in two binary numbers
    - Hamming distans between 00 and 11 is 2
    - Hamming distans between 00 and 01 is 1

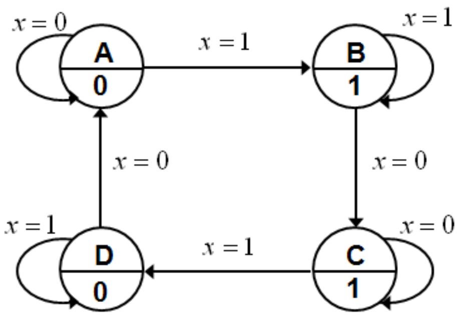
Good state encoding

- Procedure to obtain good codes:
  1) Draw the transition diagram along the edges in hypercubes (Gray code) formed by the codes
  2) Remove any crossing lines by
     a) change the position of two adjacent nodes
     b) utilize available unused codes
        (exploit unstable conditions)
     c) introduce hypercube of more dimensions

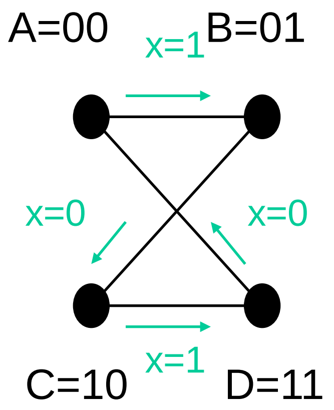
Poor coding of the parity circuit

The poor state encoding





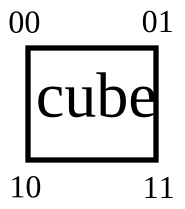
Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
A 00	00	01	0
B 01	10	01	1
C 10	10	11	1
D 11	00	11	0

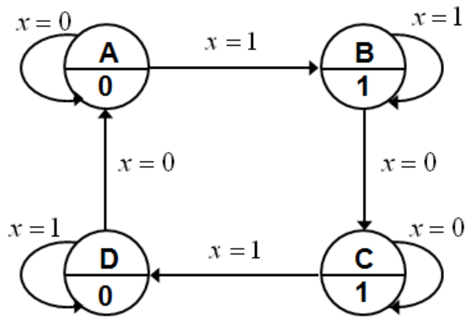


Poor encoding –
Hamming Distance = 2
(**crossing lines**)

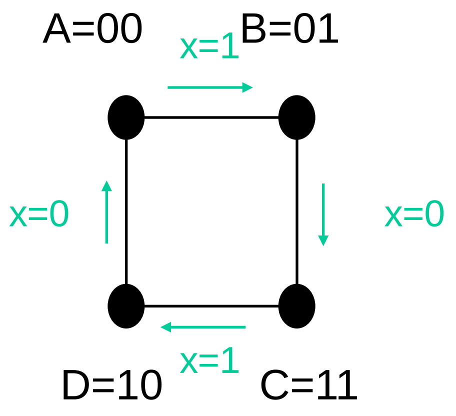
Good coding of the parity circuit

The good state encoding





Pres state	Next State		Q
	X=0	1	
y_2y_1	Y_2Y_1		
A 00	00	01	0
B 01	11	01	1
C 11	11	10	1
D 10	00	10	0

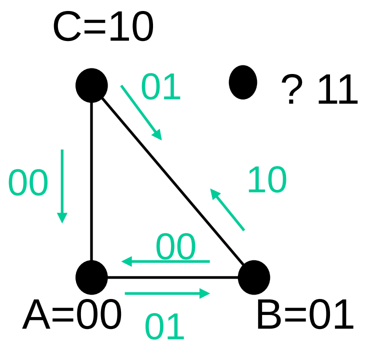


Good encoding
Hamming Distance = 1
(**no crossing lines**)

Problems with non-stable states

Ex. an other circuit:

Present state	Nextstate $r_2r_1 =$ 00	01	10	11	Output g_2g_1
A 00	Ⓐ	B	C	–	00
B 01	A	Ⓑ	C	Ⓑ	01
C 10	A	B	Ⓒ	Ⓒ	10

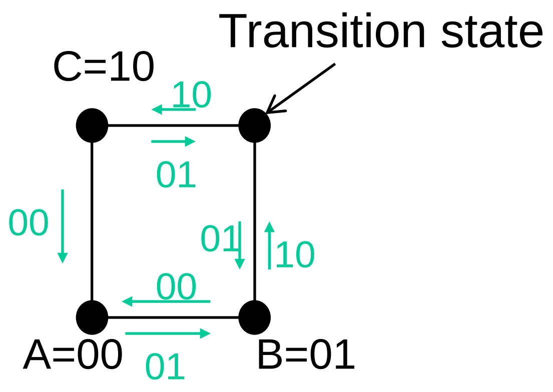


Bad encoding

At the transition between **B** to **C** (or **C** to **B**) is the Hamming distans 2 (10↔01)!
Chance to get stuck in an unspecified state (with the code 11)!

