Razvoj digitalnih sistemov

Predstavitev strani in
izvajanja predmeta

Domača stran predmeta

http://rds.fe.uni-lj.si

Splošno	Predavanja	Laboratorijske vaje	Komentar

Razvoj digitalnih sistemov

Izpiti

Pisni			Ustni		
Datum	*Prostor*	*Ura*	*Datum*	*Prostor*	*Ura*
Izpitni roki so razpisani v sistemu e-študent.					

Morebitna vprašanja mi pošljite na elektronsko pošto (matej.mozek@fe.uni-lj.si**).**

Razvoj digitalnih sistemov

Uvod v logična vezja:
Spremenljivke, funkcije,
pravilnostne tabele, vrata, vezja

Spremenljivke in funkcije

- **Konjunkcija (AND)** $L(x_1, x_2) = x_1 \cdot x_2$
  - L='1' če (in samo če) sta x_1 in (AND) x_2 enaka '1'

"·" AND operator
"Λ" AND operator
"&" AND operator

Skrajšano: $x_1 \cdot x_2 = x_1 x_2$

- **Disjunkcija (OR)** $L(x_1, x_2) = x_1 + x_2$
  - L='1' če je ali x_1 ali x_2 ali oba

Simbol: "+" ali "V" OR operator

- **Negacija (NOT)** $\bar{x}$
  - ='1' če je x='0' in L='0' če je x='1'

Oznake: $\bar{x}$, x', NOT x

Vhodni vektor

- Več spremenljivk tvori vhodni vektor

  Tri spremenljivke (x_1 , x_2 , x_3)

0	$w_0 = 0\ 0\ 0$
1	$w_1 = 0\ 0\ 1$
2	$w_2 = 0\ 1\ 0$
3	$w_3 = 0\ 1\ 1$
4	$w_4 = 1\ 0\ 0$
5	$w_5 = 1\ 0\ 1$
6	$w_6 = 1\ 1\ 0$
7	$w_7 = 1\ 1\ 1$

Negacija funkcije

- Če je funkcija definirana kot
  - $f(x_1, x_2) = x_1 + x_2$
- Potem je komplement (negacija) funkcije f:
  - $\overline{f}(x_1, x_2) = \overline{x_1 + x_2} = (x_1 + x_2)'$
- Podobno, če
  - $f(x_1, x_2) = x_1 \cdot x_2$
- Potem je komplement funkcije f
  - $\overline{f}(x_1, x_2) = \overline{x_1 \cdot x_2} = (x_1 \cdot x_2)'$

Pravilnostna tabela

- Izhodna vrednost funkcije za vhodni vektor (vseh kombinacij vhodnih spremenljivk).

x_1	x_2	$x_1 \cdot x_2$
0	0	0
0	1	0
1	0	0
1	1	1

AND

x_1	x_2	$x_1 + x_2$
0	0	0
0	1	1
1	0	1
1	1	1

OR

x_1	x_1'
0	1
1	0

NOT

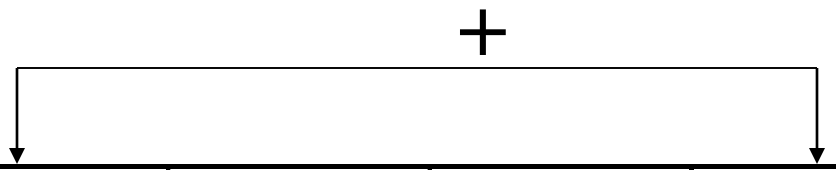
Pravilnostna tabela

- Pravilnostna tabela za AND in OR funkciji treh spremenljivk

x_1	x_2	x_3	$x_1 \cdot x_2 \cdot x_3$	$x_1 + x_2 + x_3$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Pravilnostna tabela

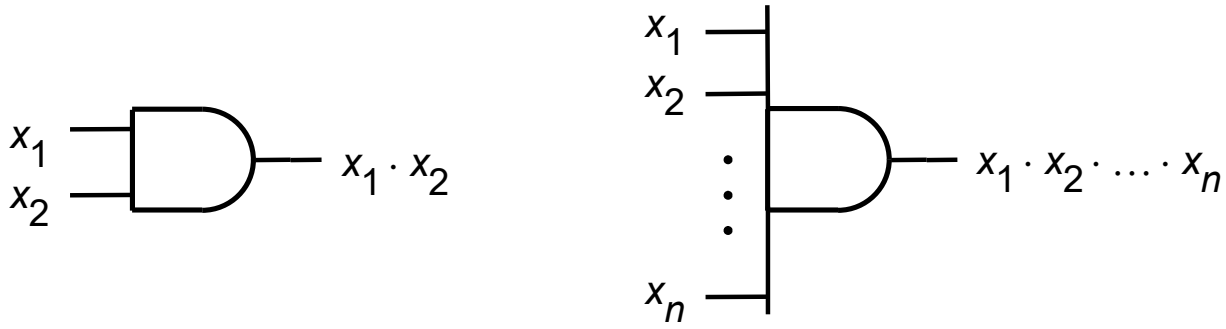
- Določite pravilnostno tabelo funkcije $L(x, y, z) = x + y \cdot z$



x	y	z	yz	x+yz
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

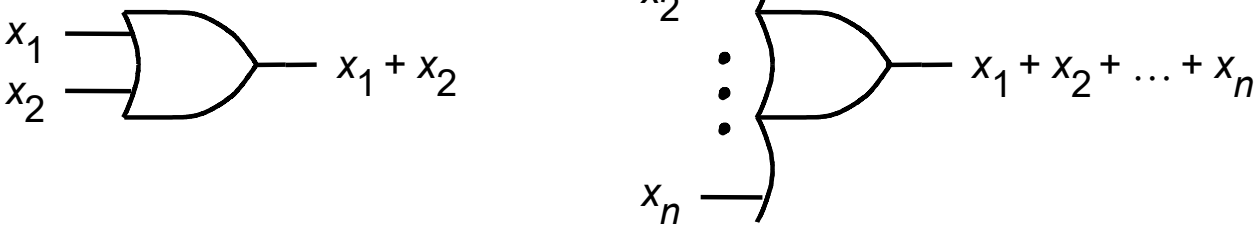
Logična vrata in vezja

- Osnovno logično operacijo (AND, OR, NOT) lahko izvedemo z elementom vezja, ki mu pravimo ***logična vrata***.
- Logična vrata imajo enega ali več vhodov in en izhod, ki je funkcija vhodov v vrata.

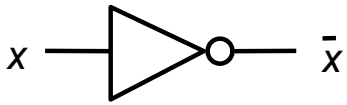


AND vrata

Logična vrata in vezja



OR vrata

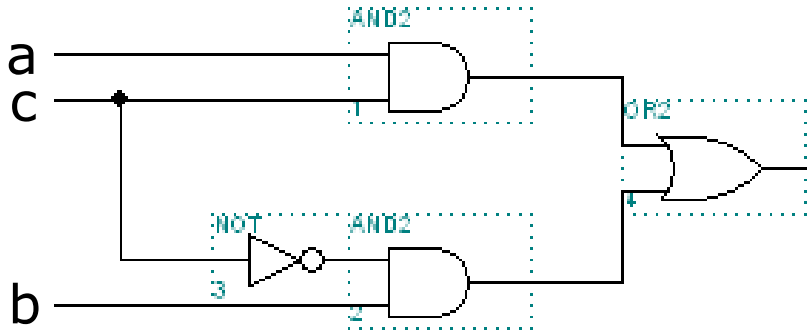


NOT vrata – inverter - negator

Logična vrata in vezja

- Določite pravilnostno tabelo in logično vezje podane funkcije:
  F(a, b, c) = a·c + b·c’

a	b	c	ac	bc'	ac+bc'
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	1	0	1



Razvoj digitalnih sistemov

Uvod v logična vezja:
Boole-ova algebra

Aksiomi Boole-ove algebre

- Boole-ova algebra temelji na nizu pravil, ki so izpeljana na osnovi predpostavk (aksiomov)

1a	$0 \cdot 0 = 0$	1b	$1 + 1 = 1$
2a	$1 \cdot 1 = 1$	2b	$0 + 0 = 0$
3a	$0 \cdot 1 = 1 \cdot 0 = 0$	3b	$1 + 0 = 0 + 1 = 1$
4a	Če $x=0$ potem je $x'=1$	4b	Če $x=1$ potem je $x'=0$

Teoremi z eno spremenljivko

Iz aksiomov so izvedena nekatera pravila za računanje z eno spremenljivko:

- Operacije s konstanto

5a	$x \cdot 0 = 0$	*5b*	$x + 1 = 1$

- Nevtralni element

6a	$x \cdot 1 = x$	*6b*	$x + 0 = x$

- Idempotenca

7a	$x \cdot x = x$	*7b*	$x + x = x$

- Komplementarnost

8a	$x \cdot x' = 0$	*8b*	$x + x' = 1$

- Dvojna negacija

9	$x'' = x$

Dualnost

- Aksiome in teoreme za eno spremenljivko lahko izrazimo v parih
  - f(a, b)=a + b dualna funkcija: f(a, b)=a · b
  - f(x)=x+0 dualna funkcija: f(x)=x·1

$$f(a,b,c) = f'(a',b',c')$$

- Dualnost resnične izjave je resnična izjava.

Teoremi dveh in več spremenljivk

- 10a. $x \cdot y = y \cdot x$ *Komutativnost*
- 10b. $x + y = y + x$

- 11a. $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ *Asociativnost*
- 11b. $x + (y + z) = (x + y) + z$

- 12a. $x \cdot (y + z) = x \cdot y + x \cdot z$ *Distributivnost*
- 12b. $x + y \cdot z = (x + y) \cdot (x + z)$

- 13a. $x + x \cdot y = x$ *Vsebovanje*
- 13b. $x \cdot (x + y) = x$

Teoremi dveh in več spremenljivk

- 14a. $x \cdot y + x \cdot y' = x$ *Kombinacija*
- 14b. $(x+y) \cdot (x+y') = x$ *(sosednost)*

- 15a. $(x \cdot y)' = x' + y'$ *DeMorganov*
- 15b. $(x+y)' = x' \cdot y'$ *teorem*

- 16a. $x + x' \cdot y = x + y$
- 16b. $x \cdot (x' + y) = x \cdot y$

Dokaz $x+x'\cdot y=x+y$

- Dokažite $x+x'\cdot y=x+y$ z uporabo popolne indukcije in lastnosti Boole-ove algebre

x	y	$x'y$	$x+x'y$	$x+y$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

ekvivalentna

$$x+x'\cdot y=(x+y)\cdot(x+x') \qquad (12b)$$

$$x+x'\cdot y=(x+y)\cdot 1 \qquad (8b)$$

$$x+x'\cdot y=(x+y) \qquad (6a)$$

Dokaz $(x \cdot y)' = x' + y'$

- Uporabite popolno indukcijo in dokažite $(x \cdot y)' = x' + y'$

x	y	xy	$(xy)'$	x'	y'	$x'+y'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

ekvivalentna

Algebrajski dokaz

- Dokažite: $(X+A)\cdot(X'+A)\cdot(A+C)\cdot(A+D)\cdot X = A\cdot X$

$(X+A)(X'+A)(A+C)(A+D)X$

$(X+A)(X'+A)(A+CD)X$ (*12b*)-distributivnost $x+y\cdot z=(x+y)\cdot(x+z)$

$(X+A)(X'+A)(A+CD)X$

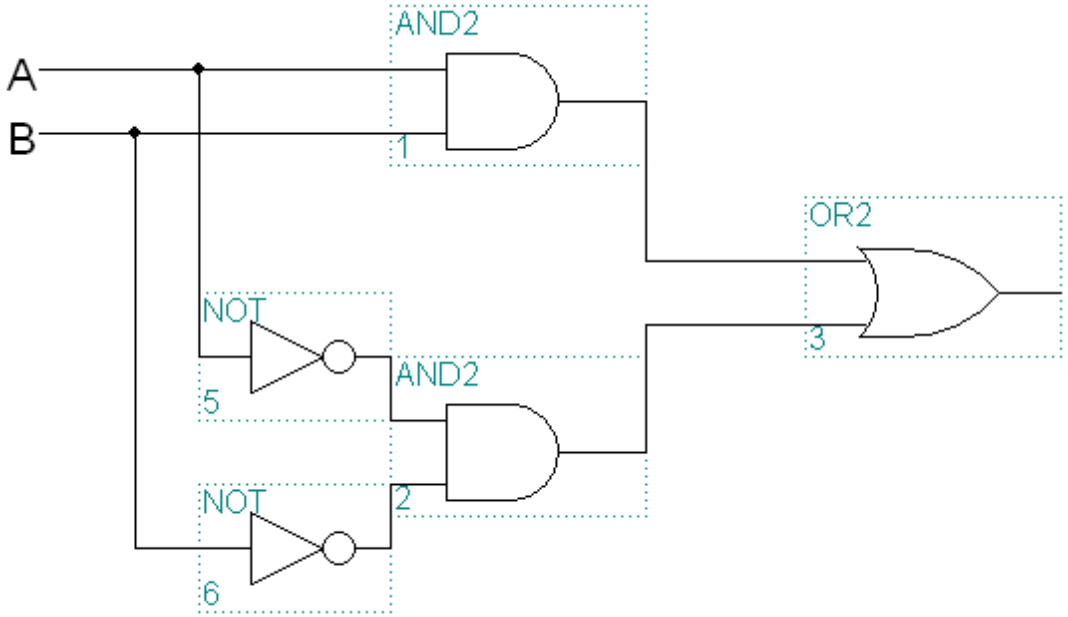
$(A)(A+CD)X$ (*14b*) - sosednost

$(A)(A+CD)X$

AX (*13b*) - vsebovanje

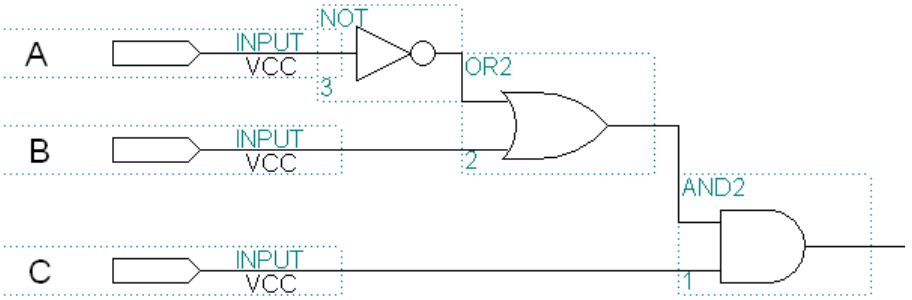
Prioriteta logičnih operatorjev

- Če v logičnem izrazu **ni oklepajev**, potem se logične operacije izvajajo v zaporedju
  - **NOT, AND, OR**
- Določite zaporedje operatorjev v izrazu: A·B+A'·B'

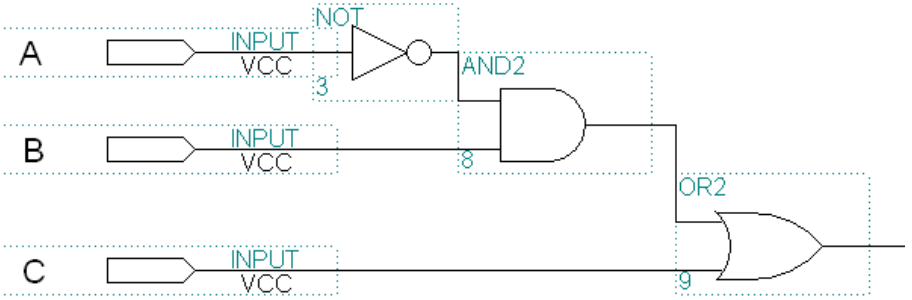


Prioriteta logičnih operatorjev

- Narišite vezje ki realizira naslednji logični funkciji:
  - f(a,b,c)=(a’+b)·c



  - f(a,b,c)=a’·b+c

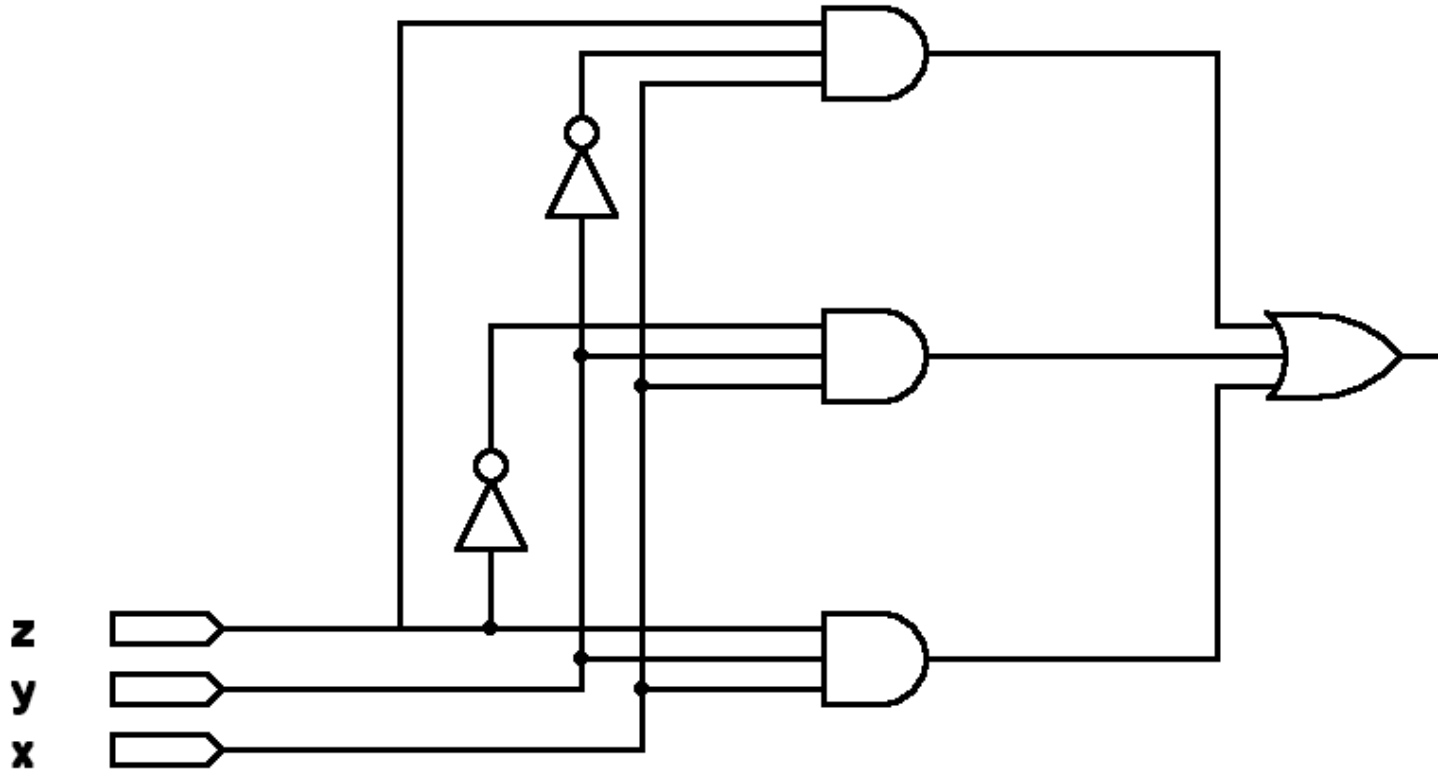


Primer načrtovanja logične funkcije

- Izhod vezja *f* naj bo '1', ko je *x*='1' **in** ko sta **ali** *y* **ali** *z* (**ali** oba) enaka '1'
  - Tri možne kombinacije
    - x=1, y=0, z=1 → x·y'·z
    - x=1, y=1, z=0 → x·y·z'
    - x=1, y=1, z=1 → x·y·z
- Funkcijo lahko zapišemo kot
  - f(x,y,z)=x·y'·z+x·y·z'+x·y·z

Primer načrtovanja logične funkcije

$$f(x,y,z)=x\cdot y'\cdot z+x\cdot y\cdot z'+x\cdot y\cdot z$$



Primer načrtovanja logične funkcije

x	*y*	*z*	*f*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

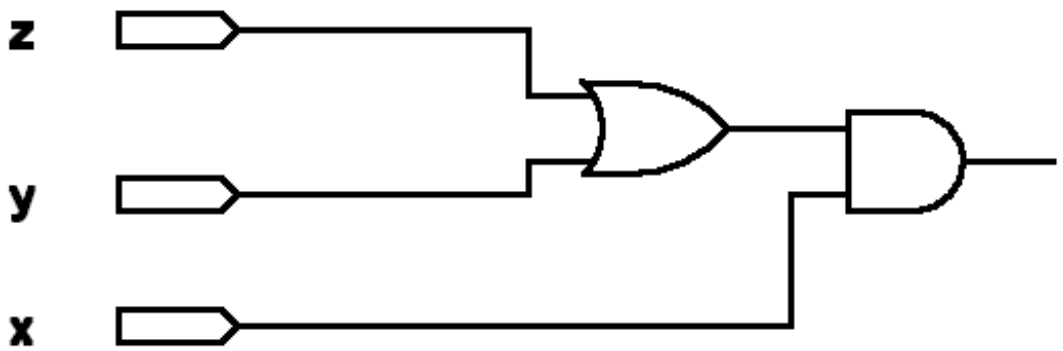
$$f(x, y, z) = x \cdot y' \cdot z + x \cdot y \cdot z' + x \cdot y \cdot z$$

$$= xy'z + xy(z' + z)$$

$$= xy'z + xy \qquad (14a)$$

$$= x(y'z + y) \qquad (12a)$$

$$= x(y + z) \qquad (16a)$$



Razvoj digitalnih sistemov

Uvod v logična vezja:

Sinteza logičnih funkcij z uporabo AND, OR in NOT vrat

Zapisi logične funkcije

- Normalna oblika (kanonska)
  - popolno normalna oblika, dvonivojska,
  - normalna oblika,
  - minimalno normalna oblika.
- Nenormalna oblika (večnivojska)
  - $g(x, y, z, w) = ((x'y') + z) \cdot x + y \cdot w'$

Elementarne logične funkcije dveh spremenljivk

2 spremenljivki ➔ 16 elementarnih funkcij

- 4 skupine funkcij:
  - Enostavne logične funkcije (iz Boole-ove algebre)
    - konstanta '0', '1'
    - original in negacija x_1, x_2
    - AND, OR operatorja
  - NOR (Pierce ↓), NAND (Sheffer ↑)
  - XOR, XNOR (oz. EQU)
  - implikacija in negacija implikacije x_1, x_2 (redko)

Elementarne logične funkcije dveh spremenljivk

x1	x2	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

F0 konstanta '0'

F1 NOR (Pierce)

F2 negacija implikacije

F3 negacija spremenljivke

F4 negacija implikacije

F5 negacija spremenljivke

F6 XOR (seštevanje po modulu 2)

F7 NAND (Sheffer)

F15 konstanta '1'

F14 OR (disjunkcija)

F13 implikacija x2→x1

F12 spremenljivka x1

F11 implikacija x1→x2

F10 spremenljivka x2

F9 XNOR (ekvivalenca)

F8 AND (konjunkcija)

Zapisi logične funkcije

- Popolna normalna oblika
  - **P**opolna **D**isjunktivna **N**ormalna **O**blika (**PDNO**)
    (1. nivo AND, 2. nivo: *ena* OR vrata)
  - **P**opolna **K**onjuktivna **N**ormalna **O**blika (**PKNO**)
    (1. nivo OR, 2. nivo: *ena* AND vrata)

Mintermi

- Funkcijo *n* spremenljivk *f(a, b, c,…)* lahko zapišemo s pomočjo mintermov
  - f(a, b, c) - pravilni primeri mintermov:
    a·b·c, a’·b·c, a·b·c’
  - f(a, b, c) - nepravilni primeri mintermov: a·b, c’, a’·c

– Funkcija *n* spremenljivk ima 2^n možnih mintermov.

Mintermi

Minterm	*x*	*y*	*z*	
m_0	0	0	0	$x' \cdot y' \cdot z'$
m_1	0	0	1	$x' \cdot y' \cdot z$
m_2	0	1	0	$x' \cdot y \cdot z'$
m_3	0	1	1	$x' \cdot y \cdot z$
m_4	1	0	0	$x \cdot y' \cdot z'$
m_5	1	0	1	$x \cdot y' \cdot z$
m_6	1	1	0	$x \cdot y \cdot z'$
m_7	1	1	1	$x \cdot y \cdot z$

Zapis mintermov

- Enačbo funkcije zapišemo z mintermi:

$f(a, b, c) = m_0 + m_1 + m_2 + m_4$

$f(a, b, c) = a'b'c' + a'b'c + a'bc' + ab'c'$

$\underbrace{000}_{0} \quad \underbrace{001}_{1} \quad \underbrace{010}_{2} \quad \underbrace{100}_{4}$

a	*b*	*c*	*f*
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$f^3 = V(0, 1, 2, 4)$

Primeri zapisa z mintermi

- Zapišite podano funkcijo z mintermi:
  - f(a, b, c)= abc + a’bc + abc’ + a’b’c

- Zapišite podano funkcijo v PDNO:
  - $f^3 = \bigvee(1,5,6,7)$

Logična sinteza funkcij

- Če je možno funkcijo *f* sintetizirati ob upoštevanju pravilnostne tabele, ki ima vrstice *f*='1', potem je možno isto funkcijo *f* sintetizirati iz vrstic pravilnostne tabele v katerih je *f*='0'.
- Tak pristop uporablja komplemente (negacije) mintermov – ***maksterme***.

Makstermi

Maksterm	*x*	*y*	*z*	
M_7	0	0	0	x+y+z
M_6	0	0	1	x+y+z′
M_5	0	1	0	x+y′+z
M_4	0	1	1	x+y′+z′
M_3	1	0	0	x′+y+z
M_2	1	0	1	x′+y+z′
M_1	1	1	0	x′+y′+z
M_0	1	1	1	x′+y′+z′

Zapis makstermov

- Enačbo funkcije lahko zapišemo v smislu notacije z makstermi (M-notacija)

$f(a, b, c)=M_0 \cdot M_1 \cdot M_2 \cdot M_4$

$f(a, b, c)=(a+b'+c')(a'+b+c')(a'+b'+c)(a'+b'+c')$

$$\underbrace{0\,1\,1}_{4} \quad \underbrace{1\,0\,1}_{2} \quad \underbrace{1\,1\,0}_{1} \quad \underbrace{1\,1\,1}_{0}$$

$f^3=\&(0, 1, 2, 4)$

#	a	b	c	F
7	0	0	0	1
6	0	0	1	1
5	0	1	0	1
4	0	1	1	0
3	1	0	0	1
2	1	0	1	0
1	1	1	0	0
0	1	1	1	0

Primeri zapisa z makstermi

- Zapišite podano funkcijo v notaciji z makstermi:
  - $f(a, b, c)=(a+b+c)(a'+b+c)(a+b+c')(a'+b'+c)$

$$f^3=\&(7, 3, 6, 1)$$

- Izpišite podano funkcijo v PKNO
  - f(a, b, c)= &(1,5,6,7)

$$f(a, b, c)=(a'+b'+c)(a+b'+c)(a'+b'+c)(a+b+c)$$

Pretvorba med zapisi funkcij

Dana oblika	**Želena oblika**			
	$f=\Sigma m$	$f=\Pi M$	$f'=\Sigma m$	$f'=\Pi M$
$f=\Sigma m$ (0,2,5,7)		Številke, ki *niso* na seznamu mintermov. (1,3,4,6)		Številke, ki *so* na seznamu mintermov. (0,2,5,7)
$f=\Pi M$ (1,3,4,6)	Številke, ki *niso* na seznamu makstermov (0,2,5,7)		Številke, ki *so* na seznamu makstermov (1,3,4,6)	Številke, ki *niso* na seznamu makstermov (0,2,5,7)

Pretvorbe PDNO ⇐⇒ PKNO

- Funkcijo v **PDNO**: $f(x_1, x_2, x_3) = f^3 = V(3, 4, 6, 7)$ pretvorite v **PKNO**:
  - **Manjkajoče** minterme ➔ maksterme
  - m_i ➔ M_j (enačba) $j = 2^n\text{-}1\text{-}i = 7\text{-}i$ (za 3 spremenljivke)

$$f^3 = \&(7, 6, 5, 2)$$

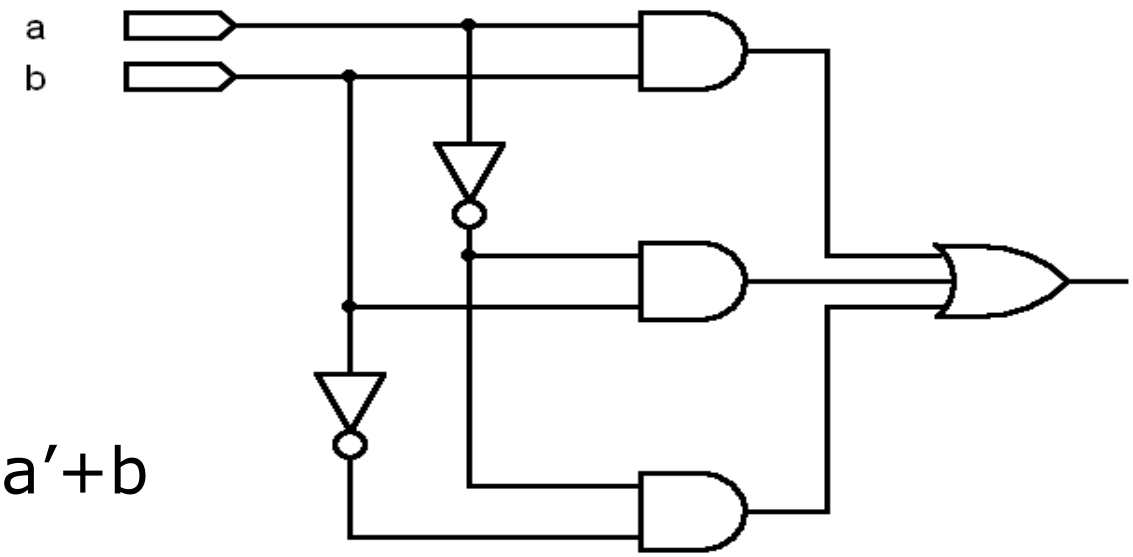
- Funkcijo v **PKNO**: $f(x_1, x_2, x_3) = f^3 = \&(7, 6, 5, 2)$ pretvorite v **PDNO**:
  - **Manjkajoče** maksterme ➔ minterme
  - m_i ➔ M_j (enačba) $i = 2^n\text{-}1\text{-}j = 7\text{-}j$ (za 3 spremenljivke)

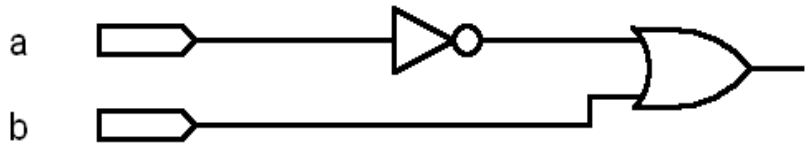
$$f^3 = V(3, 4, 6, 7)$$

DNO in minimalnost funkcije

a	*b*	*f*
0	0	1
0	1	1
1	0	0
1	1	1

f(a,b)=a’b’+a’b+ab=a’+b





f(a,b)=a’+b

Razvoj digitalnih sistemov

Uvod v logična vezja:
CAD orodja in VHDL

Uvod v CAD orodja

- CAD sistem navadno vključuje naslednja orodja za načrtovanje
  - Vnos načrtovanih funkcij
    (Design entry)
  - Sinteza in optimizacija načrtovanih funkcij
    (Synthesis and optimization)
  - Simulacija načrtovanih funkcij
    (Simulation)
  - Fizična izvedba in načrtovanje
    (Physical design and implementation)

Vnos načrtovanih funkcij

- Postopek vnosa opisa načrtovanih funkcij v CAD sistem
  (*design entry)*
- Tipične metode za vnos načrtovanih funkcij
  - Pravilnostne tabele
    (truth tables)
  - Shematski vnos funkcije
    (schematic capture)
  - Jezik strojnega opisa vezja
    (Hardware Description Language)

Jeziki strojnega opisa vezja

- Razširjeni HDL jeziki
  - VHDL (VHSIC Hardware Description Language, kjer je VHSIC: very-high-speed integrated circuit)
  - Verilog
  - Več drugih (lastni proizvajalcu)
- VHDL in Verilog sta standardizirana

Sinteza

- ***Prevajanje***
  - (*compiling*)
- **Optimiranje** glede na hitrost in/ali velikost – logična optimizacija

  (logic synthesis ali *logic optimization)*
- **tehnološka postavitev**

  (technology mapping)
- **sinteza postavitve**

  (layout synthesis)

Simulacija

- **Funkcionalna simulacija**

  (*functional simulation)*
  praviloma v obliki časovnega diagrama)

  Uporabnik sam preveri dobljene izhode vezja s svojimi pričakovanimi izhodi.

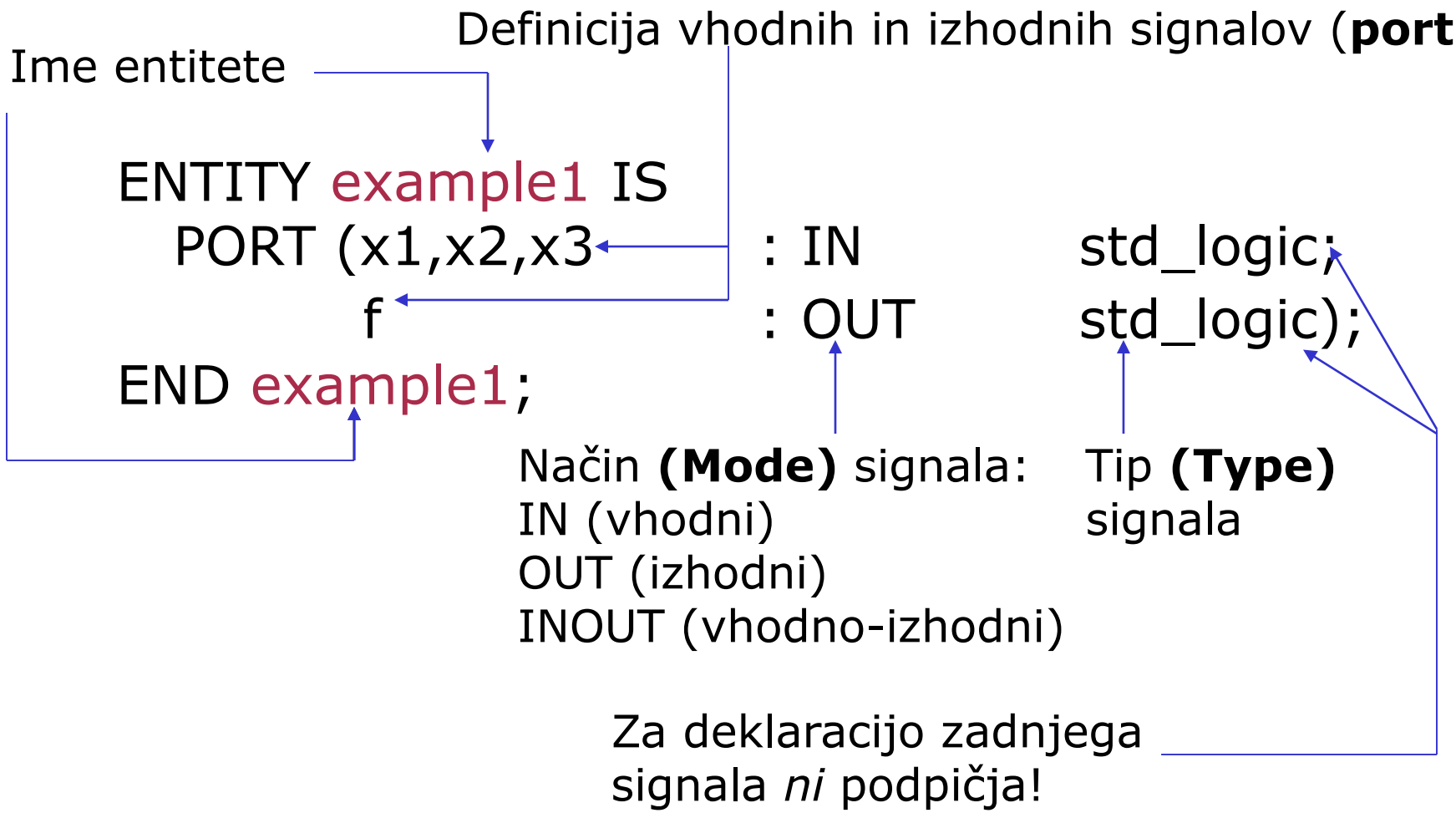
- Simulatorji navadno privzamejo, da je časovna zakasnitev širjenja preko več logičnih vrat zanemarljiva.

Uvod v VHDL

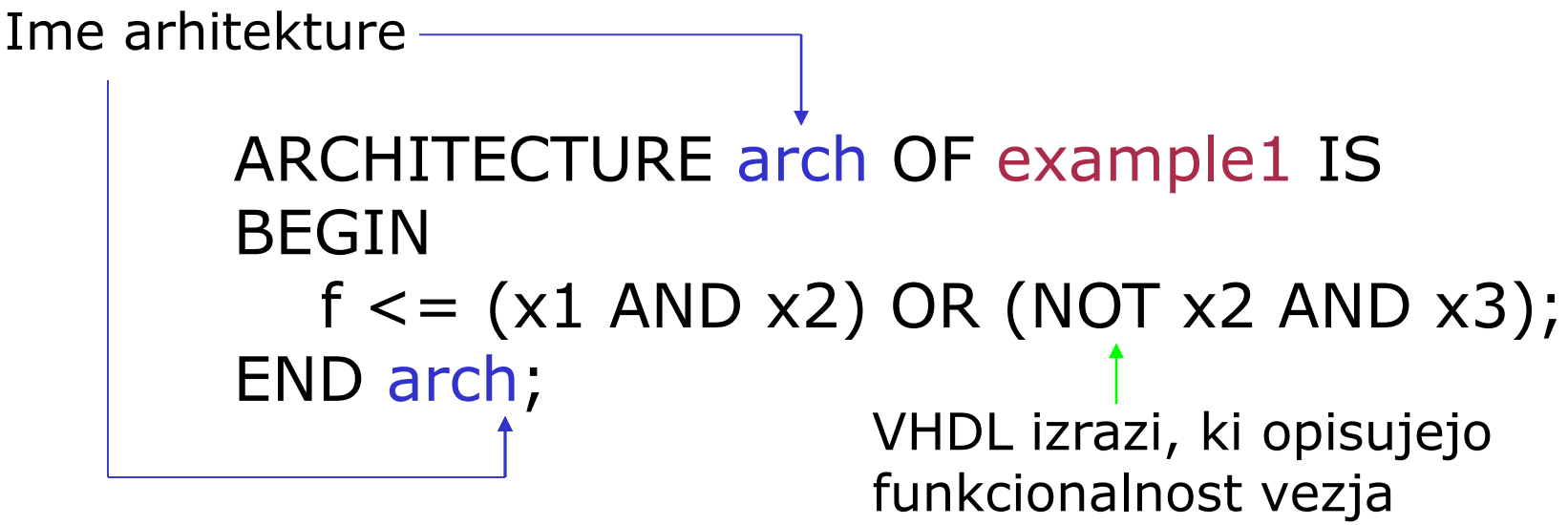
- Predstavitev logičnih signalov v VHDL
  - Logični signali tipa **std_logic** lahko zavzamejo **več** vrednosti:
    'U', 'X', '0', '1', 'Z', 'W', 'H', 'L', in '–'

•'U': Neinicializiran.
Signal še ni bil postavljen na nobeno vrednost (simulacija).
•'X': Neznan.
Vrednosti/rezultata ni bilo mogoče določiti (simulacija).
•'Z': Visoka impedanca (High Z)
•'W': Weak driver (za razliko od strong driver, ki da 0 ali 1)
ni mogoče določiti ali je 0 oz. 1.
•'L': Weak low signal, katerega vrednost naj bi bila 0
•'H': Weak high signal, katerega vrednost naj bi bila 1
•'-': Poljubna vrednost/Redundanca
(don't care).

Pisanje enostavne VHDL kode



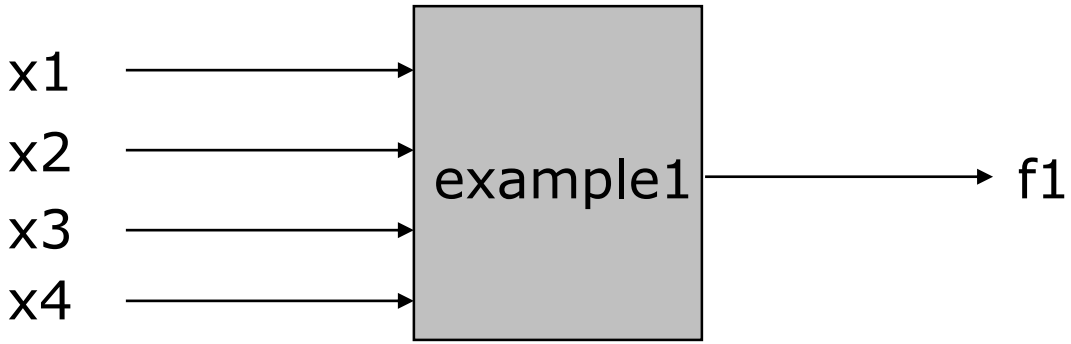
Pisanje enostavne VHDL kode



Pisanje enostavne VHDL kode

interni signal

```
ENTITY example1 IS
  PORT (  x1, x2, x3          : IN          std_logic;
          x4                  : INOUT       std_logic;
          f1                  : OUT         std_logic );
END example2;
ARCHITECTURE arch OF example1 IS
SIGNAL x5 : std_logic := '0';
BEGIN
          x4 <= x1 AND x2;
          x5 <= NOT x2 AND x3;
          f1 <= x4 OR x5;
END arch;
```

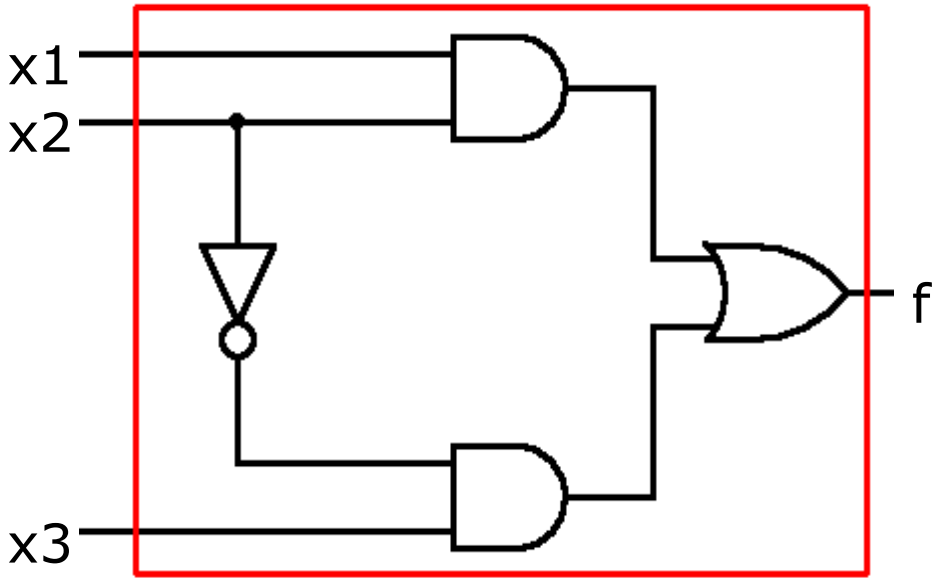


Boole-ovi operatorji v VHDL

- Nekaj logičnih operatorjev v VHDL:
  - **AND** logična konjunkcija
  - **OR** logična disjunkcija
  - **NOT** logična negacija
  - **NAND**, **NOR**, **XOR**, **XNOR**
- Operator prireditve **<=**
- VHDL NIMA prioritete logičnih operatorjev
- *Enostavna logična izjava*
  (*simple assignment statement*)

Enostaven zgled VHDL kode

- Izbiralnik 2/1




```
ENTITY example1 IS
PORT (x1,x2,x3 : IN std_logic;
        f          : OUT std_logic
        );
END example1;
```

```
ARCHITECTURE arch OF example1 IS
BEGIN
   f <= (x1 AND x2) OR (NOT x2 AND x3);
END arch;
```

Enostaven zgled VHDL kode

- Napišite VHDL kodo vezja 3-bitne funkcije večine *(majority function)*
- Funkcija večine postavi izhod f='1' ko je večina od vhodov (x1,x2,x3) enaka '1'
- Načrtajte konstrukt entitete in arhitekturni konstrukt.
  - Konstrukt entitete imenujte ***Majority***,
  - Arhitekturni konstrukt imenujte ***MajorityFunc***

Razvoj digitalnih sistemov

Optimiranje logičnih funkcij:
Veitch-ev, Karnaugh-jev diagram
in MDNO
Vezja z več izhodi

Diagrami za minimizacijo funkcij

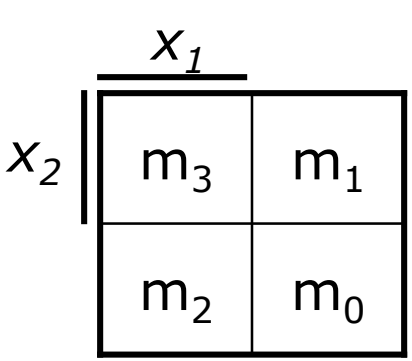
- Najti izražavo funkcije z minimalnim številom členov (mintermov oz. makstermov).
- Minimalna izražava (MDNO, MKNO) temelji na lastnosti **sosednosti**
  (14a ➔ MDNO, 14b ➔ MKNO)
  - f(x,y,z) = xyz+xyz′ = xy(z+z′) = xy(1) = xy
- Sistematična pot določitve te funkcije
  - ***Veitch-ev diagram*** (V-diagram)
  - ***Karnaugh-jev diagram*** (K-diagram)

Diagrami za minimizacijo funkcij

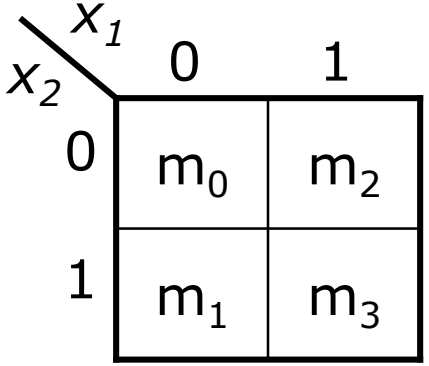
- Pravilnostna tabela za 2 spremenljivki in ustrezna diagrama:

x_1	x_2	f
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

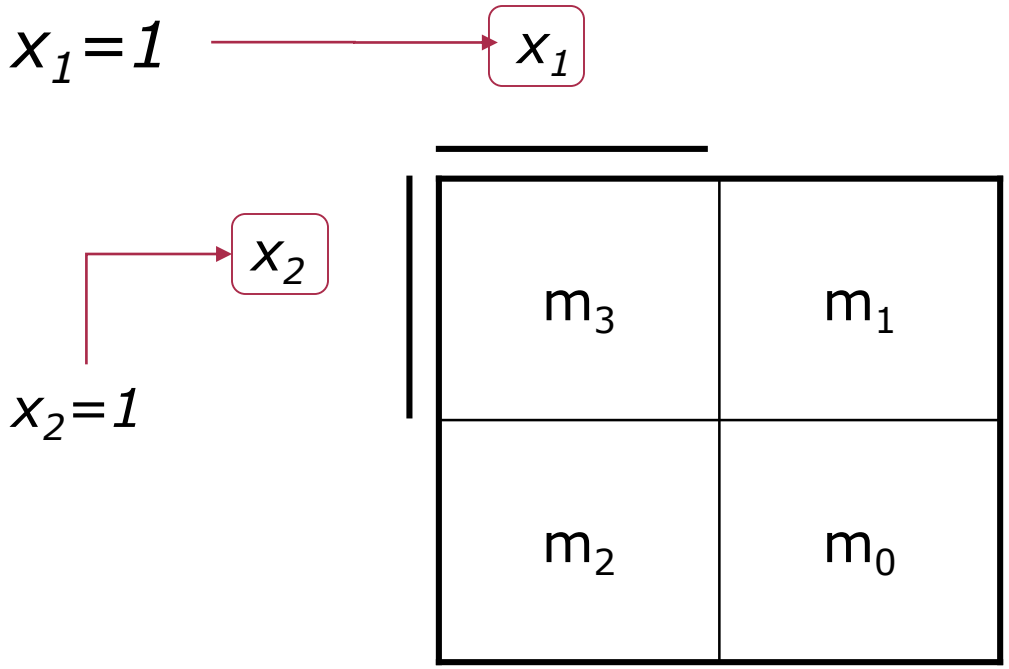
Veitch



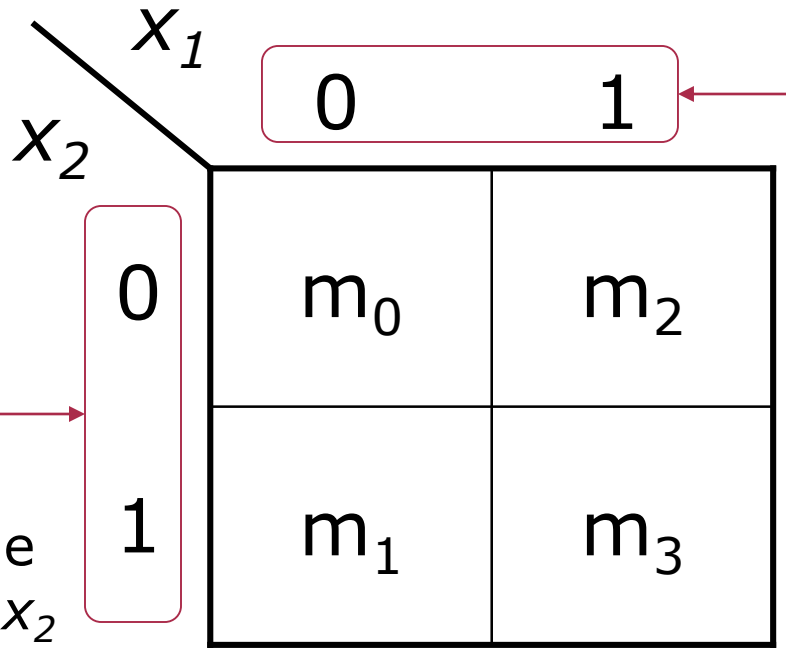
Karnaugh



Veitch-ev diagram



Karnaugh-jev diagram

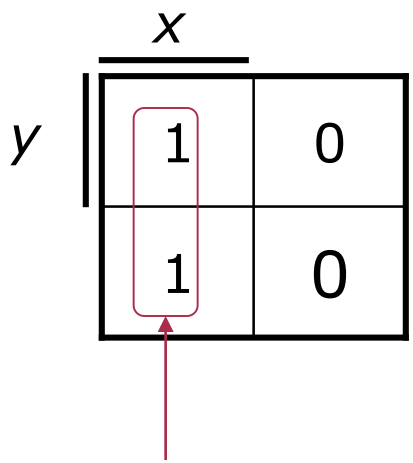


Vrednosti prve spremenljivke x_1 sta zapisani preko vrha diagrama

Vrednosti druge spremenljivke x_2 sta zapisani preko leve strani diagrama

Združevanje v V-diagramu

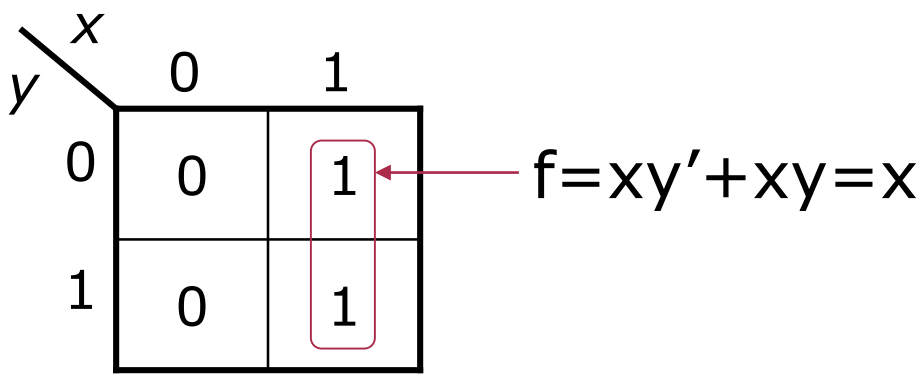
x	y	f
0	0	0
0	1	0
1	0	1
1	1	1



$f = xy' + xy = x(y' + y) = x$

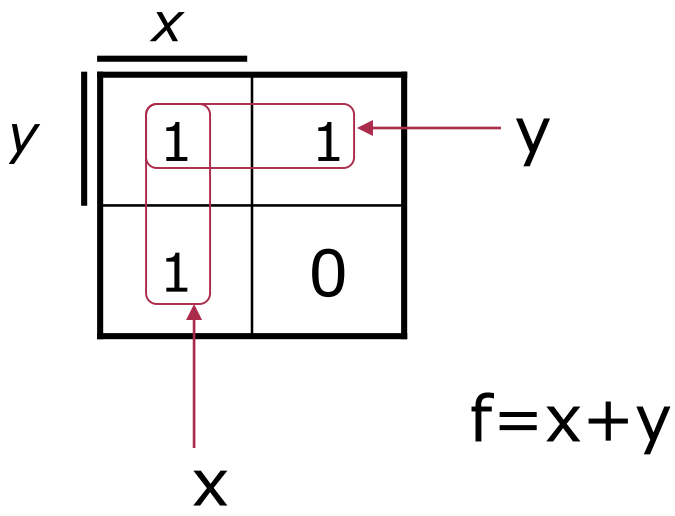
Združevanje v K-diagramu

x	y	f
0	0	0
0	1	0
1	0	1
1	1	1



Primer združevanja

x	y	f
0	0	0
0	1	1
1	0	1
1	1	1



Primer združevanja v K-diagramu

- Narišite Karnaugh-jev diagram za spodnjo funkcijo in izrazite minimalno obliko logične funkcije.
- Prikažite postopek združevanja v Karnaugh-jevem diagramu

x	y	f
0	0	1
0	1	1
1	0	1
1	1	0

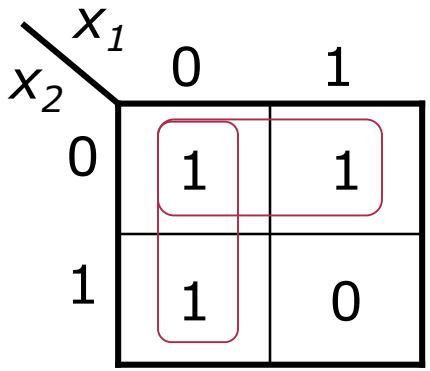
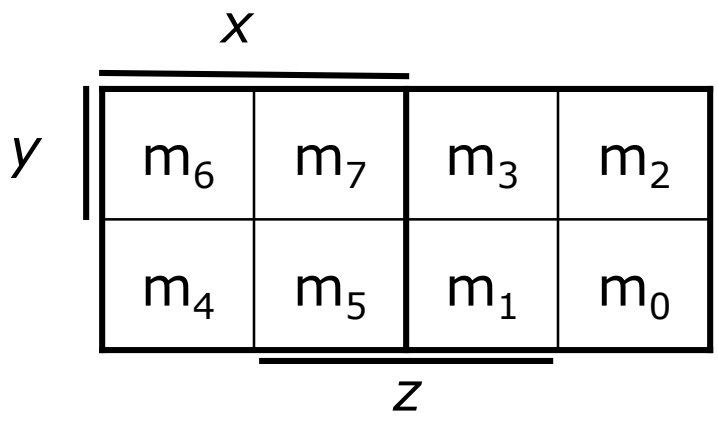
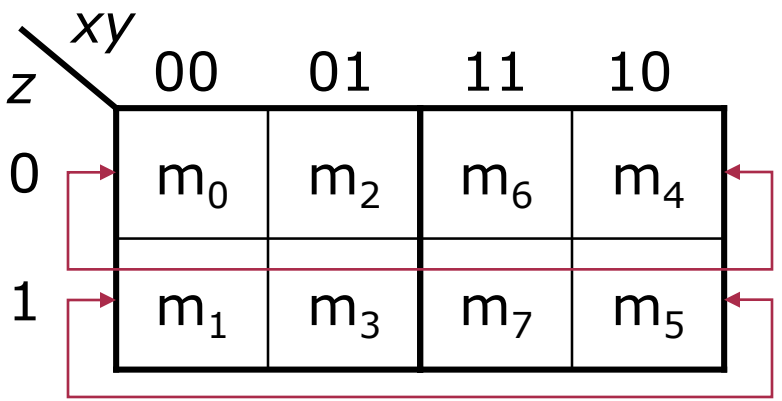


Diagrama treh spremenljivk

x	*y*	*z*	*minterm*
0	0	0	$m_0 = x'y'z'$
0	0	1	$m_1 = x'y'z$
0	1	0	$m_2 = x'yz'$
0	1	1	$m_3 = x'yz$
1	0	0	$m_4 = xy'z'$
1	0	1	$m_5 = xy'z$
1	1	0	$m_6 = xyz'$
1	1	1	$m_7 = xyz$



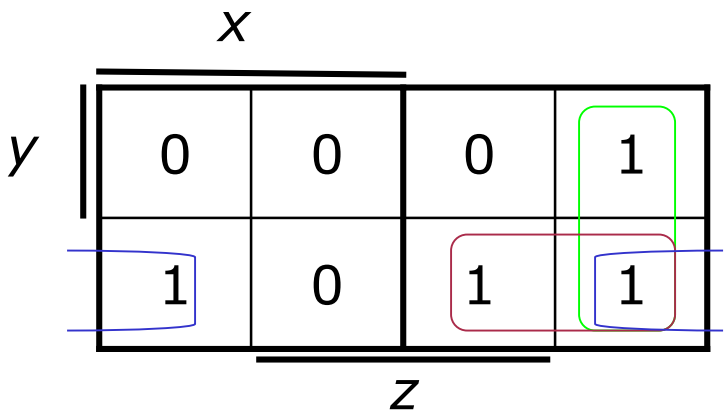


Končne celice so ‘soležne’

