

Digital Design IE1204

Kursomgång för Högskoleingenjörsinriktningarna:

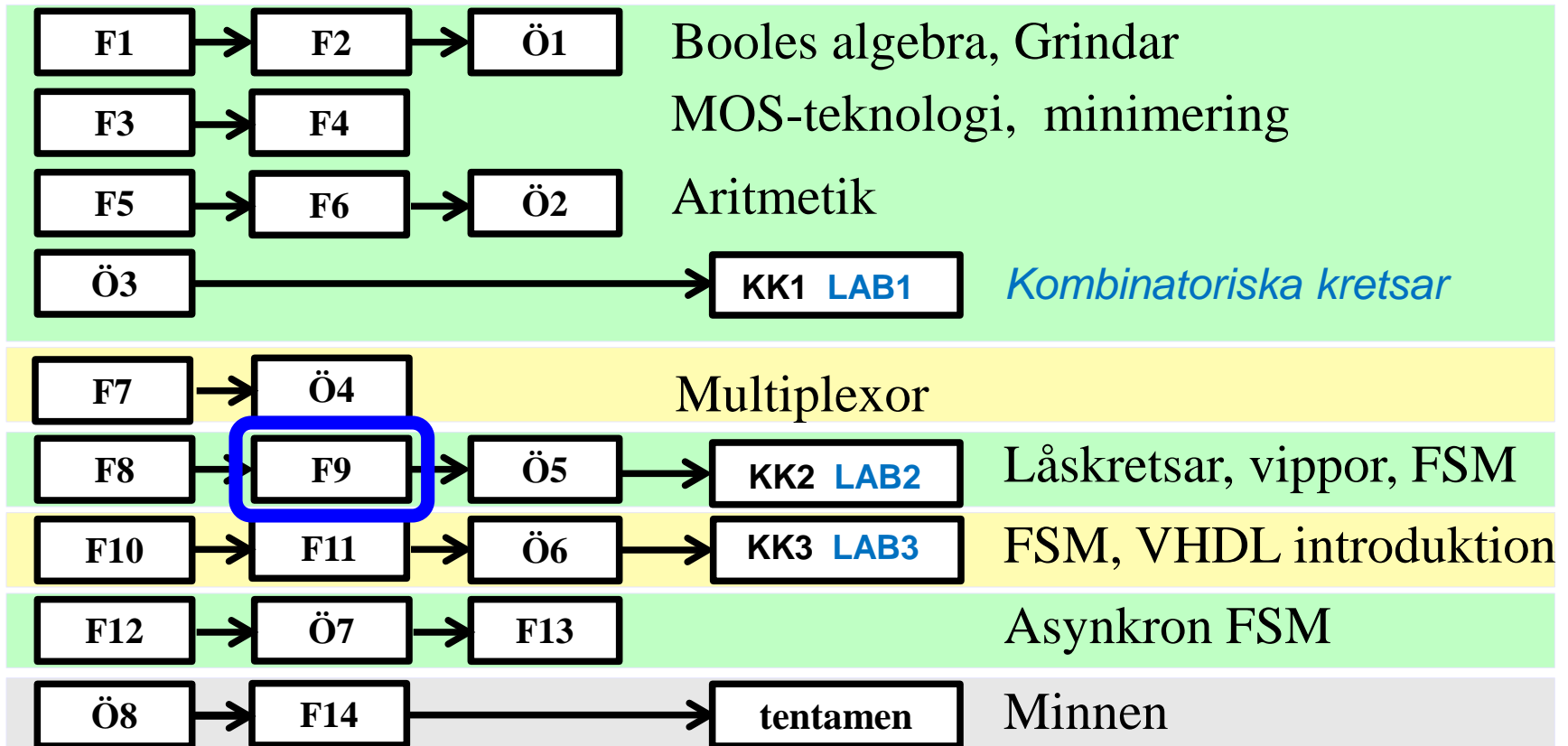
Datateknik, Elektronik och Datorteknik.

Kandidatinriktningen: **IT-kandidat.**

F9 Tillståndsautomater del 1

william@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

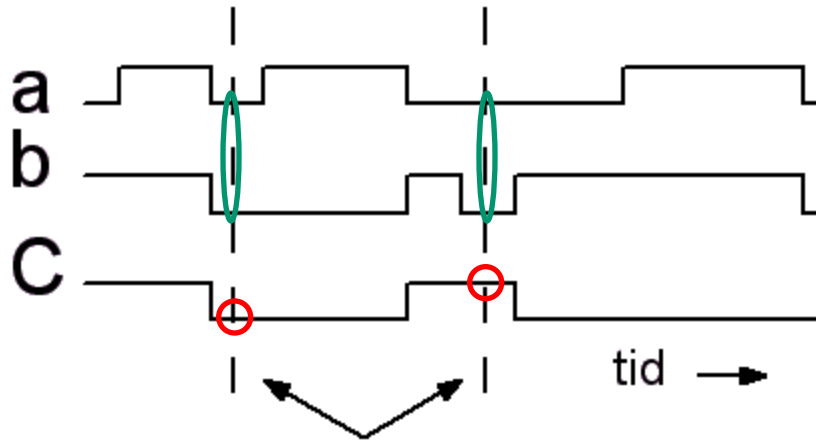
Prioritetsenkoder Kodomvandlare

VHDL introduktion

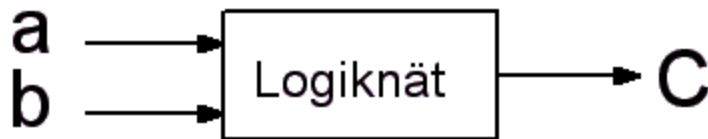
Vippor och Låskretsar SR-latch D-latch D-vippa JK-vippa T-vippa Räknare

Skiftregister Vippor i VHDL

Sekvensnät



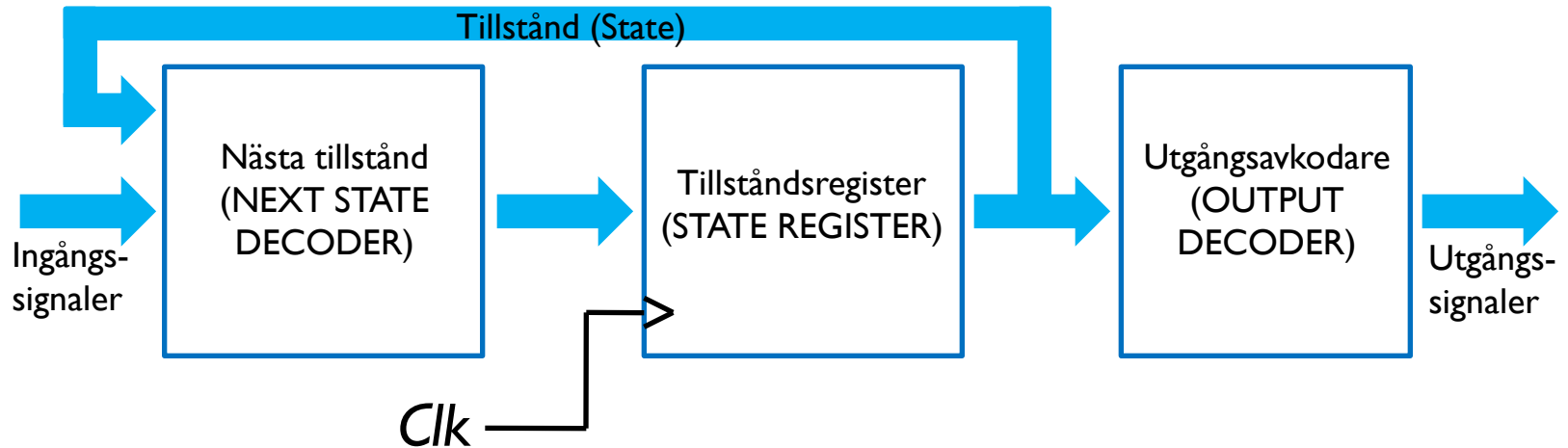
Samma insignal
kan ge olika utsignal



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!

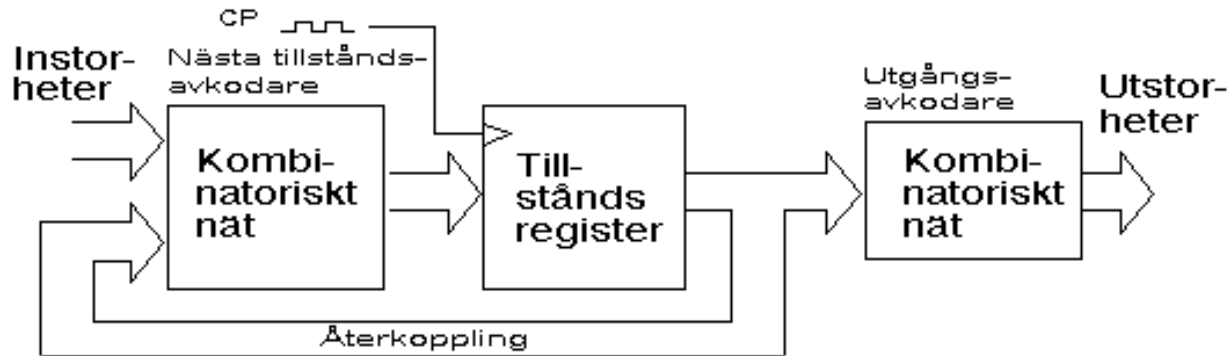
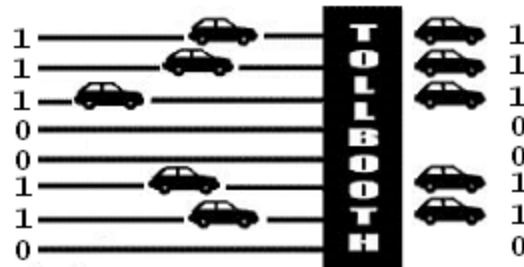
Moore-automat



