Konstruktionsmetodik för sekvenskretsar

Föreläsning 7
Digitalteknik, TSEA22
Mattias Krysander
Institutionen för systemteknik

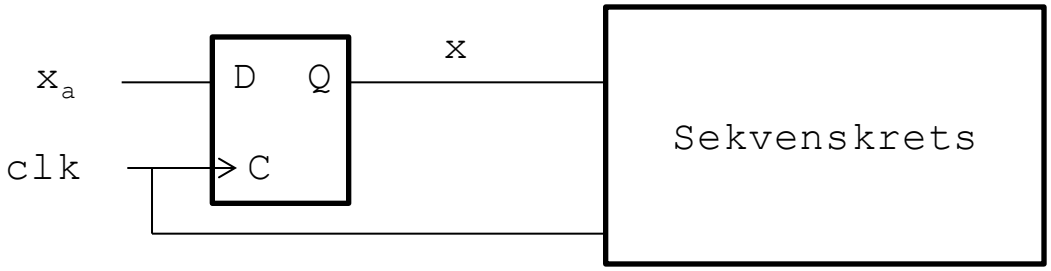
LINKÖPINGS UNIVERSITET

Dagens föreläsning

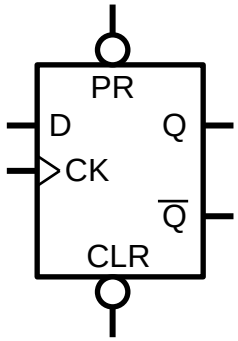
- Initiering av starttillstånd
- Programmerbar logik
- Syntesflödet
- Hårdvarubeskrivande språk VHDL
- Från problemformulering till tillståndsdiagram

Synkronisera asynkrona insignaler

- Asynkron insignaler x_a kommer från
  - Brytare
  - Sensorer
- Använd synkroniseringsvippa:



D-vippa med asynkrona ingångar



PR	CLR	CK	D	Q^+
1	0	X	X	0
0	1	X	X	1
1	1	0	X	Q
1	1	1	X	Q
1	1	↑	0	0
1	1	↑	1	1

Asynkrona ingångar:
Clear (CLR), Preset (PR) aktivt låg

CLR = 0 => Q = 0

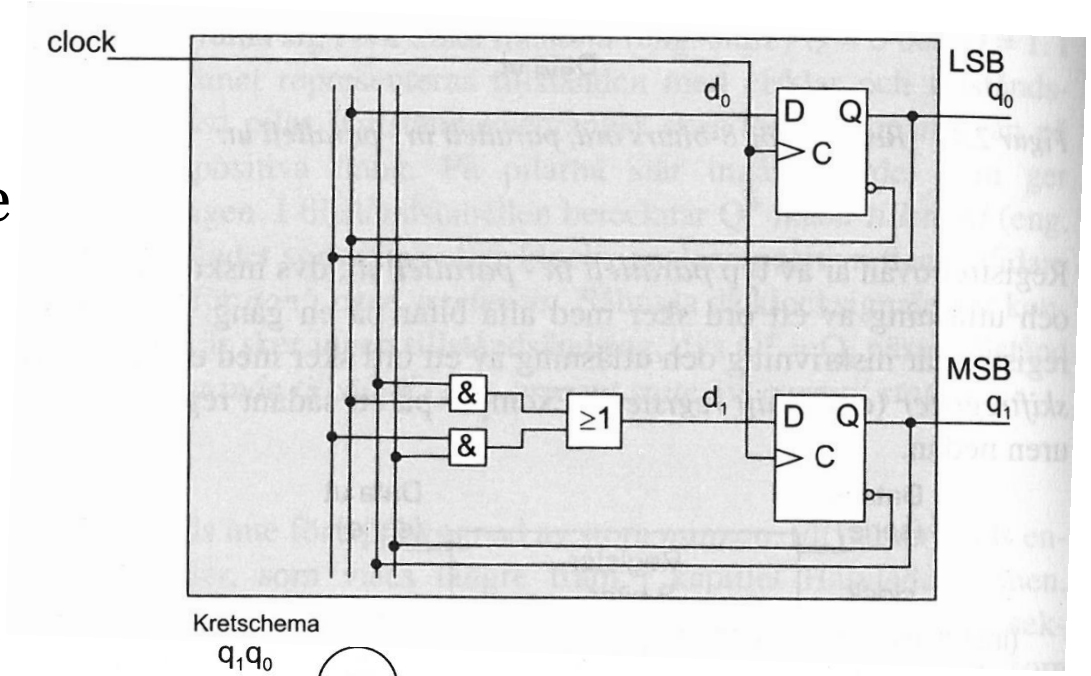
PR = 0 => Q = 1

Vipporna på labben har CLR men ej PR.