Solution to unstable state

- **Solution**: The introduction of a transition state that ensure that you do not end up in an undefined state!



Good encoding

	Present state	Nextstate				Output
		$r_2r_1 =$ 00	01	11	10	
	y_2y_1	Y_2Y_1				g_2g_1
A	00	00	01	–	10	00
B	01	00	01	01	11	01
-	11	–	01	–	10	--
C	10	00	11	10	10	10

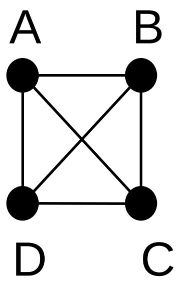
$$01 \rightarrow 11 \rightarrow 10$$

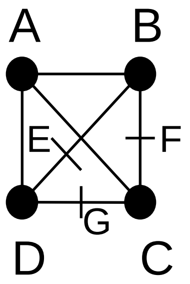
$$10 \rightarrow 11 \rightarrow 01$$

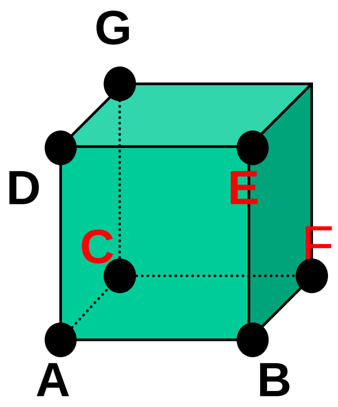
Transition state

Extra states – more dimensions

- One can increase the number of dimensions in order to implement secure state transitions



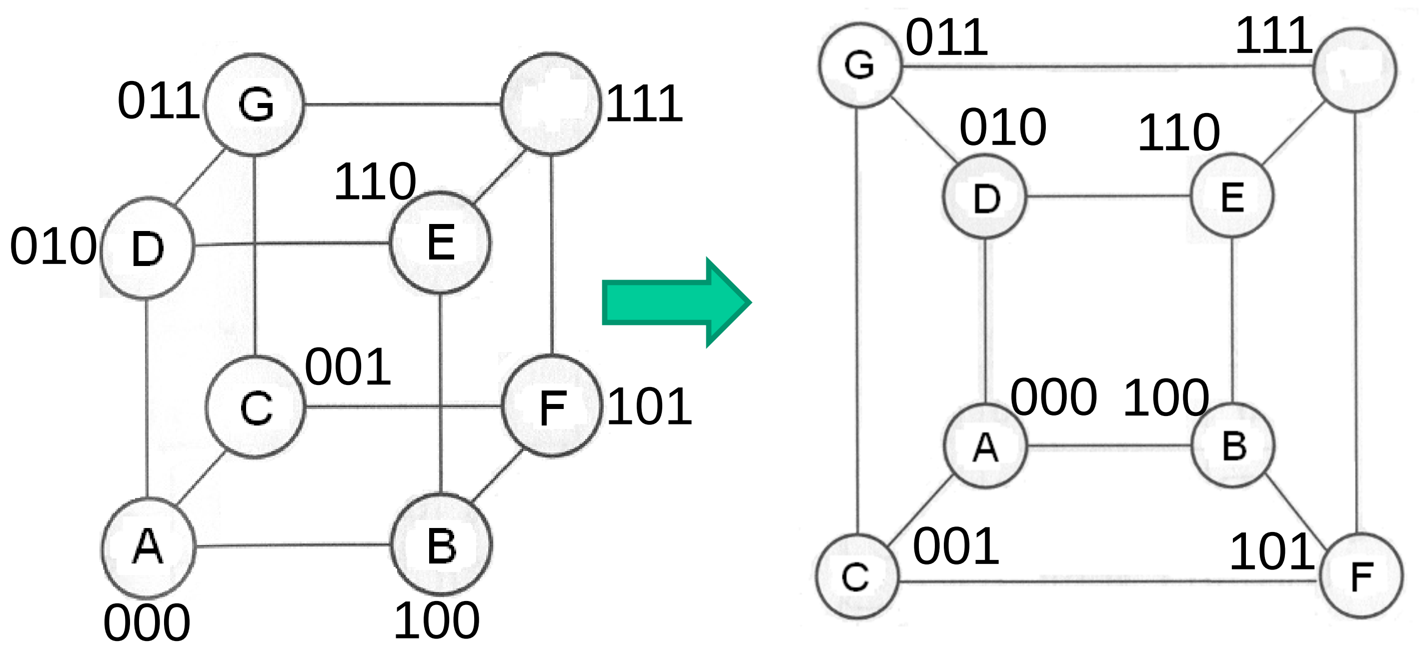




If there is no way redraw the chart to HD = 1 you may add states by increasing the dimension of the hypercube. You then drag the transitions through the then available non-stable states.