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

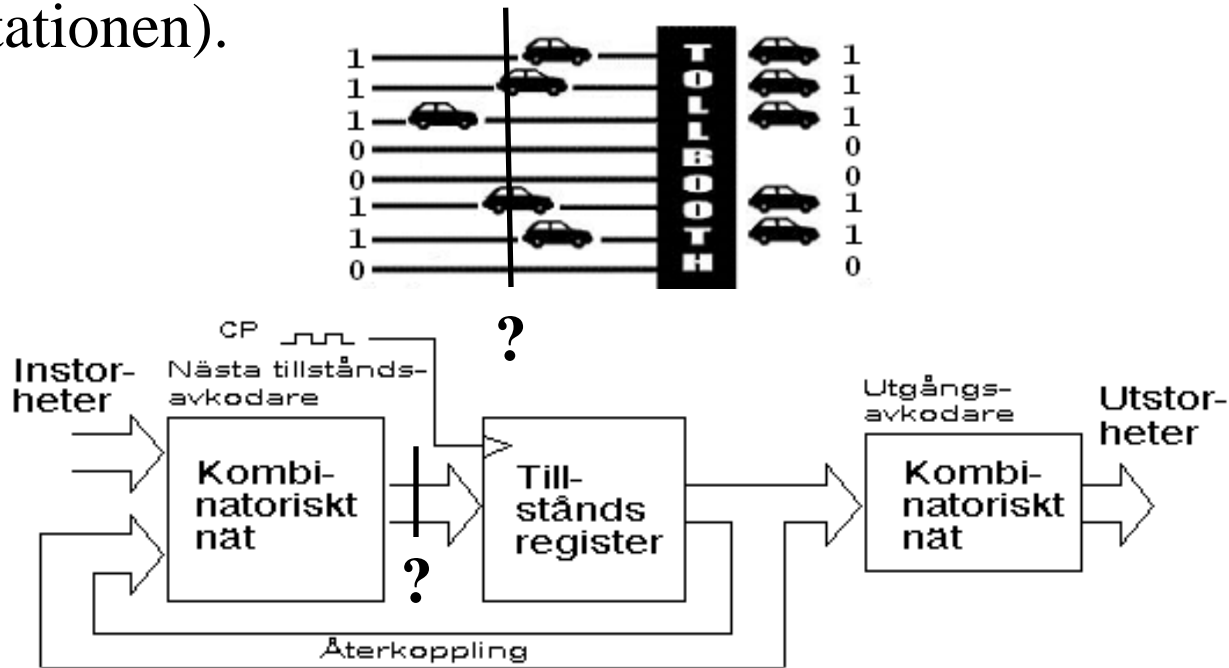
Tillståndsregistrets D-vippor

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



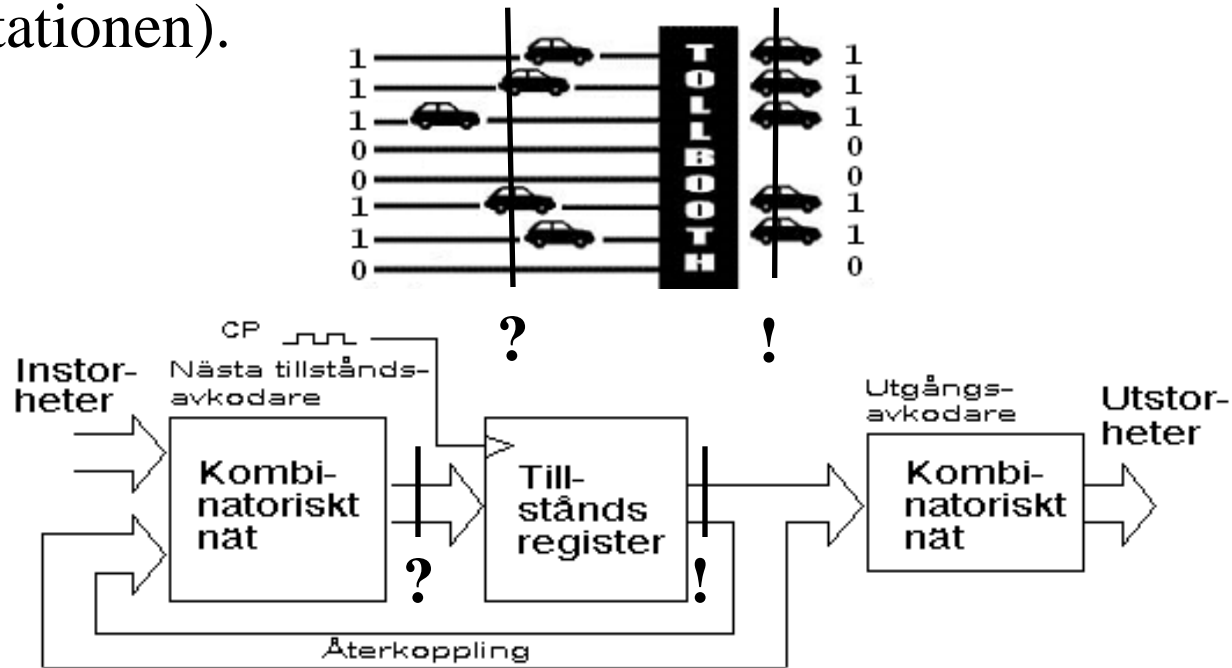
Tillståndsregistrets D-vippor

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



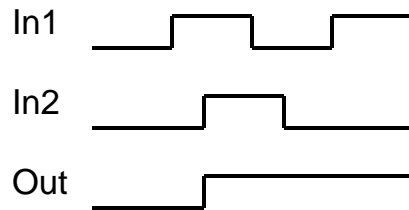
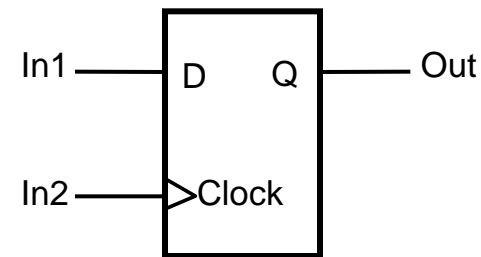
Tillståndsregistrets D-vippor

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

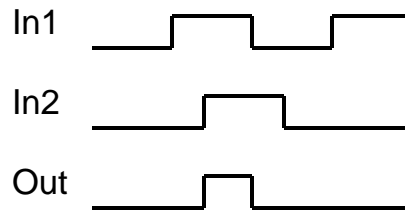


Snabbfråga Vippor

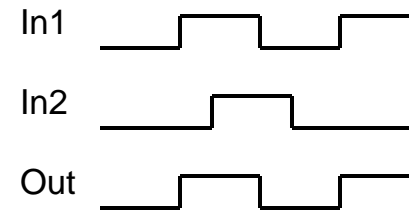
Vilket av följande tidsdiagram är giltigt för en flanktriggad D flip-flop?



Alt: A



Alt: B

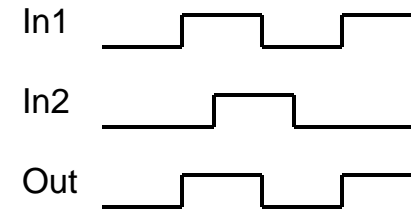
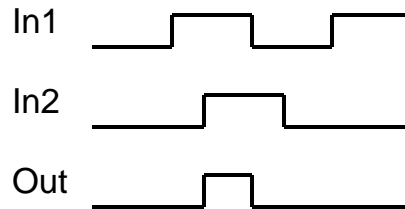
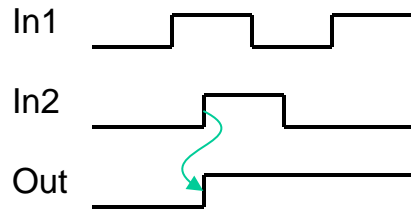
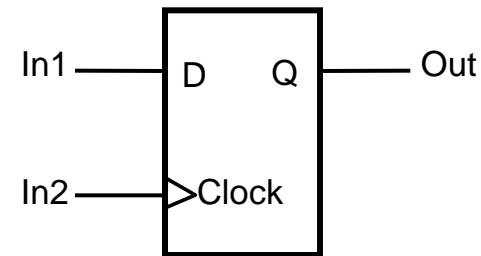


Alt: C



Snabbfråga Vippor

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



Alt: A

Alt: B

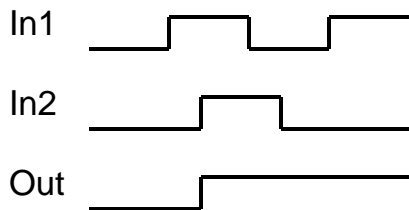
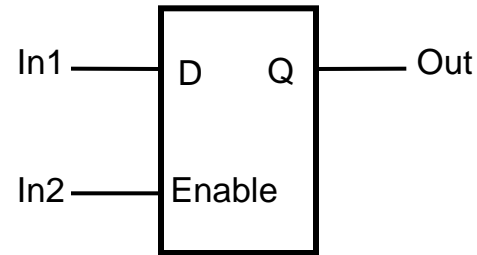
Alt: C



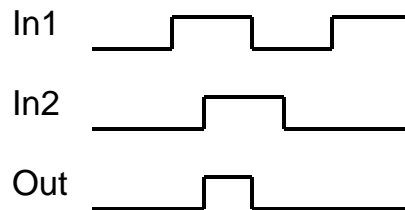
*D kopieras till utgången vid flanken,
dvs när Clock går från 0 till 1*

Snabbfråga Latch

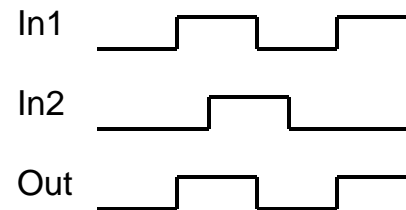
Vilket av följande tidsdiagram är giltigt för en D Latch?



Alt: A



Alt: B

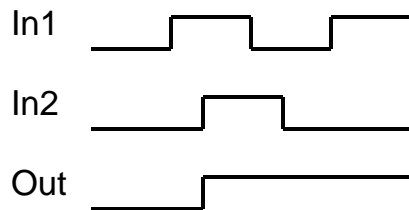
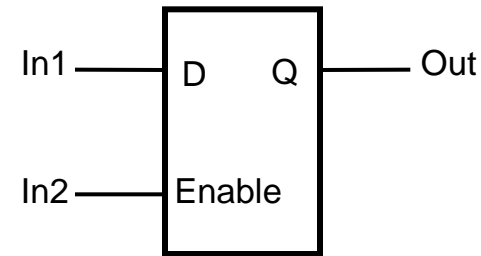


Alt: C

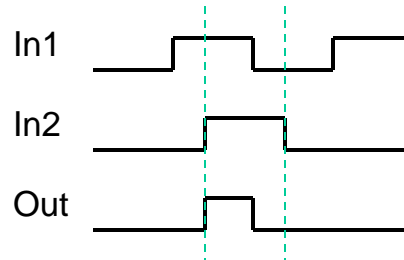


Snabbfråga Latch

Vilket av följande tidsdiagram är giltigt för en D Latch?



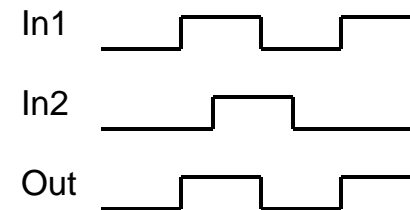
Alt: A



Alt: B

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

William Sandqvist william@kth.se

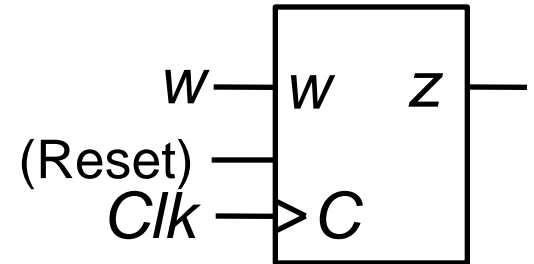


Alt: C



Designexempel ”två i rad”

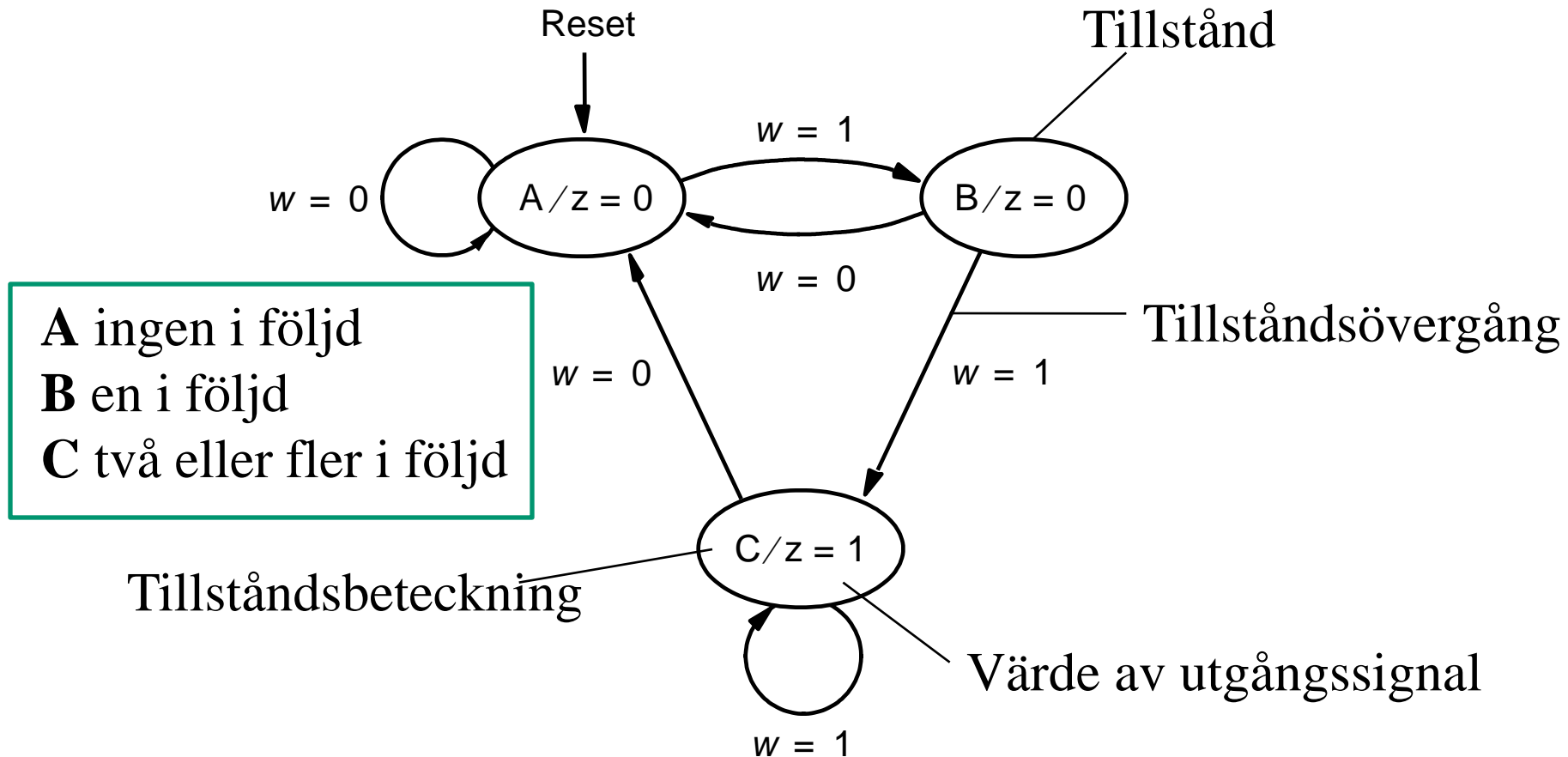
Sekvensdetektor. Om w har varit 1 under två (eller fler) klockcykler i rad skall $z = 1$.



