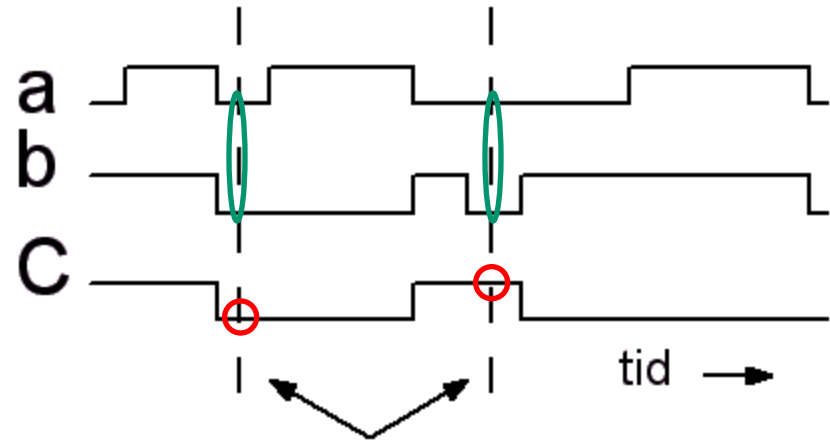
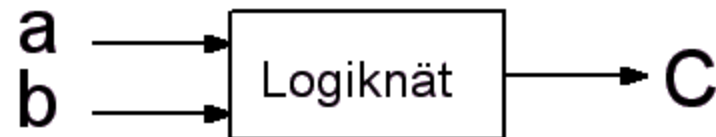


Sekvensnät



Som Du kommer ihåg ...

Samma insignal
kan ge olika utsignal

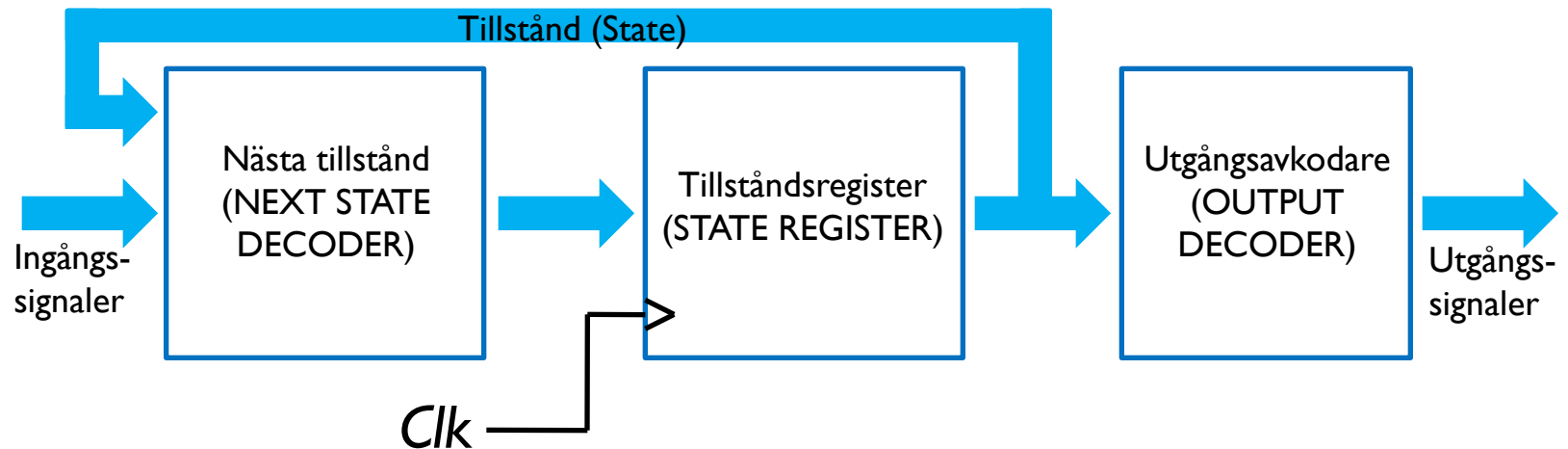


Designmetodik

Grundläggande designmetodik för tillståndsmaskiner.

1. Analysera specifikationen för kretsen
2. Skapa tillståndsdigram
3. Ställ upp tillståndstabellen
4. Minimera tillståndstabellen
5. Tilldela koder för tillstånden
6. Välj typ av vippor
7. Realisera kretsen mha Karnaugh-diagram.

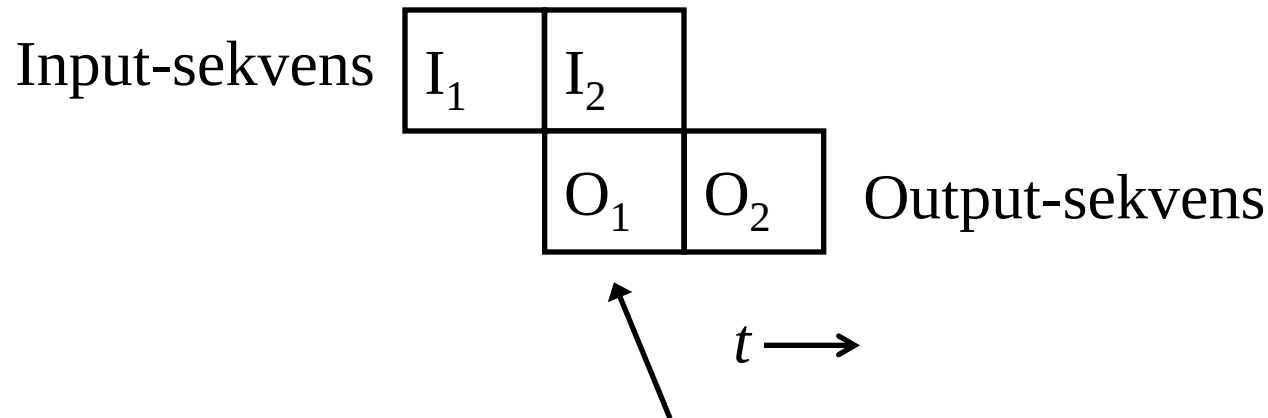
Moore-automat



För Moore-automaten beror utsignalerna på det inre tillståndet.

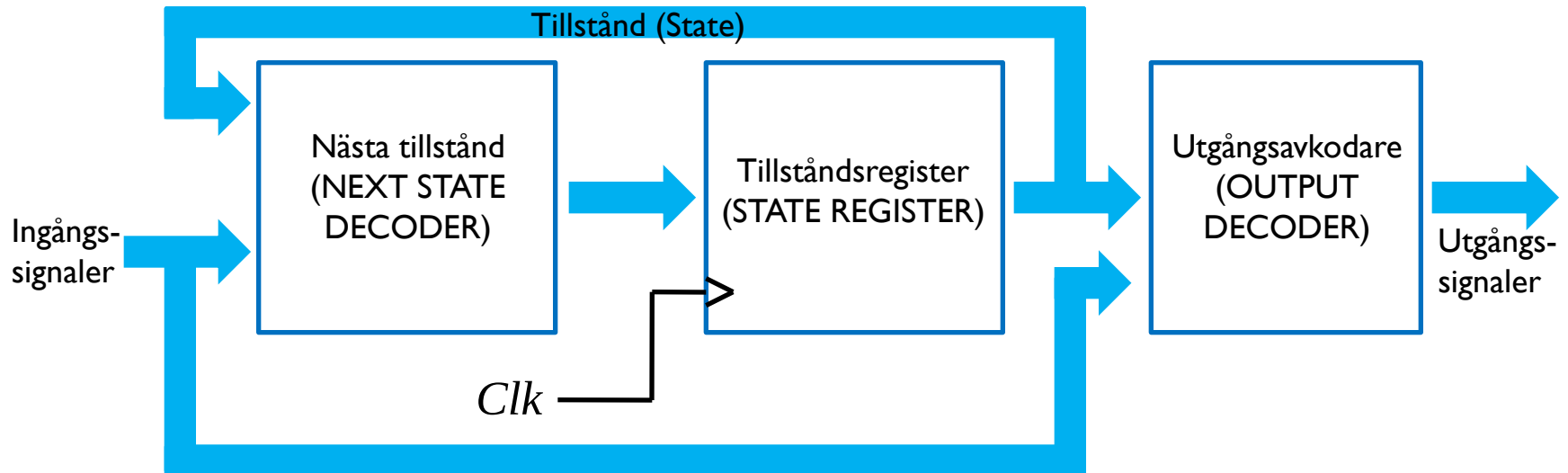
Moore: input och output

Tillståndet (State) ändras
här (på klockflanken) /



Output syns *efter* att tillståndet (state) har ändrats

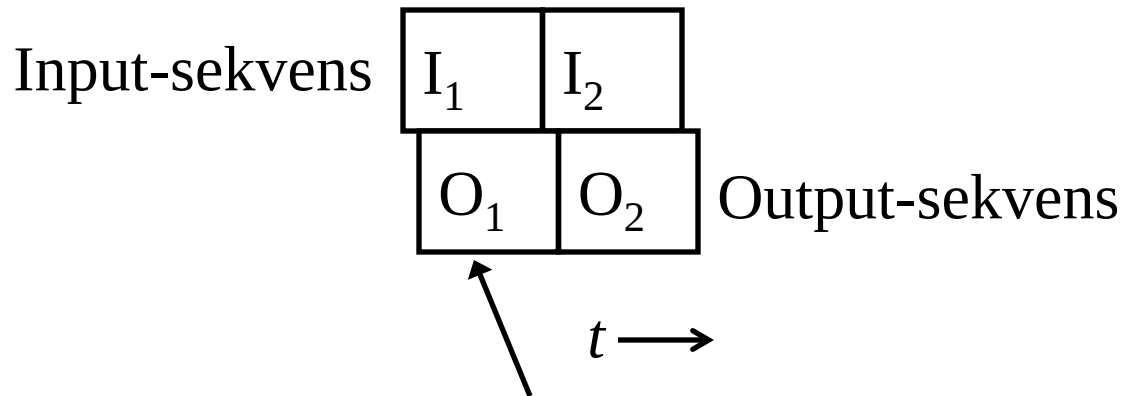
Mealy-automat



I en **Mealy**-Automat beror utgångssignalerna både på nuvarande tillstånd **och** ingångarna

Mealy: input och output

Tillståndet (State) ändras
här (på klockflanken)



Output syns ***direkt efter*** att input har ändrats

Överblivna tillstånd?

- Ibland får man några states *över* när man väljer kod.

(Totala antalet states är alltid potenser av 2)

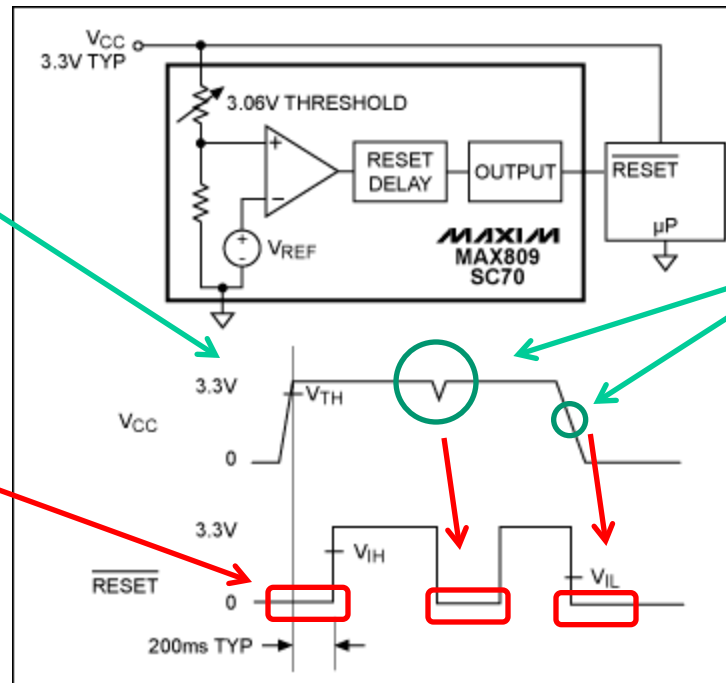
- Överblivna states måste tas om hand så att inte statemaskinen låser sig vid uppstart

Ett annat sätt är att man alltid (tex. automatiskt) gör RESET vid uppstart.