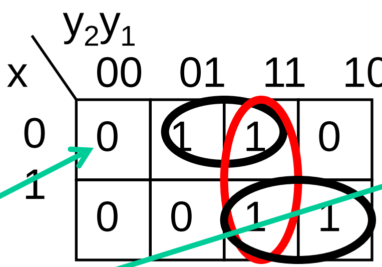
Extra states – more dimensions

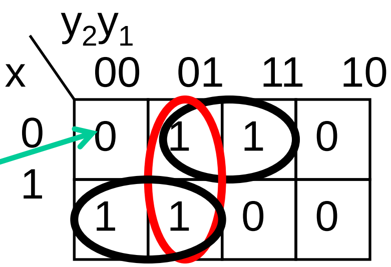
- It's easier to draw a "flat" 3D cube (perspective, is then from the front)



Karnaugh maps

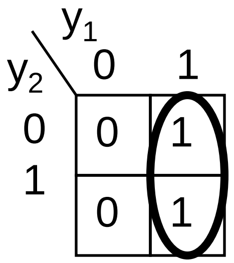
Pres state y2y1	Next State Y2Y1 X=0	1	Q
00	00	01	0
01	11	01	1
11	11	10	1
10	00	10	0





$$Y_2 = \bar{x}y_1 + \boxed{y_2y_1} + xy_2$$

$$Y_1 = x\bar{y}_2 + \boxed{\bar{y}_2y_1} + \bar{x}y_1$$



$$Q = y_1$$

Groupings in red are to avoid Hazard (see later in course)!

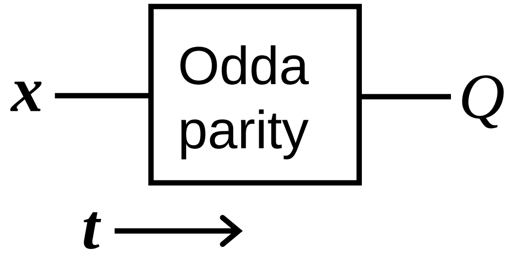
The complete circuit

x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y_2 = \overline{x}y_1 + y_2y_1 + xy_2$$

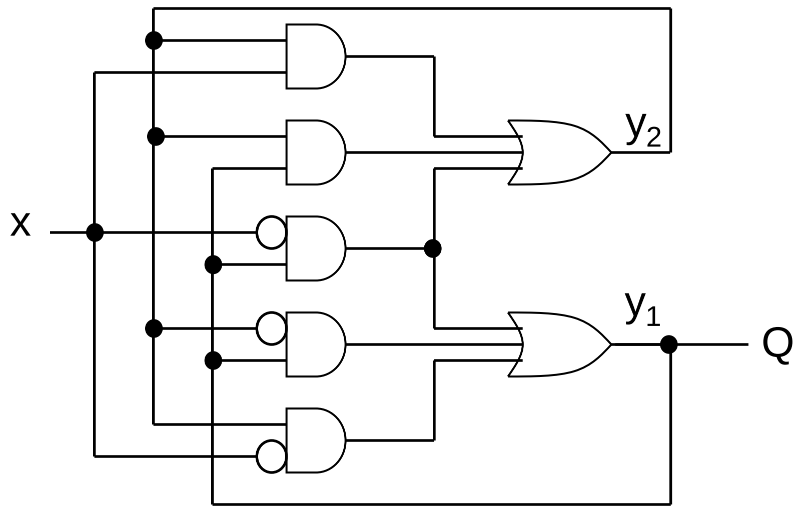
y_2 \ y_1	0	1
0	0	1
1	0	1

$$Q = y_1$$

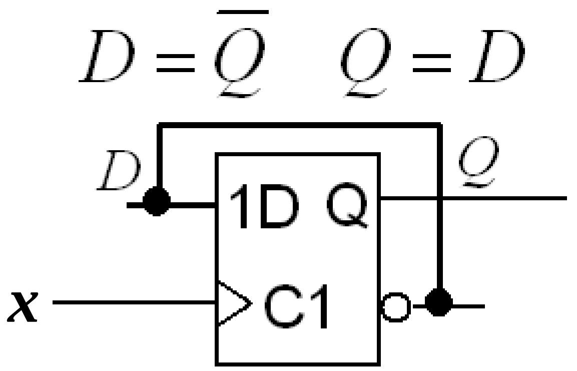


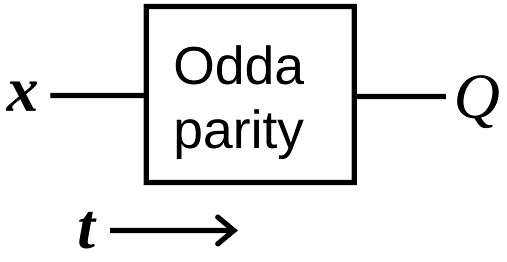
x \ y_2y_1	00	01	11	10
0	0	1	1	0
1	1	1	0	0

$$Y_1 = x\overline{y}_2 + \overline{y}_2y_1 + \overline{x}y_1$$



(easier with D-flip-flop)