Primer V-diagramov 3 spremenljivk

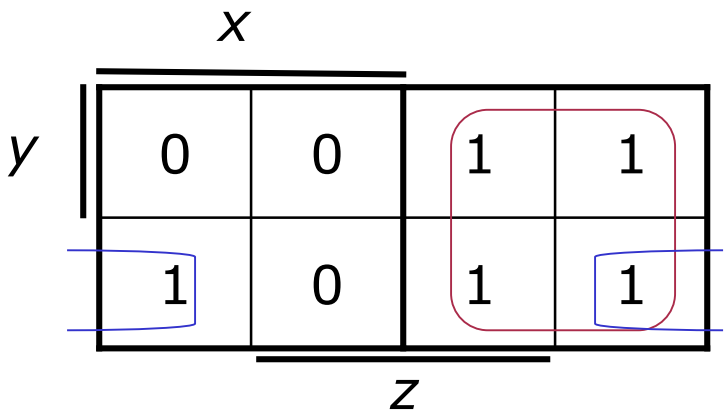
$f(x, y, z) = \Sigma m(0,1,2,4)$

$= x'y' + x'z' + y'z'$



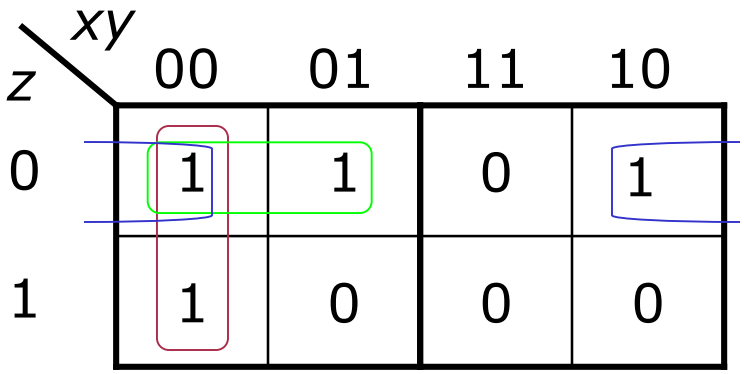
$f(x, y, z) = \Sigma m(0,1,2,3,4)$

$= x' + y'z'$

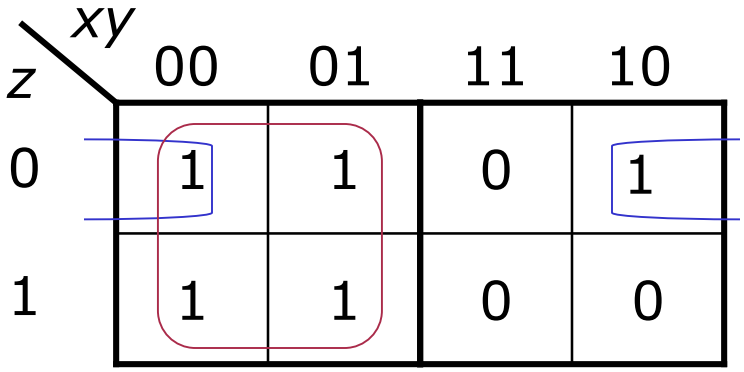


Primer K-diagramov 3 spremenljivk

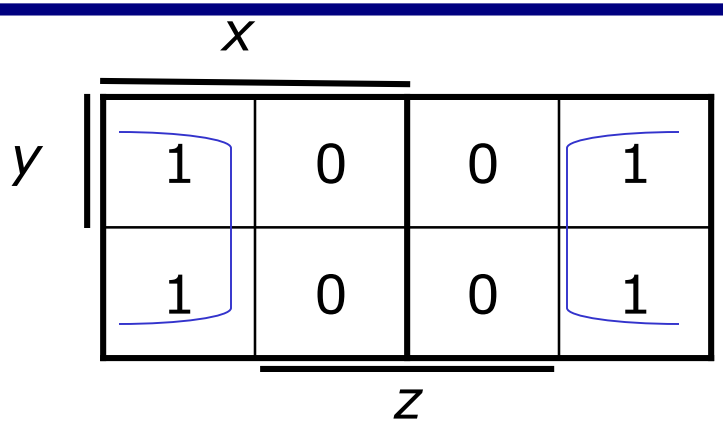
$f(x,y,z)=\Sigma m(0,1,2,4)$
$=x'y'+x'z'+y'z'$



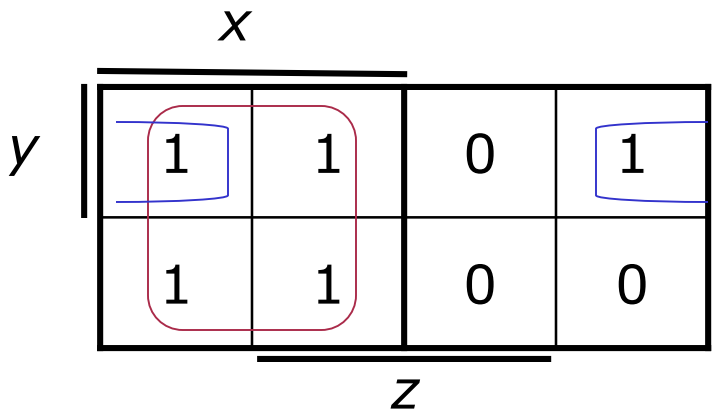
$f(x,y,z)=\Sigma m(0,1,2,3,4)$
$=x'+y'z'$



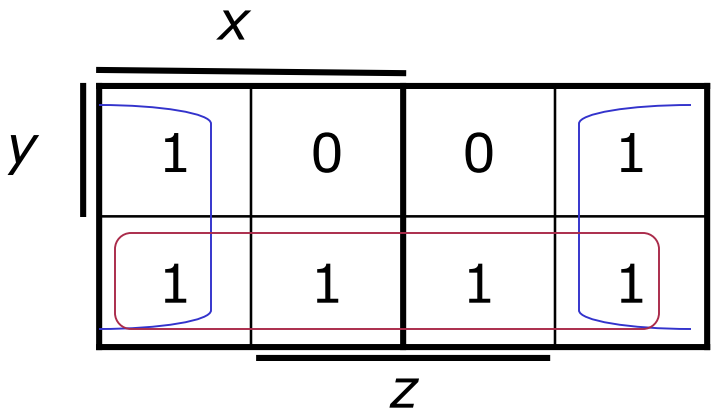
Primeri združevanja v V-diagramih



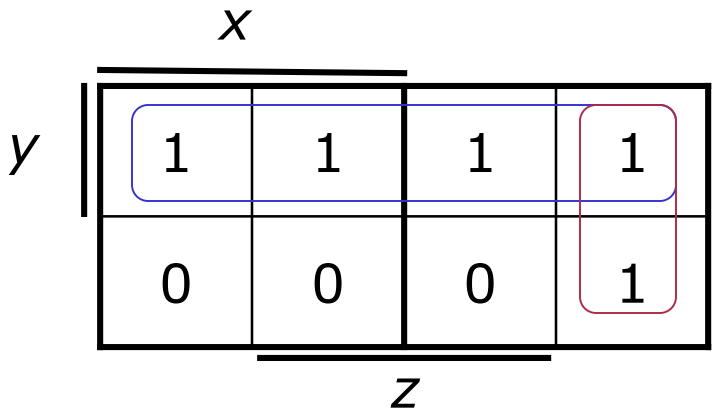
f=z′



f=yz′+x

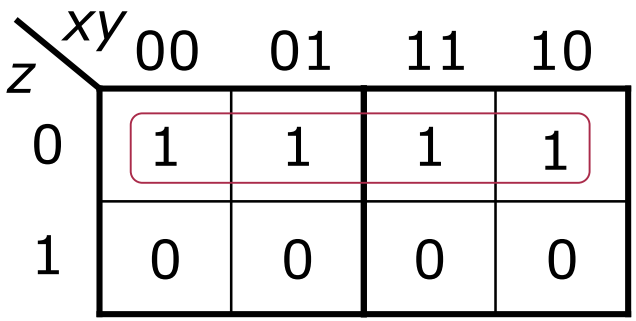


f=y′+z′

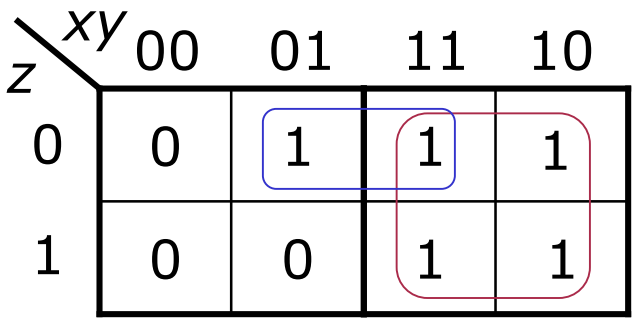


f=y+x′z′

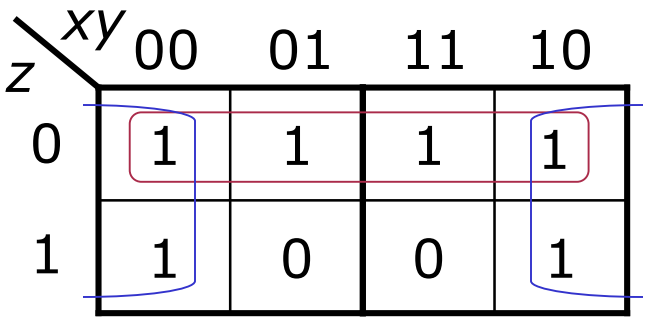
Primeri združevanja v K-diagramih



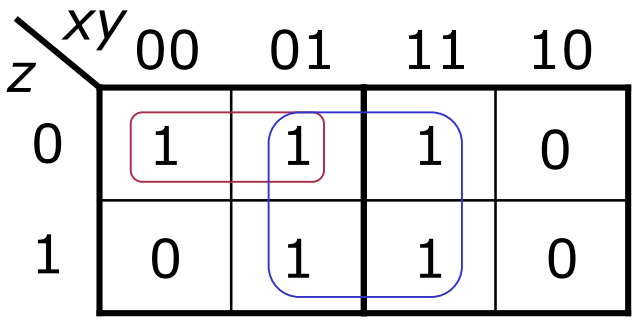
f=z′



f=yz′+x



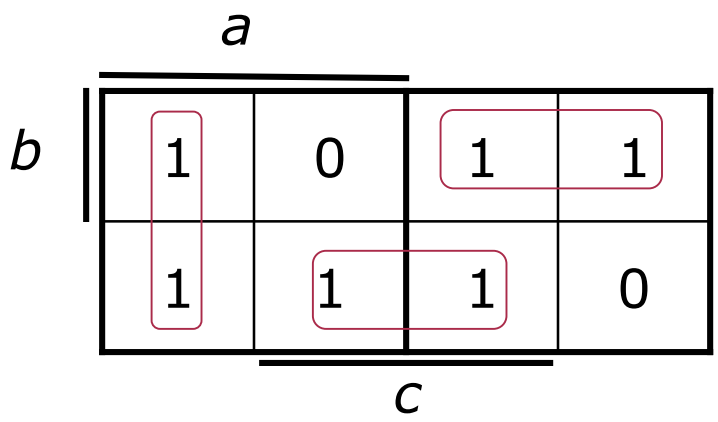
f=z′+y′

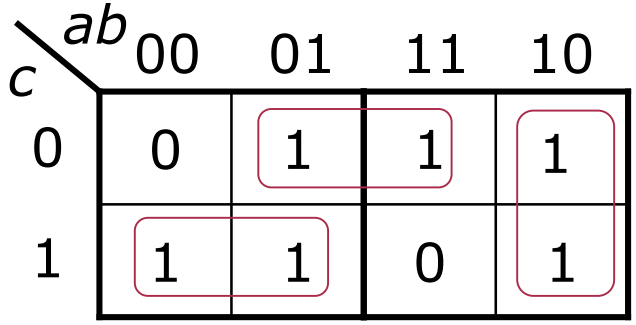


f=y+x′z′

Primeri združevanja v K-diagramih

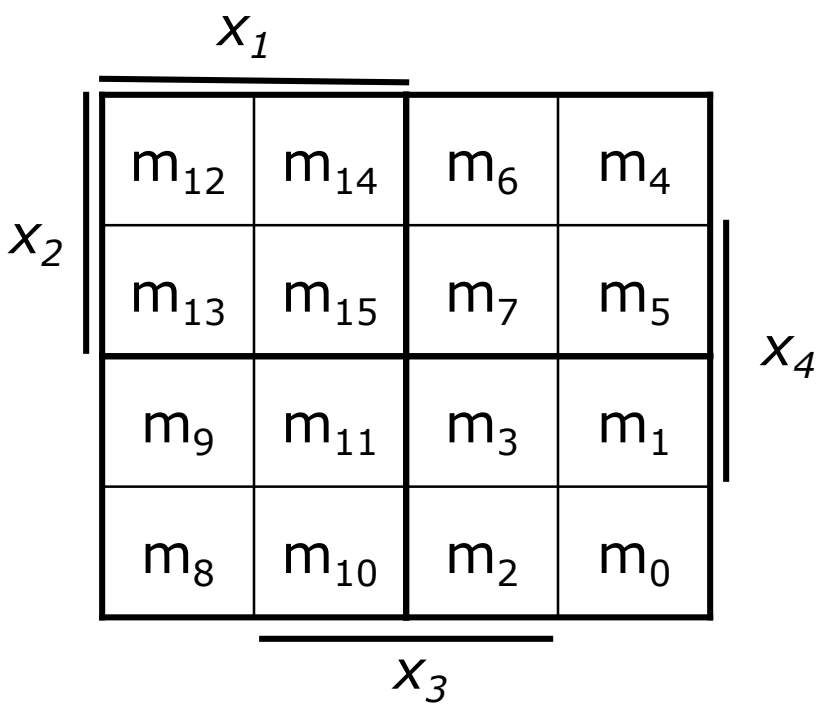
- Narišite K&V diagrama za podano funkcijo in izrazite MDNO: f(a,b,c)=Σm(1,2,3,4,5,6)
- Prikažite postopek združevanja v obeh diagramih





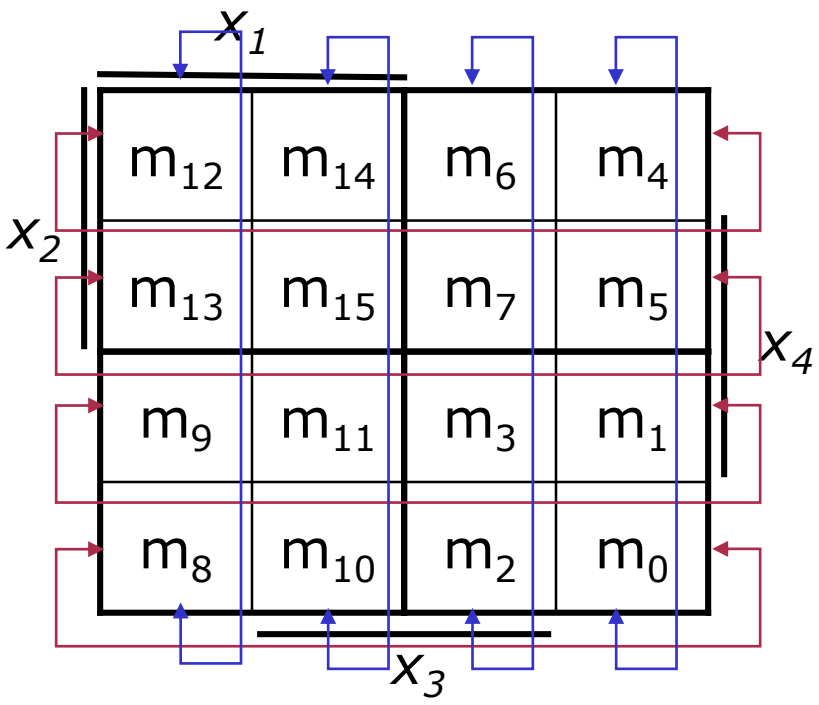
f(a,b,c)=ab’+a’c+bc’

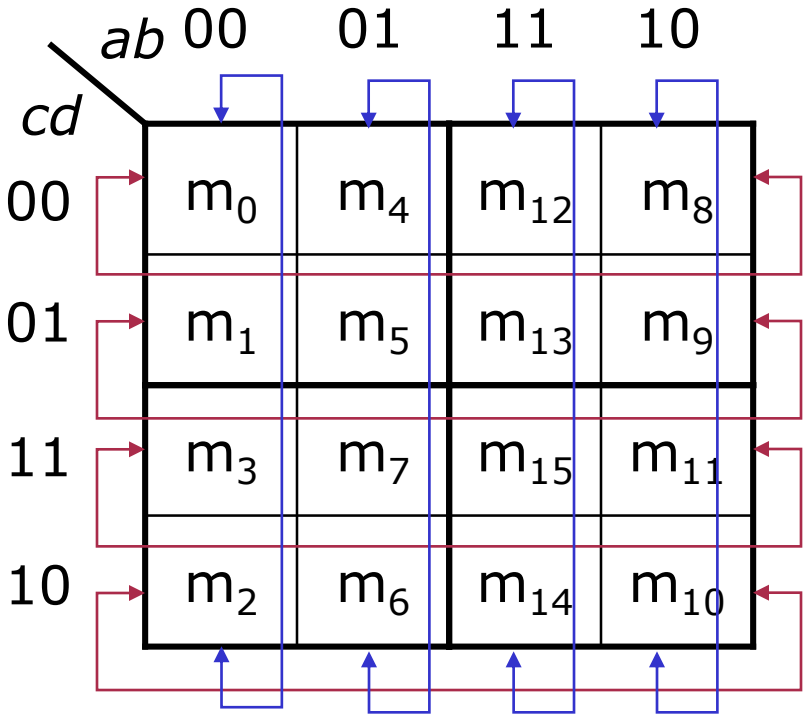
Diagrama 4 spremenljivk



cd \ *ab*	00	01	11	10
00	m_0	m_4	m_{12}	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

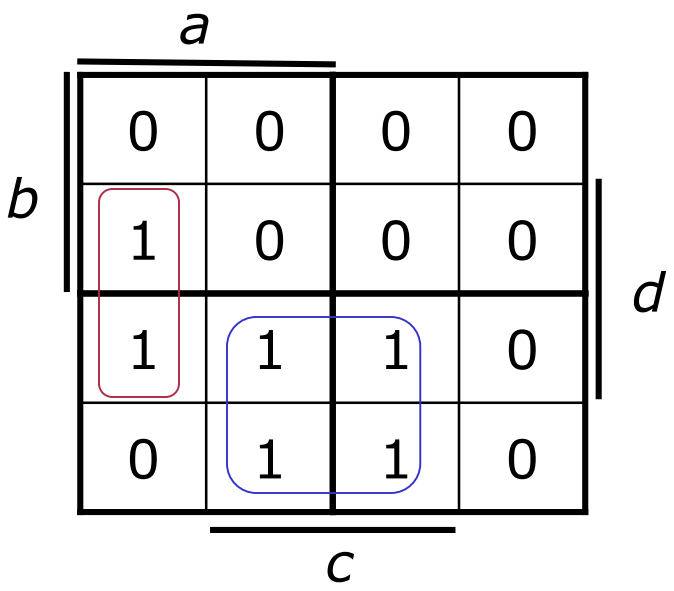
Diagrama 4 spremenljivk

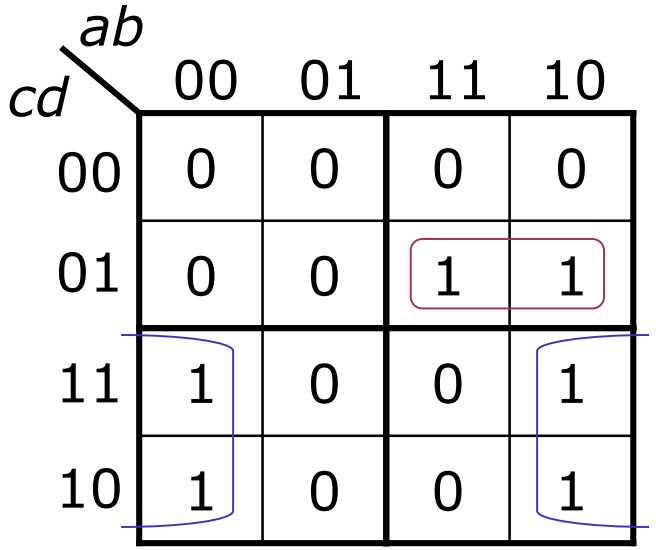




Primeri združevanja v diagramih

$f(a,b,c,d)=\Sigma m(2,3,9\text{-}11,13) = ac'd + b'c$

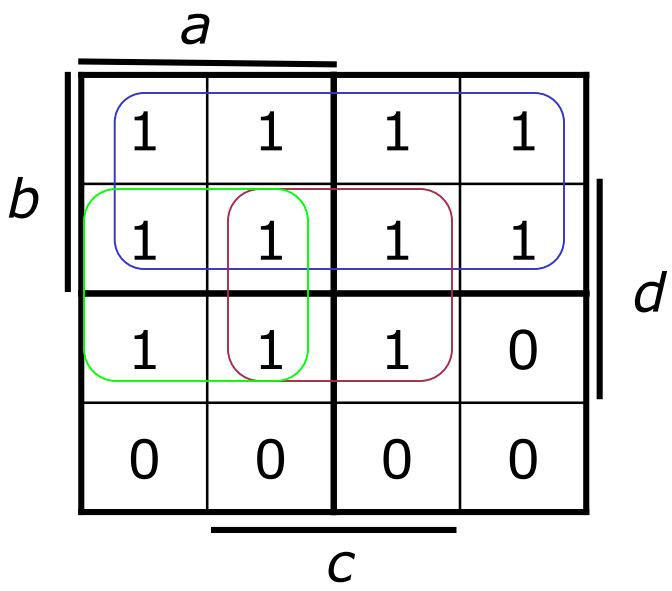


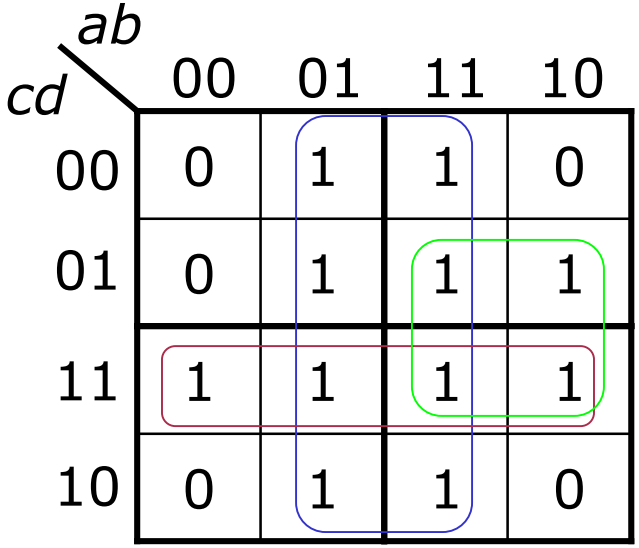


Primeri združevanja v diagramih

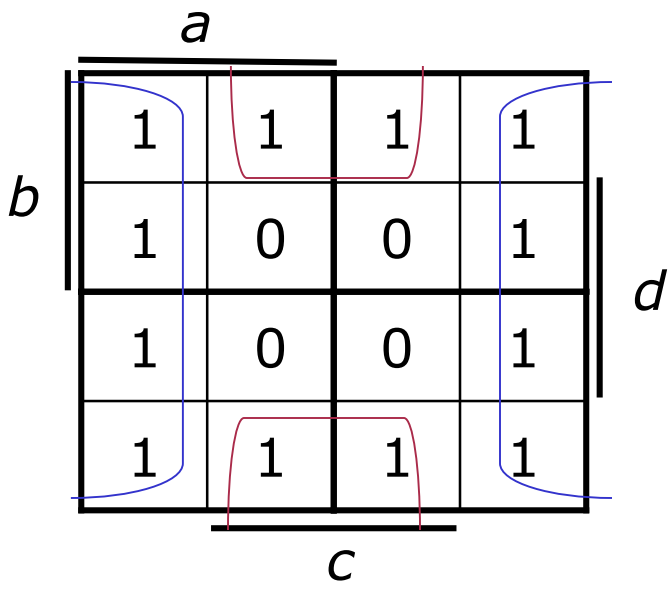
$f(a,b,c,d)=\Sigma m(3\text{-}7,9,11\text{-}15)$

$=b+cd+ad$



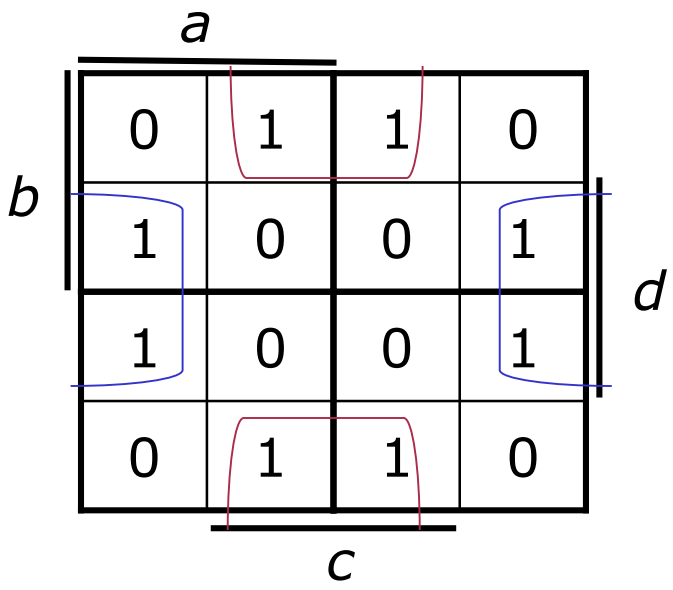


Primeri združevanja v diagramih



f(a,b,c,d)=c’+d’

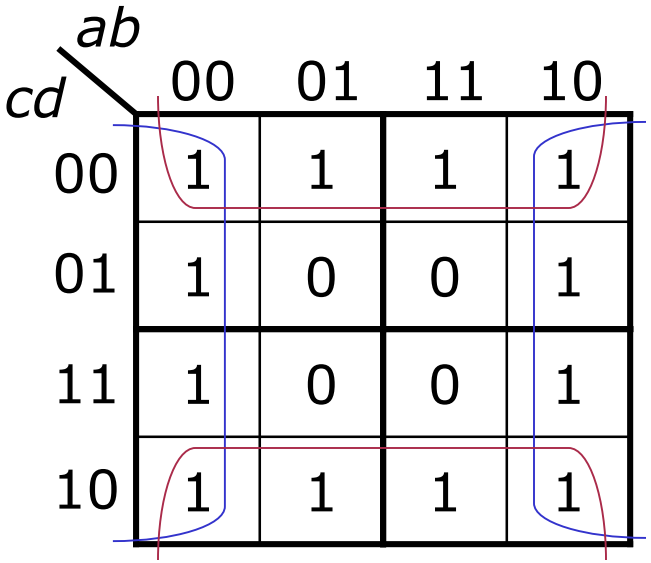
f(a,b,c,d)=(c·d)’
NAND



f(a,b,c,d)=c’d+cd’

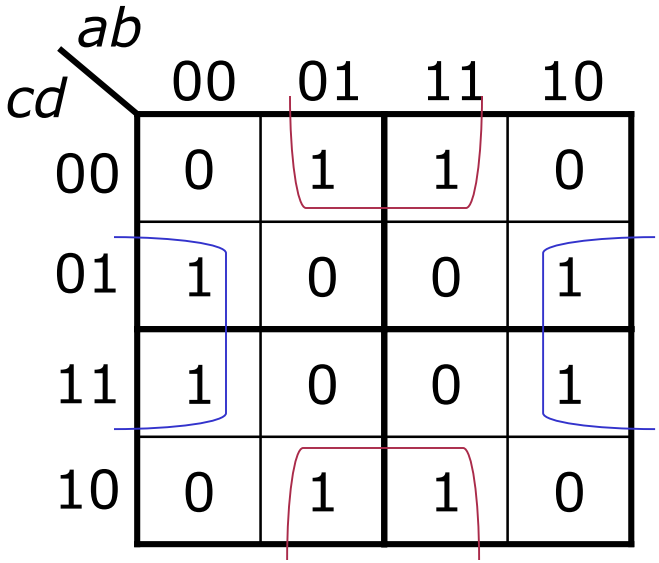
f(a,b,c,d)= c XOR d

Primeri združevanja v diagramih



f(a,b,c,d)=b’+d’

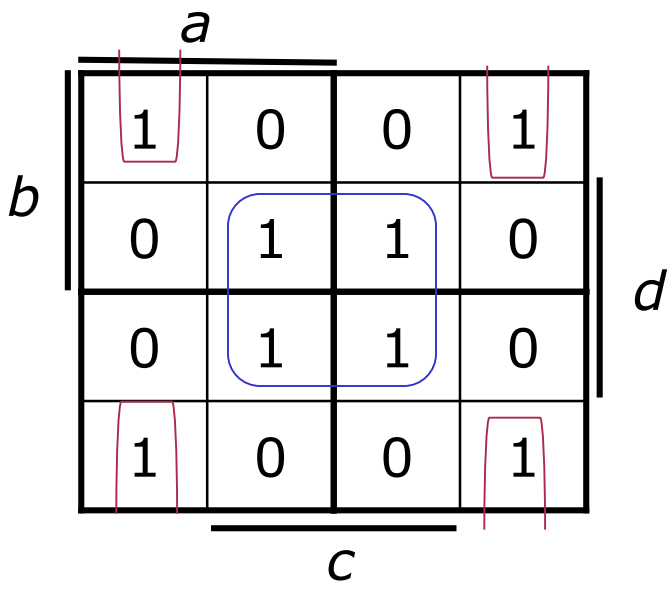
f(a,b,c,d)= b NAND d



f(a,b,c,d)=b’d+bd’

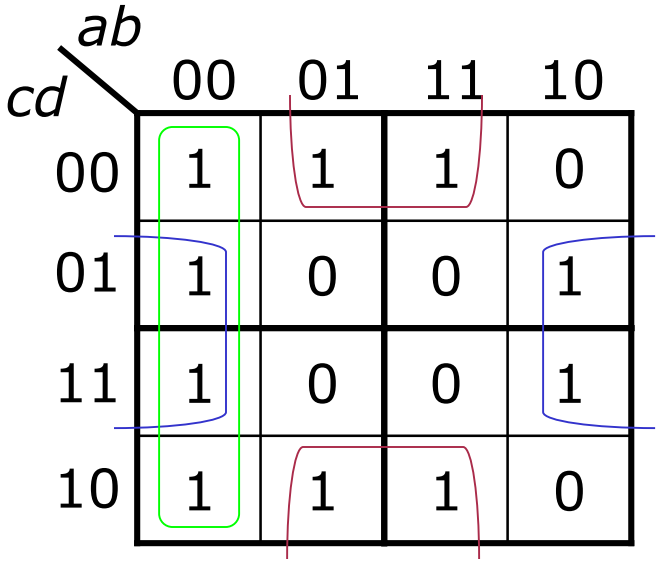
f(a,b,c,d)= b XOR d

Primeri združevanja v diagramih



f(a,b,c,d)=c’d’+cd

f(a,b,c,d)=c XNOR d



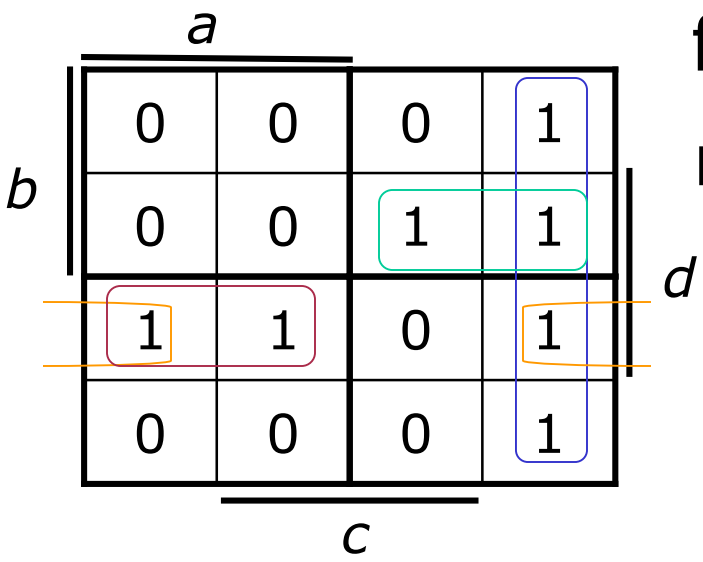
f(a,b,c,d)=b’d+bd’+a’b’

Razvoj digitalnih sistemov

Optimizacija logičnih funkcij:

Minimizacija funkcij, MKNO,
nepopolno določene funkcije

Glavni vsebovalniki



$f(a,b,c,d)=V(0,1,4,5,7,9,11)$

Primeri vsebovalnikov:

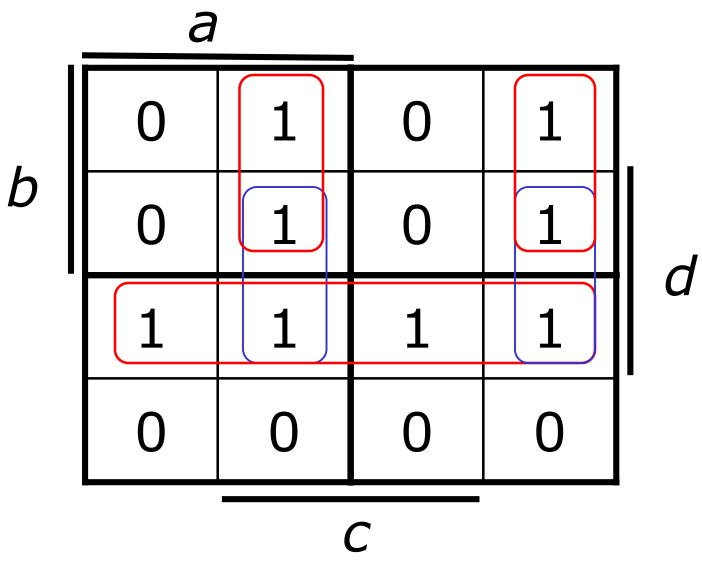
vse same '1',
a'c', a'b'c',
a'bd, ab'd

Glavni vsebovalniki: a'c', a'bd, ab'd, b'c'd

$f(a, b, c, d)_{MDNO} = a'c'+a'bd+ab'd$

MDNO vsebuje *samo* (ne pa nujno vseh) glavne vsebovalnike!

Razlikovanje glavnih vsebovalnikov



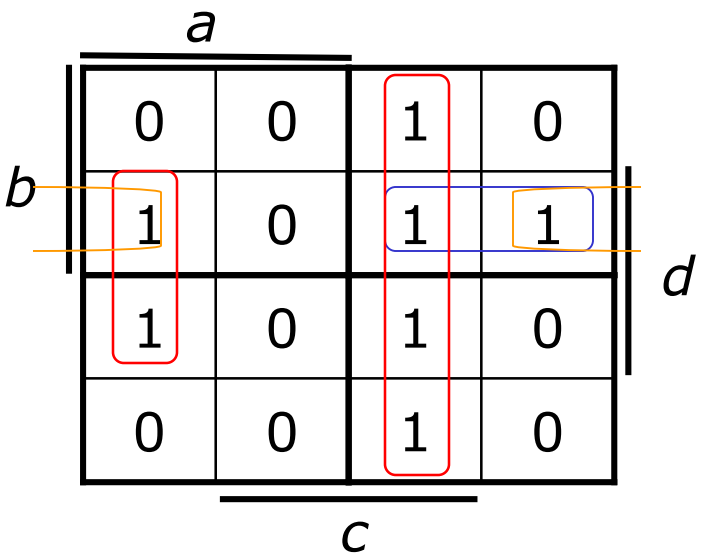
Vsi glavni vsebovalniki:
b'd, a'bc', abc, a'c'd, acd

Bistveni: b'd, a'bc', abc

Nebistveni: a'c'd, acd

$f(a,b,c,d)_{MDNO}$= b'd+a'bc'+abc

Primer razlikovanja vsebovalnikov



Bistveni: a'c, ac'd

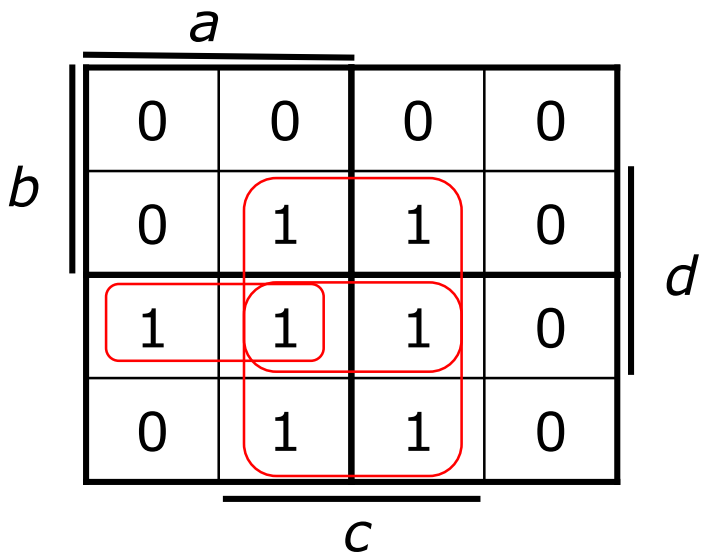
Nebistveni: a'bd, bc'd

Eden od teh dveh mora biti vključen v MDNO

$$f(a,b,c,d)_{MDNO} = a'c + ac'd + \left\{ \begin{matrix} a'bd \\ bc'd \end{matrix} \right\}$$

Primer razlikovanja vsebovalnikov

Določite vse glavne vsebovalnike v danem diagramu.
Kateri so bistveni in kateri nebistveni? Določite MDNO funkcije.



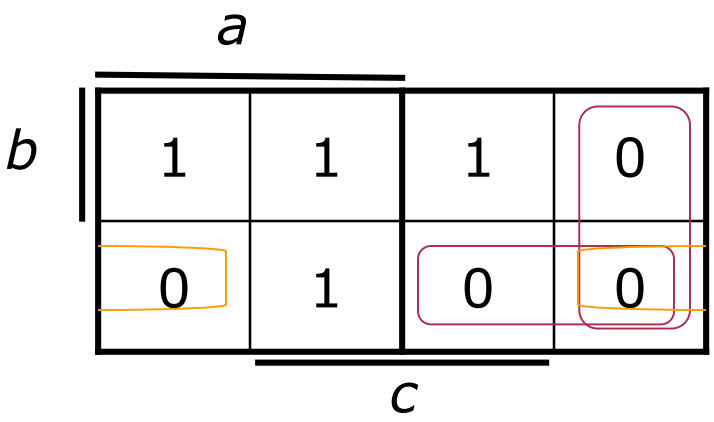
$f_{MDNO}(a,b,c,d)=dc+b'c+ab'd$

MKNO

$f_{PKNO}(a,b,c)=(a+b'+c')(a'+b+c')(a'+b'+c)(a'+b'+c')$

$f_{PKNO}(a,b,c)=\Pi M(3,5,6,7) = \&(3,5,6,7)$

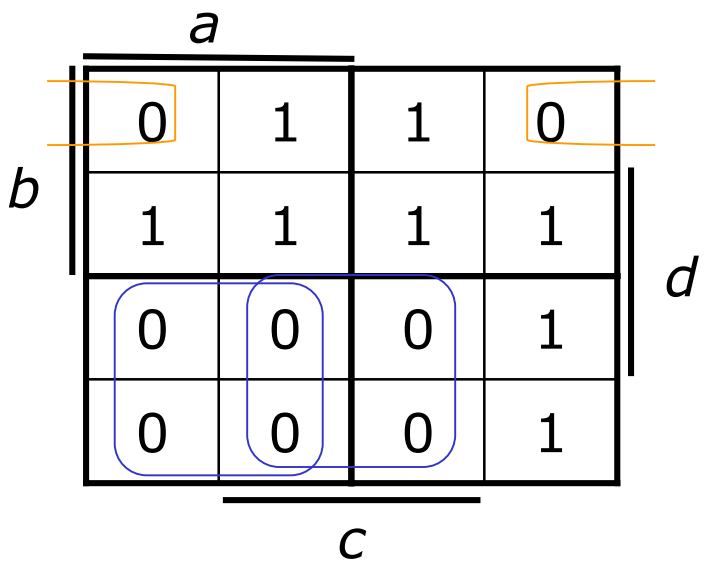
$f_{PDNO}(a,b,c)=V(7,6,5,3)$



$f_{MKNO}(a,b,c)=(a+c)(b+c)(a+b)$

Zgled MKNO

$f_{PDNO}(a, b, c, d)=\Sigma m(0,1,5\text{-}7,13\text{-}15)$



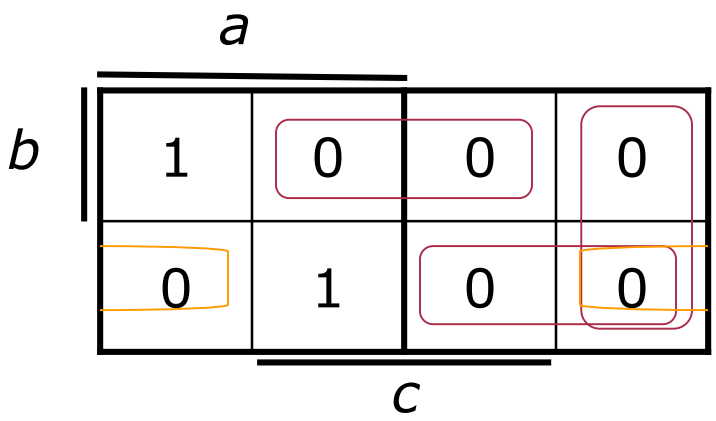
$f'(a, b, c, d)=ab'+b'c+bc'd'$

$f =(ab'+b'c+bc'd')'$ ➔ De Morgan

$f_{MKNO}=(a'+b)(b+c')(b'+c+d)$

Zgled

- Narišite Veitch-ev diagram in določite MKNO.
  - f(a, b, c)=ΠM(0,2,3,5-7)
- Prikažite združevanja v diagramu



f’(a,b,c)=a’+b’c’+bc ➔ De Morgan

f(a,b,c)= a(b’c’)’(bc)’

$f(a,b,c)_{MKNO}$=a(b+c)(b’+c’)

Nepopolno določene funkcije

x	*y*	*z*	*f*
0	0	0	1
0	0	1	1
0	1	0	X
0	1	1	X
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$f(x, y, z) = \Sigma m(0,1,4,5)$

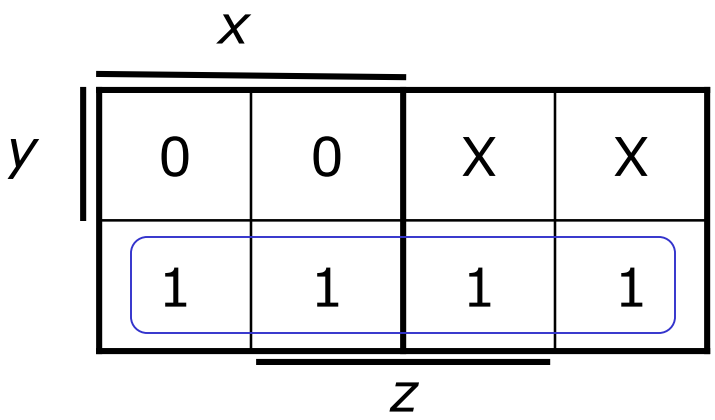
Kombinacija xy=01 se nikdar ne pojavi.

$$f_{PDNO}(x, y, z) = V(0,1,4,5) + V_x(2,3)$$

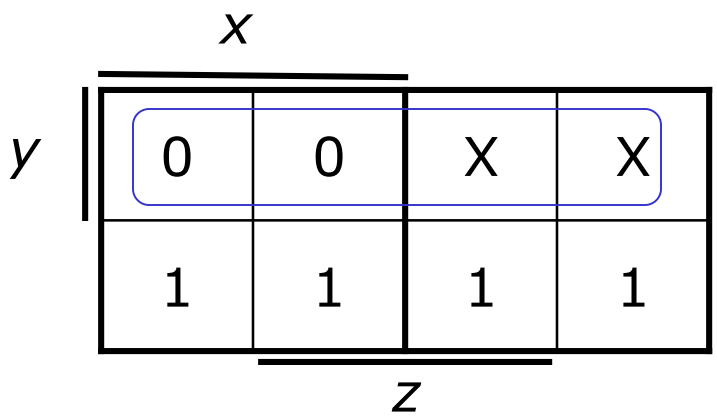
$$f_{PKNO}(x, y, z) = \&(0,1) \cdot \&_x(4,5)$$

Zgled nepopolno določene funkcije

$f(x,y,z) = V(0,1,4,5) + V_x(2,3)$



$f(x,y,z) = \&(0,1) \cdot \&_x(4,5)$



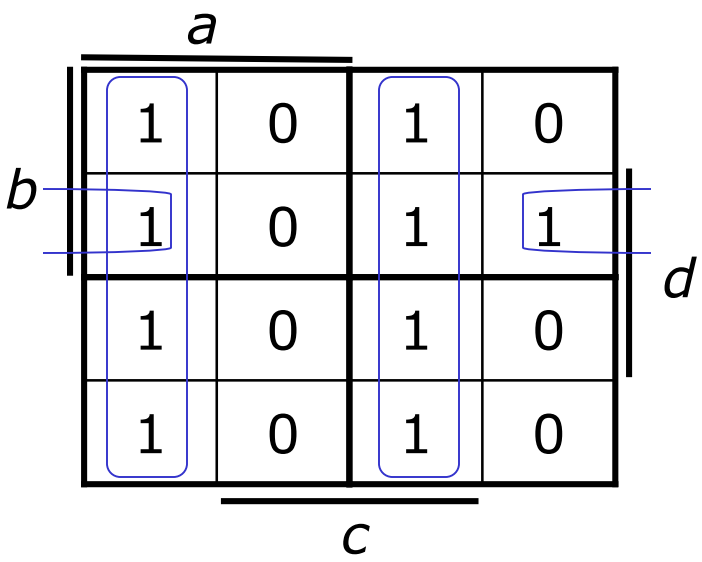
$f'(x,y,z) = y$

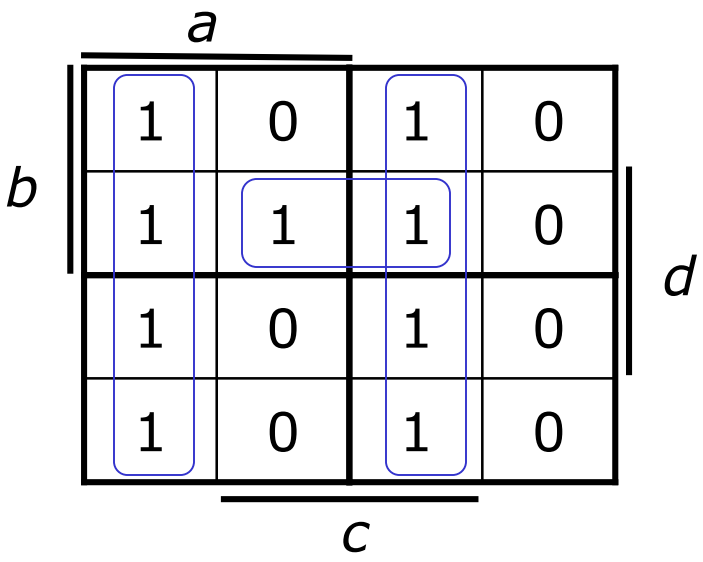
$f(x,y,z) = y'$ ⟷ $f(x,y,z) = y'$

MNO (Minimalna normalna oblika)

- Zapišemo MDNO
- Zapišemo MKNO
- Izračunamo COST obeh oblik
- Izberemo nižji COST
- Oblika, ki ima nižji COST je MNO
  (če je COST isti, sta obe obliki enakovredni)

Zgled vezja z več izhodi



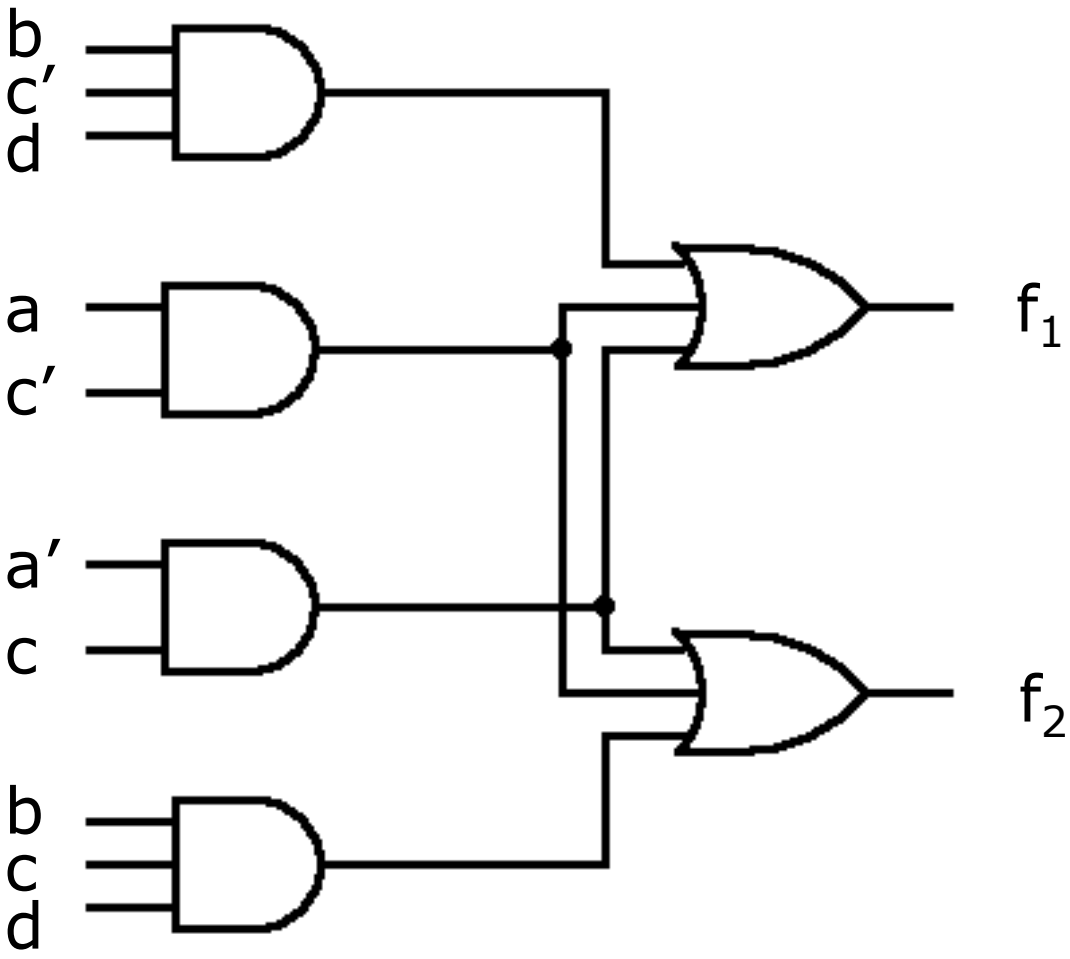


$f_1(a,b,c,d)=ac'+a'c+bc'd$ $f_2(a,b,c,d)=ac'+a'c+bcd$

COST=4 vrat+10 vhodov COST=4 vrat+10 vhodov

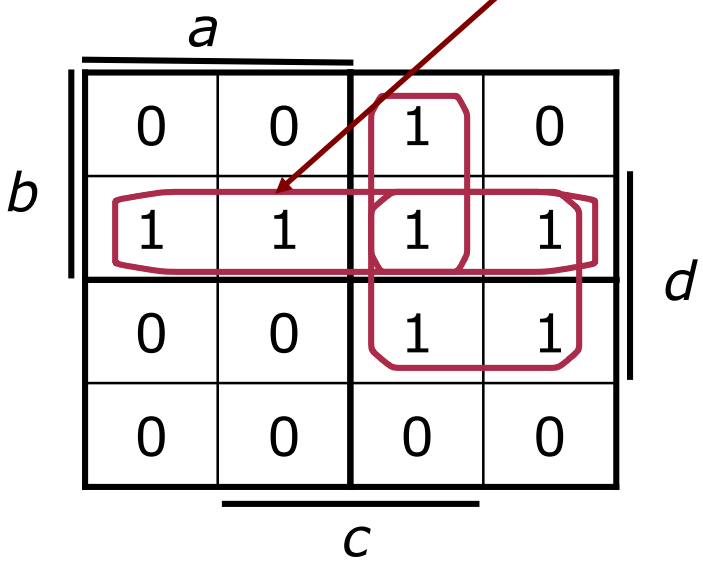
V COST niso všteti inverterji!

Zgled vezja z več izhodi

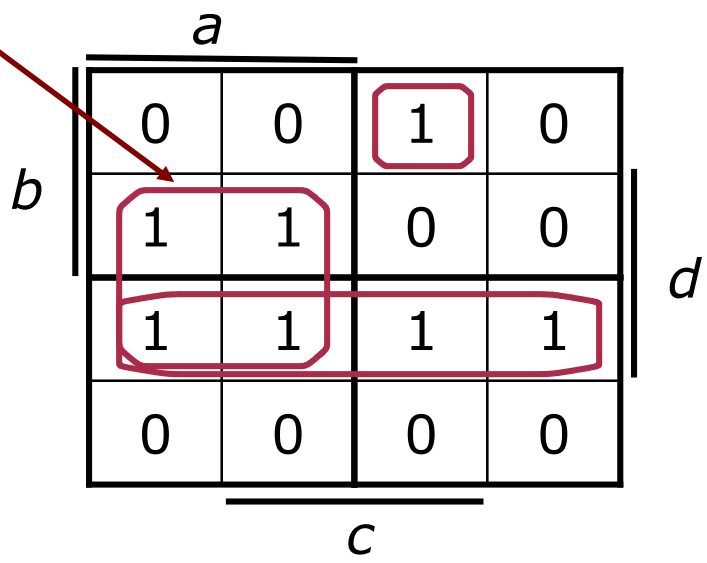


Normalna realizacija vezja z več izhodi

skupni člen, ki ga realiziramo dvakrat (ni optimalno)



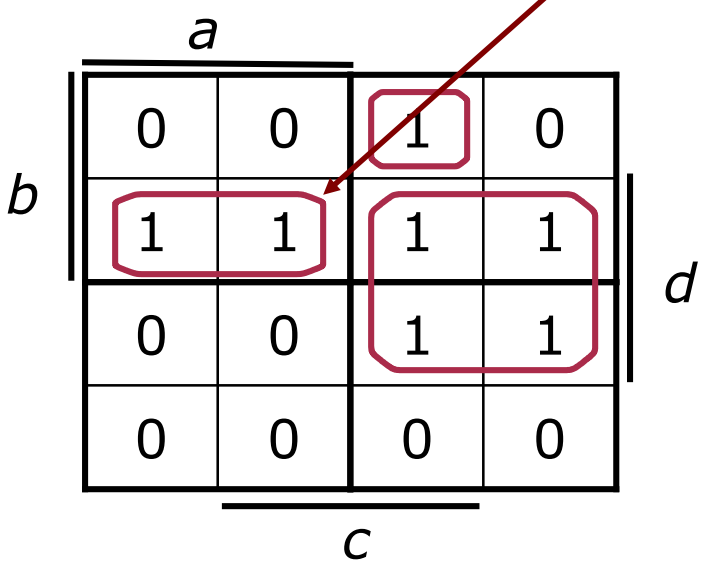
Optimalna realizacija
$f_3 = a'd + bd + a'bc$

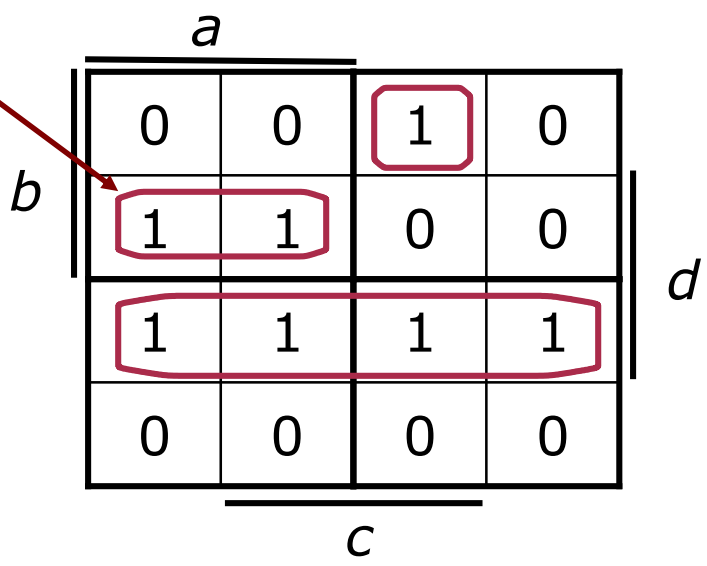


Optimalna realizacija
$f_4 = ad + b'd + a'bcd'$

Optimalna realizacija vezja z več izhodi

skupni člen, ki ga realiziramo samo enkrat!