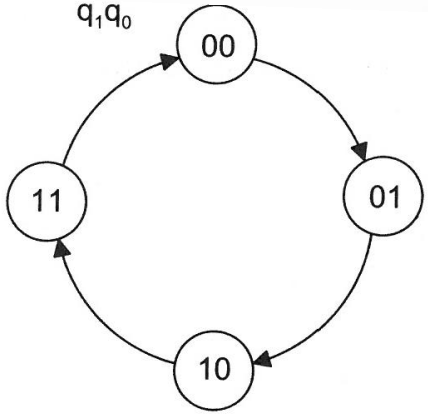
Nollställning ett exempel

Autonom 2-bitsräknare ska förses med nollställning

- Asynkront
- Synkront



Kretschema



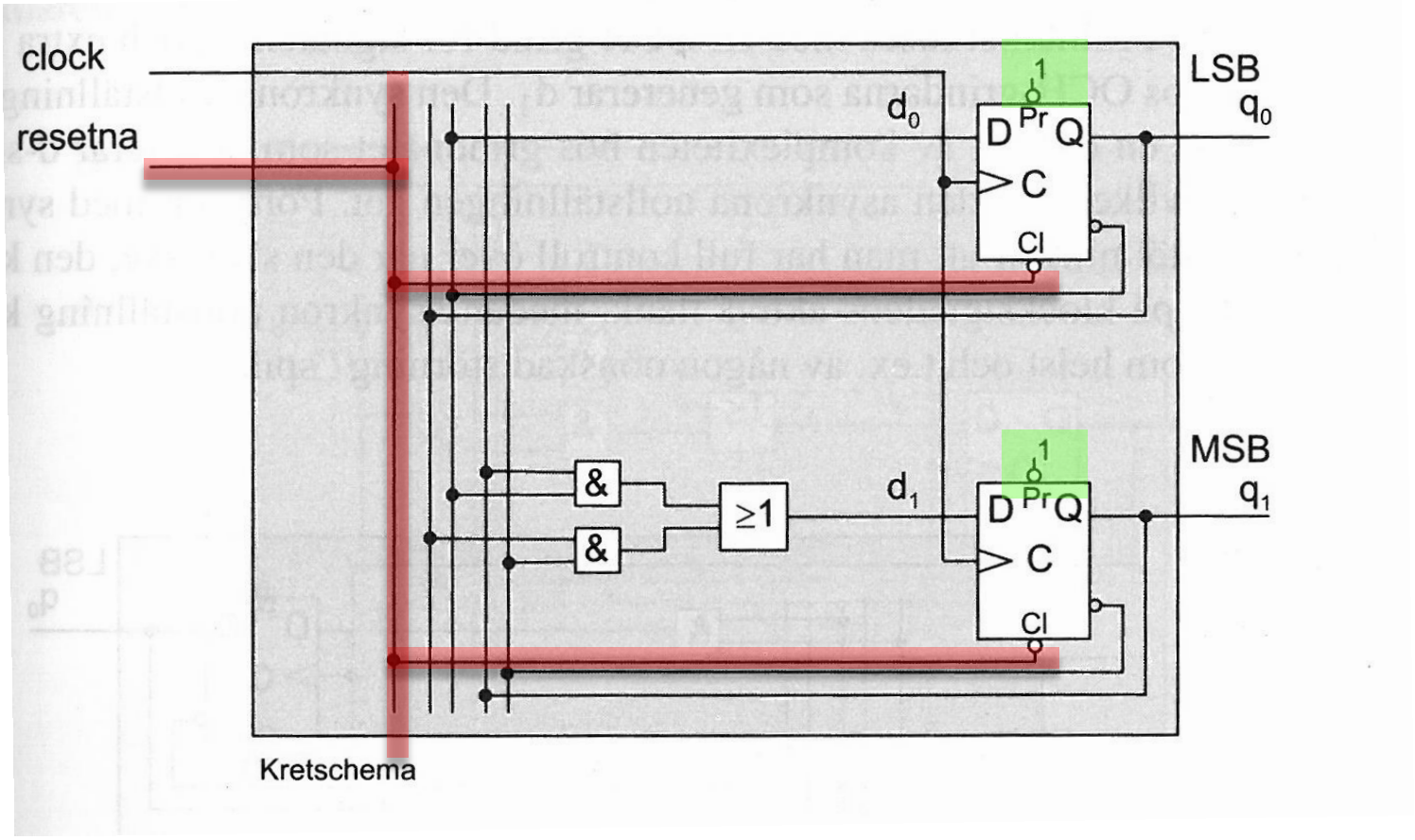
Tillståndsdiagram

Nuvarande tillstånd	Nästa tillstånd
00	01
01	10
10	11
11	00

Tillståndstabell

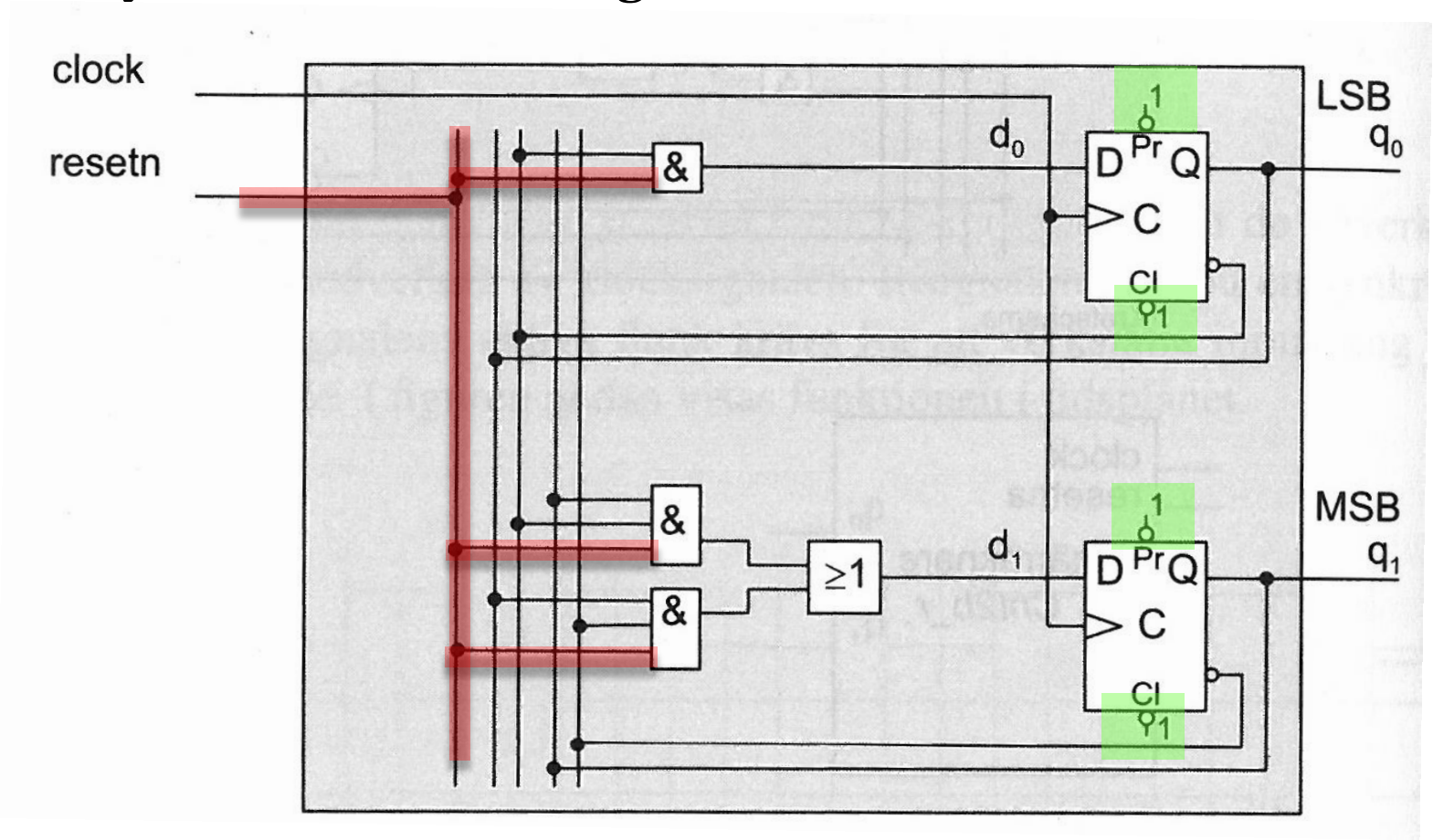
Asynkron nollställning

- Nollställning sker med resetna = 0, utan medverkan av klockan.