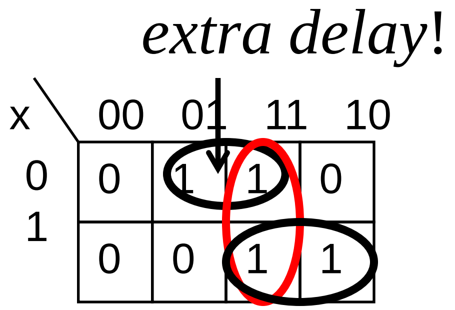


We have made an "every other time" earlier in the course. Then with a D flip-flop. But now it was more exiting!

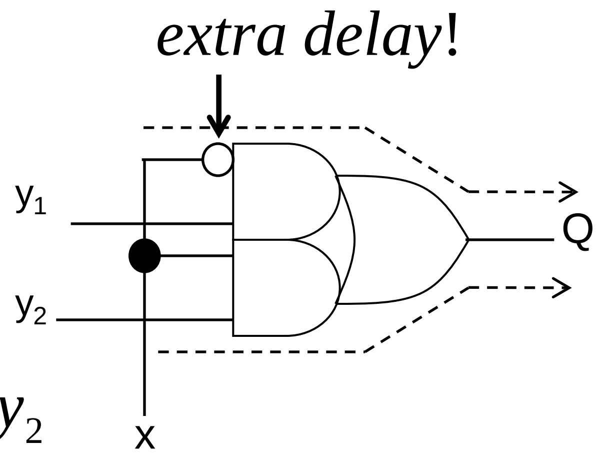
What is Hazard?

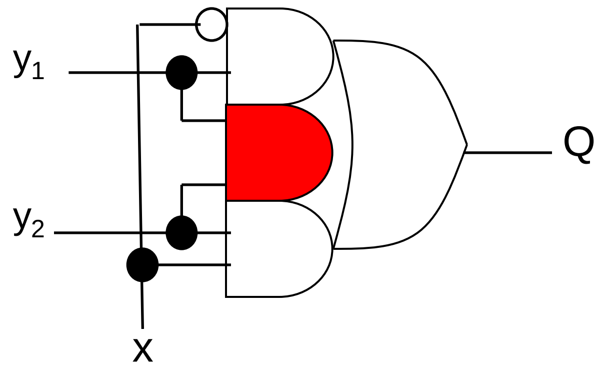
- Hazard is a term that means that there is a danger that the output is not stable, but it may “flicker” at certain input combinations.
- Hazard occurs if there is a different distance from the various inputs to an output, there will be an signal-race.
- In order to counteract this, one must add the prime implikants to cover up the dangerous transition.

Exemple of Hazard – MUX



$$Y_2 = \bar{x}y_1 + \boxed{y_2y_1} + xy_2$$





At the transition from xy_2y_1=(111) $\rightarrow$ (011) the output *Q* could ***flicker***, because the road from *x* to *Q* are longer via the upper AND-gate than the lower (race).

MORE ABOUT HAZARD IN THE NEXT LECTURE!

State Minimizing

Asynchronous state machines has many "unspecified" positions in the flow table that can be exploited to minimize the number of states.

The probability that less number of states leads to a simpler implementation is high in the case of asynchronous circuits!

State Minimizing

Two steps:

Equivalency - equivalent state. The same steps as the state minimization of synchronous sequential circuits, full flexibility remain.

Compatibility - compatible states will be different for Moore or Mealy compliant realization, the choices you make now affect the future flexibility.