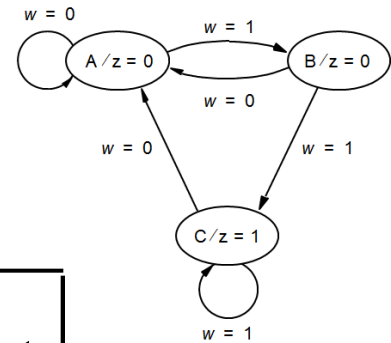
Specifikation

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

Tillståndsdigram ”två i rad”



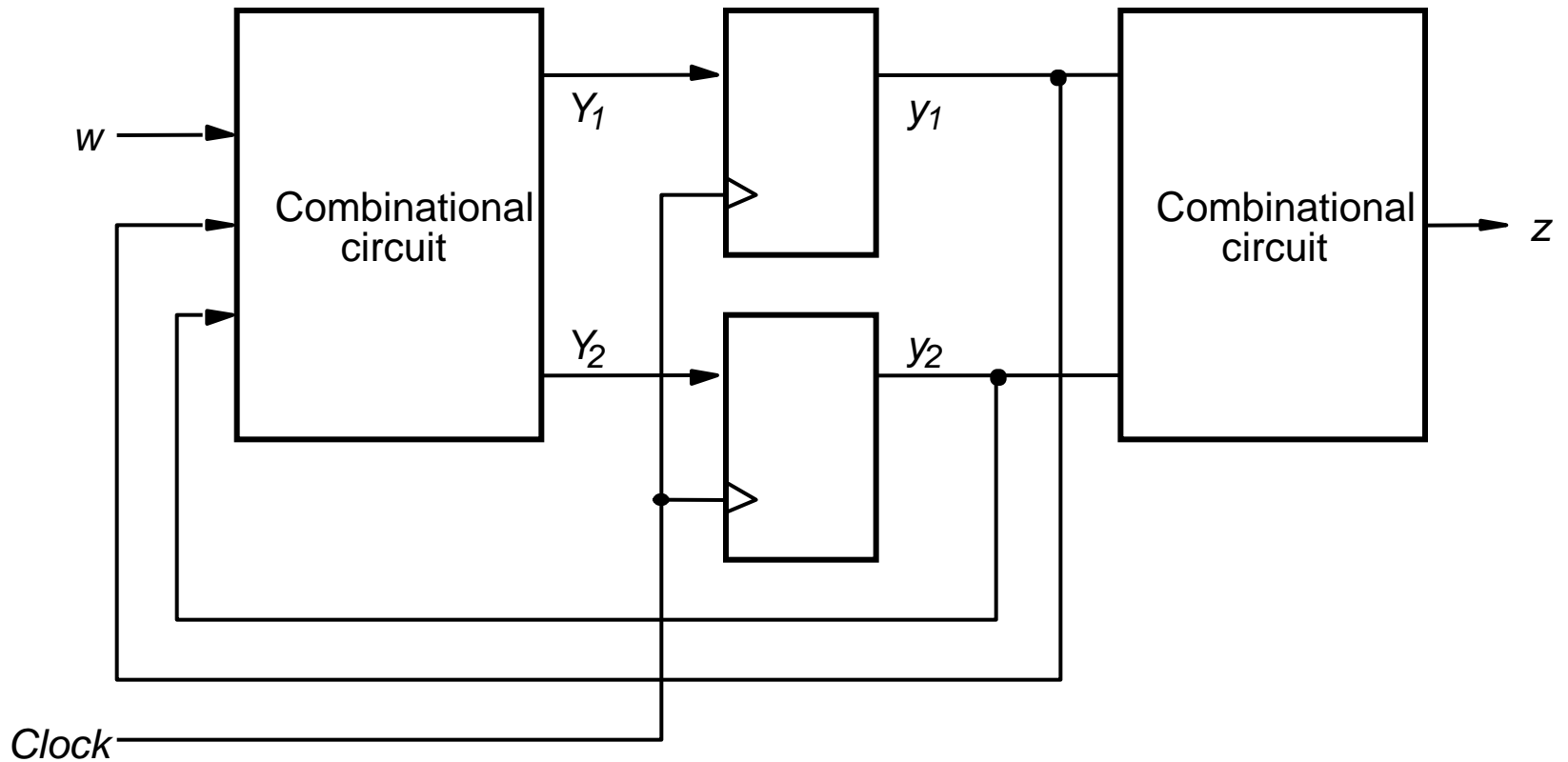
Tillståndstabell



Present state	Next state		Output z
	$w = 0$	$w = 1$	
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.

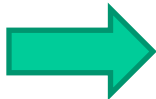
Kodad tillståndstabell

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

A = 00

B = 01

C = 10



A
B
C

Present state $y_2 y_1$	Next state		Output z
	$w = 0$	$w = 1$	
	$Y_2 Y_1$	$Y_2 Y_1$	
00	00	01	0
01	00	10	0
10	00	10	1
11	<i>dd</i>	<i>dd</i>	<i>d</i>

$$Y_2 Y_1 = f(y_2 y_1 w) \quad z = f(y_2 y_1)$$

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.

$$Y_2 Y_1 = f(y_2 y_1 w) \quad Y_2 = f(y_2 y_1 w) \quad Y_1 = f(y_2 y_1 w)$$

$$z = f(y_2 y_1)$$

Från kodad tillståndstabell till Karnaughdiagram

$$Y_2 Y_1 = f(y_2 y_1 w) \quad Y_2 = f(y_2 y_1 w) \quad Y_1 = f(y_2 y_1 w)$$

Present state $y_2 y_1$	Next state		Output z
	$w = 0$	$w = 1$	
	$Y_2 Y_1$	$Y_2 Y_1$	
00	00	01	0
01	00	10	0
10	00	10	1
11	dd	dd	d



		Y_1			
		$w=0, y_2 y_1=00$	$w=0, y_2 y_1=01$	$w=0, y_2 y_1=11$	$w=0, y_2 y_1=10$
w	0	0	0	d	0
	1	1	0	d	0

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

Present state $y_2 y_1$	Next state		Output z
	$w = 0$	$w = 1$	
	$Y_2 Y_1$	$Y_2 Y_1$	
00	00	01	0
01	00	10	0
10	00	10	1
11	dd	dd	d



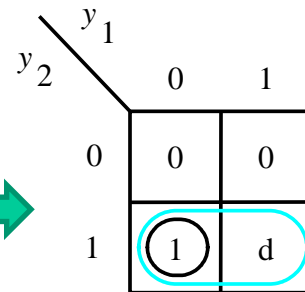
		Y_2			
		$w=0, y_2 y_1=00$	$w=0, y_2 y_1=01$	$w=0, y_2 y_1=11$	$w=0, y_2 y_1=10$
w	0	0	0	d	0
	1	0	1	d	1

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

Utgångsavkodaren

$$z = f(y_2 y_1)$$

Present state $y_2 y_1$	Next state		Output z
	$w = 0$	$w = 1$	
	$Y_2 Y_1$	$Y_2 Y_1$	
00	00	01	0
01	00	10	0
10	00	10	1
11	<i>dd</i>	<i>dd</i>	<i>d</i>



$$z = y_2$$

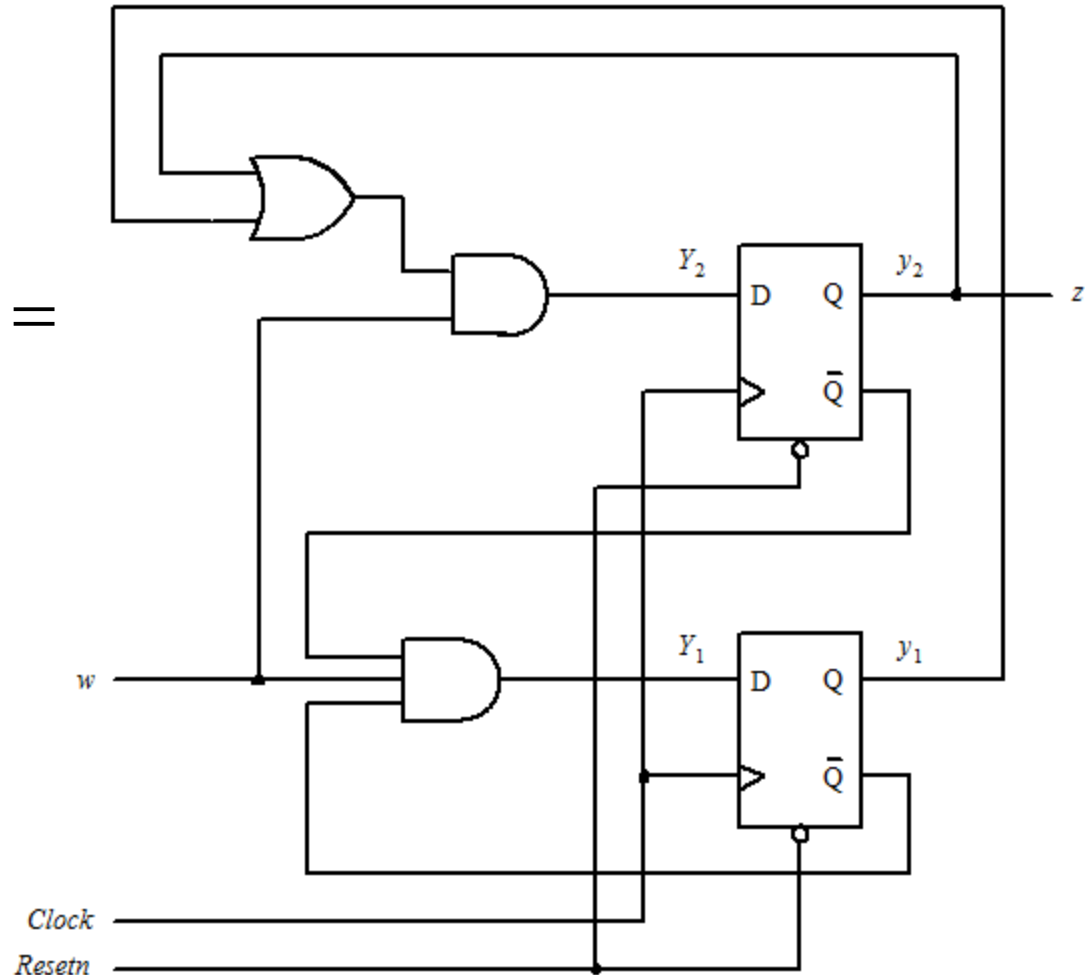
Implementeringen

$$Y_2 = wy_1 + wy_2 =$$

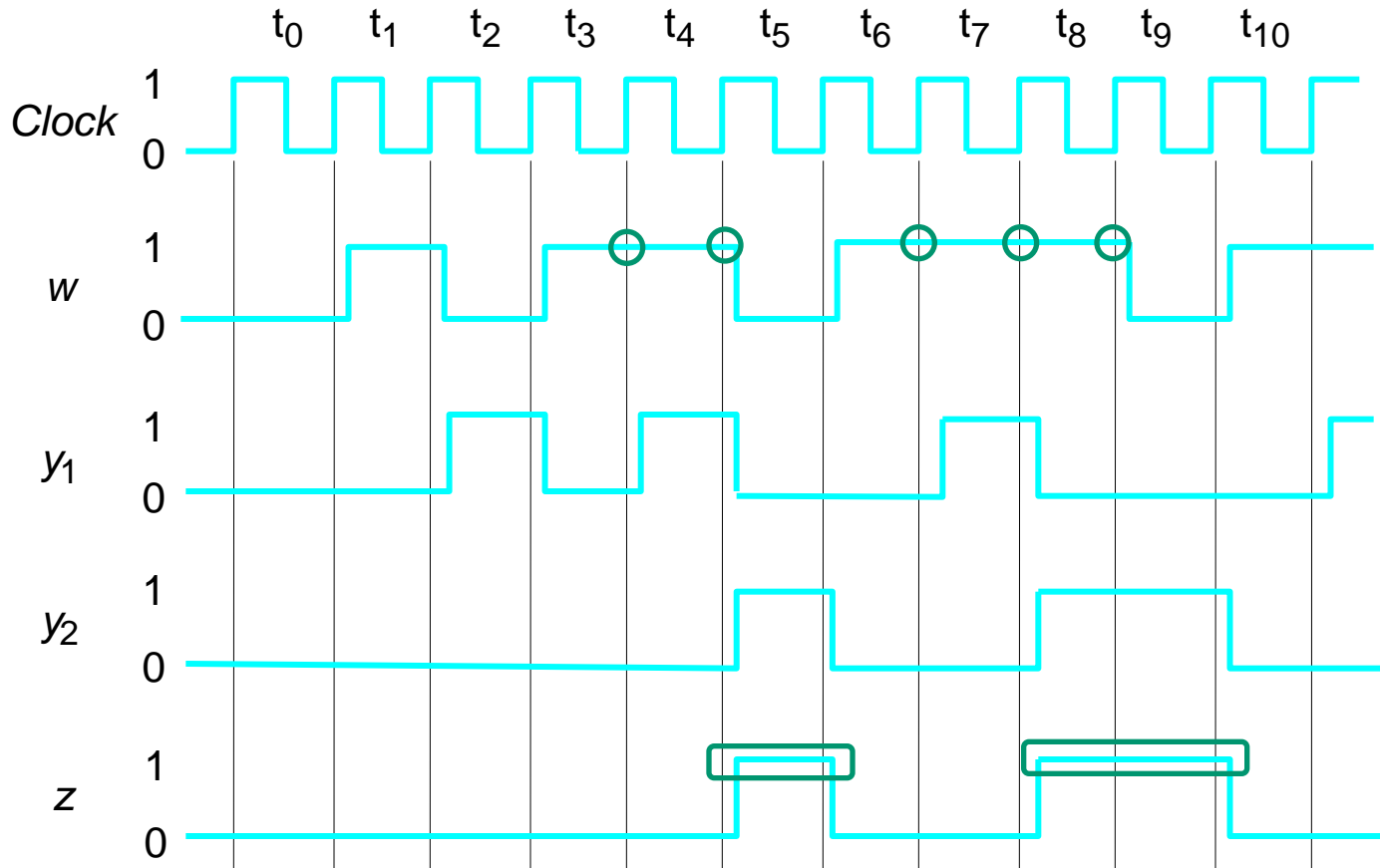
$$= w(y_1 + y_2)$$

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

$$z = y_2$$



Tidsdiagram "två i rad"



Tillståndsöver-
gångar sker
bara på den
positiva
klockflanken!