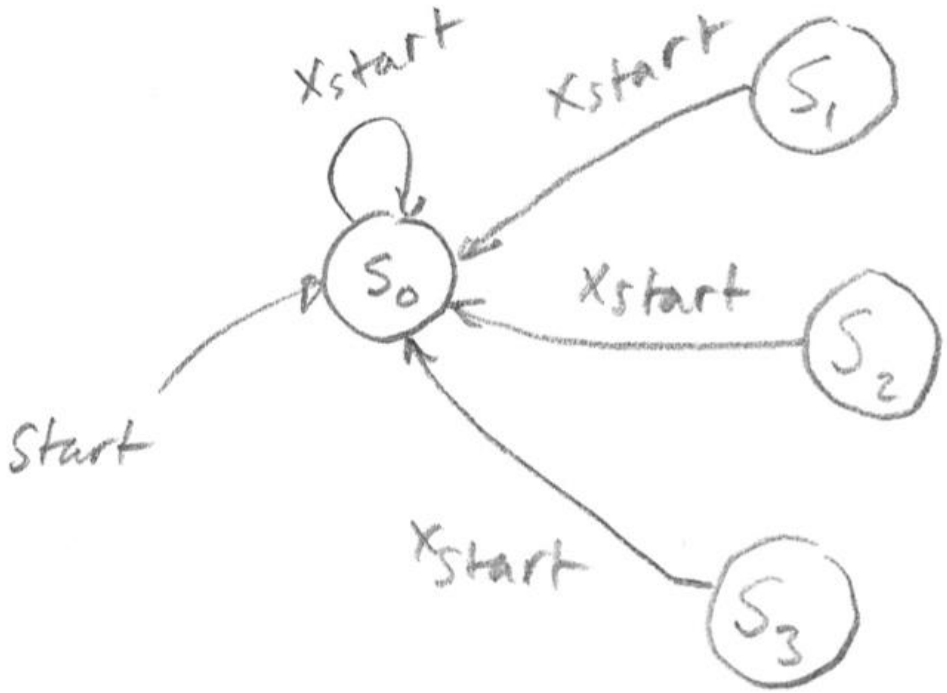
Synkron nollställning

- Synkron nollställning aktiverad med resetn = 0



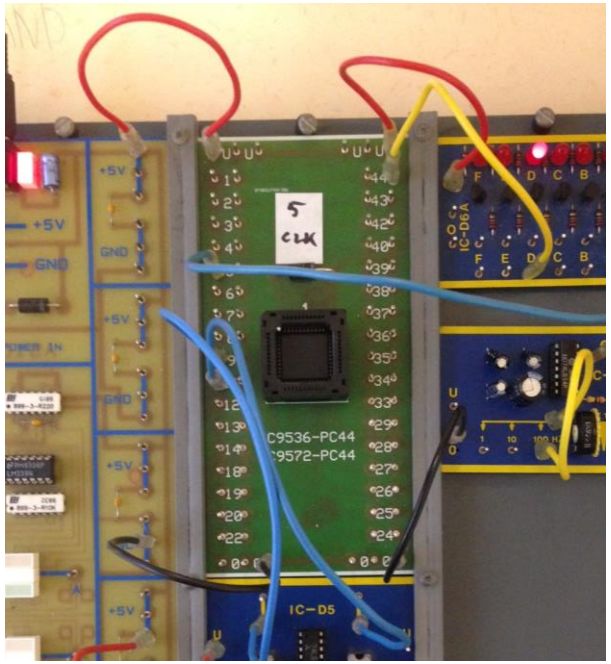
Synkron nollställning

- En insignalkombination sätter nästa tillstånd till starttillståndet oberoende av nuvarande tillstånd.
  - Hanteras därmed som vilken insignal som helst



Programmerbara kretsar

- Istället för att koppla ihop grindar eller konstruera egna integrerade kretsar så finns det kretsar vars funktion kan programmeras
- Programmable logic device (PLD)
- Olika tekniker har använts
  - PROM kan användas för godtycklig funktion
  - Idag används primärt CPLD eller FPGA

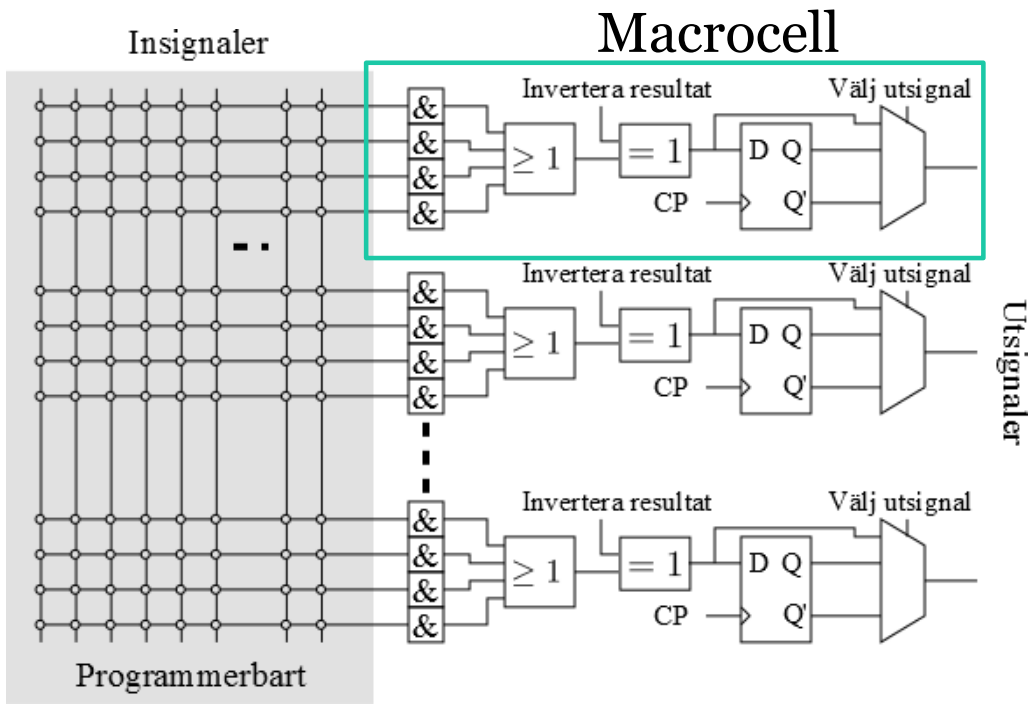


Programmerbara kretsar

- CPLD = complex PLD
  - I princip flera PLD:er på ett chip
  - Ex: 108 vippor + 540 produkttermer
- FPGA = field programmable gate array
  - 5 000 000 vippor
  - 2 000 000 Look-up-tables (LUT), 4-6 ingångar
  - 500 Mb RAM
  - Digital Signal Processing (DSP)
  - Processor

