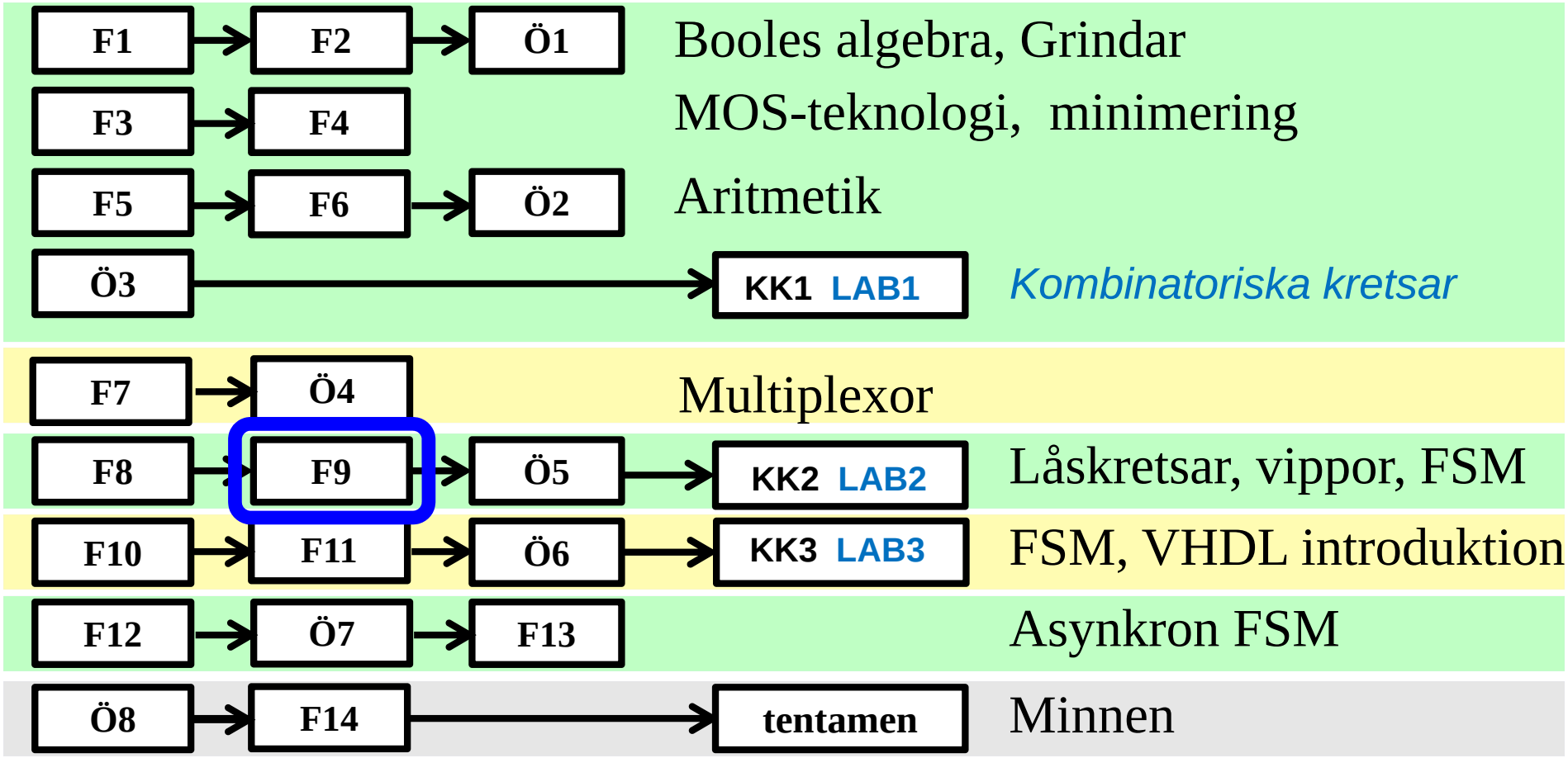
Digital Design IE1204

Föreläsningsbilder av William Sandqvist

F9 Tillståndsautomater del1

Carl-Mikael Zetterling
bellman@kth.se

IE1204 Digital Design



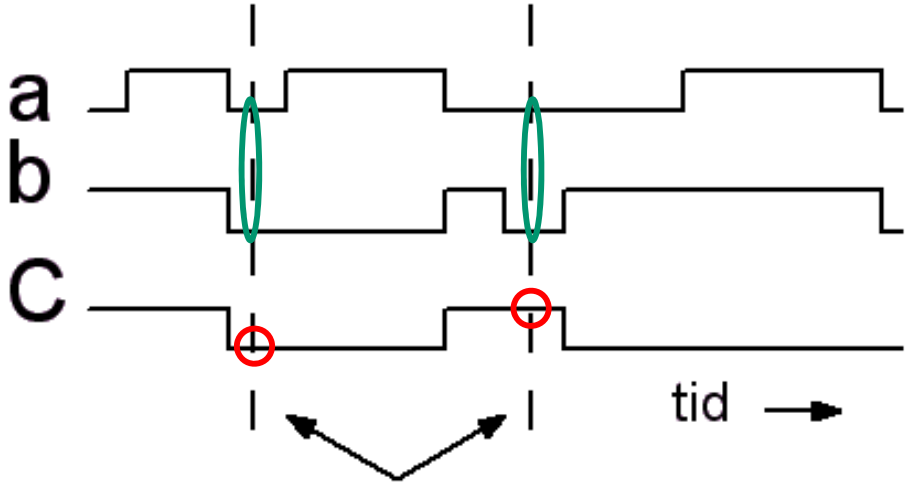
Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!

Detta har hänt i kursen …

Decimala, hexadecimala, oktala och binära talsystemen
AND OR NOT EXOR EXNOR Sanningstabell, mintermer Maxtermer PS-form
Booles algebra SP-form deMorgans lag Bubbelgrindar Fullständig logik
NAND NOR CMOS grindar, standardkretsar Minimering med Karnaugh-diagram 2, 3, 4, 5, 6 variabler
Registeraritmetik tvåkomplementrepresentation av binära tal
Additionskretsar Multiplikationskrets Divisionskrets
Multiplexorer och Shannon dekomposition Dekoder/Demultiplexor Enkoder
Prioritetsenkoder Kodomvandlare
VHDL introduktion

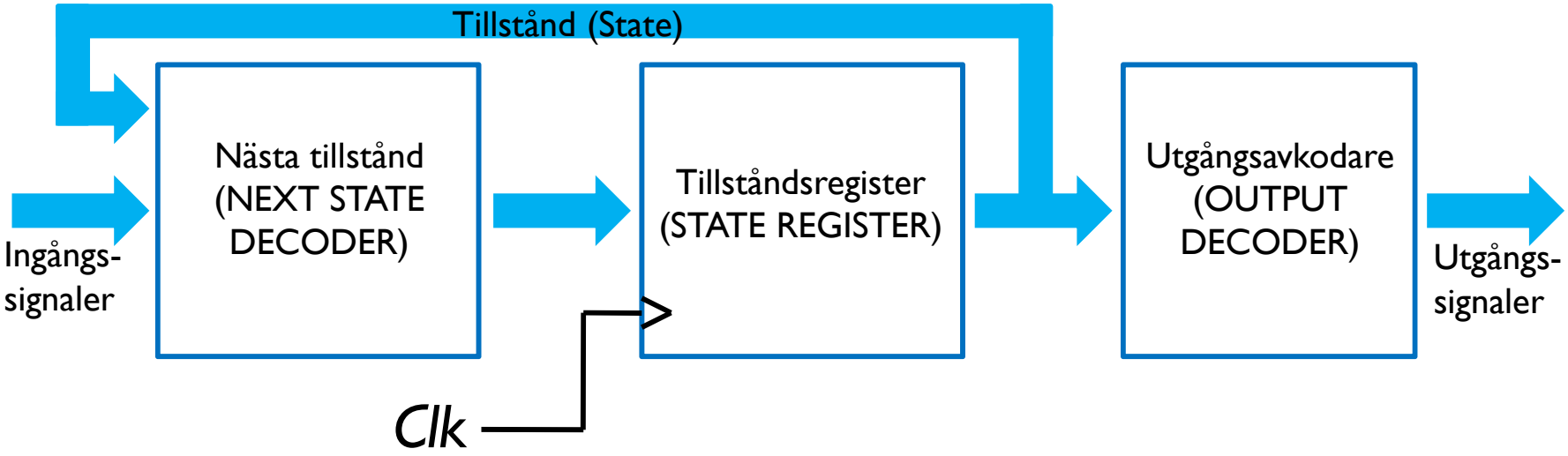
Vippor och Låskretsar SR-latch D-latch D-vippa JK-vippa T-vippa Räknare
Skiftregister Vippor i VHDL

Sekvensnät



Om en och samma insignal kan ge upphov till olika utsignal, är logiknätet ett sekvensnät.
Det måste då ha ett ***inre minne*** som gör att utsignalen påverkas av både nuvarande och föregående insignaler!

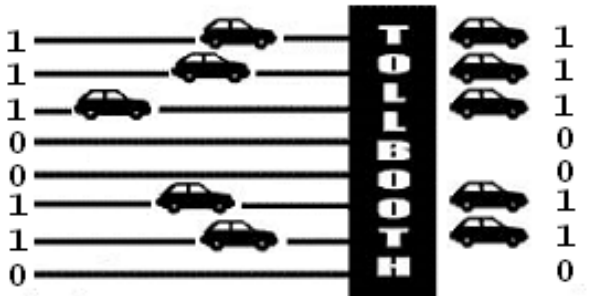
Moore-automat

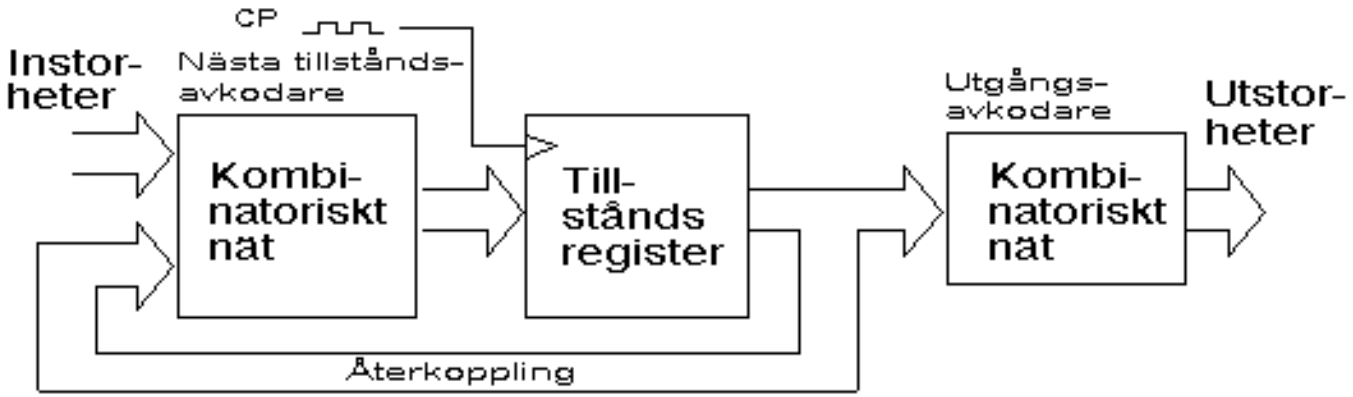


För Moore-automaten beror utsignalerna på insignalerna och det inre tillståndet. Det inre minnet är tillståndsregistret som består av **D-vippor**.

Tillståndsregistrets D-vippor

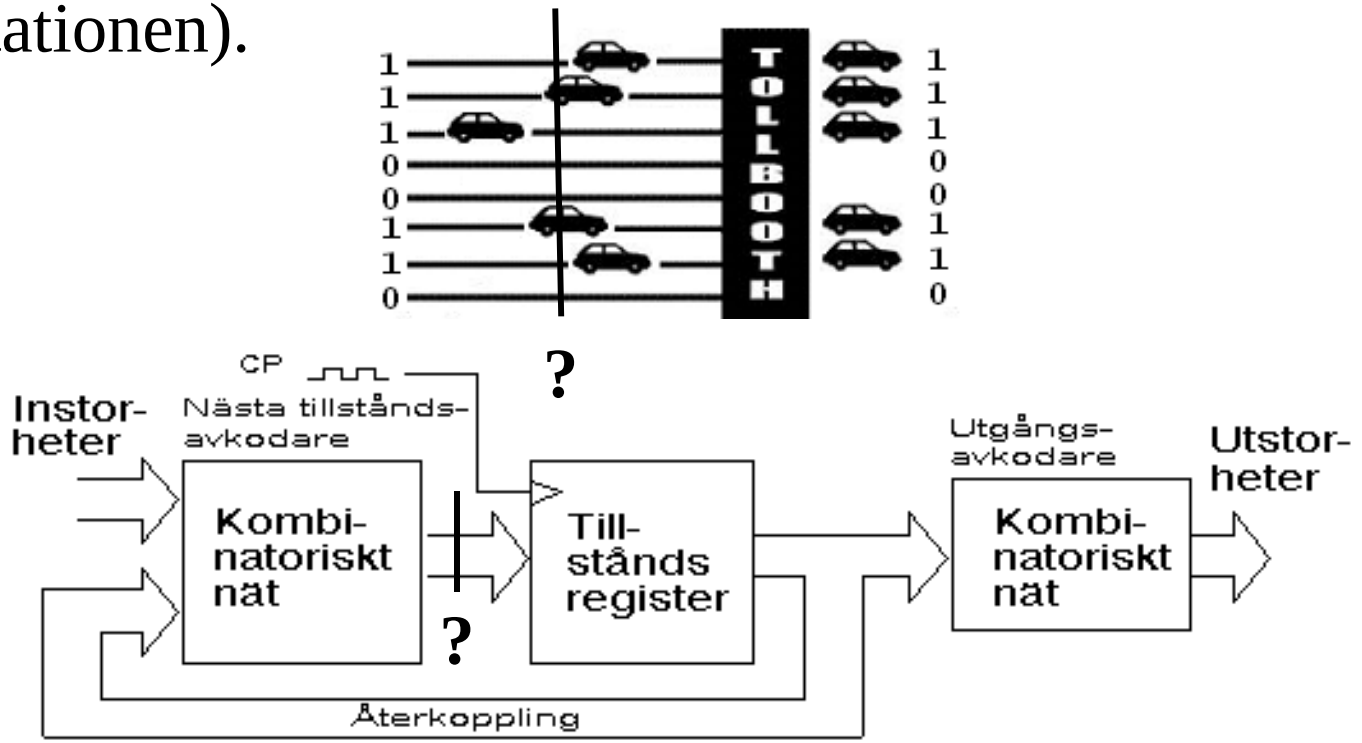
Tillståndsregistrets D-vippor bromsar upp *kapplöpningen* mellan signalerna tills värdet är stabilt. (Jämför med tullstationen).