Optimalna realizacija
$f_3 = a'd + abd + a'bcd'$

Optimalna realizacija
$f_4 = abd + b'd + a'bcd'$

Optimalna skupna realizacija f_3 in f_4

Razvoj digitalnih sistemov

Realizacija optimalnih logičnih funkcij:

NAND in NOR logična vezja

Funkcijsko polni sistemi

- **AND, NOT**
  $((x_1 + x_2)')' = (x_1'x_2')'$ ⟷ DeMorgan
- **OR, NOT**
  $((x_1\, x_2)')' = (x_1' + x_2')'$ ⟷ DeMorgan
- **NAND** (Sheffer-jev operator ↑)
  $(x_1x_2)' = x_1 \uparrow x_2$
- **NOR** (Piercev operator ↓)
  $(x_1 + x_2)' = x_1 \downarrow x_2$
- **XOR, AND, 1**

NAND – Sheffer-jev operator

Negacija (NOT):

$x' = x' + x' = (x \cdot x)' = x \uparrow x$

Konjunkcija (AND):

$x_1 \cdot x_2 = (x_1 \cdot x_2)'' = (x_1 \uparrow x_2)' = (x_1 \uparrow x_2) \uparrow (x_1 \uparrow x_2)$

Disjunkcija (OR):

$x_1 + x_2 = (x_1 + x_2)'' = (x_1' \cdot x_2')'$

$(x_1' \cdot x_2')' = x_1' \uparrow x_2' = (x_1 \uparrow x_1) \uparrow (x_2 \uparrow x_2)$

NOR – Pierce-v operator

Negacija (NOT):

$x' = x' \cdot x' = (x+x)' = x \downarrow x$

Konjunkcija (AND):

$x_1 \cdot x_2 = (x_1 \cdot x_2)'' = (x_1' + x_2')' = (x_1' \downarrow x_2')' =$
$= (x_1 \downarrow x_1) \downarrow (x_2 \downarrow x_2)$

Disjunkcija (OR):

$x_1 + x_2 = (x_1 + x_2)'' = (x_1 \downarrow x_2)' = x_1' \downarrow x_2' =$
$= (x_1 \downarrow x_2) \downarrow (x_1 \downarrow x_2)$

XOR, AND, 1

Negacija (NOT):

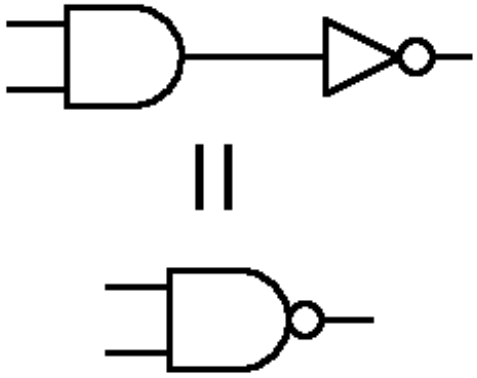
$x'=x \oplus 1$

Disjunkcija (OR):

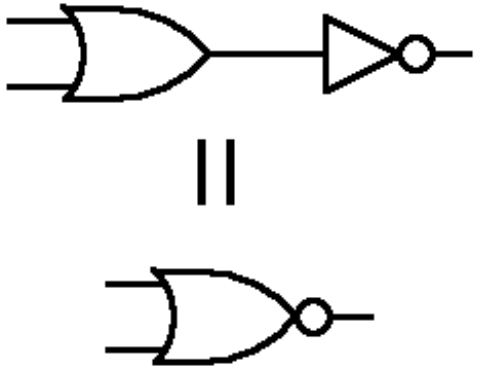
$x_1+x_2=(x_1+x_2)''=(x_1'x_2')'=$

$=((x_1 \oplus 1)(x_2 \oplus 1)) \oplus 1$

NAND in NOR logična vezja



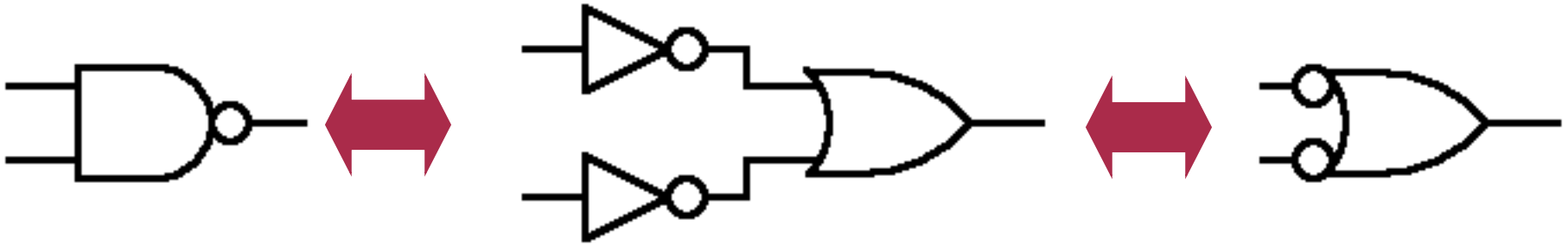
x_1	x_2	$(x_1 \cdot x_2)'$ $x_1 \uparrow x_2$
0	0	1
0	1	1
1	0	1
1	1	0



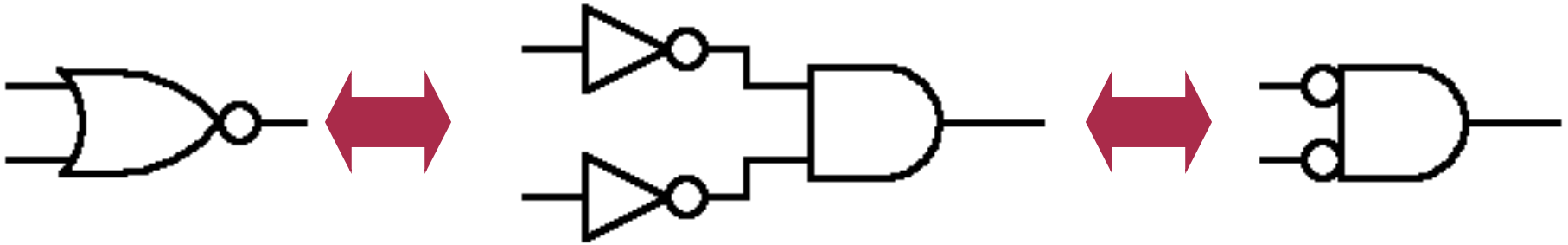
x_1	x_2	$(x_1+x_2)'$ $x_1 \downarrow x_2$
0	0	1
0	1	0
1	0	0
1	1	0

DeMorgan-ov teorem v vezjih

Potiskanje mehurčka (ang. pushing the bubble)

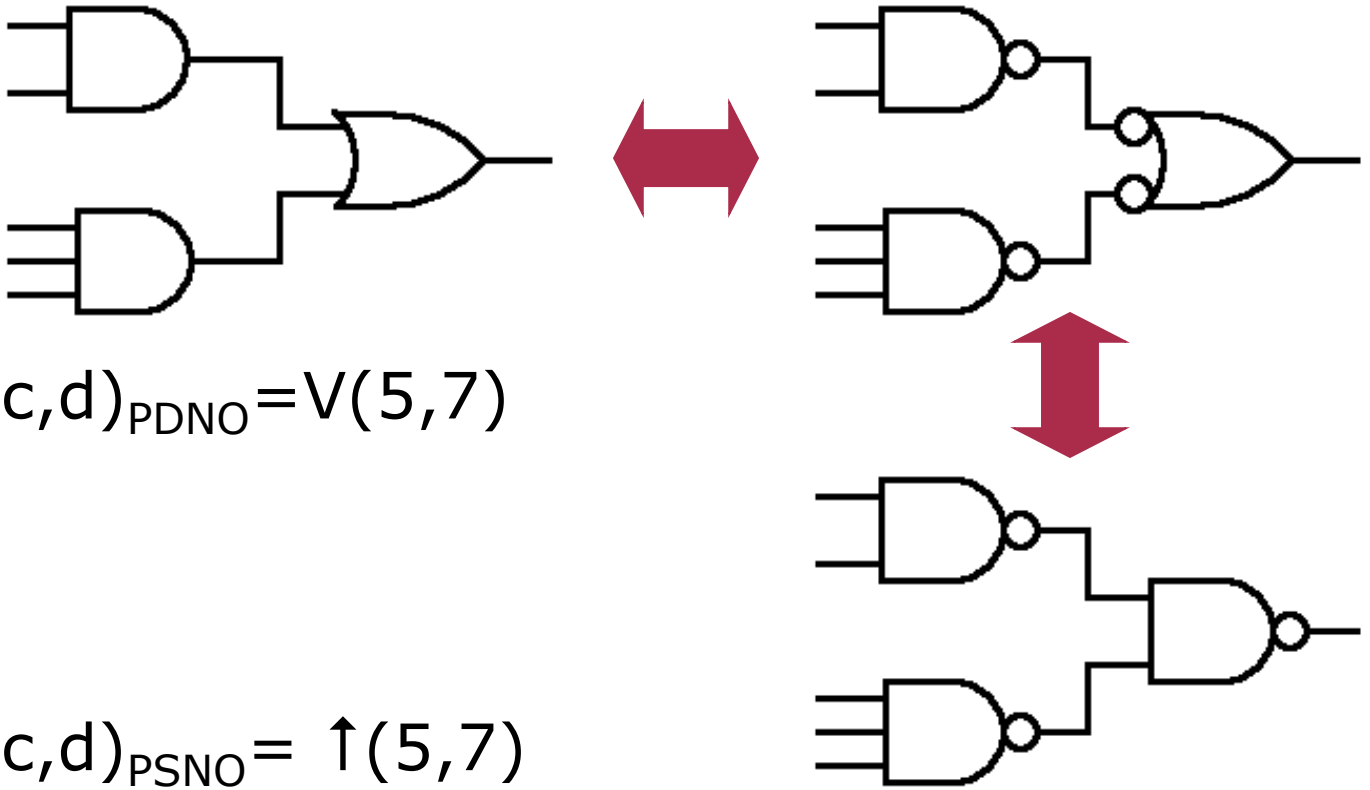


(ab)’ = a’+b’



(a+b)’ = a’b’

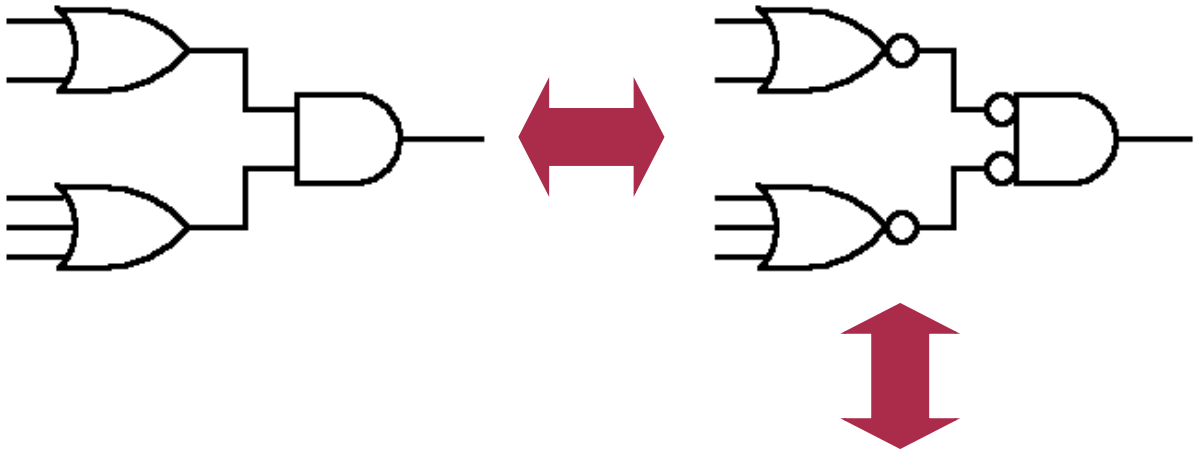
PDNO ➔ NAND vezja (PSNO)



$f(a,b,c,d)_{PDNO}=V(5,7)$

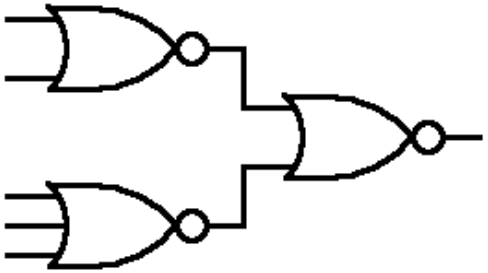
$f(a,b,c,d)_{PSNO}= \uparrow(5,7)$

PKNO ➔ NOR vezja (PPNO)



$f(a,b,c,d)_{PKNO}=\&(6,3)$

$f(a,b,c,d)_{PPNO}=\downarrow(6,3)$



Zapis DNO z NAND = SNO

$$f(x_1,x_2,x_3)=x_1x_2+x_2'x_3+x_3'=$$
$$=(x_1x_2+x_2'x_3+x_3')''=$$
$$=((x_1x_2)'(x_2'x_3)'(x_3')')'=$$
$$=(x_1 \uparrow x_2)\uparrow(x_2' \uparrow x_3) \uparrow(x_3)$$

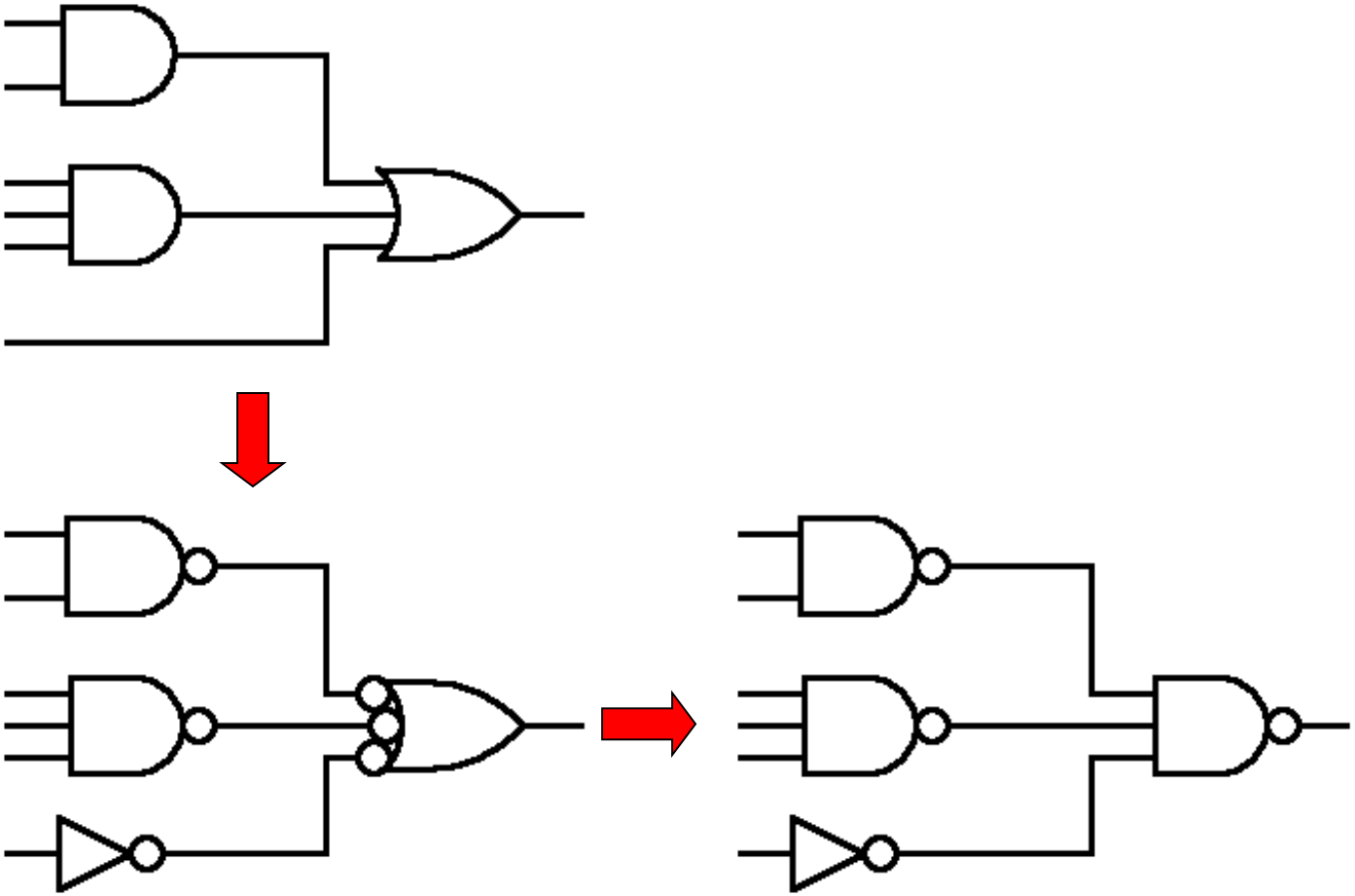
Enak pristop lahko uporabljamo na večnivojskih vezjih

Zapis KNO z NOR = PNO

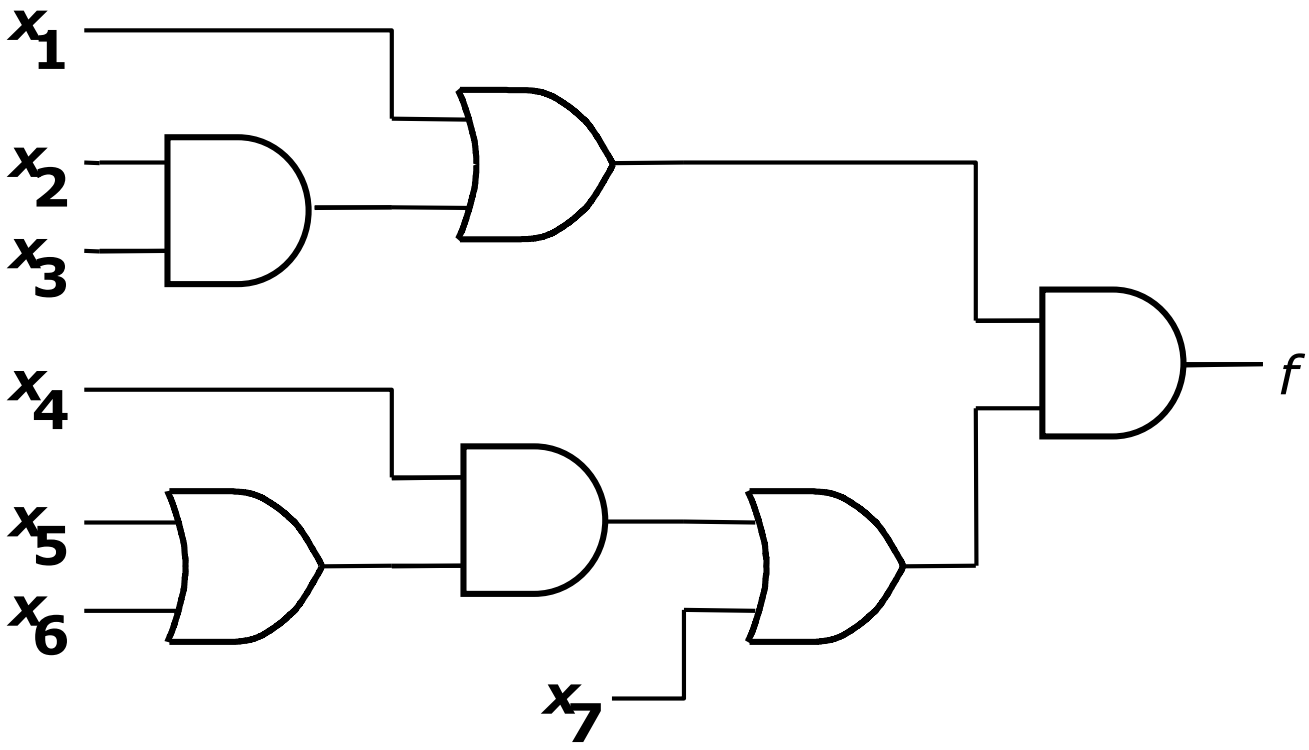
$$f(x_1,x_2,x_3)=(x_1+x_2)(x_2'+x_3')(x_3)=$$
$$=((x_1+x_2)(x_2'+x_3')(x_3))''=$$
$$=((x_1+x_2)'+(x_2'+x_3')'+(x_3)')'=$$
$$=(x_1 \downarrow x_2) \downarrow (x_2' \downarrow x_3') \downarrow x_3'$$

Enak pristop lahko uporabljamo na večnivojskih vezjih

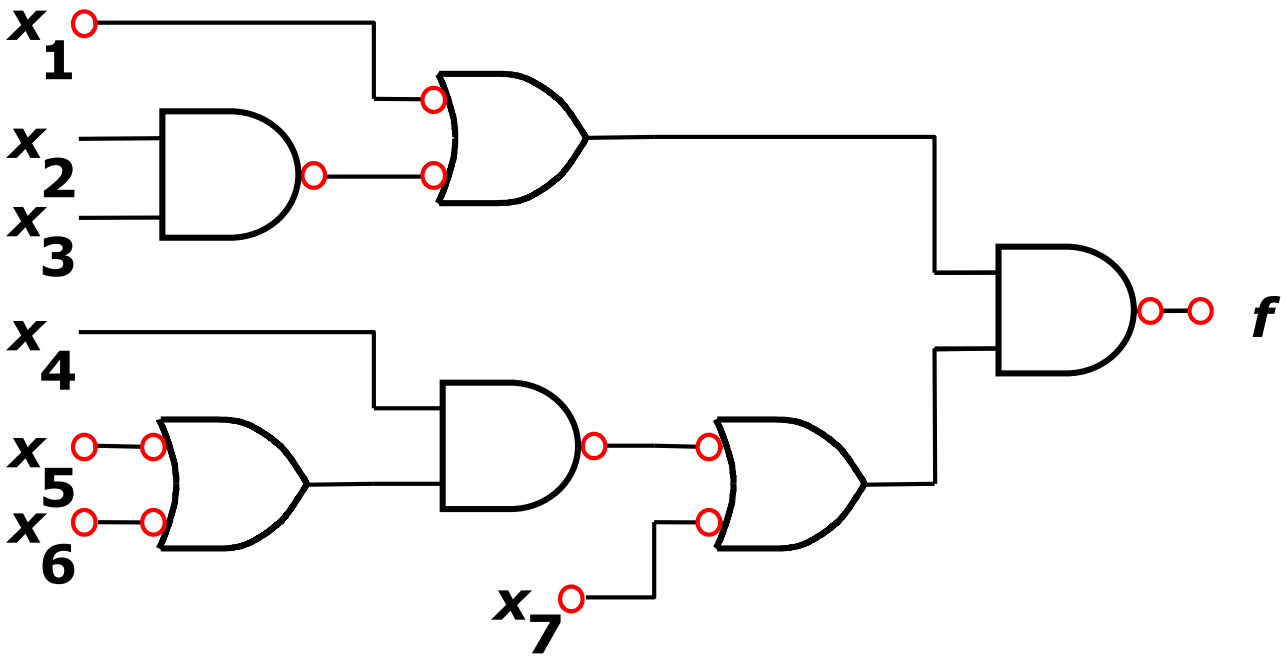
DNO➔Večnivojski (NAND-NAND)



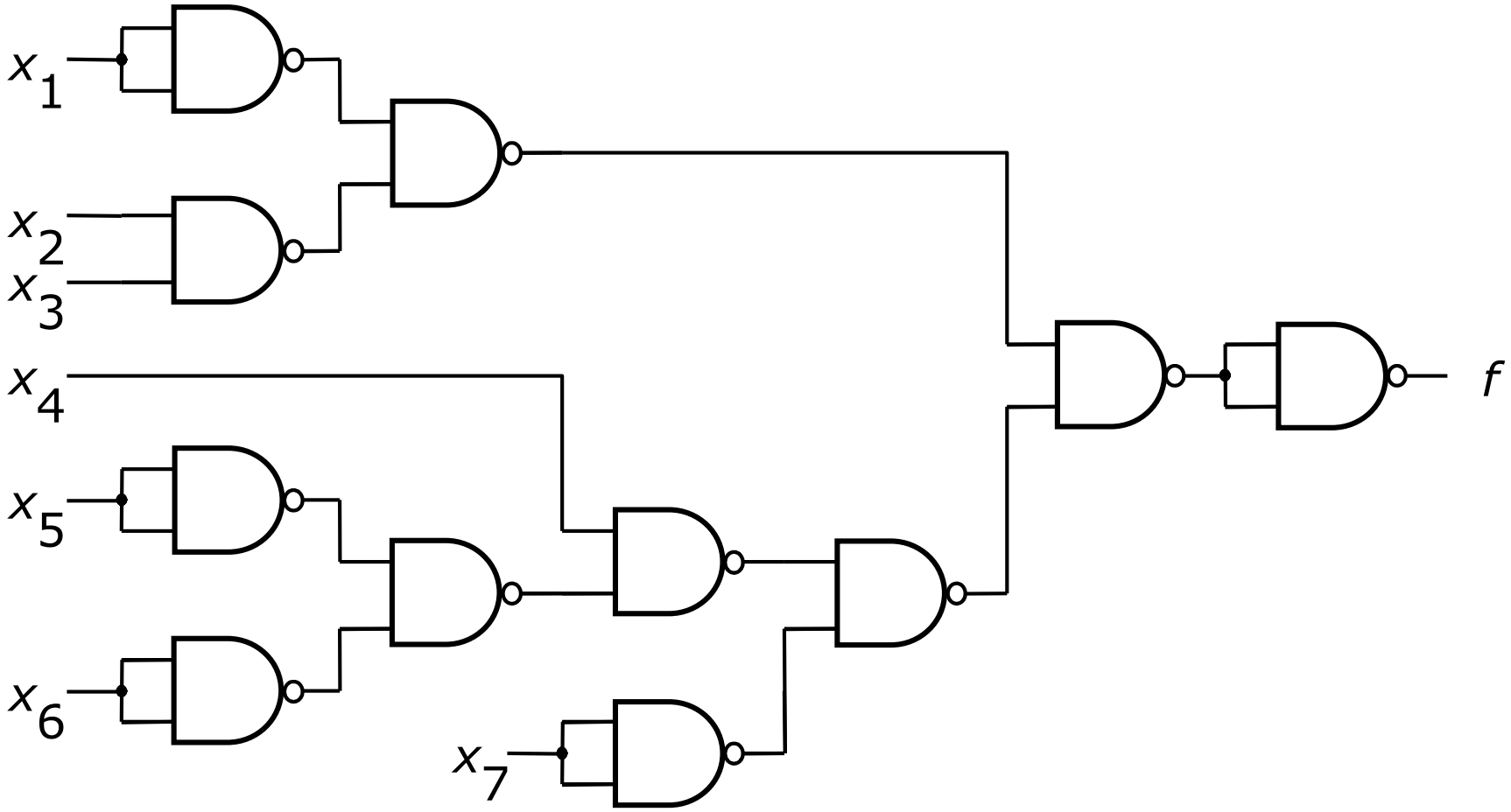
DNO➔Večnivojski (NAND-NAND)



DNO➔Večnivojski (NAND-NAND)

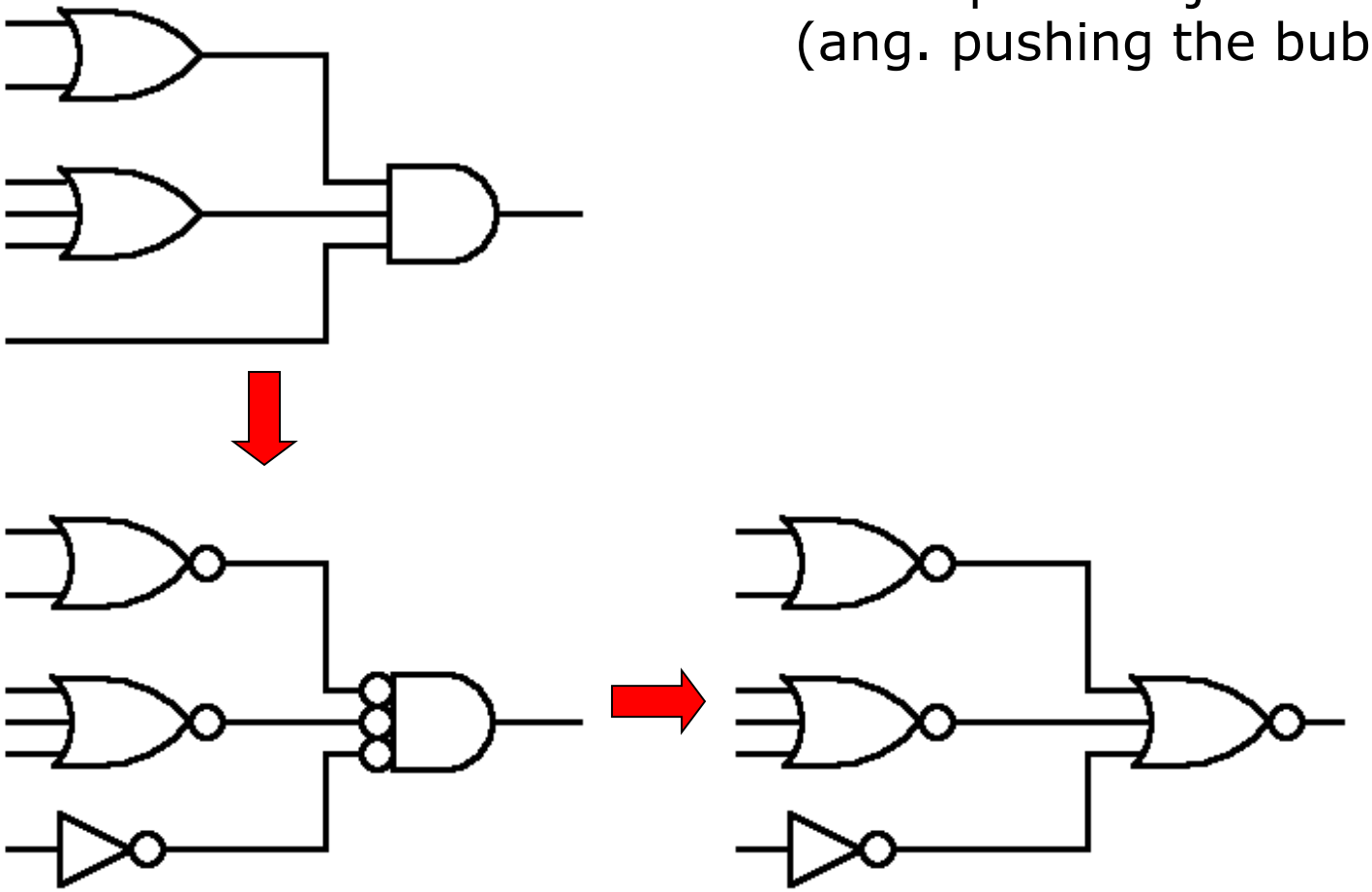


DNO➔Večnivojski (NAND-NAND)

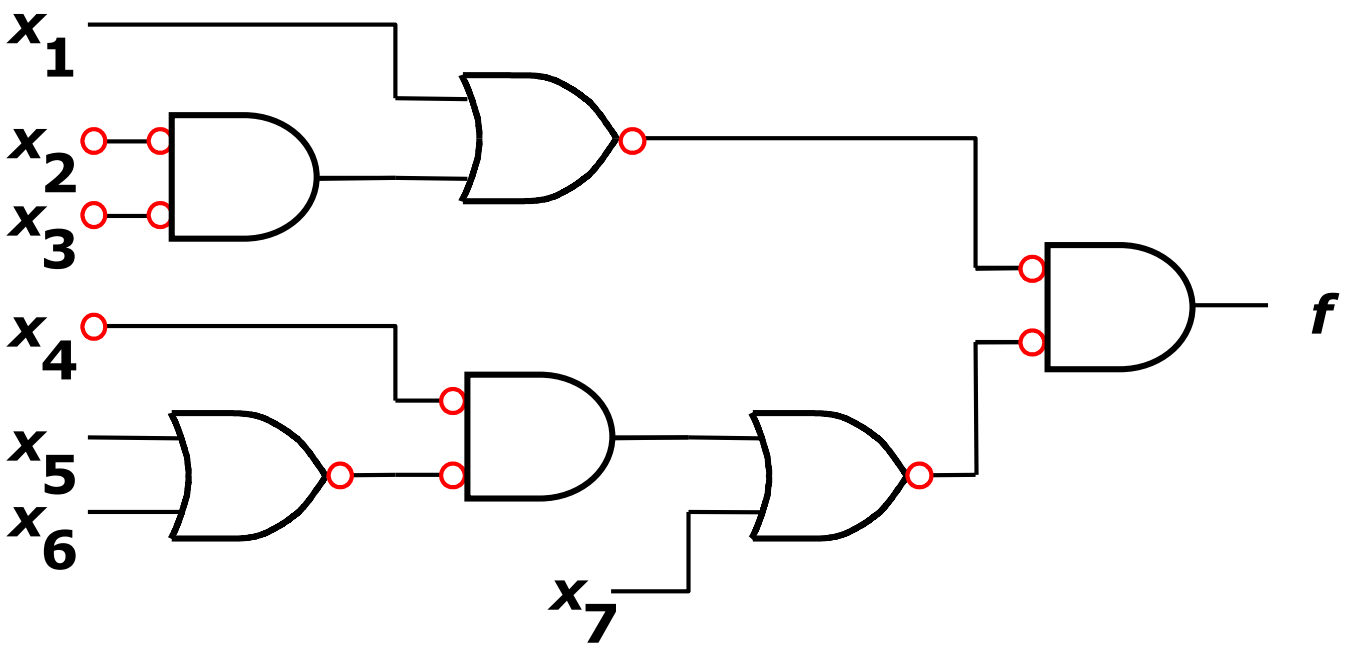


KNO➔ Večnivojski (NOR-NOR)

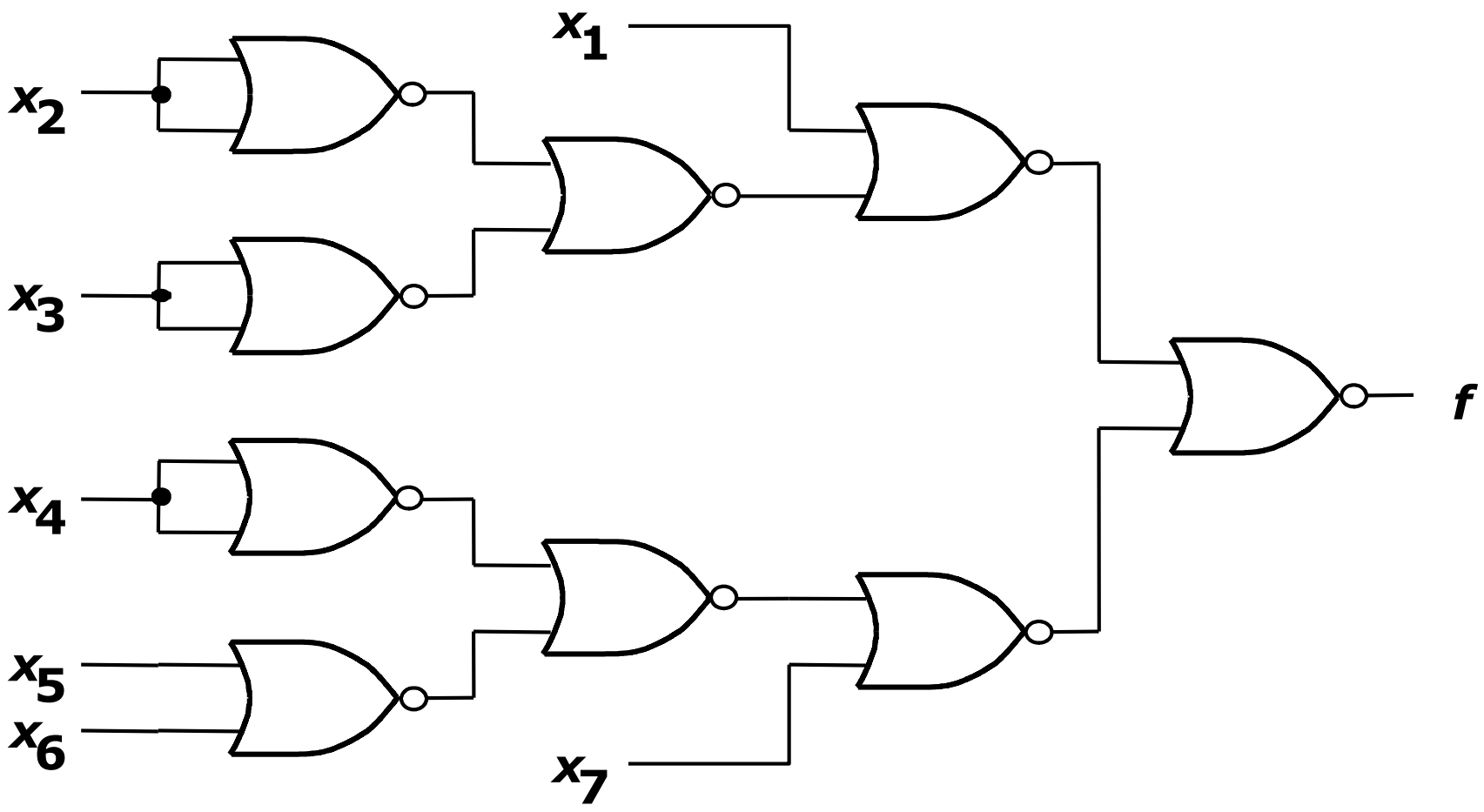
Prikaz potiskanja mehurčka
(ang. pushing the bubble)



KNO➔ Večnivojski (NOR-NOR)



KNO➔ Večnivojski (NOR-NOR)



Preostale 2-nivojske oblike

Dani nabor operatorjev: AND, OR, NAND, NOR.

AND/OR (DNO)	OR/AND (KNO)
NAND/NAND (SNO)	NOR/NOR (PNO)
NOR/OR	NAND/AND
OR/NAND	AND/NOR

2-nivojske oblike iz DNO

AND/OR (DNO)
NAND/NAND (SNO)
NOR/OR
OR/NAND

$$f(x_1, x_2, x_3) = x_1 + x_2 \cdot x_3'$$

2-nivojske oblike iz DNO

AND/OR ali (DNO)

$$f(x1; x2; x3) = x1 + x2 \cdot x3'$$

NAND/NAND ali (SNO): dvakrat negiramo celotno logično funkcijo in uporabimo DeMorgan-ov teorem

NOR/OR - dvakrat negiramo samo konjunkcije prvega nivoja in uporabimo DeMorgan-ov teorem

$$f(x1, x2, x3) = x1 + (x2\ x3')''$$
$$f(x1, x2, x3) = x1 + (x2' + x3)'$$
$$= x1 + (x2' \downarrow x3)$$

OR/NAND - dvakrat negiramo celotno logično funkcijo in uporabimo DeMorgan-ov teorem

$$f(x1, x2, x3) = x1 + x2 \cdot x3' = (x1 + x2 \cdot x3')''$$
$$= x1' \uparrow (x2' + x3)$$

2-nivojske oblike iz KNO

OR/AND (KNO)
NOR/NOR (PNO)
NAND/AND
AND/NOR

$$f(x_1, x_2, x_3) = x_2 \cdot (x_1' + x_2' + x_3)$$

2-nivojske oblike iz KNO

NOR/NOR ali (PNO) - dvakrat negiramo celotno logično funkcijo in uporabimo DeMorgan-ov teorem

NAND/AND - dvakrat negiramo samo disjunkcije prvega nivoja in uporabimo DeMorgan-ov teorem

$$f(x1, x2, x3) = x2 \cdot (x1' + x2' + x3)'' = x2 \cdot (x1x2x3')'$$
$$= x2 \cdot (x1 \uparrow x2 \uparrow x3')$$

AND/NOR - dvakrat negiramo celotno logično funkcijo in uporabimo DeMorgan-ov teorem

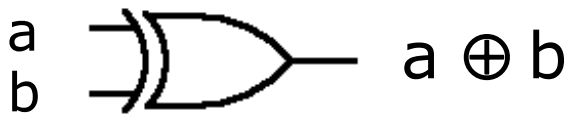
$$f(x1, x2, x3) = (x2 \cdot (x1' + x2' + x3))''$$
$$= (x2' + (x1' + x2' + x3)')' = x2' \downarrow (x1' + x2' + x3)'$$
$$= x2' \downarrow (x1x2x3')$$

XOR, AND, 1

V obliki DNO:

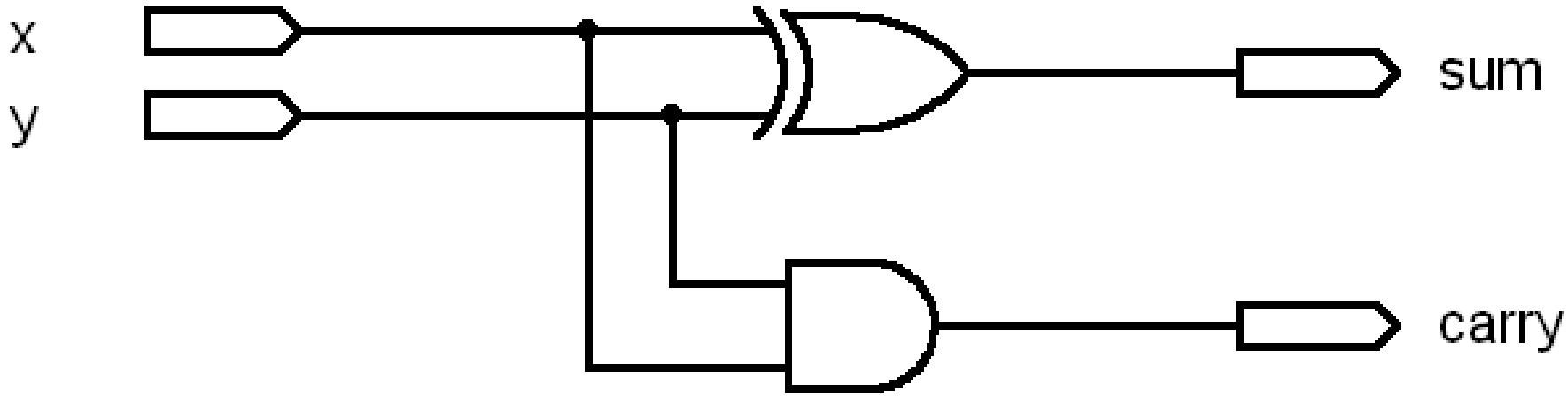
$f(a,b) = a \text{ XOR } b = a \oplus b = a \cdot b' + a' \cdot b$

a	*b*	a ⊕ b
0	0	0
0	1	1
1	0	1
1	1	0



Zgled uporabe XOR vezij

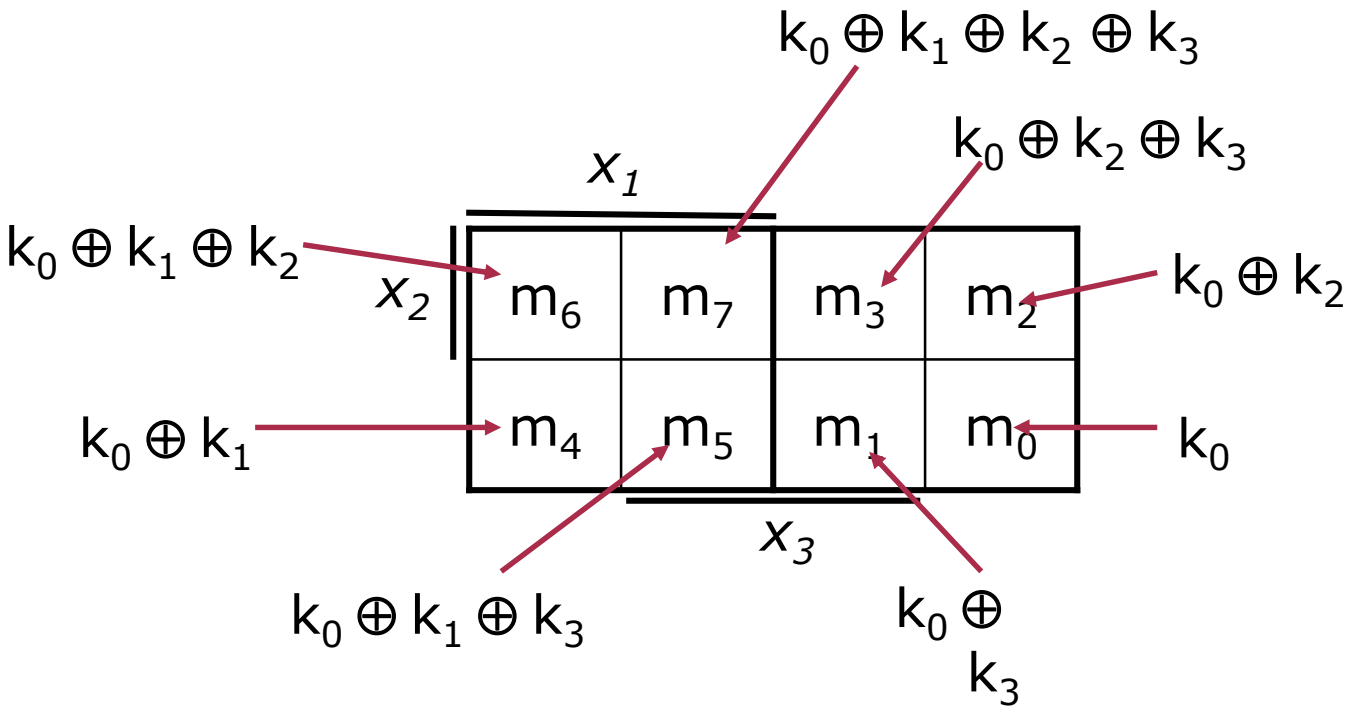
- Vezje polovičnega seštevalnika (HA)
  - sum=x·y’+x’·y
  - carry=x·y



Linearnost funkcije

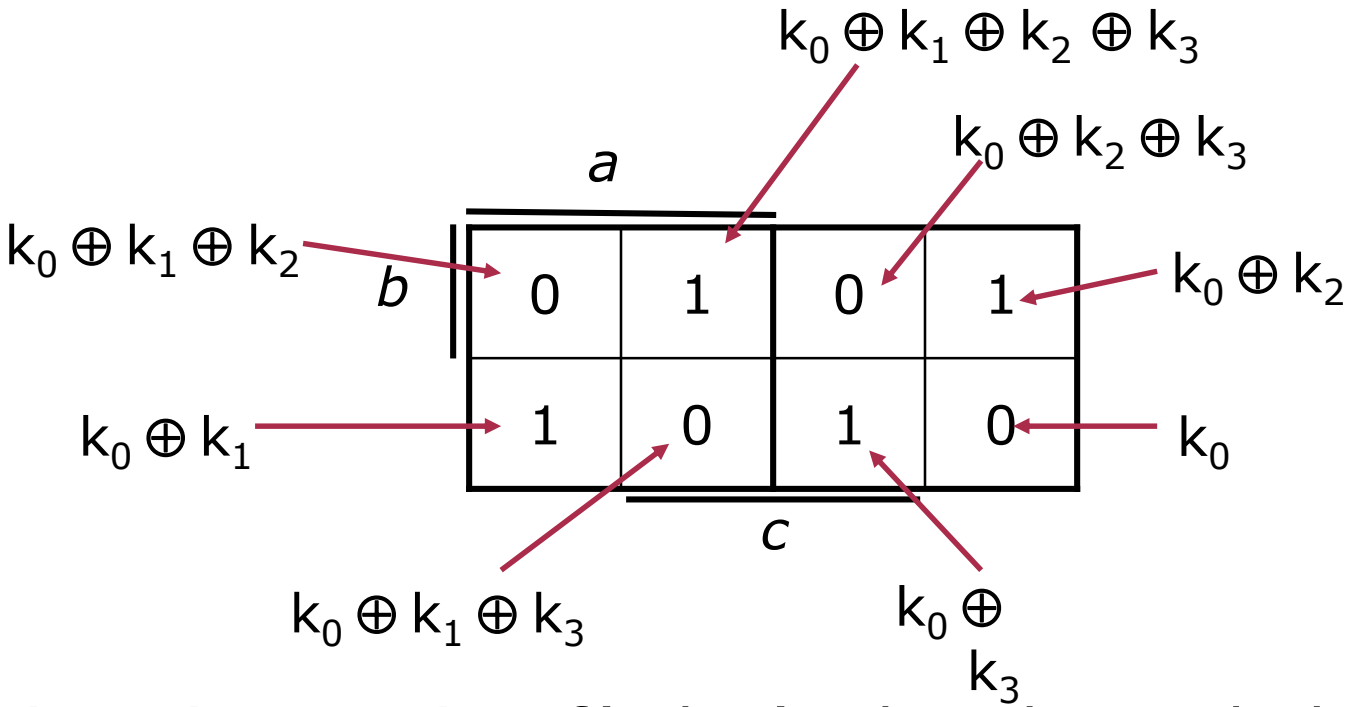
$f(x_1,x_2,x_3 \ldots x_n)=k_0 \oplus k_1x_1 \oplus k_2x_2 \oplus k_3x_3 \ldots \oplus k_nx_n$

Koeficienti $k_0, k_1 \ldots k_n$ so lahko 0 ali 1.



XOR treh spremenljivk

- Kakšna je PDNO oblika naslednjega izraza?
  $f(a, b, c) = a \oplus b \oplus c$

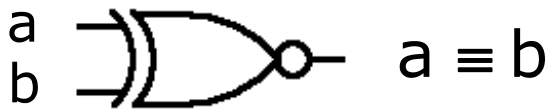


$f(a,b,c) = V(1,2,4,7)$ $f(a,b,c) = k_0 \oplus k_1\, a \oplus k_2\, b \oplus k_3\, c$

Ekvivalenca - Exclusive NOR (XNOR)

- Izvedena iz XOR funkcije: Predstavlja komplement XOR funkcije – vendar samo za 2 spremenljivki.
- XNOR se označuje s ≡ simbolom
  f =a ≡ b = a EQU b = a XNOR b
- V obliki DNO:
  f =a ≡ b = (a ⊕ b)′ = a·b+a′·b′

a	*b*	a ≡ b
0	0	1
0	1	0
1	0	0
1	1	1



XNOR treh spremenljivk

- Kakšna je PDNO oblika naslednjega izraza?
  $f(a, b, c) = a \equiv b \equiv c = (a \equiv b) \equiv c$

a	b	c	(a ≡ b)	f
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

$f(a,b,c) = V(1,2,4,7)$

Razvoj digitalnih sistemov

Realizacija optimalnih logičnih funkcij:

Večnivojska sinteza in analiza

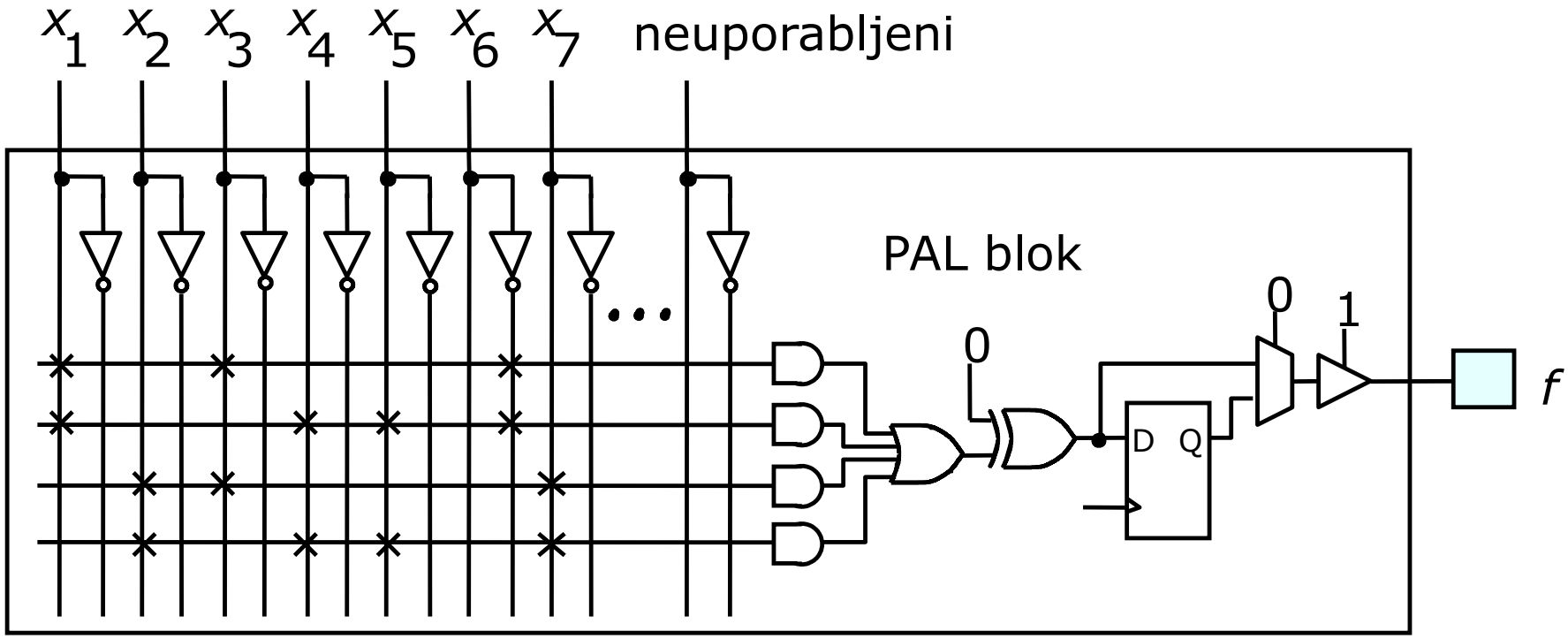
Večnivojska sinteza

- Število vhodov narašča ➔ realizacija privede do problemov s ***fan-in***
- **Fan-in**: Število vhodov v določena logična vrata ali komponento vezja

Večnivojska sinteza

- Določite minimalno COST funkcijo za izraz

  $f(x_1\ldots,x_7)=x_1x_3x_6'+x_1x_4x_5x_6'+x_2x_3x_7+x_2x_4x_5x_7$



Večnivojska sinteza

- Da bi problem realizacije funkcije *f* z FPGA rešili, moramo funkcijo zapisati v obliki, ki ima več nivojev logičnih operacij.
  - Taki obliki rečemo **večnivojski logični izraz** (*multilevel logic expression*)
- Metode za sintezo večnivojskih logičnih izrazov:
  - ***Faktorizacija***
    preoblikovanje dvonivojske logične funkcije v večnivojsko tako, da določi potencialno skupne izraze ali podfunkcije.
  - ***Dekompozicija***
    (nadomesti eno logično funkcijo z množico novih izrazov)
  - ***Ekstrakcija, Substitucija, Razpad***

Faktorski zapis funkcije

$f(x_1,\ldots,x_7)=x_1x_3x_6'+x_1x_4x_5x_6'+x_2x_3x_7+x_2x_4x_5x_7$

$f(x_1,\ldots,x_7)=x_1x_3x_6'+x_2x_3x_7+x_1x_4x_5x_6'+x_2x_4x_5x_7$

$f(x_1,\ldots,x_7)=x_3\cdot(x_1x_6'+x_2x_7)+x_4x_5\cdot(x_1x_6'+x_2x_7)$

Faktorska oblika (sekvenca 2-nivojskih izrazov)

$f(x_1,\ldots,x_7)=(x_1x_6'+x_2x_7)\cdot(x_3+x_4x_5)$

Problemi s fan-in

- Imamo funkcijo
  - $f=x_1x_2'x_3x_4'x_5x_6+ x_1x_2x_3'x_4'x_5'x_6$
- Direktna (2-nivojska) realizacija te funkcije
  COST = (3, 14) oziroma 2 AND6, 1 OR2 in 14 vhodov
- Faktorizacija funkcije v obliko
  $f= (x_1 \cdot x_4' \cdot x_6) \cdot (x_2'x_3x_5+ x_2x_3'x_5')$
  COST = (5, 13) oziroma 3 AND3, 1 OR2, 1 AND2 in 13 vhodov

Problemi s fan-in

- Faktorizirajte izraz tako, da bo realizacija zahtevala samo AND2 in OR2 vrata.

$f(x_1,\ldots,x_7)=x_1x_2'x_4'x_5+x_1x_2'x_6x_7'+x_3'x_4'x_5+x_3'x_6x_7'$

$f(x_1,\ldots,x_7)=x_1x_2'\cdot(x_4'x_5+x_6x_7')+x_3'\cdot(x_4'x_5+x_6x_7')$

$f(x_1,\ldots,x_7)=(x_1x_2' +x_3')\cdot(x_4'x_5+x_6x_7')$

COST = (6, 12) oziroma 4 AND2 in 2 OR2 vrata in 12 vhodov

Vplivi na zapletenost ožičenja

- Prostor v integriranem vezju zavzema:
  - Vezja logičnih vrat
  - Povezave med posameznimi vrati
- V logičnem izrazu vsaka spremenljivka ustreza eni povezavi v vezju, preko katere se prenaša logični signal
- Faktorizacija zmanjšuje število spremenljivk, zato zmanjša kompleksnost ožičenja logičnega vezja
- Med logično sintezo CAD orodja upoštevajo parametre kot so: COST (število vrat), fan-in, hitrost (zakasnitve) vezja in zapletenost ožičenja

Dekompozicija funkcije

- Podana funkcija
  - $f(w, x, y, z)=x\cdot y\cdot w'+x'\cdot z+y'\cdot z$
- Direktna realizacija:

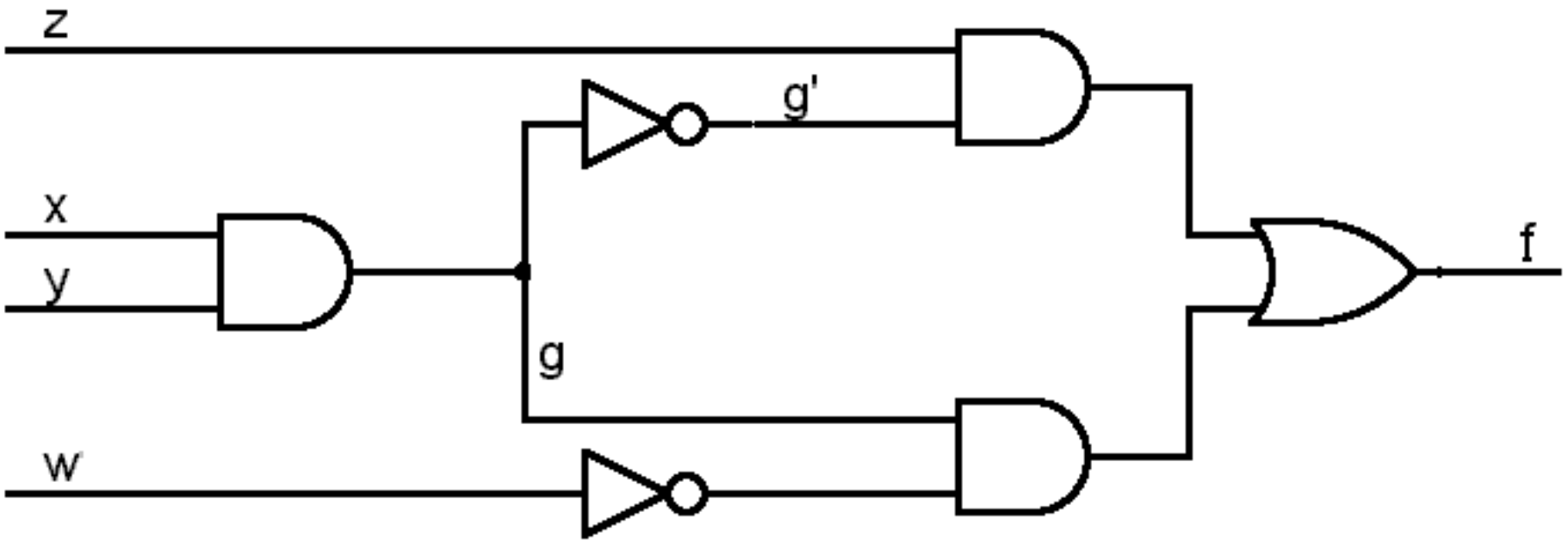
  1 AND3 vrata, 2 AND2 vrati in 1 OR3 vrata.
  COST= 4 vrata + 10 vhodov = 14
- Funkcijo f zapišemo v naslednjo obliko
  - $f(w, x, y, z)=(x\cdot y)\cdot w'+(x'+y')\cdot z$
- Označimo $g(x, y)=x\cdot y$ ➔ $g'=x'+y'$

Zgled dekompozicije

- Funkcija f postane
  - f(g,w,z)=g·w’+g’·z

Brez realizacije g(x,y) ➔ COST=(3, 6) = 9

Če upoštevamo še g(x,y) ➔ COST = (4, 7) = 11



Ekstrakcija funkcije

Ekstrakcija je uporabna v primeru, ko realiziramo hkrati več funkcij (recimo F, G, H). Z ekstrakcijo poiščemo podfunkcije (recimo X in Y), ki so skupne funkcijam (F, G, H). Primer:

$F=(x_1+x_2)\cdot x_3\cdot x_4+x_5$ ➔ COST = (3, 7)

$G=(x_1+x_2)\cdot x_5'$ ➔ COST = (2, 4)

$H=x_3\cdot x_4\cdot x_5$ ➔ COST = (1, 3)

Če izluščimo (ekstrahiramo) skupni funkciji:

$X=x_1+x_2$ ➔ COST=(1,2) in $Y=x_3\cdot x_4$ ➔ COST=(1,2):

$F=X\cdot Y+x_5$	$G=X\cdot x_5'$	$H=Y\cdot x_5$
COST = (2,3)	COST = (1,2)	COST = (1,2)
ΣCOST = (6,14)	:	ΣCOST = (6,11)

Substitucija in razpad funkcije

Substitucija je uporaba nove funkcije G v funkciji F. Primer:

$F(x_1,x_2,x_3)=x_1+x_2\cdot x_3=(x_1+x_2)\cdot(x_1+x_3)= G\cdot(x_1+x_3)$
kjer za novo funkcijo določimo $G=(x_1+x_2)$

Razpad je obraten substituciji. Primer:

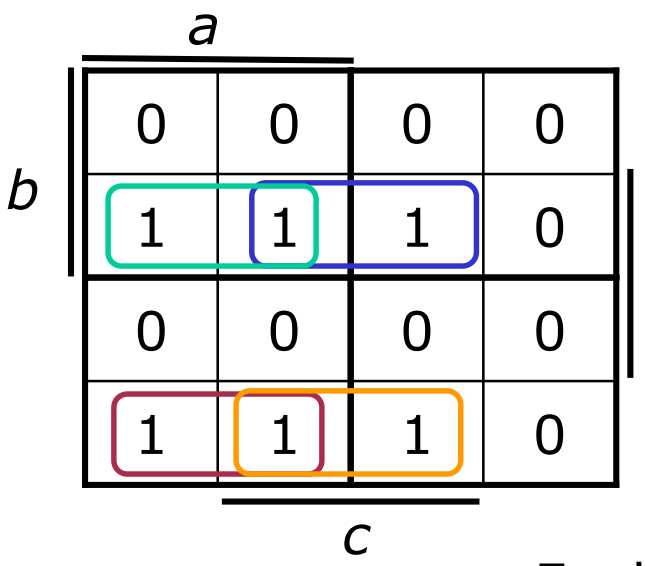
$F(x_1,x_2,x_3)=(x_1+x_2)\cdot(x_1+x_3)$

$F=x_1\cdot x_1+x_1\cdot x_3+x_1\cdot x_2+x_2\cdot x_3$

$F(x_1,x_2,x_3)=x_1+x_2\cdot x_3$

Zgled večnivojske funkcije

Funkcijo $f^4=V(2,7,8,10,13,15)$ zapišite v večnivojski obliki.