State Minimizing

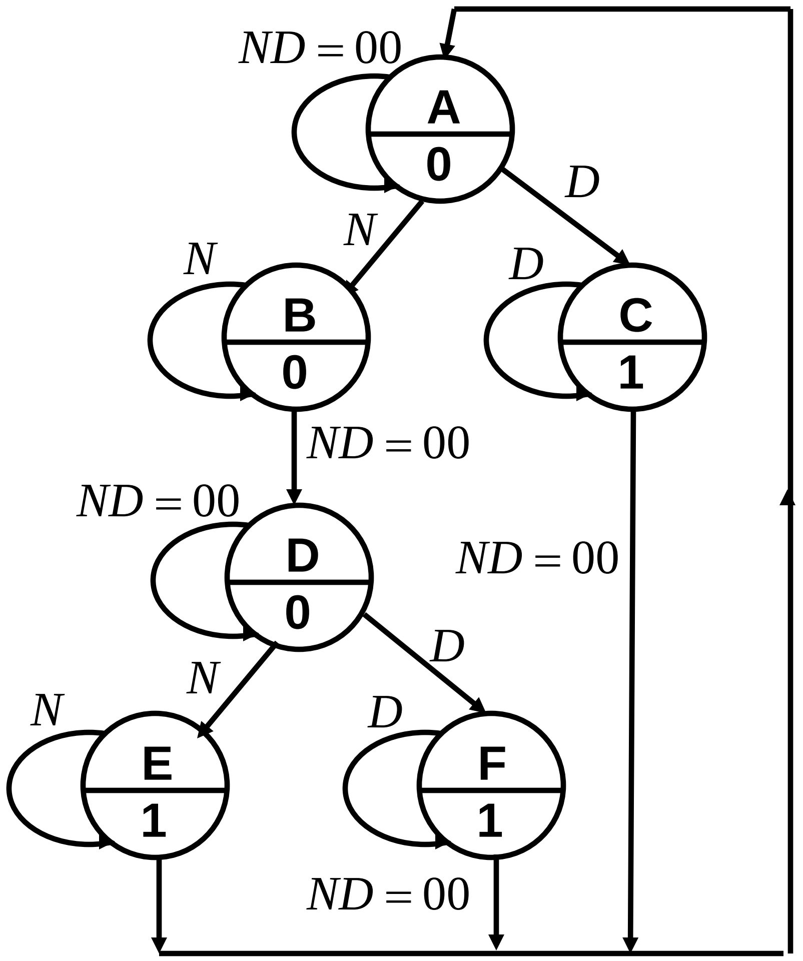
- **Procedure for minimizing the number of states**

1. Forming **equivalence** groups.
   To be in the same group, the following shall apply:
   - Outputs must have the same value
   - Stable states must be in the same place (column)
   - Don’t cares for next state muste be at the same place (column)
2. Minimize equivalence groups (state-reduction)
3. Form **merger diagram** different for Mealy or Moore.
4. Merge compatible states in groups. Minimize the number of groups simultaneously. Each state may only be part of one group.
5. Construct the reduced flow table by merging rows in the selected groups
6. Repeat step 3-5 to see if more minimizations may be done

Candy Machine (BV p. 610)

- Candy Machine has two inputs:
  - ***N***: Nickel (5 cent)
  - ***D***: Dime (10 cent)
- A candy costs 10 cent
- The machine does not return any money if there are 15 cent in the machine (one candy is returned)
- Output ***z*** is active when there is enough money for a candy

State diagram, Flow table



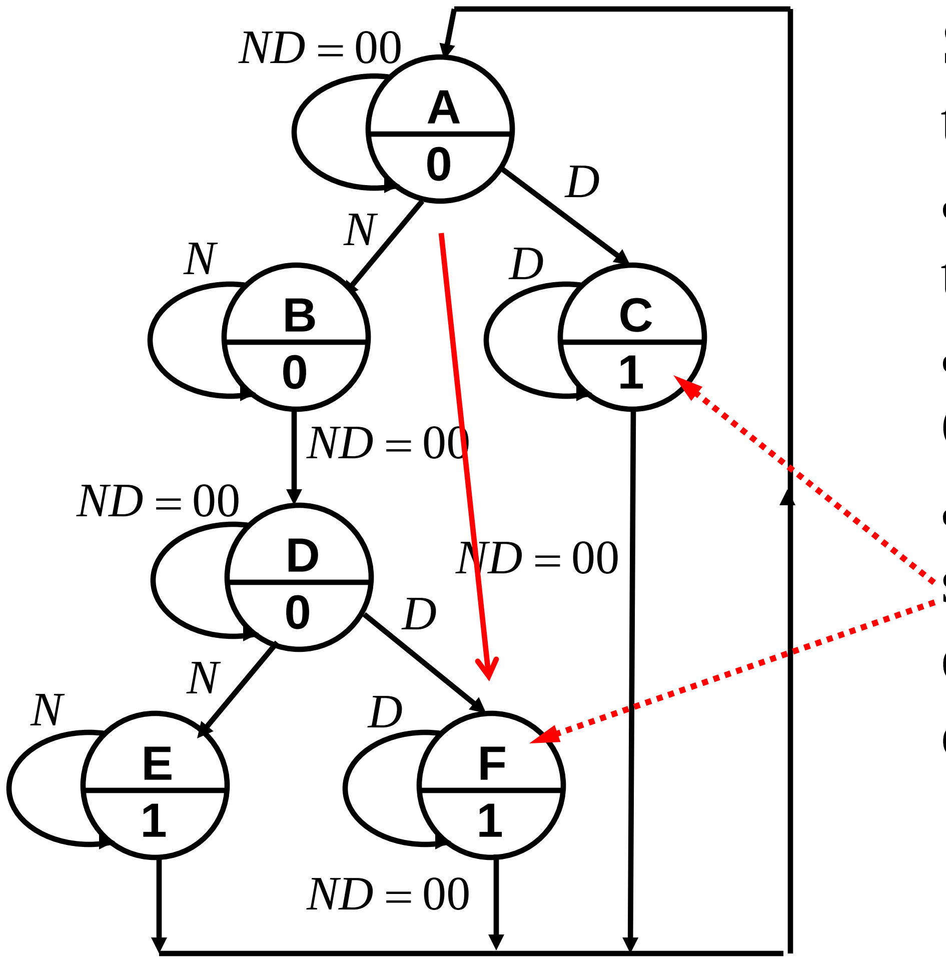
- *No ” double changes” of input signals!*
- *You can’t insert two coins at the same time!*