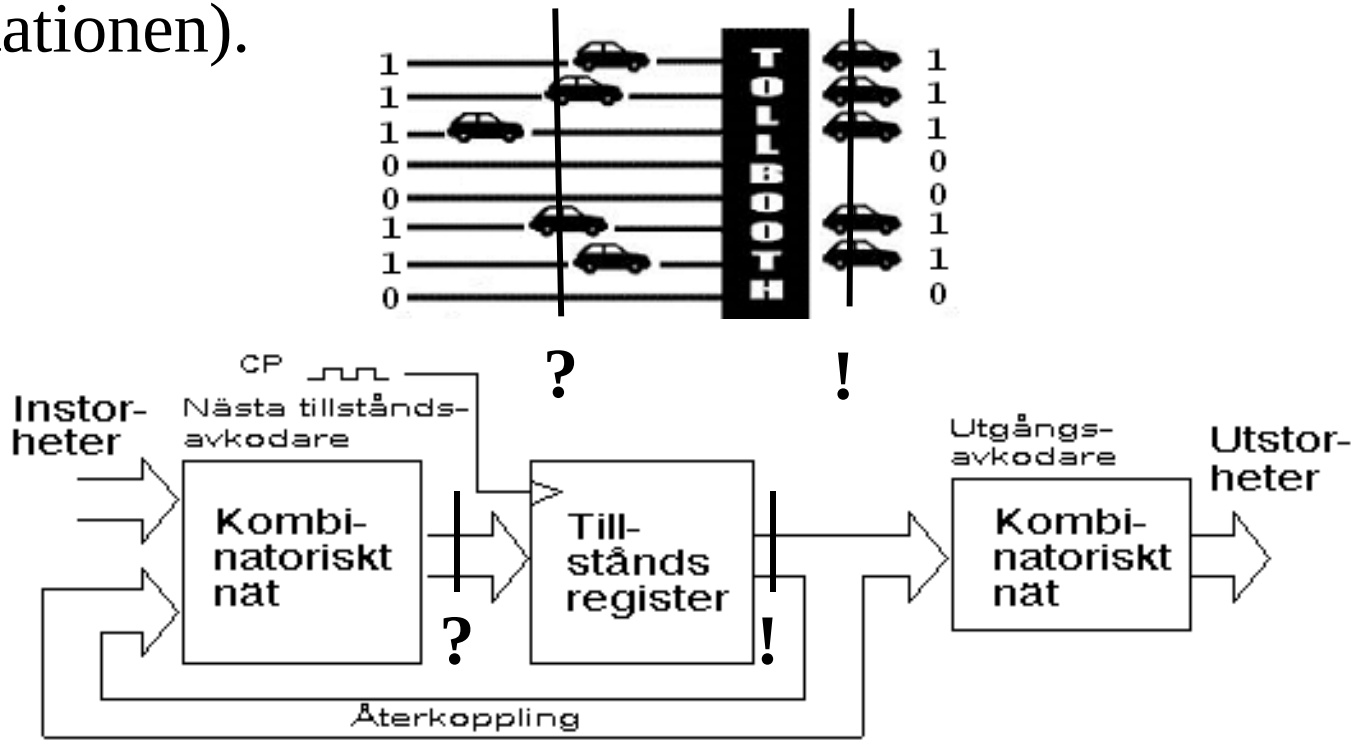
Tillståndsregistrets D-vippor

Tillståndsregistrets D-vippor bromsar upp *kapplöpningen* mellan signalerna tills värdet är stabilt. (Jämför med tullstationen).



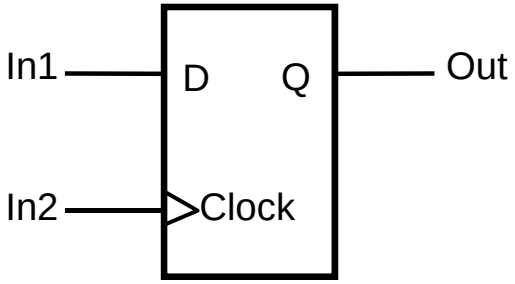
Tillståndsregistrets D-vippor

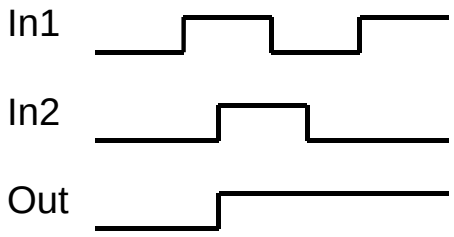
Tillståndsregistrets D-vippor bromsar upp *kapplöpningen* mellan signalerna tills värdet är stabilt. (Jämför med tullstationen).



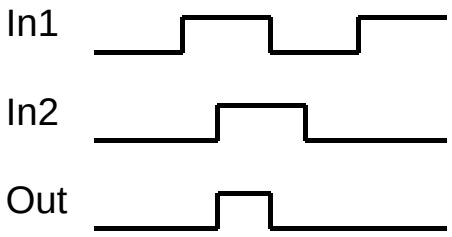
Snabbfråga Vippor

Vilket av följande tidsdiagram är giltigt för en flanktriggad D flip-flop?

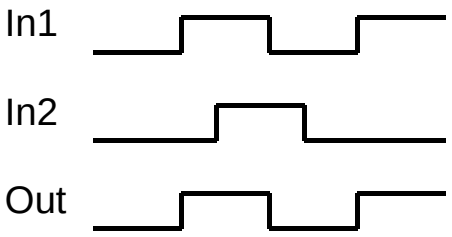




Alt: A



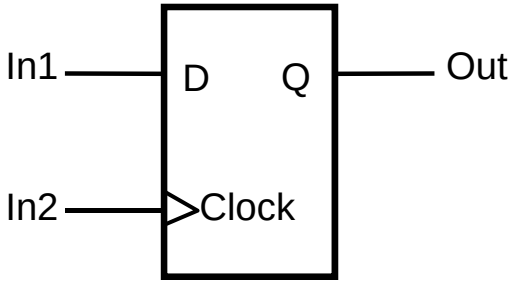
Alt: B

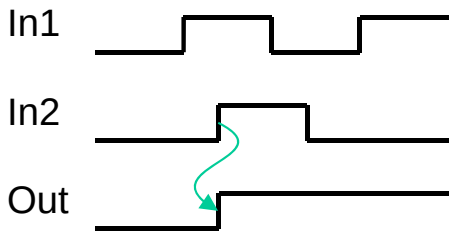


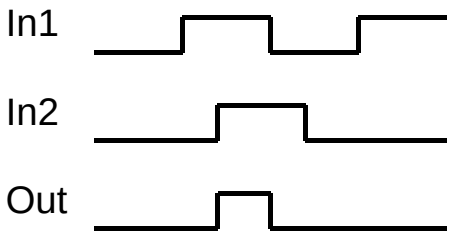
Alt: C

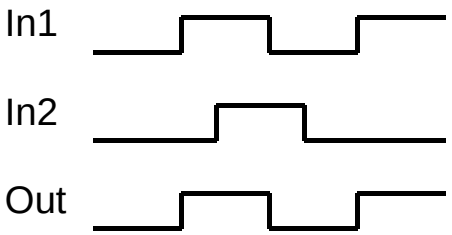
Snabbfråga Vippor

Vilket av följande tidsdiagram är giltigt för en flanktriggad D flip-flop?









Alt: A

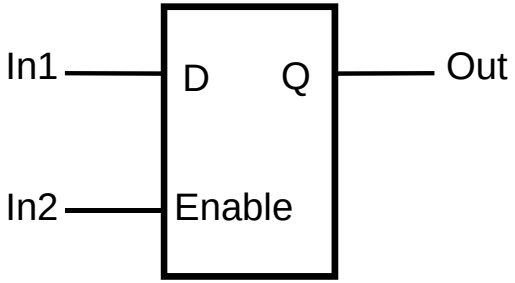
Alt: B

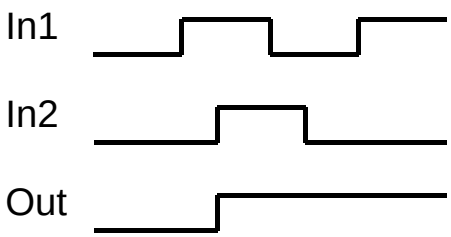
Alt: C

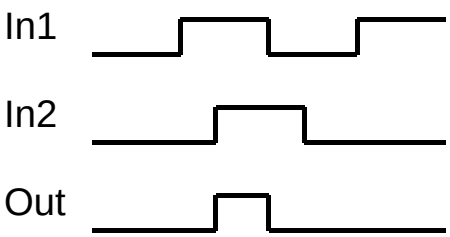
D kopieras till utgången vid flanken,
dvs när Clock går från 0 till 1

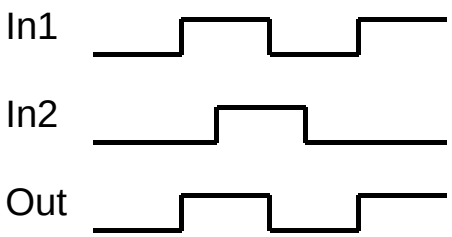
Snabbfråga Latch

Vilket av följande tidsdiagram är giltigt för en D Latch?









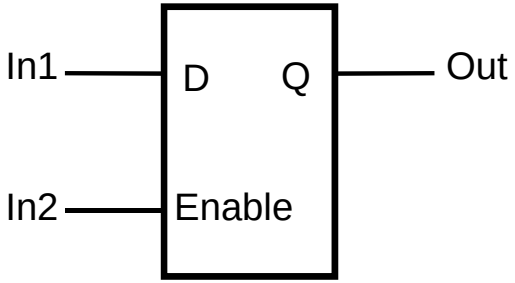
Alt: A

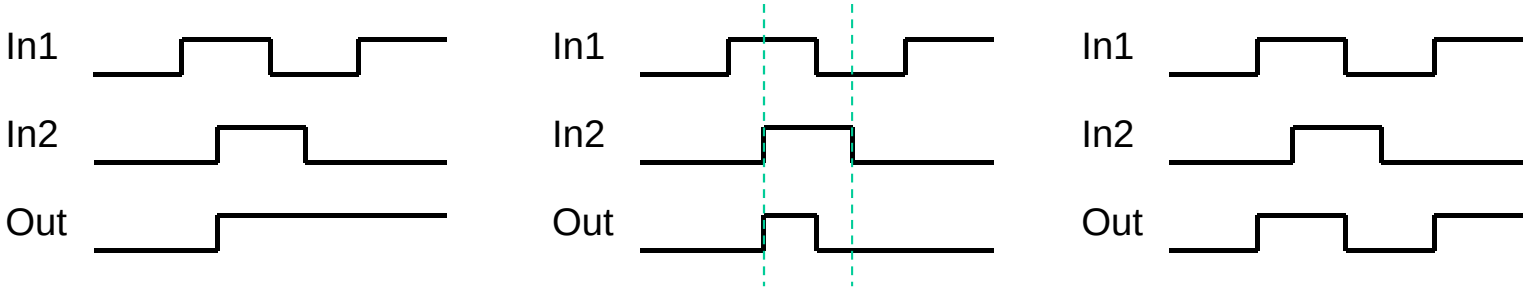
Alt: B

Alt: C

Snabbfråga Latch

Vilket av följande tidsdiagram är giltigt för en D Latch?





Alt: A

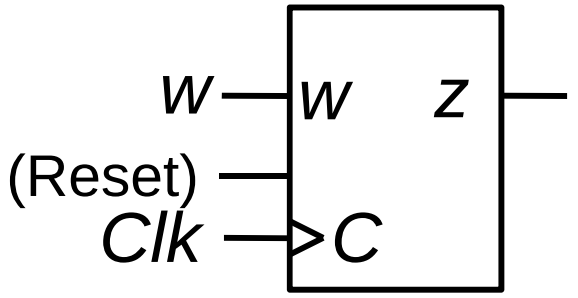
Alt: B

Alt: C

D kopplas till utgången när Enable är 1, låses när Enable är 0

Designexempel ”två i rad”

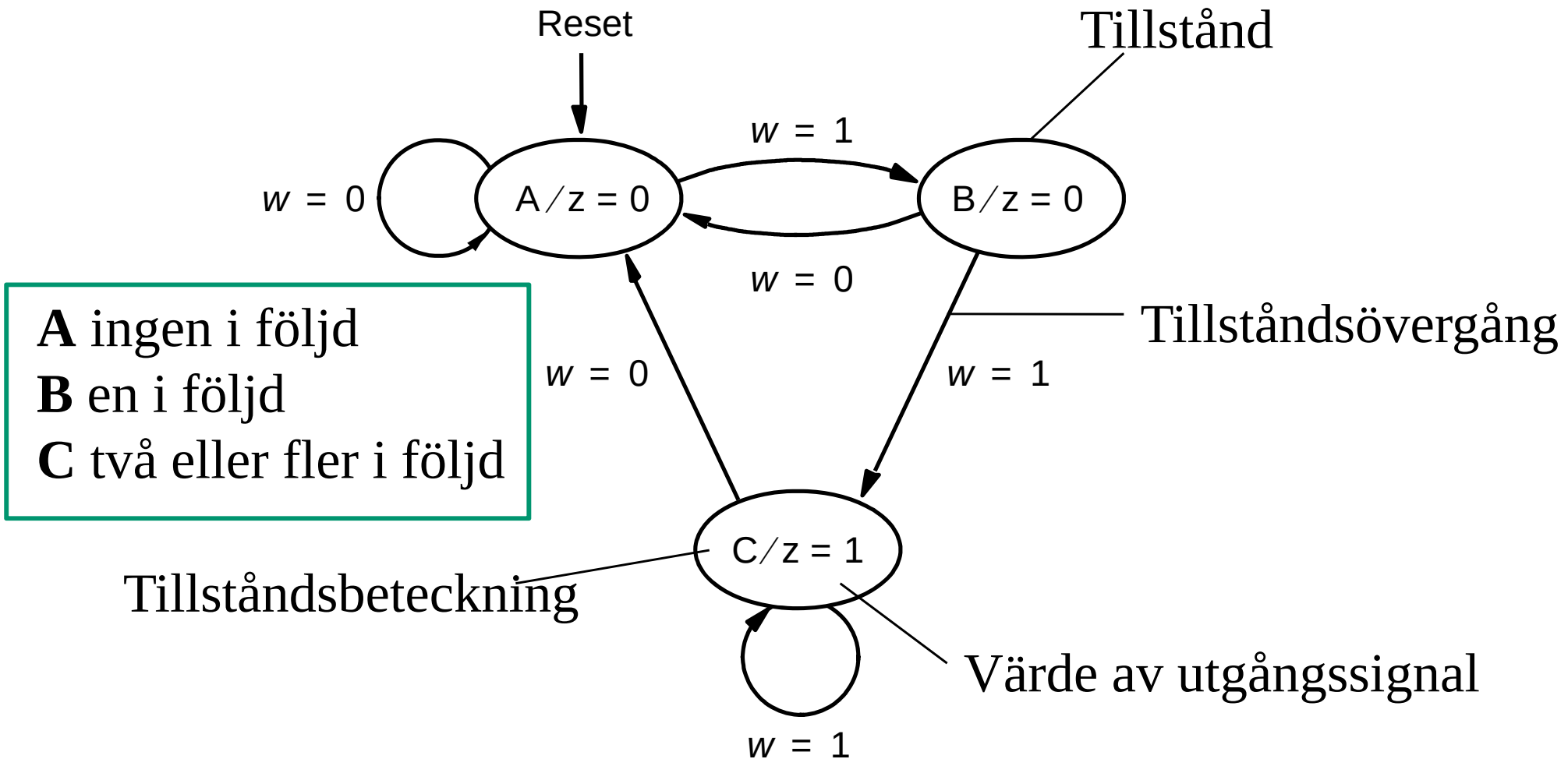
Sekvensdetektor. Om *w* har varit 1 under två (eller fler) klockcykler i rad skall $z = 1$.