$f(a,b,c,d)_{MDNO}=abd+ab'd'+bcd+b'cd'$
COST = (5, 16)

$f(a,b,c,d)=abd+ab'd'+bcd+b'cd'$

$f(a,b,c,d)=a(b\cdot d+b'\cdot d')+c(b\cdot d+b'\cdot d')$

$f(a,b,c,d)=(a+c)\cdot(b\cdot d+b'\cdot d')$

$F_1=b\cdot d$
COST = (1, 2)

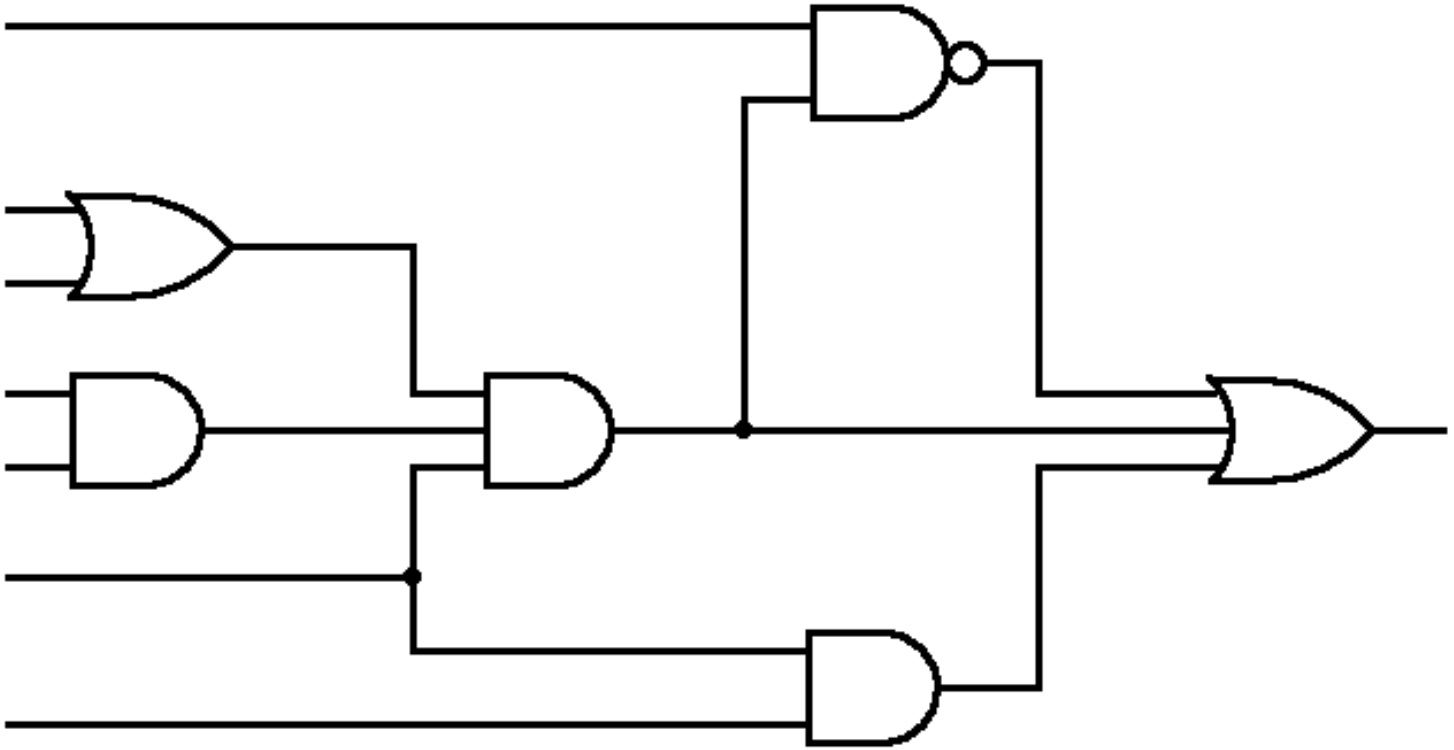
$F_2=b'\cdot d'$
COST = (1, 2)

$F_3=(a+c)$
COST = (1, 2)

$f(a,b,c,d)= F_3\cdot(F_1+F_2)$
COST = (2, 4)

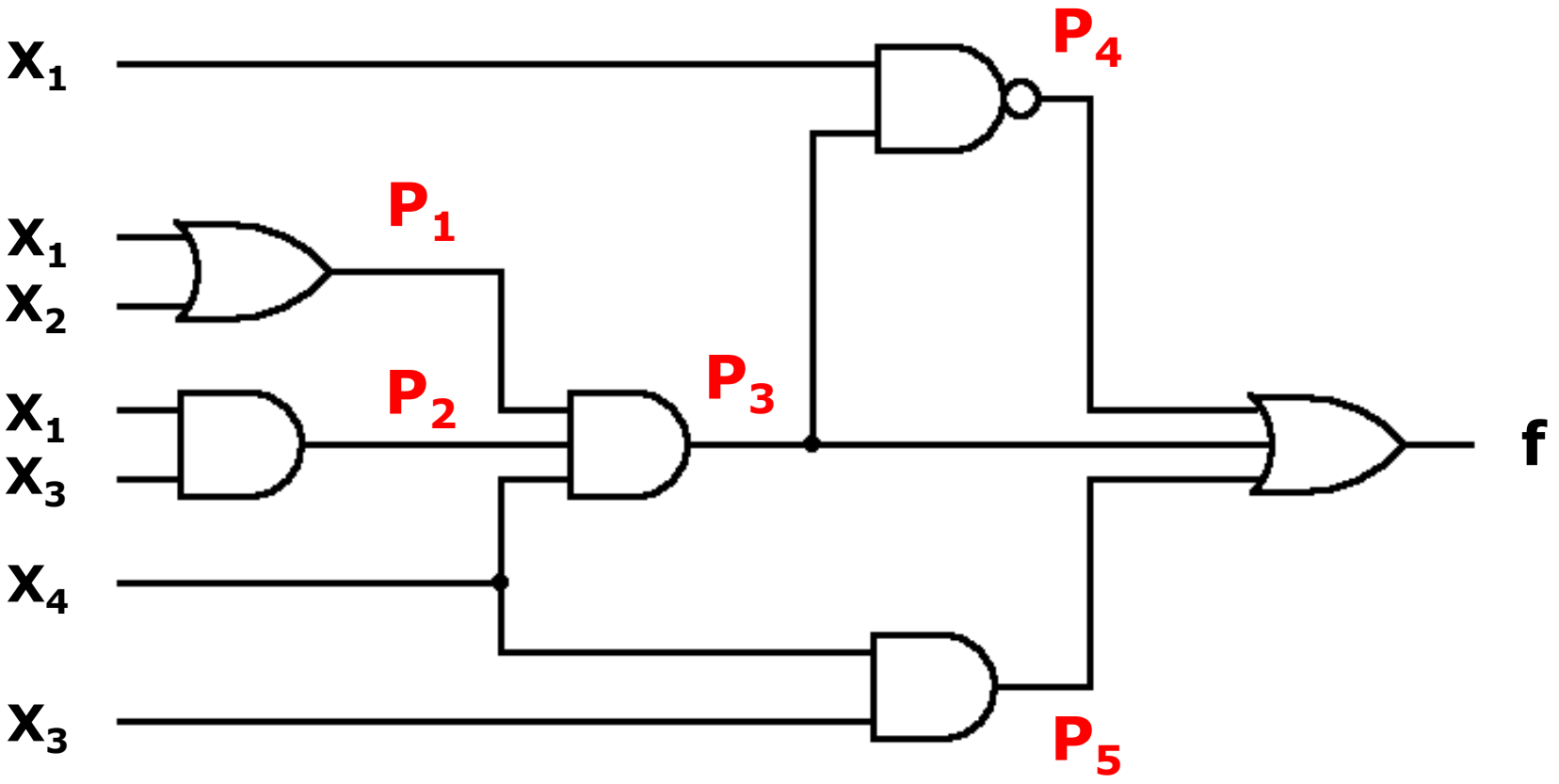
Zgled večnivojske pretvorbe

- Poiščite ekvivalentno NAND vezje spodnjega vezja:



Analiza večnivojskih vezij

Označimo izhod vseh vrat kot podfunkcijo



Analiza večnivojskih vezij

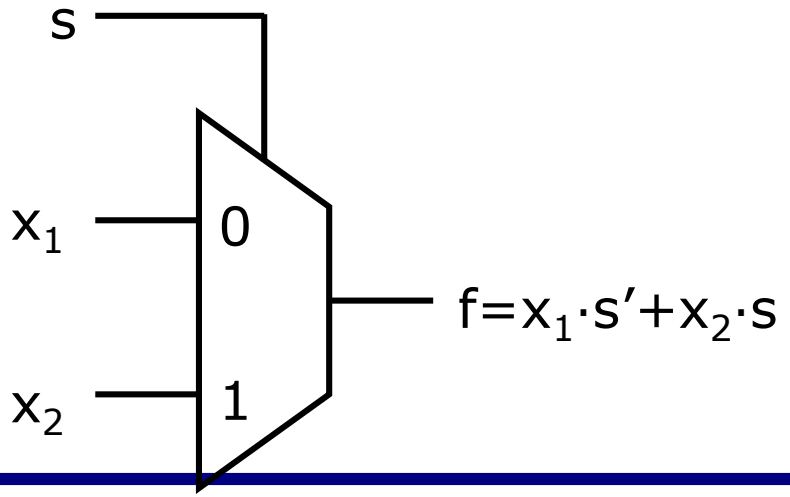
- Združimo (in po možnosti poenostavimo) dobljene podfunkcije (P)
  - $P_1=x_1+x_2$
  - $P_2=x_1x_3$
  - $P_3=P_1P_2x_4=(x_1+x_2)x_1x_3x_4=x_1x_3x_4+x_2x_1x_3x_4 = x_1x_3x_4$
  - $P_4=(x_1P_3)'=(x_1x_1x_3x_4)'= (x_1x_3x_4)'= x_1'+x_3'+x_4'$
  - $P_5=x_3x_4$
  - $f=P_3+P_4+P_5= \underbrace{(x_1x_3x_4)}_{A}+\underbrace{(x_1x_3x_4)'}_{A'}+(x_3x_4)=1$

Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:
Izbiralniki (multiplekserji)

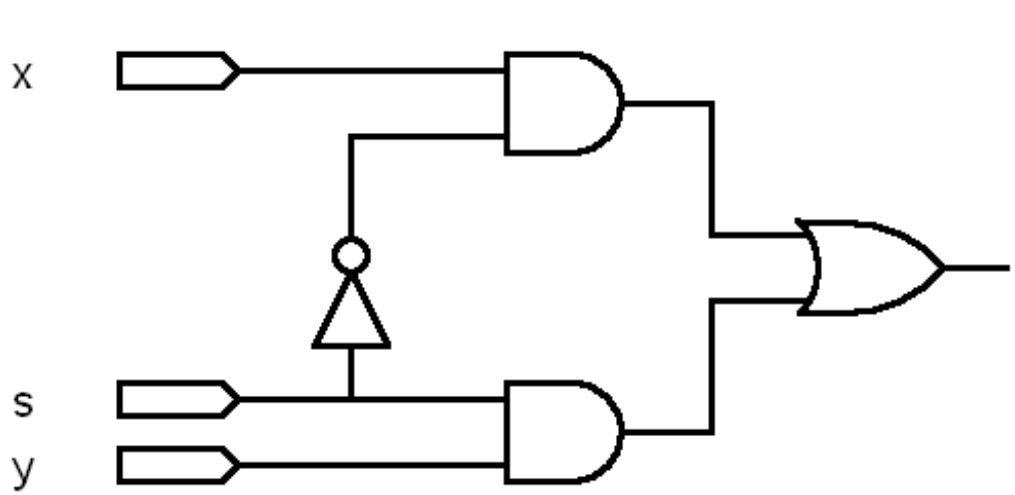
Izbiralniki

- Vezje izbiralnika oz. multiplekserja (MUX) ima
  - Nekaj podatkovnih vhodov (n)
  - Enega (ali več) izbirnih vhodov (2^n)
  - En izhod (f)
- Signal na enem od vhodov *(x_1, x_2)* pošlje na svoj izhod *(f)*.
- Kateri vhod je trenutno izbran določajo izbirni vhodi (ang. *select* ali *s*)

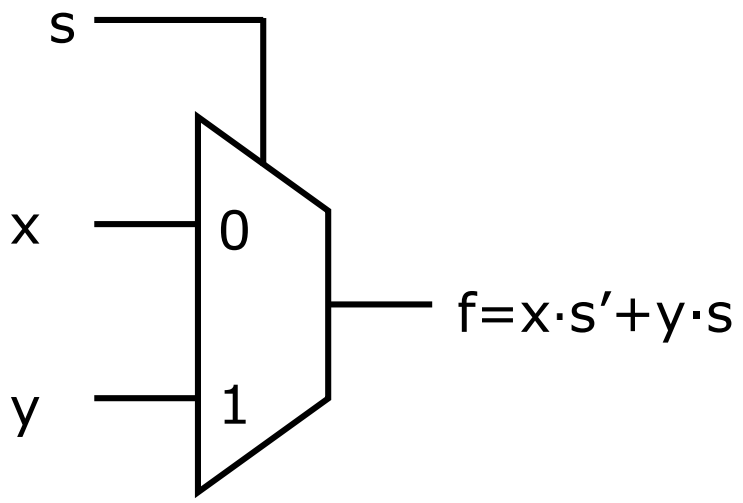


s	$f(s, x_1, x_2)$
0	x_1
1	x_2

Izvedbe multiplekserjev

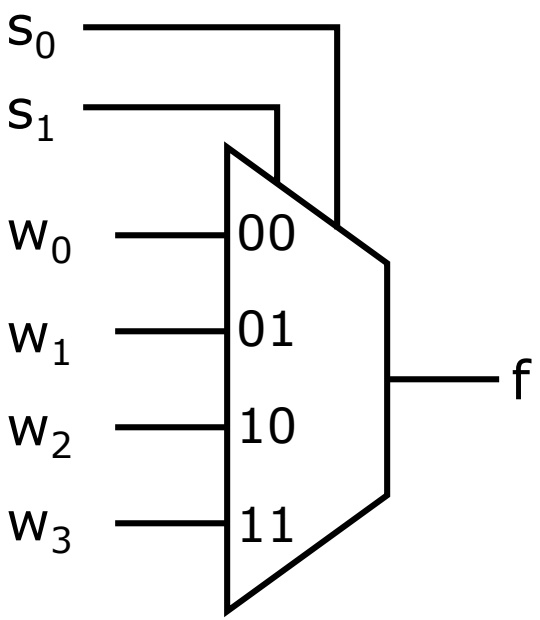


Osnovna izvedba



4/1 izbiralnik

- 4-vhodni multiplekser (4/1 MUX) na izhodu f 'izbira' enega od 4 podatkovnih vhodov (w_0, w_1, w_2, w_3). Kateri vhod je trenutno izbran določa stanje 2 izbirnih linij (s_1, s_2)

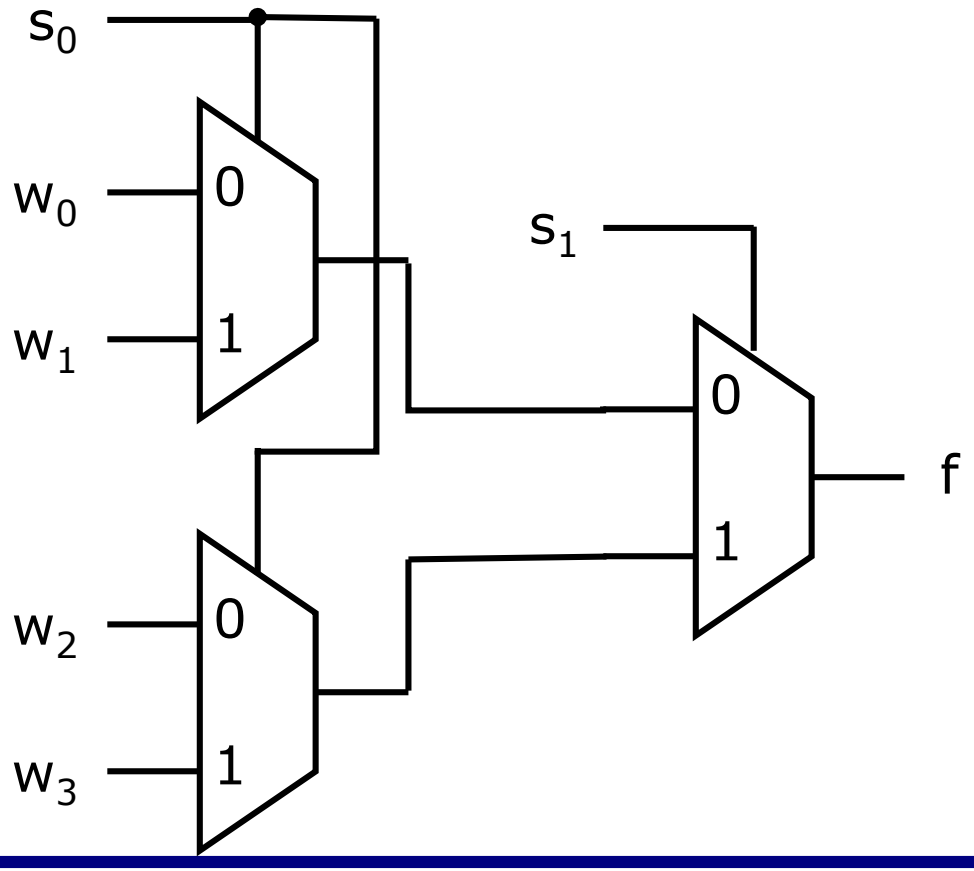


s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

$$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$$

Načrtovanje 4/1 izbiralnika

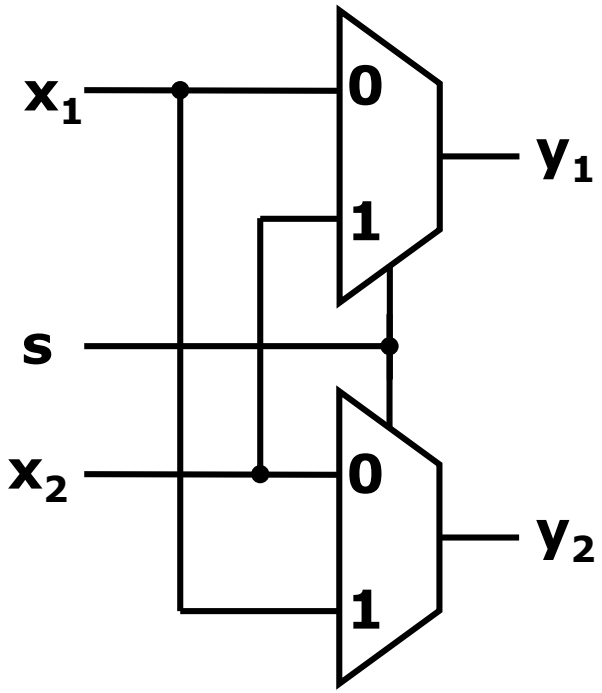
- Izdelamo ga s pomočjo treh 2/1 izbiralnikov

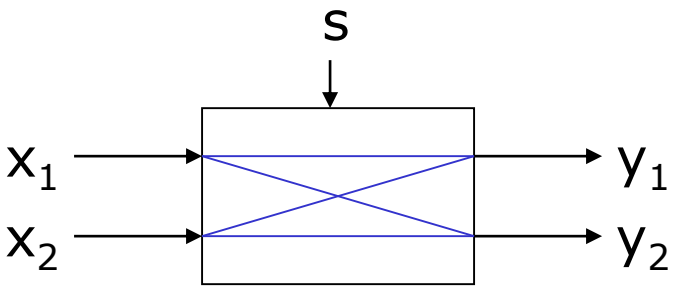


s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

Uporaba MUX: križno stikalo

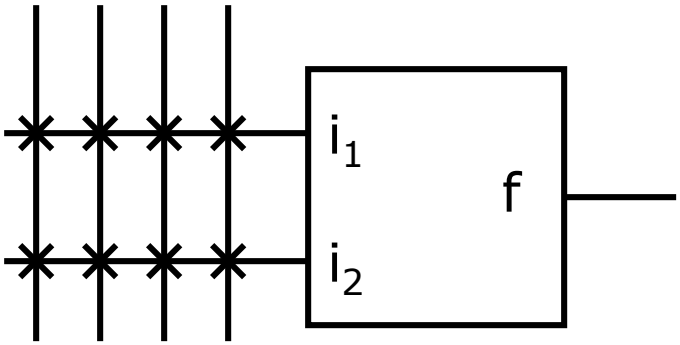
- Vezje z *n* vhodi in *k* izhodi
- Poveže katerikoli vhod na katerikoli izhod
- To je n x k križno stikalo (***n x k crossbar switch***)



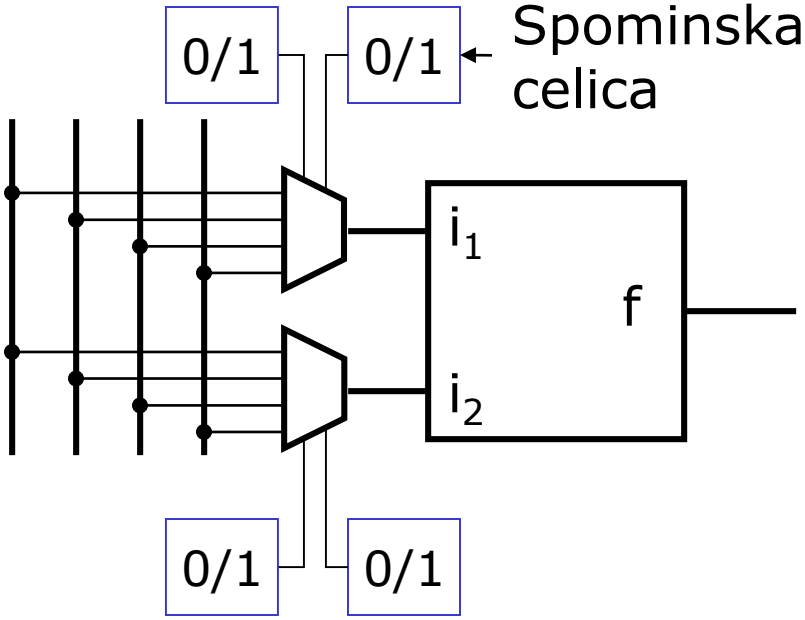


Uporaba MUX: Programabilno stikalo

- V programabilnih vezjih (PLD, CPLD in FPGA) programabilna stikala povezujejo povezave znotraj vezja
  - Tovrstna stikala so izvedena z izbiralniki



Logični blok vezja FPGA
s programabilnimi vhodi



Izvedba MUX

Izvedba logičnih funkcij z MUX

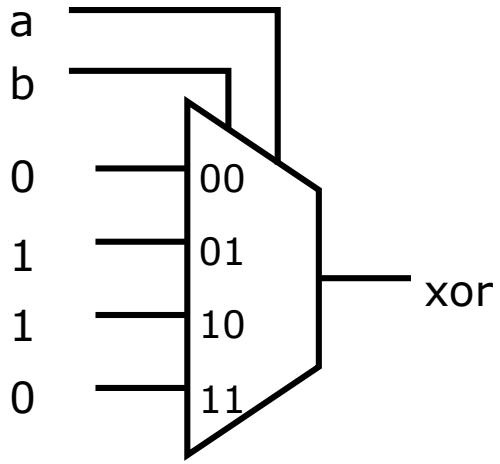
- MUX lahko uporabljamo pri sintezi (tvorbi) logičnih funkcij
  - Vpogledne tabele (LUT) so izvedene z MUX, s katerimi izbiramo (konstantno) vrednost iz vpogledne tabele.
- Vzemimo primer XOR in EQU (XNOR) funkcij

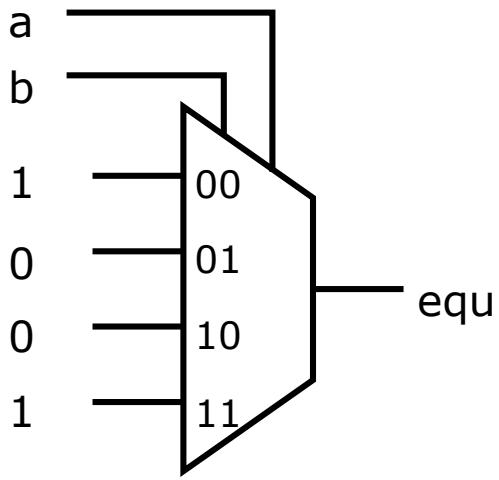
n	**naslovni vhodi**	**podatkovni vhodi**
r	*r* spremenljivk	0, 1 (t.i. trivialna realizacija)
r+1	*r* spremenljivk	funkcije preostale spremenljivke: $0, x_i, x'_i, 1$
r+2	*r* spremenljivk	funkcije preostalih dveh spremenljivk:
		$0, x_i, x'_i, x_j , x'_j , , ..., 1$

Izvedba logičnih funkcij z MUX

- MUX lahko uporabljamo pri sintezi (tvorbi) logičnih funkcij
  - Vpogledne tabele (LUT) so izvedene z izbiralniki, s katerimi izbiramo (konstantno) vrednost iz vpogledne tabele.
- Vzemimo primer XOR in EQU funkcij

a	*b*	*xor*	*equ*
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1



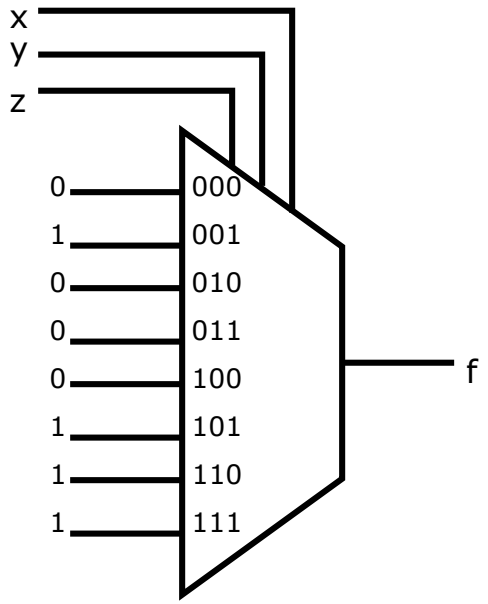


Trivialna realizacija (r=n)

Za funkcijo z n spremenljivkami vzamemo
r = n-naslovni multiplekser.

Primer: Realizacija funkcije
f(x, y, z)=V(1,5,6,7)

x	*y*	*z*	*f*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

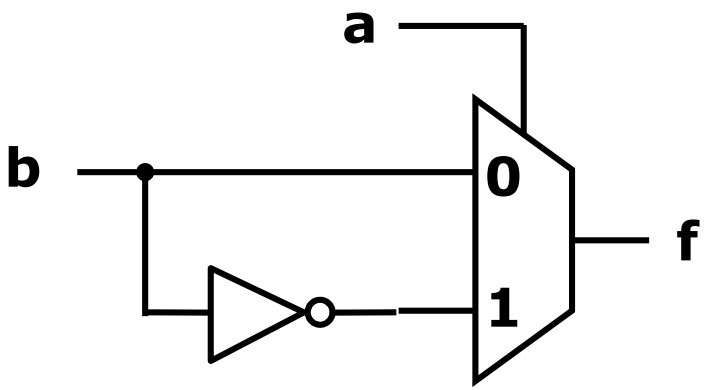


Izvedba z MUX (r=n-1) - primer

- Realizirajte funkcijo XOR dveh spremenljivk s pomočjo inverterjev in 2/1 izbiralnika.

a	*b*	*f*	
0	0	0	Ko je a=0, f=b
0	1	1	
1	0	1	Ko je a=1, f=b’
1	1	0	

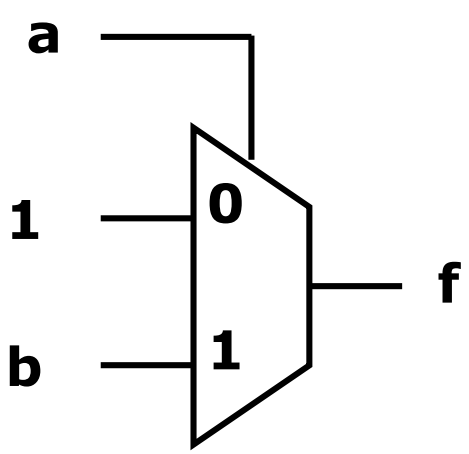
a	*f*
0	b
1	b’



Izvedba z MUX (r=n-1) - primer

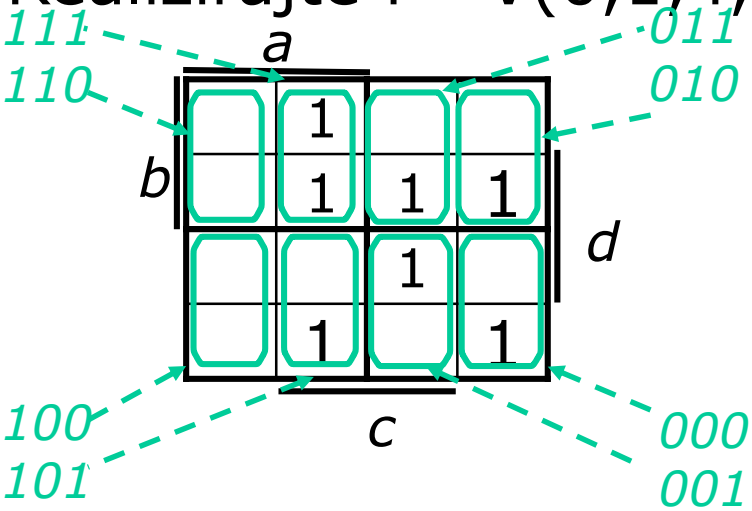
- Z uporabo 2/1 izbiralnikov in logičnih vrat realizirajte spodnjo funkcijo

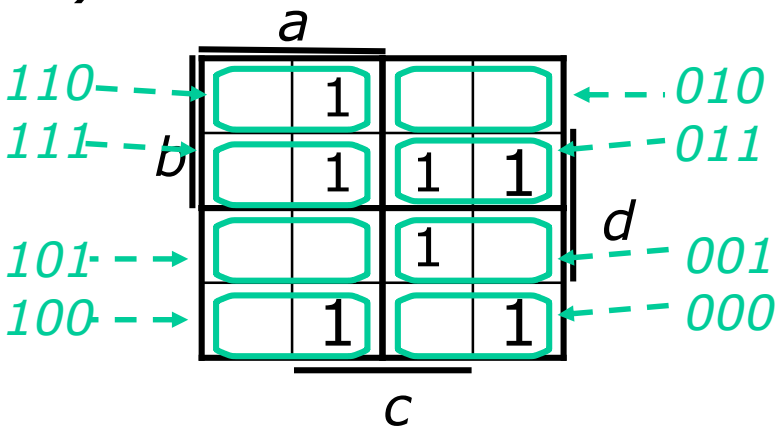
a	*b*	*f*
0	0	1
0	1	1
1	0	0
1	1	1

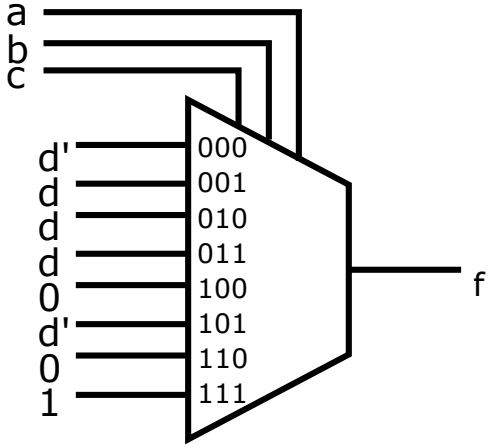


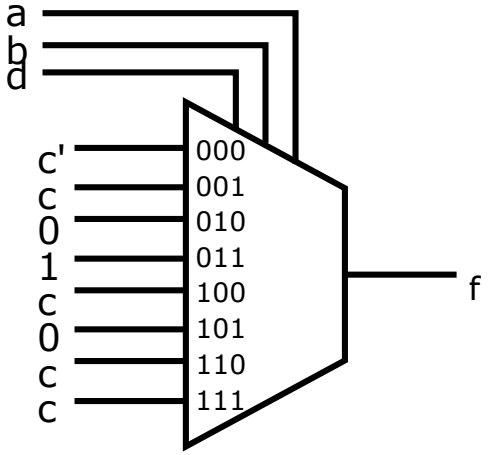
Izvedba funkcij z MUX (r=n-1)

Realizirajte $f^4=V(0,1,4,7,9,10,14)$.





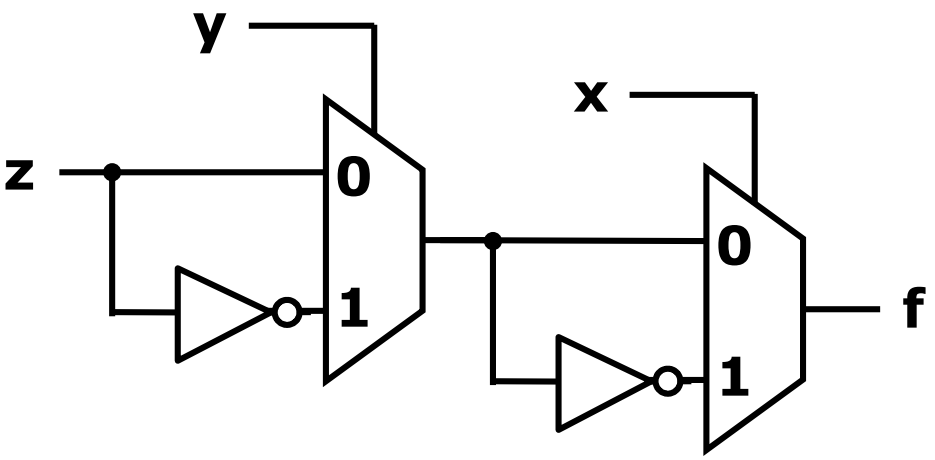




Izvedba funkcije z MUX (r=n-2) – primer

- Realizirajte 3-vhodna XOR vrata z uporabo 2/1 MUX

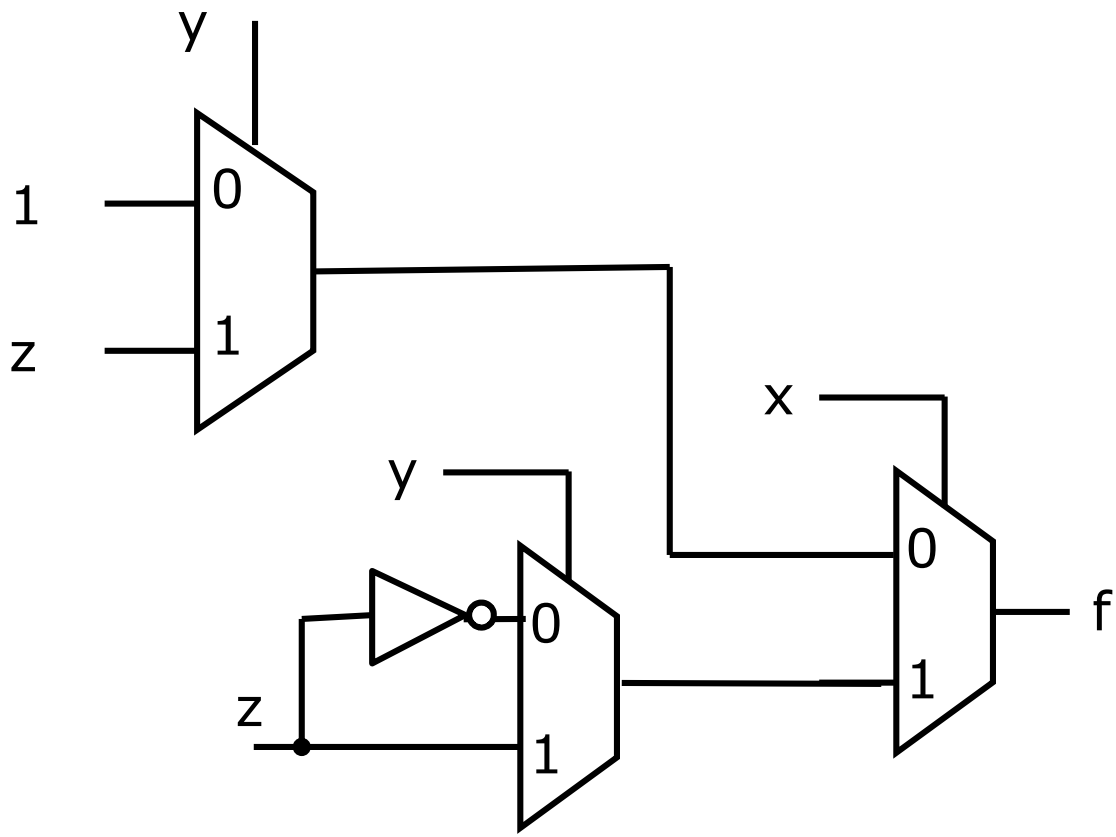
x	*y*	*z*	*f*	
0	0	0	0	y⊕z
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	(y⊕z)’
1	0	1	0	
1	1	0	0	
1	1	1	1	



Izvedba funkcije z MUX (r=n-2)

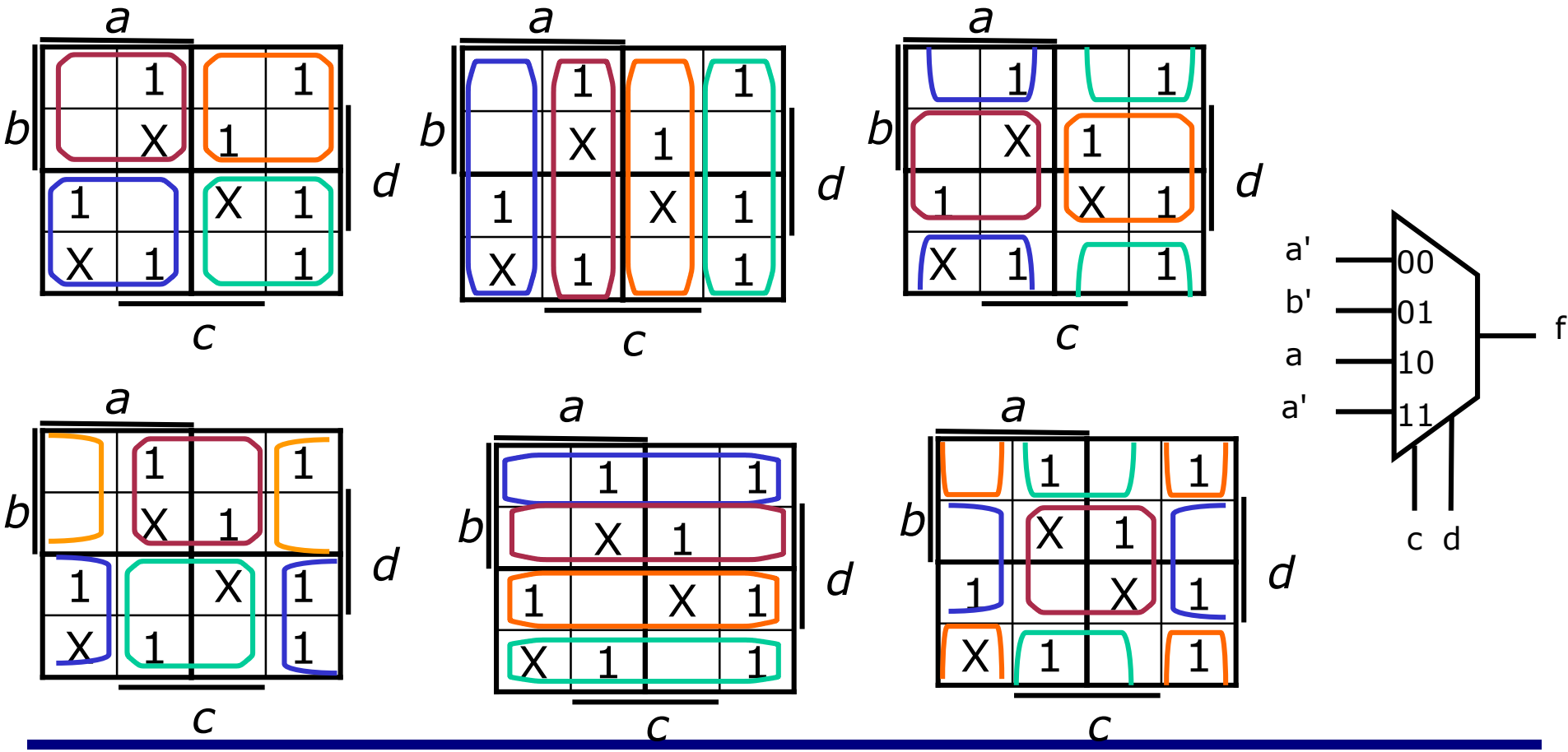
- Z uporabo 2/1 MUX in logičnih vrat realizirajte spodnjo funkcijo

x	y	z	f	
0	0	0	1	'1'
0	0	1	1	
0	1	0	0	z
0	1	1	1	
1	0	0	1	(y⊕z)'
1	0	1	0	
1	1	0	0	
1	1	1	1	



Izvedba funkcije z MUX (r=n-2) – primer

Realizirajte $f^4=V(0,1,4,7,9,10,14)$ in $V_x(3,8,15)$ z enim 4/1 izbiralnikom in inverterji.



Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:
Shannon-ov razvoj funkcije

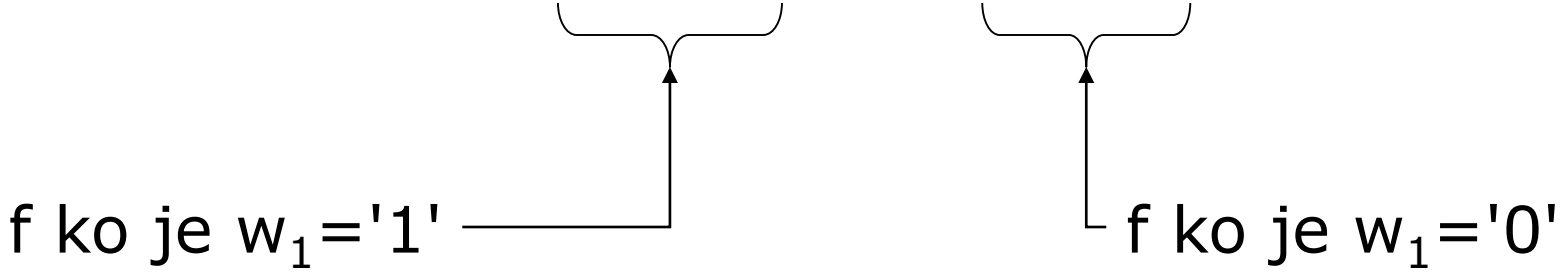
Shannon-ov razvoj funkcije

- Katerokoli Boolovo funkcijo $f(w_1,\ldots,w_n)$ lahko zapišemo v obliki

  $f(w_1,\ldots, w_n)=(w_1)'\cdot f(0,w_2,\ldots,w_n)+(w_1)\cdot f(1,w_2,\ldots,w_n)$

- Razvoj lahko naredimo po katerikoli izmed *n* spremenljivk funkcije
- Če je $f(w_1, w_2, w_3)= w_1\cdot w_2+w_1\cdot w_3+w_1'\cdot w_2\cdot w_3$
  - Razvoj po w_1 nam da rezultat

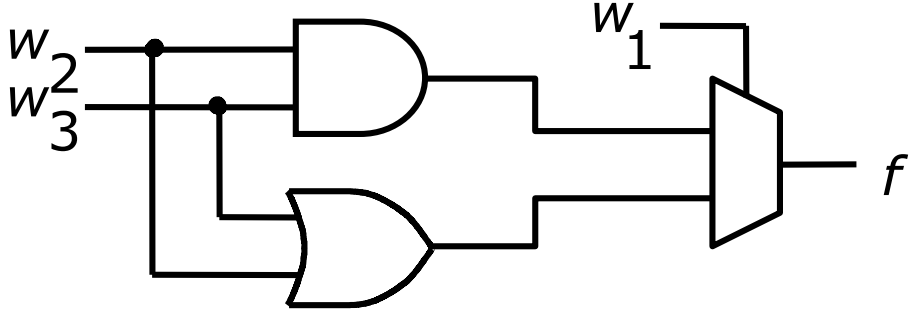
    $f(w_1, w_2, w_3)= w_1\cdot(w_2+w_3)+w_1'\cdot(w_2\cdot w_3)$



Zgled Shannon-ovega razvoja

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	$w_2 \cdot w_3$
1	$w_2 + w_3$



Zgled Shannon-ovega razvoja

$f(x, y, z) = x' \cdot y' \cdot z' + x' \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y' \cdot z' + x \cdot y' \cdot z$

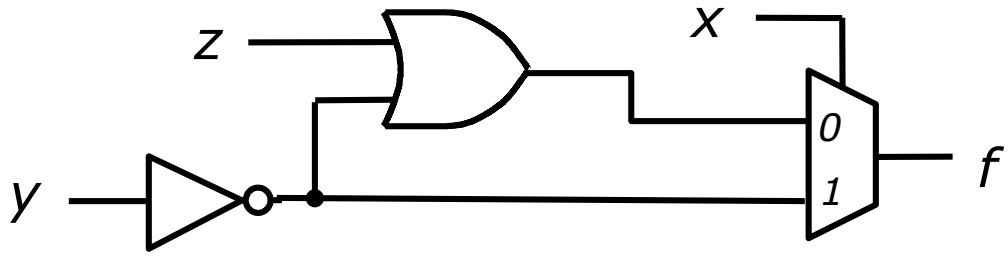
Če izberemo x kot spremenljivko razvoja

$f = x' \cdot (y' \cdot z' + y' \cdot z + y \cdot z) + x \cdot (y' \cdot z' + y' \cdot z)$

$f = x' \cdot (y' \cdot z' + y' \cdot z + y \cdot z + y' \cdot z) + x \cdot (y')$

$f = x' \cdot (y' \cdot (z' + z) + (y + y') \cdot z) + x \cdot (y')$

$f = x' \cdot (y' + z) + x \cdot (y')$



x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Zgled Shannon-ovega razvoja

$f(x, y, z) = x' \cdot y' \cdot z' + x' \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y' \cdot z' + x \cdot y' \cdot z$

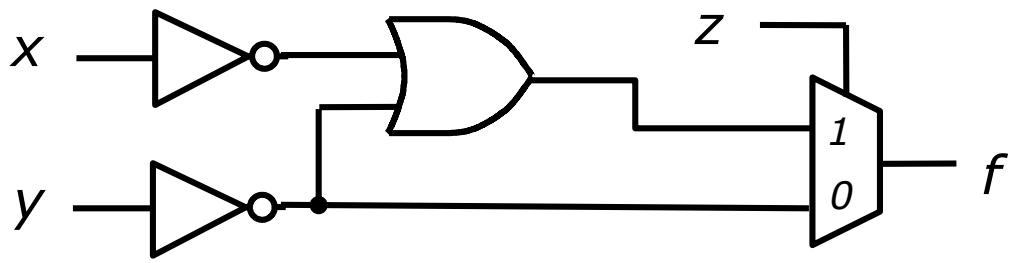
Če izberemo z kot spremenljivko razvoja

$f = z' \cdot (x' \cdot y' + x \cdot y') + z \cdot (x' \cdot y' + x' \cdot y + x \cdot y')$

$f = z' \cdot (x' \cdot y' + x \cdot y') + z \cdot (x' \cdot y' + x' \cdot y + x \cdot y' + x' \cdot y')$

$f = z' \cdot y'(x + x') + z \cdot (x' \cdot (y' + y) + (x' + x) \cdot y')$

$f = z' \cdot y' + z \cdot (x' + y')$



x	y	z	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Zgled Shannon-ovega razvoja

$f(x, y, z) = x' \cdot y' \cdot z' + x' \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y' \cdot z' + x \cdot y' \cdot z$

Če izberemo $x \cdot y$ kot spremenljivki razvoja

x	y	z	f	
0	0	0	1	F_0
0	0	1	1	
0	1	0	0	F_1
0	1	1	1	
1	0	0	1	F_2
1	0	1	1	
1	1	0	0	F_3
1	1	1	0	

$f = x' \cdot y' \cdot z' + x' \cdot y' \cdot z + x' \cdot y \cdot z + x \cdot y' \cdot z' + x \cdot y' \cdot z$

$f = x' \cdot y' \cdot (z' + z) + x' \cdot y \cdot z + x \cdot y' \cdot (z' + z)$

$f = x' \cdot y' \cdot (F_0) + x' \cdot y \cdot (F_1) + x \cdot y' \cdot (F_2)$

$f = x' \cdot y' \cdot (1) + x' \cdot y \cdot (z) + x \cdot y' \cdot (1)$

Kaskadna realizacija z MUX

$f(x_1,x_2,x_3,x_4,x_5,x_6)=x_1x_2x_3+x_1'x_2x_4'x_6'+x_1x_2'x_4x_5x_6$

Na I. nivoju izberemo x_1x_2 kot spremenljivko razvoja

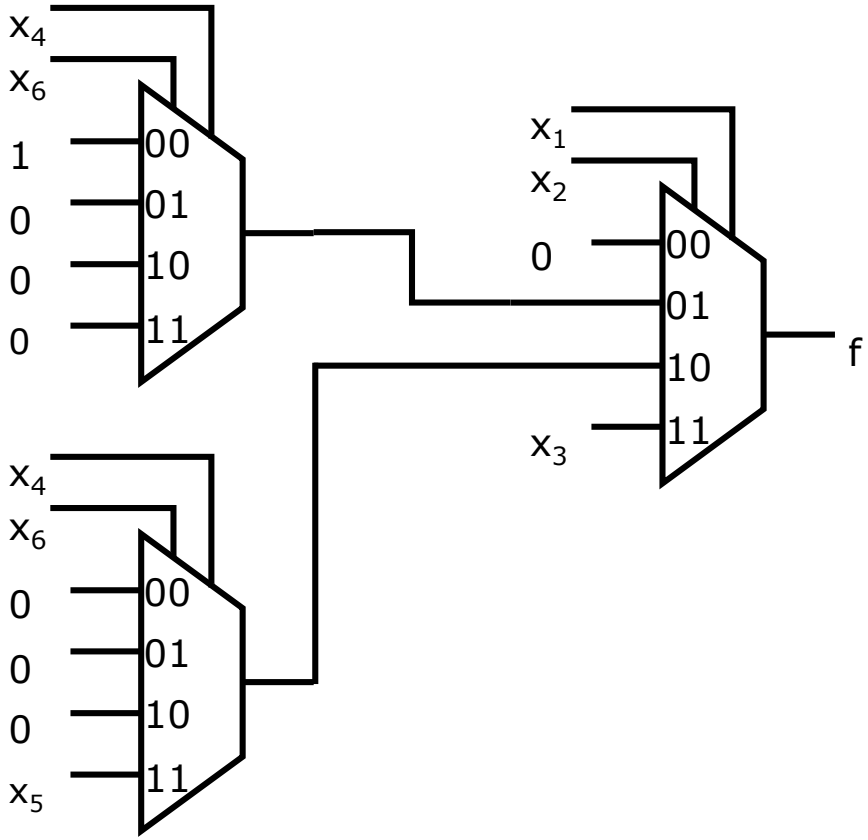
$f(x_1,x_2,x_3,x_4,x_5,x_6)=x_1x_2(x_3)+x_1'x_2(x_4'x_6')+x_1x_2'(x_4x_5x_6)$

Na II. nivoju izberemo x_4x_6 kot spremenljivko razvoja

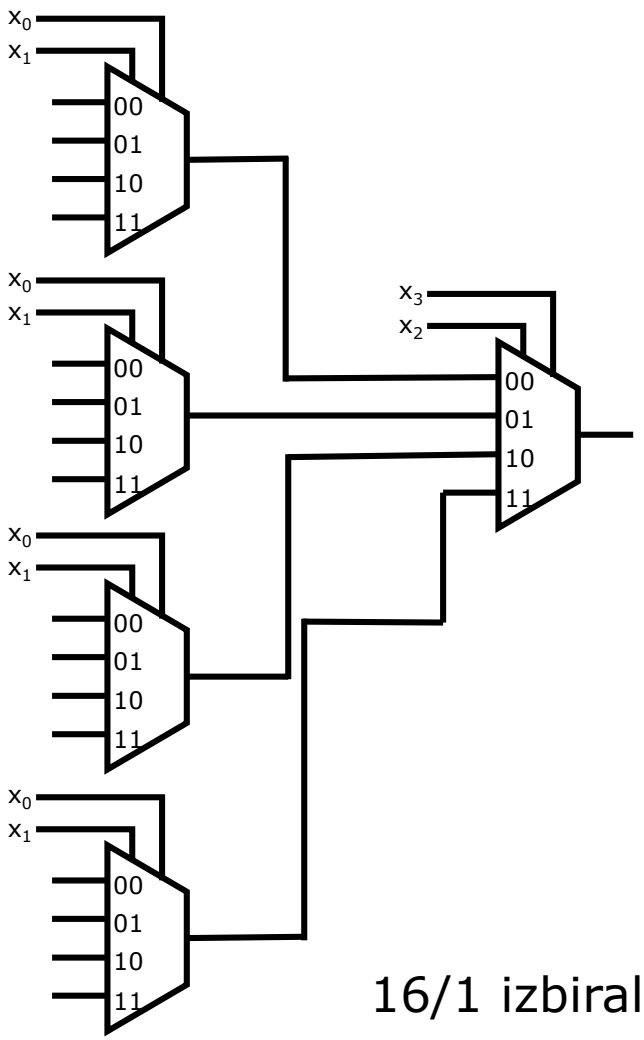
$f(x_1,x_2,x_3,x_4,x_5,x_6)=x_1x_2(x_3)+x_1'x_2(x_4'x_6')+x_1x_2'(x_4x_5x_6)$

Kaskadna realizacija z MUX

$f(x_1,x_2,x_3,x_4,x_5,x_6)=x_1x_2(x_3)+x_1'x_2(x_4'x_6')+x_1x_2'(x_4x_5x_6)$



Izvedba večjih MUX z manjšimi



16/1 izbiralnik:

Imamo več možnosti, katere od manjših izbiralnikov uporabiti.

MUX 8/1 je lahko iz:

- 4/1 MUX
- 2/1 MUX
- 4/1 MUX na višjem in 2/1 MUX na nižjem nivoju
- 2/1 MUX na višjem in 4/1 MUX na nižjem nivoju
- 8/1 v diskretni (MSI) izvedbi obstaja
- 32/1 ne obstaja v MSI obliki (imel bi 32+1+5+1+2 = 41 priključkov).

Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:
Izbiralniki v VHDL

Večbitna števila v VHDL (vektorji)

- Število v jeziku VHDL je večbitni podatkovni tip

```
SIGNAL C: STD_LOGIC_VECTOR (0 TO 3);
```

- C je 4-bitni STD_LOGIC signal
  - C je 4-bitna veličina
  - Prireditev vrednosti C <= "1110"; ali krajše C <= x"E";
  - Dostop do bitov: C(0) – 1-bitna veličina (LSB)
    C(3) – 1-bitna veličina (MSB)

- Zaporedje bitov lahko obrnemo

```
SIGNAL X:       STD_LOGIC_VECTOR (3 DOWNTO 0);
```

- X je 4-bitni STD_LOGIC signal
  - X(3) je MSB
  - X(0) je LSB

Sestavljanje večbitnih števil v VHDL

```
entity concat_demo is
Port( X : in  STD_LOGIC_VECTOR (2 downto 0);
   Q0, Q1, Q2 : out  STD_LOGIC_VECTOR (2 downto 0)
   );
end concat_demo;
architecture arch of concat_demo is
signal REG : STD_LOGIC_VECTOR(2 downto 0):="001";
begin
REG <= X;
Q0 <= REG(0) & REG(2 downto 1);
Q1 <= REG(0) & REG(1) & REG(2);
Q2 <= X;
end arch;
```

Konstante (CONSTANT)

Konstanta (**CONSTANT**)
nima strojnega ekvivalenta!
(služi lažjemu programiranju)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity const_demo is
Port ( x1 : in STD_LOGIC;
       x2 : in STD_LOGIC_VECTOR(3 downto 0);
       y1 : out STD_LOGIC;
       y2 : out STD_LOGIC_VECTOR(3 downto 0) );
end const_demo;
architecture arch of const_demo is
constant ENA : STD_LOGIC := '1';      -- enobitna konstanta
constant PETNAJST_2 : STD_LOGIC_VECTOR(3 downto 0) := "1111";-- bin
constant PETNAJST_16 : STD_LOGIC_VECTOR(3 downto 0) := x"F";-- hex
constant C15 : STD_LOGIC_VECTOR(3 downto 0) := (others => '1');-- hex
constant C1 : STD_LOGIC_VECTOR(3 downto 0) := (0 => '1', others =>
'0');-- splošna konstanta "0001"
begin
 y1 <= x1 xor ENA; -- y1 = not x1
 y2 <= x2 xor PETNAJST_16; -- y2 = not x2 (stiribitna operacija)
end arch;
```

Določitveni izrazi (Assignment statements)

- VHDL pozna nekaj tipov izrazov, ki jih lahko uporabimo za določanje logičnih vrednosti signalom
  - Enostavni določitveni izrazi
  - Izbirni izrazi signalov (with select izrazi)
  - Pogojni izrazi signalov (when … else izrazi)
  - If-then-else izrazi
  - Case izrazi

WHEN OTHERS (4/1 MUX)

```
ENTITY mux4to1 IS
PORT (     w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
           s : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
           f : OUT STD_LOGIC );
END mux4to1;
ARCHITECTURE arch OF mux4to1 IS
BEGIN
     WITH s SELECT
     f <= w(0) WHEN "00",
          w(1) WHEN "01",
          w(2) WHEN "10",
          w(3) WHEN OTHERS;
END arch;
```

WHEN ELSE (2/1 MUX)

```
ENTITY mux2to1 IS
PORT ( w0, w1, s : IN STD_LOGIC;
       f : OUT STD_LOGIC
     );
END mux2to1;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
     f <= w0 WHEN s = '0' ELSE w1;
END Behavior;
```

VHDL ARHITEKTURE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity mux2_to_1 is
Port (   x1, x2, s: in STD_LOGIC;
         f : out STD_LOGIC);
end mux2_to_1;
architecture arch1 of mux2_to_1 is
begin
with s select
         f <= x1 when '0',
              x2 when others;
end arch1;
architecture arch2 of mux2_to_1 is
begin
         f <= x1 when s = '0' else x2;
end arch2;
...
```

```
...
architecture arch3 of mux2_to_1 is
begin
process ( x1, x2 ,s)
begin
         if s = '0' then
                  f <= x1;
         else
                  f <= x2;
         end if;
end process;
end arch3;

architecture arch4 of mux2_to_1 is
begin
process ( x1, x2 , s)
begin
case s is
         when '0'    => f <= x1;
         when others => f <= x2;
end case;
end process;
end arch4;
```

Proces

- **Sočasni določitveni izrazi** (v logičnih enačbah) (*concurrent assignment statements)*
  - **Zaporedje** teh izrazov v VHDL kodi **ne vpliva** na dejanski pomen kode
- **Sekvenčni določitveni izrazi** (v procesnih stavkih) (*sequential assignment statements*)
  - Zaporedje teh izrazov v VHDL kodi *lahko vpliva* na dejanski pomen kode
  - If-then-else in Case izrazi so sekvenčni. VHDL zahteva, da se sekvenčni določitveni izrazi nahajajo znotraj procesnega stavka (***process***)
- Kombinacijski stavki in procesni stavki se izvajajo vzporedno.

Proces (MUX 2/1)

```
ARCHITECTURE arch OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
BEGIN
   IF s = '0' THEN
     f <= w0 ;
   ELSE
     f <= w1 ;
   END IF ;
END PROCESS ;
END arch ;
```

Seznam občutljivosti

IF-THEN-ELSE izraz ki realizira MUX funkcijo

Proces (MUX 2/1)

```
ARCHITECTURE arch OF mux2to1 IS
BEGIN
PROCESS ( w0, w1, s )
   BEGIN
      CASE s IS
         WHEN '0'       => f <= w0;
         WHEN OTHERS    => f <= w1;
      END CASE;
   END PROCESS;
END arch;
```

Proces – spremenljivke (VARIABLE)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity xor_v is
Port ( A, B, C : in STD_LOGIC;
       X,Y : out STD_LOGIC);
end xor_v;
architecture arch of xor_v is
begin
       process (A, B, C)
       variable D : STD_LOGIC;
       begin
              D := A; -- D se priredi vrednost A
              X <= C xor D; -- A se uporabi pri izracunu X=C⊕A
              D := B; -- D se priredi vrednost B
              Y <= C xor D; -- B se uporabi pri izracunu Y=C⊕B
       end process;
end arch;
```

Spremenljivka (VARIABLE) nima strojnega ekvivalenta! (služi lažjemu programiranju)

Vezana je na določen proces.

Ovrednoti se **SPROTI**.

Proces – signali (SIGNAL)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity xor3_s is
Port ( A, B, C : in STD_LOGIC;
               X,Y : out STD_LOGIC);
end xor3_s;
architecture arch of xor3_s is
signal D : STD_LOGIC;
begin
       process (A,B,C)
       begin
              D <= A;  -- se ne upošteva
              X <= C xor D;
              D <= B;        -- velja zadnja prireditev!
              Y <= C xor D; -- Y=C⊕B ter X=C⊕B
       end process;
end arch;
```

Signal ima strojni ekvivalent!
ŽICA.

Signal je "globalen".

WARNING:Xst:647 - Input <A> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.

Povratne zanke v VHDL - BUFFER

- Priključkov tipa OUT, ne moremo uporabiti kot vhode (IN) v isto ali drugo entiteto.
- Prva možnost implementacije povratnih zank (izhod je obenem vhod v funkcijo):
  - Izhoda ne deklariramo kot OUT ampak kot BUFFER
    C <= A or (B and C).

Žal morajo biti vse spremenljivke
na katere se veže C tipa BUFFER.

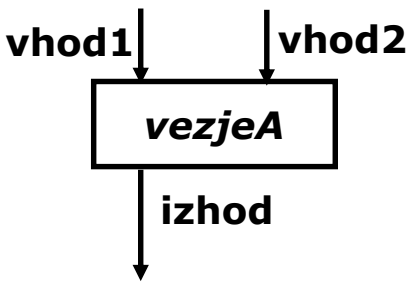
Povratne zanke v VHDL - SIGNAL

- Druga možnost implementacije povratnih zank je uvedba **žice** (signal):
  - Izhod C deklariramo kot OUT
    C <= A or (B and C).

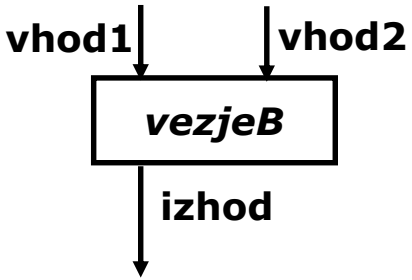
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity loopback is
        Port (   A : in STD_LOGIC;
                 B : in STD_LOGIC;
                 C : out STD_LOGIC);
end loopback;
architecture arch of loopback is
signal C_signal : STD_LOGIC;
begin
        C <= C_signal;
        C_signal <= A or ( B and C_signal );
end arch;
```

Povezovanje komponent v VHDL

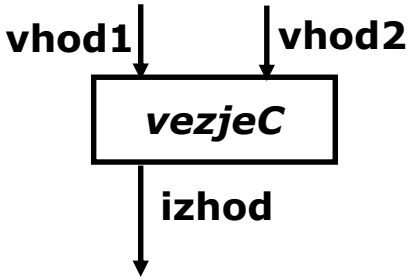
```
entity vezjeA is
Port ( vhod1, vhod2: in STD_LOGIC;
       izhod : out STD_LOGIC);
end vezjeA;
architecture arch of vezjeA is
begin
 izhod  <= vhod1 and vhod2;
end arch;
```

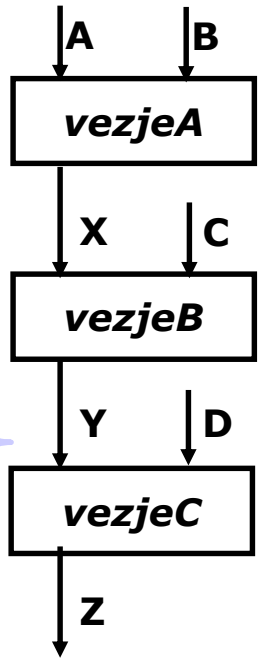



```
entity vezjeB is
Port (vhod1, vhod2: in STD_LOGIC;
       izhod : out STD_LOGIC);
end vezjeB;
architecture arch of vezjeB is
begin
 izhod <= vhod1 xor vhod2;
end arch;
```




```
entity vezjeC is
Port (vhod1, vhod2: in STD_LOGIC;
       izhod : out STD_LOGIC);
end vezjeC;
architecture arch of vezjeC is
begin
 izhod <= vhod1 or vhod2;
end arch;
```

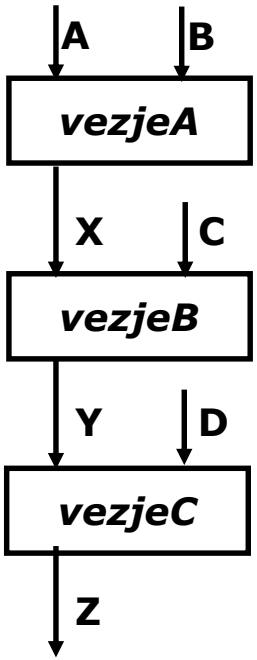


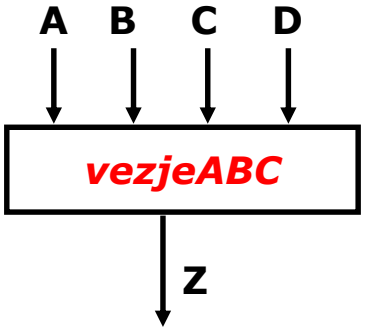


Povezovanje komponent v VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity vezjeABC is
Port ( A, B, C, D : in STD_LOGIC;
            Z : out STD_LOGIC);
end vezjeABC;
architecture arch of vezjeABC is
signal X, Y : STD_LOGIC;
component vezjeA is
Port ( vhod1, vhod2: in STD_LOGIC;
            izhod : out STD_LOGIC);
end component;
component vezjeB is
Port ( vhod1, vhod2: in STD_LOGIC;
            izhod : out STD_LOGIC);
end component;
component vezjeC is
Port ( vhod1, vhod2: in STD_LOGIC;
            izhod : out STD_LOGIC);
end component;
begin

 U1: vezjeA port map( A, B, X ); --polozajno povezovanje (positional association)
 U2: vezjeB port map( C, X, Y );
 U3: vezjeC port map( vhod2 => Y,
                      vhod1 => D,
                      izhod => Z);  --imensko povezovanje – lahko zamenjamo vrstni red
end arch;
```





Povezovanje komponent v VHDL

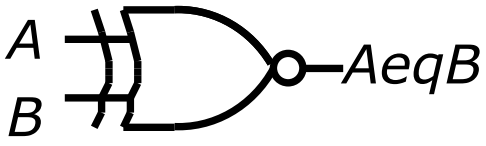
- **U1: vezjeA port map(A, B, X);**
  - Definira primer komponente **vezjeA** z imenom **U1**
  - Uporablja *položajno povezovanje* (*positional association)*
  - Vhodi in izhodi, navedeni v PORT MAP se pojavljajo v enakem zaporedju kot v stavku COMPONENT (Deklaracijo entitete kopiramo in zamenjamo ENTITIY ←→ COMPONENT)
  - **Pisanje je hitrejše, zato hitro lahko zamenjamo dva položaja. VHDL primerja samo velikost povezav in take napake ne odkrije, zato tega načina ne uporabljajte!**
- **U3: vezjeC port map(vhod2 => Y, vhod1 => D, izhod => Z);**
  - Definira primer komponente **vezjeC** z imenom **U3**
  - Uporablja *imensko povezovanje* (*named association)*
  - Vhodi in izhodi, navedeni v PORT MAP so povezani z določenim poimenovanjem signala v stavku COMPONENT
  - Nepovezanih vhodov ne izpišemo, zaradi popolnosti pa raje uporabimo vhod2=>**open**

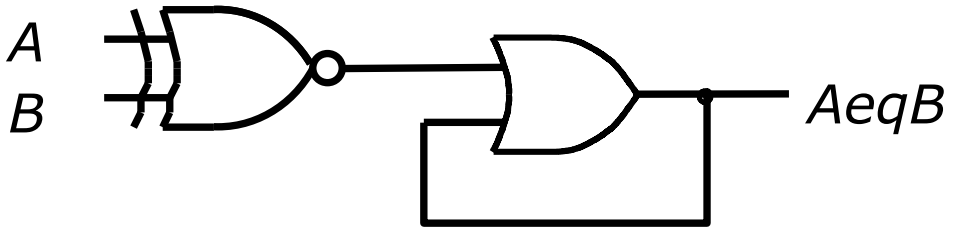
Implicitni spomin v procesu

```
ARCHITECTURE arch OF c1 IS
BEGIN
PROCESS ( A, B )
BEGIN
   AeqB <= '0';
   IF A = B THEN
      AeqB <= '1' ;
    END IF ;
END PROCESS ;
END arch ;
```

```
ARCHITECTURE arch OF c1 IS
BEGIN
PROCESS ( A, B )
BEGIN

   IF A = B THEN
       AeqB <= '1' ;
   END IF ;
END PROCESS ;
END arch ;
```



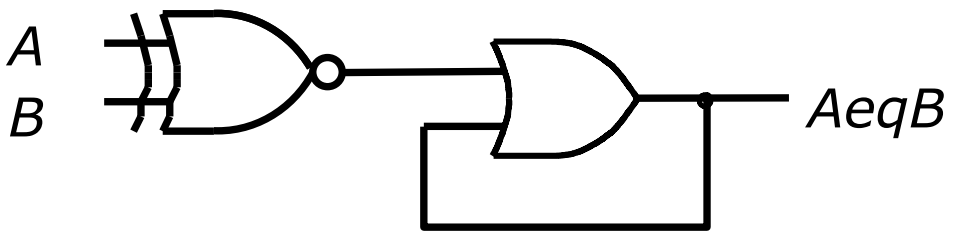


Izogibanje zatičem v VHDL

- Stavek **IF** brez **ELSE** *ohranja staro stanje* spremenljivke, ki ji v **IF** stavku prirejamo vrednost!
- Ohranjanje stanja ➔ izhod vezja je odvisen:
  - od vrednosti vhodov in
  - od stare vrednosti izhoda
- Velja za vse pogojne stavke znotraj procesnega stavka (**IF**, **CASE**)

```
ARCHITECTURE arch OF c1 IS
BEGIN
PROCESS ( A, B )
BEGIN
   IF A = B THEN
        AeqB <= '1';
   END IF ;
END PROCESS ;
END arch ;
```

```
WARNING:Xst :737 - Found 1-
  bit latch for signal
  <AeqB>.
```

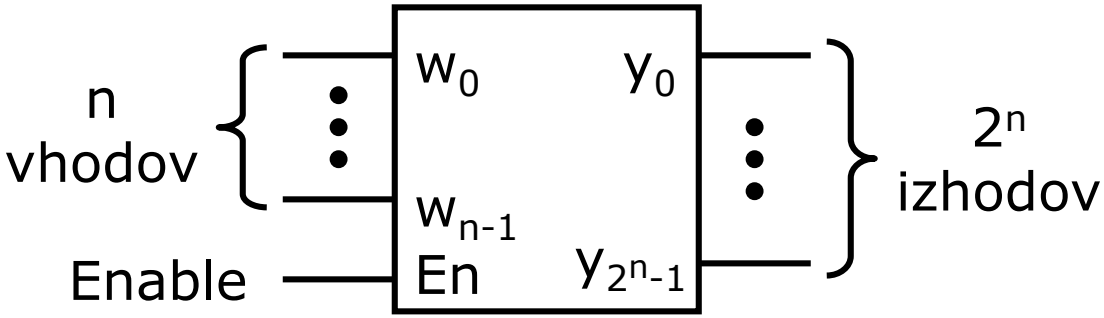


Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:

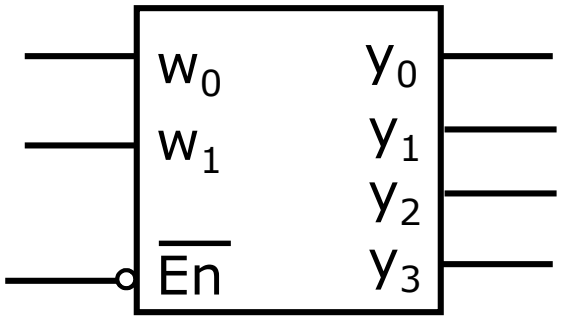
Dekoderji, demultiplekserji, kodirniki, pretvorniki kode in primerjalniki

Dekoderji

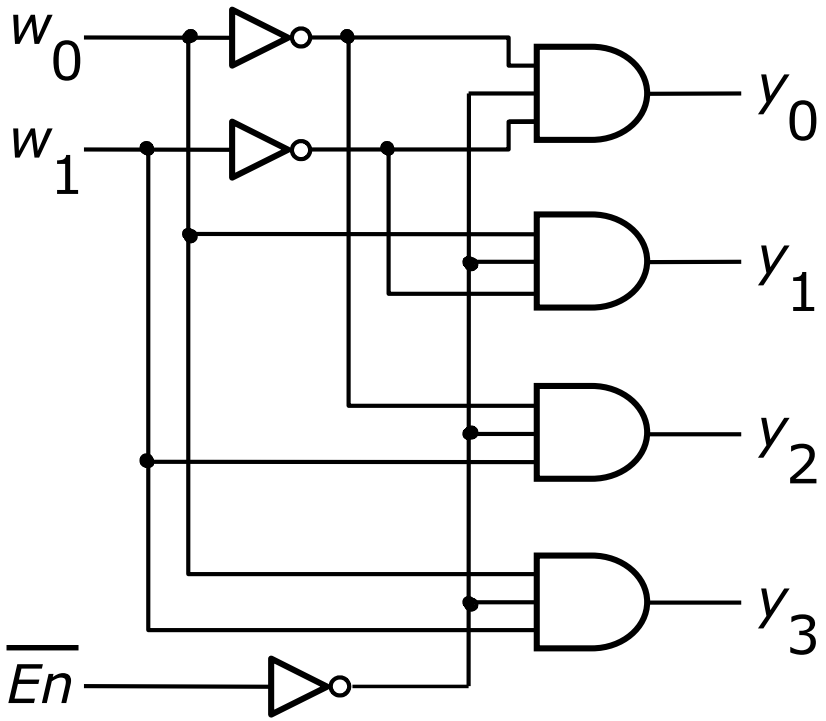


- Enable='0' ➔ ni izbran noben izhod (vsi onemogočeni)
- Enable='1' ➔ izbran samo eden od izhodov glede na kombinacijo vhodov (koda 1-od-N oz. koda "ena naenkrat")

Vezje dekoderja 2/4

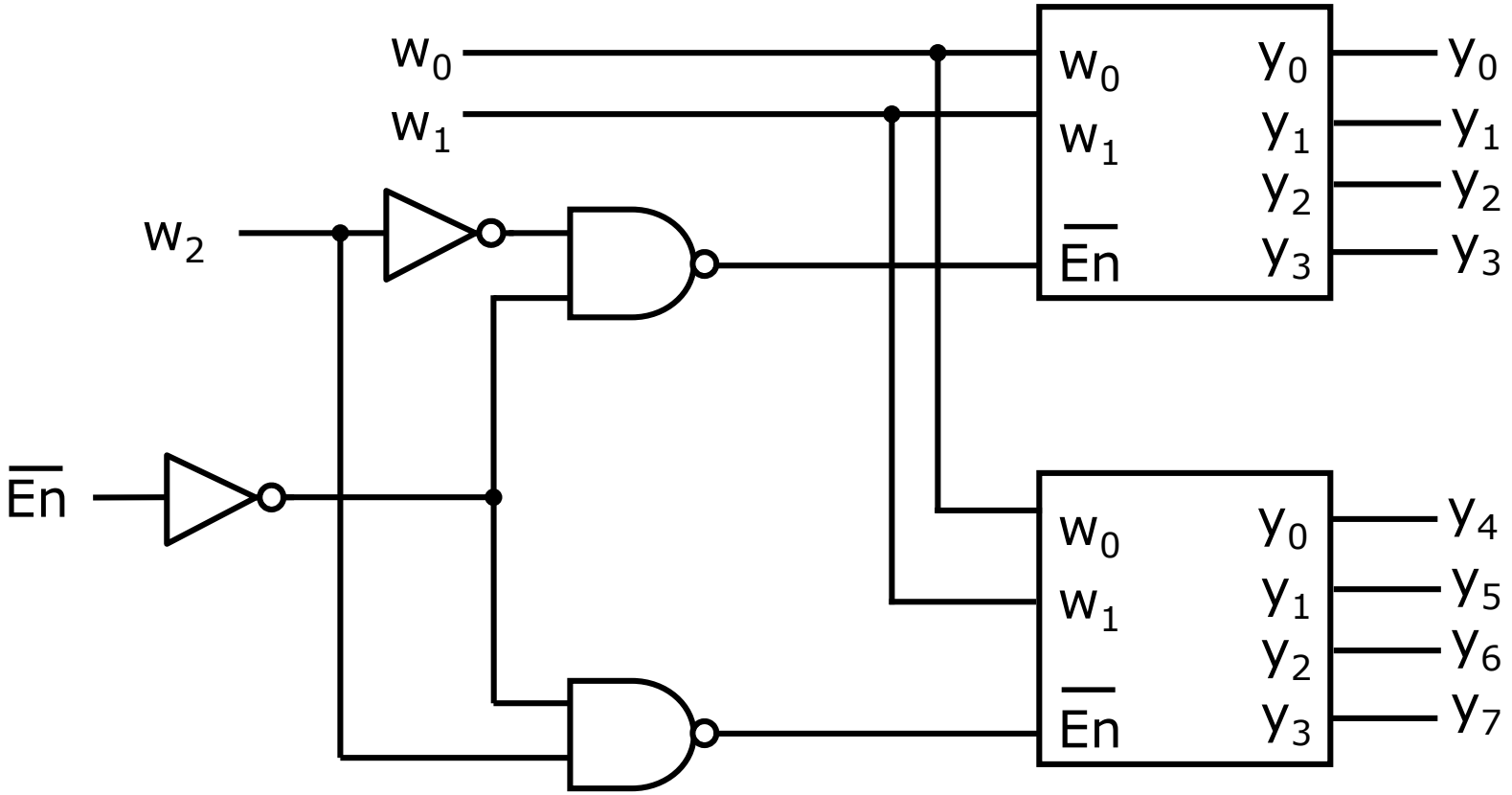


$\overline{En}$	w_1	w_0	y_0	y_1	y_2	y_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	X	0	0	0	0



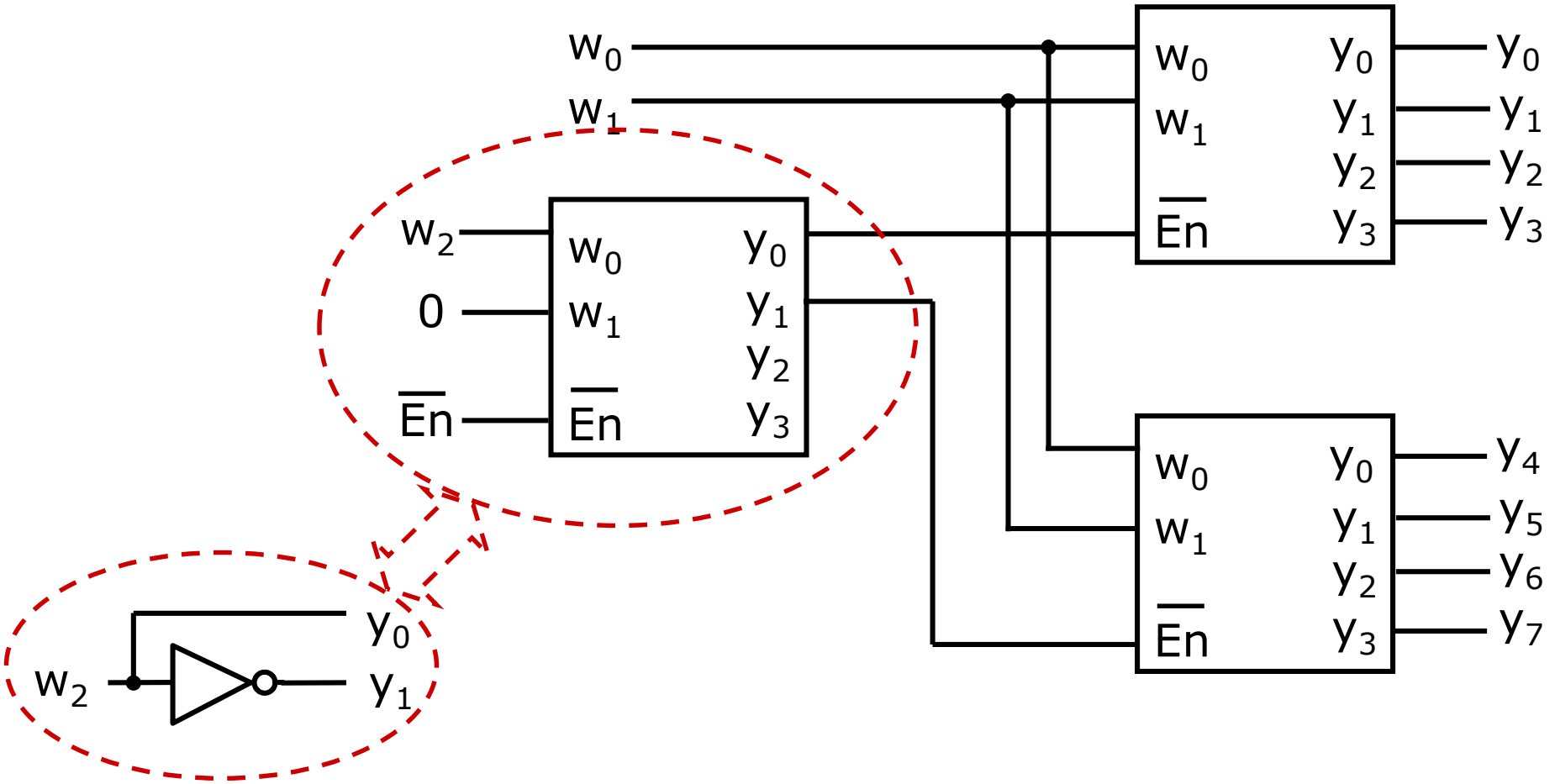
Realizacija večjih dekoderjev iz manjših

DMUX 3/8

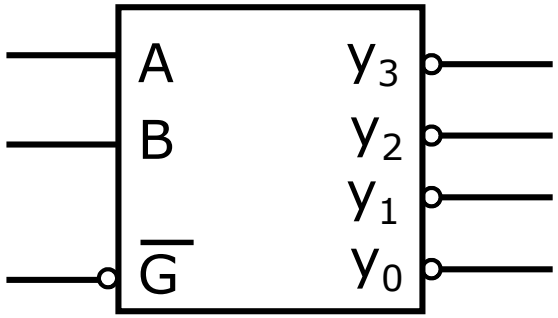


Realizacija večjih dekoderjev iz manjših

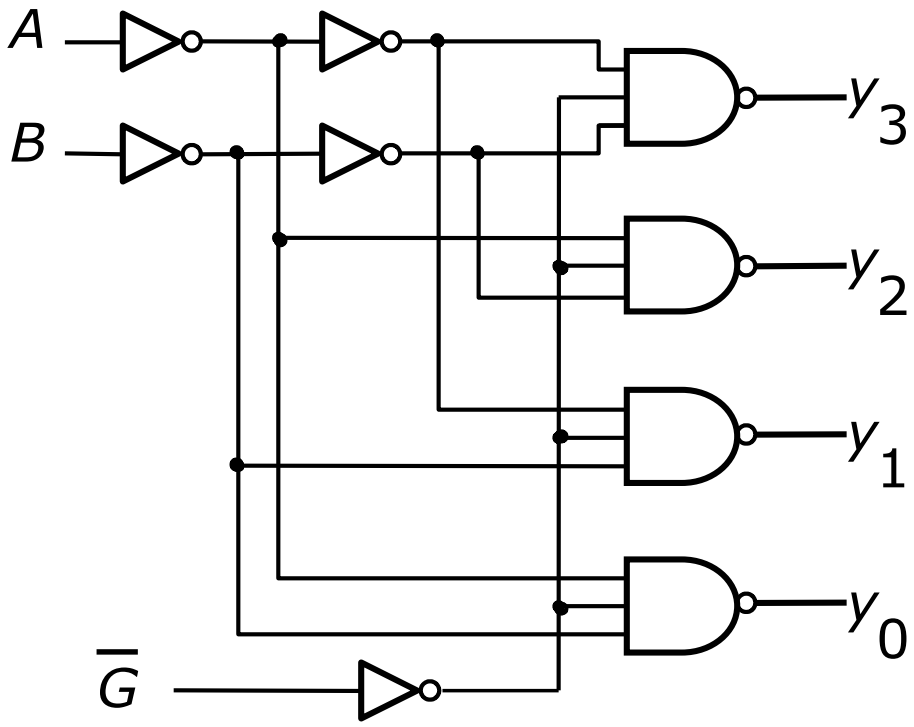
dekoder 3/8



Dvojni dekoder 2/4 - 74139

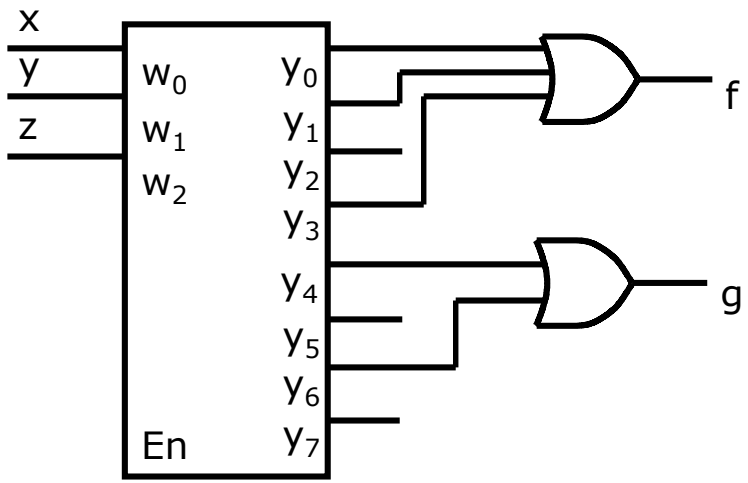


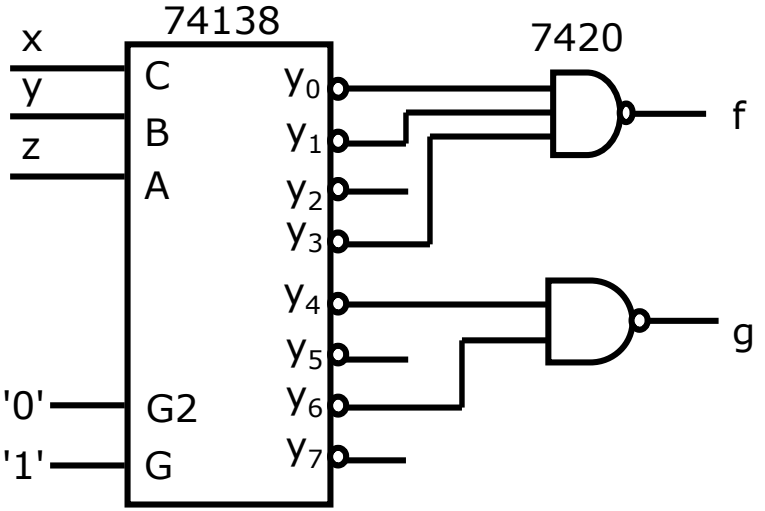
$\overline{G}$	B	A	y_0	y_1	y_2	y_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0



Uporaba dekoderjev

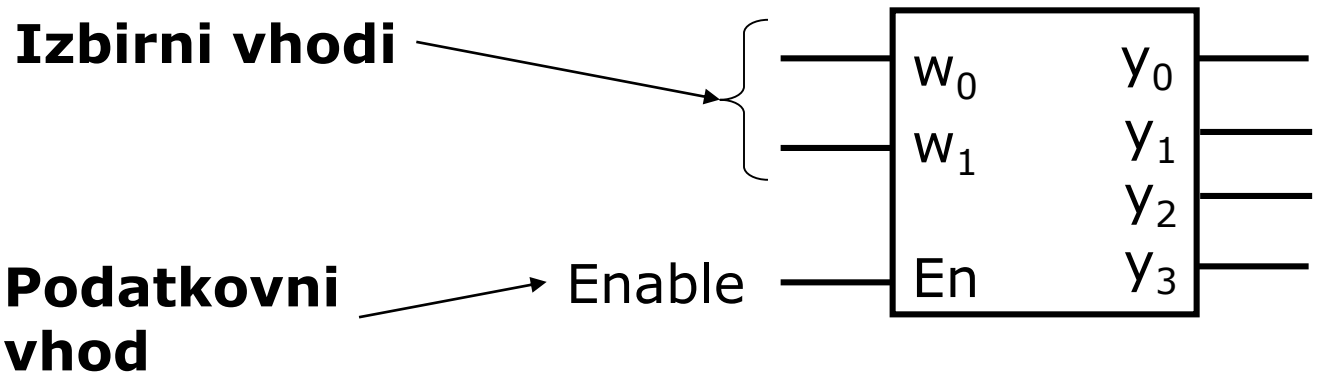
- Tipična uporaba dekoderjev je pri realizaciji funkcij v DNO obliki – generator mintermov
- Z uporabo dekoderjev 3/8 in OR vrat realizirajte funkciji: $f(x, y, z) = x'\cdot y'\cdot z' + x'\cdot y'\cdot z + x'\cdot y\cdot z$ in $g(x,y,z) = x\cdot y'\cdot z' + x\cdot y\cdot z'$





Demultiplekser

- MUX izbira med *n* podatkovnimi vhodi in enim izhodom. Vezje, ki deluje obratno je demultiplekser.
- Demultiplekser postavi vrednost enega vhoda na izhod, ki je trenutno izbran.
- Dekoder $n/2^n$ lahko realizira $1/n$ demultiplekser, če ima podatkovni vhod (Enable)



DMUX v VHDL

```
ENTITY dec2to4 IS
PORT (      w : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            En : IN STD_LOGIC;
            y : OUT STD_LOGIC_VECTOR(0 TO 3));
END dec2to4;
ARCHITECTURE arch OF dec2to4 IS
SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    Enw <= En & w;
    WITH Enw SELECT
            y <=    "1000" WHEN "100",
                    "0100" WHEN "101",
                    "0010" WHEN "110",
                    "0001" WHEN "111",
                    "0000" WHEN OTHERS;
END arch;
```

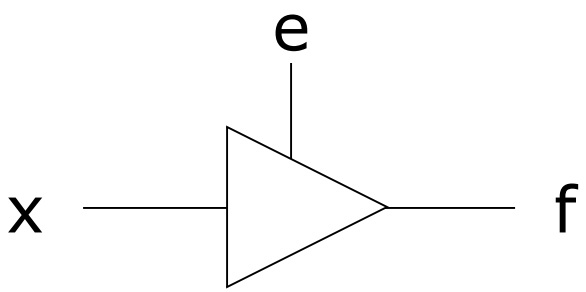
Razvoj digitalnih sistemov

Tehnologija realizacije:

Ojačevalniki (buffer), tri-stanjska (Tri-state) vrata, Prenosna vrata (Transmission gate)

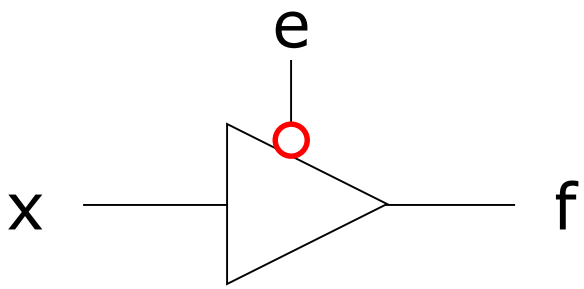
3-stanjski ojačevalniki in vrata

Tri-stanjski ojačevalnik (3-state buffer)



f=x če je e=1, sicer f=Z

f<=x **when** e='1' **else** 'Z';



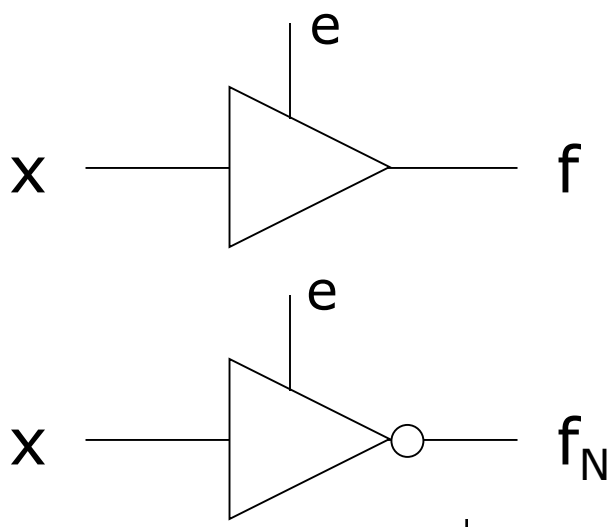
f=x če je e=0, sicer f=Z

f<=x **when** e='0' **else** 'Z';

Če je izhod invertiran,
je to tristanjski inverter (3-state inverter)

3-stanjski ojačevalniki

- e=1 ➔ f=x
  (ojačevalnik na svojem izhodu dal enako vrednost kot f)
- e=0, ➔ f=Z
  (ojačevalnik svoj izhod izključi od obeh logičnih nivojev).
- Tok takrat ne bo tekel:
  - Ne v izhod (sink),
  - Ne iz izhoda (source)

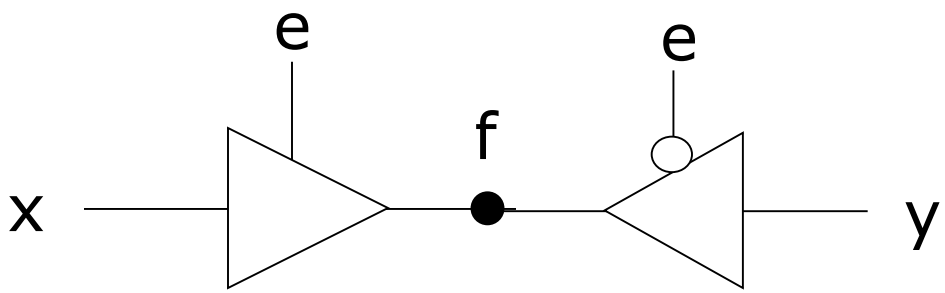


e	x	f_N	f
0	0	Z	Z
0	1	Z	Z
1	0	1	0
1	1	0	1

Tipi tri-stanjskih ojačevalnikov

- 4 konfiguracije 3-stanjskih ojačevalnikov glede na:
  - tip izhoda
    - Invertirajoč (inverter), neinvertirajoč (buffer)
  - tip kontrolnega signala (e)
    - akt. visok (active-high), akt. nizek (active-low)
  - Vezja 74XX družine:
    - '244 (neinv., akt. nizek) '240 (inv., akt. nizek)
- Aktivno nizek pomeni, da je izhod aktiven (f=x) ko je kontrolni signal (e=0).

Povezovanje več izhodov - vodilo



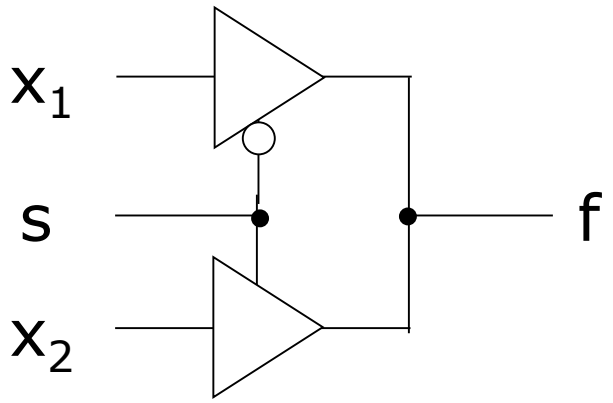
f=x samo če je e=1, sicer f=Z

f=y samo če je e=0, sicer f=Z

VHDL: f<=x **when** e='1' **else** 'Z';

Uporaba tri-stanjskih ojačevalnikov

MUX 2/1 s tristanjskimi ojačevalniki



s	x1	x2	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Izhodi tristanjskih ojačevalnikov so vezani skupaj!

```
f <= x1 when s='0' else x2;
```

Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:
Kodirniki

Pretvorniki kode (code converters)

- Pretvorijo kodiranje vhodnih signalov v drug tip kodiranja izhodnih signalov.

- Primera:
  - 3/8 dekodirnik:
    pretvori število na vhodu v zaporedno številko izhoda (kodiranje "ena naenkrat" oz. "1-od-N", ang. "one-hot encoding")

  - 8/3 kodirnik:
    pretvori vhod z zaporedno številko v število na izhodu

BIN ➔ BCD pretvornik kode

00000 → 0 0000	$(0\ 0_{BCD})$	01010 → 1 0000	$(1\ 0_{BCD})$
00001 → 0 0001	$(0\ 1_{BCD})$	01011 → 1 0001	$(1\ 1_{BCD})$
00010 → 0 0010	$(0\ 2_{BCD})$	01100 → 1 0010	$(1\ 2_{BCD})$
00011 → 0 0011	$(0\ 3_{BCD})$	01101 → 1 0011	$(1\ 3_{BCD})$
00100 → 0 0100	$(0\ 4_{BCD})$	01110 → 1 0100	$(1\ 4_{BCD})$
00101 → 0 0101	$(0\ 5_{BCD})$	01111 → 1 0101	$(1\ 5_{BCD})$
00110 → 0 0110	$(0\ 6_{BCD})$	10000 → 1 0110	$(1\ 6_{BCD})$
00111 → 0 0111	$(0\ 7_{BCD})$	10001 → 1 0111	$(1\ 7_{BCD})$
01000 → 0 1000	$(0\ 8_{BCD})$	10010 → 1 1000	$(1\ 8_{BCD})$
01001 → 0 1001	$(0\ 9_{BCD})$	10011 → 1 1001	$(1\ 9_{BCD})$
		10100 →10 0000	$(2\ 0_{BCD})$

Pomik levo eno mesto je množenje z 2

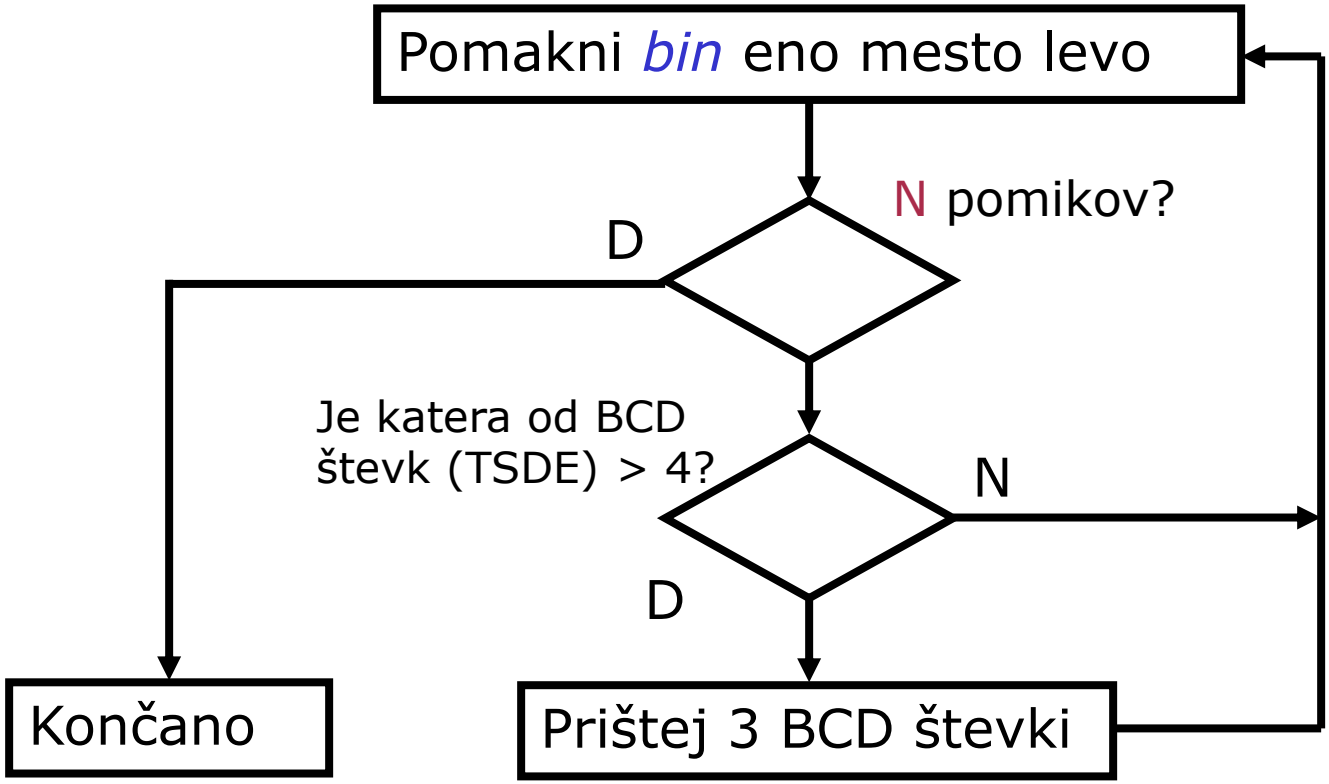
BIN ➔ BCD pretvornik kode

01010 → **1 000**0 (1 0_{BCD})
01011 → **1 000**1 (1 1_{BCD})
01100 → **1 001**0 (1 2_{BCD})
01101 → **1 001**1 (1 3_{BCD})
01110 → **1 010**0 (1 4_{BCD})
01111 → **1 010**1 (1 5_{BCD})
10000 → **1 011**0 (1 6_{BCD})
10001 → **1 011**1 (1 7_{BCD})
10010 → **1 100**0 (1 8_{BCD})
10011 → **1 100**1 (1 9_{BCD})

2^{i+4}	2^{i+3}	2^{i+2}	2^{i+1}	2^{i}					
0	0	0	0	X	0	0	0	0	X
0	0	0	1	X	0	0	0	1	X
0	0	1	0	X	0	0	1	0	X
0	0	1	1	X	0	0	1	1	X
0	1	0	0	X	0	1	0	0	X
0	1	0	1	X	1	0	0	0	X
0	1	1	0	X	1	0	0	1	X
0	1	1	1	X	1	0	1	0	X
1	0	0	0	X	1	0	1	1	X
1	0	0	1	X	1	1	0	0	X

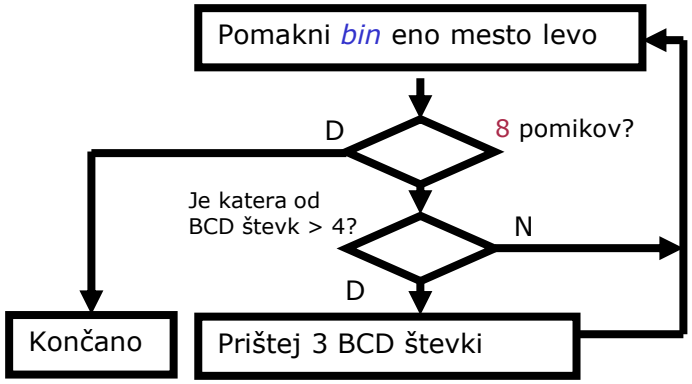
BIN ➔ BCD pretvornik kode

- *"Double dabble"* algoritem za pretvorbo N bitnega števila (*bin*) v BCD zapis (TSDE)



4 bitni BIN ➔ BCD primer 14_{10}

D	E	bin	operacija	opomba
		1110	pomik1	
	1	110	pomik2	<4
	11	10	pomik3	<4
	111	0	+3	>4
	1010	0	pomik4	
1	0100		končano	
1	*4*			



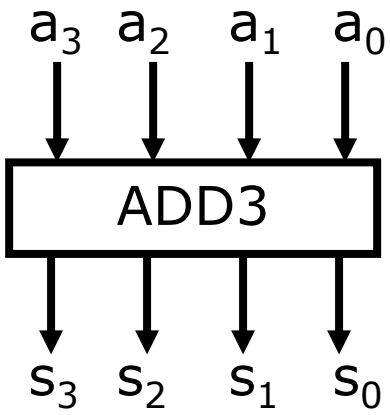
8 bitni BIN ➔ BCD primer 255_{10}

S	D	E	F	F	operacija	opomba
			1111	1111	pomik1	
		1	1111	111	pomik2	
		11	1111	11	pomik3	
		111	1111	1	+3	>4
		1010	1111	1	pomik4	
	1	0101	1111		+3	>4
	1	1000	1111		pomik5	
	11	0001	111		pomik6	
	110	0011	11		+3	>4
	1001	0111	11		pomik7	
1	0010	1010	1		+3	>4
1	0010	0101	1		pomik8	
10	0101	0101			končano	
2	5	5				

8 bitni BIN ➔ BCD primer 143_{10}

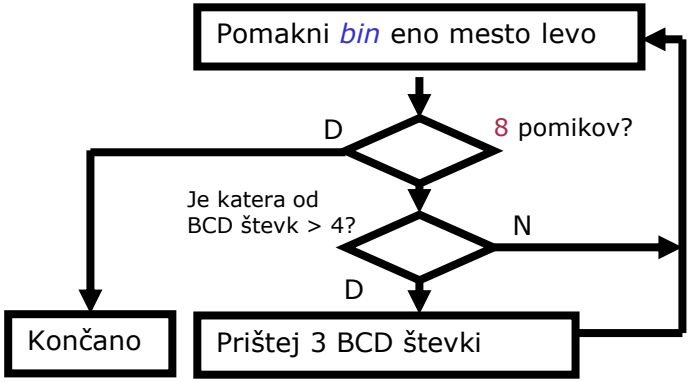
S	D				E				Š	T	E	V	I	L	O	operacija
									1	0	0	0	1	1	1	1	POMIK1
								1	0	0	0	1	1	1	1		POMIK2
							1	0	0	0	1	1	1	1			POMIK3
						1	0	0	0	1	1	1	1				POMIK4
					1	**0**	**0**	**0**	1	1	1	1					**+3**
					1	0	1	1	1	1	1	1					POMIK5
				1	0	1	1	1	1	1	1						+3
				1	**1**	**0**	**1**	**0**	1	1	1						POMIK6
			1	1	0	1	0	1	1	1							+3
			1	1	**1**	**0**	**0**	**0**	1	1							POMIK7
		1	1	1	0	0	0	1	1								+3
	1	**0**	**1**	**0**	0	0	0	1	1								POMIK8
1	0	1	0	0	0	0	1	1									
1			**4**				**3**										

BIN ➔ BCD: blok prištej 3 (ADD3)



a_3	a_2	a_1	a_0	s_3	s_2	s_1	s_0	operacija
0	0	0	0	0	0	0	0	<4
0	0	0	1	0	0	0	1	<4
0	0	1	0	0	0	1	0	<4
0	0	1	1	0	0	1	1	<4
0	1	0	0	0	1	0	0	=4
0	1	0	1	1	0	0	0	>4 (+3)
0	1	1	0	1	0	0	1	>4 (+3)
0	1	1	1	1	0	1	0	>4 (+3)
1	0	0	0	1	0	1	1	>4 (+3)
1	0	0	1	1	1	0	0	>4 (+3)
1	0	1	0	X	X	X	X	ni eno BCD mesto
1	0	1	1	X	X	X	X	ni eno BCD mesto
1	1	0	0	X	X	X	X	ni eno BCD mesto
1	1	0	1	X	X	X	X	ni eno BCD mesto
1	1	1	0	X	X	X	X	ni eno BCD mesto
1	1	1	1	X	X	X	X	ni eno BCD mesto

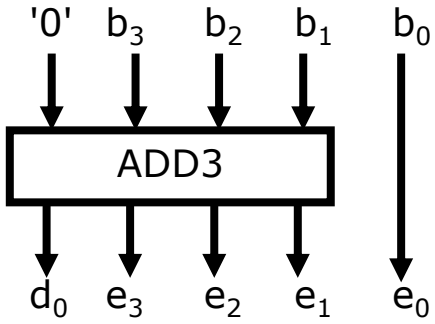
BIN ➔ BCD: realizacija v prostoru



$$B = \sum_{i=0}^{n-1} b_i \cdot 2^i = \left(\left(b_{n-1} \cdot 2 + b_{n-2}\right) \cdot 2 + b_{n-3}\right) \cdot 2 + ... + b_0$$

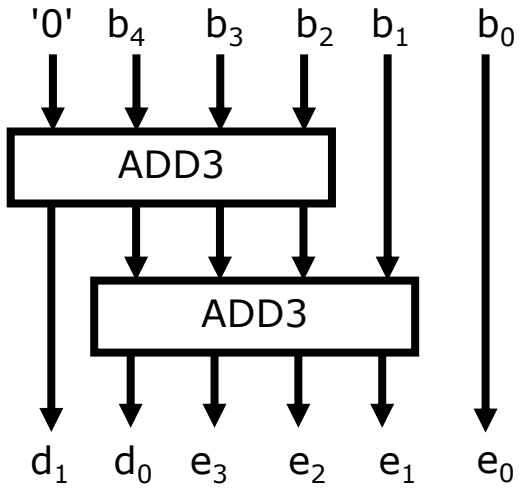
$$B = \left(\left(b_3 \cdot 2 + b_2\right) \cdot 2 + b_1\right) \cdot 2 + b_0$$

BCD		BIN	operacija	opomba
d_0	$e_3e_2e_1e_0$	$b_3b_2b_1b_0$		
		1 1 1 0	pomik1	
	1	1 1 0	pomik2	<4
	1 1	1 0	pomik3	<4
	1 1 1	0	+3	>4
	1 0 1 0	0	pomik4	
1	0 1 0 0		končano	
1	*4*			



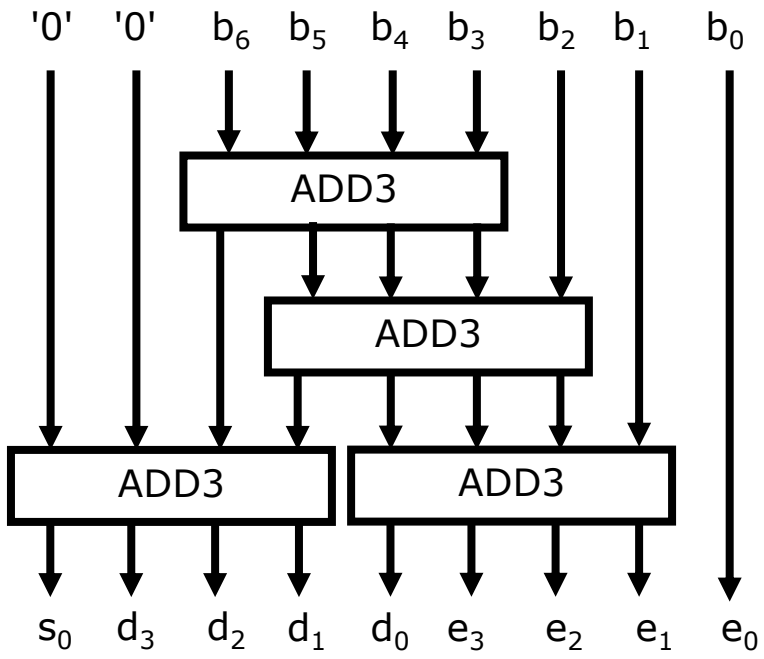
BIN ➔ BCD: realizacija v prostoru

BCD		BIN	operacija	opomba
d_1 d_0	$e_3e_2e_1e_0$	b_4 b_3 b_2 b_1 b_0		
		1 1 1 1 0	pomik1	
	1	1 1 1 0	pomik2	<4
	1 1	1 1 0	pomik3	<4
	1 1 1	1 0	+3	>4
	1 0 1 0	1 0	pomik4	
1	0 1 0 1	0	+3	>4
1	1 0 0 0	0	pomik5	
1 1	*0 0 0 0*		končano	
3	*0*			