(RESET-generator chip)



Matnings-
spänning "på"
ger **RESET** i
200 ms



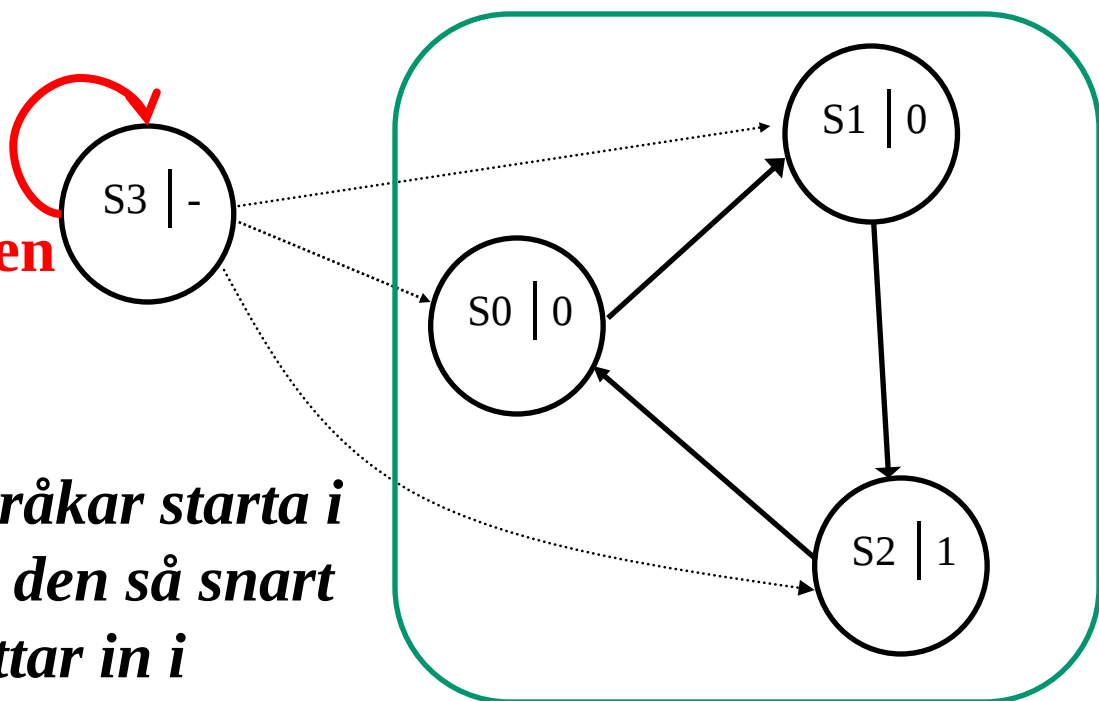
Om matnings-
spänningen
får problem,
eller sjunker
under viss
nivå, så blir
det **RESET**

Bättre än att behöva skaffa extra skydd, är att designa förebyggande och från början "ta hand om" alla tillstånd ...

Ex. räknare {0,1,2}

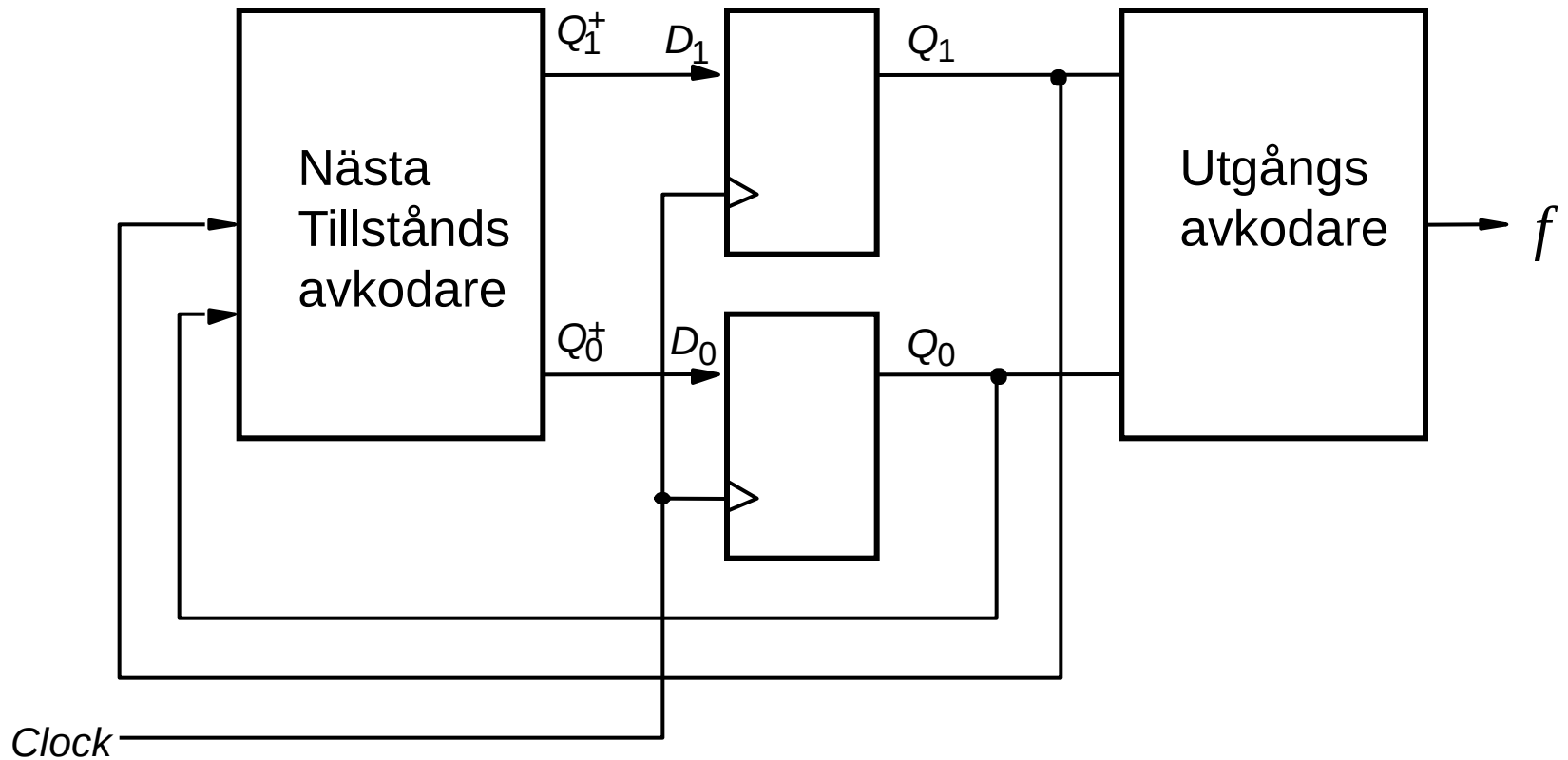
3 tillstånd $\rightarrow$ 2 vippor $\rightarrow 2^2 = 4$ tillstånd. Ett tillstånd blir över ...

**Farlig
övergång
(automaten
låser sig)**



Om maskinen råkar starta i S3 så vill vi att den så snart som möjligt hittar in i sekvensen!

Räknaren som automat



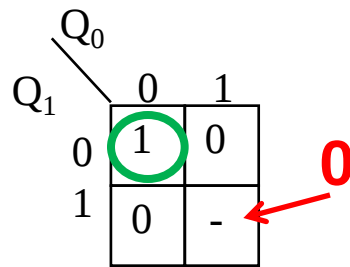
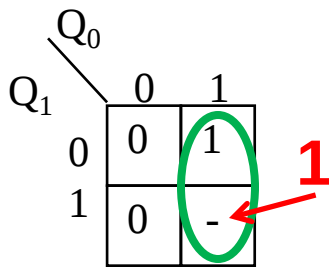
Nextstate-funktion

	Nuv. värde	Utsignal	Nästa värde	D-vippa
	Q_1Q_0	f	$Q_1^+Q_0^+$	D_1D_0
S0	0 0	0	0 1	0 1
S1	0 1	0	1 0	1 0
S2	1 0	1	0 0	0 0
	1 1	-	- - (ej 11)	- -

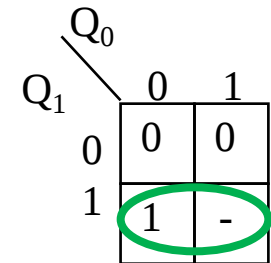
Vi kan specificera vad som helst här – **utom** att ”stanna kvar (dvs. ej 11)!

Karnaughdiagram

	Nuv. värde	Utsignal	Nästa värde	D-vippa
	$Q_1 Q_0$	f	$Q_1^+ Q_0^+$	$D_1 D_0$
S0	0 0	0	0 1	0 1
S1	0 1	0	1 0	1 0
S2	1 0	1	0 0	0 0
	1 1	-	- - (ej 11)	- -



$$Q_1^+ = D_1 = Q_0 \quad Q_0^+ = D_0 = \overline{Q_1} \overline{Q_0}$$



$$f = Q_1$$

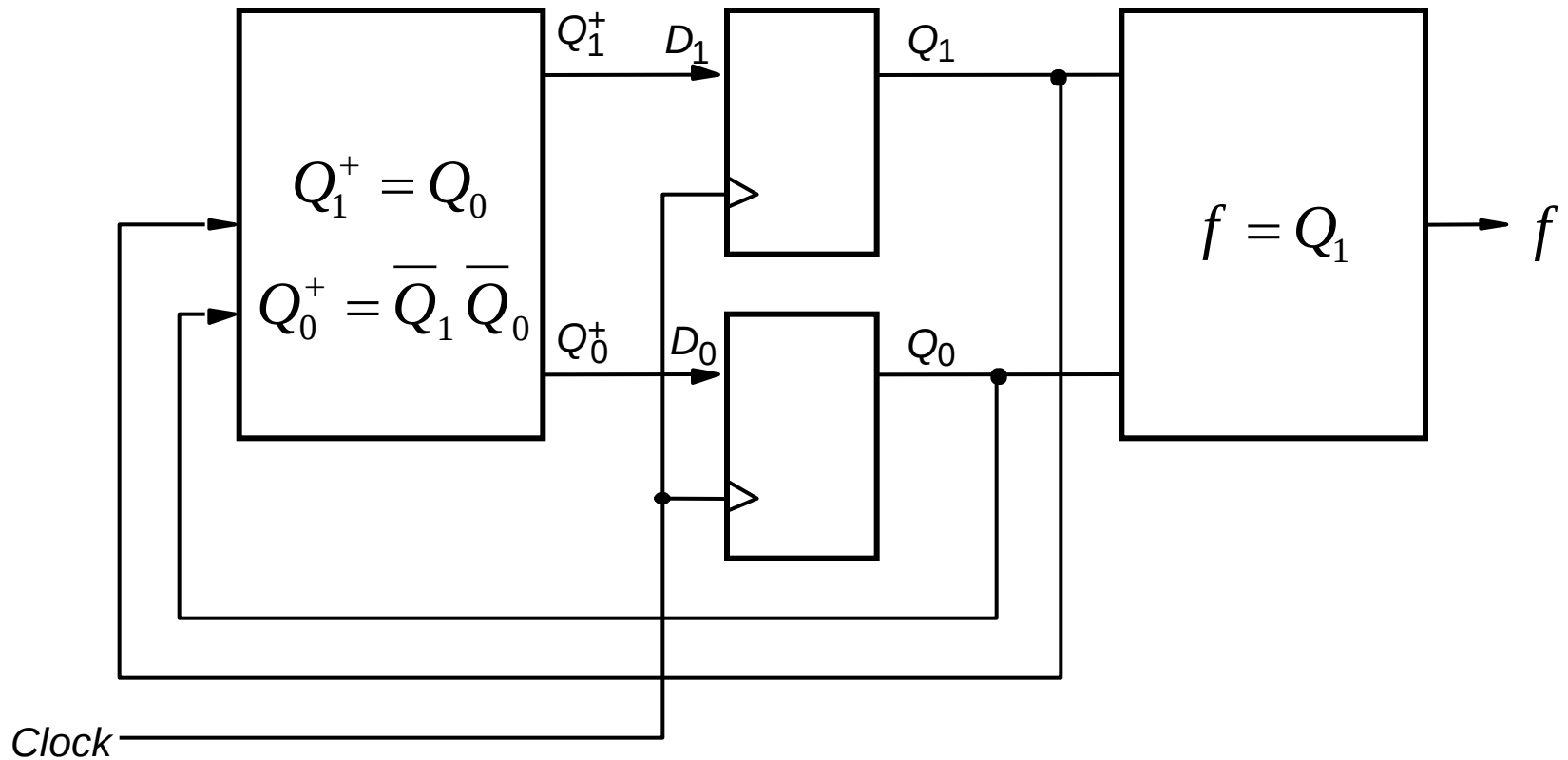
• OK, 10 *inte* 11!