Specifikation

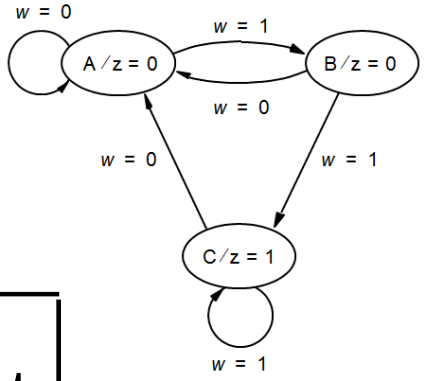
Sekvenskretsen har en ingång *w* och en utgång *z*
Om ingången *w* har varit 1 under nuvarande och föregående klockcykel så ska utgången *z* sättas till 1
Använd en Moore-automat med D-vippor för att realisera konstruktionen

Tillståndsdiagram ”två i rad”



A ingen i följd
B en i följd
C två eller fler i följd

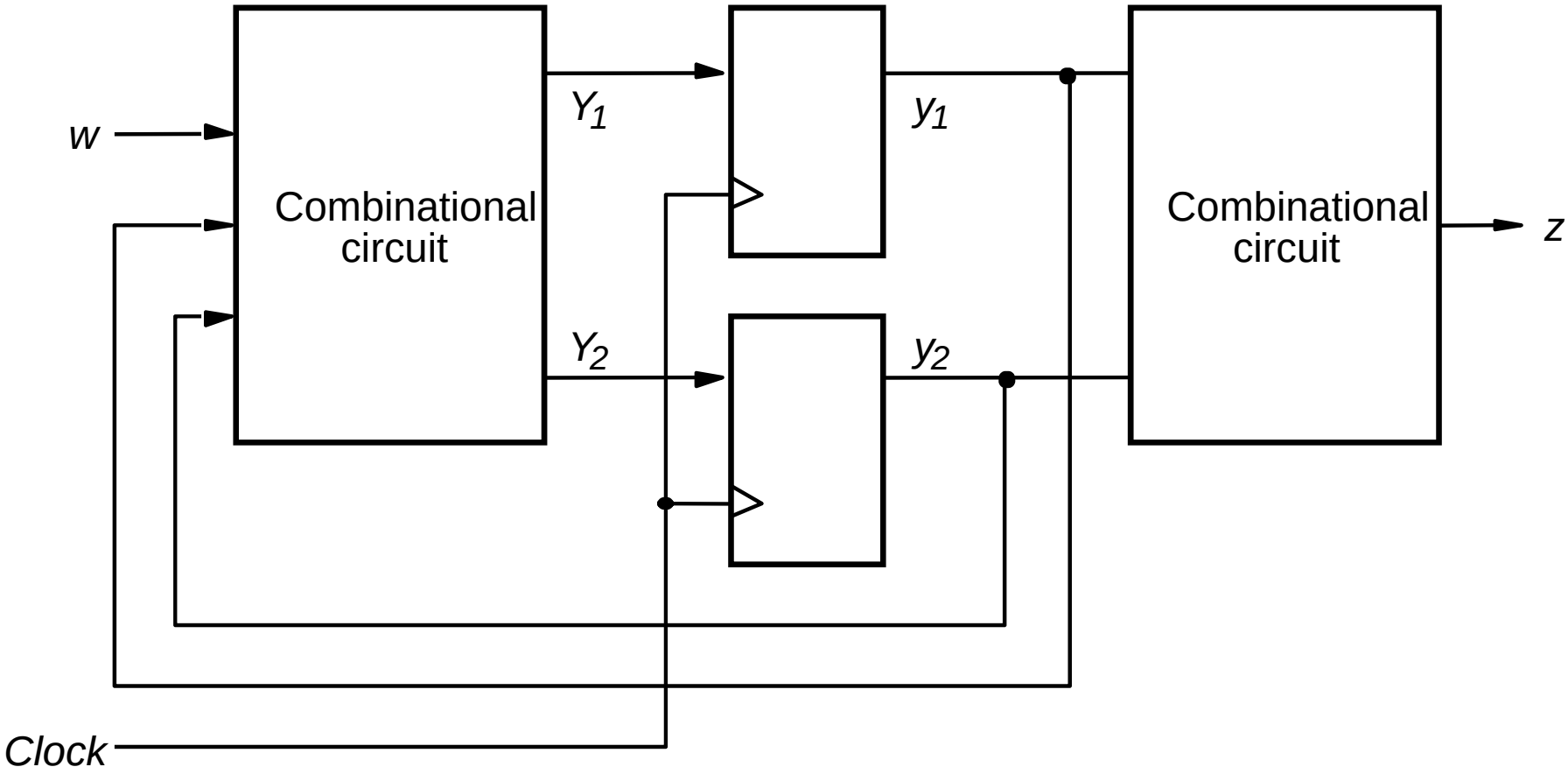
Tillståndstabell



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

Tre tillstånd – två vippor behövs
för att minnas tillståndets nummer!

”två i rad” som Moore-automat



Designbeslut

- Designern måste bestämma vilka vippor som ska användas

  D-, T-, eller JK-vippa
- Designern måste välja koden för varje tillstånd

Designbeslut

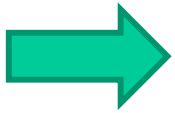
Denna gång givet:

- D-vippor
- Tillståndsavkodning A = 00, B = 01, C = 10
- Koden 11 ska *inte* förekommer.
  Vi väljer don’t care.

Kodad tillståndstabell

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

A = 00
B = 01
C = 10



	Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

$$Y_2Y_1 = f(y_2y_1w) \quad z = f(y_2y_1)$$

Nästa tillståndsavkodare

Nästa tillståndsavkodaren består av de två logiknäten som finns som ingångsnät till de två vipporna.
För att kunna minimera logiknäten skriver man in sanningstabellerna i Karnaughdiagram.

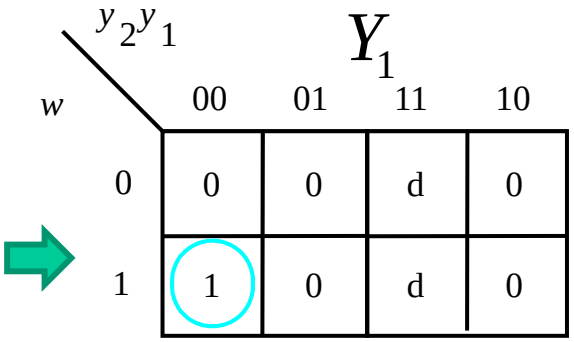
$$Y_2Y_1 = f(y_2y_1w) \quad Y_2 = f(y_2y_1w) \quad Y_1 = f(y_2y_1w)$$

$$z = f(y_2y_1)$$

Från kodad tillståndstabell till Karnaughdiagram

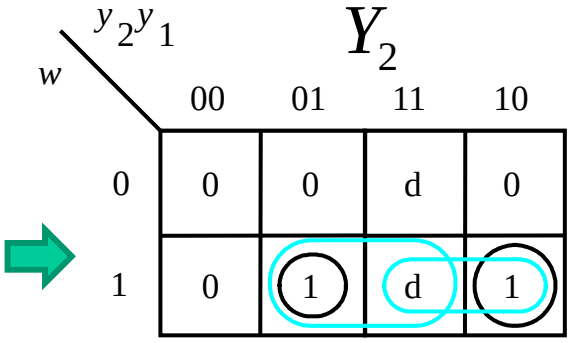
$$Y_2Y_1 = f(y_2y_1w) \quad Y_2 = f(y_2y_1w) \quad Y_1 = f(y_2y_1w)$$

Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*



$$Y_1 = w\bar{y}_1\bar{y}_2$$

Present state y_2y_1	Next state $w=0$ Y_2Y_1	Next state $w=1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*

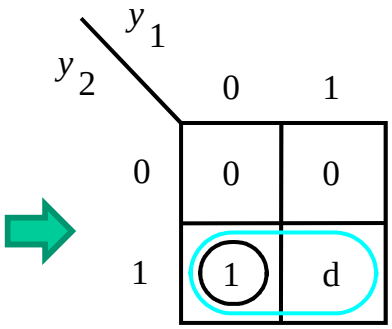


$$Y_2 = wy_1 + wy_2 =$$
$$= w(y_1 + y_2)$$

Utgångsavkodaren

$$z = f(y_2 y_1)$$

Present state y_2y_1	Next state $w = 0$ Y_2Y_1	Next state $w = 1$ Y_2Y_1	Output z
00	00	01	0
01	00	10	0
10	00	10	1
11	*dd*	*dd*	*d*



$$z = y_2$$

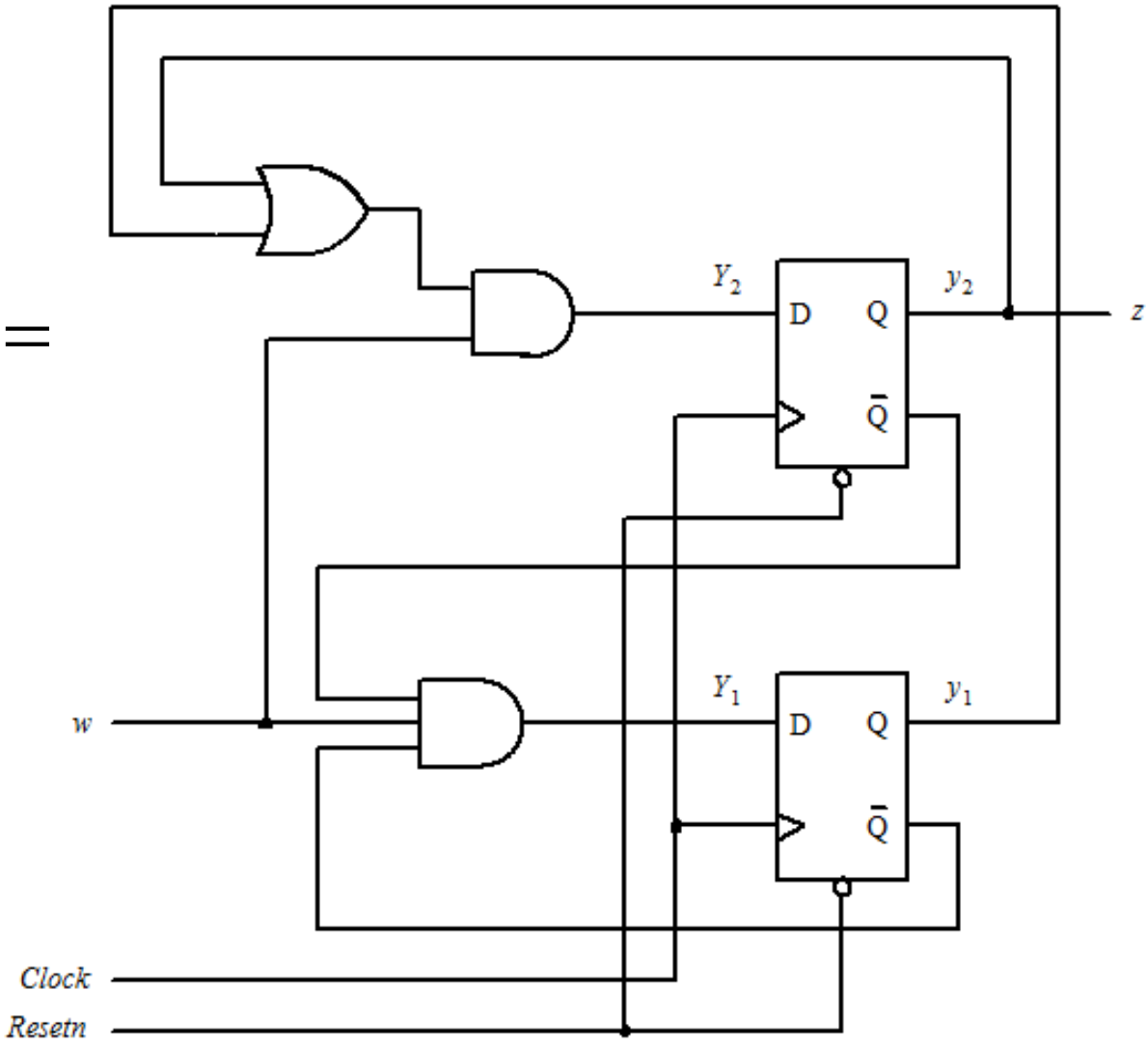
Implementeringen

$$Y_2 = wy_1 + wy_2 =$$

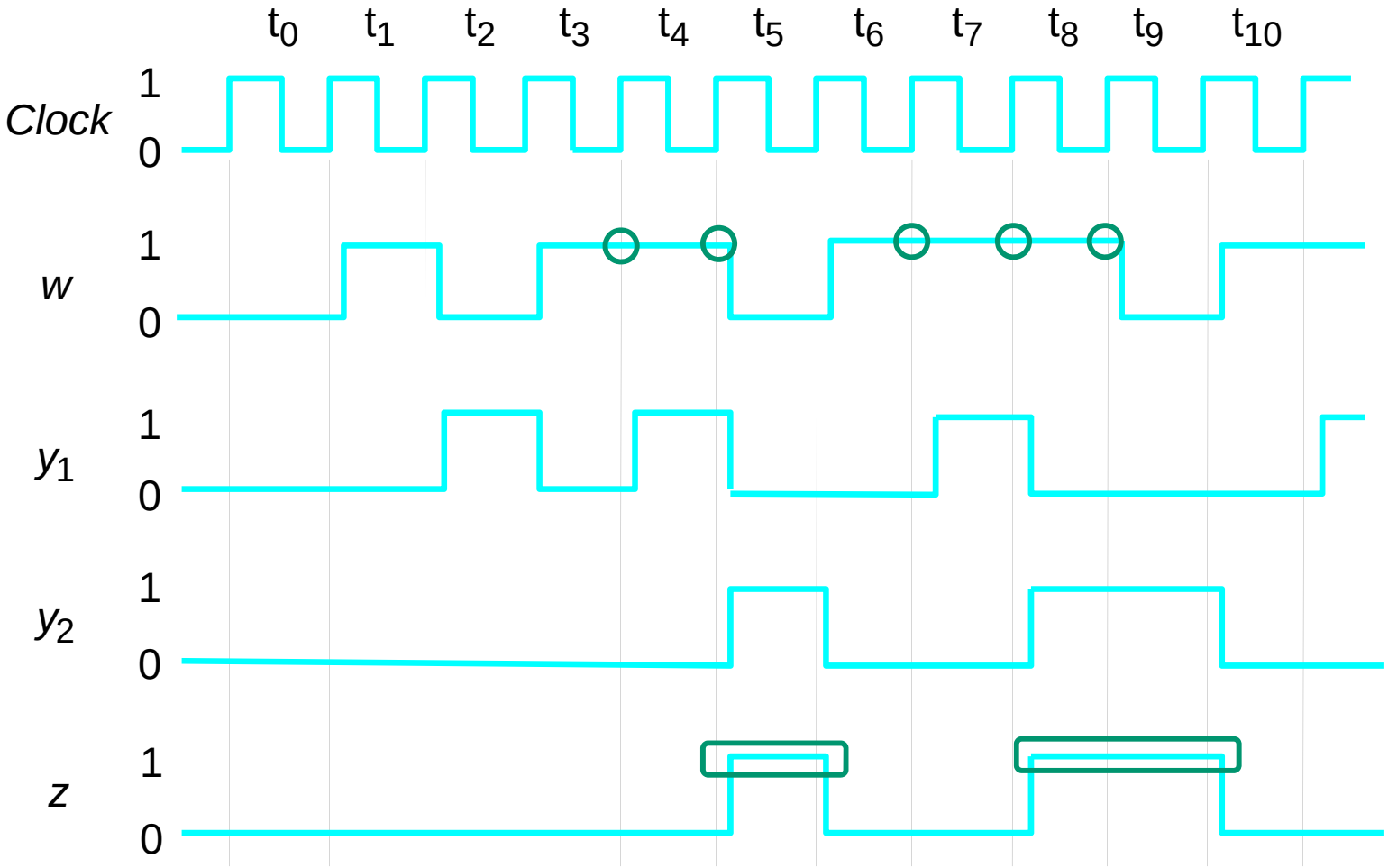
$$= w(y_1 + y_2)$$

$$Y_1 = w\bar{y}_1\bar{y}_2$$

$$z = y_2$$



Tidsdiagram ”två i rad”



Tillståndsöver-gångar sker bara på den positiva klockflanken!