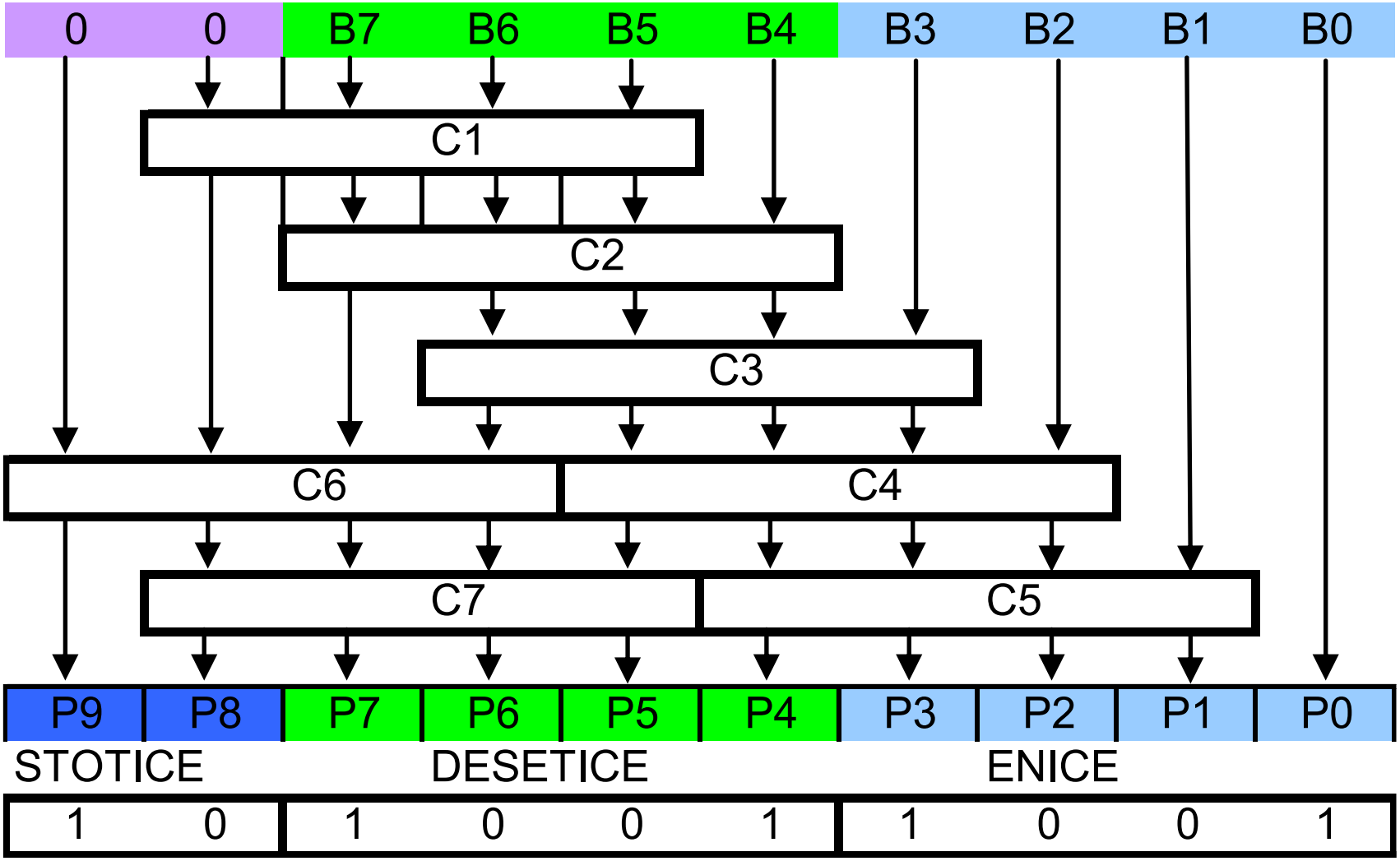


BIN ➔ BCD: realizacija v prostoru

s_0	$d_3d_2d_1d_0$	$e_3e_2e_1e_0$	$b_6b_5b_4b_3b_2b_1b_0$	
			1 1 1 1 1 1 0	pomik1
		1	1 1 1 1 1 0	pomik2
		1 1	1 1 1 1 0	pomik3
		1 1 1	1 1 1 0	+3
		1 0 1 0	1 1 1 0	pomik4
	1	0 1 0 1	1 1 0	+3
	1	1 0 0 0	1 1 0	pomik5
	1 1	0 0 0 1	1 0	pomik6
	1 1 0	0 0 1 1	0	+3
	1 0 0 1	0 0 1 1	0	pomik6
1	*0 0 1 0*	*0 1 1 0*		



8 bit BIN ➔ BCD pretvornik kode



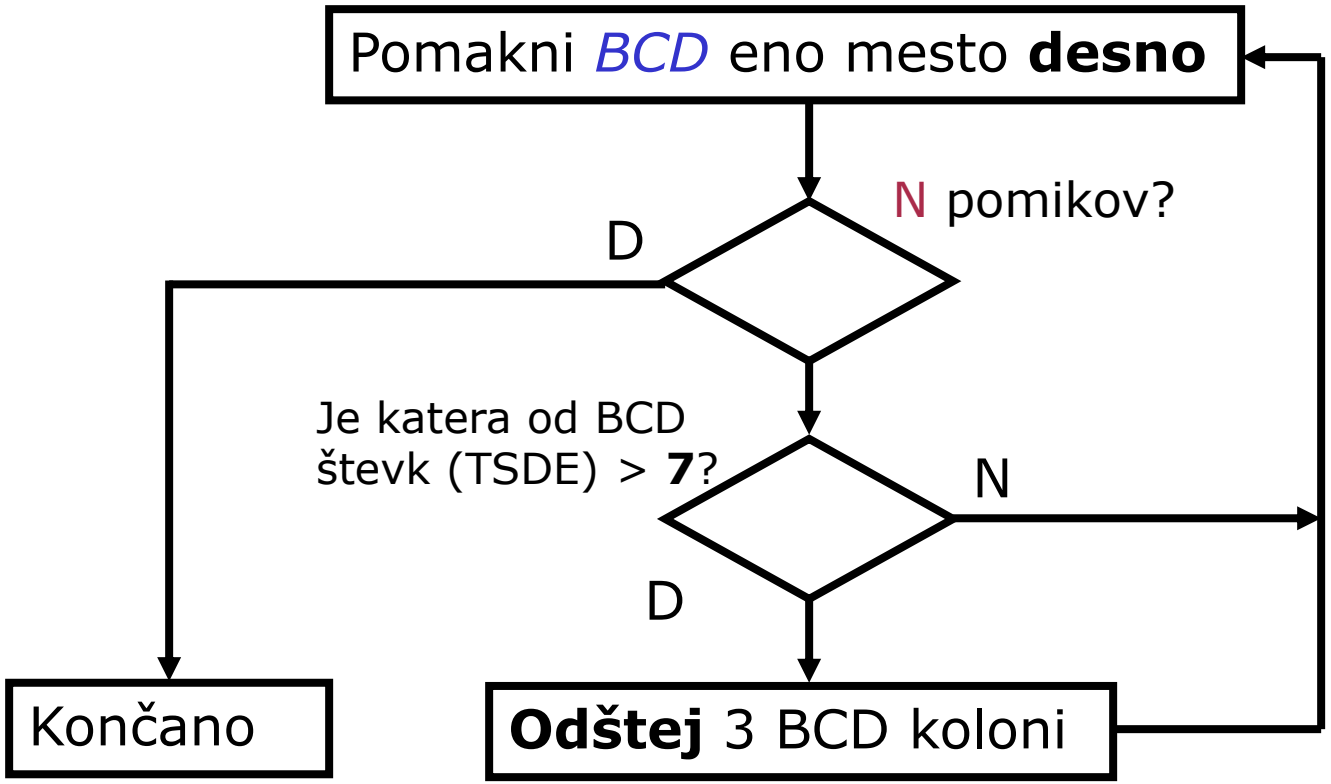
BCD→BIN pretvornik kode

- 4-bitni prekodirnik odšteje 3 od števil, ki so večja ali enaka 8.
- Če je BCD števka >7 odštej 3
- Večje (ali enako) od binarne 8 pomeni večje (ali enako) od 10_{BCD}

$(1\ 0_{BCD})$**1 000**0 → **0101**0
$(1\ 1_{BCD})$**1 000**1 → **0101**1
$(1\ 2_{BCD})$**1 001**0 → **0110**0
$(1\ 3_{BCD})$**1 001**1 → **0110**1
$(1\ 4_{BCD})$**1 010**0 → **0111**0
$(1\ 5_{BCD})$**1 010**1 → **0111**1
$(1\ 6_{BCD})$**1 011**0 → **1000**0
$(1\ 7_{BCD})$**1 011**1 → **1000**1
$(1\ 8_{BCD})$**1 100**0 → **1001**0
$(1\ 9_{BCD})$**1 100**1 → **1001**1

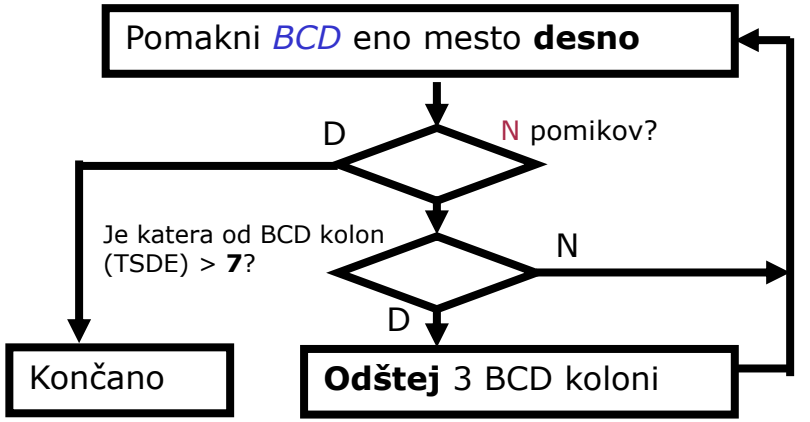
BCD➔BIN pretvornik kode

- Algoritem za pretvorbo BCD števila (TSDE) v N bitno dvojiško število (*bin*)



4 bit BCD ➔ BIN primer 14_{10}

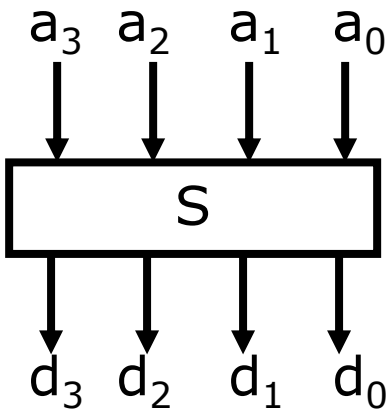
D	E	bin	operacija	opomba
1	0100		pomik1	<7
	1010	0	-3	>7
	0111	0	pomik2	<7
	011	10	pomik3	<7
	01	110	pomik4	<7
	0	1110	končano	



8 bitni BCD ➔ BIN primer $8F_{16}$

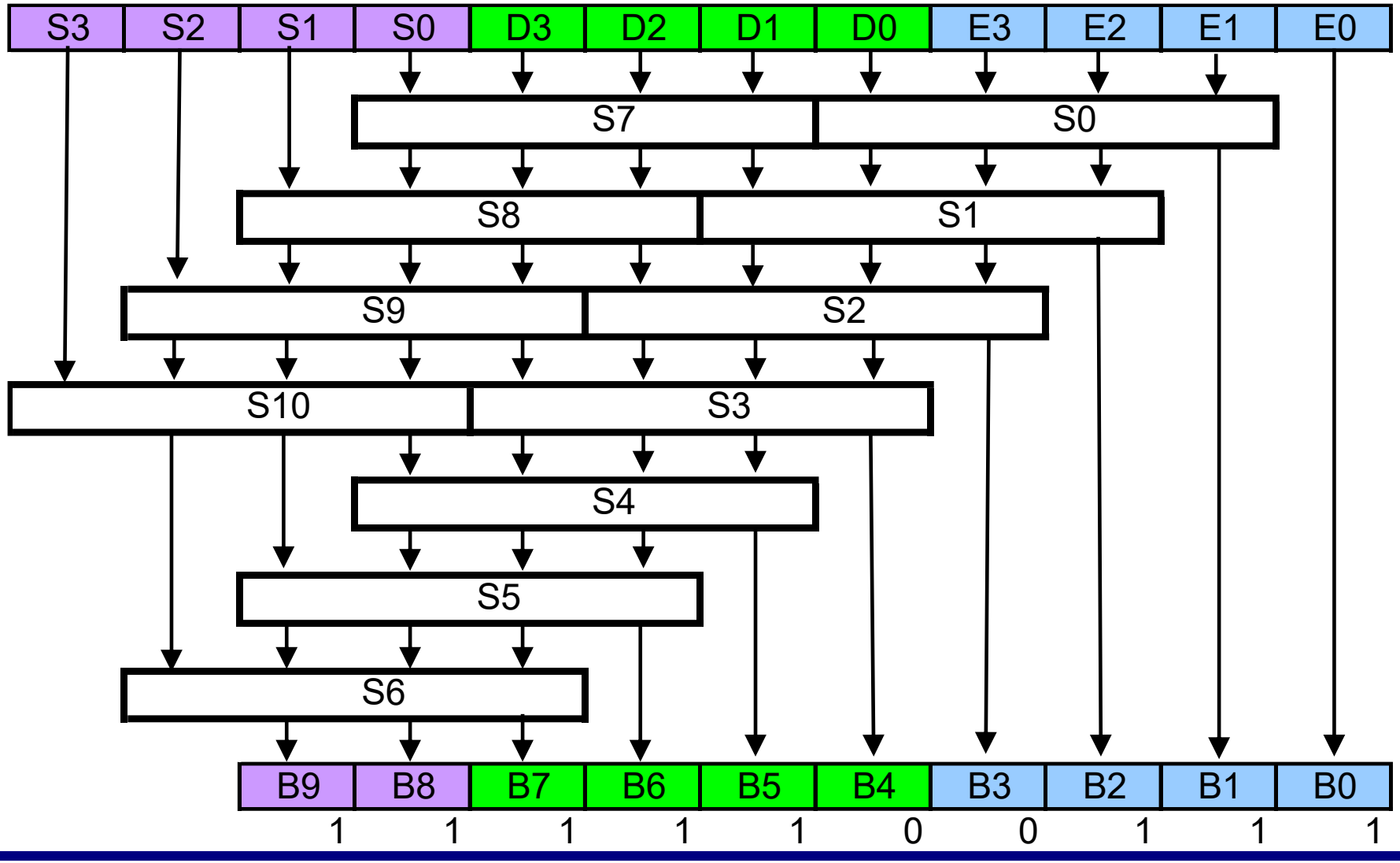
S				D				E				ŠTEVILO								operacija
0	0	0	1	0	1	0	0	0	0	1	1									POMIK1
	0	0	0	**1**	**0**	**1**	**0**	0	0	0	1	1								-3
	0	0	0	0	1	1	1	0	0	0	1	1								POMIK2
		0	0	0	0	1	1	**1**	**0**	**0**	**0**	1	1							-3
		0	0	0	0	1	1	0	1	0	1	1	1							POMIK3
			0	0	0	0	1	**1**	**0**	**1**	**0**	1	1	1						-3
			0	0	0	0	1	0	1	1	1	1	1	1						POMIK4
				0	0	0	0	**1**	**0**	**1**	**1**	1	1	1	1					-3
				0	0	0	0	1	0	0	0	1	1	1	1					POMIK5
					0	0	0	0	1	0	0	0	1	1	1	1				POMIK6
						0	0	0	0	1	0	0	0	1	1	1	1			POMIK7
							0	0	0	0	1	0	0	0	1	1	1	1		POMIK8
								0	0	0	0	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**$8F_{16}$**

BCD ➔ BIN : blok odštej 3 (SUB3)



a_3	a_2	a_1	a_0	d_3	d_2	d_1	d_0	operacija
0	0	0	0	0	0	0	0	<7
0	0	0	1	0	0	0	1	<7
0	0	1	0	0	0	1	0	<7
0	0	1	1	0	0	1	1	<7
0	1	0	0	0	1	0	0	<7
0	1	0	1	0	1	0	1	<7
0	1	1	0	0	1	1	0	<7
0	1	1	1	0	1	1	1	=7
1	0	0	0	0	1	0	1	>7 (-3)
1	0	0	1	0	1	1	0	>7 (-3)
1	0	1	0	0	1	1	1	>7 (-3)
1	0	1	1	1	0	0	0	>7 (-3)
1	1	0	0	1	0	0	1	>7 (-3)
1	1	0	1	X	X	X	X	ni možno!
1	1	1	0	X	X	X	X	ni možno!
1	1	1	1	X	X	X	X	ni možno!

BCD ➔ BIN : shema pretvorbe 999_{10}



Digitalne strukture

Gradniki kombinacijskih vezij:
Kodirniki: Gray-eva koda

Gray-eva koda

00 0 0
01 0 1
10 1 1
11 1 0

000 0 00
001 0 01
010 0 11
011 0 10
100 1 10
101 1 11
110 1 01
111 1 00

0000 0 000
0001 0 001
0010 0 011
0011 0 010
0100 0 110
0101 0 111
0110 0 101
0111 0 100
1000 1 100
1001 1 101
1010 1 111
1011 1 110
1100 1 010
1101 1 011
1110 1 001
1111 1 000

n_2 ➔ Gray-eva koda

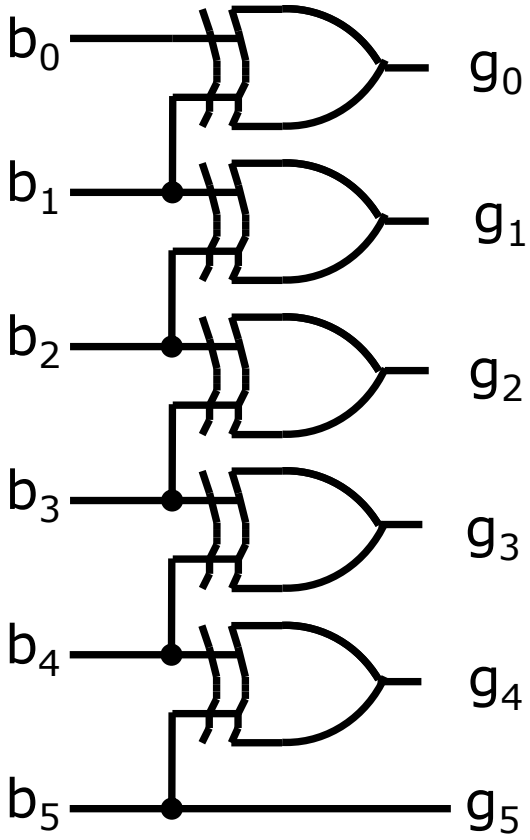
DK: 0 1 1 0 1 0 0 1 1 0

\+ 0 1 1 0 1 0 0 1 1

GK: 0 1 0 1 1 1 0 1 0 1

- Dvojiško kodo za eno mesto zamaknemo eno mesto desno
- Dobljeno vrednost prištejemo brez prenosov (XOR med bitoma):
  - Vsoti 0+1=1 in 1+0=1 ➔ 1 v GK
  - Vsoti 0+0=0 in 1+1=0 ➔ 0 v GK.

Strojna izvedba pretvornika n_2 ➔ GK



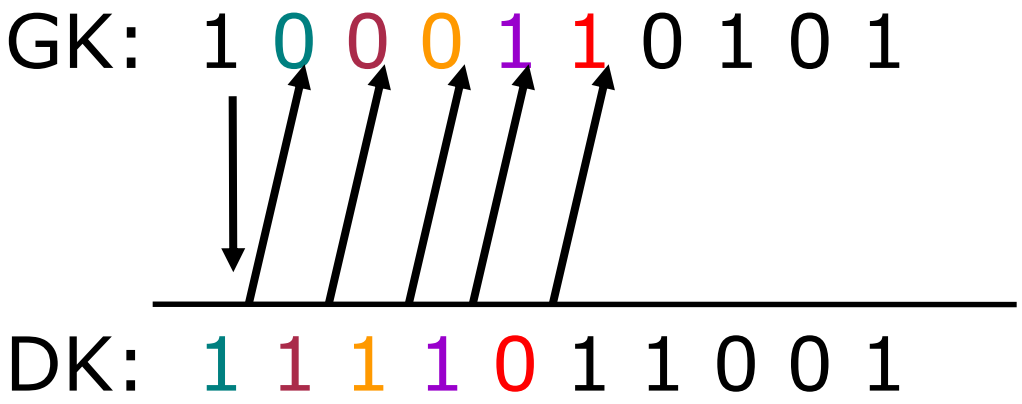
$g_0 = b_0 \oplus b_1$

$g_1 = b_1 \oplus b_2$

$\vdots$

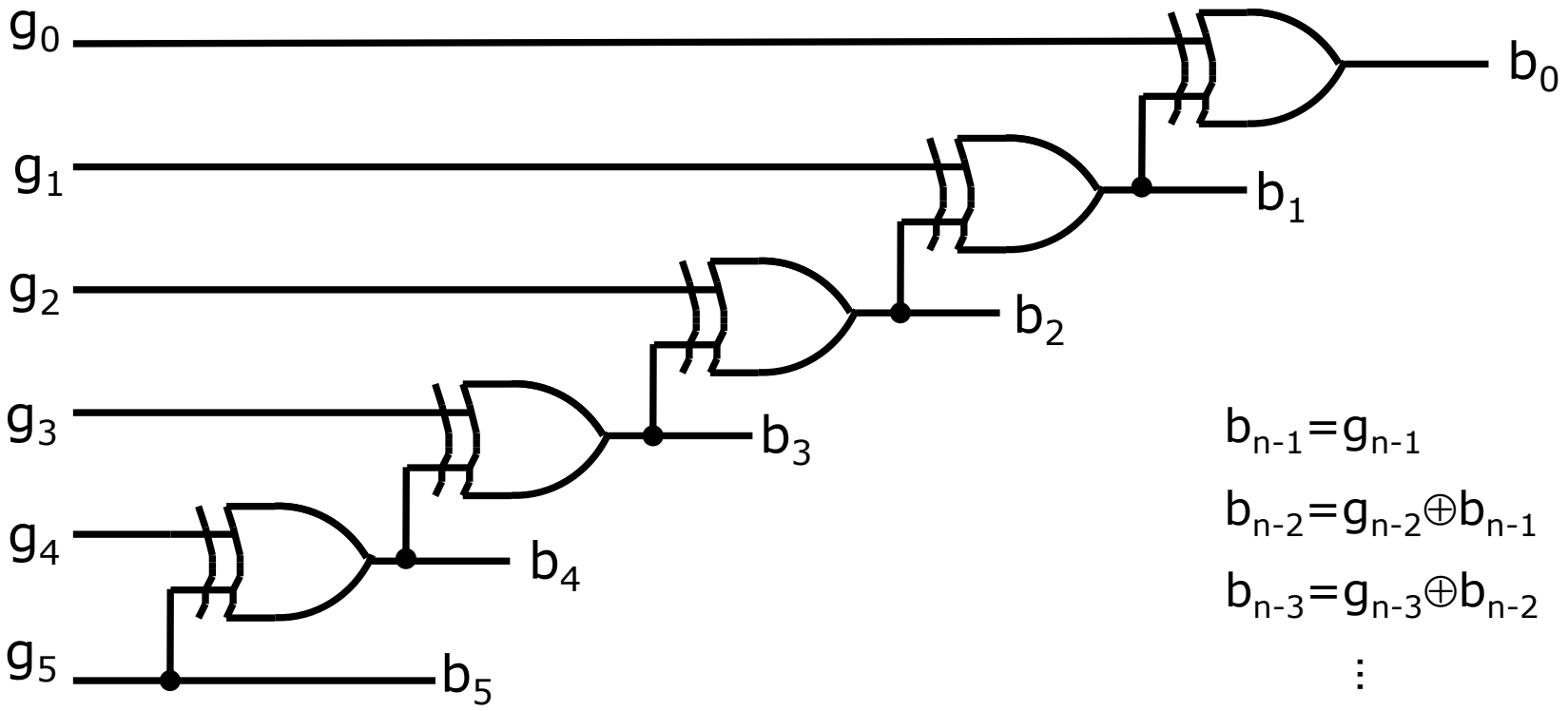
$g_{n-1} = b_{n-1}$

Gray-eva koda ➔ n_2



- Prepišemo MSB bit Grayeve kode na prvo mesto dvojiške kode,
- XOR z naslednjim bitom Grayeve kode,
- dobimo drugi bit dvojiške kode,
- dobljeni bit prištejemo naslednjemu bitu Grayeve kode,
- postopek ponavljamo do zadnjega bita Grayeve kode.

Strojna izvedba pretvornika n_2 ➔ GK



$b_{n-1}=g_{n-1}$

$b_{n-2}=g_{n-2}\oplus b_{n-1}$

$b_{n-3}=g_{n-3}\oplus b_{n-2}$

$\vdots$

Strojna izvedba pretvornika n_2 ➔ GK

$b_{n-1}=g_{n-1}$

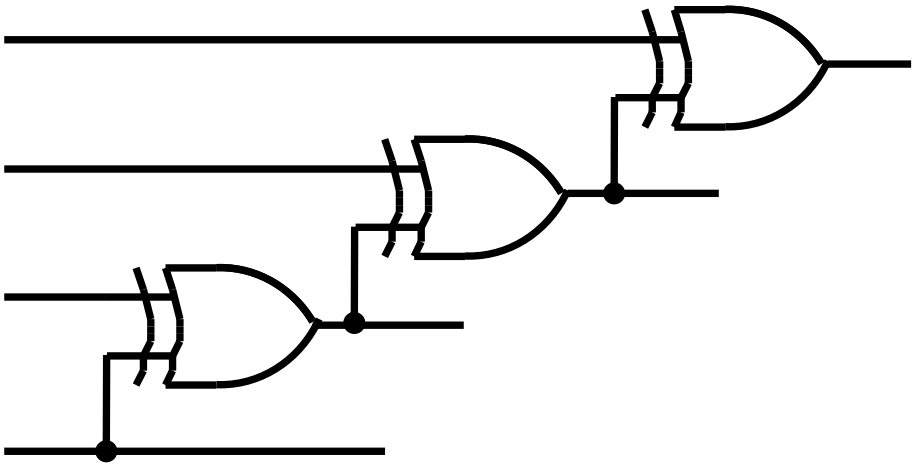
$b_{n-2}=g_{n-2}\oplus g_{n-1}$

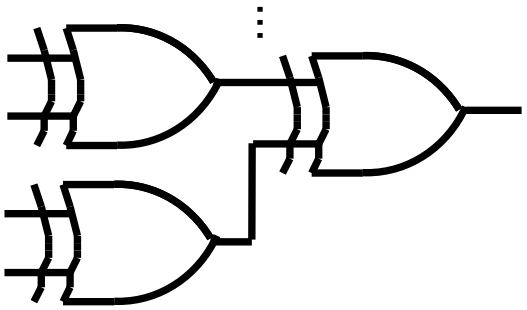
$b_{n-3}=g_{n-3}\oplus g_{n-2}\oplus g_{n-1}$

$b_{n-4}=g_{n-4}\oplus g_{n-3}\oplus g_{n-2}\oplus g_{n-1}$

Lastnost združevanja
(Boole-ova logika)

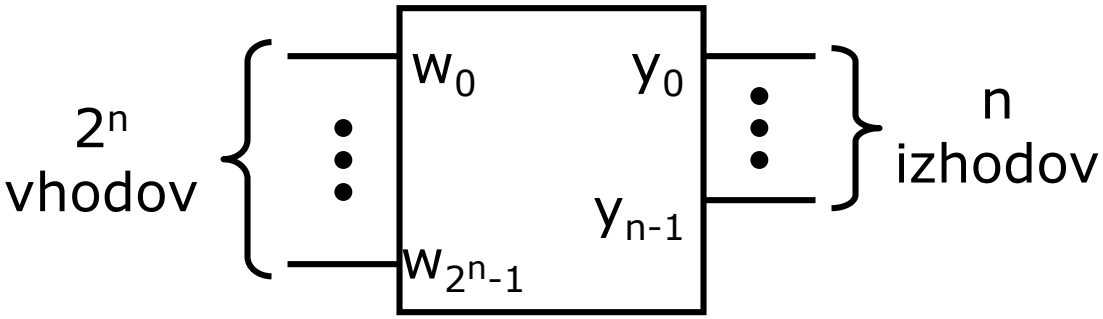
$=(g_{n-4}\oplus g_{n-3})\oplus(g_{n-2}\oplus g_{n-1})$





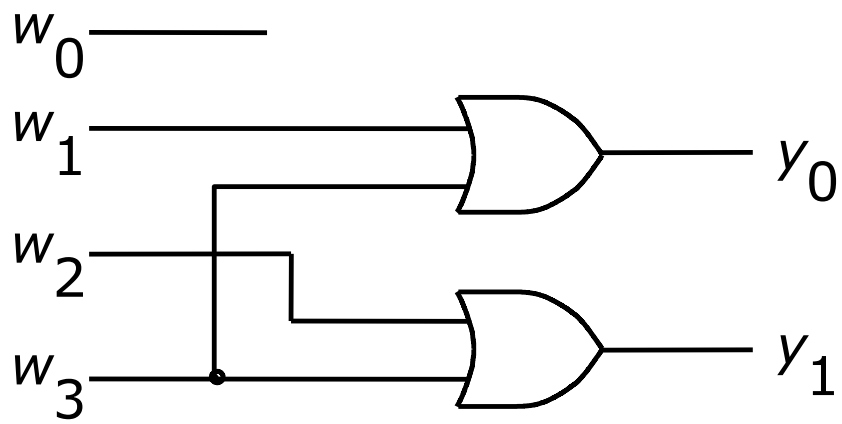
XOR vrata so samo 2-vhodna v TTL ➔ sestavimo drevesno strukturo.
(manjša zakasnitev, manj nivojev)

Kodirniki (Encoder)



$2^n/n$ binarni kodirnik

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



To bi delovalo, če bi bil aktiven *vedno samo en vhod*.
Sicer pri w=0110 dobimo y=11, česar ne želimo. Zato imamo prioriteto.

Kodirnik prioritete (Priority encoder)

- Naj ima w_0 najnižjo prioriteto in w_3 najvišjo
- Izhod ***z*** določa kdaj noben od vhodov ni 1
- Zapišemo
  - $i_0 = w_3' \cdot w_2' \cdot w_1' \cdot w_0$
  - $i_1 = w_3' \cdot w_2' \cdot w_1$
  - $i_2 = w_3' \cdot w_2$
  - $i_3 = w_3$

4/2 kodirnik prioritete

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

$y_0 = i_1 + i_3$, $y_1 = i_2 + i_3$

$z = i_1 + i_2 + i_3 + i_4$

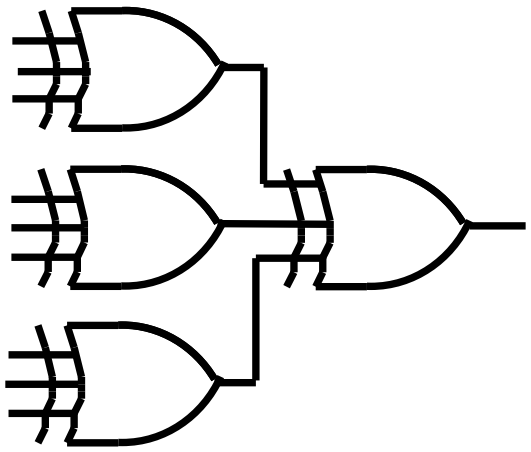
Kodirnik prioritete v VHDL

```
ENTITY priority IS
PORT (w : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
      y : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
      z : OUT STD_LOGIC);
END priority;
ARCHITECTURE arch OF priority IS
BEGIN
  y <="11" WHEN w(3) = '1' ELSE
      "10" WHEN w(2) = '1' ELSE
      "01" WHEN w(1) = '1' ELSE "00";

  z <='0' WHEN w = "0000" ELSE '1';
END arch;
```

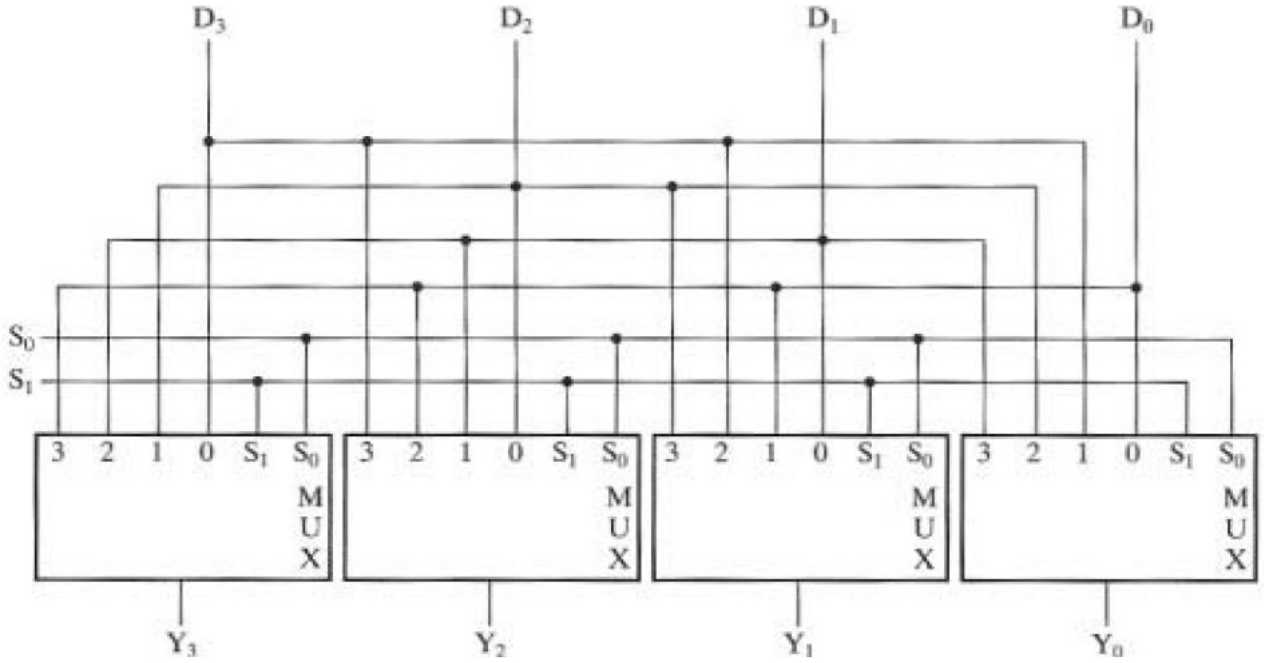
Kombinacijski generator parnosti

- f='1', če je število enic na vhodih liho, sicer je f='0'.
- XOR operacijo več spremenljivk se uporablja tudi kot generator paritete. (74280 ➔ 9-bitni MSI generator paritete)
- Realizira XOR in XNOR 9 spremenljivk.



$f=x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus \ldots \oplus x_9$

Pomikalnik podatkov – barrel shifter



S_1	S_0	**funkcija**	Y_3	Y_2	Y_1	Y_0
0	0	ni pomika	D_3	D_2	D_1	D_0
0	1	ROL 1 mesto	D_2	D_1	D_0	D_3
1	0	ROL 2 mesti	D_1	D_0	D_3	D_2
1	1	ROL 3 mesta	D_0	D_3	D_2	D_1

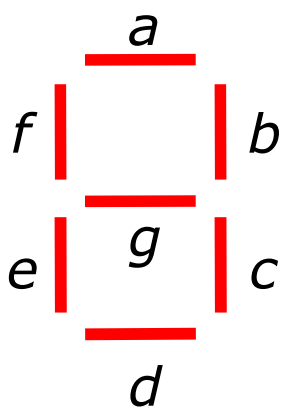
Pomikalnik podatkov v VHDL

```
architecture arch of brlshft4 is
begin
process (I,S)
begin
   case S is
     when "00" => O <= I;
     when "01" =>
              O(3) <= I(0);
              O(2 downto 0) <= I(3 downto 1);
     when "10" =>
              O(3 downto 2) <= I(1 downto 0);
              O(1 downto 0) <= I(3 downto 2);
     when "11" =>
              O(3 downto 1) <= I(2 downto 0);
              O(0) <= I(3);
     when others => O <= I;
   end case;
end process;
end arch;
```

Vhodi						Izhodi			
S1	**S0**	**I0**	**I1**	**I2**	**I3**	**O0**	**O1**	**O2**	**O3**
0	0	a	b	c	d	a	b	c	d
0	1	a	b	c	d	b	c	d	a
1	0	a	b	c	d	c	d	a	b
1	1	a	b	c	d	d	a	b	c

BCD ➔ 7-segmentni dekodirnik v VHDL

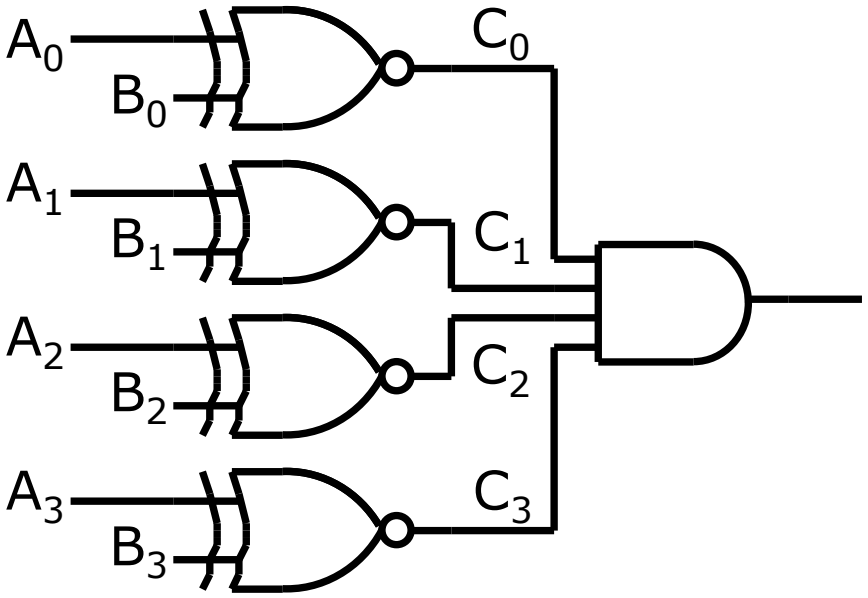
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY seg7 IS
PORT (D       : IN  STD_LOGIC_VECTOR (3 DOWNTO 0);  -- BCD vhod
      S       : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)); -- 7 segmentni izhod,
                                                    -- negativna logika: vrstni red gfedcba
END seg7;
ARCHITECTURE display OF SEG7 IS
BEGIN
s <=  "1000000" WHEN d = "0000" ELSE
      "1111001" WHEN d = "0001" ELSE
      "0100100" WHEN d = "0010" ELSE
      "0110000" WHEN d = "0011" ELSE
      "0011001" WHEN d = "0100" ELSE
      "0010010" WHEN d = "0101" ELSE
      "0000010" WHEN d = "0110" ELSE
      "1111000" WHEN d = "0111" ELSE
      "0000000" WHEN d = "1000" ELSE
      "0010000" WHEN d = "1001" ELSE
      "0001000" WHEN d = "1010" ELSE
      "0000011" WHEN d = "1011" ELSE
      "1000110" WHEN d = "1100" ELSE
      "0100001" WHEN d = "1101" ELSE
      "0000110" WHEN d = "1110" ELSE
      "0001110";
END display;
```



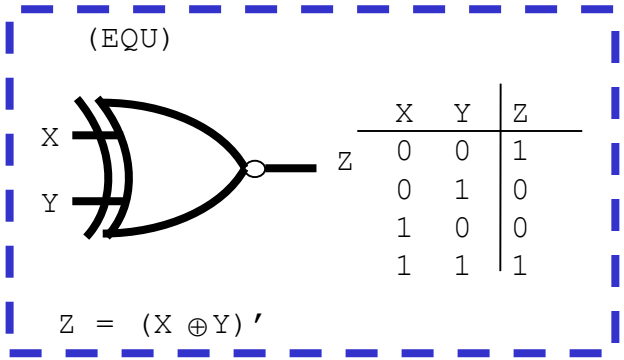
Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:
Primerjava nepredznačenih števil

4 bitni primerjalnik enakosti



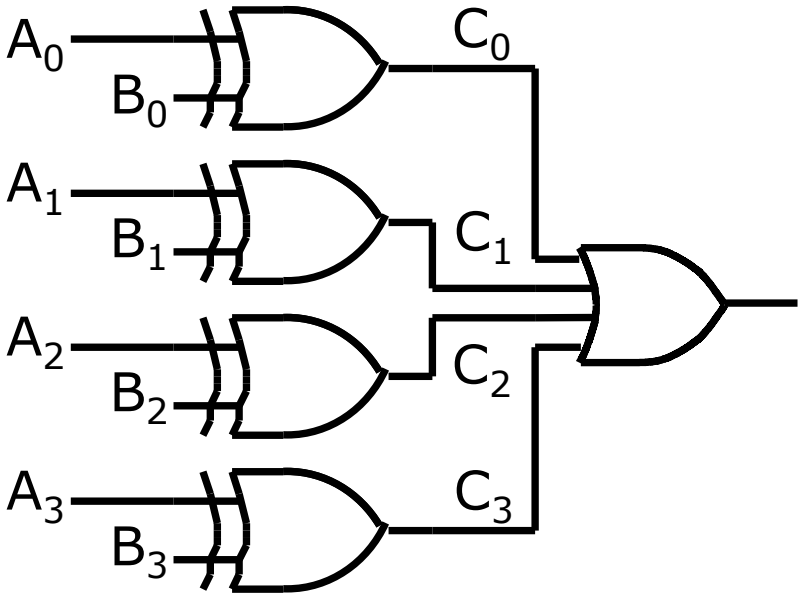
f=**1**, ko velja
(A == B)



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

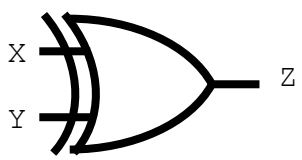
"Equality comparator"

4 bitni primerjalnik enakosti



f=**0**, ko velja
(A == B)

(XOR)



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

$Z = (X \oplus Y)$

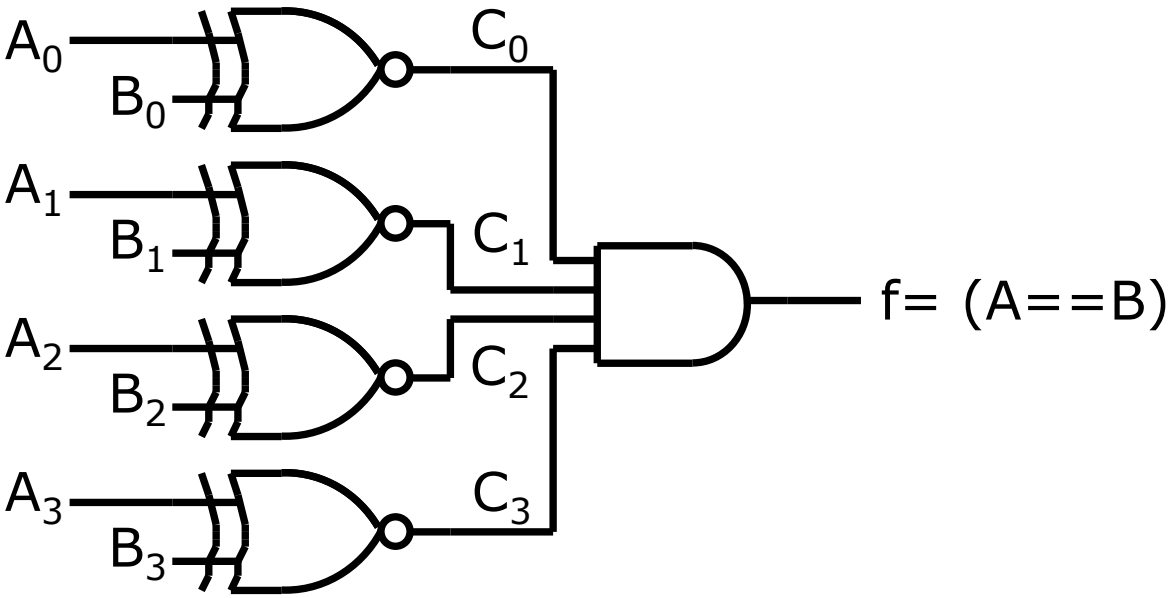
"Equality comparator"

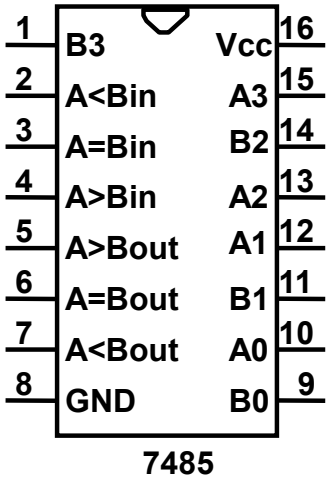
4 bitni primerjalnik velikosti

Primerja števili A in B. Kdaj bo A>B? $A=A_3A_2A_1A_0$ in $B=B_3B_2B_1B_0$

- Če je A=9 in B=7 je jasno A>B. Zakaj?
  A=1001= $A_3A_2A_1A_0$
  B=0111= $B_3B_2B_1B_0$
- Velja $A_3>B_3$, oziroma $A_3 \cdot B_3' = 1$
- Če je A=13 in B=11 je jasno A>B. Zakaj?
  A=1101
  B=1011
- Zato ker velja $A_2>B_2$, torej pišemo $A_2 \cdot B_2' = 1$
  ob pogoju, da sta A_3 in B_3 enaka
- Če je A=11 in B=10 je spet jasno A>B. Zakaj?
  A=1011
  B=1010
- Zato ker velja $A_0>B_0$, torej pišemo $A_0 \cdot B_0' = 1$
  ob pogoju, da sta enaka A_3 in B_3, A_2 in B_2, A_1 in B_1

4 bitni primerjalnik velikosti

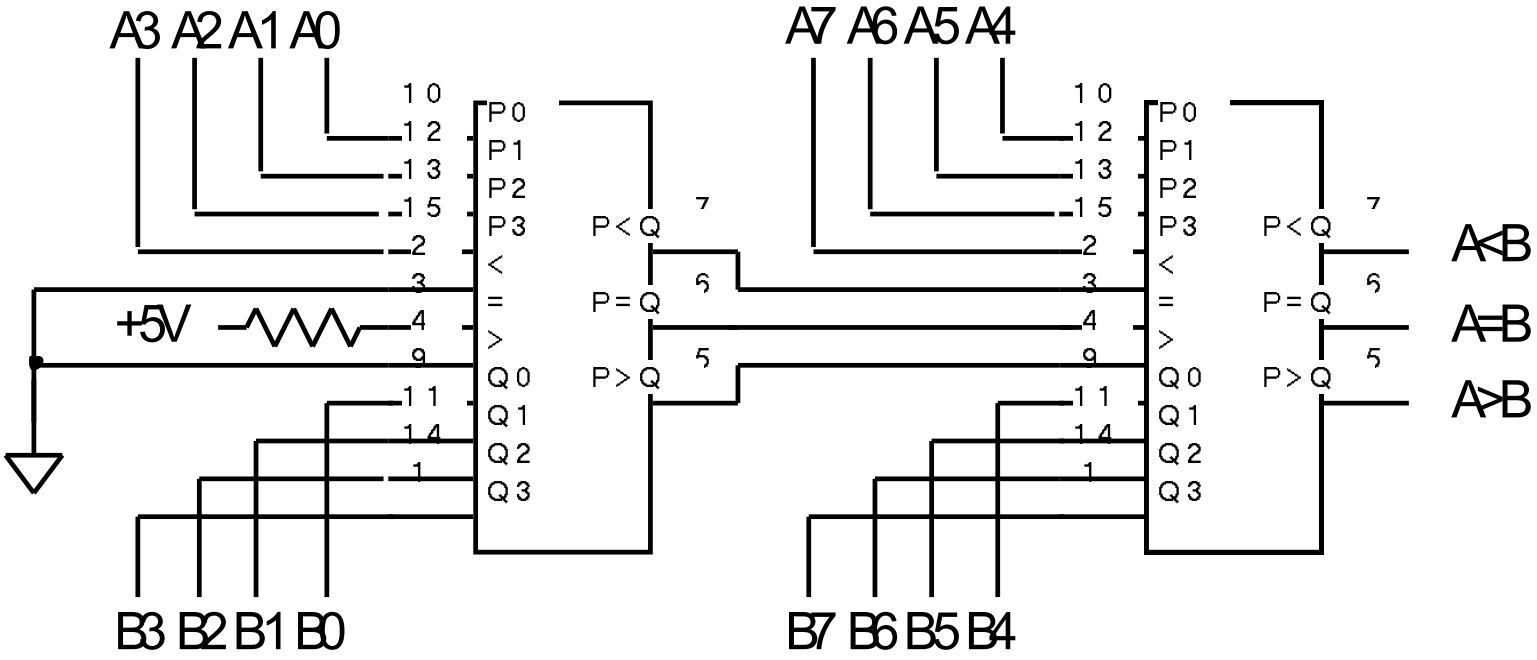




$A > B \rightarrow f = A_3 \cdot B_3' + C_3 \cdot A_2 \cdot B_2' + C_3 \cdot C_2 \cdot A_1 \cdot B_1' + C_3 \cdot C_2 \cdot C_1 \cdot A_0 \cdot B_0'$

Večbitni primerjalnik velikosti

Kako bi določili A>B? Če bi šli neposredno primerjati vse kombinacije (4 bite A + 4 bite B) bi imeli 256 kombinacij.



Kaskadna vezava dveh 7485 primerjalnikov:
8-bitni primerjalnik velikosti

Večbitni primerjalnik velikosti v VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity Comparator is
generic(n: natural :=2);
    port(
            A: in std_logic_vector(n-1 downto 0);
            B: in std_logic_vector(n-1 downto 0);
            less, equal, greater: out std_logic
    );
end Comparator;
architecture arch of Comparator is
begin
    process(A,B)
    begin
            if (A<B) then
                        less <= '1';
                        equal <= '0';
                        greater <= '0';
            elsif (A=B) then
                        less <= '0';
                        equal <= '1';
                        greater <= '0';
            else
                        less <= '0';
                        equal <= '0';
                        greater <= '1';
            end if;
    end process;
end arch;
```

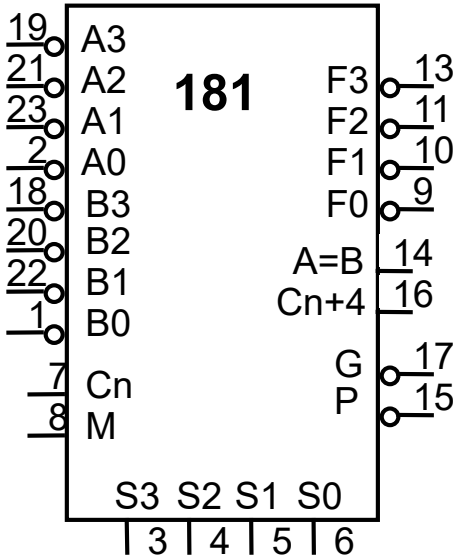
VHDL operator	Opis
=	Enakost
/=	Neenakost
<	Manj kot
<=	Manjši ali enak
>	Večji
>=	Večji ali enak

Razvoj digitalnih sistemov

Gradniki kombinacijskih vezij:
Aritmetično logična enota

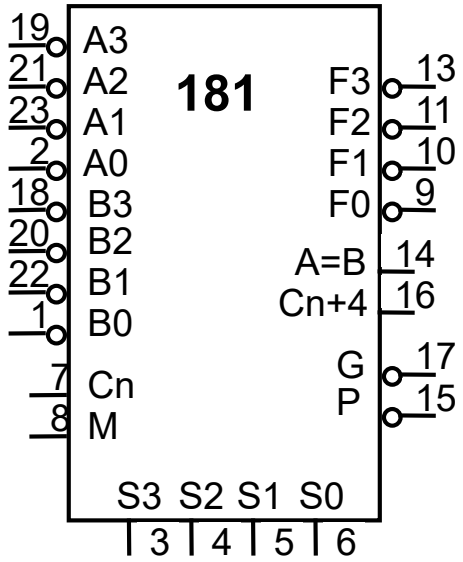
74181 - aritmetično logična enota

- Kombinacijsko vezje, ki izvaja različne aritmetične in logične operacije.
- Vhodi: A[], B[], C_n, S, M
- Izhodi: F, C_{n+4}, P, G, A==B
- M določa tip operacije:
  - M=0 za aritmetične in
  - M=1 za logične operacije.
- Vhodi S(3:0) določajo konkretno operacijo
- Realizira 48 različnih operacij.



74181 - aritmetično logična enota

$S_3S_2S_1S_0$	M=1	C=0	C=1
		M=0	
0 0 0 0	F = A′	F = A minus 1	F = A
0 0 0 1	F = (AB)′	F = AB minus 1	F = AB
0 0 1 0	F = A′∨B	F = AB′ minus 1	F = AB′
0 0 1 1	F = 1	F = minus 1 (2′K)	F = 0
0 1 0 0	F=(A∨B)′	F=A plus (A∨B′)	F = A plus (A ∨ B′) plus 1
0 1 0 1	F = B′	F=AB plus (A∨B′)	F = AB plus (A ∨ B′) plus 1
0 1 1 0	F=A≡B	F=A minus B minus 1	F = A minus B
0 1 1 1	F=A∨B′	F = A ∨ B′	F = (A ∨ B′) plus 1
1 0 0 0	F = A′B	F=A plus (A∨B)	F = A plus (A ∨ B) plus 1
1 0 0 1	F=A⊕B	F = A plus B	F = A plus B plus 1
1 0 1 0	F = B	F=AB′ plus (A∨B)	F = AB plus (A ∨ B) plus 1
1 0 1 1	F = A∨B	F = A ∨ B	F = (A ∨ B) plus 1
1 1 0 0	F = 0	F = A plus A	F = A plus A plus 1
1 1 0 1	F = AB′	F = AB plus A	F = AB plus A plus 1
1 1 1 0	F = AB	F = AB′ plus A	F = AB′ plus A plus
1 1 1 1	F = A	F = A	F = A plus 1



$F_i=f(A_i,B_i)$ i=0..3

Primer: M=1, S=1110

$F_0=A_0$ AND B_0

$F_1=A_1$ AND B_1

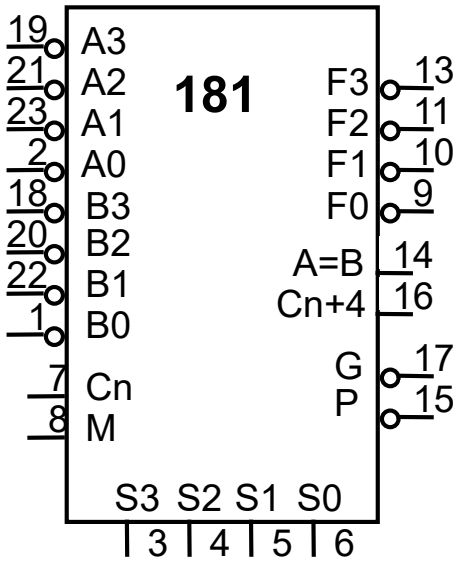
$F_2=A_2$ AND B_2

$F_3=A_3$ AND B_3

74181 - aritmetično logična enota

Kako se izognemo uporabi inverterjev na vhodih in izhodih ALU?

- Pri logičnih funkcijah uporabimo dualno funkcijo:
  $f(x_1 \ldots x_n)=(f(x_1', x_2', x_3' \ldots, x_n'))'$
  - dualna funkcija AND↔OR
  - dualna funkcija XOR↔EQU.
- Pri aritmetičnih funkcijah negiramo C_n in C_{n+4}.
  - Pri seštevanju vzamemo S=1001 (F = A plus B plus 1) ➔ C_n=1, ne C_n=0.
  - Izhodni prenos invertiramo



Normalno seštevanje/odštevanje

	0011	3_{10}		0011	3_{10}
+	0010	2_{10}	-	1110	$(-2)_{10}$
	0101	5_{10}		0101	5_{10}

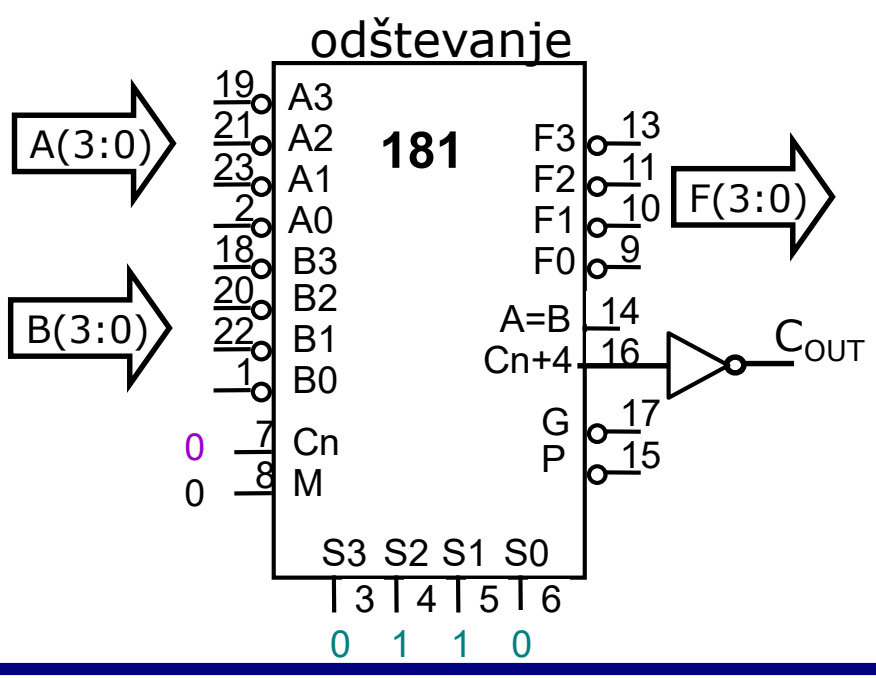
Seštevanje/odštevanje s 74181 in pozitivno logiko vhodov

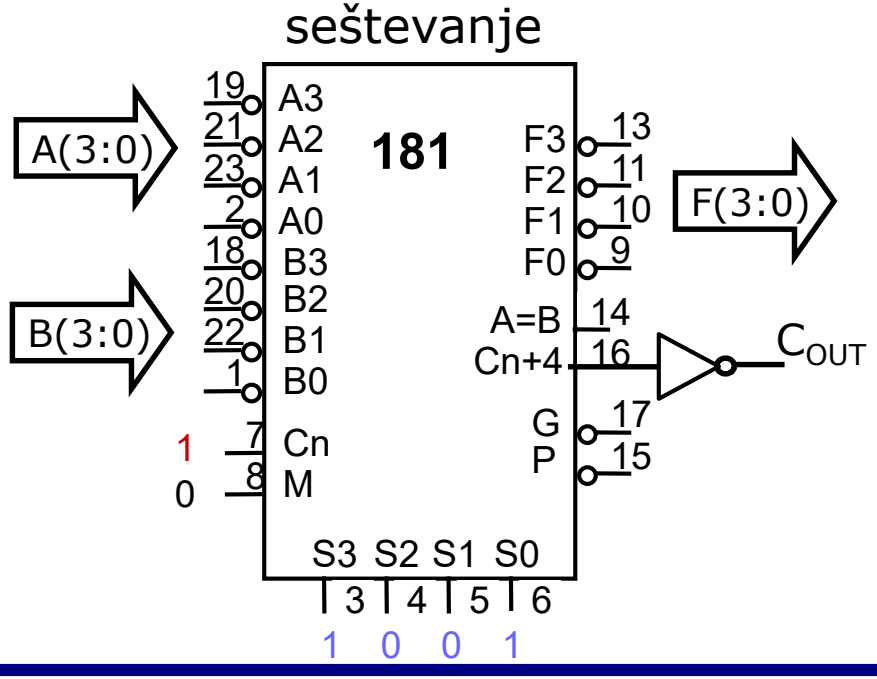
	1100			1100	
+	1101		-	0001	
+	1	C_n			
1	1010		0	1011	
(C_{n+4})	0101	$= 5_{10}$	(C_{n+4})	0101	$= 5_{10}$
0			1		

Seštevanje in odštevanje s 74181

Pri aritmetičnih funkcijah vzamemo negirane vrednosti C_n in C_{n+4}:

- Pri seštevanju vzamemo
  S=1001 (F = A plus B plus 1) ➔ C_n=1, ne C_n=0.
- Izhodni prenos C_{n+4} invertiramo.
- Pri odštevanju vzamemo
  S=0110 (F = A minus B) ➔ C_n=0, ne C_n=1.
- Izhodni prenos C_{n+4} invertiramo.