Med andra beteckningar

Nuvarande tillstånd

y

Nästa tillstånd

y^+

Övningshäfte och Hemert:

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

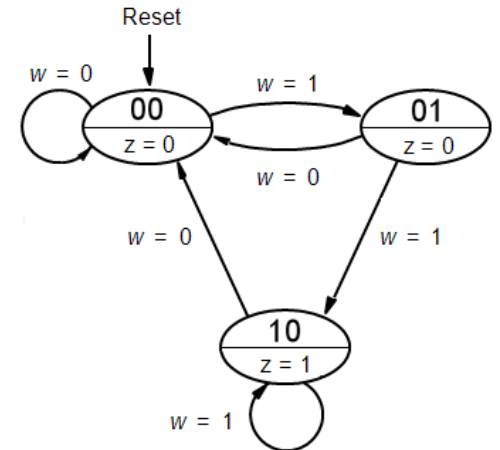
Med annan uppställning

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

		$y_2^+ y_1^+$	
		0	1
A	00	00	01
	01	00	10
11		<i>dd</i>	<i>dd</i>
C	10	00	10

		y_2^+	
		0	1
A	00	0	0
	01	0	1
11		<i>d</i>	<i>d</i>
C	10	0	1

		y_1^+	
		0	1
A	00	0	1
	01	0	0
11		<i>d</i>	<i>d</i>
C	10	0	0

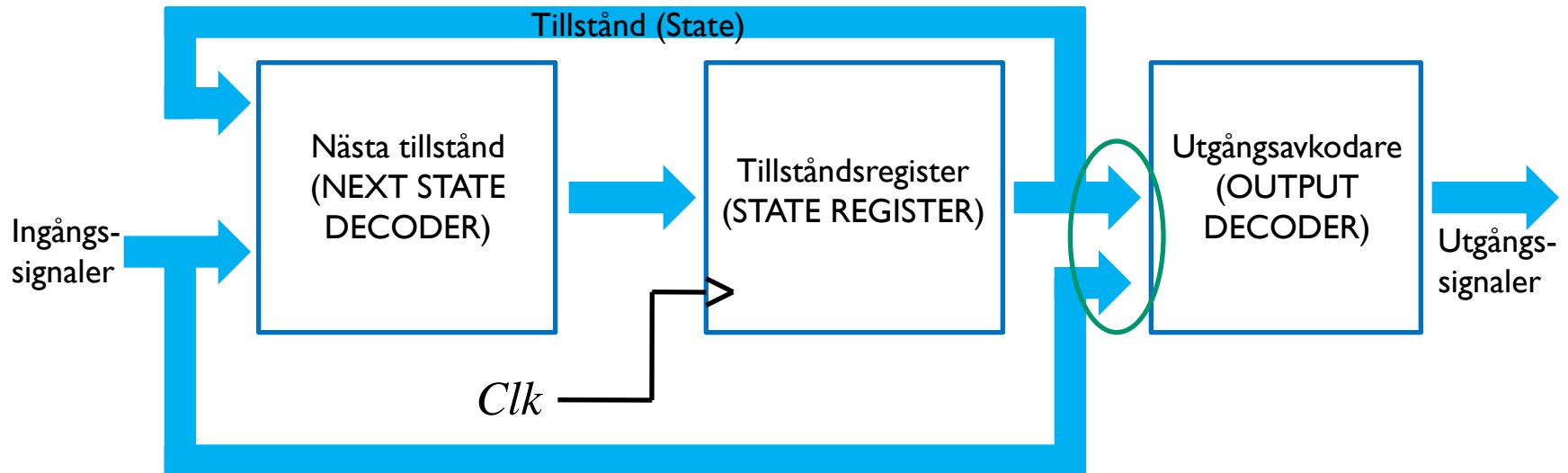


*Övningshäftet
använder
denna metod*

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

William Sandqvist william@kth.se

Mealy-automat

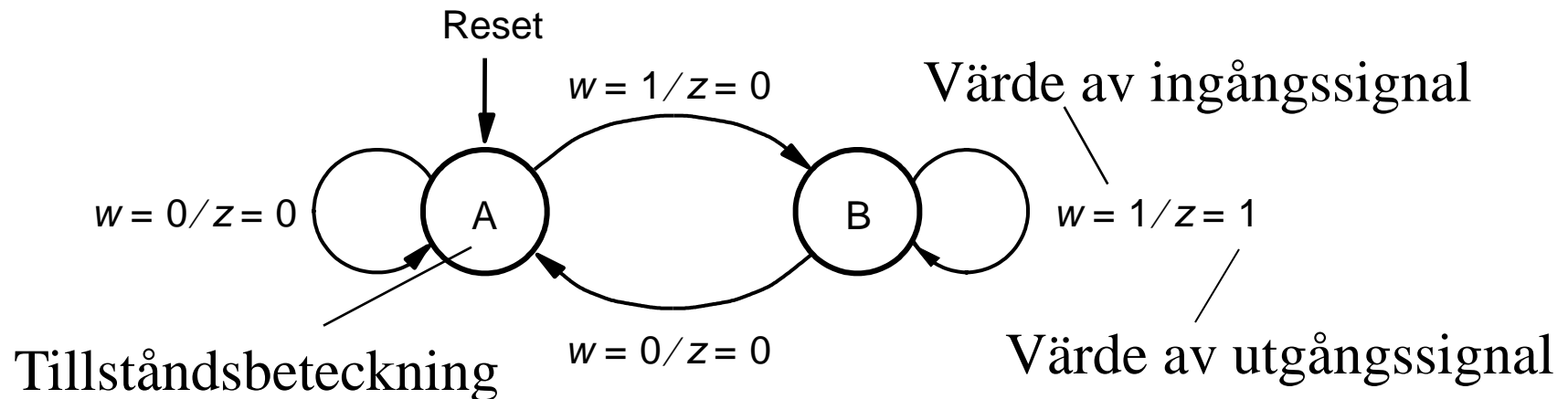


I en **Mealy**-Automat beror utgångssignalerna både på nuvarande tillstånd *och* ingångarna

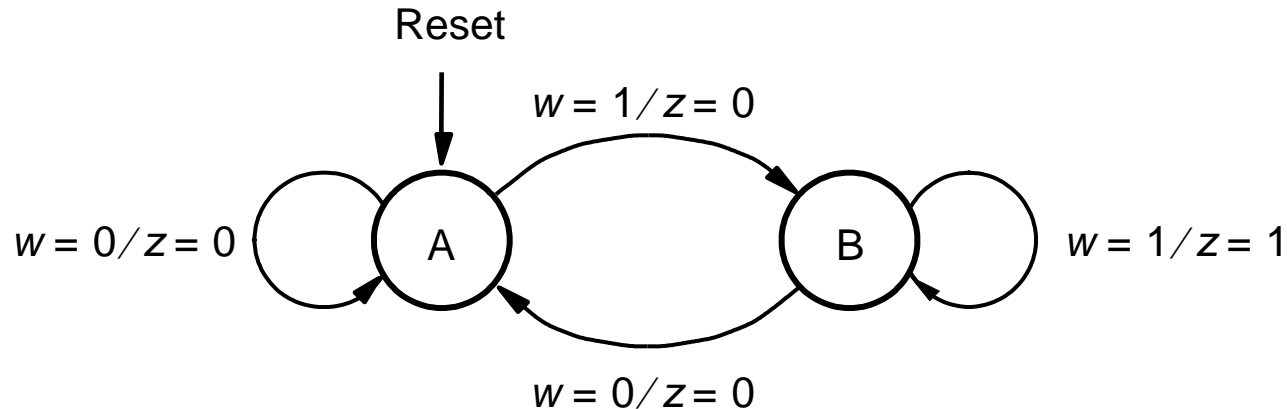
Tillståndsdigram Mealy

”Två i rad”

- Tillståndsdigrammet 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 vipa 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

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

$$\mathbf{A} = 0$$

$$\mathbf{B} = 1$$



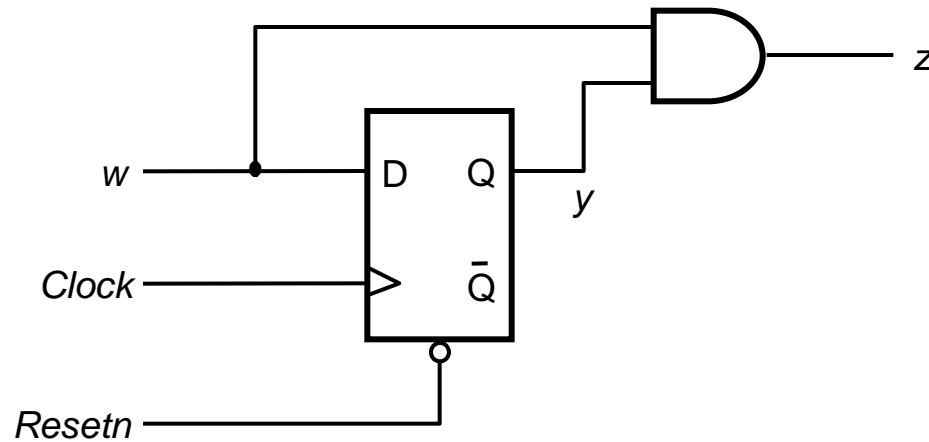
A

B

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

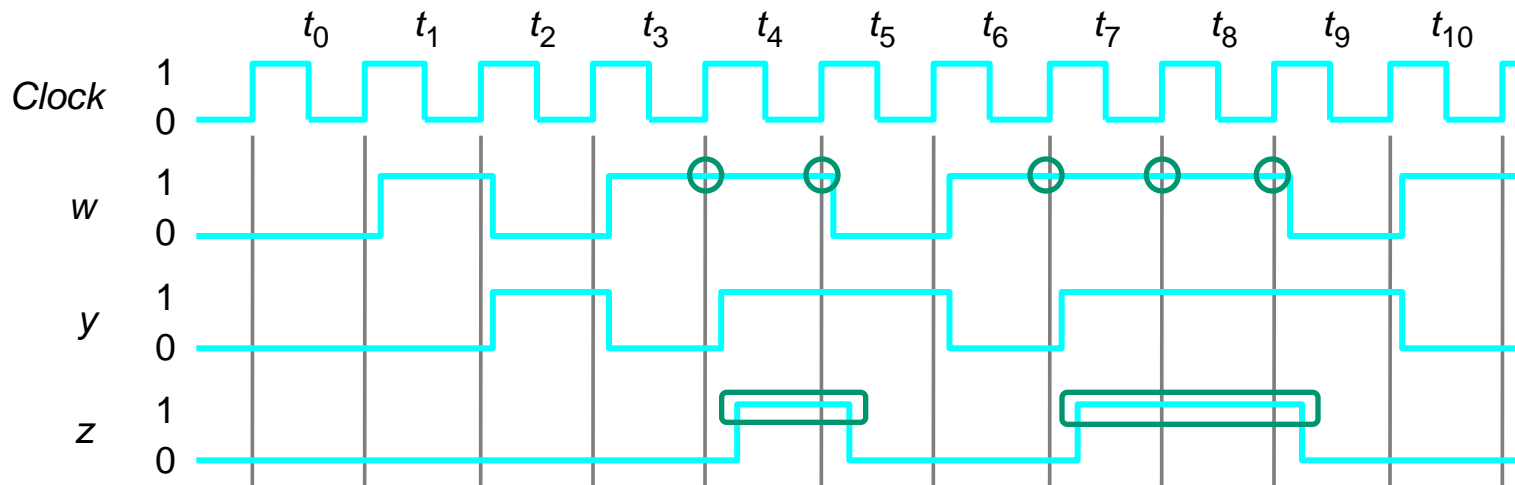
Direkt ur tabellen: $Y = w \quad z = yw$