Med andra beteckningar

Nuvarande tillstånd Nästa tillstånd

$$y \quad y^+$$

Övningshäfte och Hemert:

$$y_2^+ y_1^+ = f(y_2 y_1 w) \quad y_2^+ = f(y_2 y_1 w) \quad y_1^+ = f(y_2 y_1 w)$$

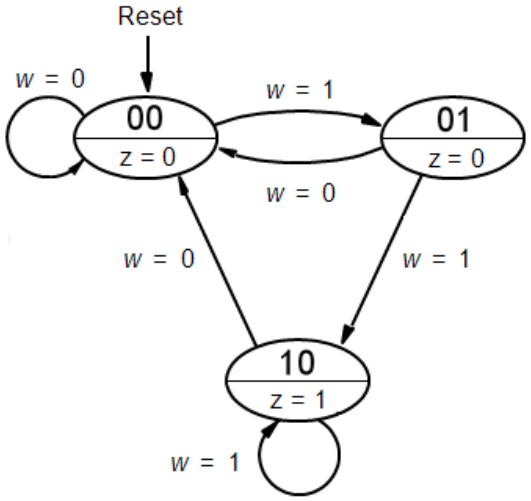
Med annan uppställning

$$y_2^+ y_1^+ = f(y_2 y_1 w) \quad y_2^+ = f(y_2 y_1 w) \quad y_1^+ = f(y_2 y_1 w)$$

$y_2 y_1$	$y_2^+ y_1^+$ $w=0$	$y_2^+ y_1^+$ $w=1$
A 00	00	01
B 01	00	10
11	*d d*	*d d*
C 10	00	10

$y_2 y_1$	y_2^+ $w=0$	y_2^+ $w=1$
00	0	0
01	0	1
11	*d*	*d*
10	0	1

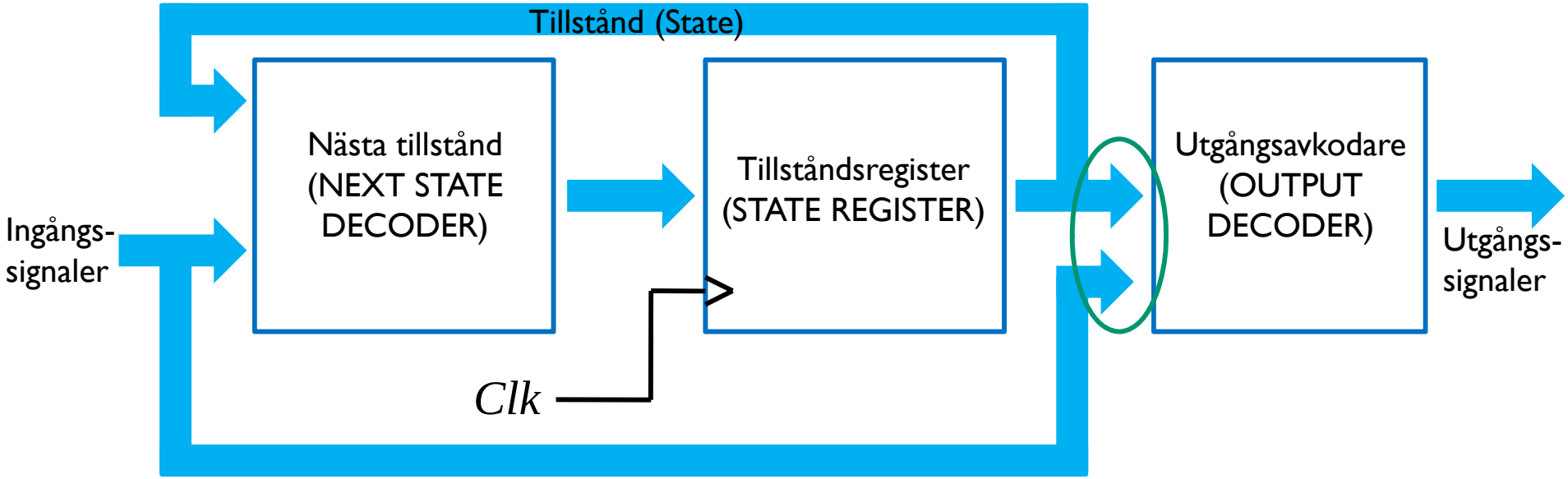
$y_2 y_1$	y_1^+ $w=0$	y_1^+ $w=1$
00	0	1
01	0	0
11	*d*	*d*
10	0	0



Övningshäftet använder denna metod

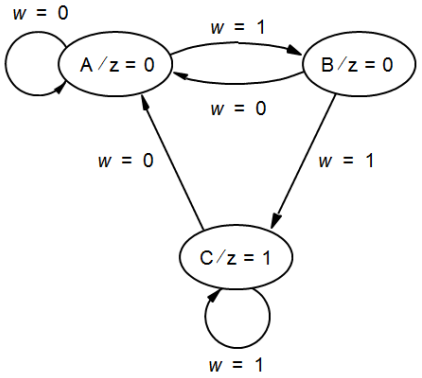
Man kan direkt ställa upp den kodade tillståndstabellen som ett ”Karnaugh-diagram”.

Mealy-automat



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

Moore automaten



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

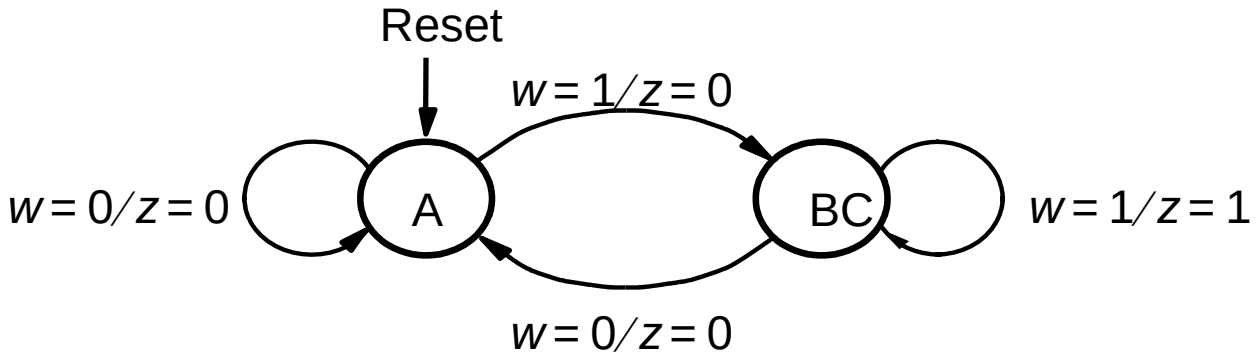
Mealy automaten

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

Två av tillstånden som bara skiljer i utsignal kan slås ihop – insignalen kan användas för att skilja utsignalerna!

Mealy automaten

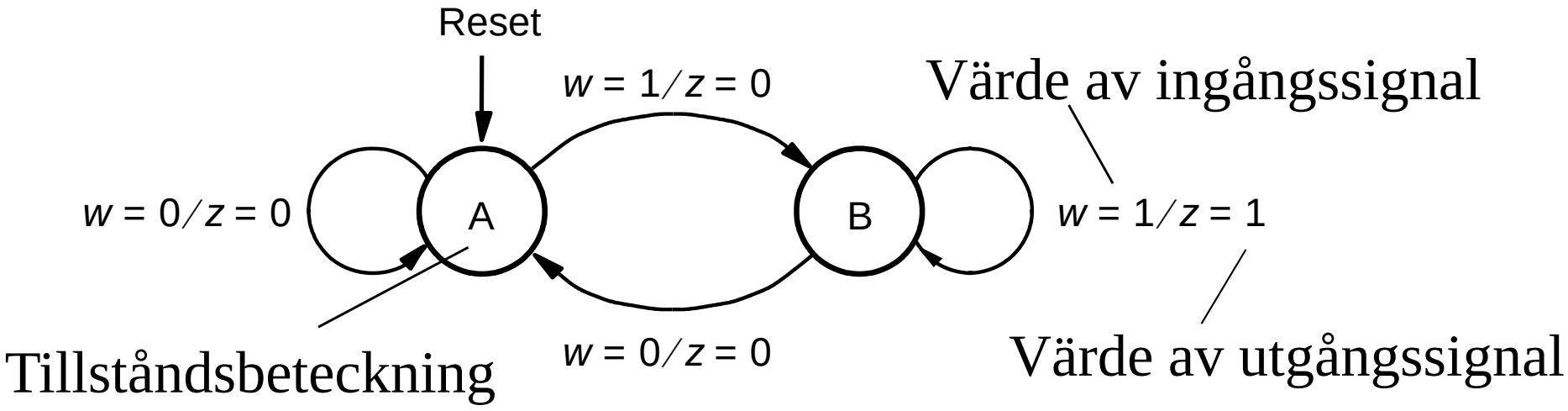
Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	BC	0	0
BC	A	BC	0	1



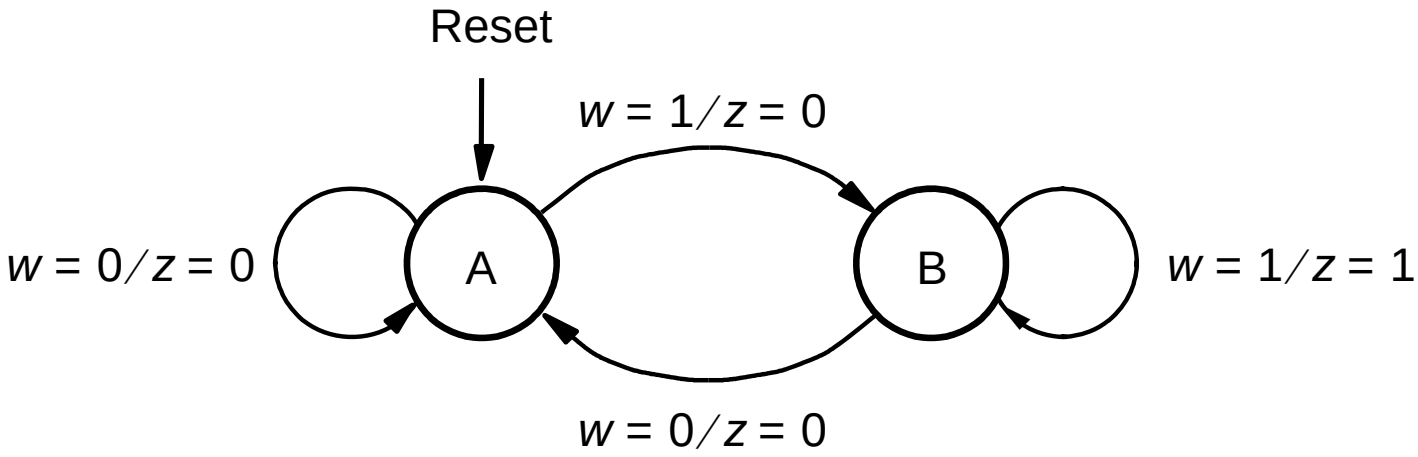
Tillståndsdiagram Mealy

”Två i rad”

- Tillståndsdiagrammet för Mealy-automaten behöver bara två tillstånd
- Utsignalen beror på ***både*** tillstånd och insignaler



Tillståndstabell



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

Två tillstånd – bara en vippa behövs!

Kodad tillståndstabell

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

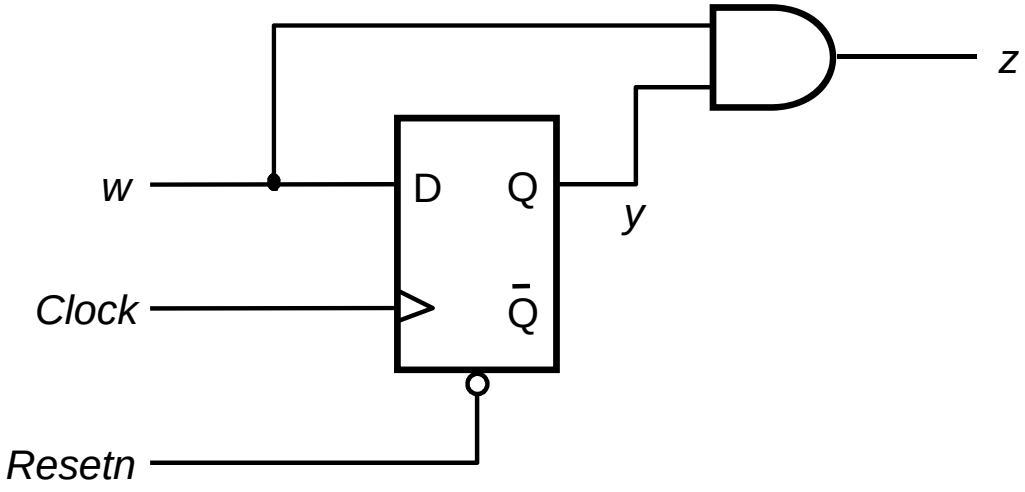
$$Y = f(y\,w) \quad z = f(y\,w)$$

A = 0
B = 1

Present state	Next state		Output	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
y	Y	Y	z	z
A: 0	0	1	0	0
B: 1	0	1	0	1

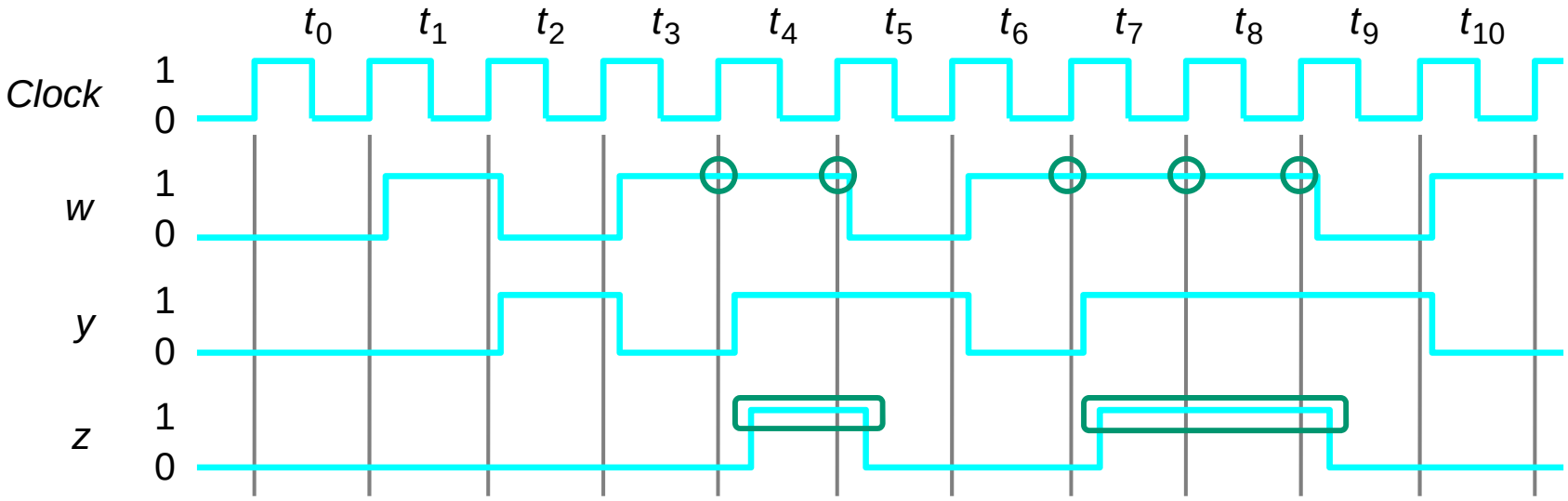
Direkt ur tabellen: $Y = w \quad z = yw$

Implementeringen



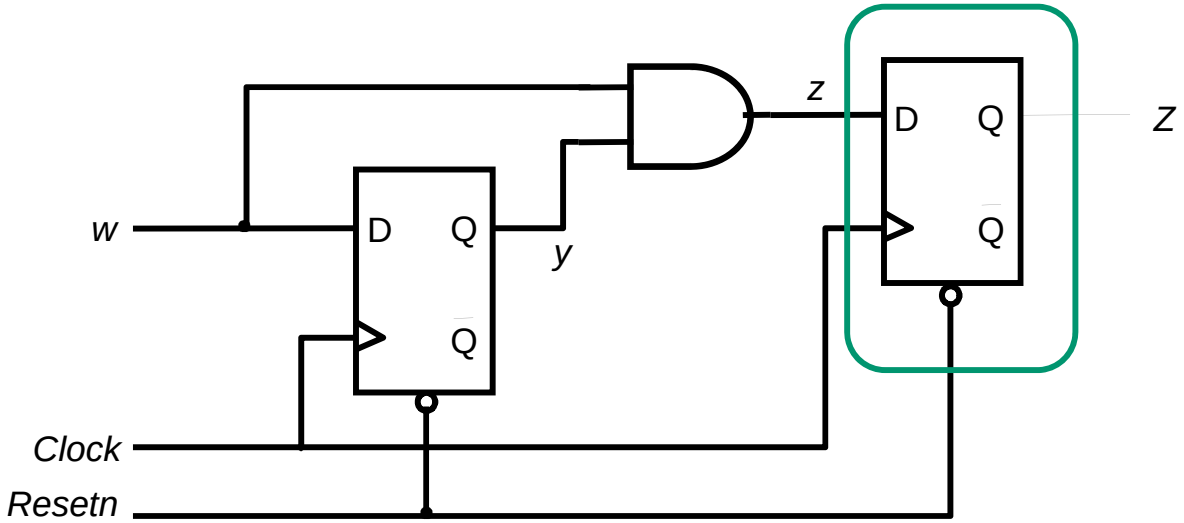
Tidsdiagram

- Utsignalen kan ändrar sig under hela klockperioden eftersom den är en funktion av insignalen
- Jämfört med Moore-automaten så ‘reagerar’ Mealy-automaten *tidigare* (bitsekvensen detekteras i t_4 jämfört med t_5 i Moore-automaten)