Razvoj digitalnih sistemov

Predstavitev števil in aritmetična vezja:

Predstavitev števil in nepredznačeno seštevanje

Števila v različnih sistemih

X_{10}	X_2	X_8	X_{16}
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7

X_{10}	X_2	X_8	X_{16}
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Pretvorba BIN ➔ HEX oz. BIN ➔ OCT

- Združujemo binarne števke v skupine (po 4 za HEX oz. 4 za OCT) in vsaki skupini določimo ustrezno šestnajstiško števko.

0110	1011	0111	
6	B	7	

- $X_2 \rightarrow X_8$:

011	010	110	111
3	2	6	7

- $X_{16} \rightarrow X_2$:

A	1	9
1010	0001	1001

- $X_8 \rightarrow X_2$:

5	0	3	1
101	000	011	001

Seštevanje dvojiških števil

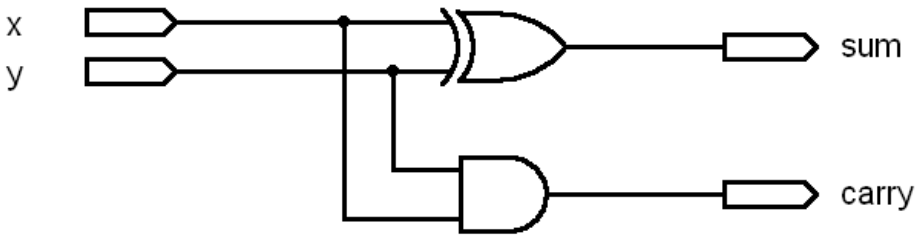
x	0	0	1	1
+ y	+ 0	+ 1	+ 0	+ 1
c s	0 0	0 1	0 1	1 0

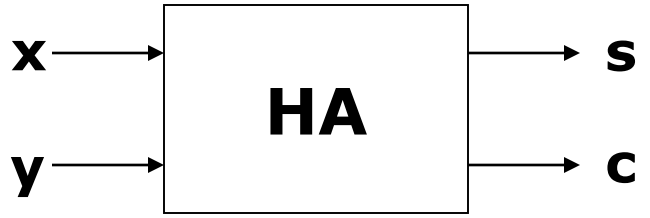
Prenos (Carry)

Vsota (Sum)

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Polovični seštevalnik (Half Adder oz. HA)





$s = (x \oplus y)$

$c = x \cdot y$

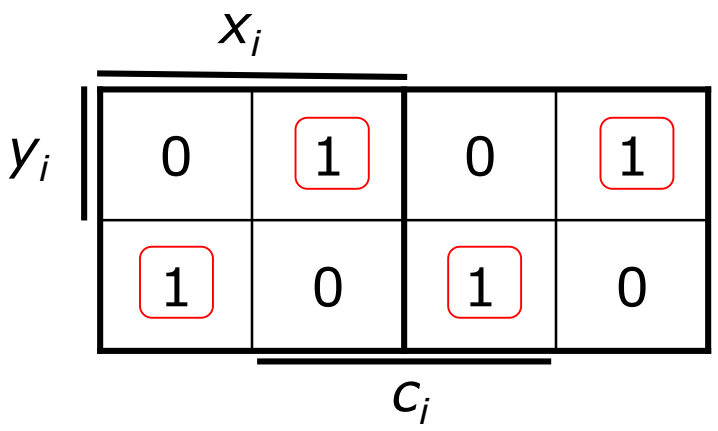
Seštevanje dvojiških števil

- Večja števila imajo več mest
  - Princip seštevanja je isti kot za eno mesto (seštevamo soležne bite)
  - Na vsakem mestu *i*, ima lahko seštevanje tudi **vhodni prenos** (carry-in) s prejšnjega mesta *i-1*

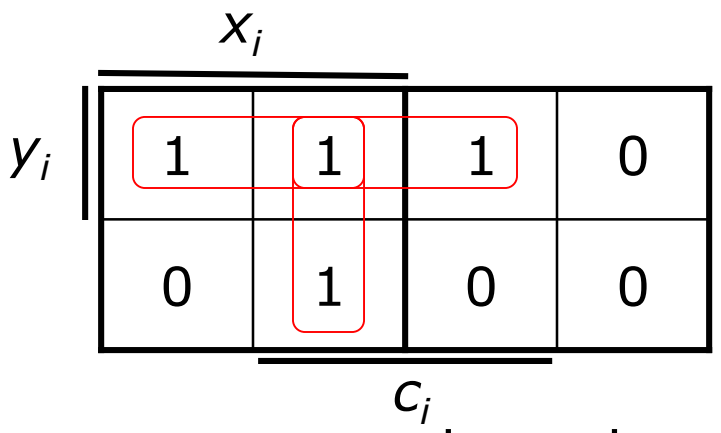
$X=x_4x_3x_2x_1x_0$	0 1 1 1 1	$(15)_{10}$
$Y=y_4y_3y_2y_1y_0$	+ 0 1 0 1 0	$(10)_{10}$
	1 1 1 0	← generirani prenosi
$S=s_4s_3s_2s_1s_0$	1 1 0 0 1	$(25)_{10}$

Polni seštevalnik (FA)

c_i	x_i	y_i	c_{i+1}	s_{i+1}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

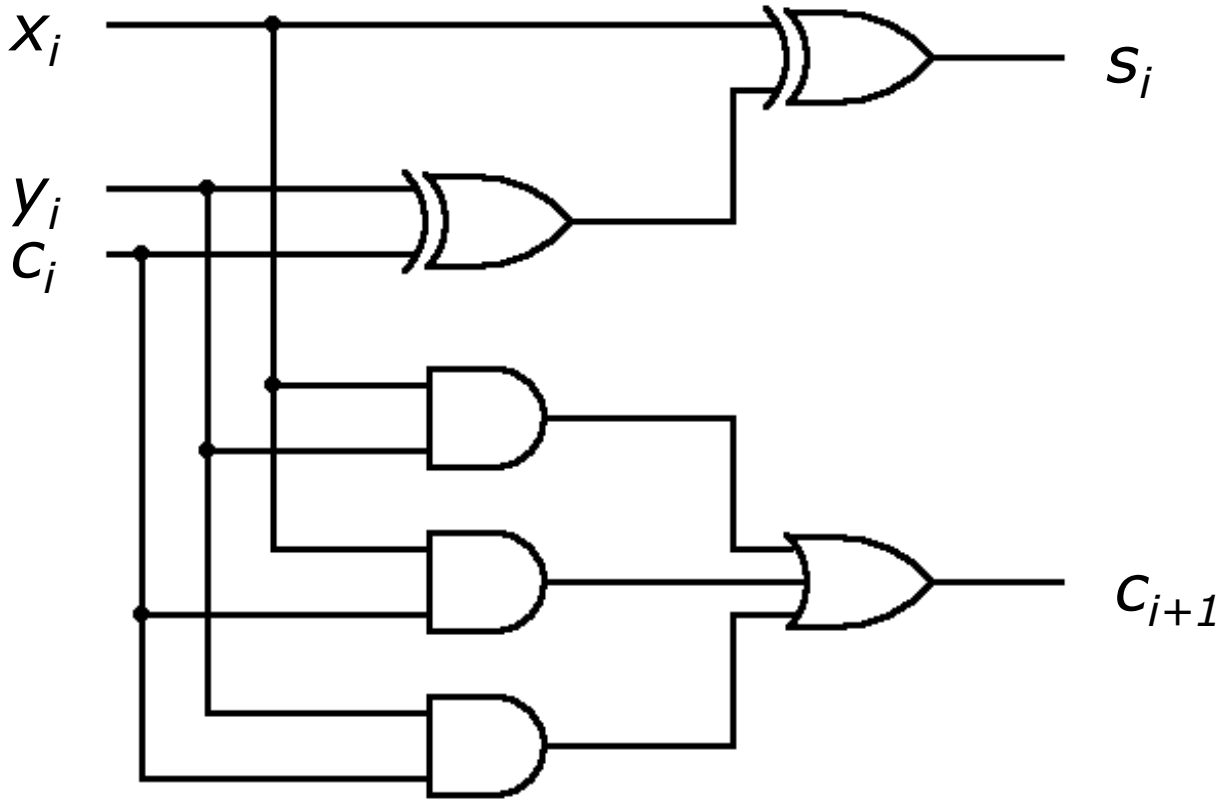


$$s_{i+1} = x_i \oplus y_i \oplus c_i$$



$$c_{i+1} = x_i \cdot y_i + y_i \cdot c_i + x_i \cdot c_i$$

Vezje polnega seštevalnika (FA)



FA sestavljen iz HA

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Iz analize HA

$s_{HA} = (x_i \oplus y_i)$

$c_{HA} = x_i \cdot y_i$

$c_{i+1} = c_i' x_i y_i + c_i x_i' y_i + c_i x_i y_i' + c_i x_i y_i$

$c_{i+1} = c_i (x_i' y_i + x_i y_i') + x_i y_i (c_i' + c_i)$

$c_{i+1} = c_i (x_i \oplus y_i) + x_i y_i$

$c_{i+1} = c_i s_{HA} + c_{HA}$

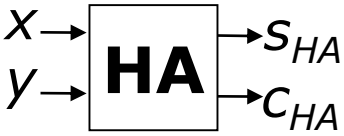
$s_i = c_i' x_i' y_i + c_i' x_i y_i' + c_i x_i' y_i' + c_i x_i y_i$

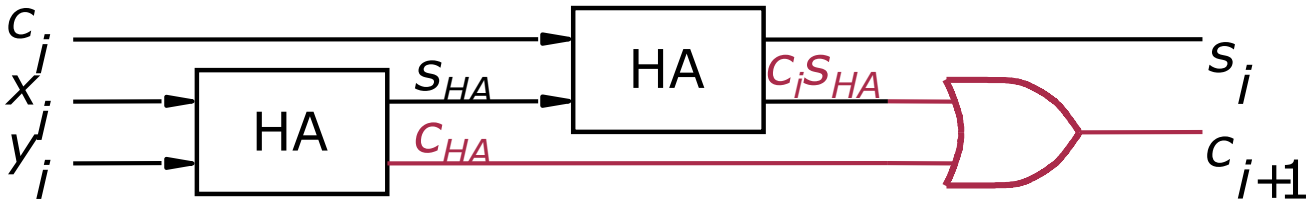
$s_i = c_i' (x_i \oplus y_i) + c_i (x_i \oplus y_i)'$

$s_i = c_i' s_{HA} + c_i s_{HA}'$

$s_i = c_i \oplus s_{HA} = c_i \oplus x_i \oplus y_i$

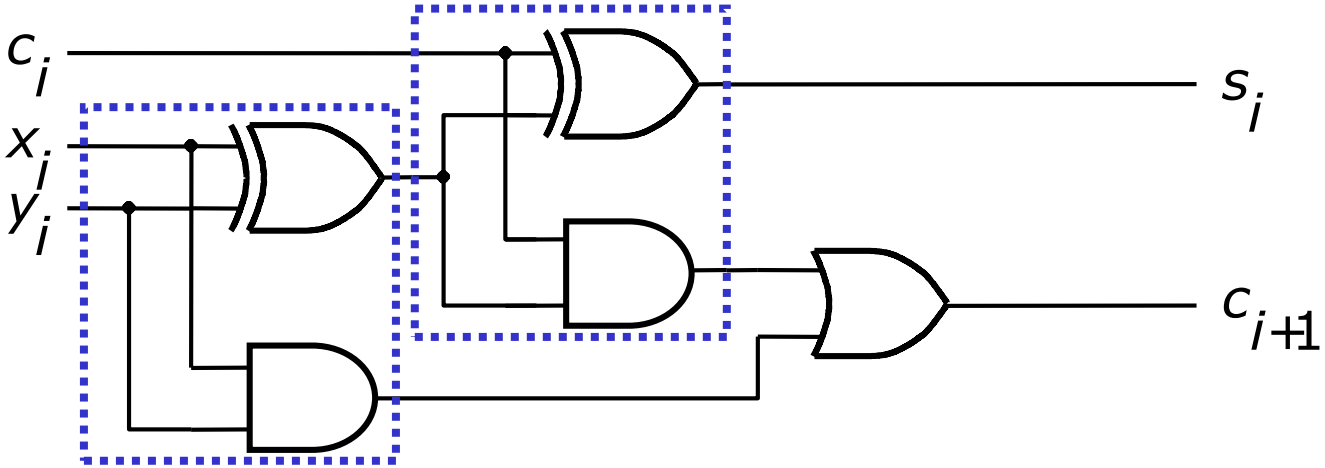
FA sestavljen iz HA

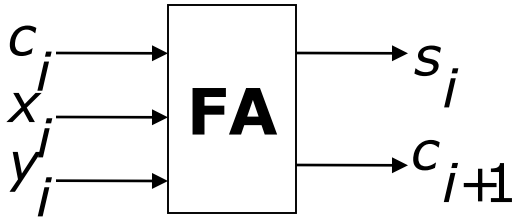




$s_i = c_i \oplus s_{HA}$

$c_{i+1} = c_i\, s_{HA} + c_{HA}$





Polni seštevalnik v jeziku VHDL

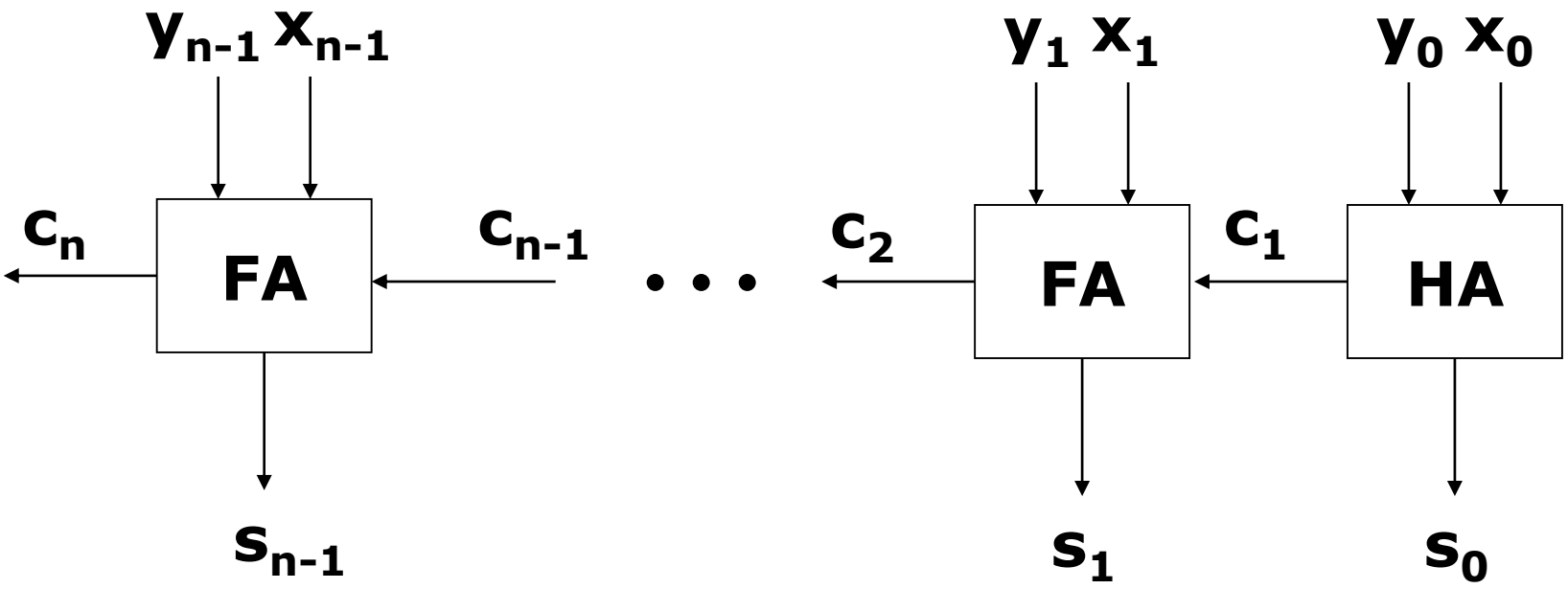
Osnovni enačbi iz analize: $s_{i+1}=x_i \oplus y_i \oplus c_i$

$$c_{i+1}=x_i \cdot y_i + y_i \cdot c_i + x_i \cdot c_i$$

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY fulladd IS
     PORT (   Cin, x, y  : IN    STD_LOGIC ;
              s, Cout    : OUT STD_LOGIC ) ;
END fulladd ;
ARCHITECTURE arch OF fulladd IS
BEGIN
     s <= x XOR y XOR Cin ;
     Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y) ;
END arch;
```

Ripple-carry (RC) seštevalnik

seštevanje n bitnih števil s=x+y



Odštevanje dvojiških števil

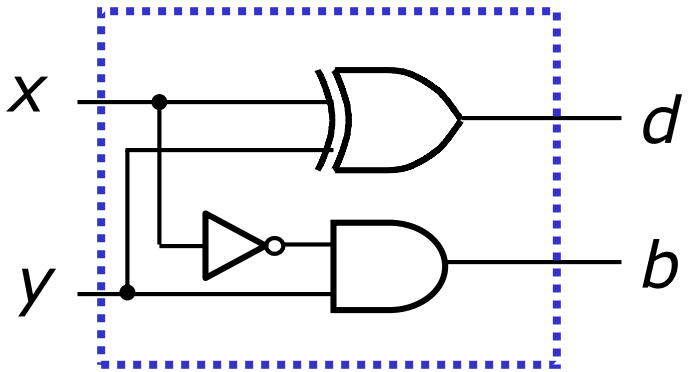
x	0	0	1	1
- y	- 0	- 1	- 0	- 1
b d	0 0	1 1	0 1	0 0

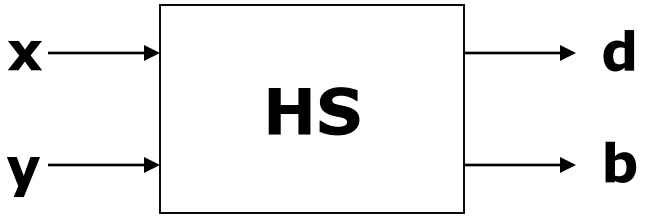
Sposodek (Borrow)

Razlika (Difference)

x	*y*	*b*	*d*
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

Polovični odštevalnik (Half Subtractor oz. HS)





$d = (x \oplus y)$

$b = x' \cdot y$

Odštevanje dvojiških števil

- Večja števila imajo več mest
  - Princip odštevanja je isti kot za eno mesto (odštevamo soležne bite)
  - Na vsakem mestu *i*, ima lahko odštevanje tudi **vhodni sposodek** (borrow-in) z višjega mesta *i+1*

$X=x_4x_3x_2x_1x_0$		0 1 1 0 0	$(12)_{10}$
$Y=y_4y_3y_2y_1y_0$	-	0 0 1 1 1	$(07)_{10}$
	-	0 1 1 1	← generirani sposodki
$D=d_4d_3d_2d_1d_0$		0 0 1 0 1	$(05)_{10}$

Odštevanje dvojiških števil

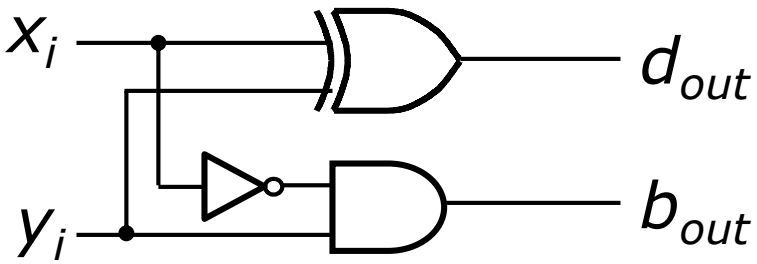
$X=x_7x_6x_5x_4x_3x_2x_1x_0$	1 1 1 0 0 1 1 1	$(E7)_{16}$	$(231)_{10}$
$Y=y_7y_6y_5y_4y_3y_2y_1y_0$	- 1 0 1 0 1 1 1 1	$(AF)_{16}$	$(175)_{10}$
	- 0 1 1 1 0 0 0		
$D=d_7d_6d_5d_4d_3d_2d_1d_0$	0 0 1 1 1 0 0 0	$(38)_{16}$	$(56)_{10}$

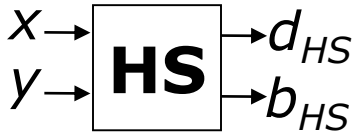
Polovični odštevalnik (HS)

x_i	y_i	d_{out}	b_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

$$d_{out}=x_i \oplus y_i$$

$$b_{out}=x_i' \cdot y_i$$





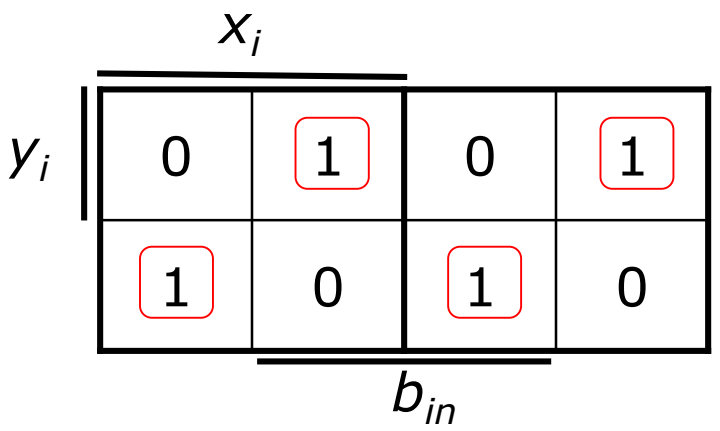
HS = half subtractor

Polni odštevalnik (FS)

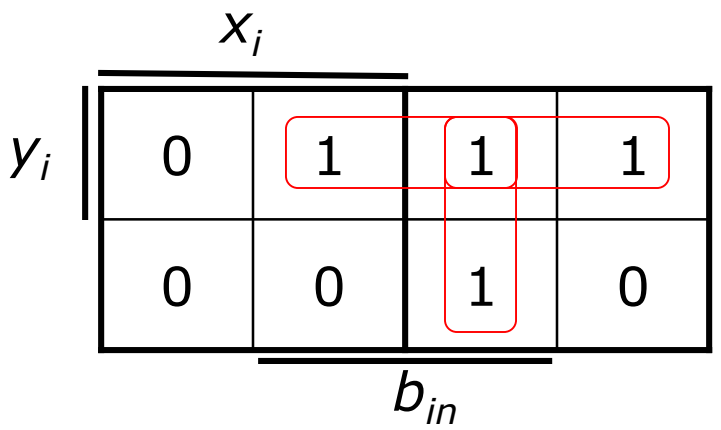
$$d_{out}=(x_i-y_i)-b_{in}$$

x_i	y_i	b_{in}	d_{out}	b_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

FS = full subtractor

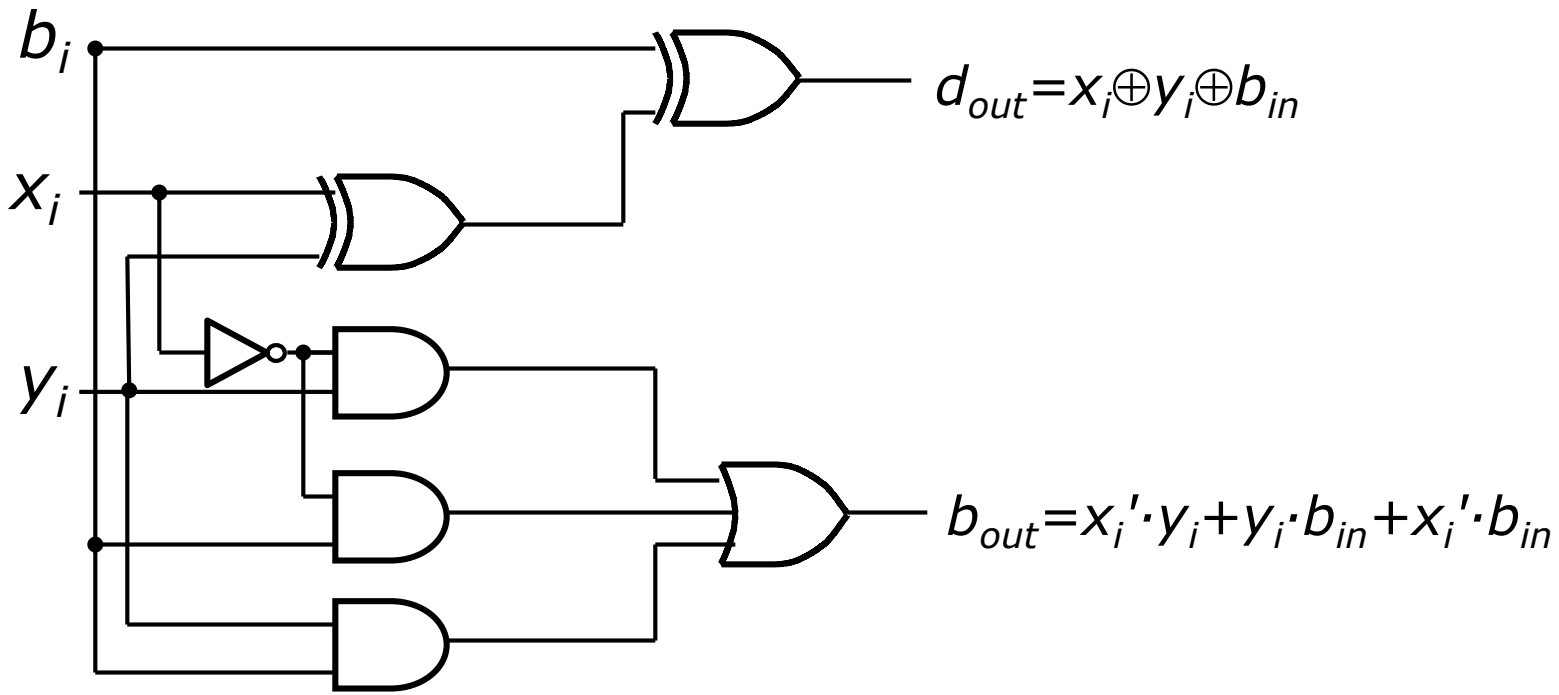


$$d_{out}=x_i \oplus y_i \oplus b_{in}$$



$$b_{out}=x_i' \cdot y_i + y_i \cdot b_{in} + x_i' \cdot b_{in}$$

Vezje polnega odštevalnika (FS)



FS sestavljen iz HS

x_i	y_i	b_{in}	d_{out}	b_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Iz analize HS:

$d_{out}=x_i \oplus y_i$

$b_{out}=x_i' \cdot y_i$

$b_{out} = b_{in}x_i'y_i'+b_{in}'x_i'y_i+b_{in}x_i'y_i+b_{in}x_iy_i$

$b_{out} = b_{in}x_i'y_i'+b_{in}'x_i'y_i+b_{in}x_i'y_i+b_{in}x_iy_i$

$b_{out} = b_{in} \cdot (x_i'y_i'+x_iy_i)+(b_{in}'+b_{in}) \cdot x_i'y_i$

$b_{out} = b_{in} \cdot (x_i \oplus y_i)'+x_i'y_i$

$d_{01}=x_i \oplus y_i \qquad b_{01}=x_i' \cdot y_i$

$b_{out}=b_{in} \cdot d_{01}'+b_{01} = b_{02}+b_{01}$

$d_{out} = x_i'y_i'b_{in}+x_i'y_ib_{in}'+x_iy_i'b_{in}'+x_iy_ib_{in}$

$d_{out} = x_i'y_i'b_{in}+x_i'y_ib_{in}'+x_iy_i'b_{in}'+x_iy_ib_{in}$

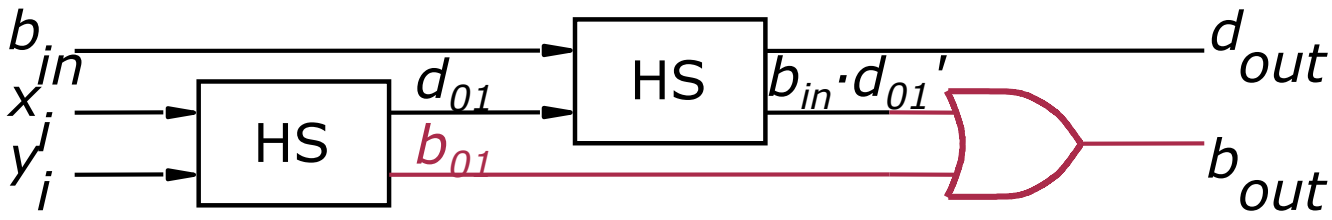
$d_{out} = (x_i'y_i'+x_iy_i)b_{in}+x_i'y_ib_{in}'+x_iy_i'b_{in}'$

$d_{out} = (x_i \text{ xor } y_i)' \cdot b_{in}+(x_i \text{ xor } y_i) \cdot b_{in}'$

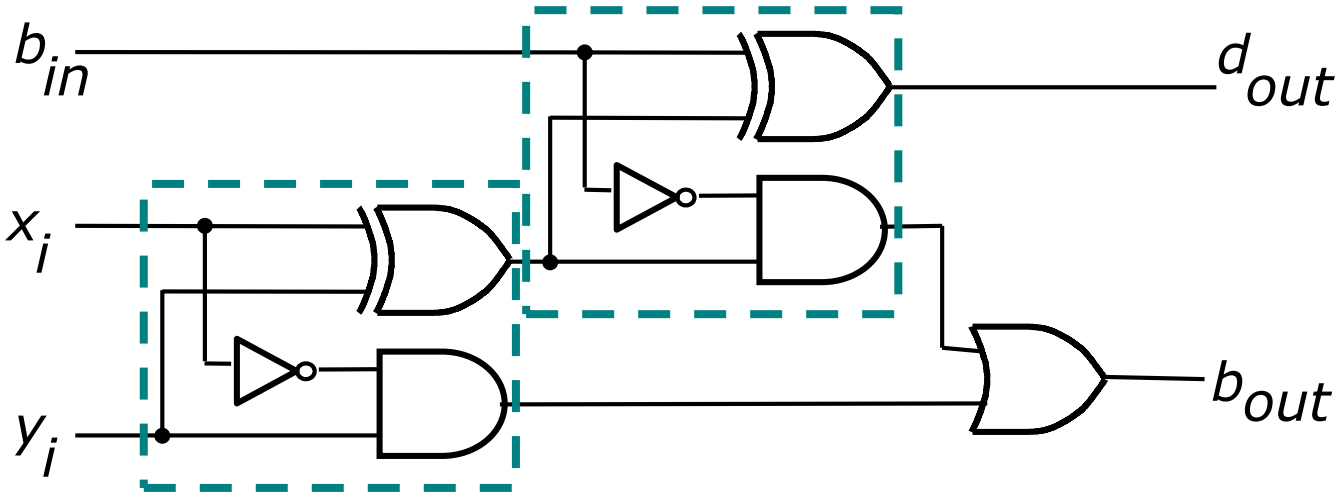
$d_{out} = (x_i \text{ xor } y_i)' \cdot b_{in}+(x_i \text{ xor } y_i) \cdot b_{in}'$

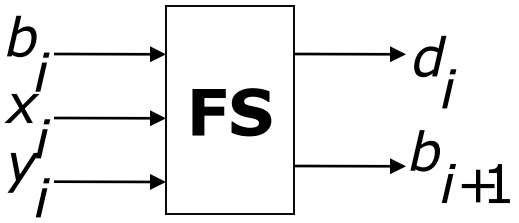
$d_{out}=x_i \oplus y_i \oplus b_{in}$

FS sestavljen iz HS



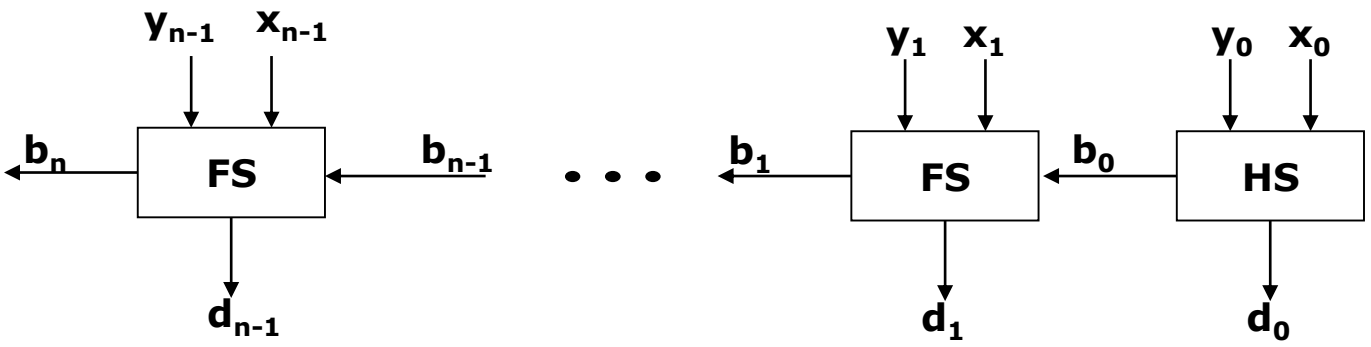
$d_{01} = x_i \oplus y_i$ $d_{out} = d_{01} \oplus b_{in}$ $b_{out} = b_{in}\, d_{01}' + b_{01}$





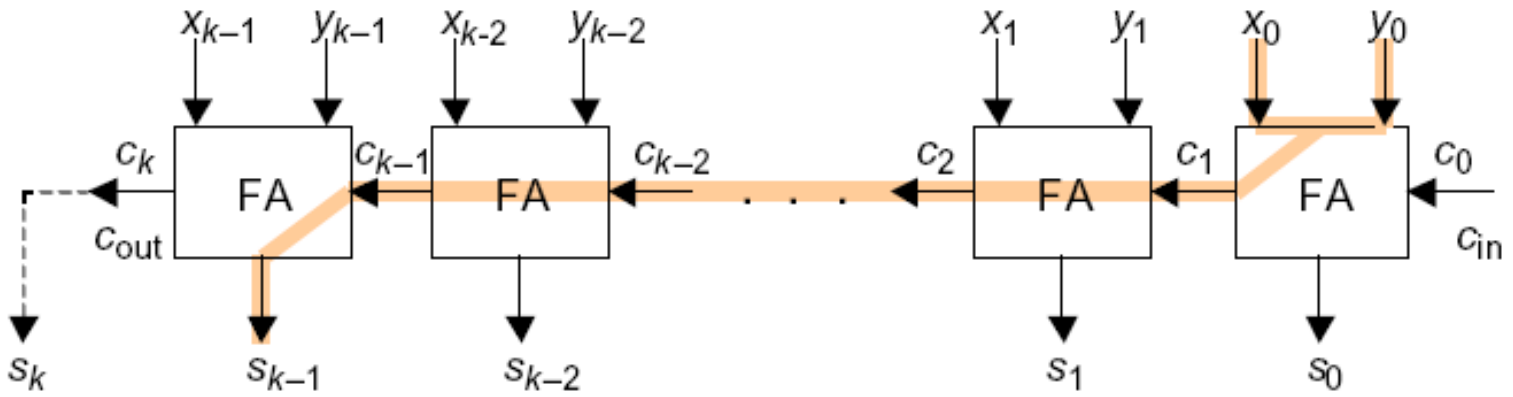
Ripple-carry (RC) odštevalnik

odštevanje n bitnih števil s=x-y



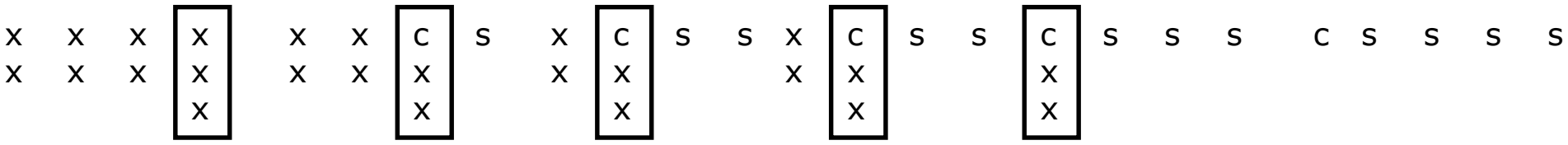
Kritična pot RC seštevalnika

$$T_{\text{ripple-add}} = T_{FA}(x,y \rightarrow c_{out}) + (k-2) \times T_{FA}(c_{in} \rightarrow c_{out}) + T_{FA}(c_{in} \rightarrow s)$$

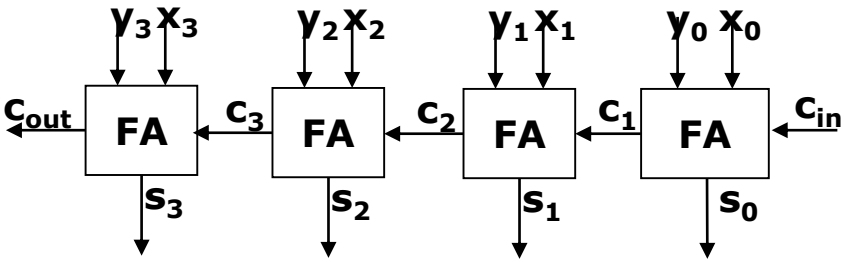


Seštevanje dveh 4-bitnih števil – simbolični zapis:

t=0 1·Δt 2·Δt 3·Δt 4·Δt rezultat



4-bitni RC seštevalnik v VHDL