CPLD - konstruktion

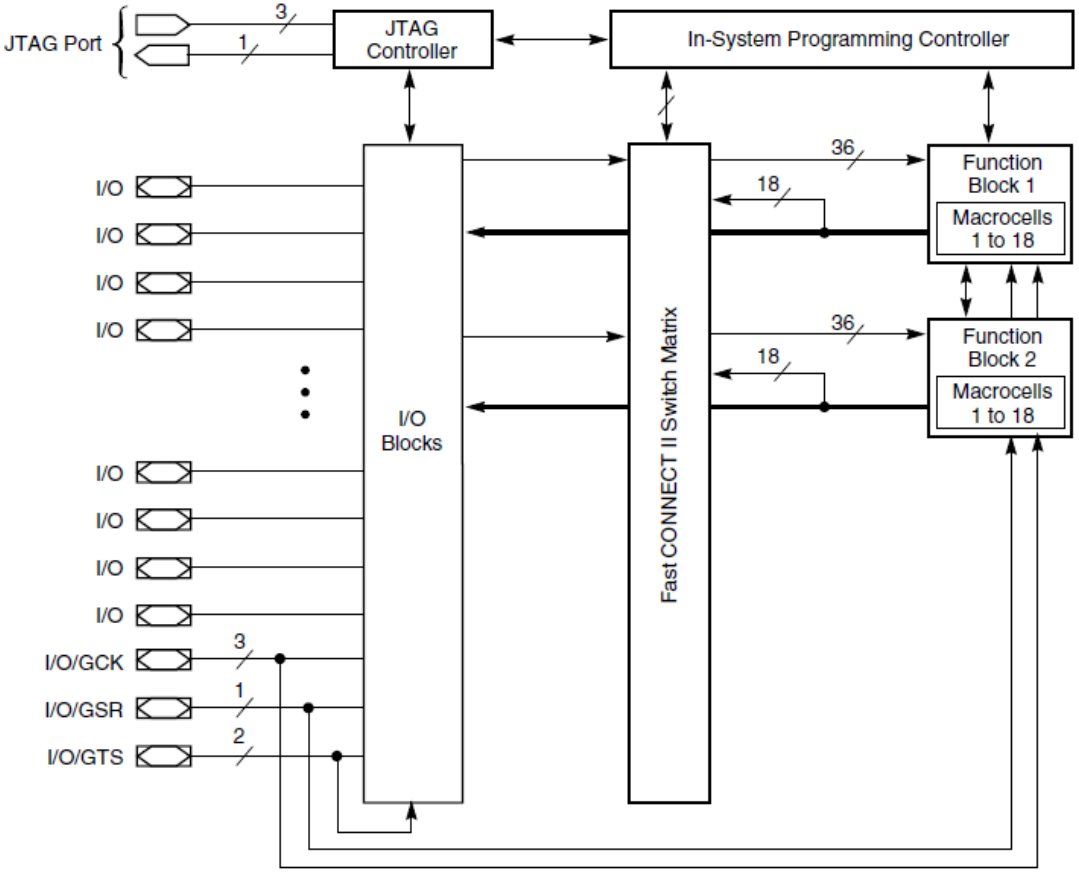
- Grundblocket i en CPLD består oftast av ett AND-OR-nät
- AND-grindarnas ingångar är programmerbara
- Kallas Programmable Logic Array (PLA)



CPLDn på labben (Xilinx XC9536)

Två 36V18-block

- 18 utsignaler från 36 insignaler
- Switchmatris för att koppla ihop in- och utgångar samt in- och utsignaler till/från blocken
- Bild från datablad



VHDL

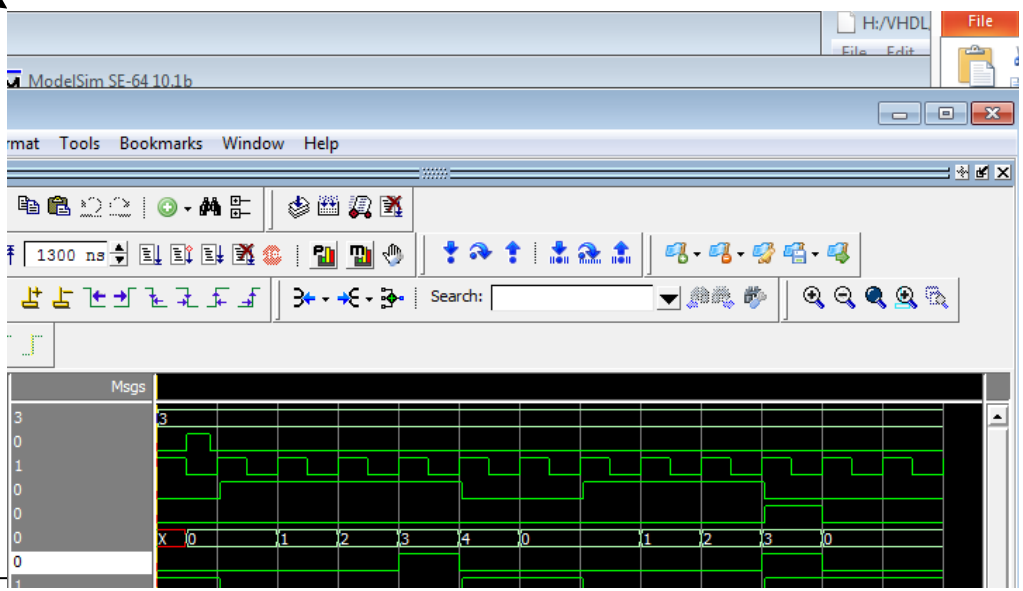
- VHDL = VHSIC Hardware Description Language
  - VHSIC = Very High Speed Integrated Circuit

- Ett programspråk för att:

Syntetisera
(Xilinx)

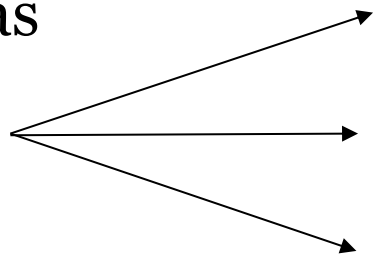
Hårdvara

Simulera



Varför VHDL?

- Hantera komplexitet
  - VHDL-koden kan simuleras
  - Beskrivning på flera olika abstraktionsnivåer
- Ökad produktivitet
  - Snabbare än schemaritning
  - Återanvändbar kod