Pres state	Next State X=00	01	10	11	Q
A	A	B	C	-	0
B	D	B	-	-	0
C	A	-	C	-	1
D	D	E	F	-	0
E	A	E	-	-	1
F	A	-	F	-	1

(X = ND, Q = z)

A flow table that only has one stable state on each row is called a ***primitive flowtable***.

State Minimizing



State Minimization means that two states may be equivalent, and if so, replaced by one state to simplify the state diagram, and network.
One can easily see that state C and F could be replaced by one state, as a candy always be ejected after a Dime regardless of previous state.

Form/minimize equivalence groups

1. **Form equivalence groups.** To be in the same group, the following applies**:**
    - Outputs must have the same value
    - Stable states must be at same place (column)
    - Don’t cares for next state must be at same place (column)
2. **Minimize equivalence groups (state reduction).**

• Equivalence groups

Pres state	Next State X=00	01	10	11	Q
A	$\textcircled{A}$	B	C	-	0
B	D	$\textcircled{B}$	-	-	0
C	A	-	$\textcircled{C}$	-	1
D	$\textcircled{D}$	E	F	-	0
E	A	$\textcircled{E}$	-	-	1
F	A	-	$\textcircled{F}$	-	1

($X = ND$, $Q = z$)

The states is divided in blocks after the output value.
ABD has output **0**, **CEF** has output **1**.
P_1 = (ABD)(CEF)
Stable states must be for same input signal (column), don't care must be for same column.

AD has a stable state for 00. **B** has a stable for 01. **CF** has a stable state for 10. **E** has a stable for 01. **AD** and **CF** has don't care for corresponding input signals.

P_2 = (AD)(B)(CF)(E)

Merge equivalence groups

*Two rows could be ”merged” if it does **not conflikt** their successor states*

Pres state	Next State				Q
	X=00	01	10	11	
A	Ⓐ	B	C	-	0
B	D	Ⓑ	-	-	0
C	A	-	Ⓒ	-	1
D	Ⓓ	E	F	-	0
E	A	Ⓔ	-	-	1
F	A	-	Ⓕ	-	1

(X = ND, Q = z)

P_2=(AD)(B)(CF)(E)
P_3=(A)(D)(B)(C)(E)
P_4=P_3.

Rows **C** and **F** can be merged with a new name **C**, while **A** and **D** which has successors in different groups *not* can merge.

$C,F_{00} \rightarrow$ (**AD**), (**AD**)
$C,F_{01} \rightarrow$ **-**, **-**
$C,F_{10} \rightarrow$ (**CF**), (**CF**)
$C,F_{11} \rightarrow$ **-**, **-**

$A,D_{00} \rightarrow$ (**AD**), (**AD**)
$A,D_{01} \rightarrow$ (B),(E)
$A,D_{10} \rightarrow$ (**CF**), (**CF**)
$A,D_{11} \rightarrow$ **-**, **-**

Resulting flow table

Pres state	Next State				Q
	X=00	01	10	11	
A	Ⓐ	B	C	-	0
B	D	Ⓑ	-	-	0
Ⓒ	A	-	Ⓒ	-	1
D	Ⓓ	E	C	-	0
E	A	Ⓔ	-	-	1

Compatibility Groups

3. Form merger charts **either** for **Mealy** or **Moore**
4. Merge compatible states into groups. Minimize the number of groups simultaneously. Each state may only be part of a group.
5. Construct the reduced flow table by merging rows in the selected groups
6. Repeat steps 3-5 to see if more minimizations can be done

Merging rules

- **Two states are "compatible", and can be merged if the following applies**
  1. at least one of the following conditions apply to all input combinations
     - both S_i and S_j has the same successor state, or
     - both S_i and S_j are stable, or
     - The successor to S_i or S_j are both unspecified
  2. Then if you want to construct a Moore-compatible statemachine it also apply
     - both S_i and S_j has the same **output value** (this is not necessary when you construct a Mealy-compatible statemachine)

Merger diagram

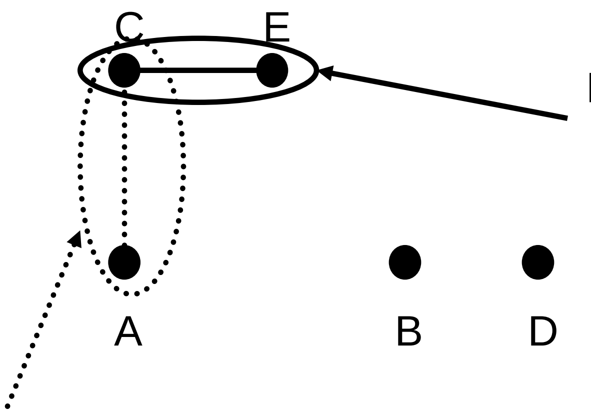
Resulting flowtable

Pres state	Next State X=00	01	10	11	Q
A	A	B	C	-	0
B	D	B	-	-	0
C	A	-	C	-	1
D	D	E	C	-	0
E	A	E	-	-	1

Each line will be a point in the Compatibility graph.