Minimerad kodad tillståndstabell

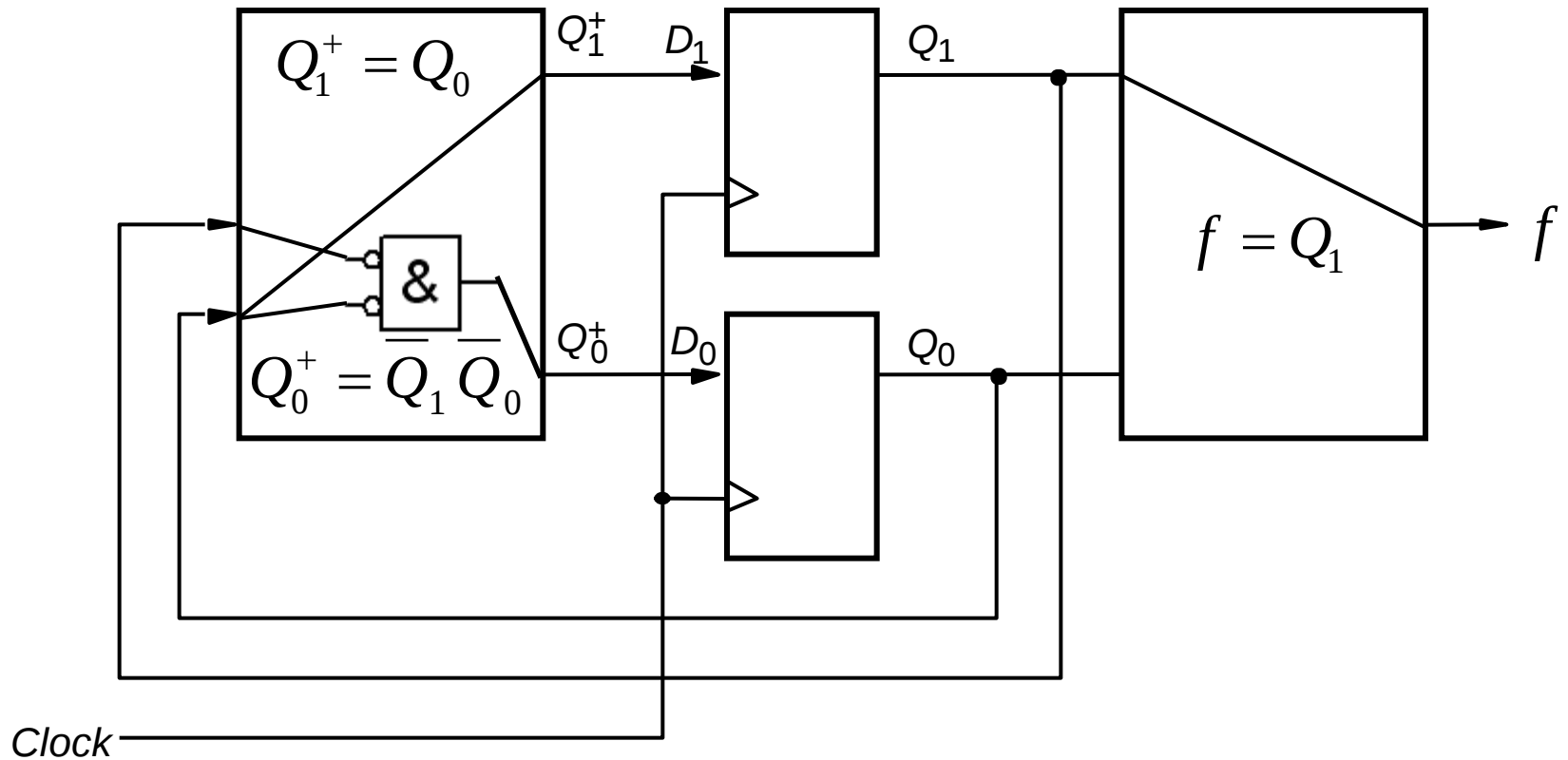
Nuv. värde	Utsignal	Nästa värde	D-vippa
Q_1Q_0	f	$Q_1^+Q_0^+$	D_1D_0
0 0	0	0 1	0 1
0 1	0	1 0	1 0
1 0	1	0 0	0 0
1 1	1	1 0 (ej 11)	1 0

**Dvs, det extra tillståndet går in till S2
i huvudsekvensen...**

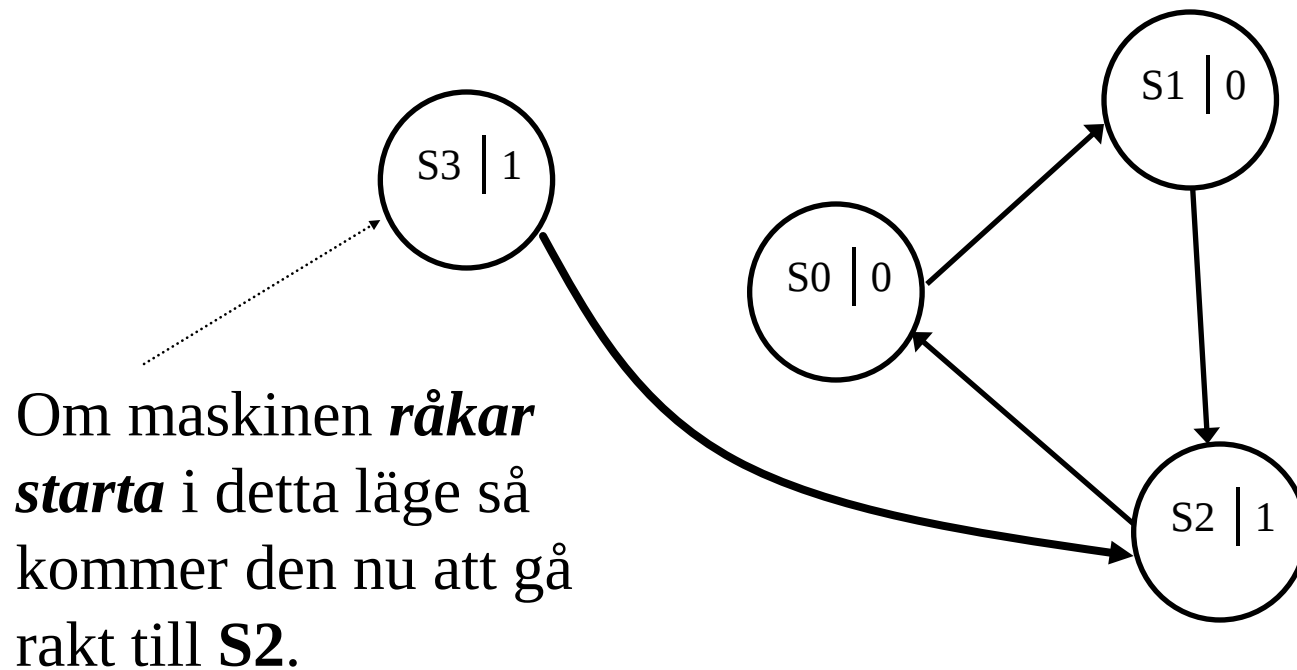
Räknaren



Räknaren

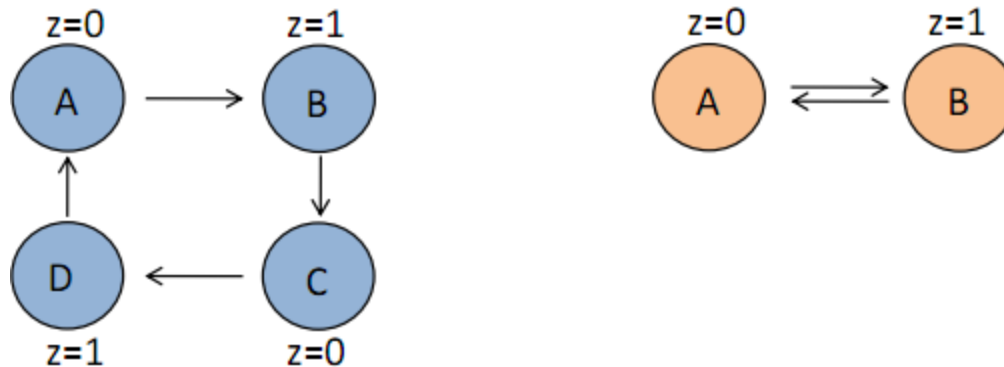


Slutgiltigt tillståndsdigram



Tillståndsminimering

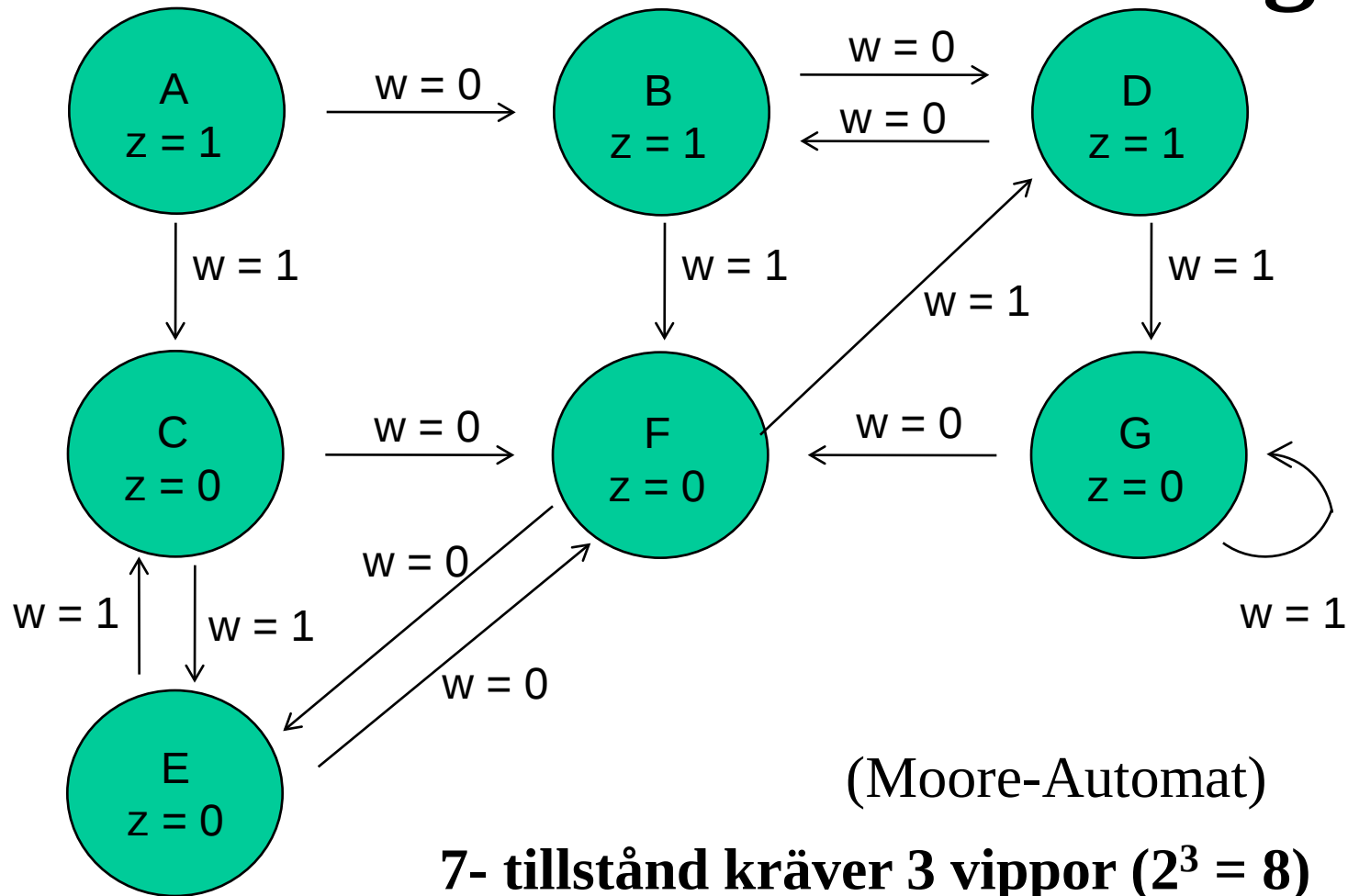
När man konstruerar komplexa tillståndsmaskiner så kan det lätt hända att det finns **ekvivalenta** och därmed redundanta tillstånd ***som kan tas bort*** för att få en effektivare implementering.