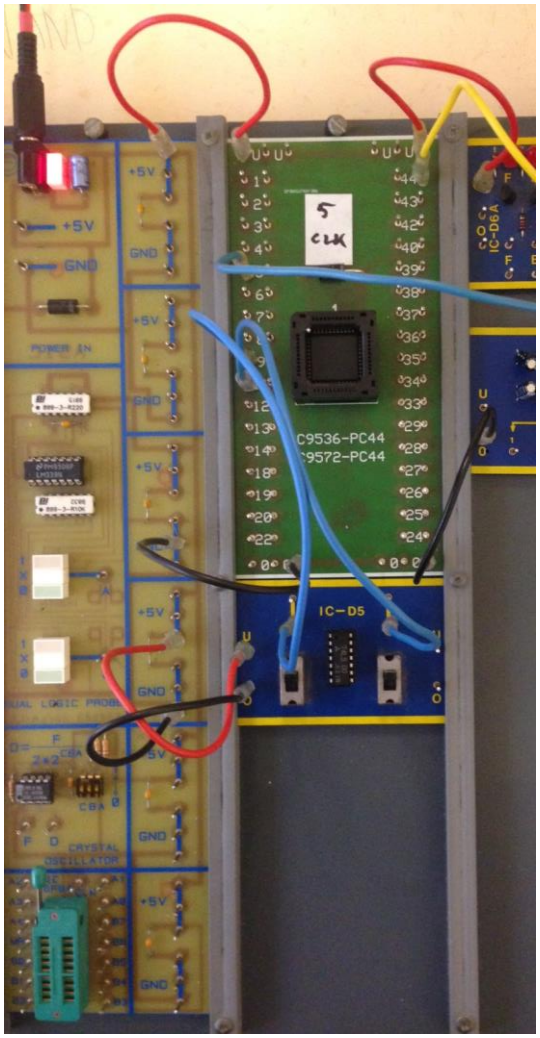
1) a+b

2) Kedja av heladderare

3) Bygg adderare med NAND-grindar

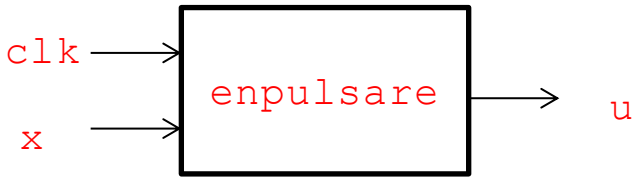
Konstruktion med CPLD

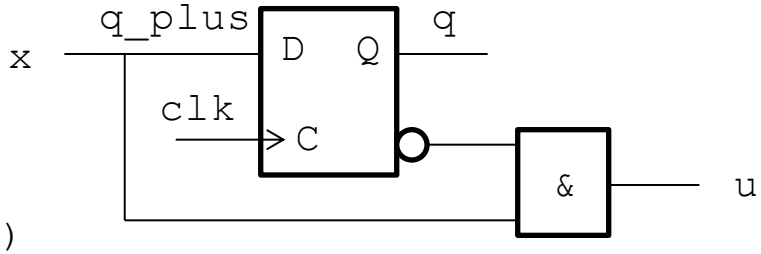
- Rita kretsschema
- Översätt till VHDL (vhd-fil)
- Syntes (skapa en jed-fil)
  - passar in (optimerar) kretsen på den aktuella CPLD:n
  - bestämmer vilka in och utgångar som kommer att användas
- Programmering (använd jed-filen)
  - speciell mjuk- och hård-vara används för att programmera CPLD:n



VHDL-exempel - enpulsaren

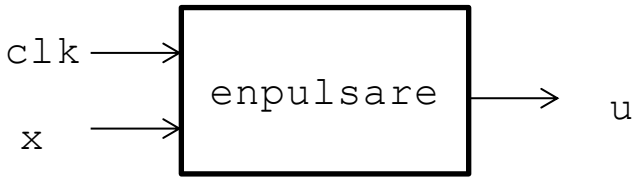
```
library ieee;
use ieee.std_logic_1164.all;

entity enpulsare is
  port(clk, x : in  std_logic;
            u : out std_logic);
end enpulsare;

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```

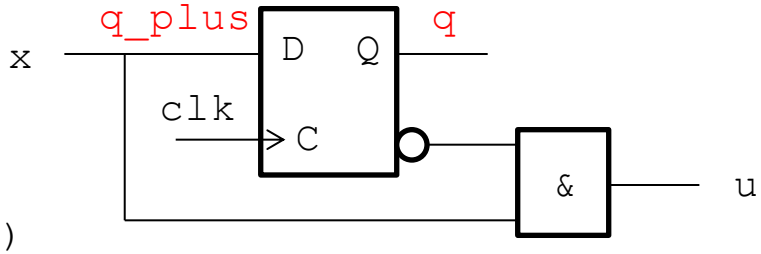




VHDL-exempel - enpulsaren

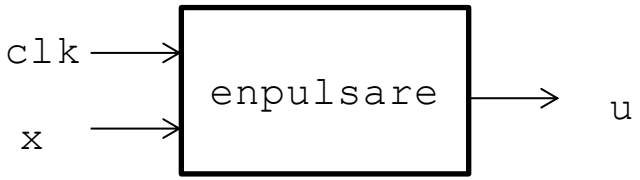
```
library ieee;
use ieee.std_logic_1164.all;

entity enpulsare is
  port(clk, x : in  std_logic;
            u : out std_logic);
end enpulsare;

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```

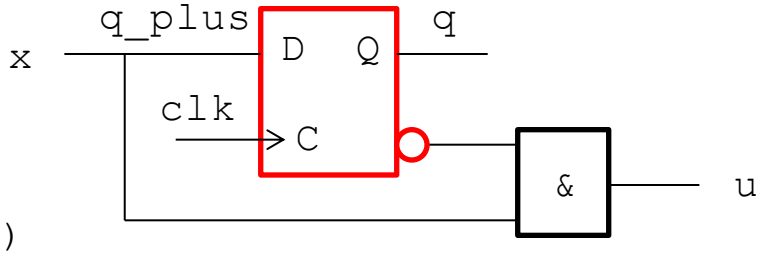




VHDL-exempel - enpulsaren

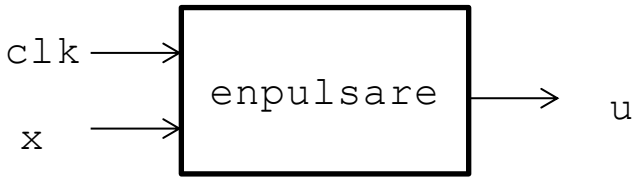
```
library ieee;
use ieee.std_logic_1164.all;

entity enpulsare is
  port(clk, x : in  std_logic;
            u : out std_logic);
end enpulsare;

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```

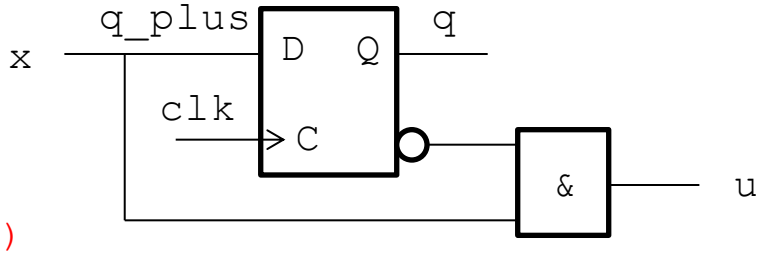




VHDL-exempel - enpulsaren

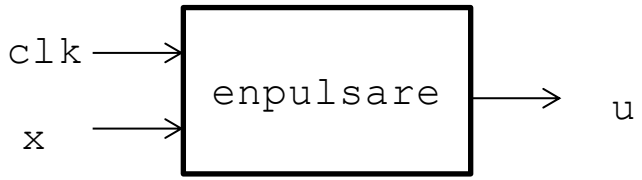
```
library ieee;
use ieee.std_logic_1164.all;

entity enpulsare is
  port(clk, x : in  std_logic;
            u : out std_logic);
end enpulsare;

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                  -- q+ = f(q,x)
  u <= (not q) and x;           -- u  = g(q,x)
end ekvationer;
```

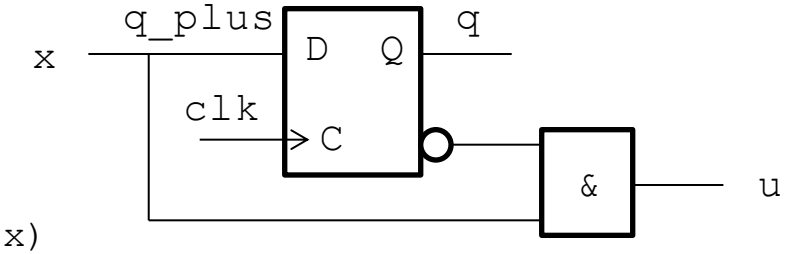




VHDL-exempel - enpulsaren