- *When there are there are several possibilities …*

Compatibily graph



Moore-compatible

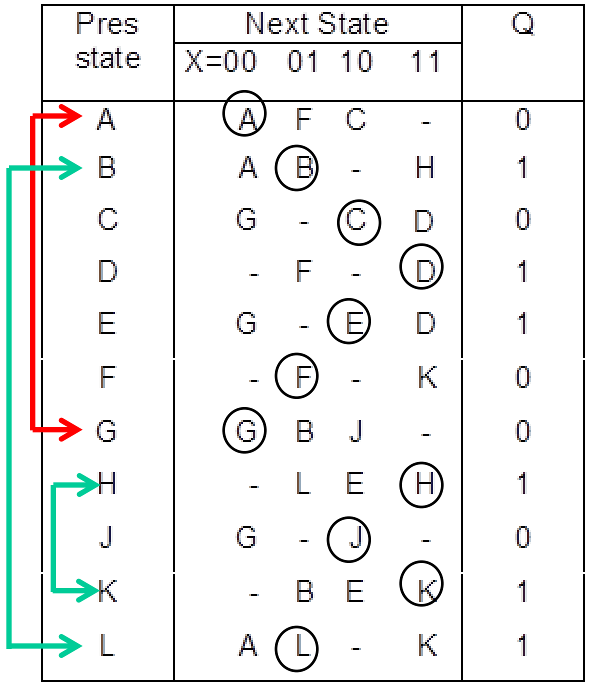
```
C(1): A-C-
E(1): AE--
```

Mealy-compatible: In state **A** (X = 00) the output is 0, in state **C** output is 1

```
C(1): A-C-
A(0): ABC-
```

An illustrative example (BV 9.8)

Primitive flowtable



Pres state	Next State X=00	01	10	11	Q
A	Ⓐ	F	C	-	0
B	A	Ⓑ	-	H	1
C	G	-	Ⓒ	D	0
D	-	F	-	Ⓓ	1
E	G	-	Ⓔ	D	1
F	-	Ⓕ	-	K	0
G	Ⓖ	B	J	-	0
H	-	L	E	Ⓗ	1
J	G	-	Ⓙ	-	0
K	-	B	E	Ⓚ	1
L	A	Ⓛ	-	K	1

equivalence classes

The same output, same position for stable states and do not care conditions (AG) (BL) (HK)

P_1= (AG)(BL)(C)(D)(E)(F)(HK)(J)

Successor state: A, G are *not* **equivalent**

$A,G_{00} \rightarrow$ (AG), (AG) $A,G_{01} \rightarrow$ (F),(BL)
$A,G_{10} \rightarrow$ (C),(J) $A,G_{11} \rightarrow$ -, -

$B,L_{00} \rightarrow$ (AG), (AG) $B,L_{01} \rightarrow$ (BL), (BL)
$B,L_{10} \rightarrow$ -, - $B,L_{11} \rightarrow$ (HK), (HK)

$H,K_{00} \rightarrow$ -, - $H,K_{01} \rightarrow$ (BL), (BL)
$H,K_{10} \rightarrow$ (E), (E) $H,K_{11} \rightarrow$ (HK), (HK)

P_2= (A)(G)$\boxed{\text{(BL)}}$(C)(D)(E)(F)$\boxed{\text{(HK)}}$(J) **P_3**=P_2

An illustrative example (BV 9.8)

equivalence classes

Primitive flowtable

Pres state	Next State				Q
	X=00	01	10	11	
A	Ⓐ	F	C	-	0
B	A	Ⓑ	-	H	1
C	G	-	Ⓒ	D	0
D	-	F	-	Ⓓ	1
E	G	-	Ⓔ	D	1
F	-	Ⓕ	-	K	0
G	Ⓖ	B	J	-	0
H	-	L	E	Ⓗ	1
J	G	-	Ⓙ	-	0
K	-	B	E	Ⓚ	1
L	A	Ⓛ	-	K	1

P_1= (AG)(BL)(C)(D)(E)(F)(HK)(J)
P_2= (A)(G)(**BL**)(C)(D)(E)(F)(**HK**)(J)
P_3=P_2

B for (**BL**)
H for (**HK**)

No unspecified states has yet been used!