Minimeringsmetod

- Följande exempel illustrerar en manuell minimeringsmetod
 - syftet är att förklara begreppet tillståndsminimering
- Observera att CAD-syntesverktyg använder andra (effektivare) algoritmer

Ex. Tillståndsminimering



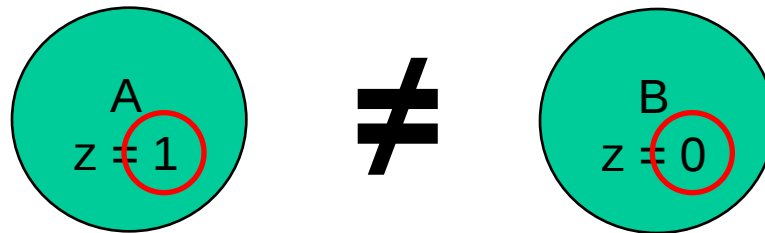
Inte ekvivalenta tillstånd

Det är mycket enklare att skilja ut tillstånd som absolut inte kan vara ekvivalenta än att direkt leta reda på ekvivalenta tillstånd ...

Minimeringens grundidé

Två tillstånd är *inte* ekvivalenta om de har olika utgångsvärden, dvs. om:

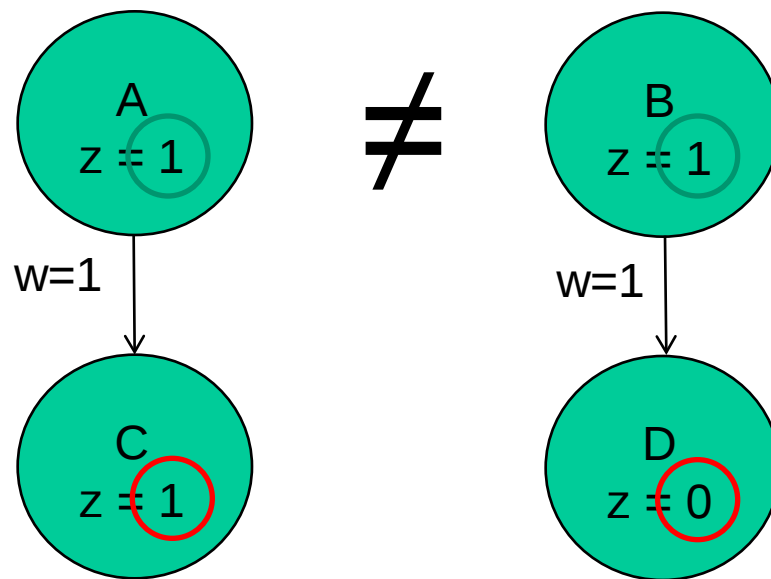
1. de har olika utgångsvärden



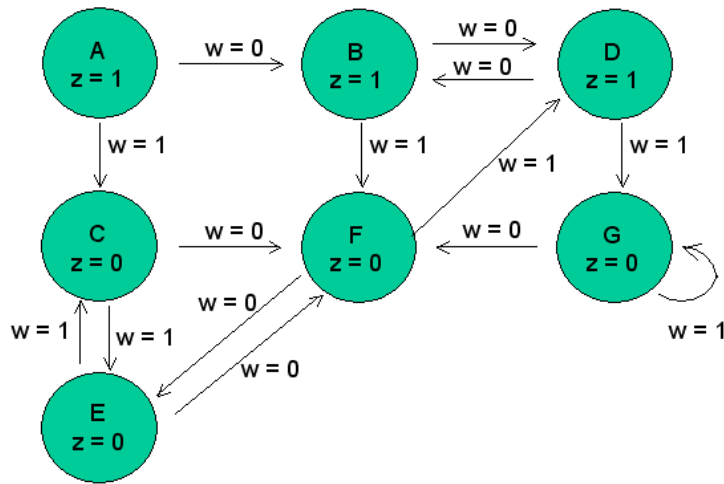
Minimeringens grundidé

Två tillstånd är *inte* ekvivalenta om de har lika utgångsvärden, men

2. om *någon* av tillståndsövergångarna leder till *olika* efterföljande utgångsvärden



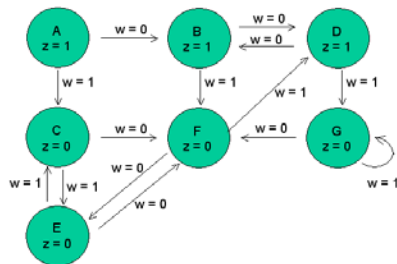
Tillståndstabell



Ursprungligt tillståndsdiagram

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

Ursprunglig tillståndstabell



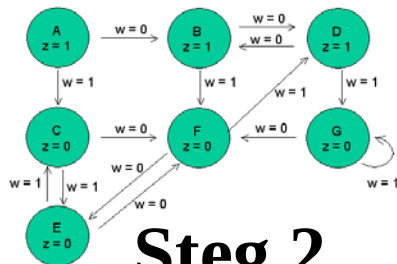
Minimera tillstånd

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

Partitioner. Grupper av tillstånd.

Start

P_1 . Från början utgör ***alla*** tillstånd ett enda block,
 $P_1 = (ABCDEFGG)$



Steg 2

Minimera tillstånd

$$P_1 = (ABDCDEFG)$$

Present state	Next state		Output z
	w = 0	w = 1	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

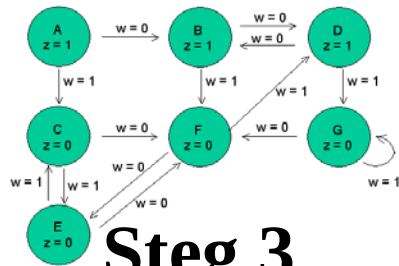
Gruppera nu tillstånden i grupper efter samma utsignal

Vilka tillstånd har samma utsignal?

- ABD har utsignalen $z = 1$
- CEFG har utsignalen $z = 0$

$$P_2 = (ABD)(CEFG)$$

Tillstånden A, B, D kan därför aldrig vara ekvivalenta med något av tillstånden C, E, F, G eller tvärtom



Minimera tillstånd

$$P_2 = (ABD)(CEFG)$$

Present state	Next state		Output z
	w = 0	w = 1	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

Steg 3

Vilka **följdtillstånd** har tillstånden?

Block (ABD)

w = 0: “0-successor”: $A \rightarrow (ABD)$, $B \rightarrow (ABD)$, $D \rightarrow (ABD)$

alla är övergångar inom egna blocket (ABD)

w = 1: “1-successor”: $A \rightarrow (CEFG)$, $B \rightarrow (CEFG)$, $D \rightarrow (CEFG)$

alla är övergångar till samma block (CEFG)

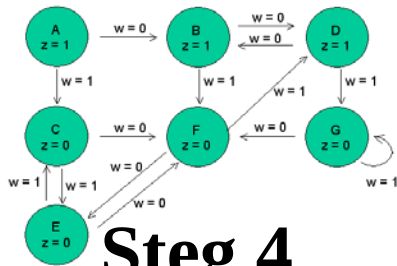
Block (CEFG)

w = 0: 0-successor : $C \rightarrow (CEFG)$, $E \rightarrow (CEFG)$, $F \rightarrow (CEFG)$,
 $G \rightarrow (CEFG)$ alla är övergångar inom egna blocket (CEFG)

w = 1: 1-successor : $C \rightarrow (CEFG)$, $E \rightarrow (CEFG)$, $F \rightarrow (ABD)$,
 $G \rightarrow (CEFG)$ C E G går till samma block

F avviker, går till ett annat block

$$P_3 = (ABD)(CEG)(F)$$



Steg 4

Minimera tillstånd

$$P_3 = (ABD)(CEG)(F)$$

Present state	Next state		Output z
	w = 0	w = 1	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

Vilka **följdtillstånd** har tillstånden?

Block (ABD)

w = 0: “0-successor”: A → (ABD), B → (ABD), D → (ABD)

alla är övergångar inom egna blocket ABD

w = 1: “1-successor”: A → (CEG), B → (F), D → (CEG)

A → C, D → G övergångar till samma block

B → F avviker, går till ett annat block

Block (CEG)

w = 0: “0-successor” : C → (F), E → (F), G → (F)

alla är övergångar till samma block block (F)

w = 1: “1-successor” : C → (CEG), E → (CEG), G → (CEG)

alla är övergångar inom egna blocket (CEG)

$$P_4 = (AD)(B)(CEG)(F)$$

Minimerat

Nästa partition $\mathbf{P}_5$ blir densamma som $\mathbf{P}_4$. Processen är därför klar. AD respektive CEG är **ekvivalenta**.

A' blir en ny beteckning för AD, C' blir ny beteckning för CEG.

$$\mathbf{P}_4 = (\text{AD})(\text{B})(\text{CEG})(\text{F}) = (\text{A}')(\text{B})(\text{C}')(\text{F})$$

Minimerad tillståndstabell

$$P_4 = (AD)(B)(CEG)(F) = (A')(B)(C')(F)$$

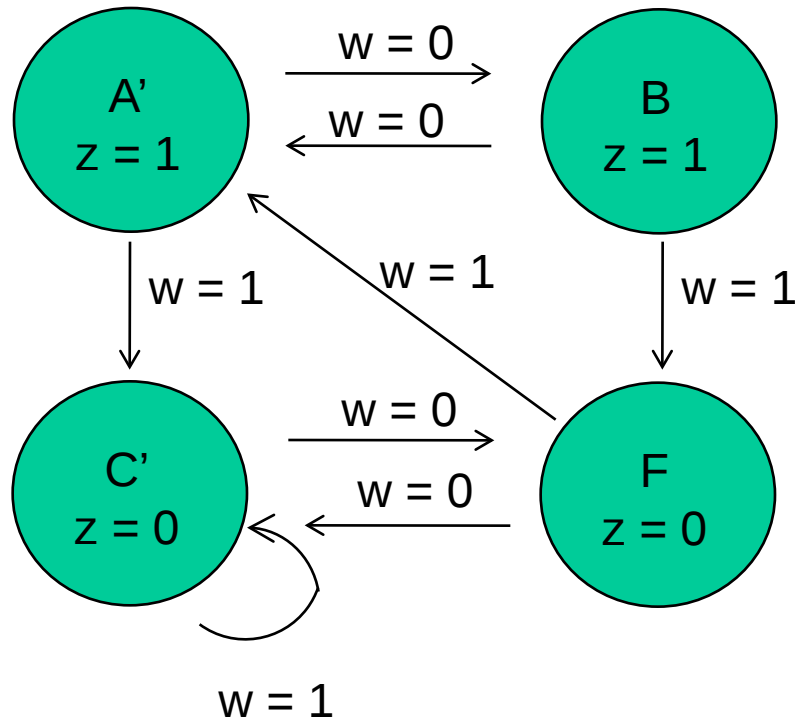
A' är ny beteckning för AD, C' är ny för CEG.