```
library ieee;
use ieee.std_logic_1164.all;

entity enpulsare is
  port(clk, x : in  std_logic;
            u : out std_logic);
end enpulsare;

architecture ekvationer of enpulsare is
  signal q, q_plus : std_logic;
  begin
  process(clk)
    begin
      if rising_edge(clk) then
        q <= q_plus;
      end if;
    end process;
  q_plus <= x;                -- q+ = f(q,x)
  u <= (not q) and x;         -- u  = g(q,x)
end ekvationer;
```

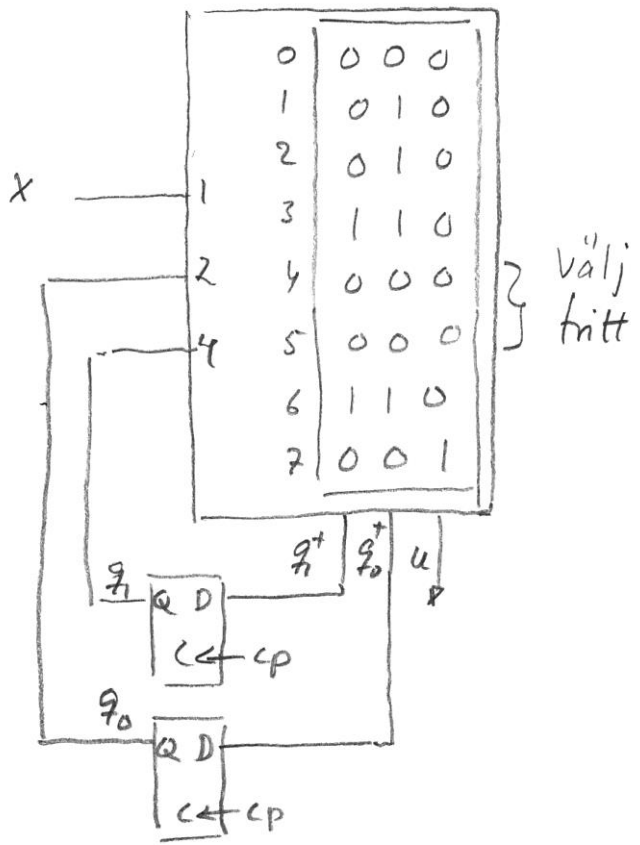




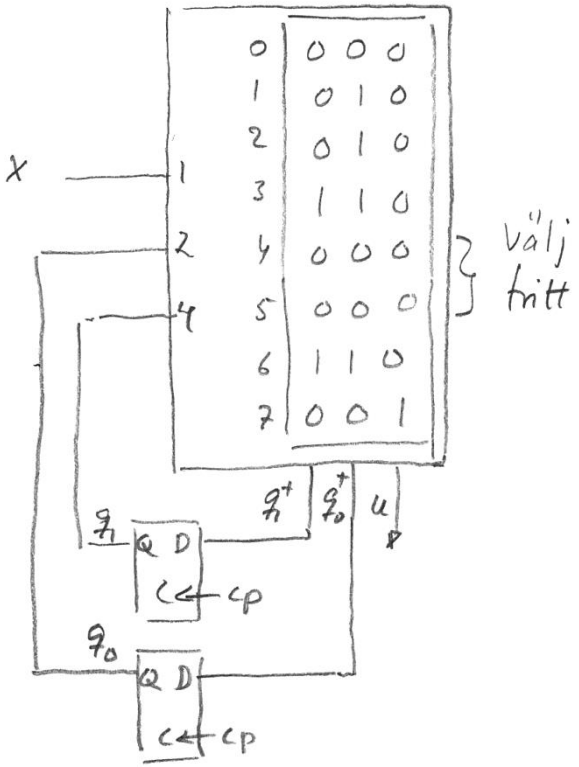
Detektion av var tredje etta i VHDL

	4 q_1	2 q_0	1 x	q_1^+	q_0^+	u
0	0	0	0	0	0	0
1	0	0	1	0	1	0
2	0	1	0	0	1	0
3	0	1	1	1	1	0
4	1	0	0	–	–	–
5	1	0	1	–	–	–
6	1	1	0	1	1	0
7	1	1	1	0	0	1

adress utsignal



Gränssnitt mot omgivningen



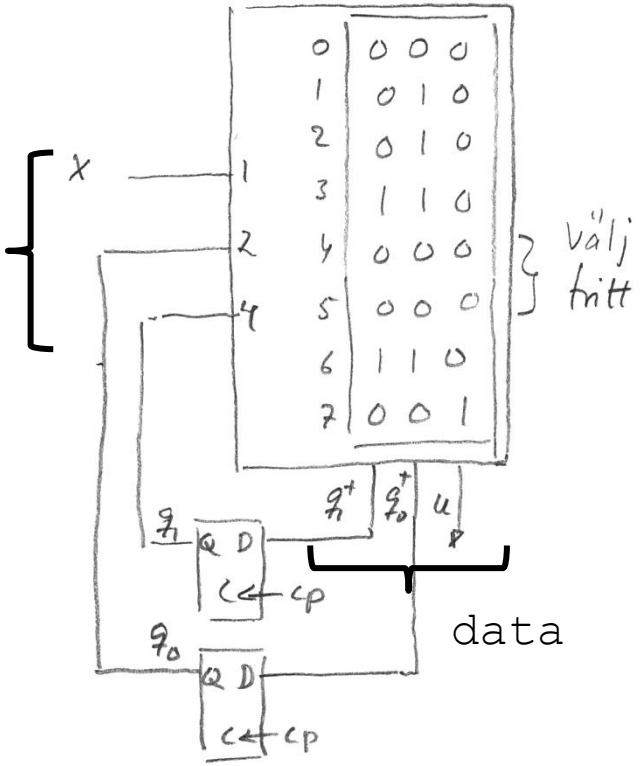

```
entity skrets is
  port( clk, x: in  std_logic;

                u: out std_logic);
end entity skrets;
```

Interna signaler

Ej kod

```
adress = (q1,q0,x)
data   = (q1_plus,q0_plus,u)
```

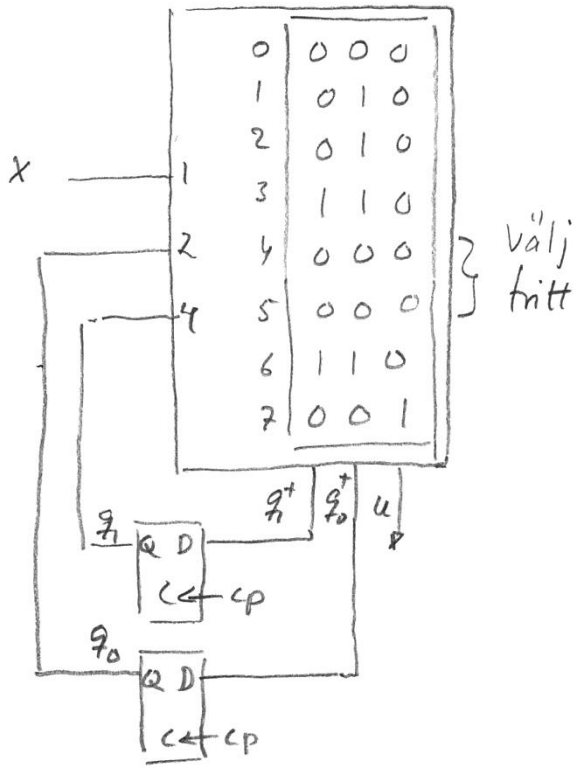
adress