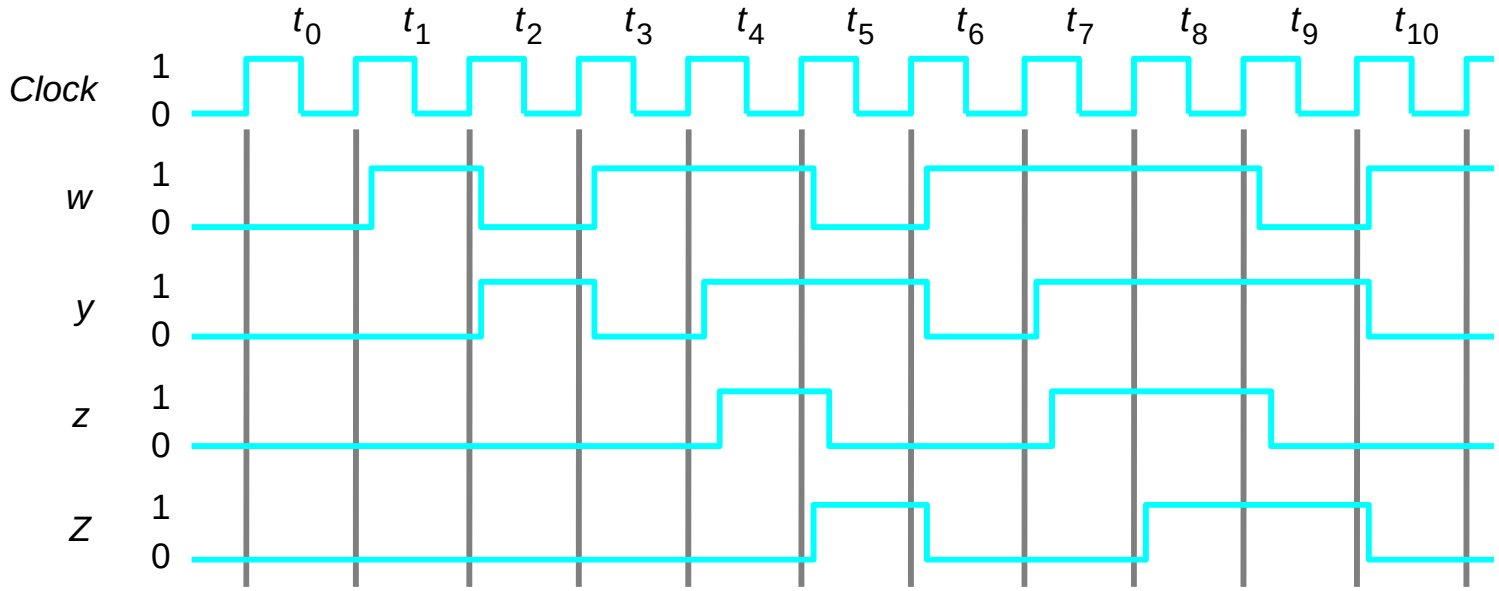
Mealy med utgångsregister

- Nackdelen med Mealy-automaten är att utsignalen kan ändras under hela klockperioden
- Man kan lägga till en register (vippa) på utgången så för att synkronisera utgången med klockflanken



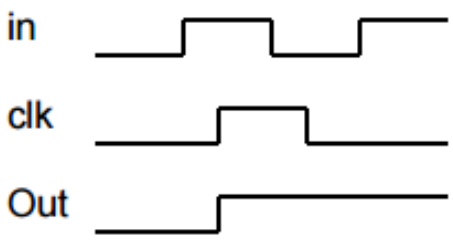
Tidsdiagram med utgångsregister

Med ett utgångsregister så försvinner skillnaden mellan tidsdiagrammen!

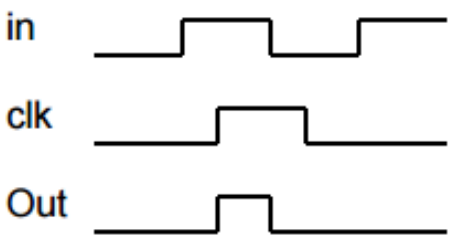


Snabbfråga automater

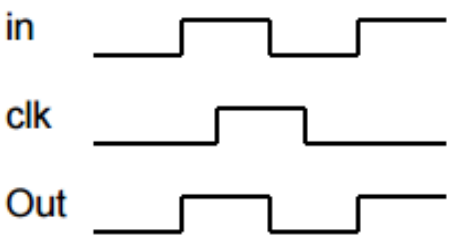
Vilket/vilka av följande tidsdiagram kan genereras från en Moore automat?



Alt: A



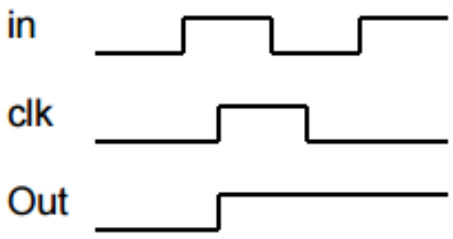
Alt: B



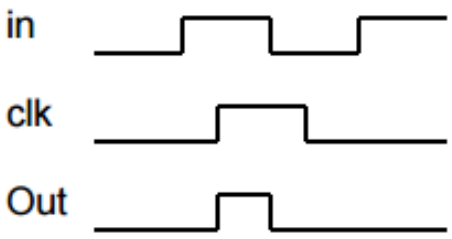
Alt: C

Snabbfråga automater

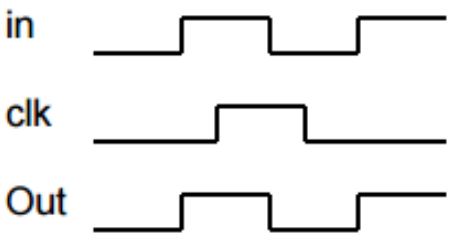
Vilket/vilka av följande tidsdiagram kan genereras från en Moore automat?



Alt: A



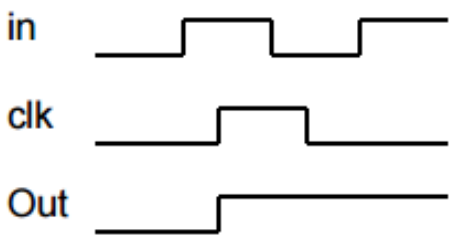
Alt: B



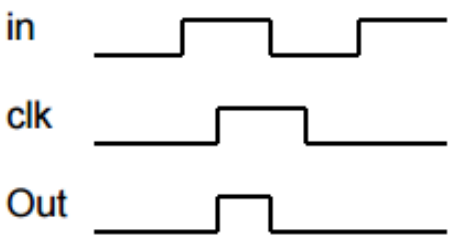
Alt: C

Snabbfråga automater

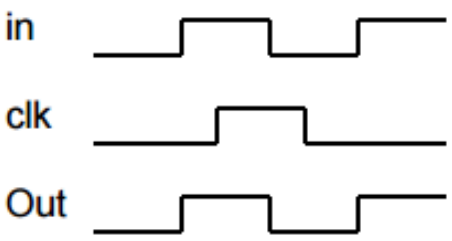
Vilket/vilka av följande tidsdiagram kan genereras från en Mealy automat?



Alt: A



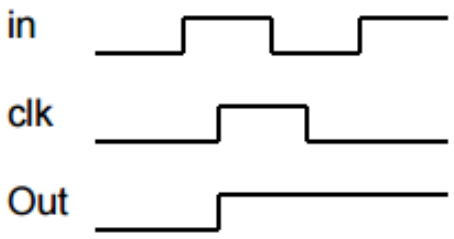
Alt: B

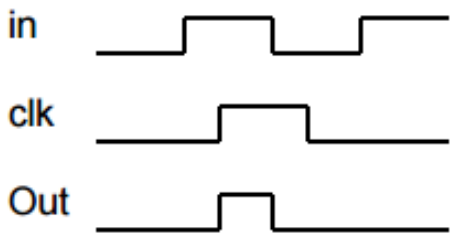


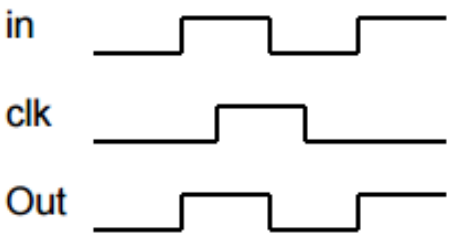
Alt: C

Snabbfråga automater

Vilket/vilka av följande tidsdiagram kan genereras från en Mealy automat?







Alt: A

Alt: B

Alt: C

Moore vs Mealy

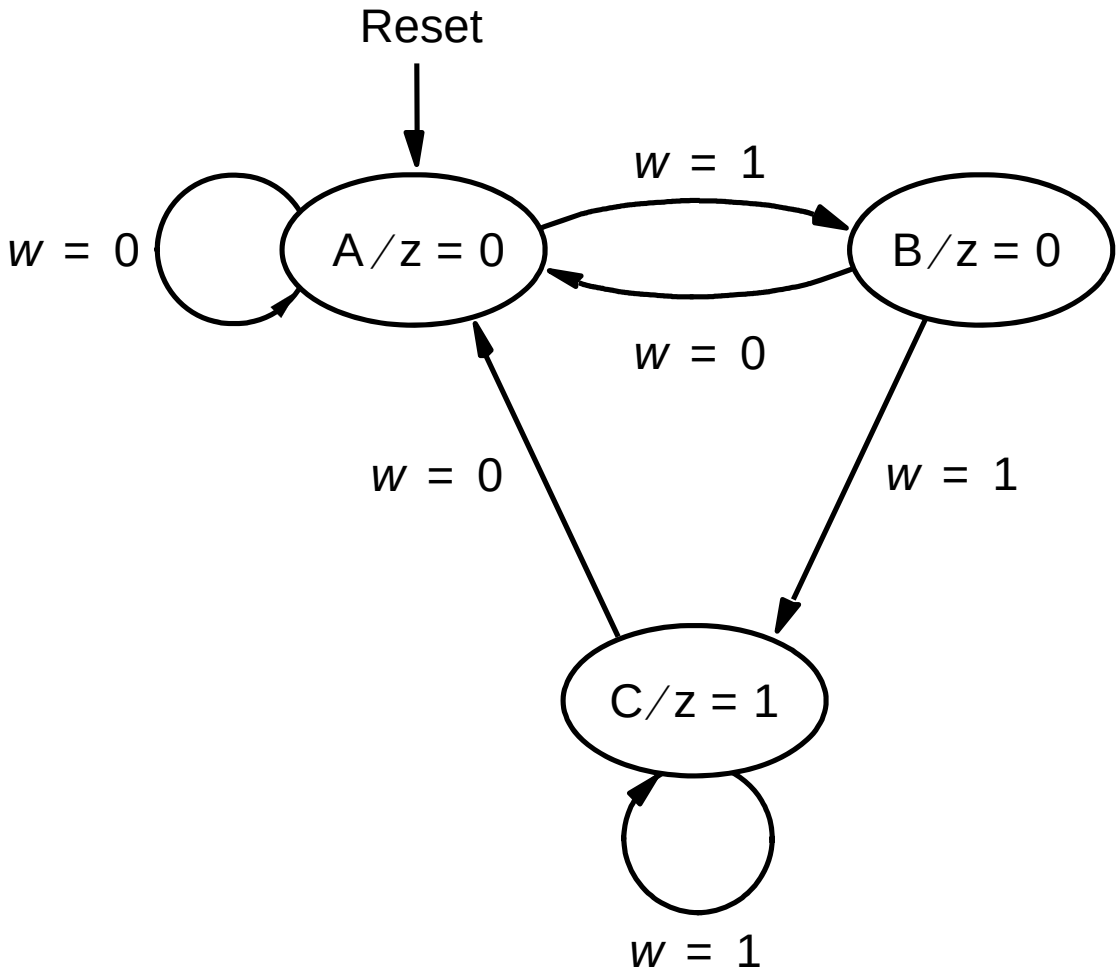
- Moore-automatens utgångsvärden beror bara på det nuvarande tillståndet
- Mealy-automatens utgångsvärden beror på det nuvarande tillståndet och värden på ingångssignalerna
- Mealy-automaten behöver ofta färre tillstånd
- Mealy-automatens utsignaler är *inte synkroniserade* med klockan, varför man ofta lägger till ett utgångsregister

Val av tillståndskodning

Valet av tillståndskodningen kan spela en stor roll för implementeringen eftersom den påverkar logiken för

- Next-state-decoder
- Utgångsavkodare

”två i rad” tillståndsdiagram



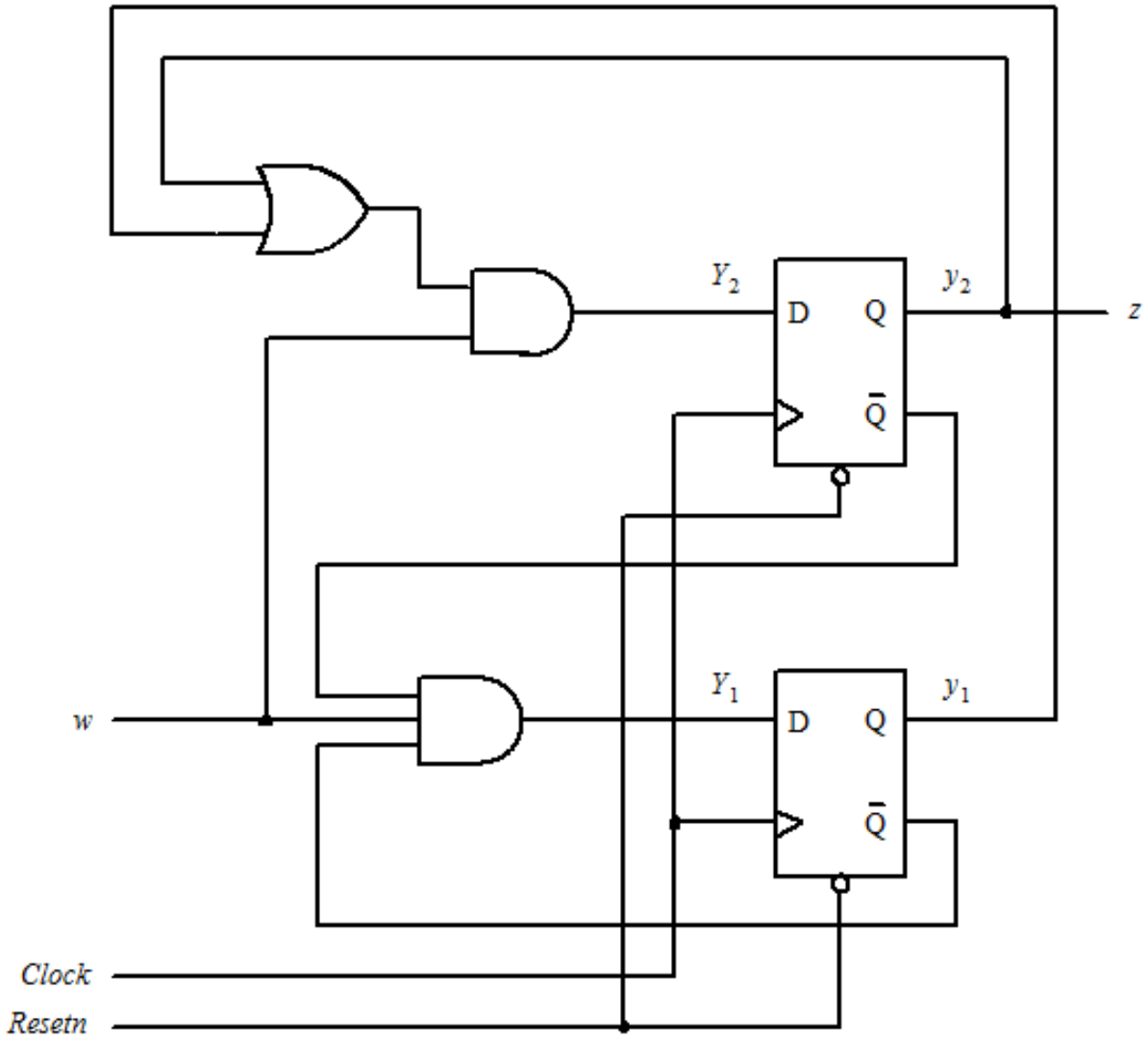
Tillståndskod = Binärkod

A = 00
B = 01
C = 10
11

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Realisering (Binärkod)

2 D-vippor
2 AND-grindar
1 OR-grind



Tillståndskod = Graykod

- I Gray-koden ändras bara en bit åt gången, dvs 00, 01, 11, 10
- Gray-koden är **bra för räknare**