Present state	Next state		Output z
	w = 0	w = 1	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

Present state	Nextstate		Output z
	w = 0	w = 1	
A'	B	C'	1
B	A'	F	1
C'	F	C'	0
F	C'	A'	0

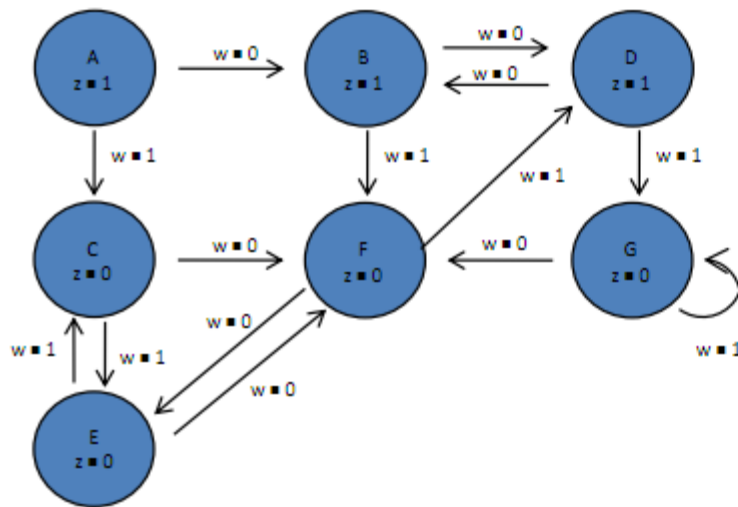
Minimerat tillståndsdigram

Present state	Nextstate		Output z
	w = 0	w = 1	
A'	B	C'	1
B	A'	F	1
C'	F	C'	0
F	C'	A'	0

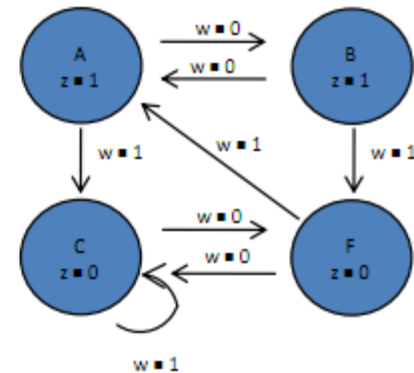


4 tillstånd kräver 2 vippor ($2^2 = 4$).

Jämförelse



Före minimering



Efter minimering

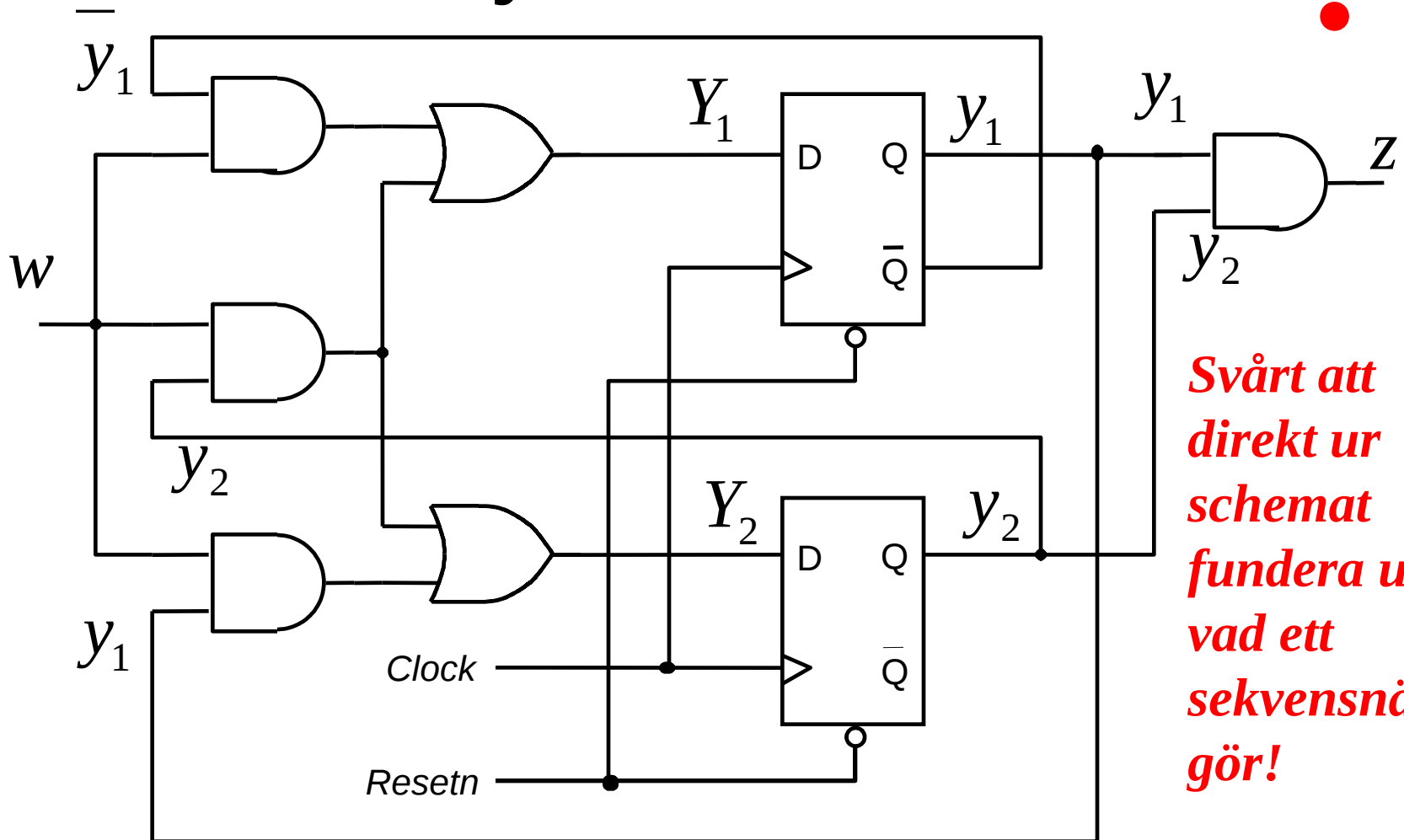
Värdet av tillståndsminimering?

- Det är *inte* säkert att färre tillstånd leder till ett *enklare* nät!

Fördelen med tillståndsminimering ligger i stället i att det blir enklare att skapa det ursprungliga tillståndsdigrammet när man inte behöver anstränga sig för att det dessutom ska bli minimalt från början!

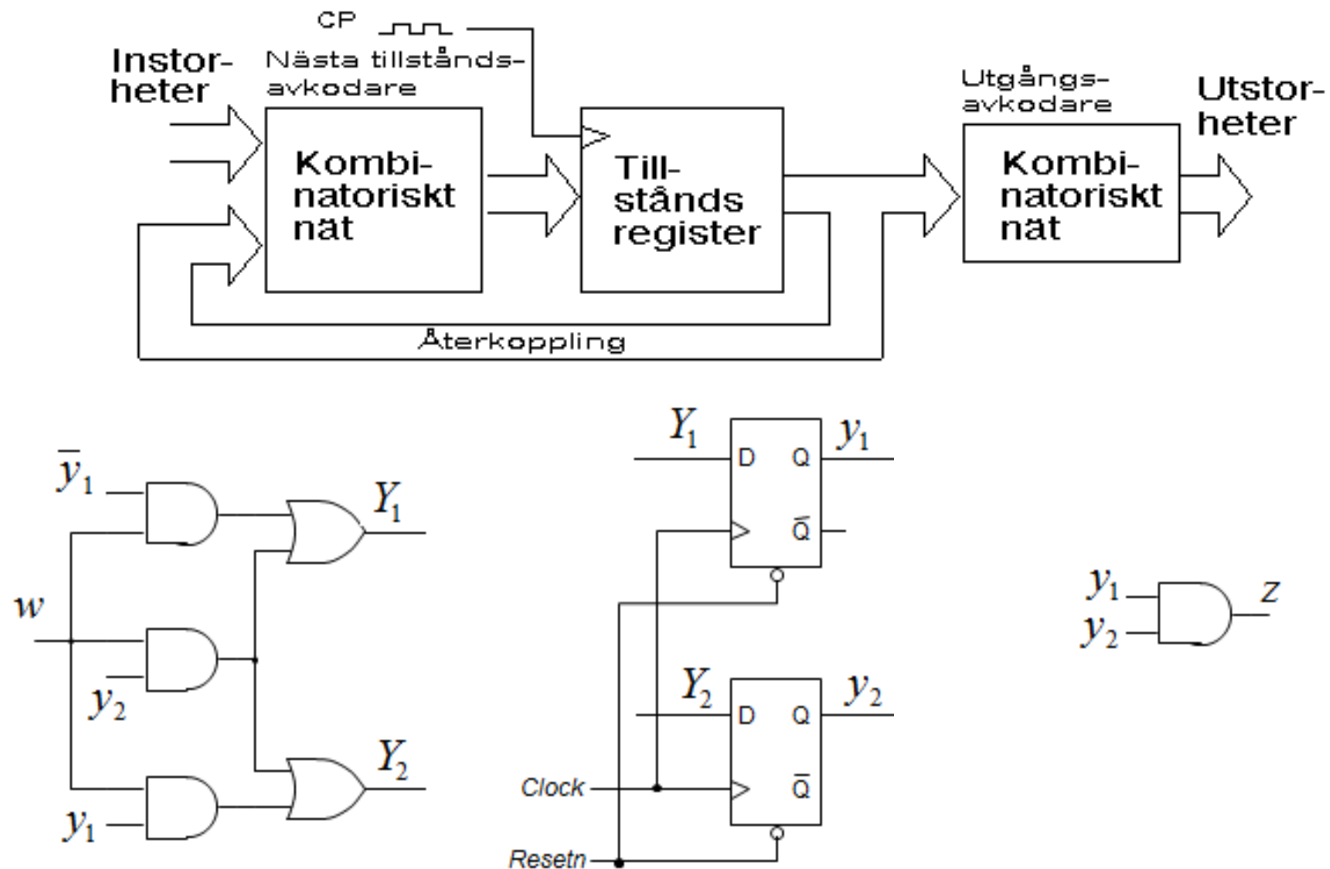
CAD-verktygen minimerar sedan det ursprungliga tillståndsdigrammet till ett slutgiltigt.

Analys av sekvensnät

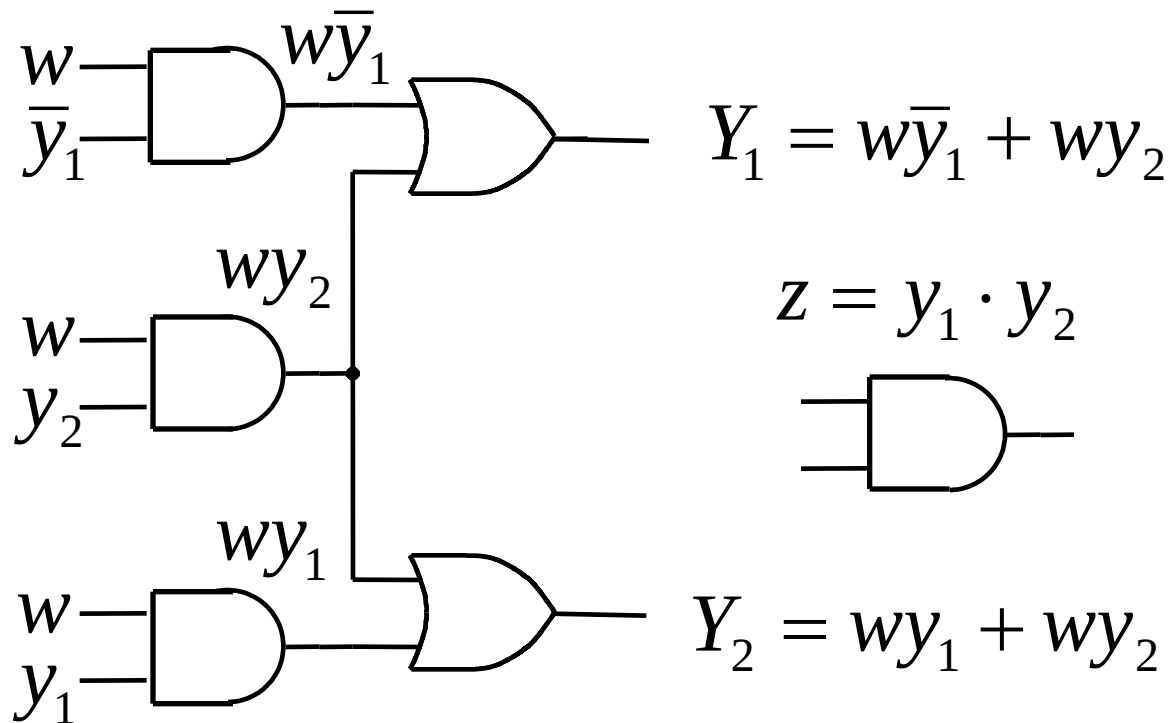


*Svårt att
direkt ur
schemat
fundera ut
vad ett
sekvensnät
gör!*

Tänk Moore-automat!