```
architecture behavior of skrets is
  signal q0, q0_plus: std_logic;
  signal q1, q1_plus: std_logic;
  signal adress : std_logic_vector(2 downto 0);
  signal data   : std_logic_vector(2 downto 0);
begin
  -- beskrivning av kretsens beteende
  -- se kommande 2 oh:ar

end behavior;
```

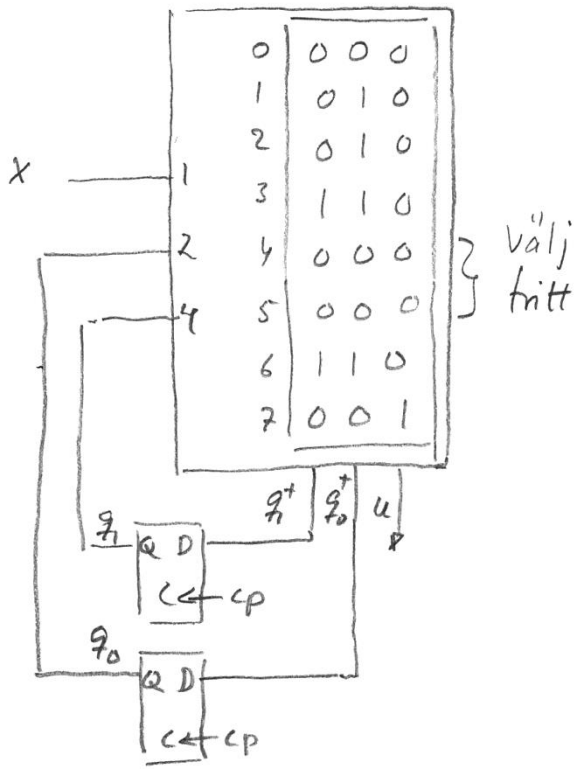
Vippor

```
-- vippor
process(clk)
begin
   if rising_edge(clk) then
        q0 <= q0_plus;
        q1 <= q1_plus;
     end if;
end process;
```



Minnet

```
-- ROM
adress <= q1 & q0 & x; -- concatenering
with adress select
    data <= "000" when "000",
            "010" when "001",
            "010" when "010",
            "110" when "011",
            "000" when "100",
            "000" when "101",
            "110" when "110",
            "001" when "111",
            "---" when others;
-- nästa tillstånd
q1_plus <= data(2); -- indexera vektor
q0_plus <= data(1);

-- utsignal
u <= data(0);
```



VHDL – bra rutiner

För att undvika problem så rekommenderas i dagsläget:

- Generera bara D-vippor i processer

```
process(clk)
…
q1 <= q1plus;
…
end process;
```

- Skapa all logik utanför processer