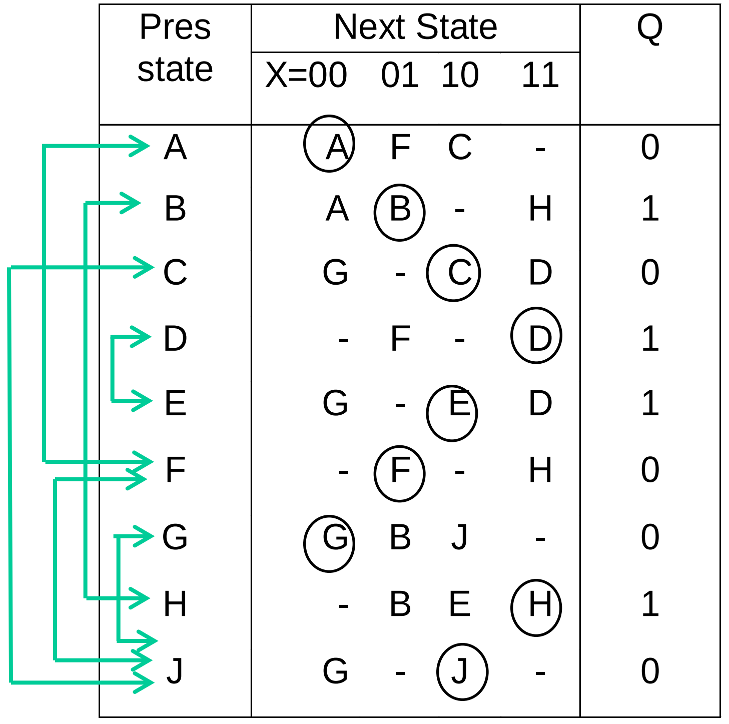
Reduced flowtable

Pres state	Next State				Q
	X=00	01	10	11	
A	Ⓐ	F	C	-	0
B	A	Ⓑ	-	H	1
C	G	-	Ⓒ	D	0
D	-	F	-	Ⓓ	1
E	G	-	Ⓔ	D	1
F	-	Ⓕ	-	H	0
G	Ⓖ	B	J	-	0
H	-	B	E	Ⓗ	1
J	G	-	Ⓙ	-	0

An illustrative example …

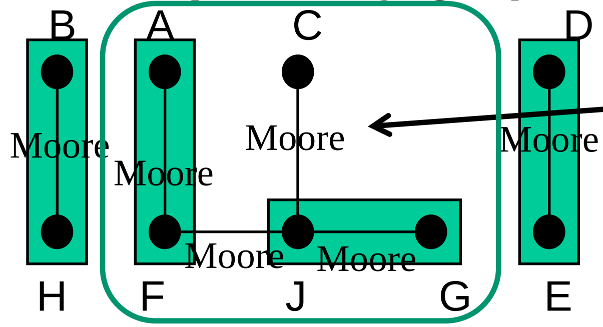
- **Compatibility**

Reduced flowtable



Pres state	Next State X=00	01	10	11	Q
A	A	F	C	-	0
B	A	B	-	H	1
C	G	-	C	D	0
D	-	F	-	D	1
E	G	-	E	D	1
F	-	F	-	H	0
G	G	B	J	-	0
H	-	B	E	H	1
J	G	-	J	-	0

Compatibility-graph



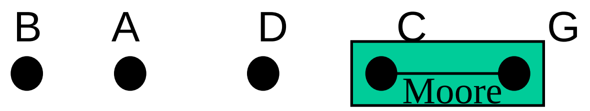
Different choices are possible

Pres state	Next State X=00	01	10	11	Q
A	A	A	C	B	0
B	A	B	D	B	1
C	G	-	C	D	0
D	G	A	D	D	1
G	G	B	G	-	0

New names **B (BH), A (AF), G (JG), D (DE)**

An illustrative example …

Compatibility-graph



More reduced flowtable

Pres state	Next State X=00	01	10	11	Q
A	Ⓐ	Ⓐ	C	B	0
B	A	Ⓑ	D	Ⓑ	1
C	G	-	Ⓒ	D	0
D	G	A	Ⓓ	Ⓓ	1
G	Ⓖ	B	Ⓖ	-	0

Slutlig flödestabell

Pres state	Next State X=00	01	10	11	Q
A	Ⓐ	Ⓐ	C	B	0
B	A	Ⓑ	D	Ⓑ	1
C	Ⓒ	B	Ⓒ	D	0
D	C	A	Ⓓ	Ⓓ	1

New name **C** for (**CG**)

Now all the unspecified conditions are used!

Summary

- **Asynchronous state machines**
  - Based on analysis of feedback combinational networks
  - All flip-flops and latches are asynchronous state machines
- **A similar theory as for synchronous state machines can be applied**
  - Only one input or state variable can be changed at a time!
  - One must also take into account the race problem