Implementeringen



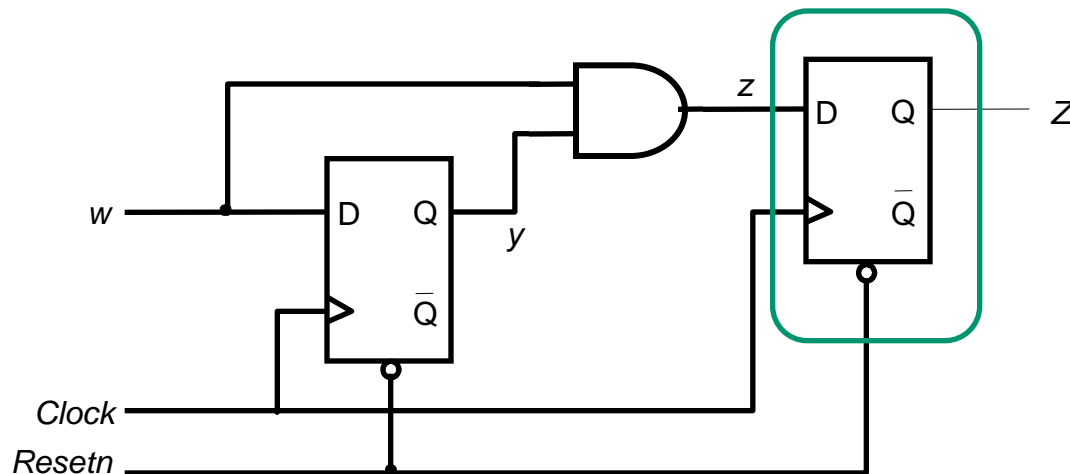
Tidsdiagram

- Utsignalen kan ändras 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)



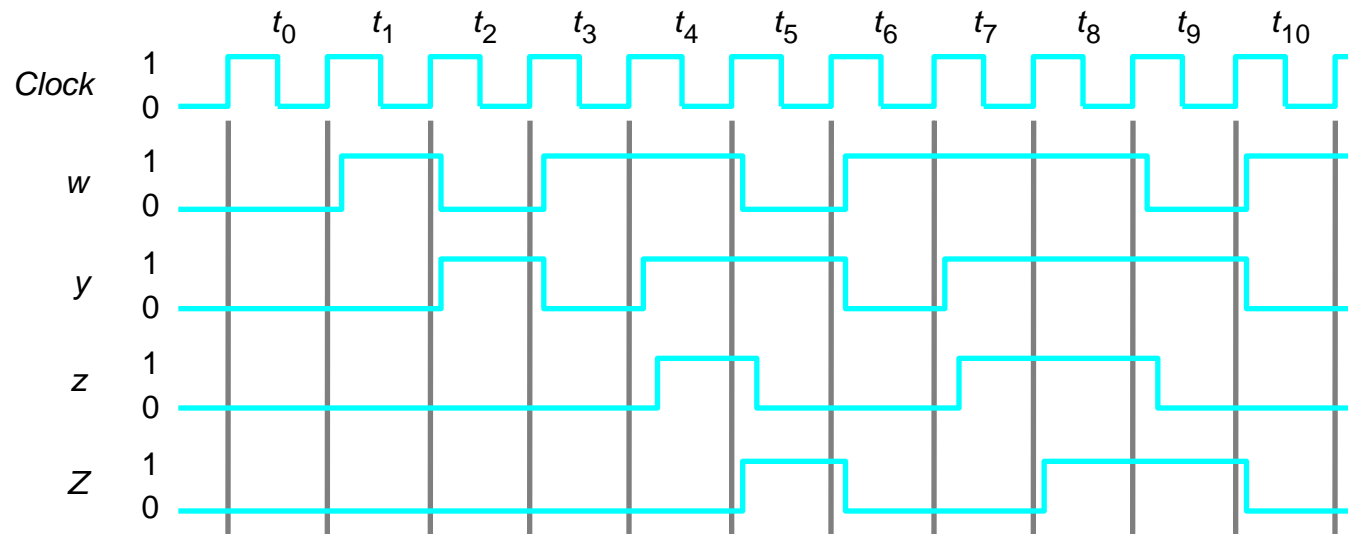
Mealy med utgångsregister

- 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!



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

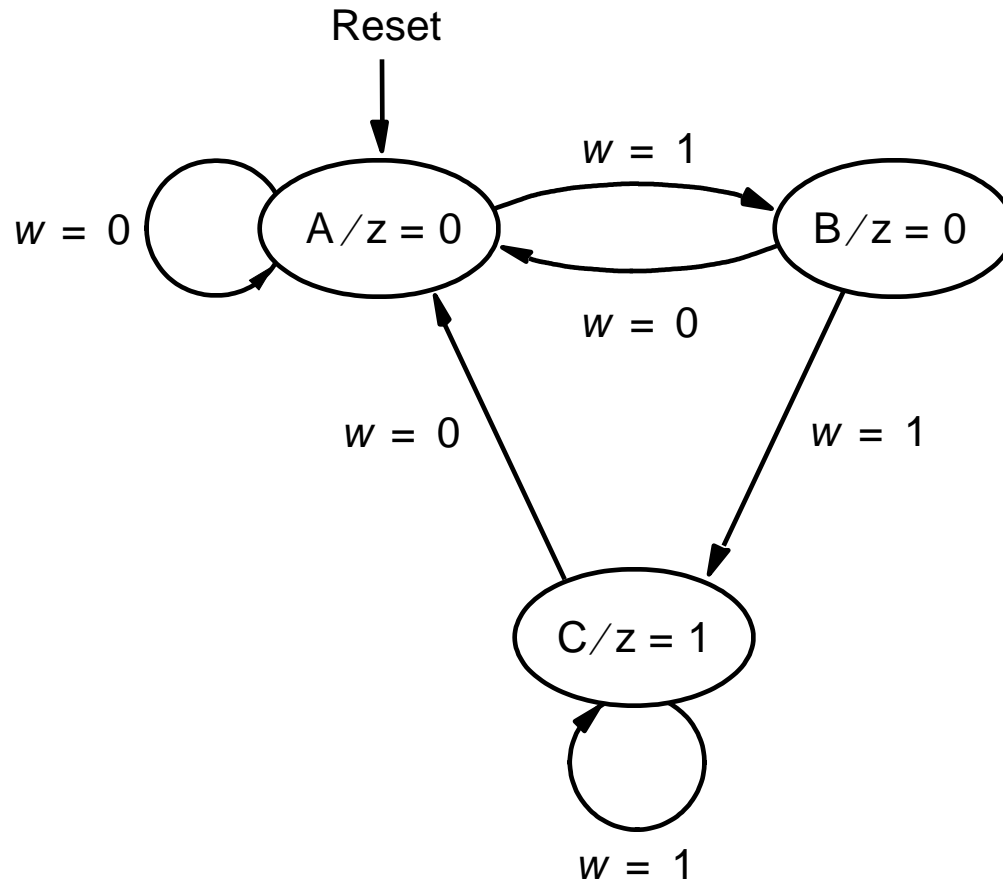
William Sandqvist william@kth.se

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

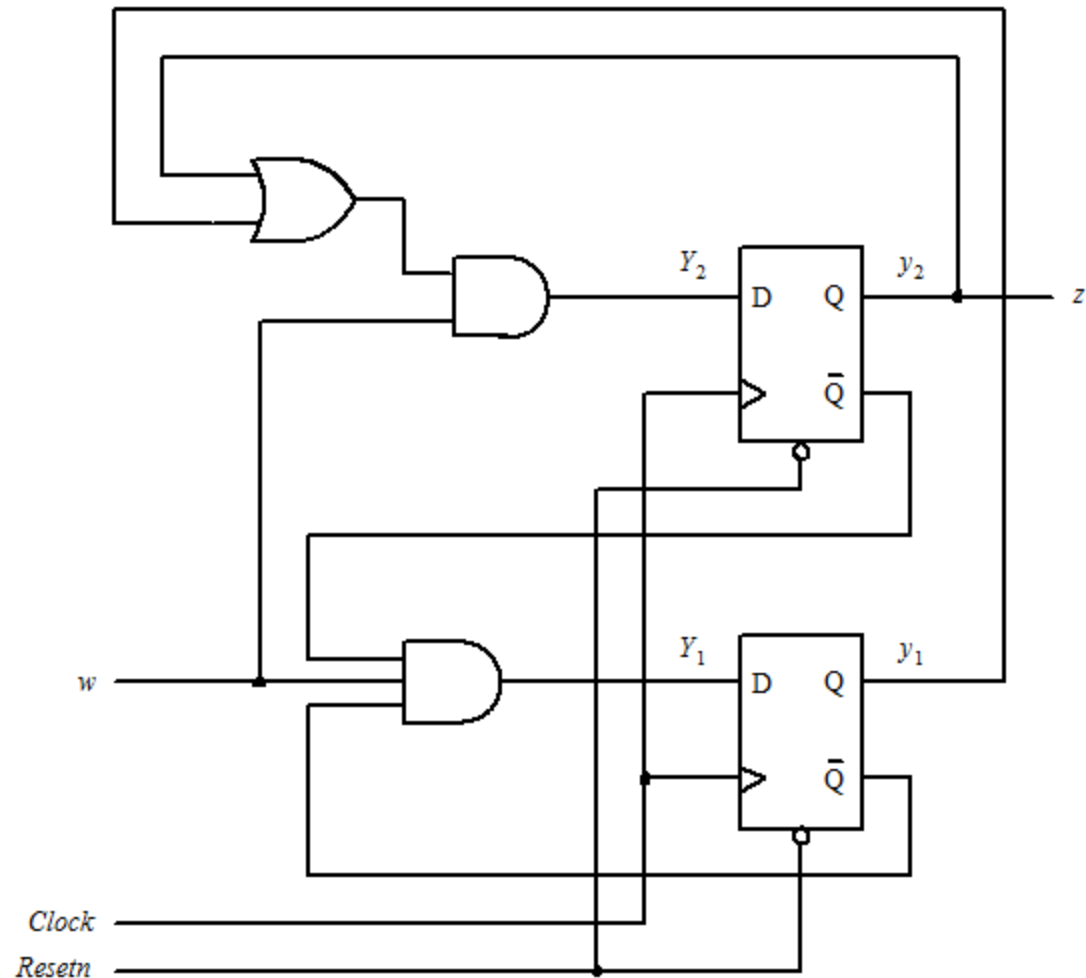
A
 B
 C

→

Present state y_2y_1	Next state		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
00	00	01	0
01	00	10	0
10	00	10	1
11	<i>dd</i>	<i>dd</i>	<i>d</i>

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



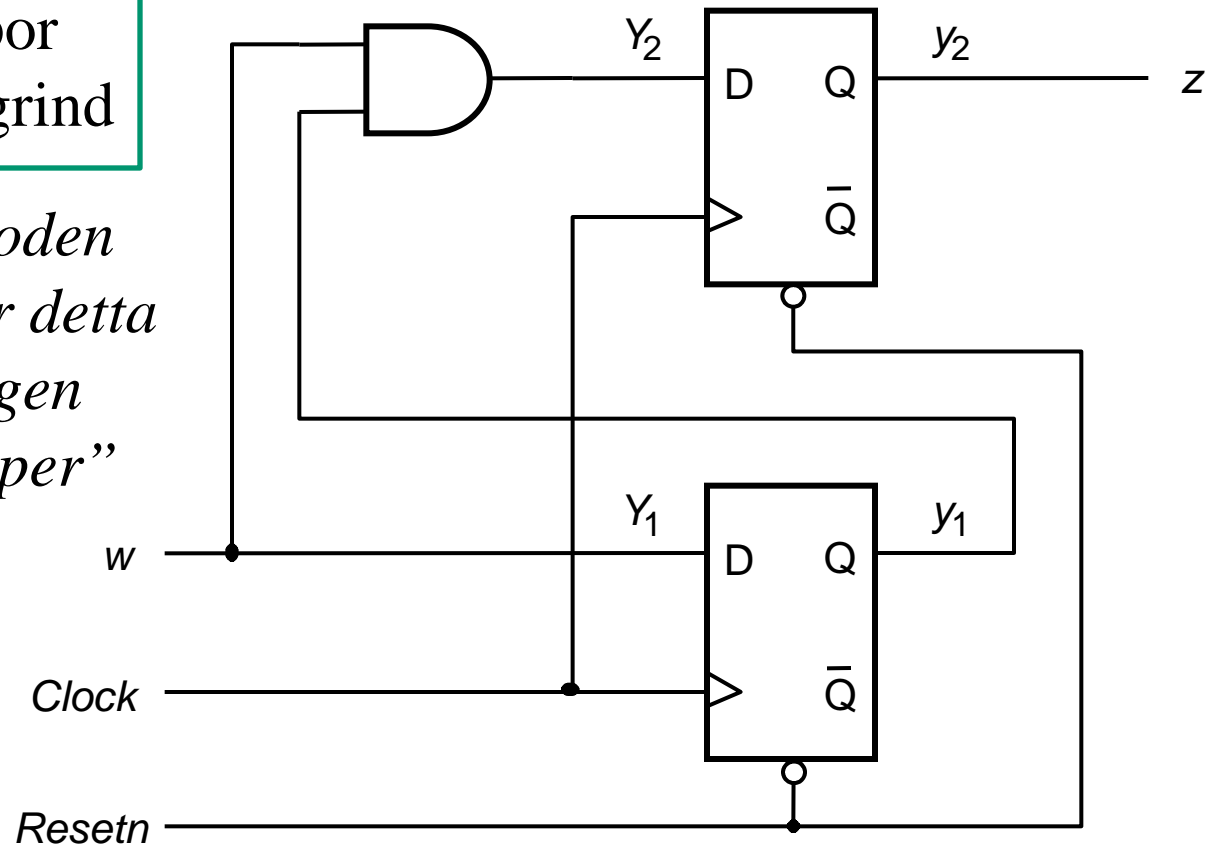
A
B
C

Present state y_2y_1	Next state		Output z
	$w = 0$	$w = 1$	
	Y_2Y_1	Y_2Y_1	
00	00	01	0
01	00	11	0
11	00	11	1
10	<i>dd</i>	<i>dd</i>	<i>d</i>