```
q1plus <= xin and (q2 or q1);
yout <= q1 or q2;
```

VHDL – bra rutiner

- `x <= a and b;`
  - Betyder att en AND-grind kopplas in mellan `a, b` och `x`
  - Endast en tilldelning på `x` tillåten.
- `x <= a and b;`
  `a <= '1';`

Ordningen på satserna oviktig utanför process-satsen

"Programmera" aldrig i VHDL!

Tänk hårdvara => översätt till VHDL

Programmera kretsen

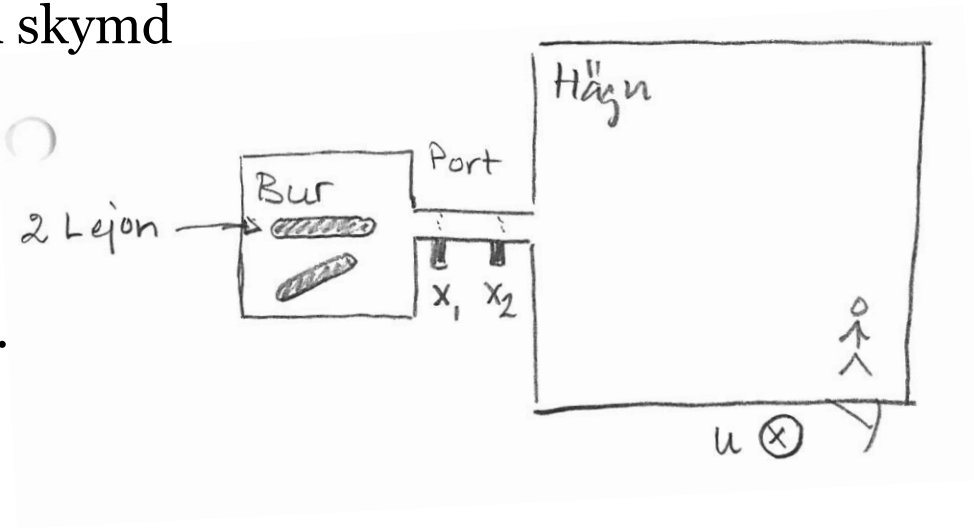
- På labben kommer ni att använda en specifik dator för att programmera kretsarna.
- **Ta hjälp av en labassistent första gången och se till att kretsen är rättvänd i sockeln varje gång!**
  - Annars förstörs kretsen och ni bränner er när ni ska plocka ut den

Dagens föreläsning

- Initiering av starttillstånd
- Programmerbar logik
- Syntesflödet
- Hårdvarubeskrivande språk VHDL
- Från problemformulering till tillståndsdiagram

Lejonburen

- Lampa skall lysa när hägnet är tomt.
- Fotoceller: x_i = { 1 fotocellen skymd; 0 annars }
- Lampa: u = { 0 släkt; 1 tänd }
- Vid start är båda lejonen i buren.
- Lejonen:
  a) Max ett lejon i porten
  b) Kan ej vända/backa i porten.
  c) Är längre än avståndet mellan x_1 och x_2.
  d) Rör sig långsamt i förhållande till klockfrekvensen.



Senarier

- Porten tom: $x = (x_1, x_2) = 00$
- Lejon i port:

  a) => ett lejon

  b) => rör sig genom porten

  Fall

  – Lejon ut ur bur:

  $\longrightarrow$

  x : 00, 01, 11, 01, 00 (x = 11, pga c))

  $\longleftarrow$

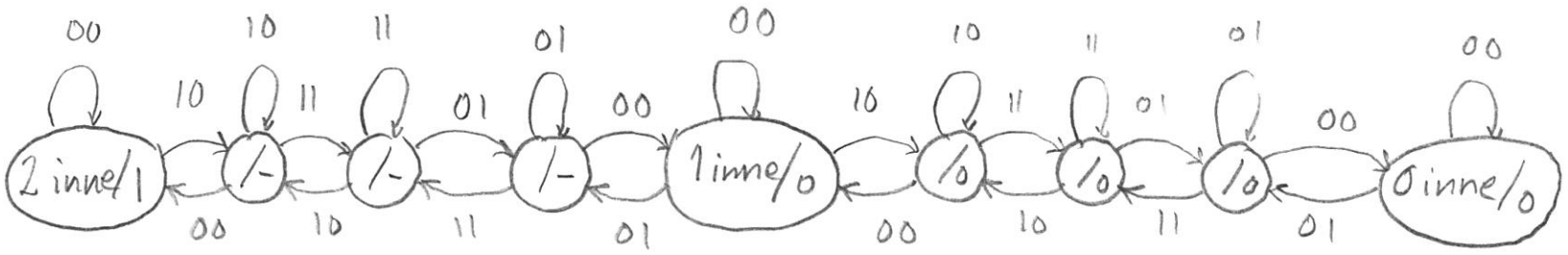
  – Lejon in i bur:

  d) => Varje insignalkombination upprepas

  => Vi kan vänta en klockpuls med att tända lampan, dvs Moore-krets är okej.

Enkel lösning

- Låt tillståndet "*i* inne" beteckna att *i* lejon är i buren.

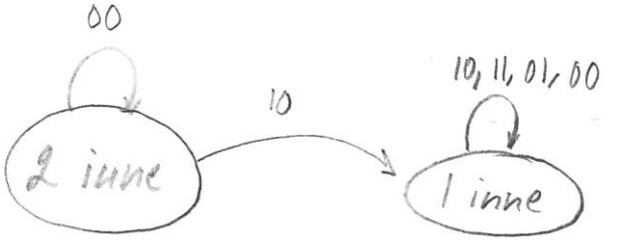


- Onödigt många tillstånd: 9 st
- Det finns algoritmer som minimerar antalet tillstånd.
- Hanterar även att lejon backar i porten

  => Onödigt komplicerat tillståndsdiagram

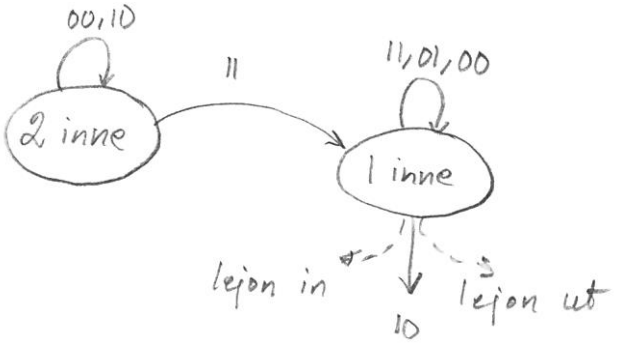
Lösning med få tillstånd

- Vi behöver ett tillstånd då lampan ska lysa (u = 1). I tillståndet “2 inne” lyser lampan i övriga tillstånd är den släkt.
- Detektera utpassage med 10



← alla insignalkombinationer => fast i tillståndet

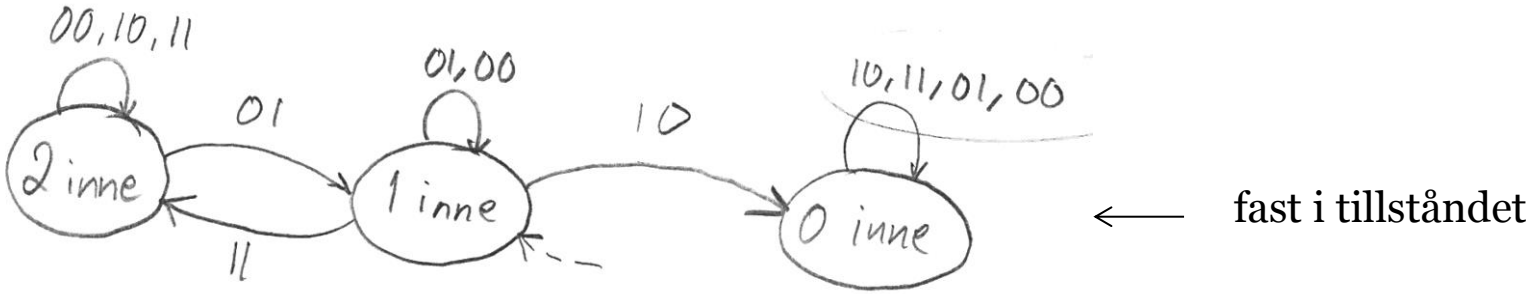
- Detektera utpassage med 11



← Går ej att skilja på fallen

Lejonbur fortsättning …

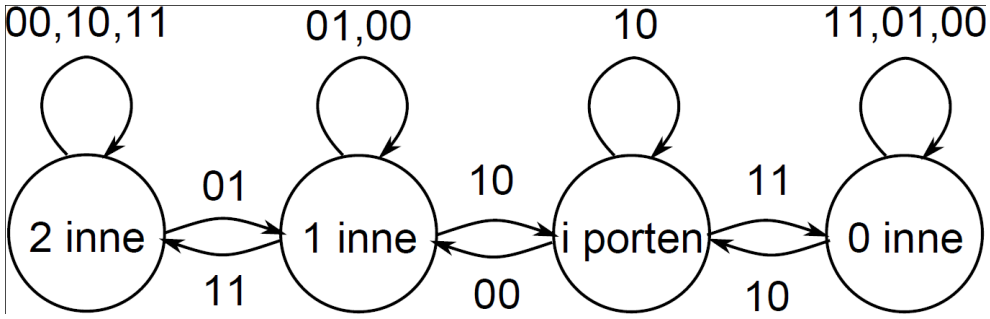
- Detektera utpassage med 01



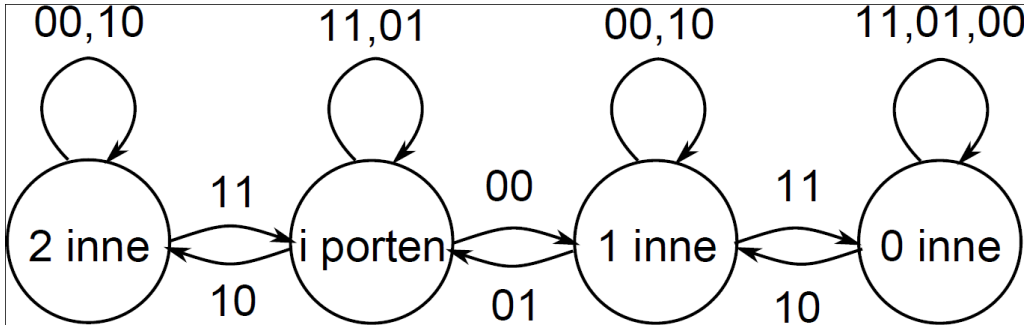
- 10, 11 kvar
  - 11 påträffas först vid inpassage
  - 10 påträffas först vid utpassage
- Vilka insignaler för streckad tillståndsövergång är tänkbar?
  - 01: funkar ej ty vi hamnar i “2 inne”
  - 11: funkar ej ty vi hamnar i “2 inne”
  - 10: funkar ej ty vi hamnar i “0 inne”
  - 00: OK, men nytt tillstånd måste införas: “i porten”

Tillståndsdiagram för lejonburen

- Tillståndsdiagram framtaget på föreläsningen:



- Alternativt tillståndsdiagram:



Digitalteknik

Mattias Krysander

www.liu.se

LINKÖPINGS UNIVERSITET