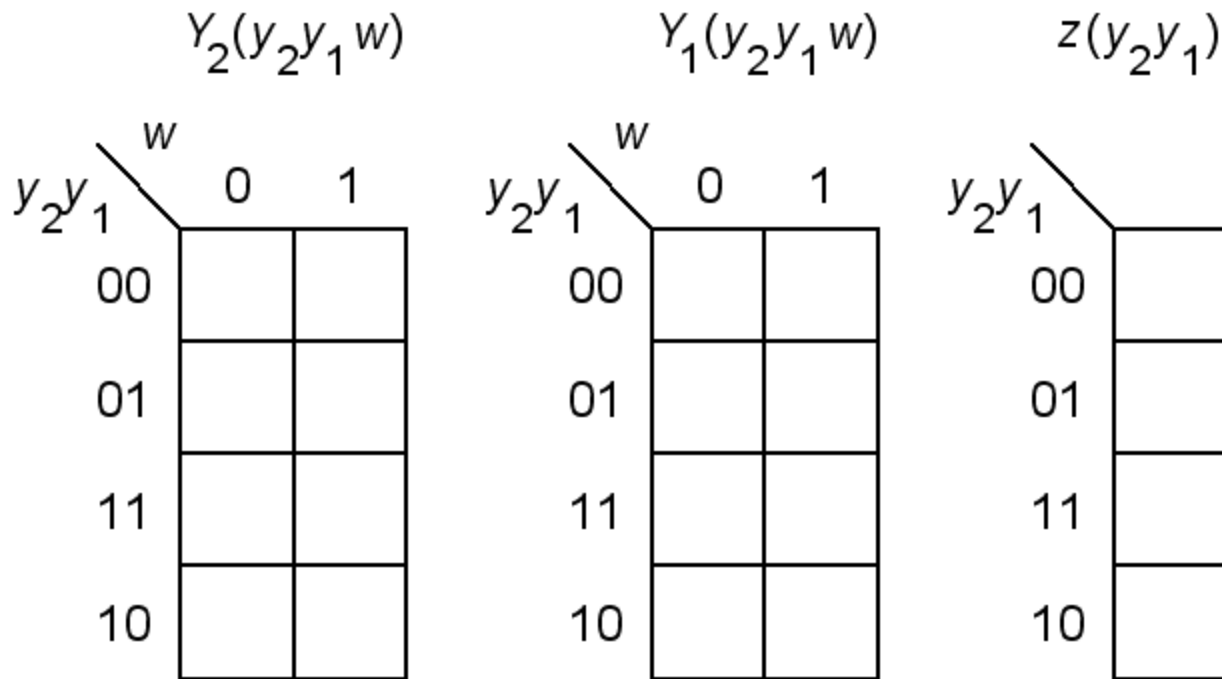


Analysera grindnäten



Fyll i Karnaughdiagram

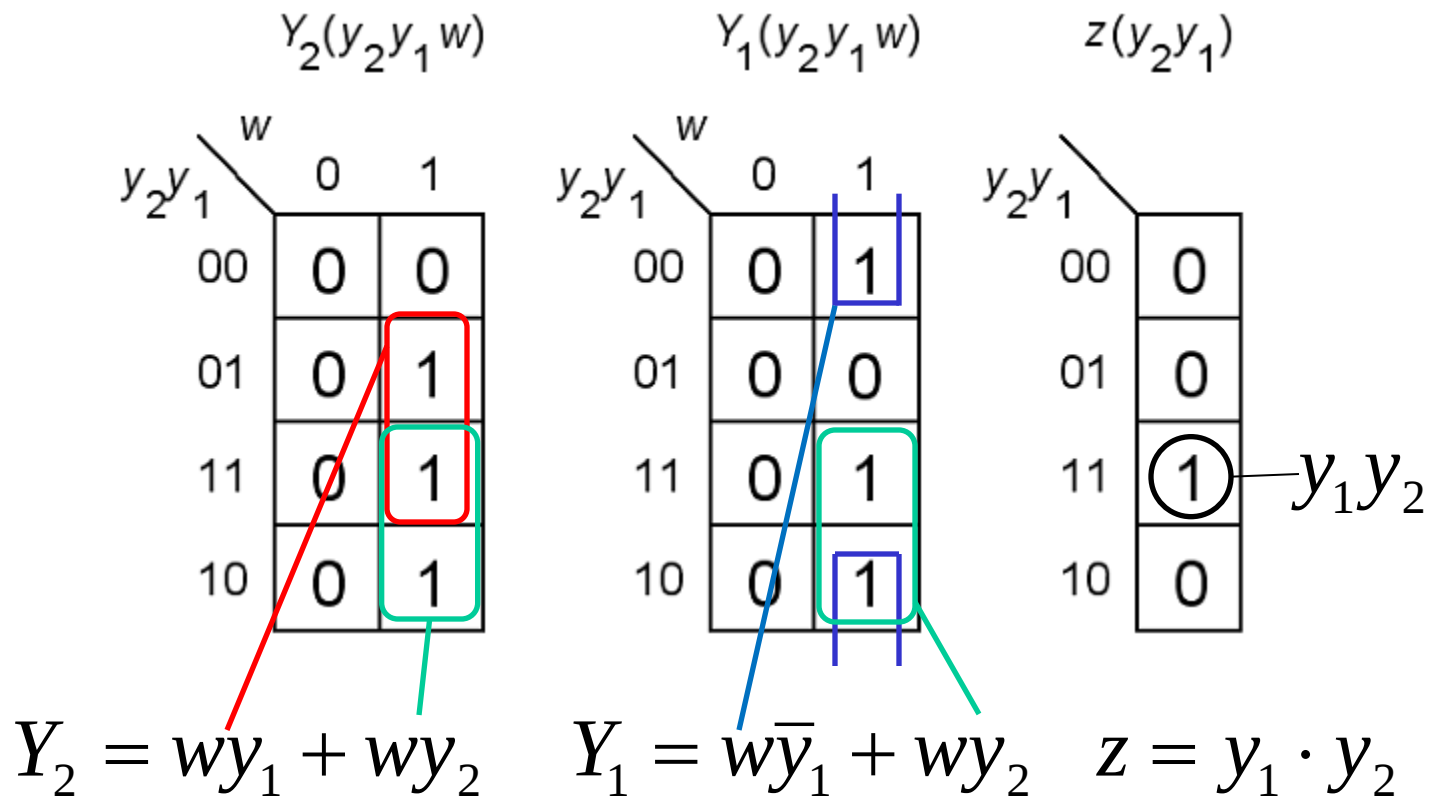
Kan Du fylla i Karnaughdiagrammen med funktionerna?



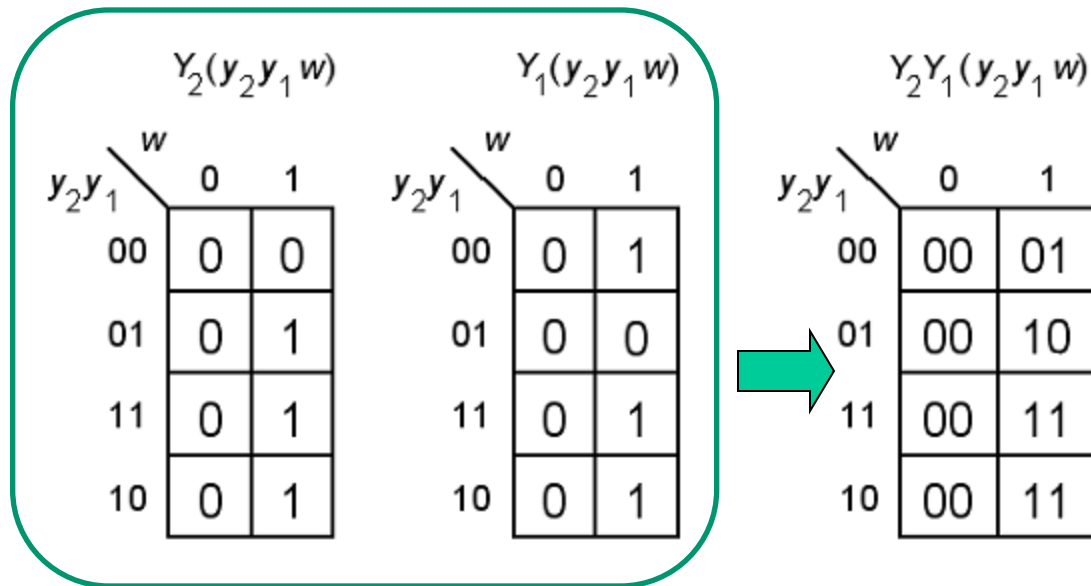
$$Y_2 = wy_1 + wy_2 \quad Y_1 = w\bar{y}_1 + wy_2 \quad z = y_1 \cdot y_2$$

Ifyllda Karnaughdiagram

Ifyllda Karnaughdiagram

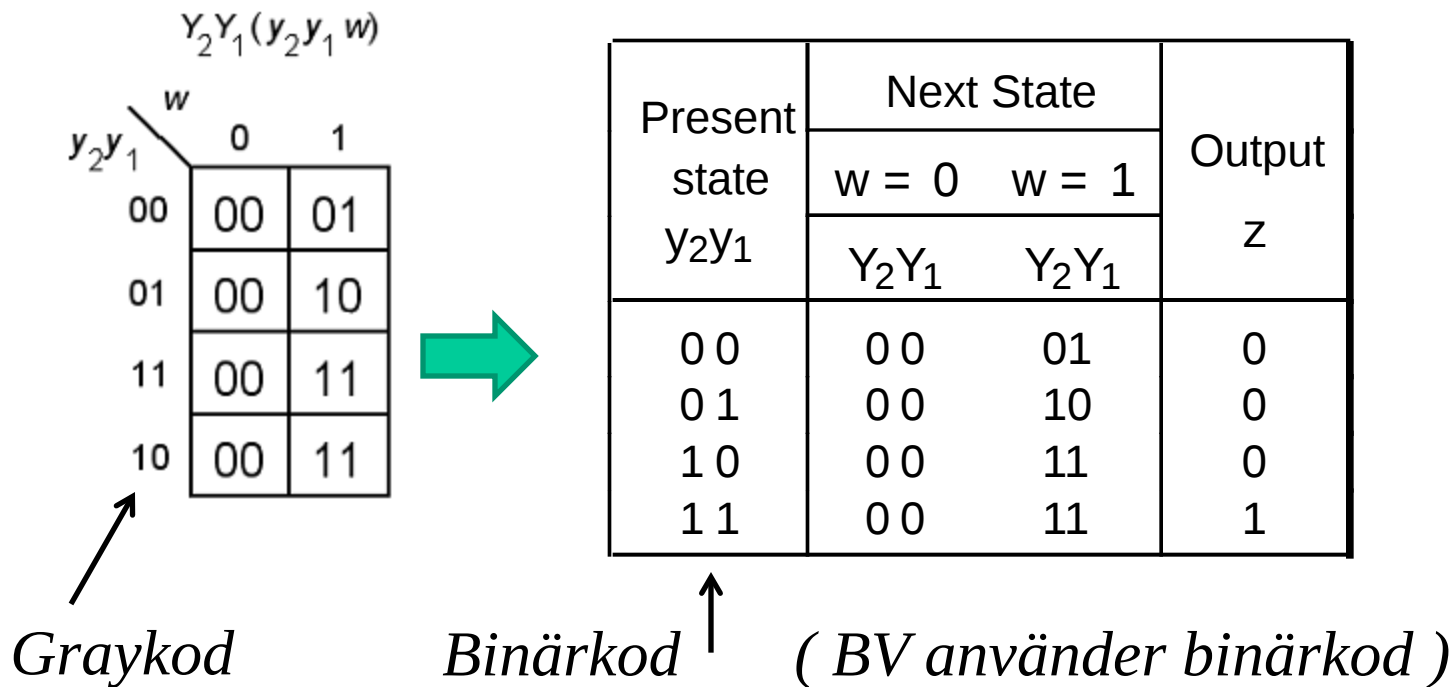


Kodad tillståndstabell



Slå ihop Karnaughdiagrammen till en kodad tillståndstabell

Kodad tillståndstabell



Tillståndstabell

Kodad tillståndstabell

Present state y_2y_1	Next State		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	0 1	0
0 1	0 0	1 0	0
1 0	0 0	1 1	0
1 1	0 0	1 1	1