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 tillståndsdigrammet
- Man kan även ha 'egna koder' som passar till konstruktionen, t ex 00, 11, 10, 01

William Sandqvist william@kth.se

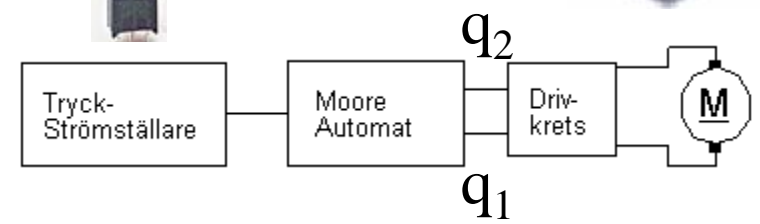
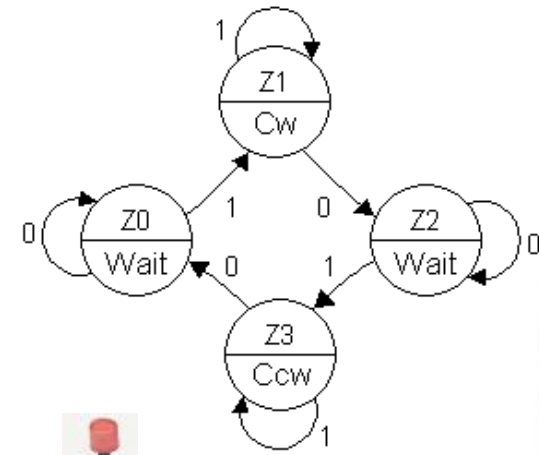
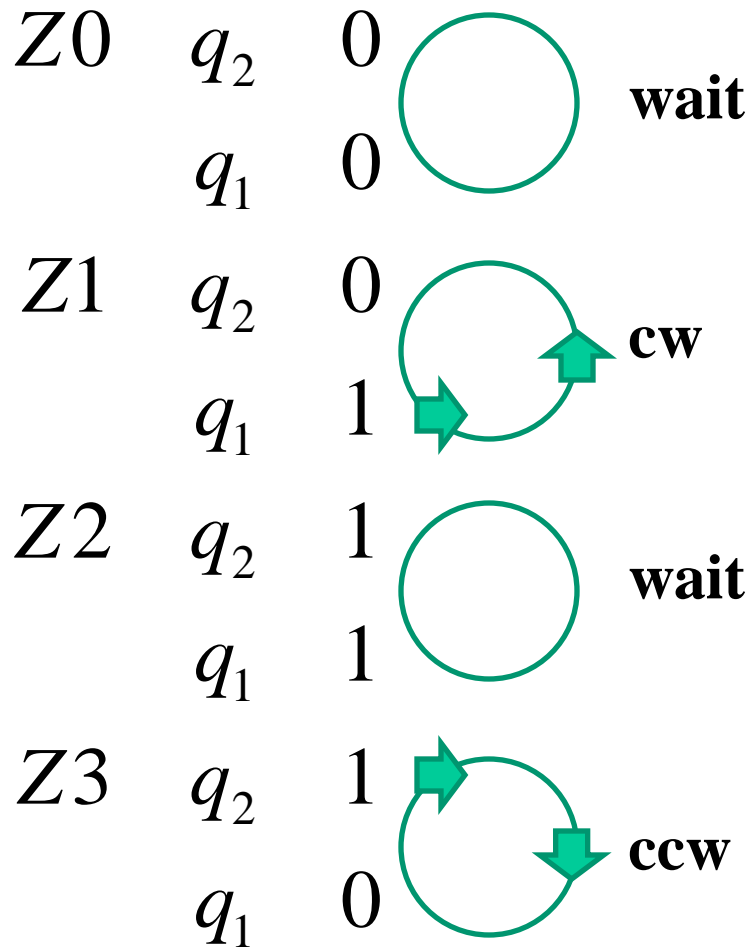
FSM-Demo

Finite State Machine (FSM)

Kör en motor fram/back varannan gång

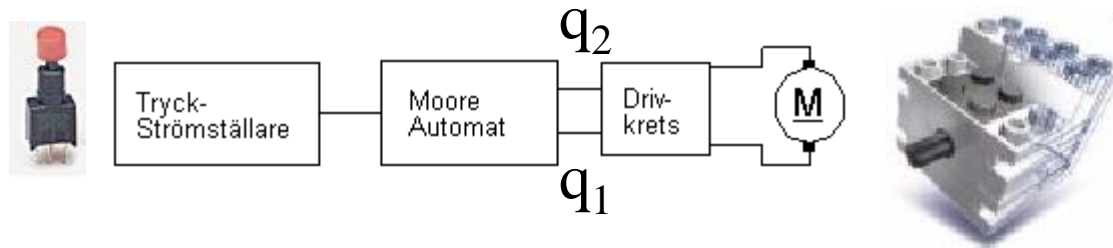
- Automaten med **logikkretsar**
- Programmerbar logik och **VHDL**

Demo. - wait - cw - wait - ccw -



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

Demo. - wait - cw - wait - ccw -



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

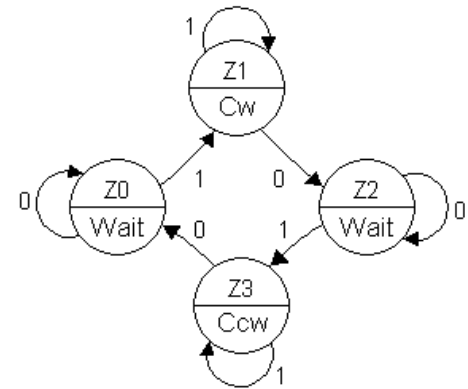
		i	
		0	1
q ₂ q ₁	Z0 00		
	Z1 01		
	Z2 11		
	Z3 10		

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

		i	
		0	1
q ₂ q ₁	0 0		
	0 1		
	1 1		
	1 0		

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

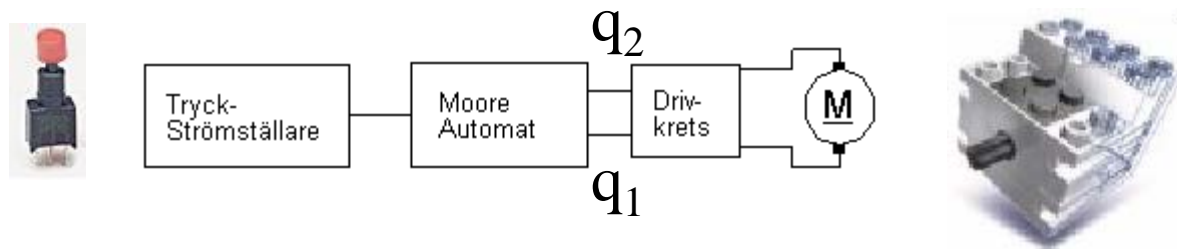
		i	
		0	1
q ₂ q ₁	0 0		
	0 1		
	1 1		
	1 0		



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

William Sandqvist william@kth.se

Demo. - wait - cw - wait - ccw -



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

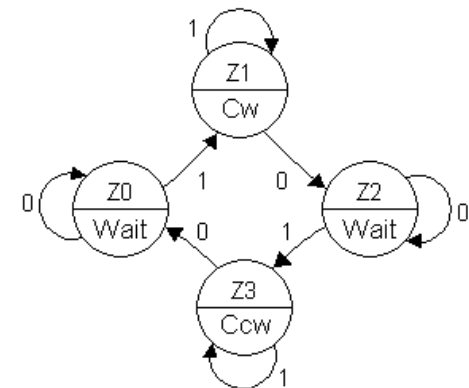
		i	
		0	1
q ₂ q ₁	Z0 00	0 0	0 1
	Z1 01	1 1	0 1
Z2 11	1 1	1 0	
Z3 10	0 0	1 0	

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

		i	
		0	1
q ₂ q ₁	0 0	0	0
	0 1	1	0
1 1	1	1	
1 0	0	1	

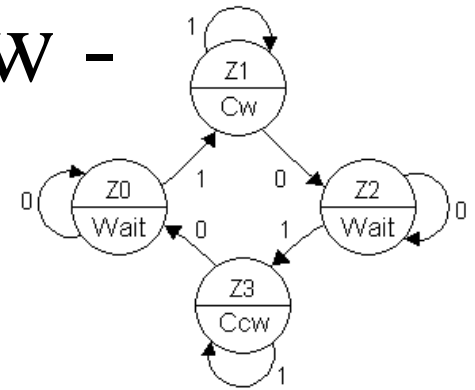
$$q_1^+ = f(i, q_2, q_1)$$

		i	
		0	1
q ₂ q ₁	0 0	0	1
	0 1	1	1
1 1	1	0	
1 0	0	0	



$$q_2^+ = \boxed{\bar{i} q_1} + \boxed{i q_2} \quad q_1^+ = \boxed{\bar{i} q_1} + \boxed{i \bar{q}_2}$$

- wait - cw - wait - ccw -



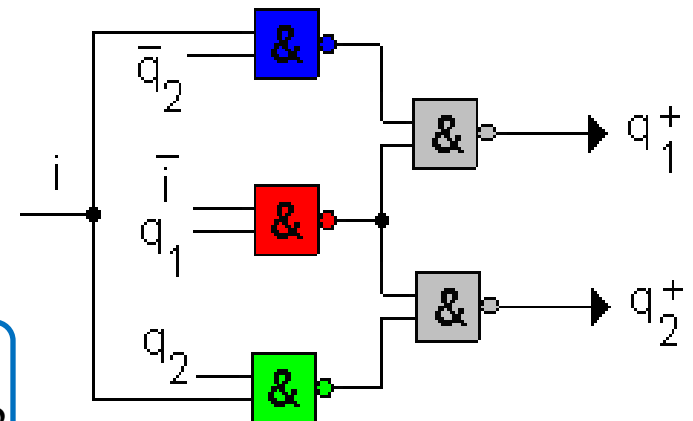
$q_2 q_1$		i	
		0	1
Z0	00	0 0	0 1
Z1	01	1 1	0 1
Z2	11	1 1	1 0
Z3	10	0 0	1 0