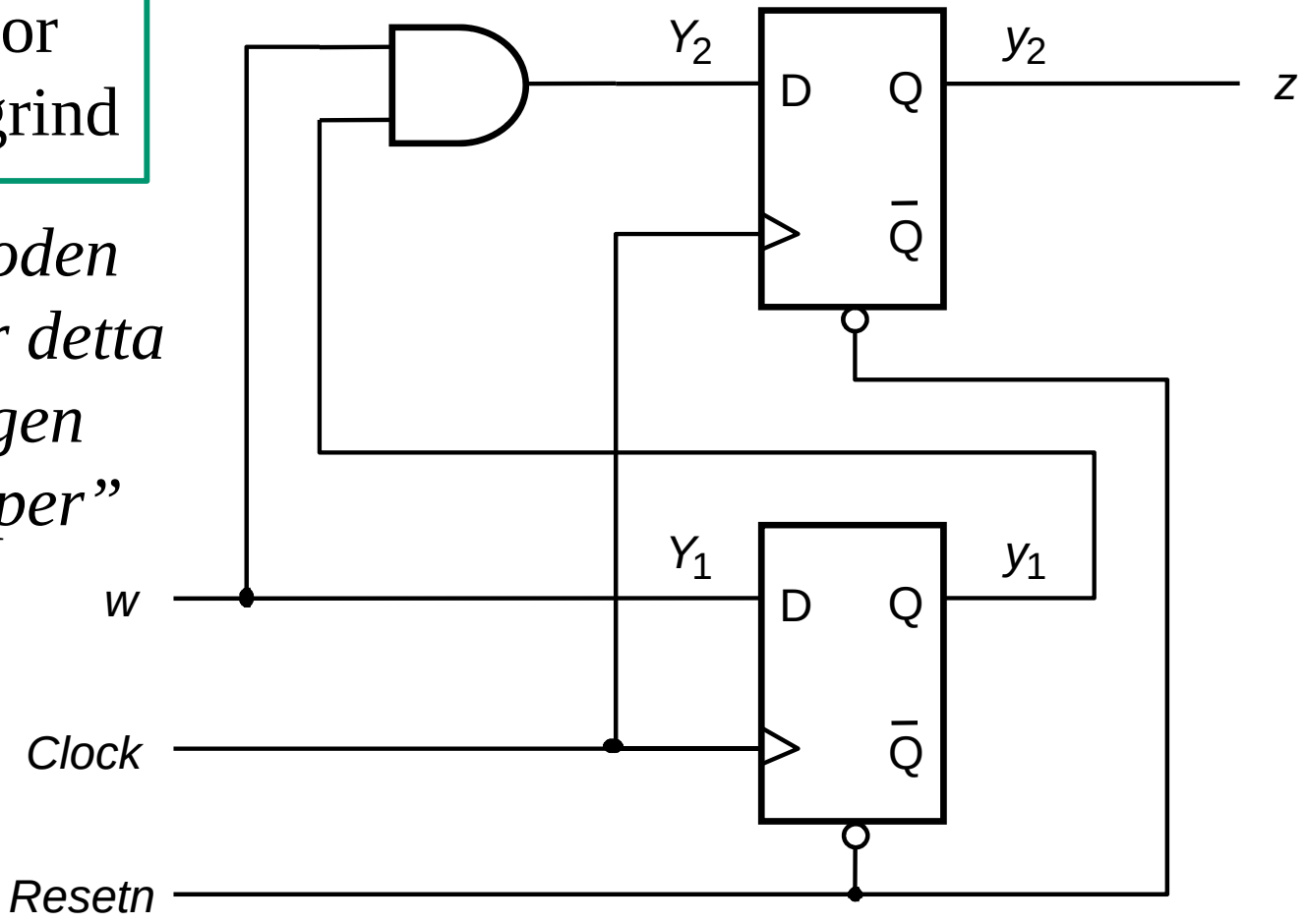
A = 00
B = 01
C = 11
10

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*

Realisering (Graykod)

2 D-vippor
1 AND-grind

Eftersom Graykoden lyckades bra har detta sekvensnät tydligen ”räknaregenskaper”



One-Hot-kodning

- One-hot-kodningen använder en vippa per tillstånd
- För varje tillstånd är en bit ‘hot’ (**1**), alla andra bitar är 0,
dvs 000**1**, 00**1**0, 0**1**00, **1**000
- One-hot kodningen minimerar den kombinatoriska logiken men *ökar* antalet vippor

Vilken kod ska man välja?

- Det finns inte en kod som är den bästa i alla lägen, utan det beror helt på tillståndsdiagrammet
- Man kan även ha ‘egna koder’ som passar till konstruktionen, t ex 00, 11, 10, 01

FSM-Demo

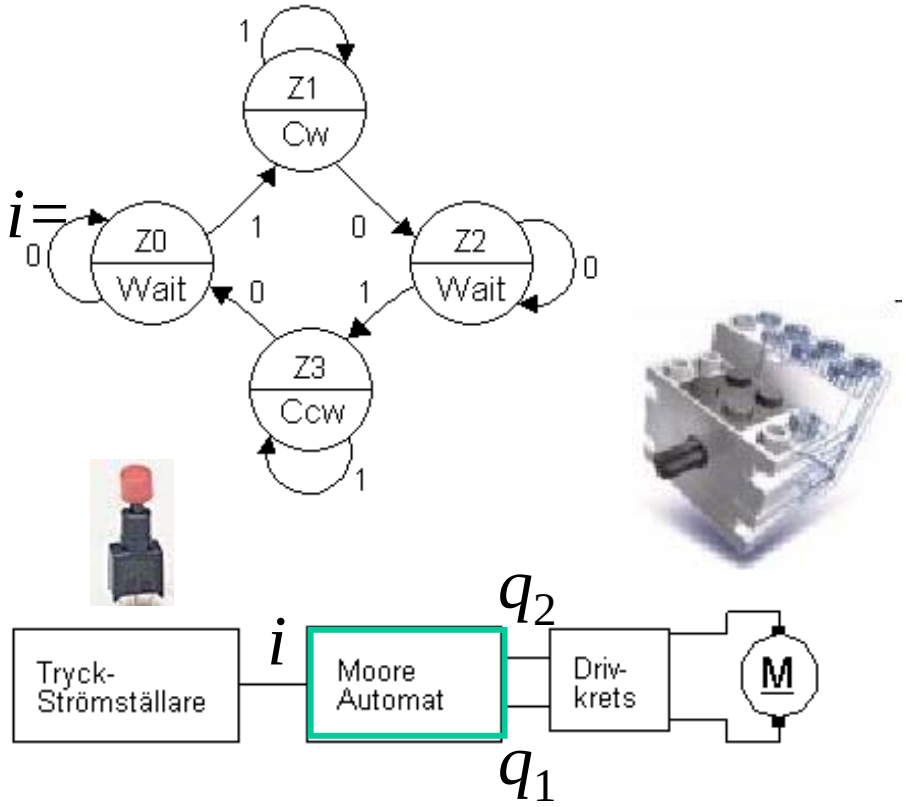
Finite **S**tate **M**achine (FSM)

Kör en motor fram/back varannan gång

- Automaten med **logikkretsar**
- Programmerbar logik och **VHDL**

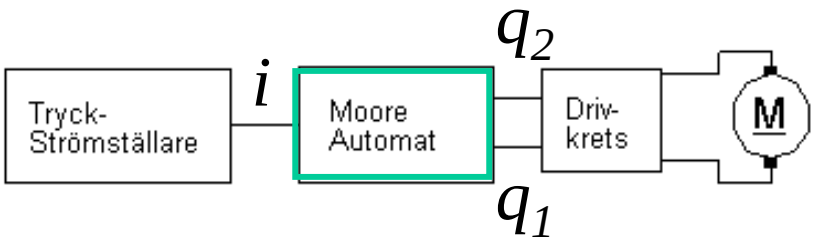
Demo. - wait - cw - wait - ccw -

$Z0$	q_2	0	**wait**
	q_1	0	
$Z1$	q_2	0	**cw**
	q_1	1	
$Z2$	q_2	1	**wait**
	q_1	1	
$Z3$	q_2	1	**ccw**
	q_1	0	



Med denna tillståndskodning (Graykod) så behövs det ingen utgångsavkodare!

Demo. - wait - cw - wait - ccw -



$q_2^+ q_1^+ = f(i, q_2, q_1)$

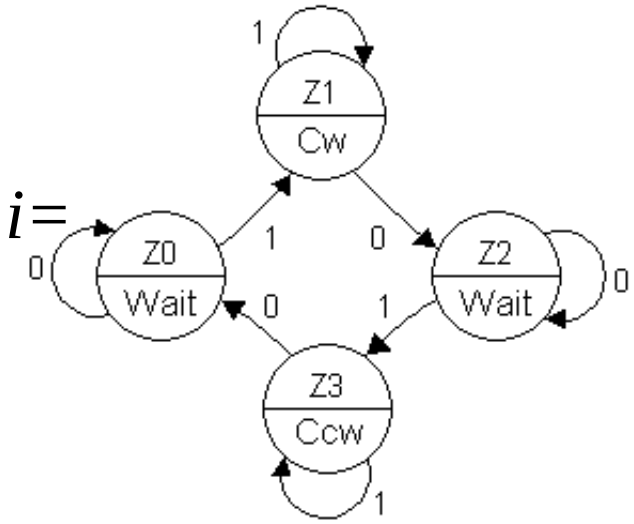
$q_2 q_1$	i = 0	i = 1
Z0 00		
Z1 01		
Z2 11		
Z3 10		

$q_2^+ = f(i, q_2, q_1)$

q_2 q_1	i = 0	i = 1
0 0		
0 1		
1 1		
1 0		

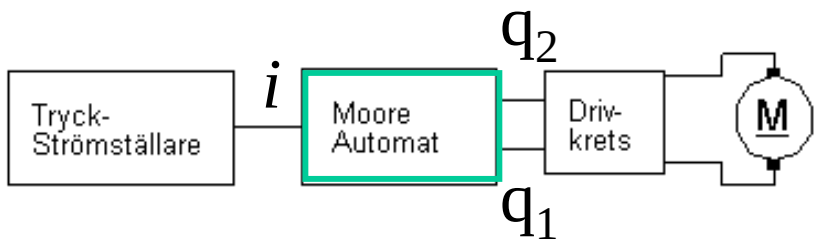
$q_1^+ = f(i, q_2, q_1)$

q_2 q_1	i = 0	i = 1
0 0		
0 1		
1 1		
1 0		



Kan Du ställa upp den kodade tillståndstabellen och Karnaughdiagrammen själv … ?

Demo. - wait - cw - wait - ccw -



$q_2^+ q_1^+ = f(i, q_2, q_1)$

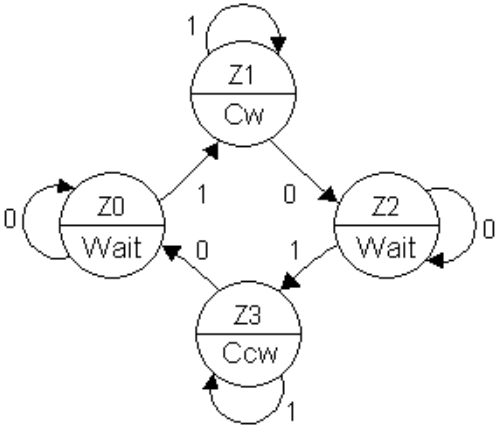
$q_2 q_1$	i=0	i=1
Z0 00	0 0	0 1
Z1 01	1 1	0 1
Z2 11	1 1	1 0
Z3 10	0 0	1 0

$q_2^+ = f(i, q_2, q_1)$

$q_2 q_1$	i=0	i=1
00	0	0
01	1	0
11	1	1
10	0	1

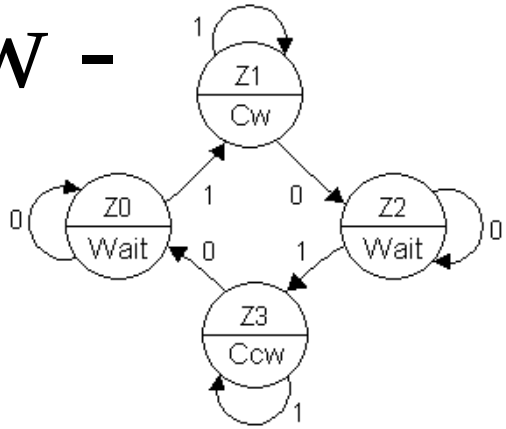
$q_1^+ = f(i, q_2, q_1)$

$q_2 q_1$	i=0	i=1
00	0	1
01	1	1
11	1	0
10	0	0



$$q_2^+ = \bar{i}\,q_1 + i\,q_2 \qquad q_1^+ = \bar{i}\,q_1 + i\,\bar{q}_2$$

- wait - cw - wait - ccw -



$q_2^+ q_1^+ = f(i, q_2, q_1)$

$q_2 q_1$ \ i	0	1
Z0 00	0 | 0	0 | 1
Z1 01	1 | 1	0 | 1
Z2 11	1 | 1	1 | 0
Z3 10	0 | 0	1 | 0