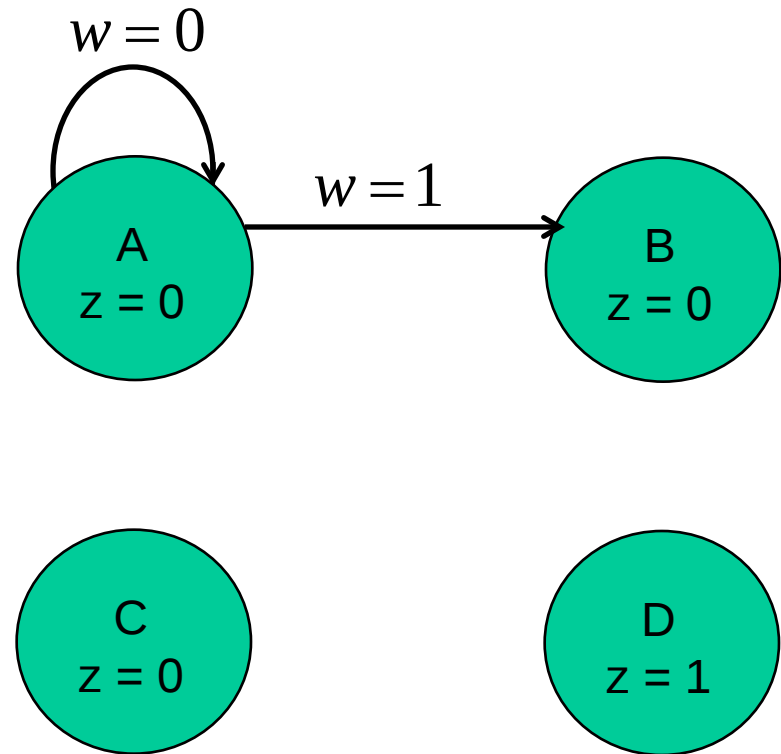
Tillståndstabell

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

Den okodade tillståndstabellen är utgångspunkt om man vill byta till en annan tillståndskodning.

Tillståndsdigram

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

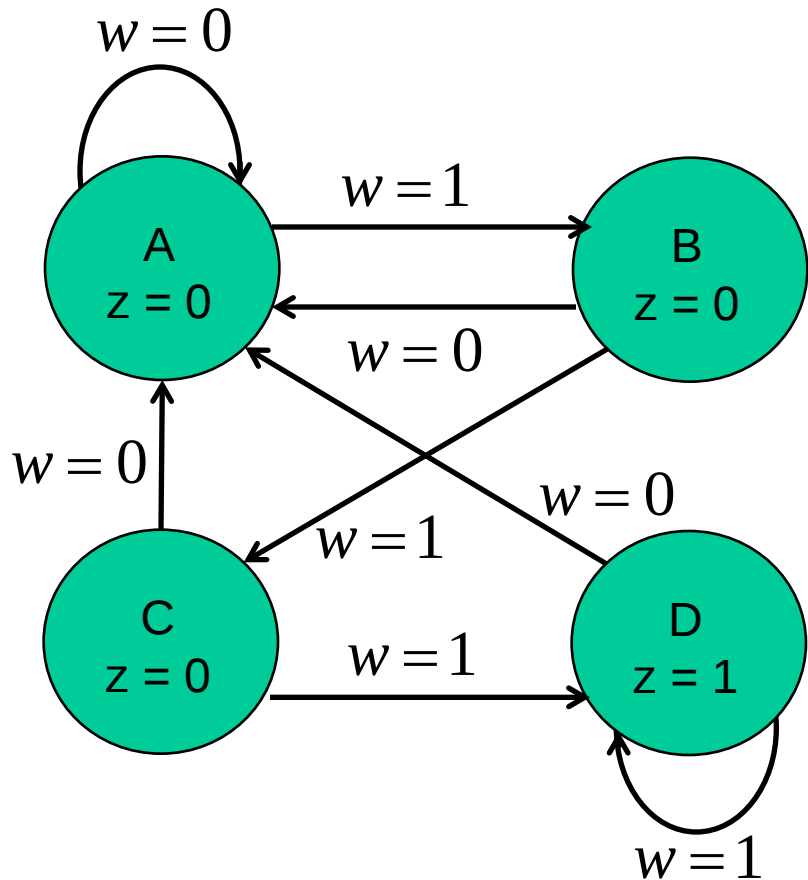


***Rita färdigt tillståndsdigrammet själv.
(På övning 6 löser vi ett liknande problem – kretsen är en "tre i rad" - krets).***

Tillståndsdigram

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

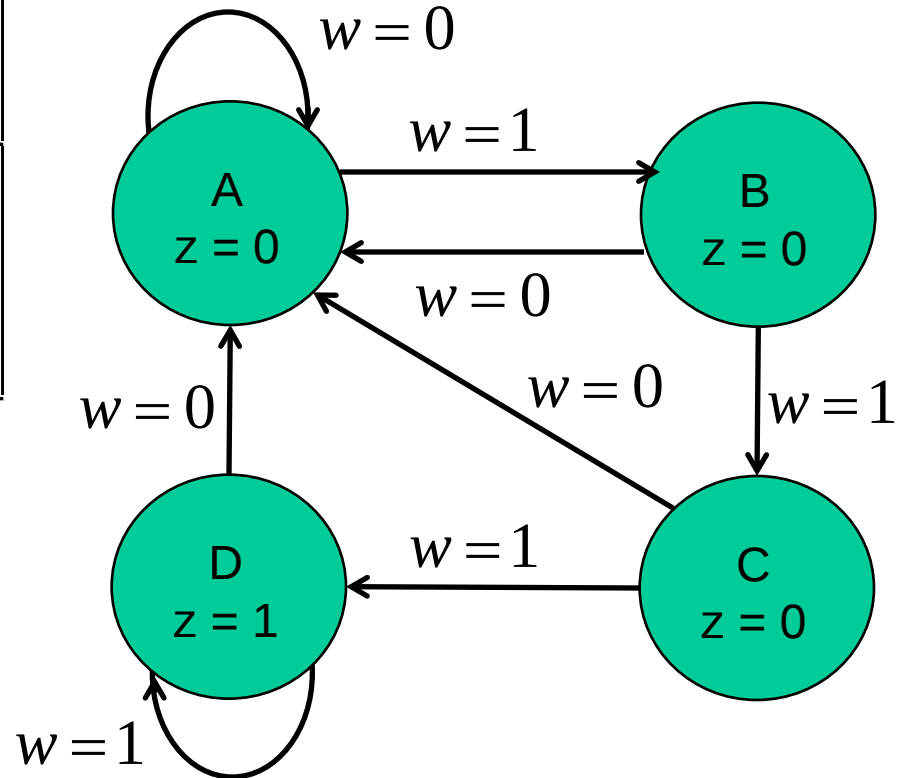
*Ibland kan man behöva
ändra ordningen på
tillstånden för att få ett
tydligare diagram*



Tillståndsdigram

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

C och D har bytt plats – snyggare, inga korsande tillståndspilar



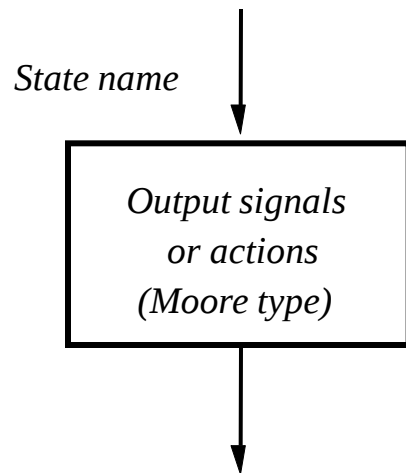
- Att kräva samma insignal ”tre gånger i rad” är en ofta använd säkerhetsåtgärd.

ASM-charts

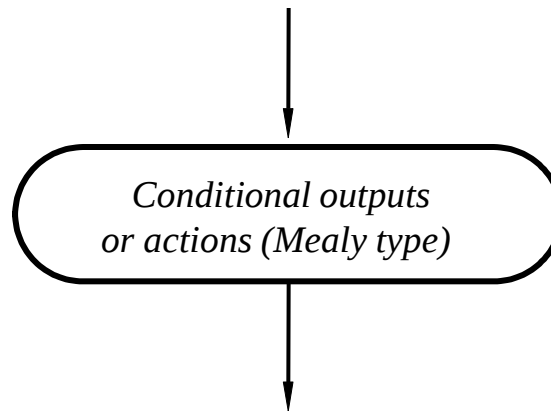
För att beskriva större tillståndsmaskiner används ofta ett annat diagram: Algorithmic State Machine (**ASM**) Charts

ASM-chart, tre byggstenar

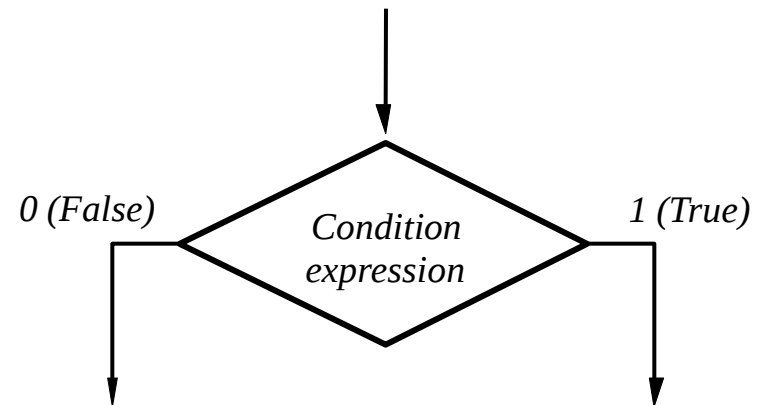
Ett ASM-chart är ett flödesdiagram som byggs upp av tre olika byggstenar.