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

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

$$q_2^+ = \bar{i} q_1 + i q_2$$

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



Med NAND-grindar

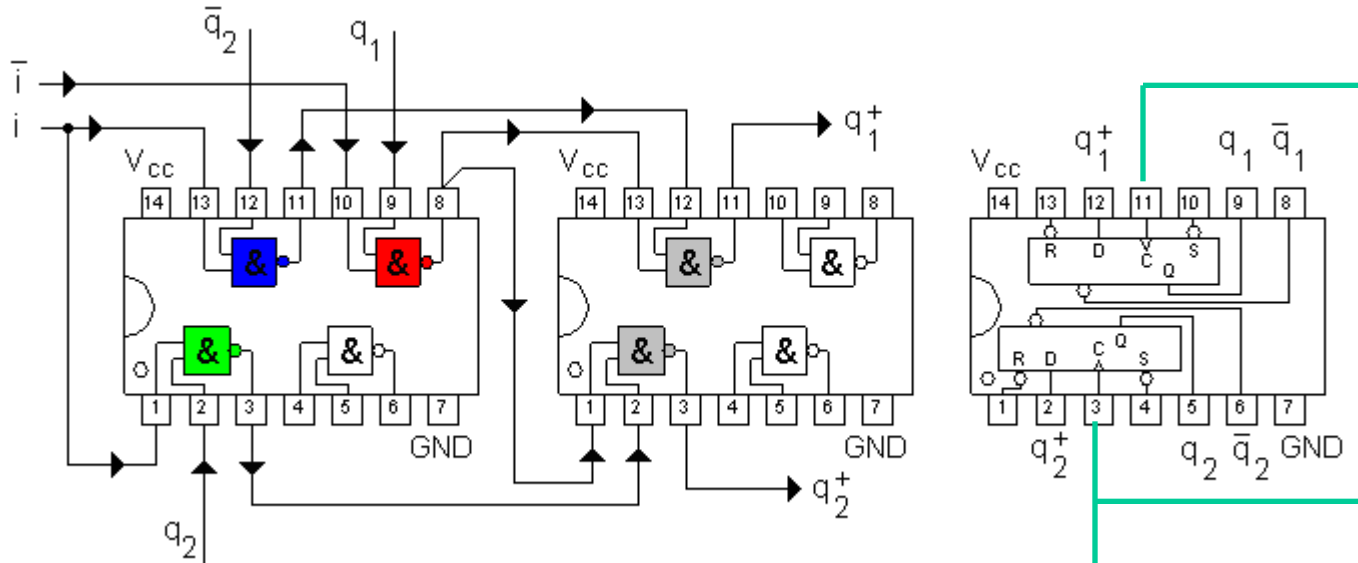
- wait - cw - wait - ccw -

Med 74-seriens standardkretsar

7400

7400

7474

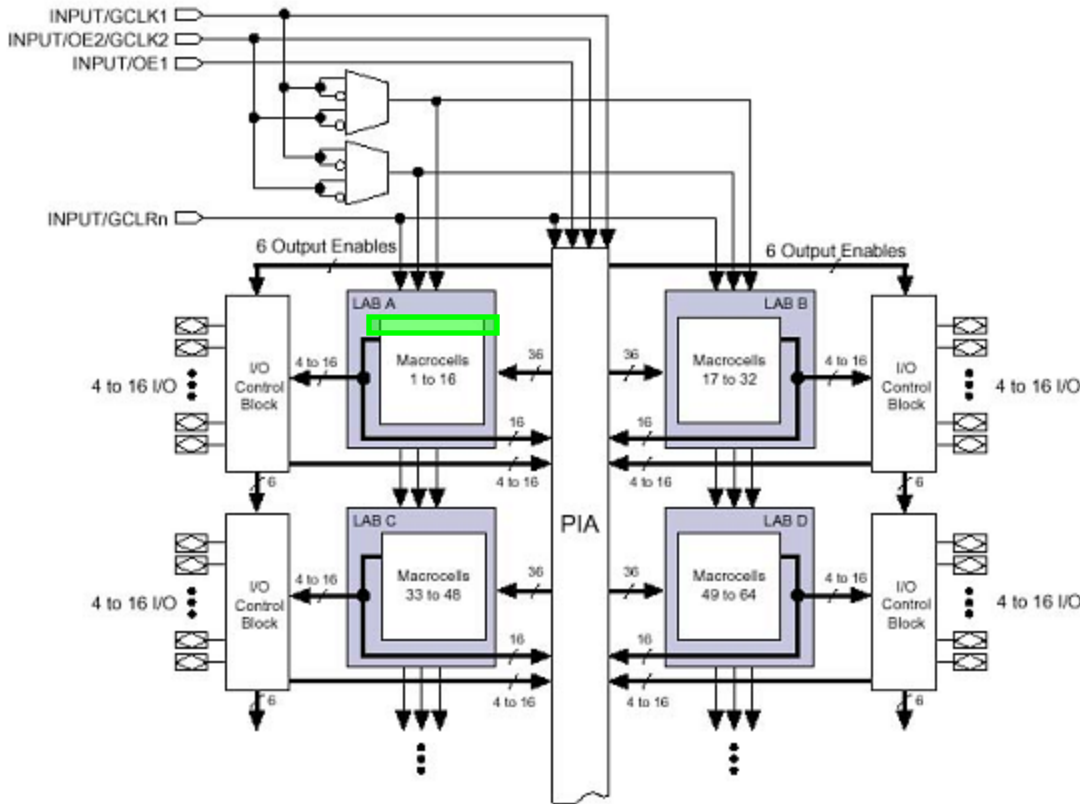


*Det behövs också klockpulser. Tumregel: tre gånger högre frekvens än insignalernas. (här ger **10 Hz** tillräckligt snabb respons samtidigt som ”kontaktstudsar” undertrycks)*



William Sandqvist william@kth.se

CPLD (MAX) ?

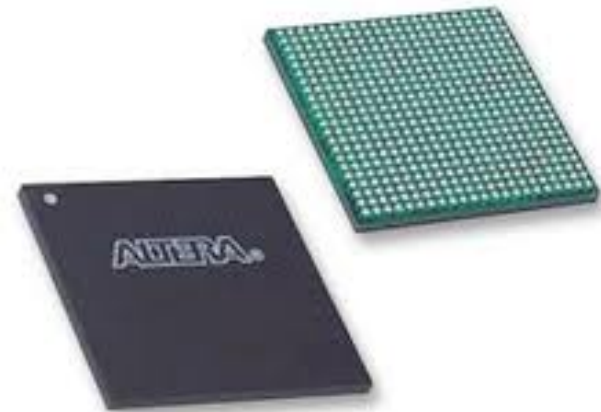
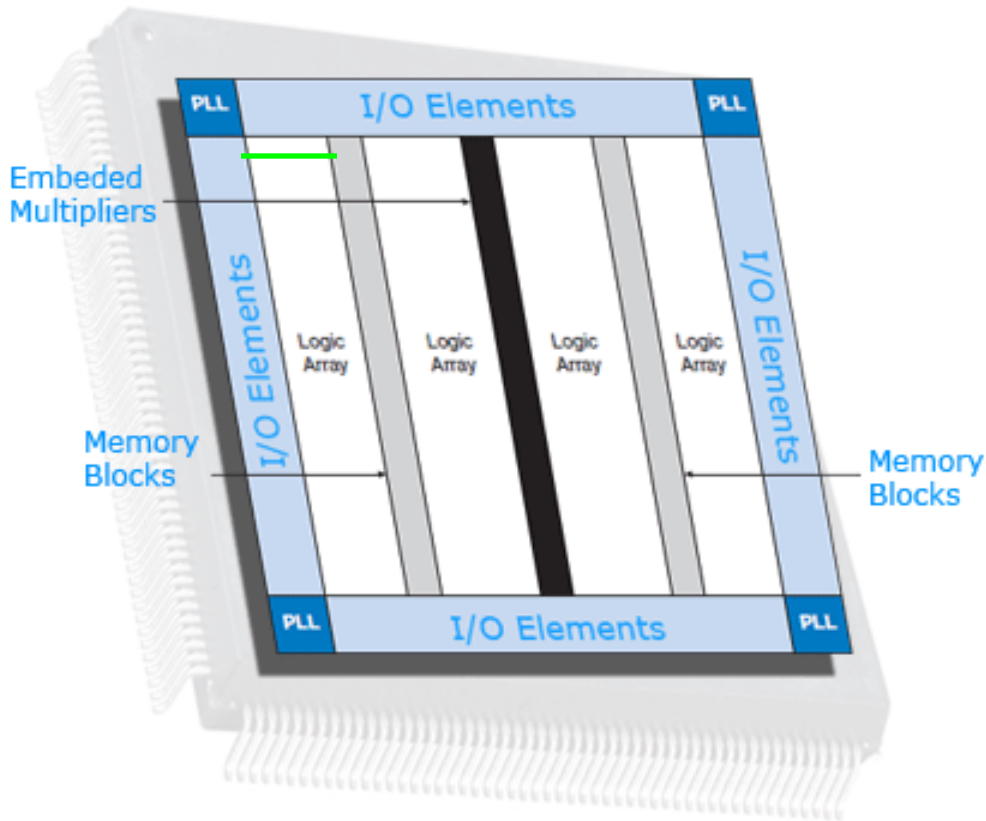


Typiskt **64**
Macroceller

Teknik:
AND-OR
array

(Större MAX
har MUX-tree)

FPGA (Cyclone II) ?



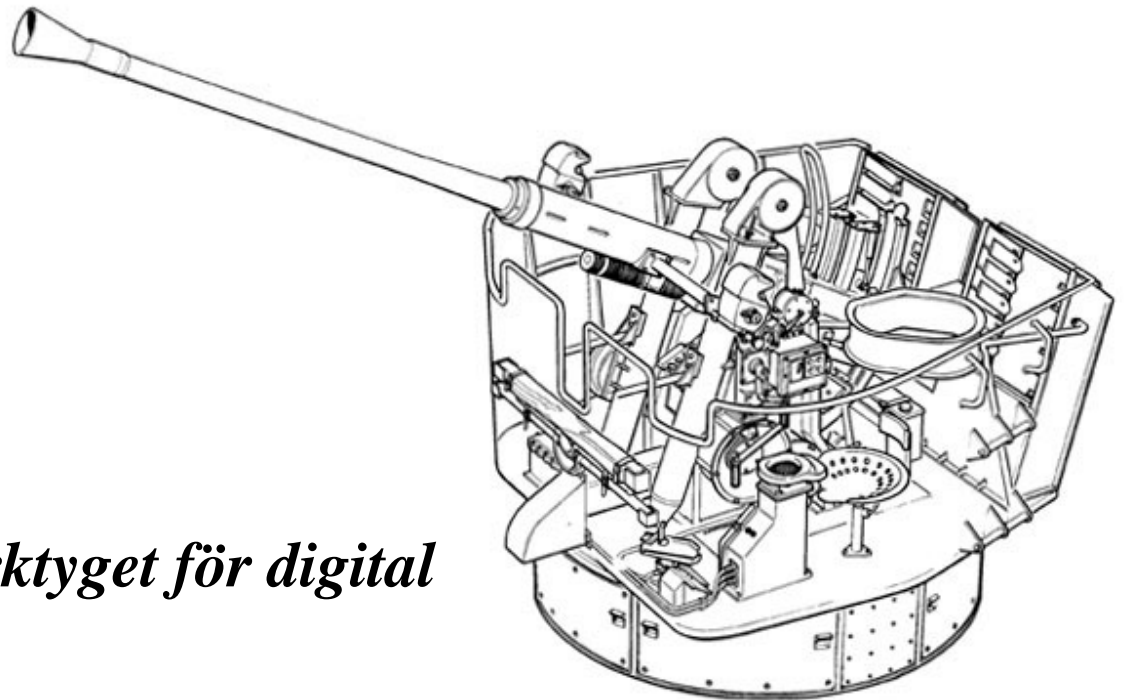
Typiskt **50000** logikelement

Teknik: MUX tree



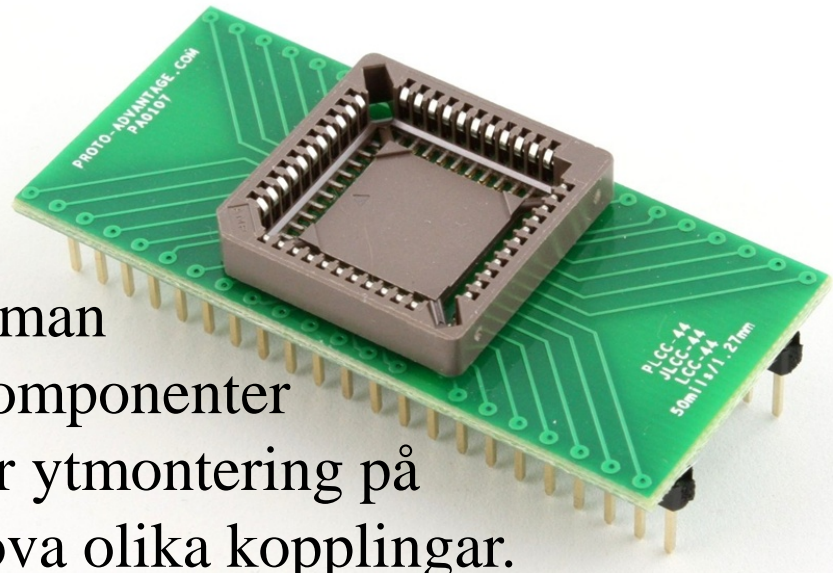
Rätt verktyg ...

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



Vilket är det rätta verktyget för digital designkursen ...

CPLD (MAX)

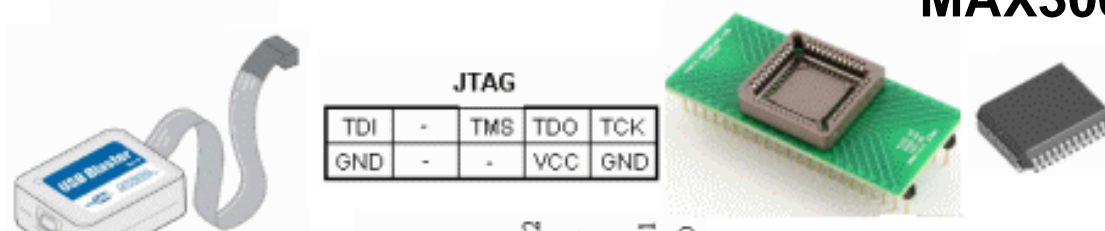


Med ett **breakoutboard** kan man använda kopplingsdäck till komponenter som egentligen är avsedda för ytmontering på kretskort. Man kan enkelt prova olika kopplingar.