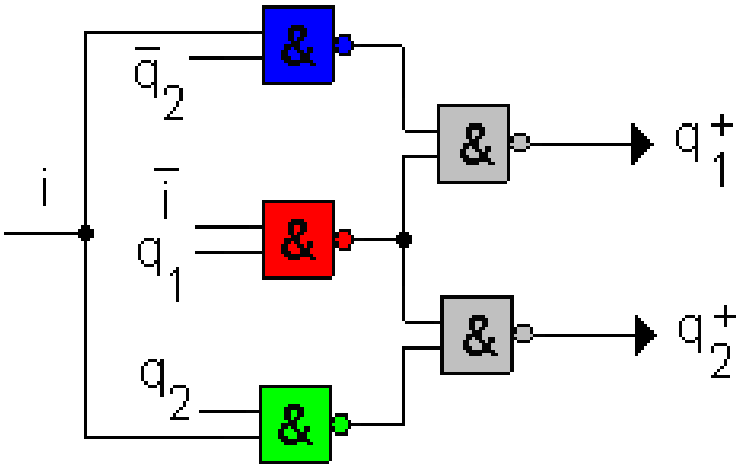
$q_2^+ = f(i, q_2, q_1)$

q_2 q_1 \ i	0	1
0 0	0	0
0 1	1	0
1 1	1	1
1 0	0	1

$q_1^+ = f(i, q_2, q_1)$

q_2 q_1 \ i	0	1
0 0	0	1
0 1	1	1
1 1	1	0
1 0	0	0

$$q_2^+ = \bar{i}\,q_1 + i\,q_2 \qquad q_1^+ = \bar{i}\,q_1 + i\,\bar{q}_2$$

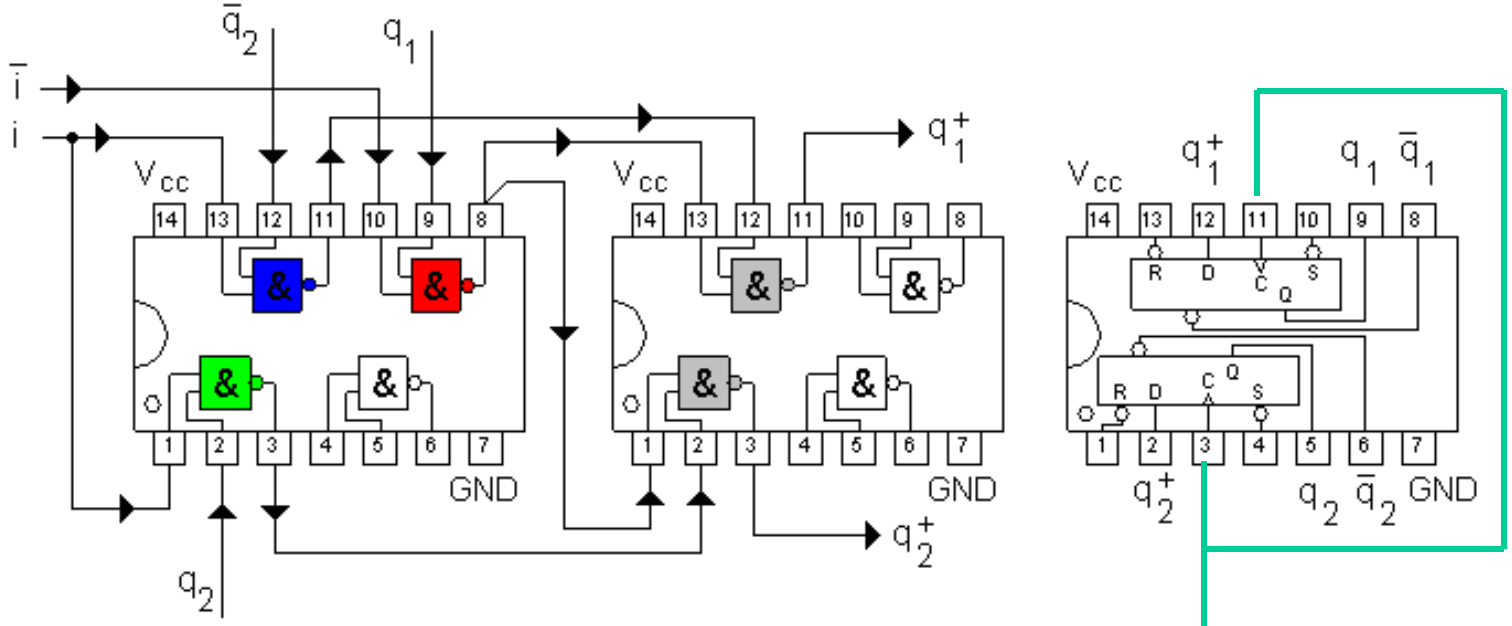


Med NAND-grindar

- wait - cw - wait - ccw -

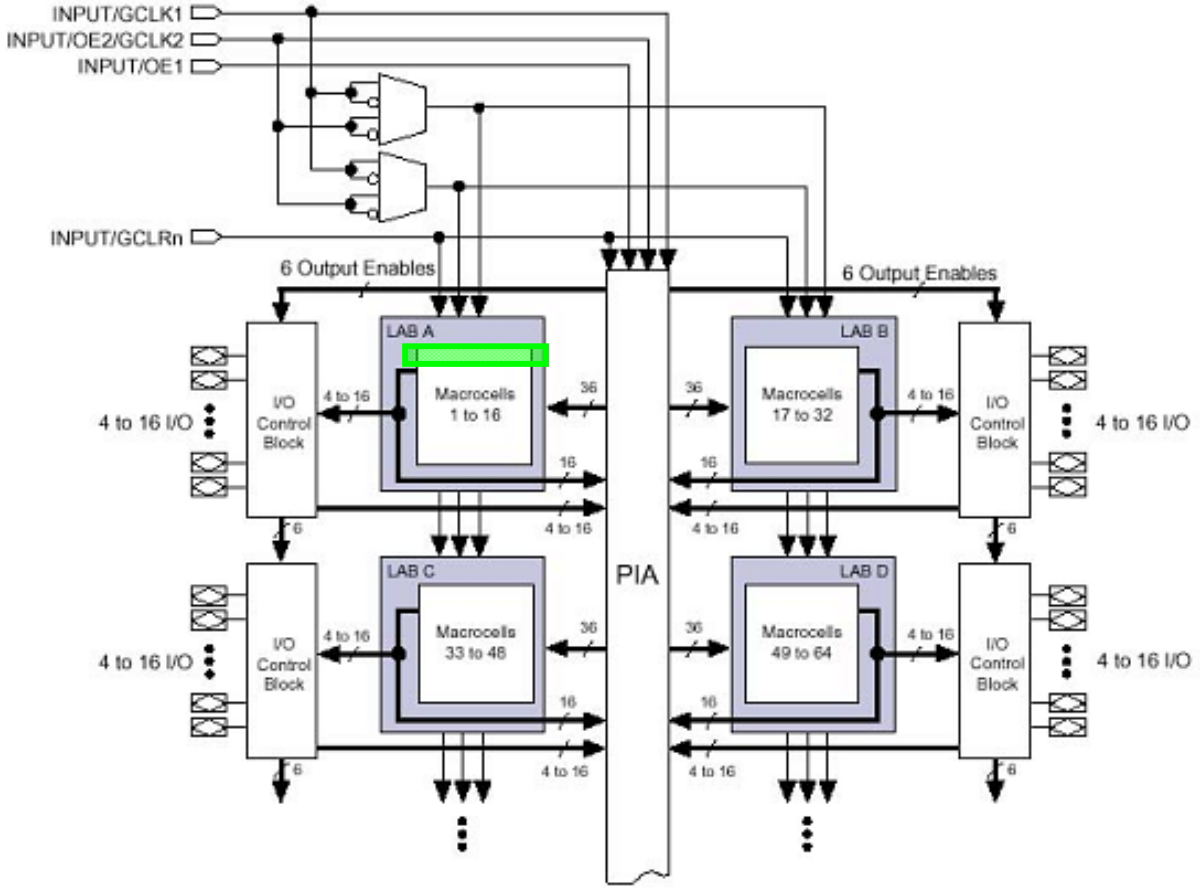
Med 74-seriens standardkretsar

7400 7400 7474



Det behövs också klockpulser. Tumregel: tre gånger högre frekvens än insignalernas.
*(här ger **10 Hz** tillräckligt snabb respons samtidigt som ”kontaktstudsar” undertrycks)*

CPLD (MAX) ?

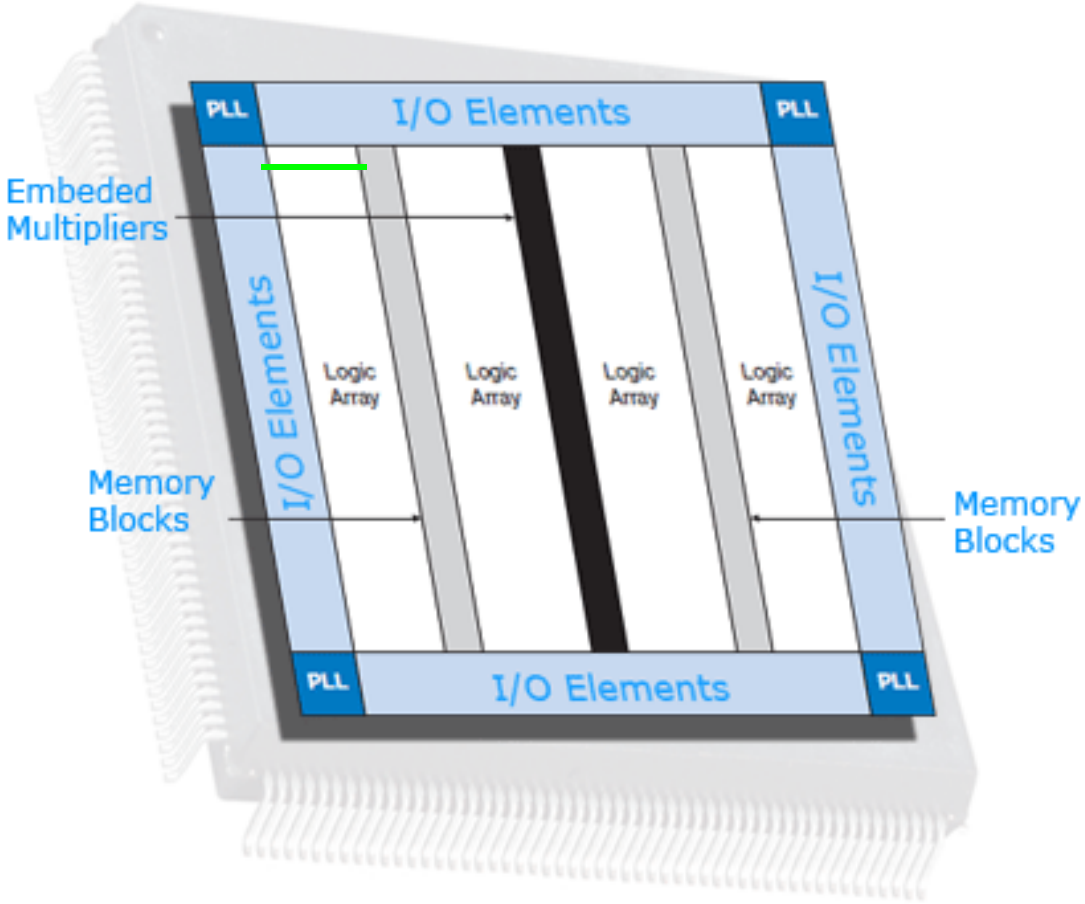


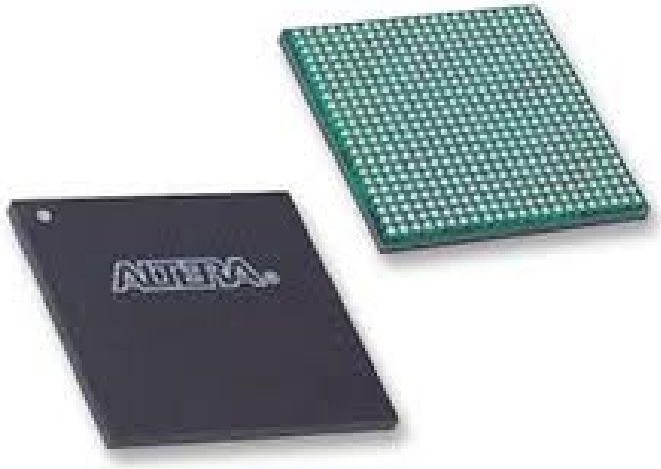
Typiskt **64** Macroceller

Teknik: AND-OR array

(Större MAX har MUX-tree)

FPGA (Cyclone II) ?



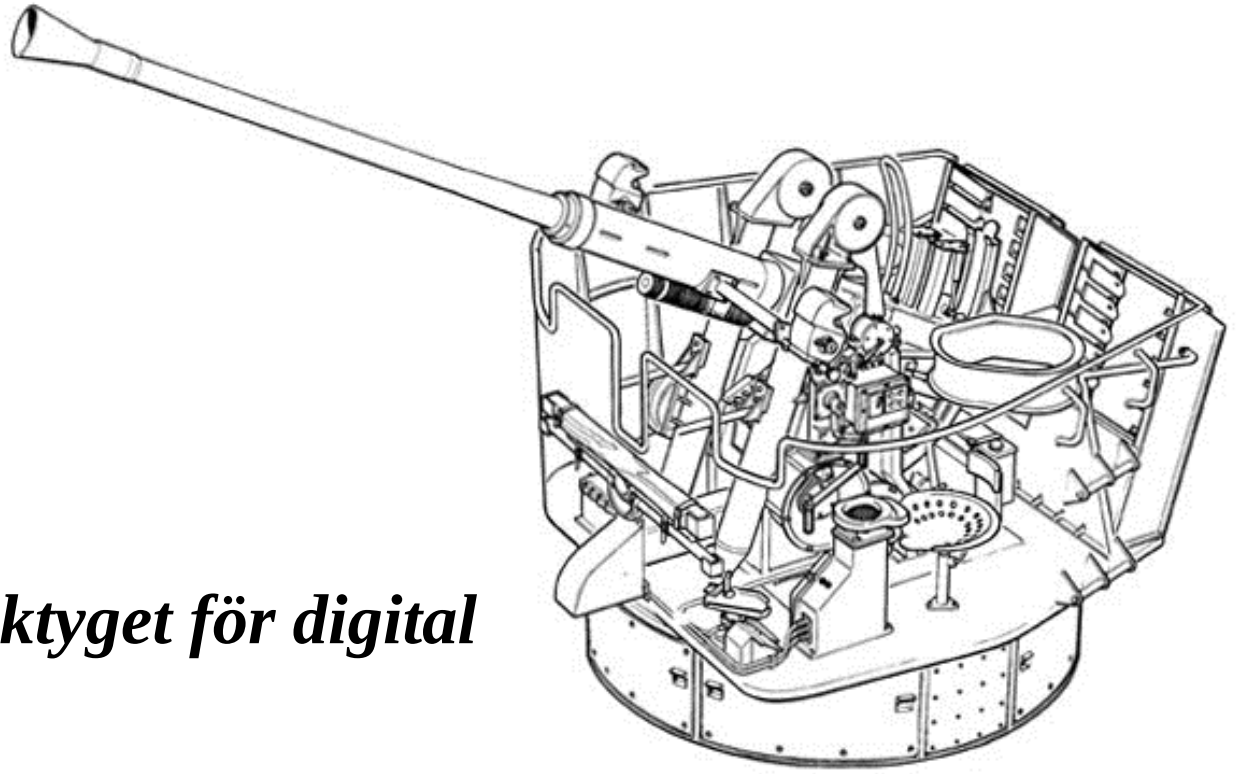


Typiskt **50000** logikelement

Teknik: MUX tree

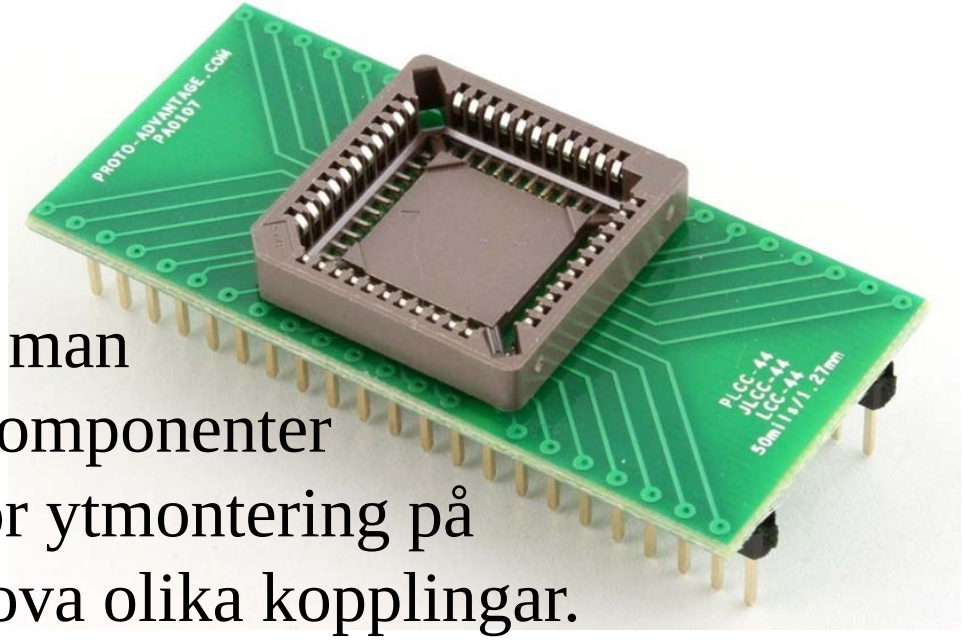
Rätt verktyg …

*Man skall väl **inte** skjuta **mygg** med luftvärnskanon – när det finns myggsprej?*



Vilket är det rätta verktyget för digital designkursen …

CPLD (MAX)



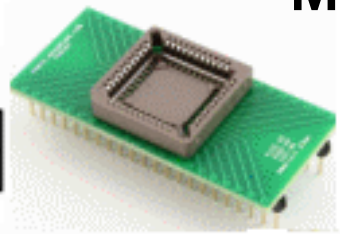
Med ett **breakoutboard** kan man använda kopplingsdäck till komponenter som egentligen är avsedda för ytmontering på kretskort. Man kan enkelt prova olika kopplingar.

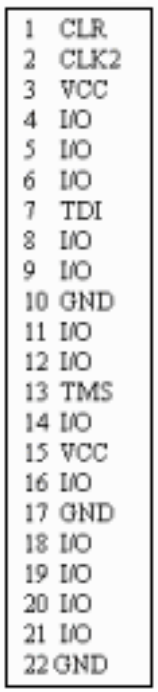
På så sätt använder vi samma teknik som i övriga laborationer – trots att vi nu går över till mer komplexa så kallade **CPLD-**kretsar och programmerar dem med VHDL-språket.