(a) State box



(c) Conditional output box



(b) Decision box

ASM-charts

Tillståndslåda (State Box)

**Representerar ett tillstånd i ett FSM
utgångsvärden för tillståndet anges här
(Moore-outputs)**

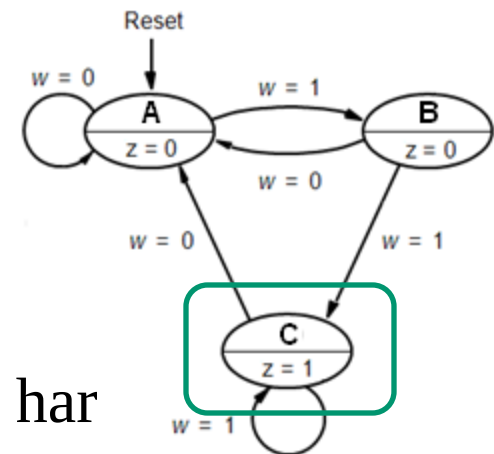
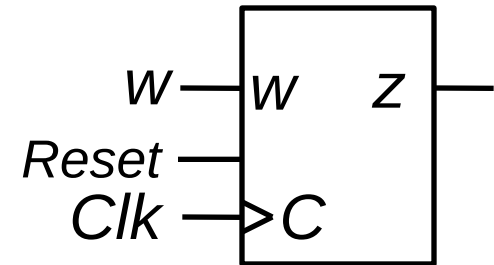
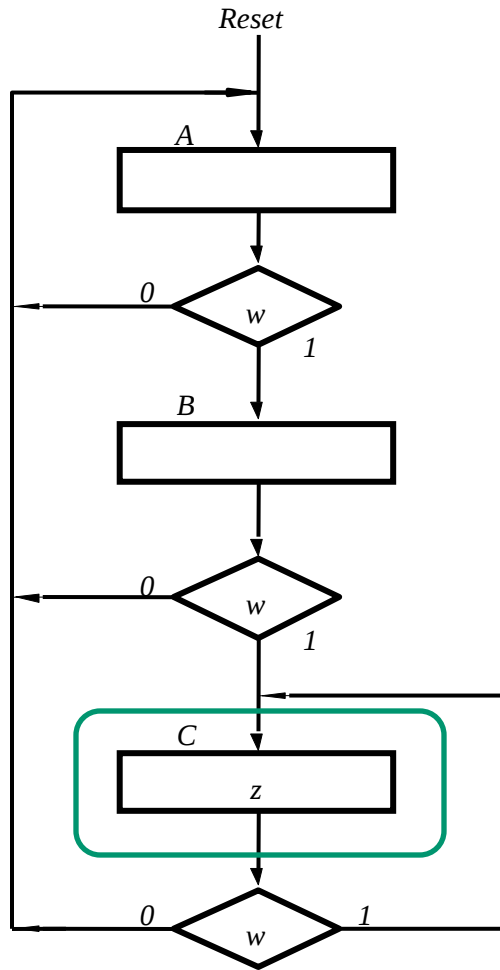
Beslutslåda (Decision Box)

**Beroende på värden på insignaler bestäms
övergången till nästa tillstånd**

Villkorlig utgångslåda (Conditional outputs)

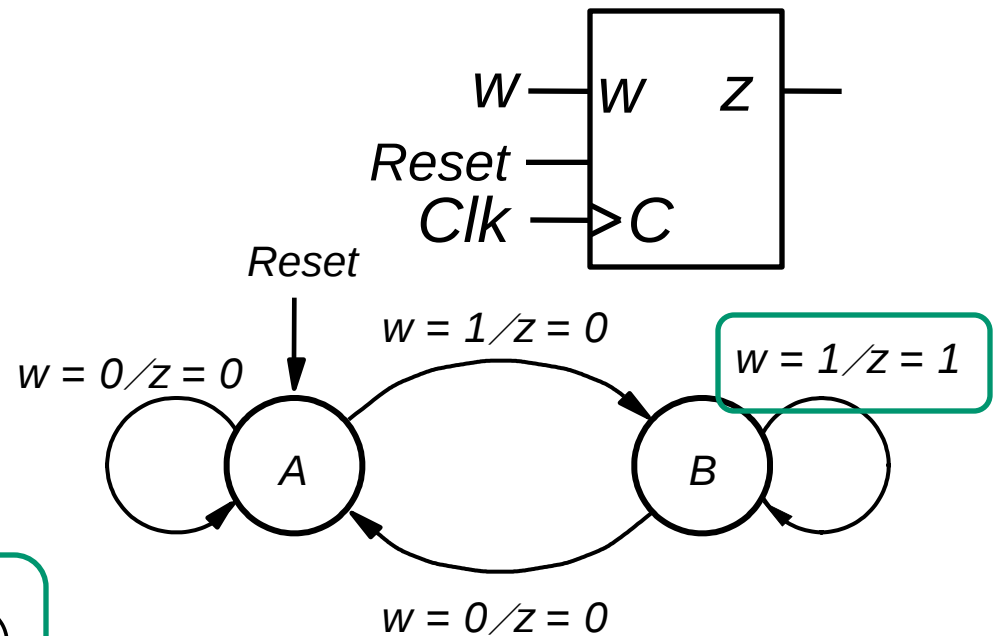
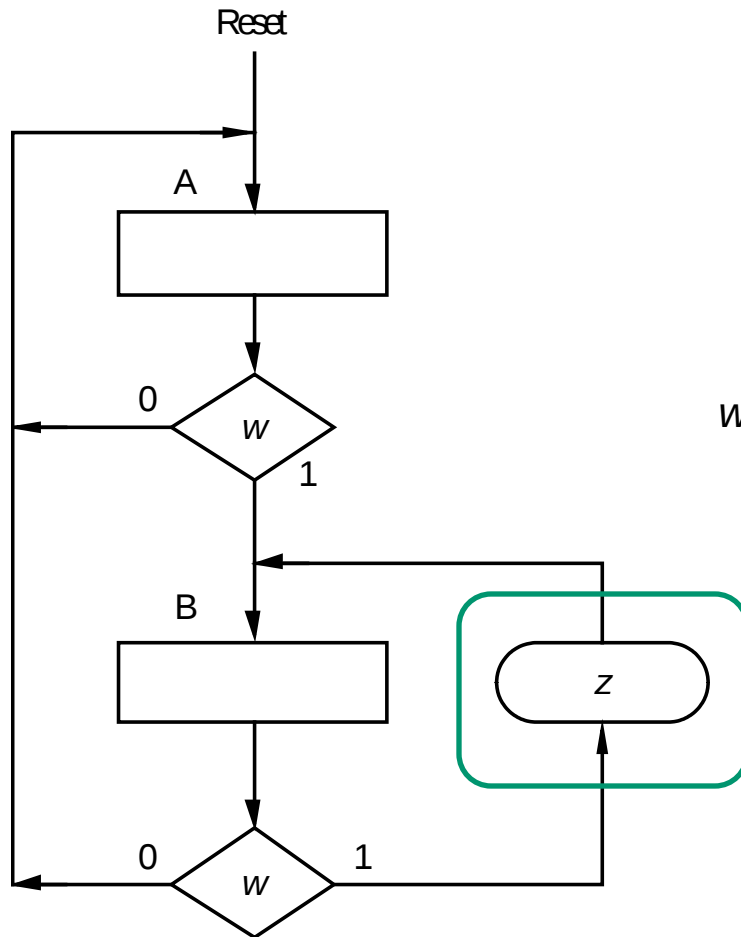
**Här anges värden av utgångarna vid en
tillståndsovergång (Mealy-outputs)**

”två i rad” Moore



Bara i tillstånd C har
z värdet 1

”två i rad” Mealy



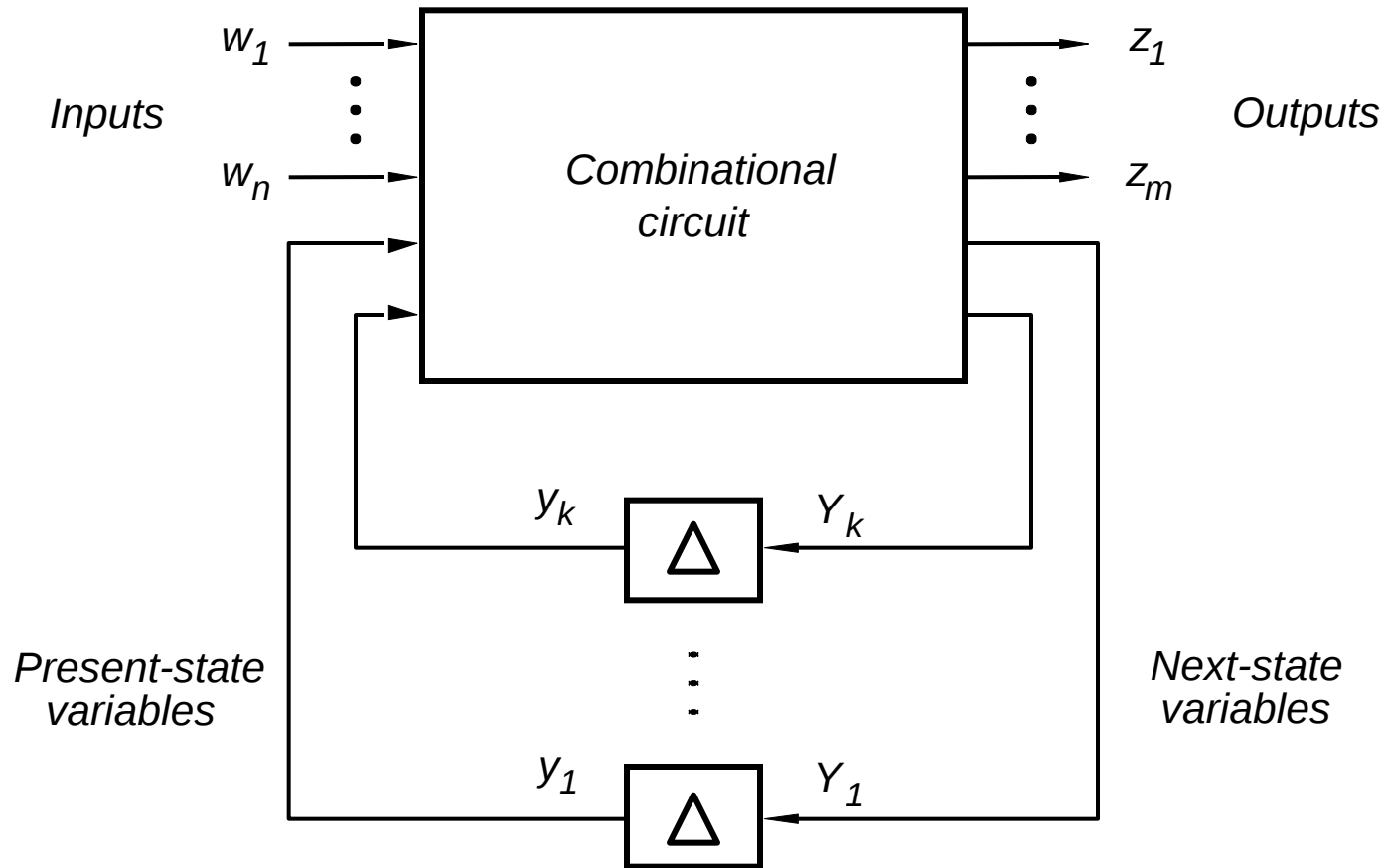
Bara vid tillståndsövergången
B-till-B med $w = 1$ har z värdet 1

Formell modell för tillståndsautomat

**För att behandla tillståndsmaskiner
matematiskt behöver man en formell
modell**

**Följande modell kan beskriva både
Moore- och Mealy-automaten**

Formell modell för tillståndsautomat



Formell modell för tillståndsautomat

En tillståndsmaskin kan formellt definieras med

$$M = (W, Z, S, \varphi, \lambda)$$

W , Z , och S beskriver ingångarna (W),
utgångarna (Z) och tillstånd (S)

φ beskriver tillståndsövergångsfunktionen

λ beskriver utgångsfunktion

Formell modell för tillståndsautomat

$$M = (W, Z, S, \varphi, \lambda)$$

$$S(t + \Delta t) = \varphi(W(t), S(t))$$

$$\lambda_{\text{Moore}}(t) = \lambda(S(t)) \quad \lambda_{\text{Mealy}}(t) = \lambda(W(t), S(t))$$

$$S(t + \Delta t) = Y_k \dots Y_1 = \varphi(w_n \dots w_1, y_k \dots y_1)$$

$$Z_{\text{Moore}} = z_m \dots z_1 = \lambda(y_k \dots y_1) \quad Z_{\text{Mealy}} = z_m \dots z_1 = \lambda(w_n \dots w_1, y_k \dots y_1)$$

Nu har Du sett hur man uttrycker detta i matematikämnet!