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

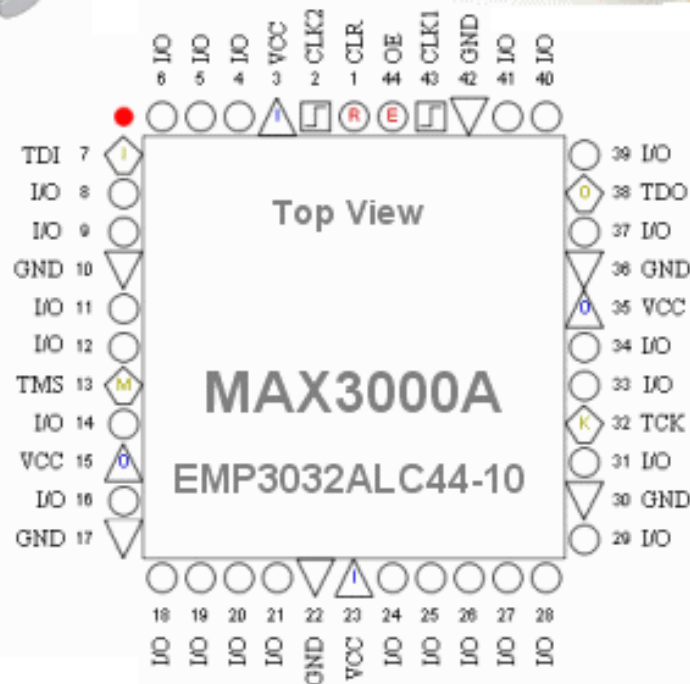
MAX3000



JTAG

TDI	-	TMS	TDO	TCK
GND	-	-	VCC	GND

- 1 CLR
- 2 CLK2
- 3 VCC
- 4 IO
- 5 IO
- 6 IO
- 7 TDI
- 8 IO
- 9 IO
- 10 GND
- 11 IO
- 12 IO
- 13 TMS
- 14 IO
- 15 VCC
- 16 IO
- 17 GND
- 18 IO
- 19 IO
- 20 IO
- 21 IO
- 22 GND



- 44 OE
- 43 CLK1
- 42 GND
- 41 IO
- 40 IO
- 39 IO
- 38 TDO
- 37 IO
- 36 GND
- 36 GND
- 35 VCC
- 34 IO
- 34 IO
- 33 IO
- 32 TCK
- 31 IO
- 30 GND
- 29 IO
- 28 IO
- 27 IO
- 26 IO
- 25 IO
- 24 IO
- 23 VCC

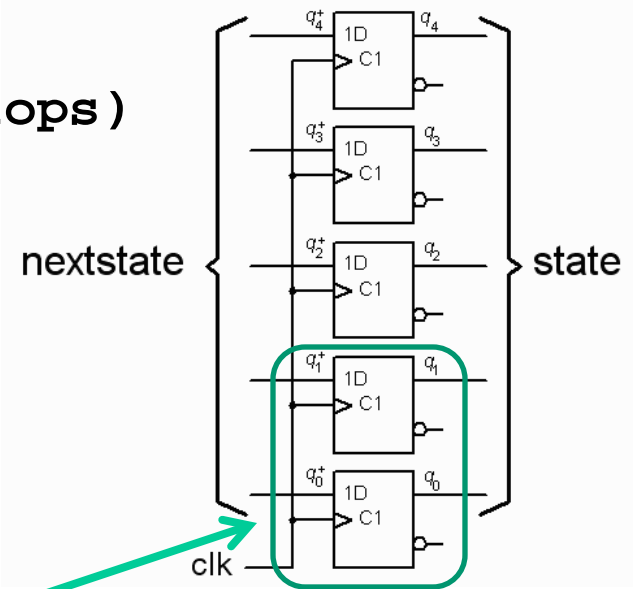


QuartusII

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;
```

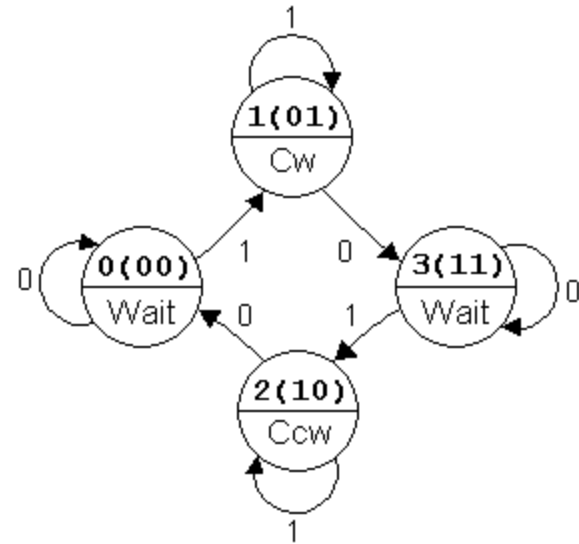
```
      when 3 => next_state <= Ooops;
```

```
      when 2 => next_state <= Ooops;
```

```
    end case;
```

```
  end if ;
```

```
end process;
```

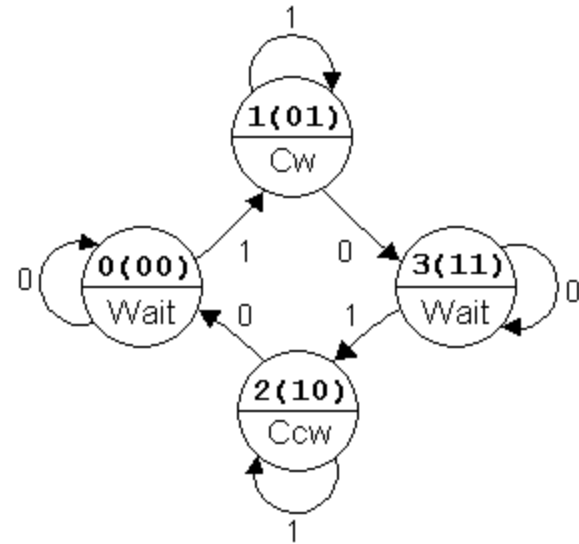


?

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

The screenshot shows the Pin Planner software interface for a MAX3000A EPM3032ALC44-10 device. The main window displays a top view of the device with pins numbered 6 through 44 and 29 through 39. Various pins are highlighted with colored circles and triangles, indicating their assigned functions. A table at the bottom of the window lists the pin assignments for various nodes.

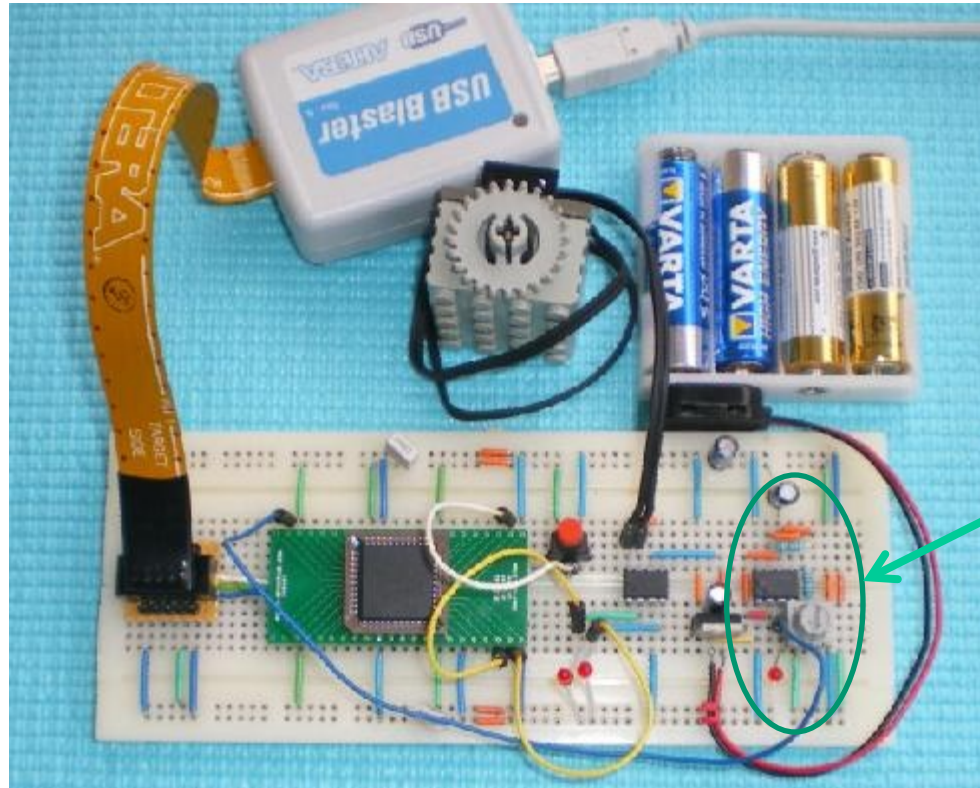
Node Name	Direction	Location	Reserved
CLK	Input	PIN_43	
I	Input	PIN_24	
Q[2]	Output	PIN_20	
Q[1]	Output	PIN_21	
TCK	Input		
TDI	Input		
TDO	Output		
TMS	Input		
<<new node>>			

Chip-programmering



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

Prova resultatet!

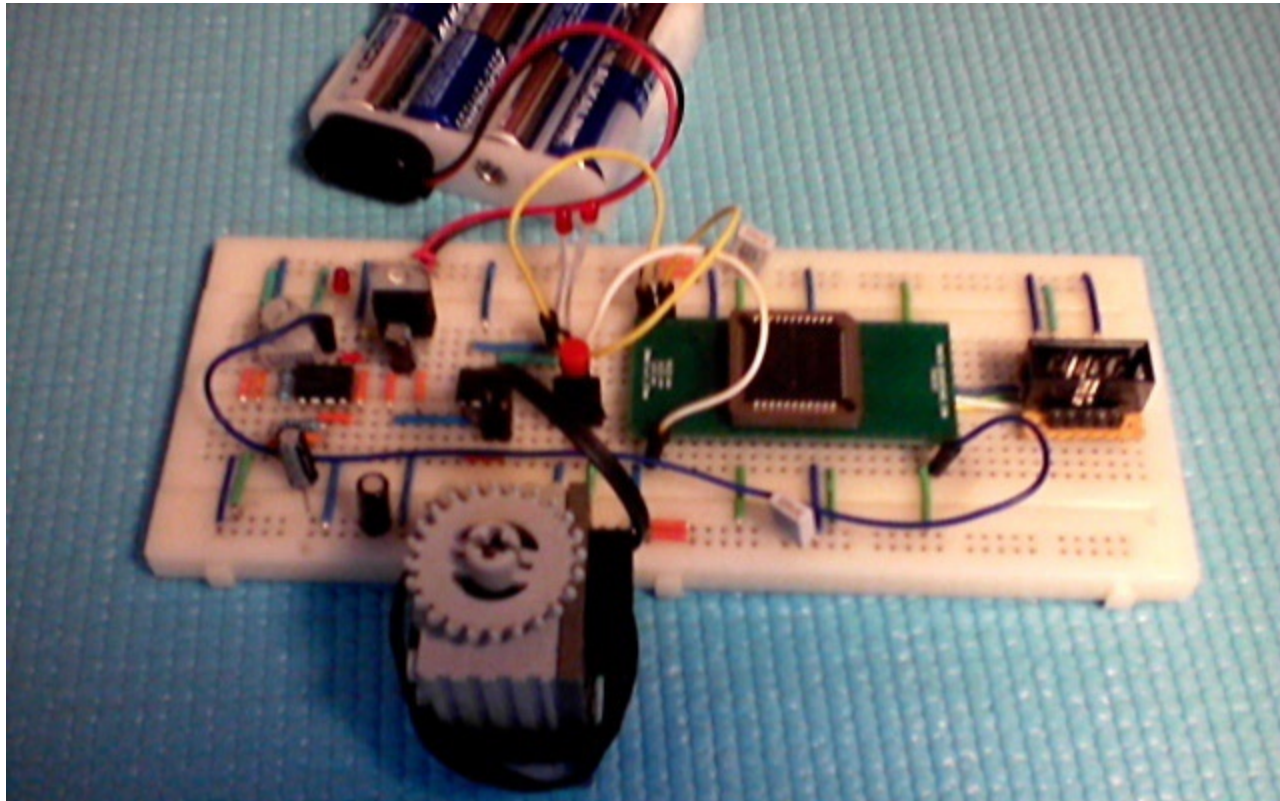


Klockpulser

10 Hz

För denna
tillämpning

MAX-krets fungerar ...



William Sandqvist william@kth.se