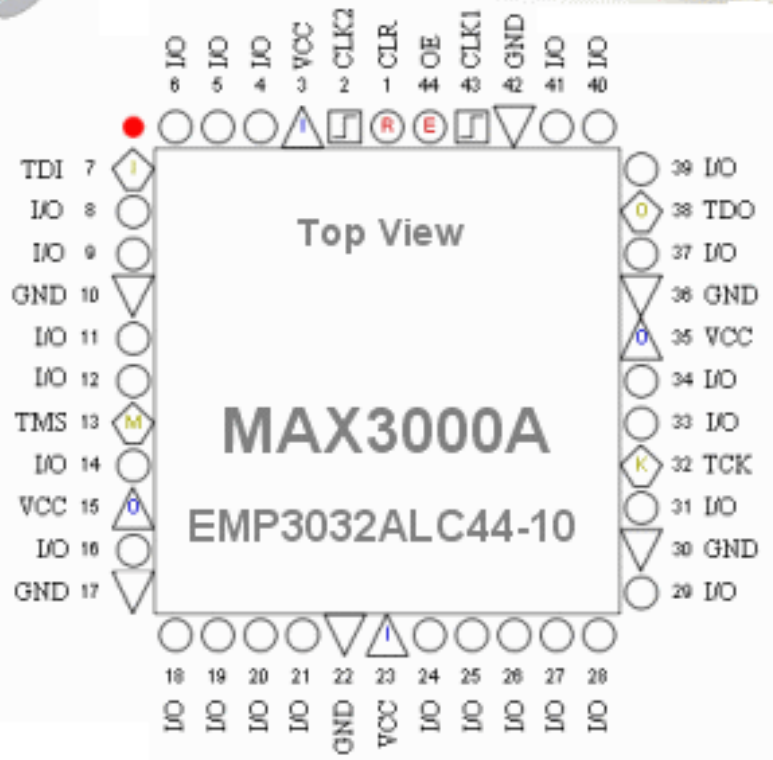
Programmerbar logik med VHDL

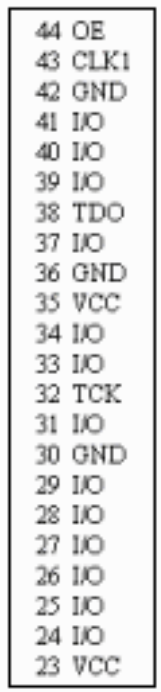
MAX3000

JTAG				
TDI	-	TMS	TDO	TCK
GND	-	-	VCC	GND







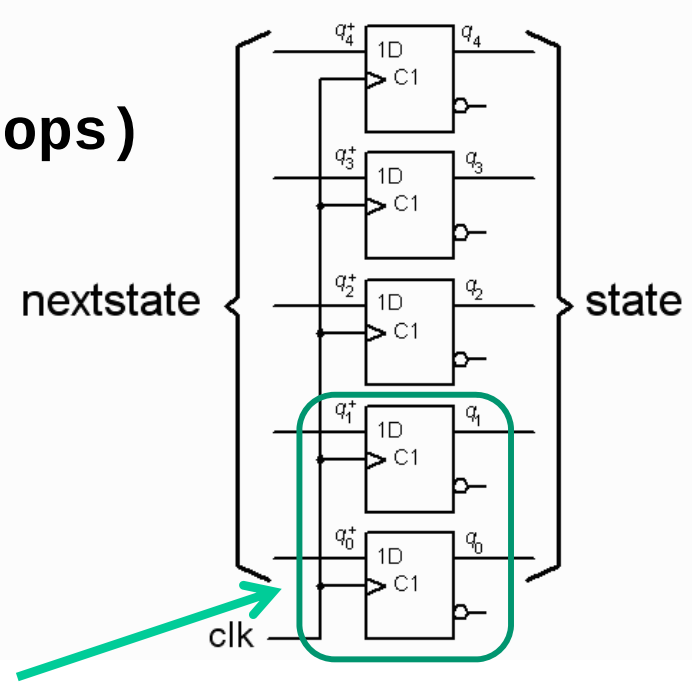


QuartusII

VHDL-koden

Tillståndsregister med D-vippor:

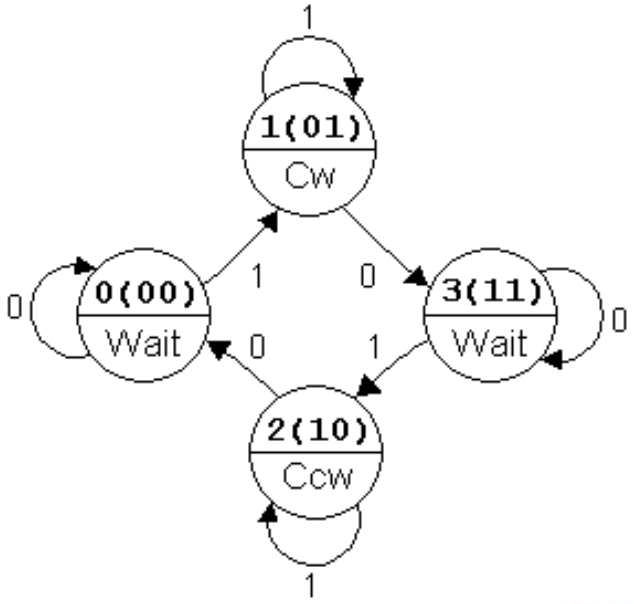
```
state_register: -- (the flipflops)
process(clk)
begin
  if rising_edge(clk) then
     state <= nextstate;
  end if;
end process;
```



Vi behöver två vippor.

VHDL-koden

```
next_state_decode: -- next state decoding part
process(present_state, I, R)
 begin
   if I = '1' then
     case present_state is
       when 0 => next_state <= 1;
       when 1 => next_state <= 1;
       when 3 => next_state <= 2;
       when 2 => next_state <= 2;
     end case;
   else  -- I = '0'
     case present_state is
       when 0 => next_state <= 0;
       when 1 => next_state <= 3;
       when 3 => next_state <= Ooops;
       when 2 => next_state <= Ooops;
     end case;
   end if ;
end process;
```

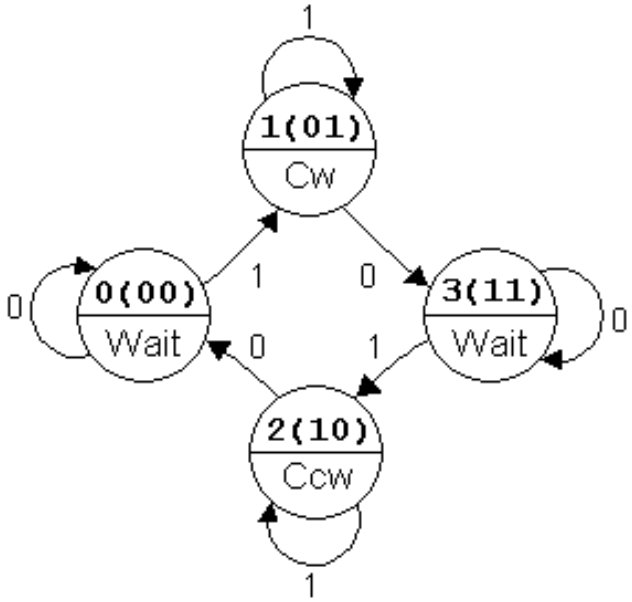


?

Vad ska stå i stället för Ooops!

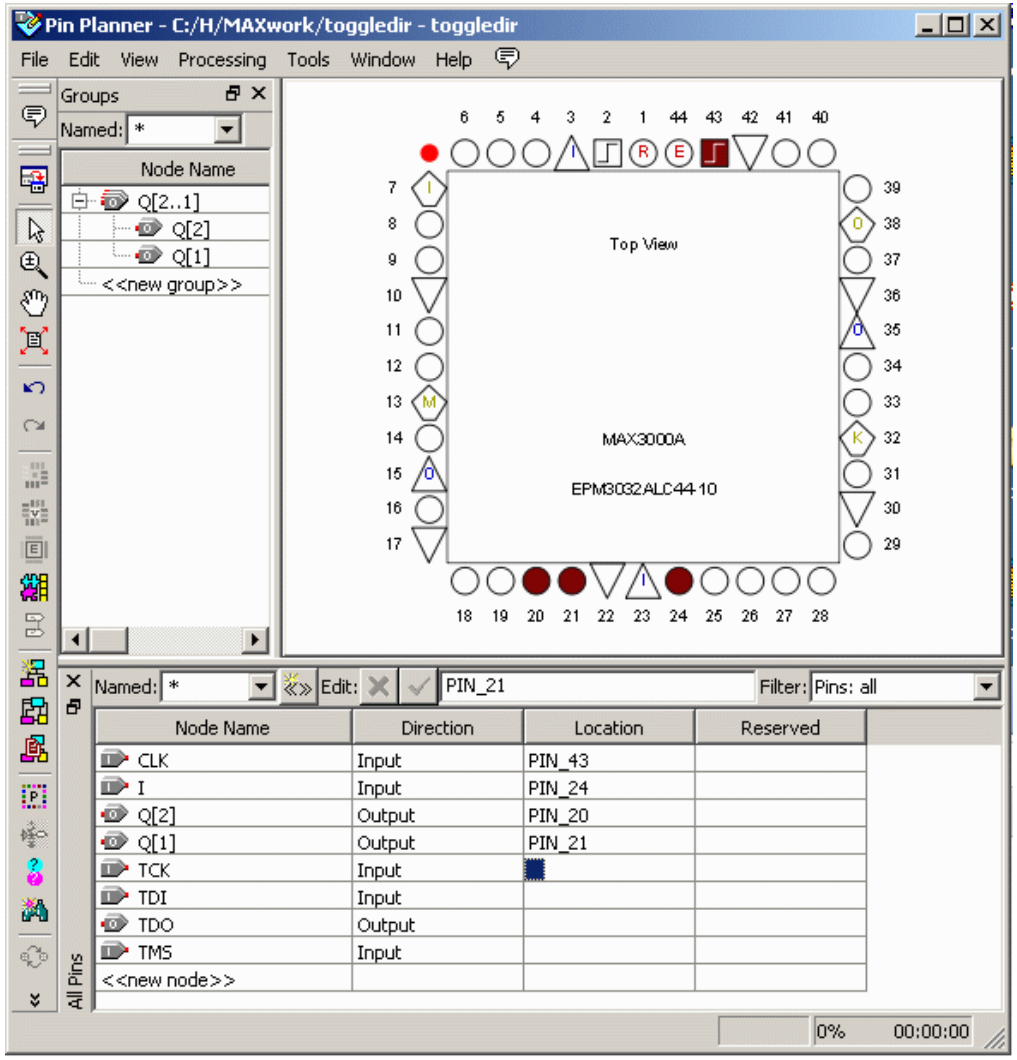
VHDL-koden

```
next_state_decode: -- next state decoding part
process(present_state, I, R)
 begin
   if I = '1' then
     case present_state is
       when 0 => next_state <= 1;
       when 1 => next_state <= 1;
       when 3 => next_state <= 2;
       when 2 => next_state <= 2;
     end case;
   else  -- I = '0'
     case present_state is
       when 0 => next_state <= 0;
       when 1 => next_state <= 3;
       when 3 => next_state <= 3;
       when 2 => next_state <= 0;
     end case;
   end if ;
end process;
```

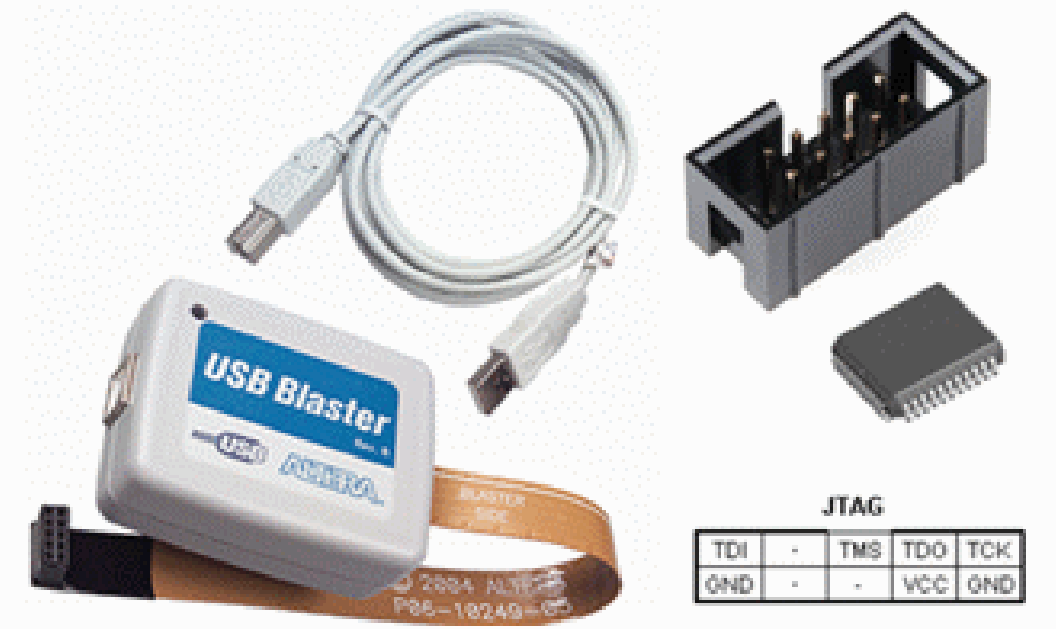


!

Pin-planering

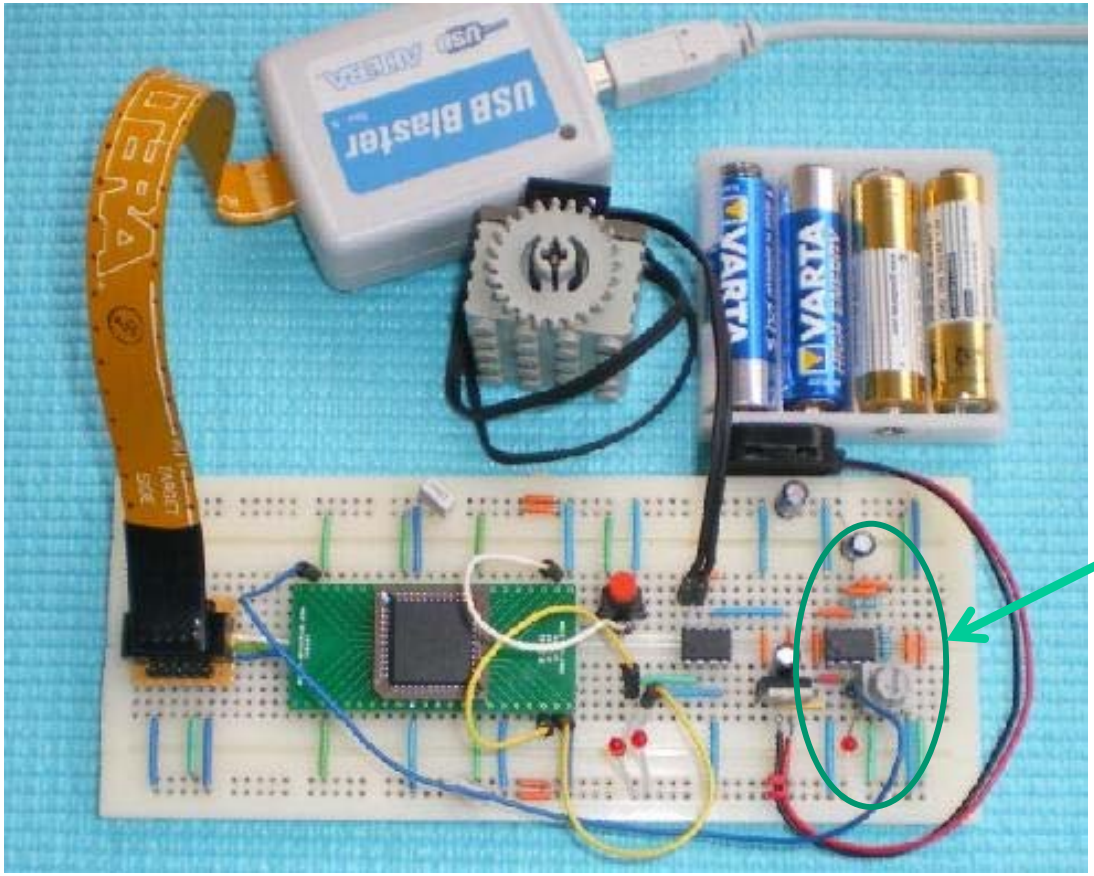


Chip-programmering



Har Du sett den här JTAG-kontakten på något datorkretskort?

Prova resultatet!



Klockpulser

10 Hz
För denna
tillämpning

MAX-krets fungerar …