```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY adder4 IS
      PORT (
      Cin                  : IN STD_LOGIC ;
      x3, x2, x1, x0       : IN STD_LOGIC ;
      y3, y2, y1, y0       : IN STD_LOGIC ;
      s3, s2, s1, s0       : OUT STD_LOGIC ;
      Cout            : OUT STD_LOGIC
);
END adder4 ;
ARCHITECTURE arch OF adder4 IS
      SIGNAL c1, c2, c3 : STD_LOGIC ;
      COMPONENT fulladd
            PORT (   Cin, x, y : IN       STD_LOGIC ;
                     Cout      : OUT      STD_LOGIC );
      END COMPONENT ;
BEGIN
      stage0: fulladd PORT MAP ( Cin, x0, y0, s0, c1 );
      stage1: fulladd PORT MAP ( c1, x1, y1, s1, c2 );
      stage2: fulladd PORT MAP ( c2, x2, y2, s2, c3 );
      stage3: fulladd PORT MAP (Cin => c3, Cout => Cout, x => x3, y => y3, s => s3  ) ;
END arch ;
```

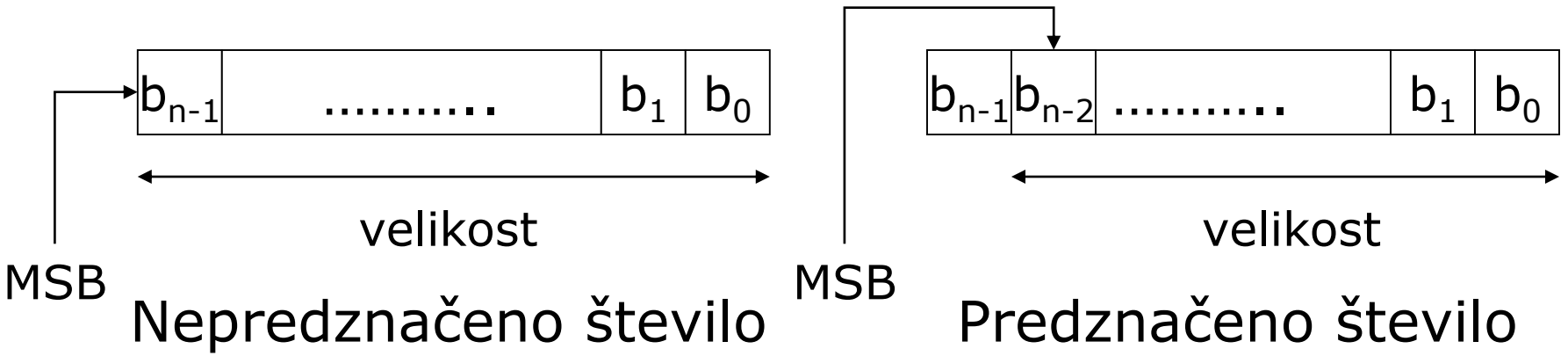
Razvoj digitalnih sistemov

Predstavitev števil in aritmetična vezja:

Predznačena števila, dvojiški seštevalniki in odštevalniki

Predznačena števila

- Za predznačena števila v dvojiškem sistemu je predznak števila (sign) označen z MSB bitom
  - 0 = pozitivno število
  - 1 = negativno število
- Za n-bitno število bo preostalih *n*-1 bitov predstavljalo velikost števila (magnitude)



Negativna števila

- Negativne vrednosti predznačenih števil lahko zapišemo v treh najbolj pogostih oblikah
  - Predznak-velikost
  - Eniški komplement
  - Dvojiški komplement
- Predznak-velikost zapis primer za 4-bitna števila

  +5=0101 -5=1101

  +3=0011 -3=1011

  +7=0111 -7=1111

Predstavitve predznačenih dvojiških števil

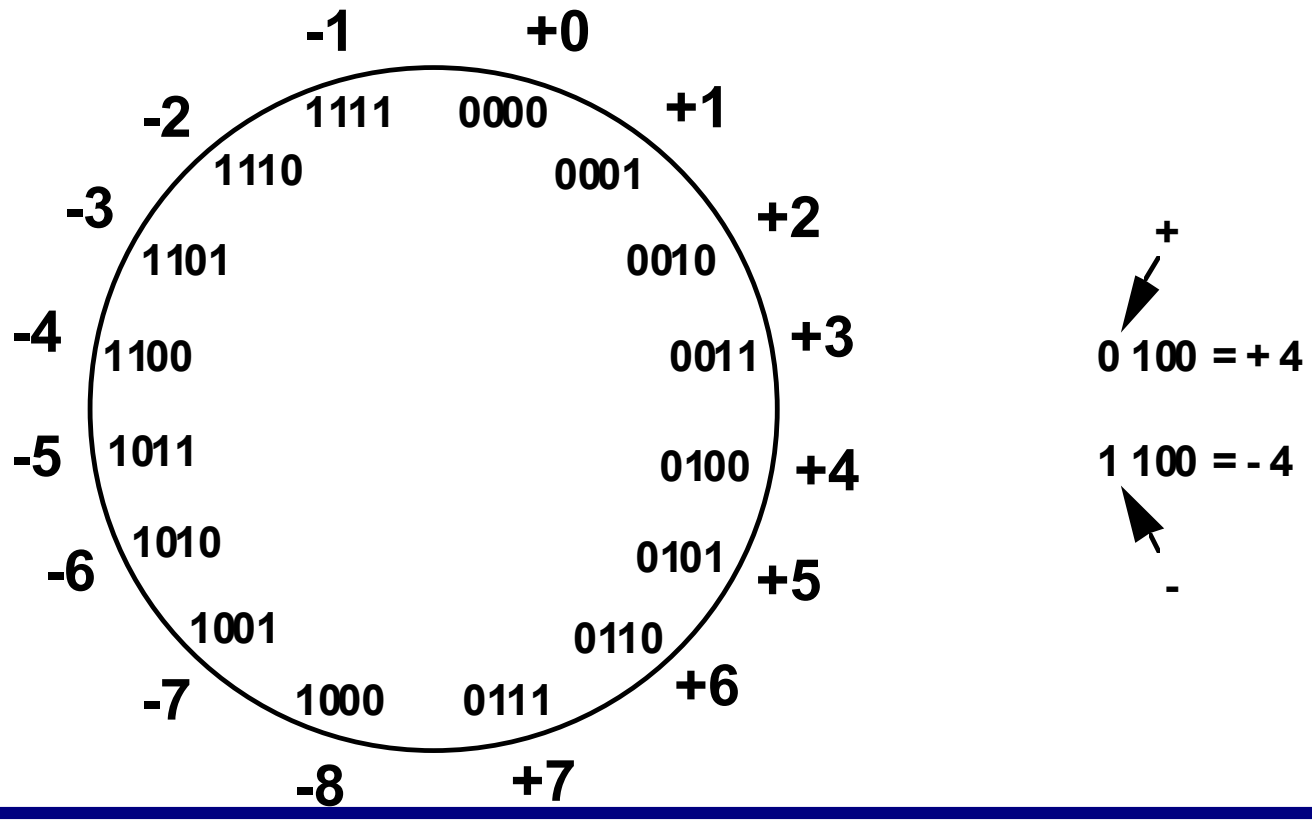
Izmed različnih predstavitev omenimo:

- predznak-veličina (sign-magnitude, SM) – zoprn za seštevanje/odštevanje.
- eniški komplement (1'K),
- dvojiški komplement (2'K) – zelo primeren za seštevanje/odštevanje

b_2	b_1	b_0	SM	1'K	2'K
0	0	0	+0	+0	+0
0	0	1	1	1	1
0	1	0	2	2	2
0	1	1	3	3	3
1	0	0	-0	-3	-4
1	0	1	-1	-2	-3
1	1	0	-2	-1	-2
1	1	1	-3	-0	-1

Dvojiški komplement (2'K)

- Ima samo eno predstavitev za število nič.
- Ima eno negativno število v obsegu več kot pozitivno: obsega zapis n-bitnih števil $[-2^{n-1} \ldots +2^{n-1}-1]$



Dvojiški komplement

- Z eniškim komplementom *n*-bitno negativno število *K* zapišemo v dvojiškem komplementu:

  $K=(2^n)-P$

- Če je *n=4*, potem

  $K=2^4-P=(16)_{10}-P=(10000)_2-P$

  $-5=(16)_{10}-5=(10000)_2-(0101)_2=(1011)_2$

  $-3=(16)_{10}-3=(10000)_2-(0011)_2=(1101)_2$

Ročno določanje dvojiškega komplementa

- Primer
  B=00110100
- Dvojiški komplement B je

B=00110100 ➔ K=11001100

Spremenjeno Nespremenjeno

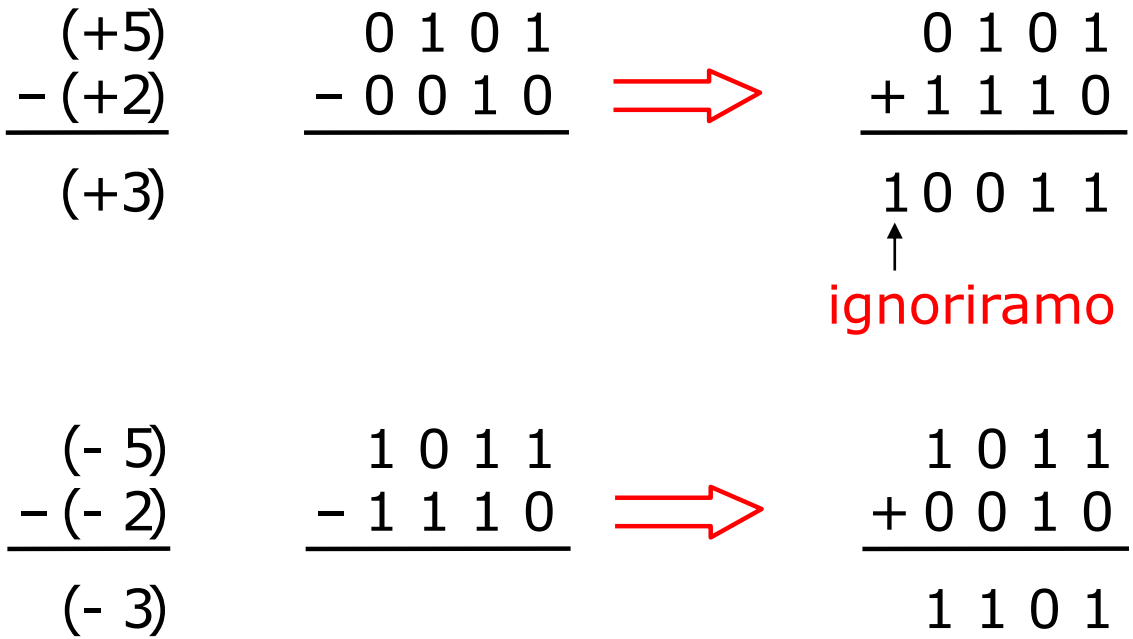
Operacije z dvojiškim komplementom

(+5)	0 1 0 1	(–5)	1 0 1 1
+(+2)	+0 0 1 0	+(+2)	+0 0 1 0
(+7)	0 1 1 1	(–3)	1 1 0 1
(+5)	0 1 0 1	(–5)	1 0 1 1
+ (–2)	+1 1 1 0	+ (–2)	+1 1 1 0
(+3)	1 0 0 1 1	(–7)	1 1 0 0 1

ignoriramo

ignoriramo

Odštevanje z dvojiškim komplementom



Preliv (overflow)

- Za zapis *predznačenega* števila uporabljamo *n* bitov
- Rezultat operacije je v območju $[-2^{n-1} \ldots +2^{n-1}-1]$
- Z *n*-bitnim zapisom v (2'K) lahko predstavimo števila v obsegu $[-2^{n-1} \ldots +2^{n-1}-1]$
  - Za $n=3$ ➔[-4..3], za $n=16$ ➔[-32768.. 32767]
- Če 32767 prištejemo 1, dobimo -32768
- To je neveljaven rezultat.

- Če rezultata ne moremo zapisati v območje predstavitve n-bitnega števila, potem govorimo o prelivu ali prekoračitvi (***arithmetic overflow***).

Primeri aritmetičnega preliva

$x_3\ x_2\ x_1\ x_0$
$+\ y_3\ y_2\ y_1\ y_0$
$c_4\ c_3\ c_2\ c_1$
$s_3\ s_2\ s_1\ s_0$

(+7)	0 1 1 1	(–7)	1 0 0 1
+(+2)	+0 0 1 0	+(+2)	+0 0 1 0
(+9)	1 0 0 1	(–5)	1 0 1 1
	$c_4 = 0$		$c_4 = 0$
	$c_3 = 1$		$c_3 = 0$

(+7)	0 1 1 1	(–7)	1 0 0 1
+(–2)	+1 1 1 0	+(–2)	+1 1 1 0
(+5)	1 0 1 0 1	(–9)	1 0 1 1 1
	$c_4 = 1$		$c_4 = 1$
	$c_3 = 1$		$c_3 = 0$

Če imajo števila različne predznake, se preliv ne zgodi!

Preliv v zapisu z 2'K

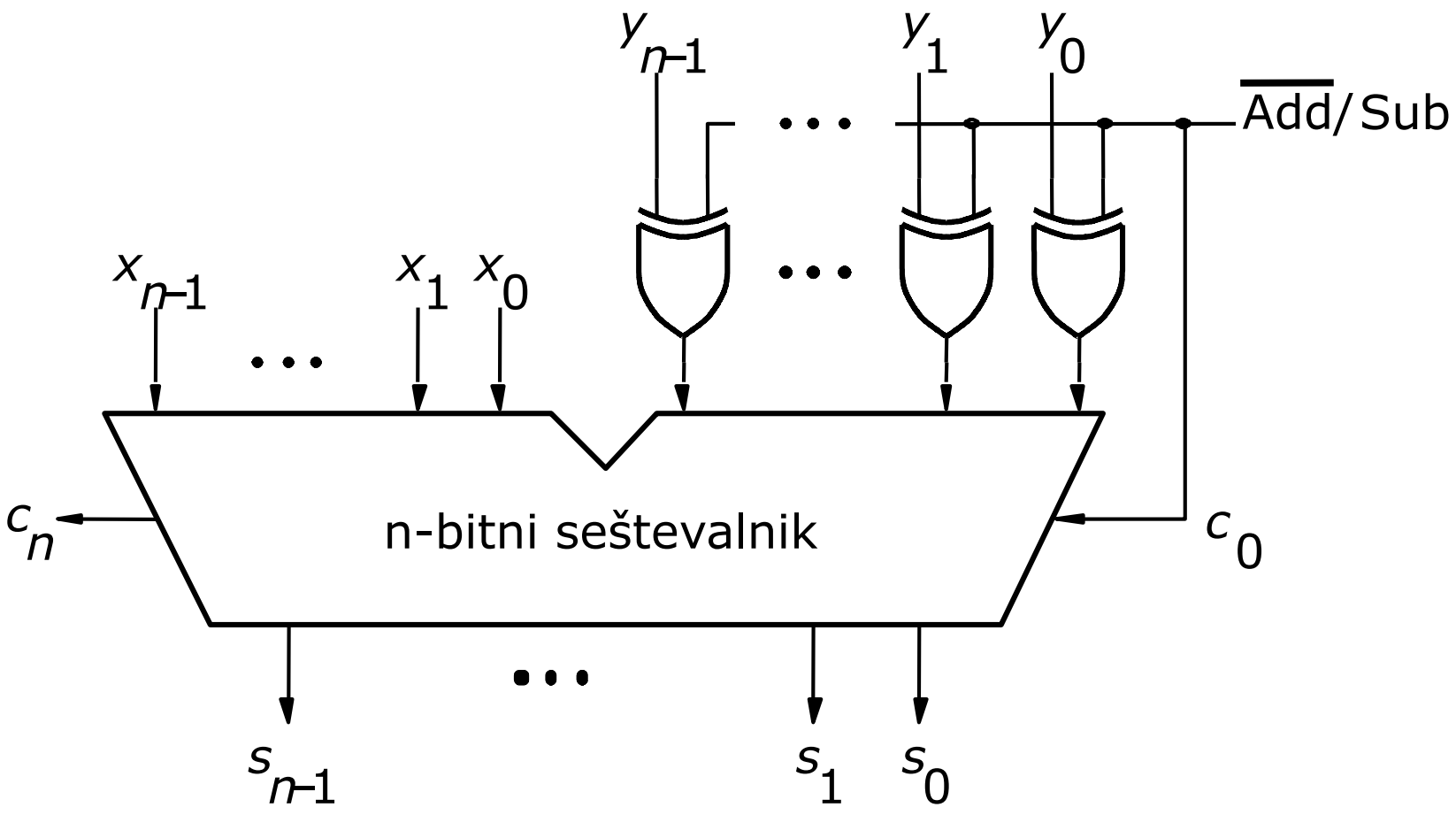
Zaznavanje preliva

Pozitivno	Negativno
+ Pozitivno	+ Negativno
= Negativno	= Pozitivno

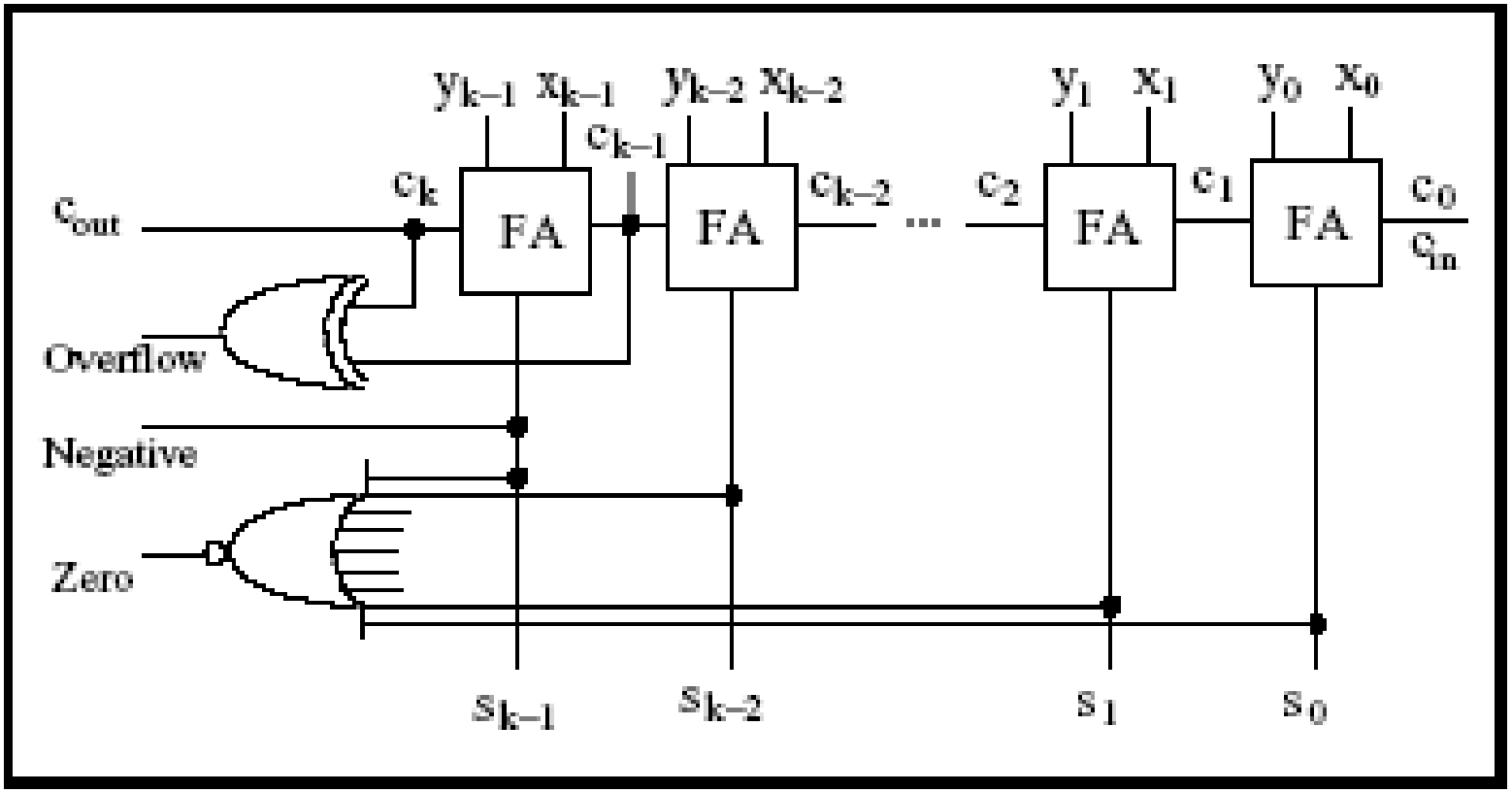
Določanje preliva

$$OF_{2'K} = \overline{a_{n-1}} \cdot \overline{b_{n-1}} \cdot s_{k-1} + a_{k-1} \cdot b_{k-1} \cdot \overline{s_{k-1}} = c_k \oplus c_{k-1}$$

Vezje seštevalnika in odštevalnika



NCVZ biti RC seštevalnika



Razvoj digitalnih sistemov

Predstavitev števil in
aritmetična vezja:

Hitre realizacije seštevalnikov,
kompromisi in primeri

Carry Look Ahead (CLA) seštevalnik

- Funkcijo izhodnega prenosa na i-ti stopnji lahko zapišemo kot

  $c_{i+1}=x_i \cdot y_i + x_i \cdot c_i + y_i \cdot c_i = x_i \cdot y_i + (x_i + y_i) \cdot c_i$
- Označimo $g_i = x_i y_i$ in $p_i = x_i + y_i$, tako da je $c_{i+1} = g_i + p_i \cdot c_i$

g_i=1 ko sta oba x_i **in** y_i enaka 1, ne glede na vhodni prenos c_i

- V tem primeru bo stopnja *i* zanesljivo tvorila izhodni prenos, zato funkciji *g* pravimo "funkcija tvorjenja prenosa" (***generate*** function)

p_i=1 ko sta x_i **ali** y_i enaka 1. Izhodni prenos se zgodi, če je vhodni prenos c_i=1

- Če povemo drugače, se bo vhodni prenos (ki je 1) širil preko stopnje *i*; zato funkciji *p* pravimo "funkcija širjenja prenosa" (***propagate*** function)

Carry Look Ahead (CLA) seštevalnik

- Funkcijo izhodnega prenosa na i-ti stopnji
  $c_{i+1}=x_i \cdot y_i+x_i \cdot c_i+y_i \cdot c_i=x_i \cdot y_i+(x_i+y_i) \cdot c_i$
- $g_i=x_i y_i$ in $p_i=x_i+y_i$,
- Izpišemo PDNO p_i:

  $p_i=x_i+y_i=x_i \cdot y_i + x'_i \cdot y_i + x_i \cdot y'_i$
- Vstavimo v izraz za prenos: $c_{i+1}=x_i \cdot y_i+(x_i+y_i) \cdot c_i$

  $c_{i+1}= x_i \cdot y_i + x_i \cdot y_i \cdot c_i + (x'_i \cdot y_i + x_i \cdot y'_i) \cdot c_i$

  Uporabimo lastnost Boole-ove algebre (x+x·y=x):

  in dobimo končni izraz za funkcijo p_i :

  $g_i=x_i \cdot y_i$ in $p_i=x_i \oplus y_i$,

Carry Look Ahead (CLA) seštevalnik

- Zapišimo izraz izhodnega prenosa za n-bitni seštevalnik

  Če je, $c_n = g_{n-1} + p_{n-1} \cdot c_{n-1}$

  in, $c_{n-1} = g_{n-2} + p_{n-2} \cdot c_{n-2}$

  potem, $c_n = g_{n-1} + p_{n-1} \cdot (g_{n-2} + p_{n-2} \cdot c_{n-2})$

  $c_n = g_{n-1} + p_{n-1} \cdot g_{n-2} + p_{n-1} \cdot p_{n-2} \cdot c_{n-2}$

- Če bi to vstavljanje ponovili na preostalih stopnjah, bi dobili

$$c_n = g_{n-1} + p_{n-1} \cdot g_{n-2} + p_{n-1} \cdot p_{n-2} \cdot g_{n-3} + \ldots + p_{n-1} \cdot p_{n-2} \ \ldots$$
$$\ldots + p_1 \cdot g_0 + p_{n-1} \cdot p_{n-2} \ldots p_0 c_0$$

Carry Look Ahead (CLA) seštevalnik

Prenos, ki se tvori na stopnji *n*-2, in se širi prek preostalih stopenj

Prenos, ki se tvori na stopnji n-2, in se širi prek preostalih stopenj

$$c_n = g_{n-1} + p_{n-1}g_{n-2} + p_{n-1}p_{n-2}g_{n-3} + \ldots + p_{n-1}p_{n-2}\ldots p_1 g_0 + p_{n-1}p_{n-2}\ldots p_0 c_0$$

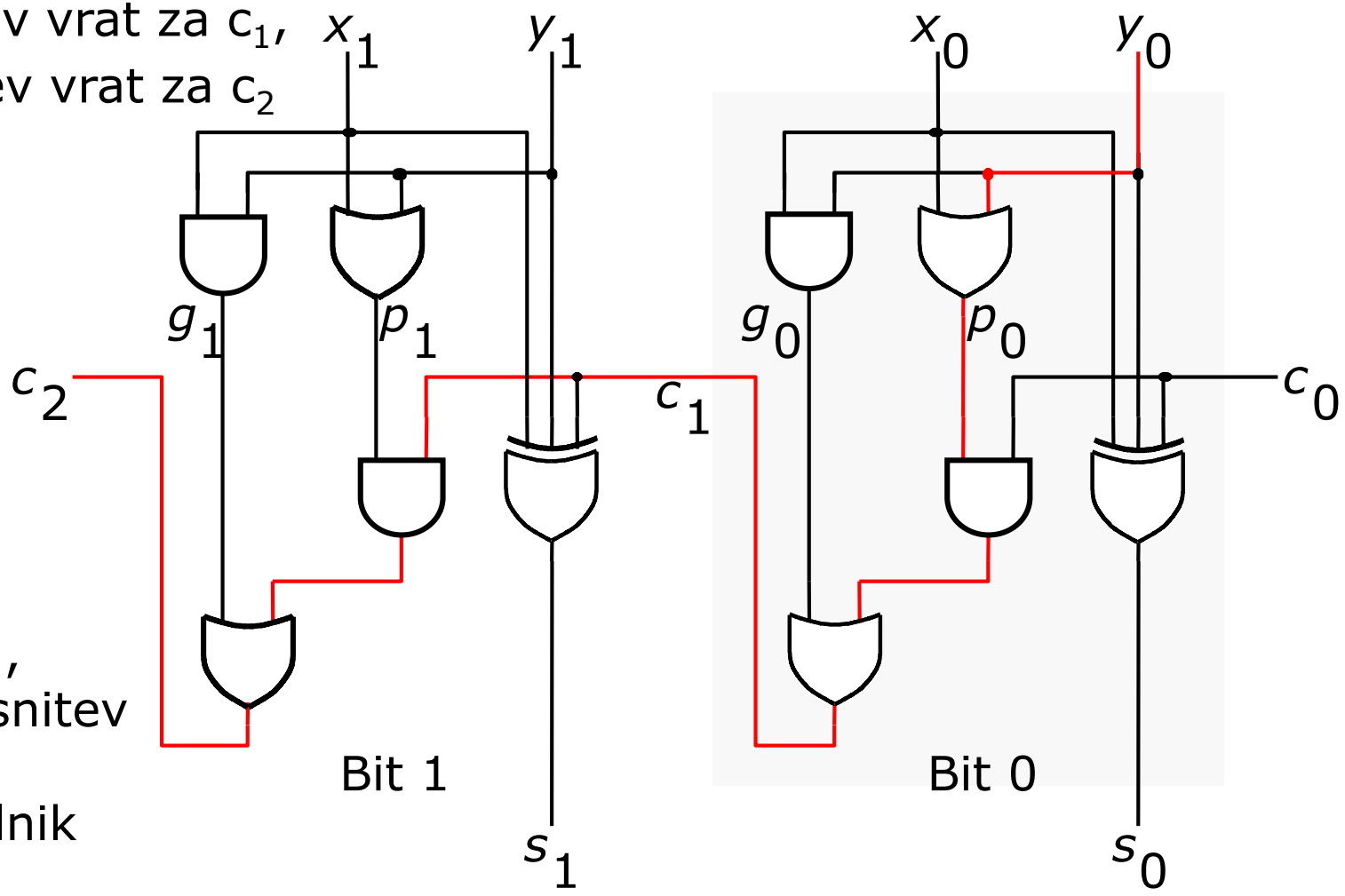
Prenos, ki se tvori na zadnji stopnji

Prenos, ki se tvori na stopnji n-3, in se širi prek preostalih stopenj

Vhodni prenos c_0 ki se širi preko vseh stopenj

Kritična pot RC seštevalnika

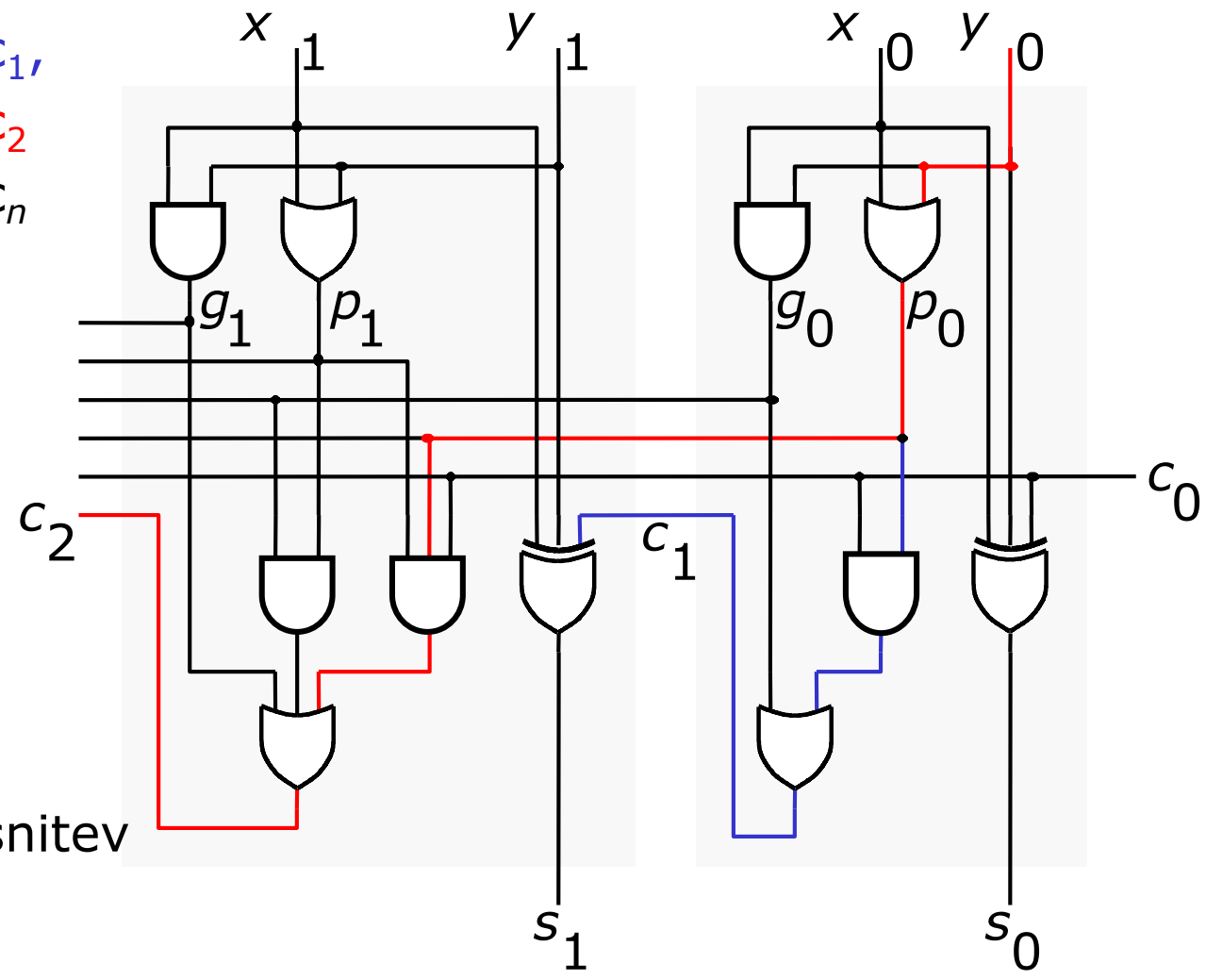
3 zakasnitev vrat za c_1,
5 zakasnitev vrat za c_2



V splošnem,
$2n+1$ zakasnitev
za n-bitni
RC seštevalnik

Kritična pot CLA seštevalnika

3 zakasnitev vrat za c_1,
3 zakasnitev vrat za c_2
3 zakasnitev vrat za c_n



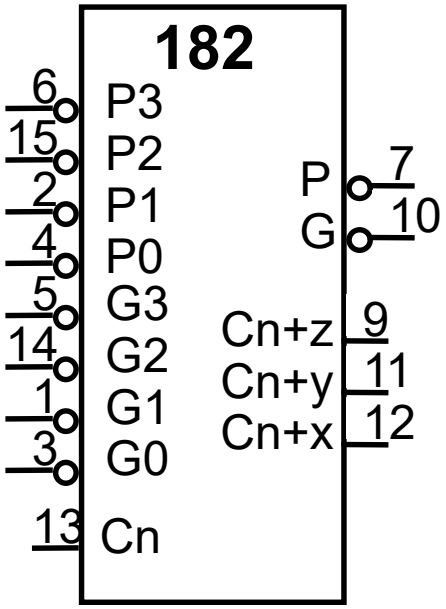
Skupna zakasnitev
n-bitnega
CLA seštevalnika je 4
zakasnitve vrat:
Vsi g_i in p_i, eno zakasnitev
Vsi c_i, dve zakasnitvi
+ zakasnitev za s_i

Carry Look Ahead Generator (CLAG)

- Realizira tvorbo izhodnih prenosov C_{n+x}, C_{n+y}, C_{n+z} na osnovi vhodnih funkcij g_0, p_0, g_1, p_1, g_2, p_2 *in* g_3, p_3 ter vhodnega prenosa C_n po enačbah:

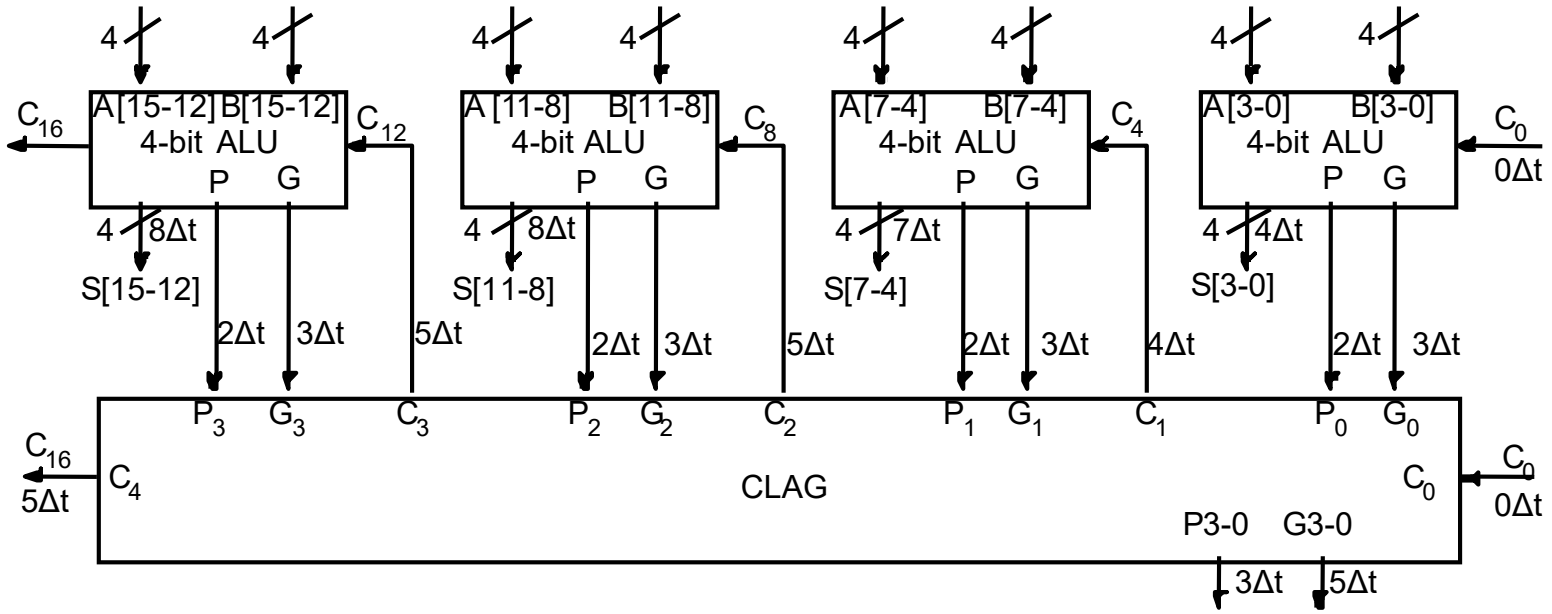
$$G = G_3 + G_2 + G_1 + G_0$$
$$P = P_3 \cdot (G_3 + P_2) \cdot (G_3 + G_2 + P_1) \cdot (G_3 + G_2 + G_1 + P_0)$$

$$C_{n+x} = P_0 \cdot (G_0 + C_n)$$
$$C_{n+y} = P_1 \cdot (G_1 + P_0 \cdot (G_0 + C_n))$$
$$C_{n+z} = P_2 \cdot (G_2 + P_1 \cdot (G_1 + P_0 \cdot (G_0 + C_n)))$$



- Primeren je za neposredno uporabo s 74181 (negirani izhodi in vhodi)

Kaskadna vezava CLA seštevalnikov



Razvoj digitalnih sistemov

Množenje in deljenje

4-bitno nepredznačeno množenje

				1	1	0	1	$= 13_{10}$
				1	0	1	1	$= 11_{10}$
				1	1	0	1	1
+			1	1	0	1		1
+		0	0	0	0			0
+	1	1	0	1				1
	1	0	0	1	1	1	1	$= 143_{10}$

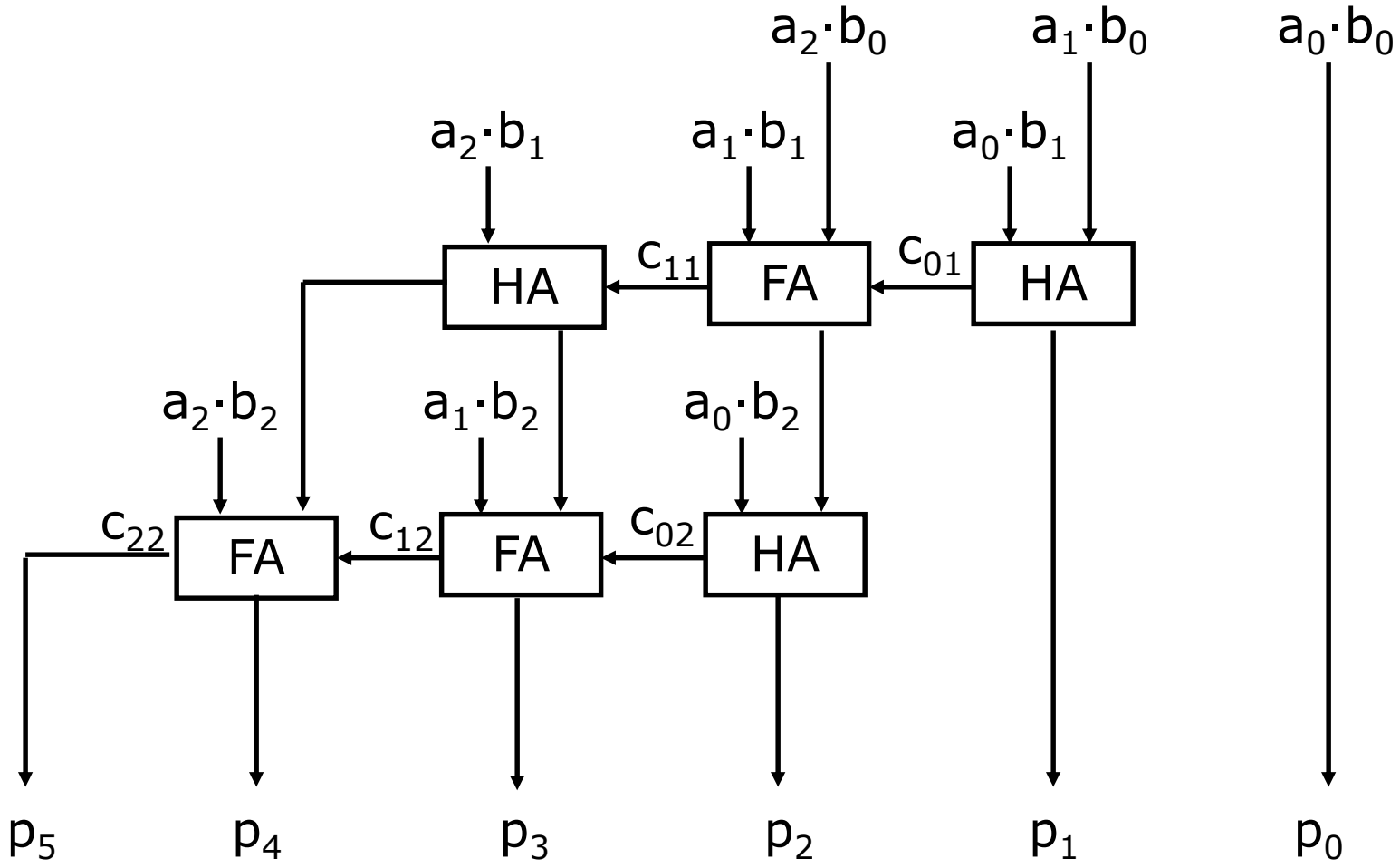
$7_{10}*6_{10}=42_{10}$

					1	1	1	
				*				
					1	1	0	
					0	0	0	
	+							
				1	1	1		
	+							
		^{1}	$^{1}1$	$^{1}1$	1			
42_{10}	=	1	0	1	0	1	0	

$7_{10}*4_{10}=28_{10}$

				1	1	1	
				*			
				1	0	0	
				0	0	0	0
+							
			0	0	0		0
+							
		1	1	1			1
28_{10}	=	1	1	1	0	0	

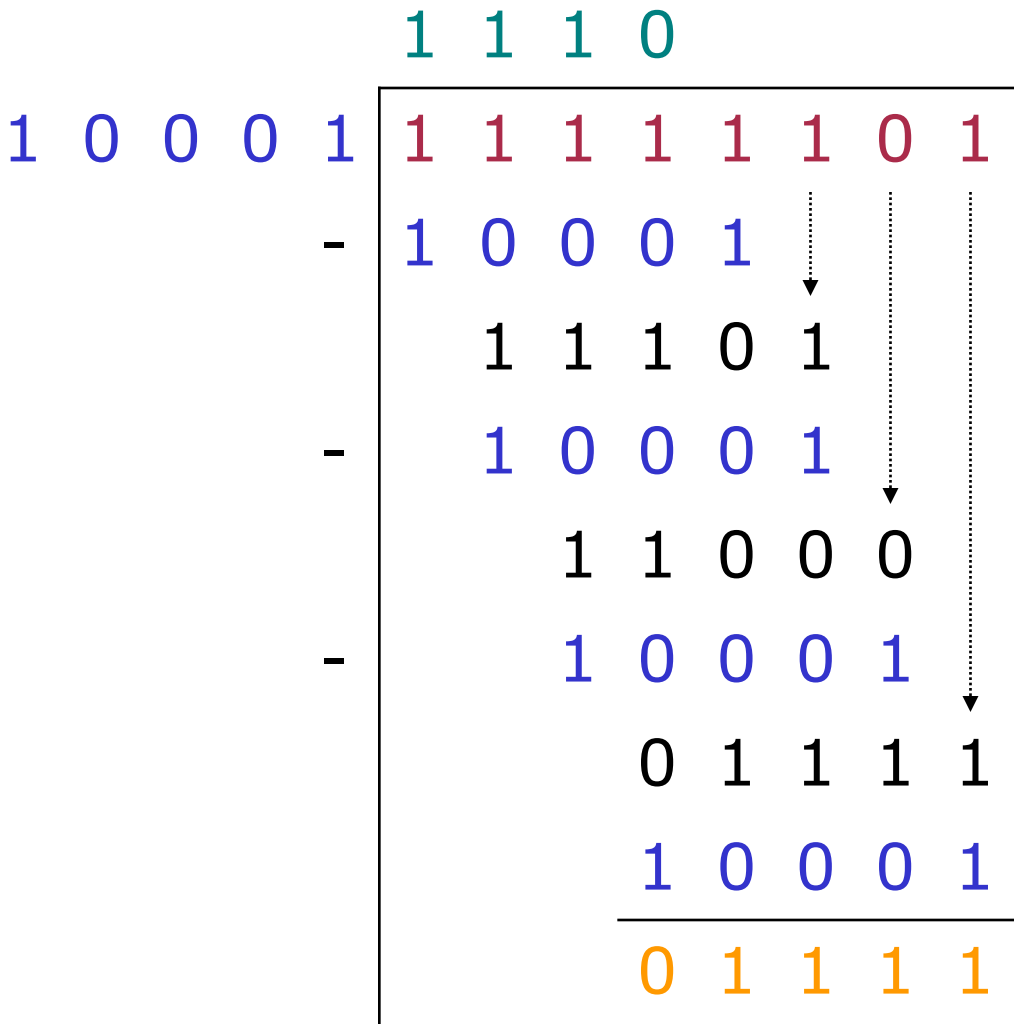
3-bitni množilnik



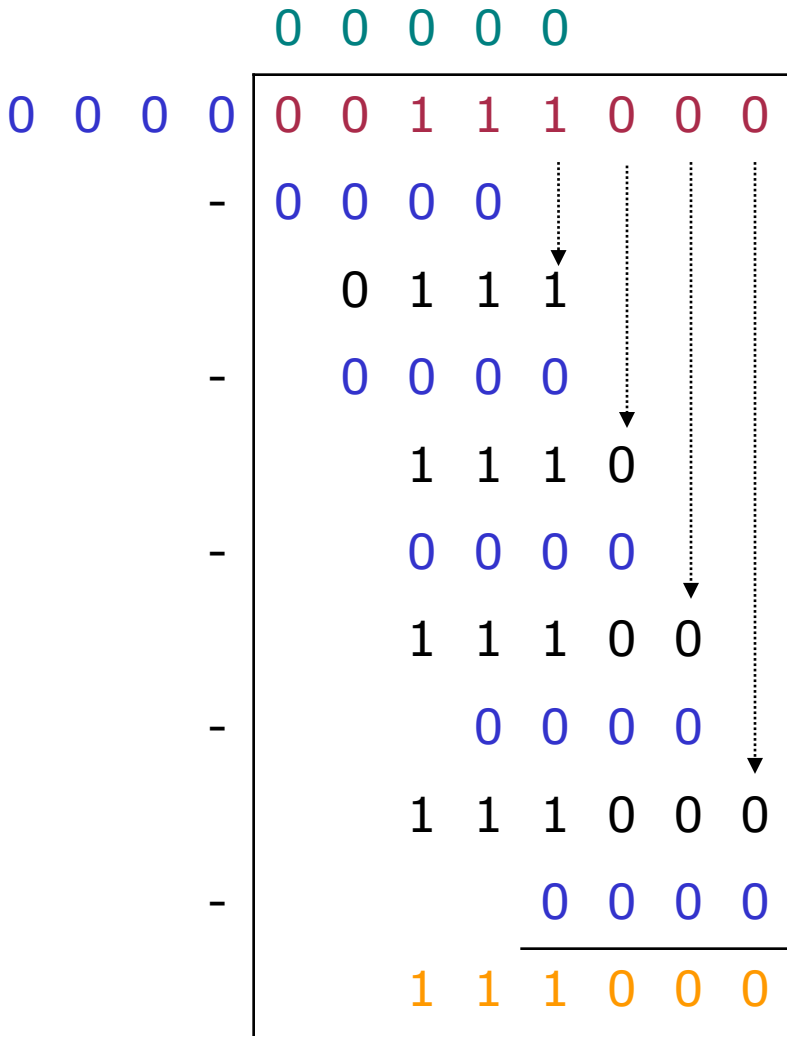
Deljenje: $53_{10}/7_{10}=7_{10}+4_{10}$

				0	1	1	1				
1	1	1		1	1	0	1	0	1		
				1	1	1					
				1	1	0	1				
			-		1	1	1				
					1	1	0	0			
			-			1	1	1			
						1	0	1	1		
			-				1	1	1		
							1	0	0		

Deljenje: $253_{10}/17_{10}=14_{10}+15_{10}$



Deljenje: $56_{10}/0_{10}=\infty$

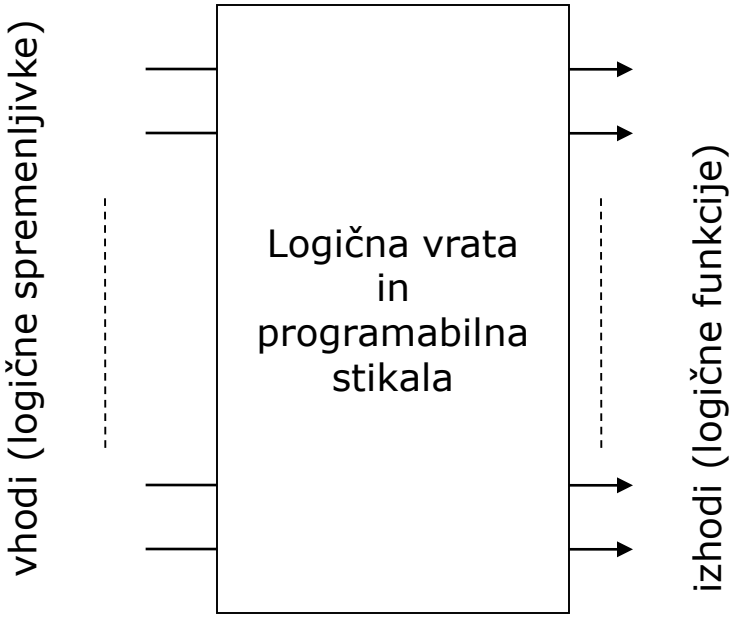


Razvoj digitalnih sistemov

Tehnologija realizacije logičnih funkcij:

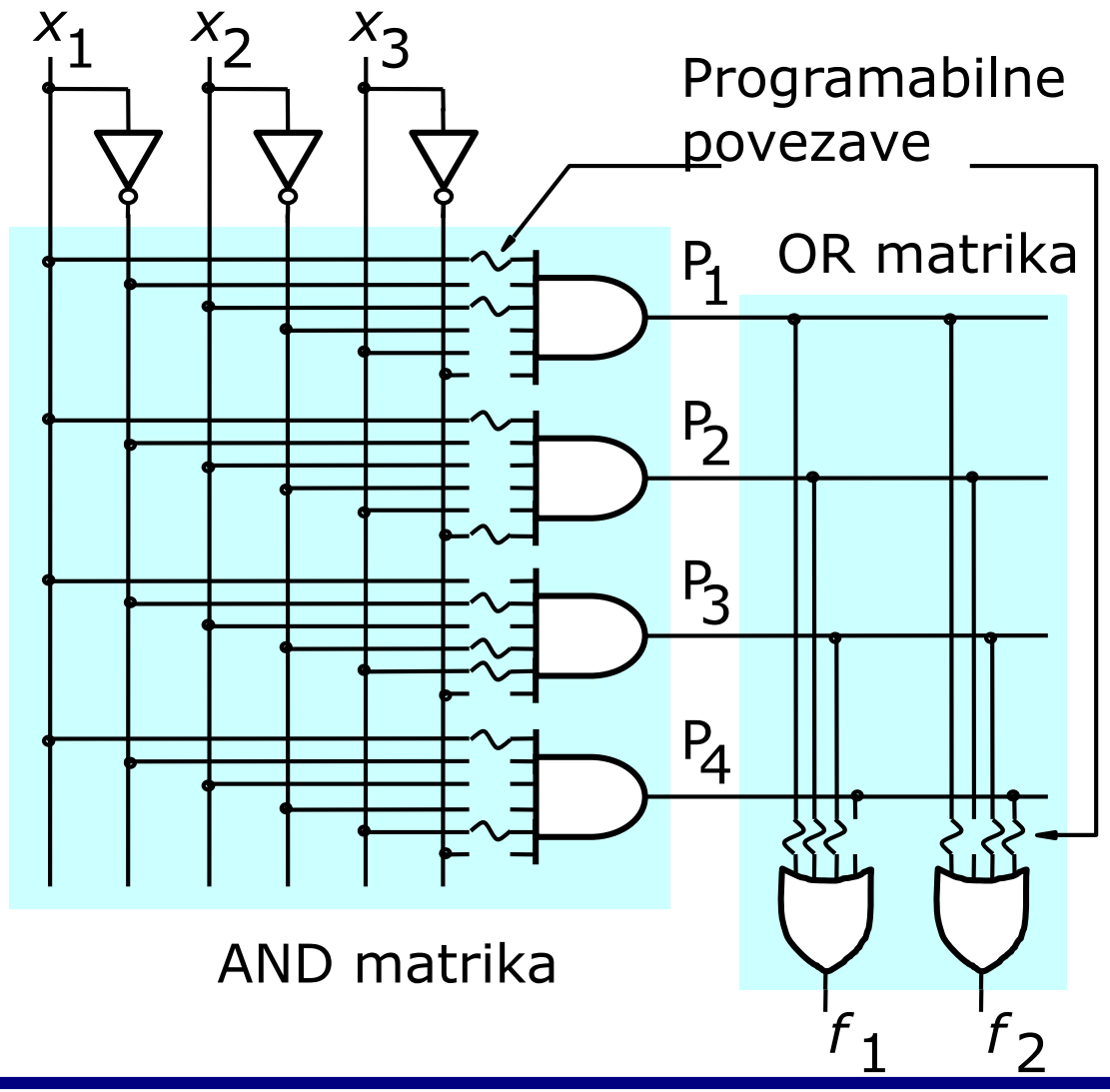
Standardna in programabilna logična vezja (PLD)

Programabilna logična vezja

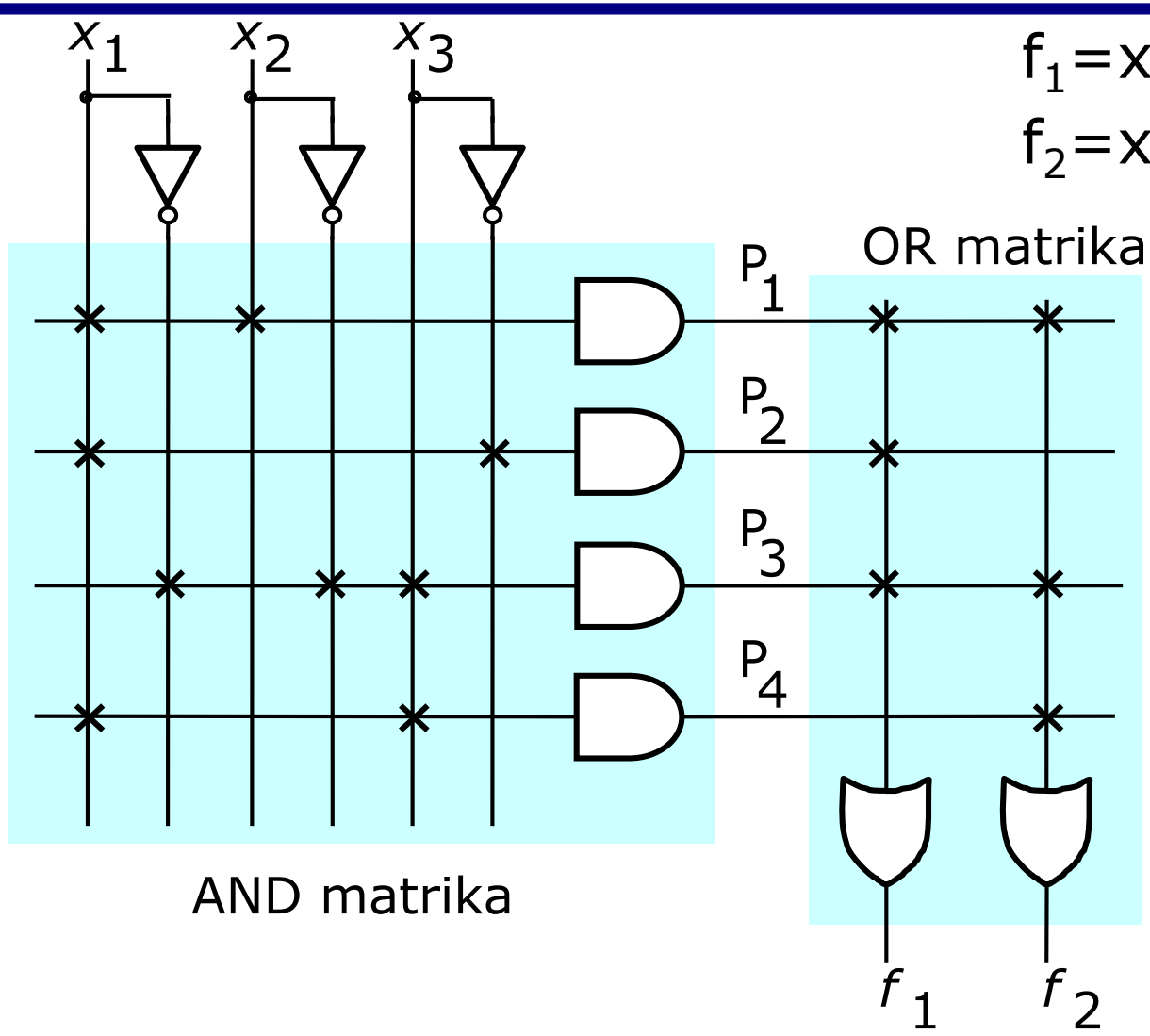


VEZJE/matrika	AND	OR
PLA	PROGRAMIRAMO	PROGRAMIRAMO
PAL	PROGRAMIRAMO	FIKSNO
ROM	FIKSNO	PROGRAMIRAMO

Diagram PLA na nivoju log. vrat



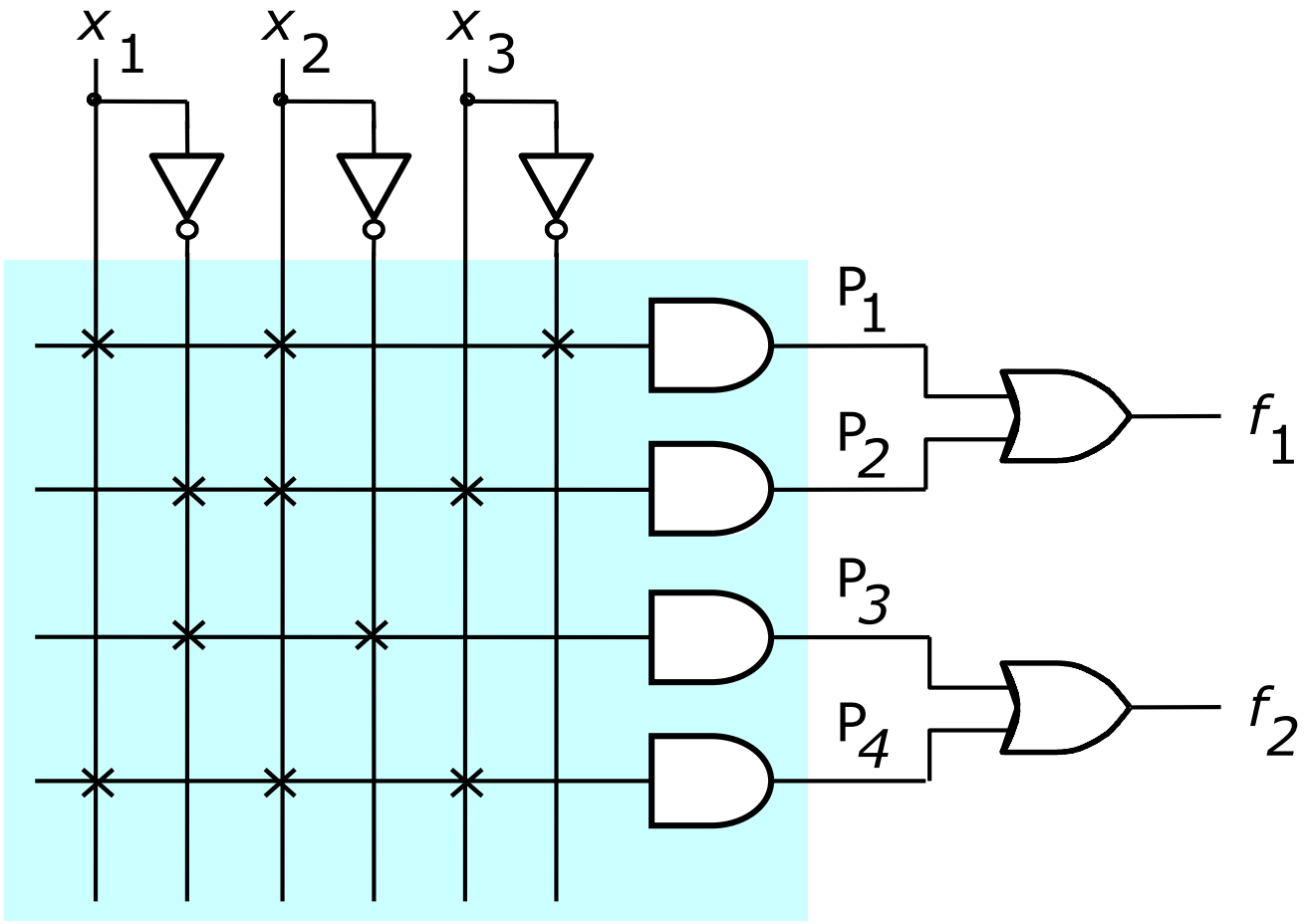
Poenostavljena shema PLA



$f_1 = x_1x_2 + x_1x_3' + x_1'x_2'x_3$

$f_2 = x_1x_2 + x_1'x_2'x_3 + x_1x_3$

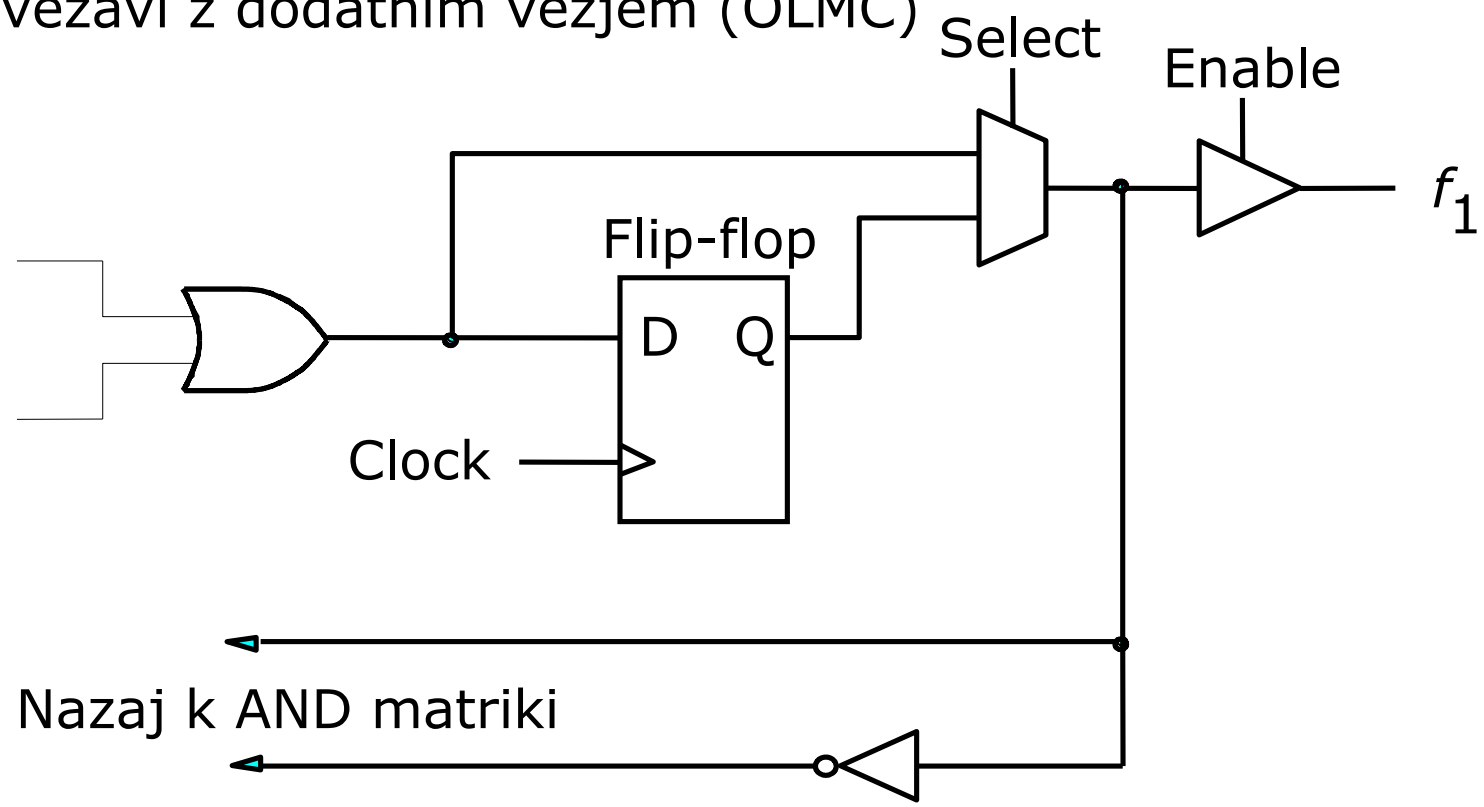
Programmable Array Logic (PAL)



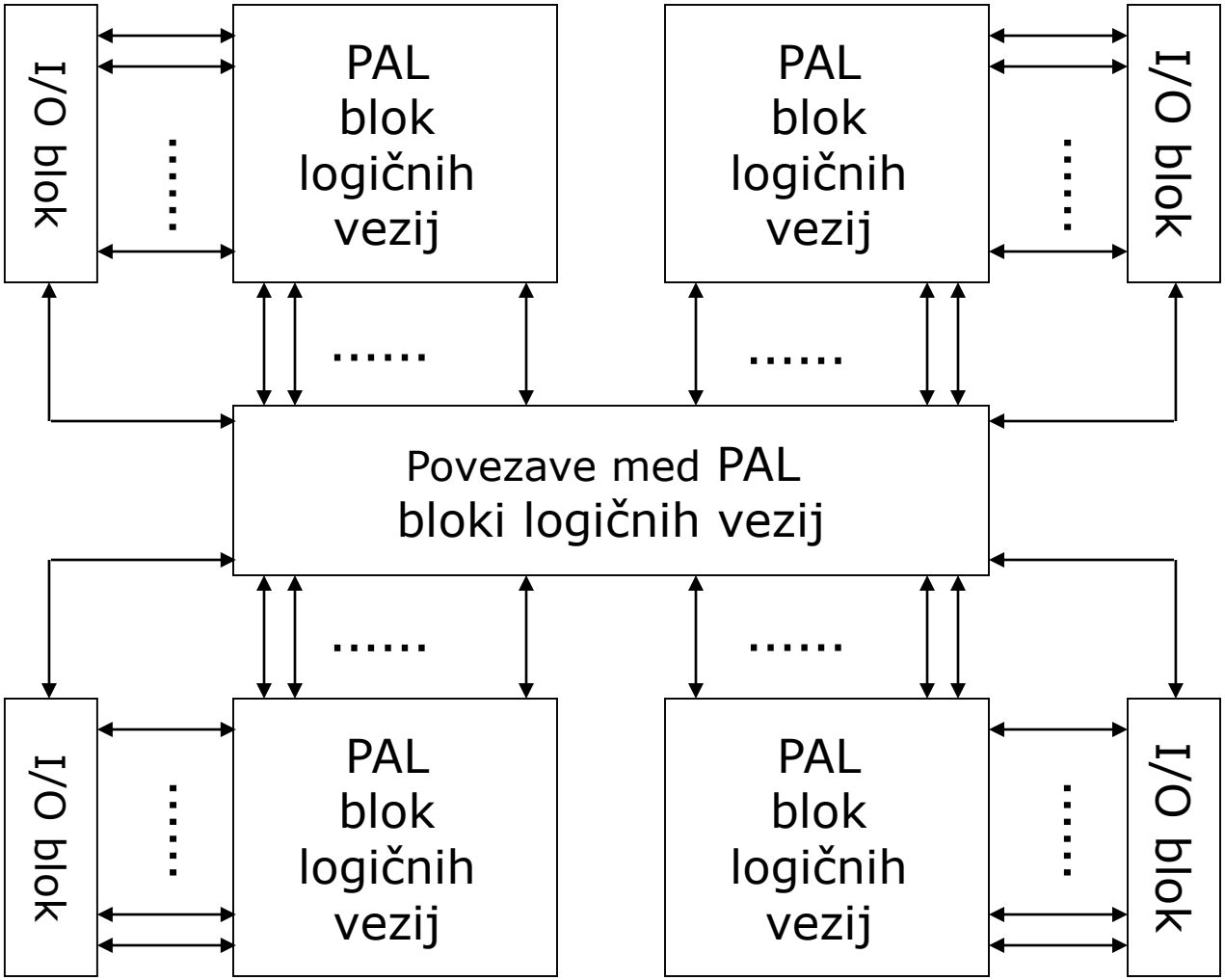
AND matrika

Dodatna vezja v PAL strukturi

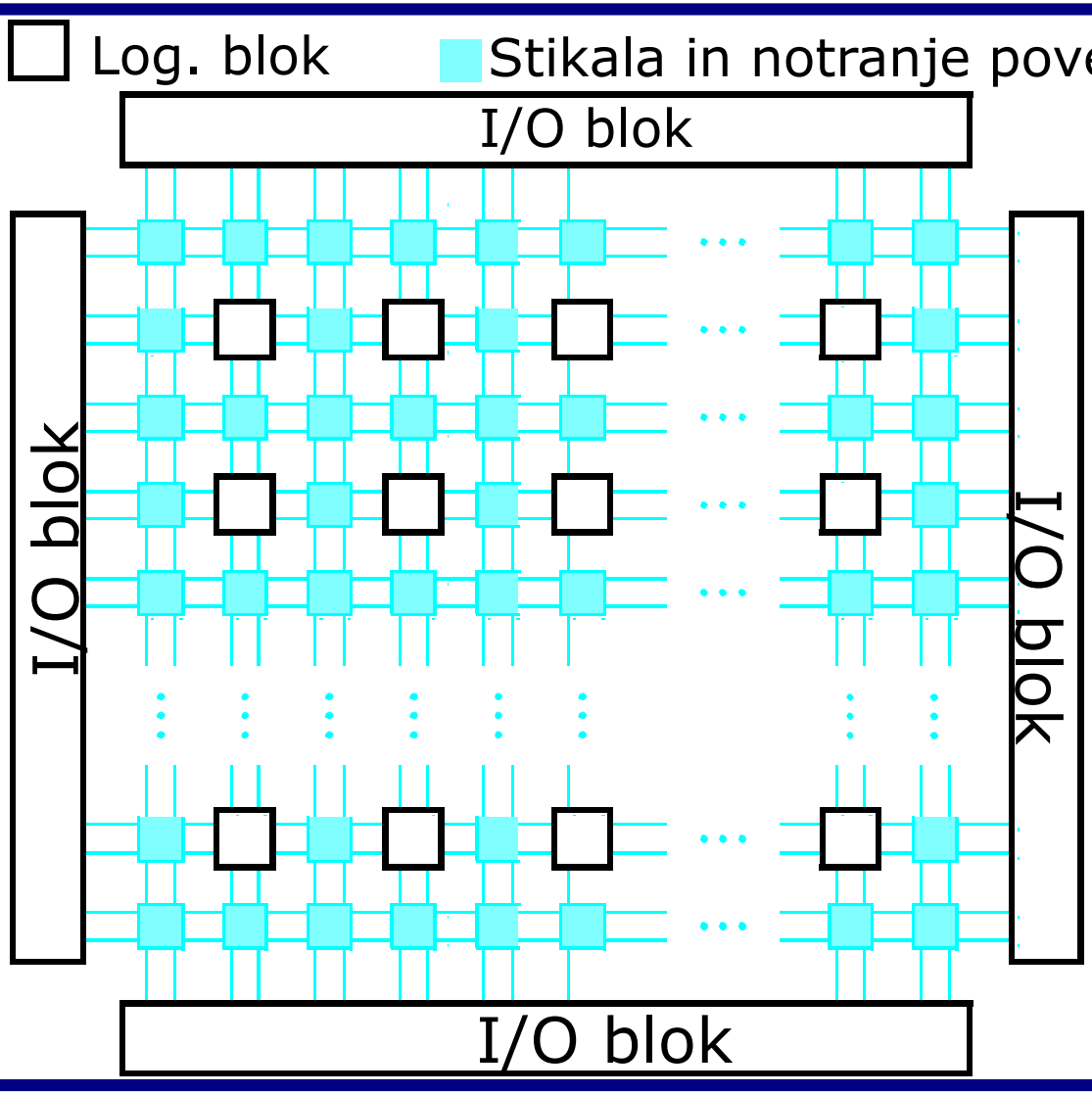
- PAL strukture vsebujejo dodatno spominsko celico na izhodu vsakih OR vrat, za realizacijo sekvenčnih funkcij
- Termin **makrocelice** (macrocell*)* se nanaša na OR vrata v povezavi z dodatnim vezjem (OLMC)



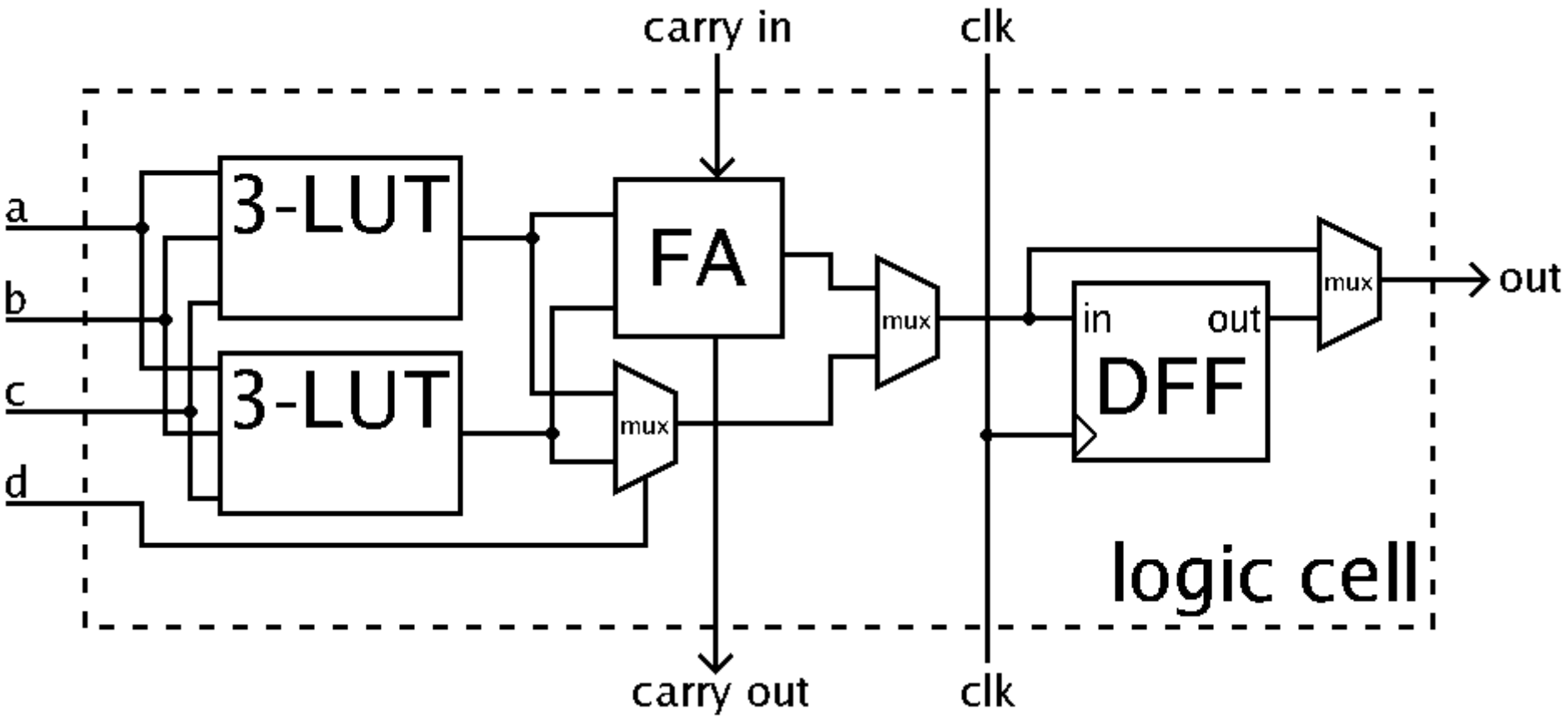
CPLD (Kompleksna PLD) vezja



FPGA vezja


Log. blok
Stikala in notranje povezave
I/O blok
I/O blok
I/O blok
I/O blok

Poenostavljen CLB v FPGA



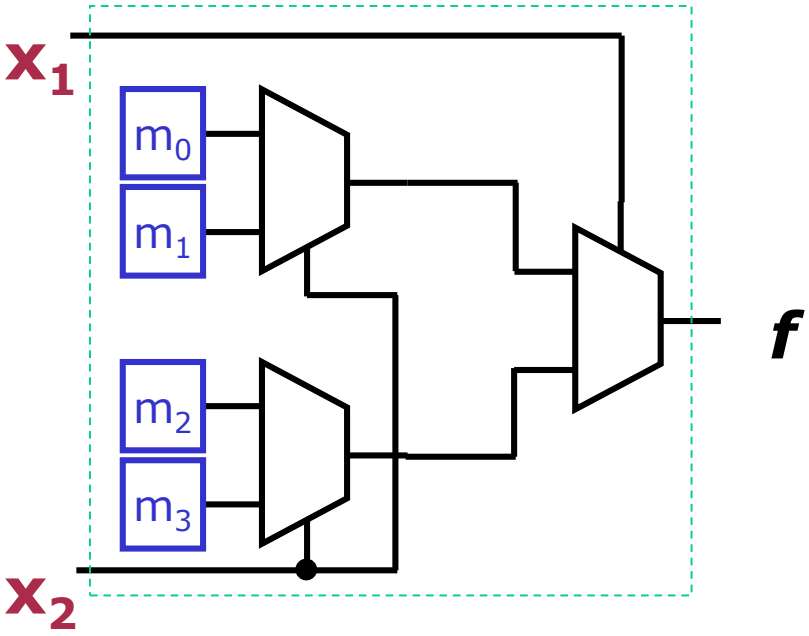
Vir: Wikipedija.

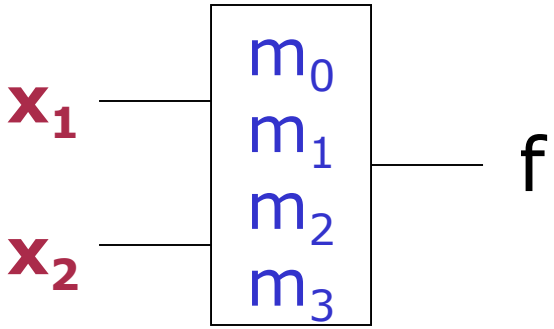
Razvoj digitalnih sistemov

Tehnologija realizacije funkcij:

Vpogledne tabele oz.
LUT (Look-Up Tables)

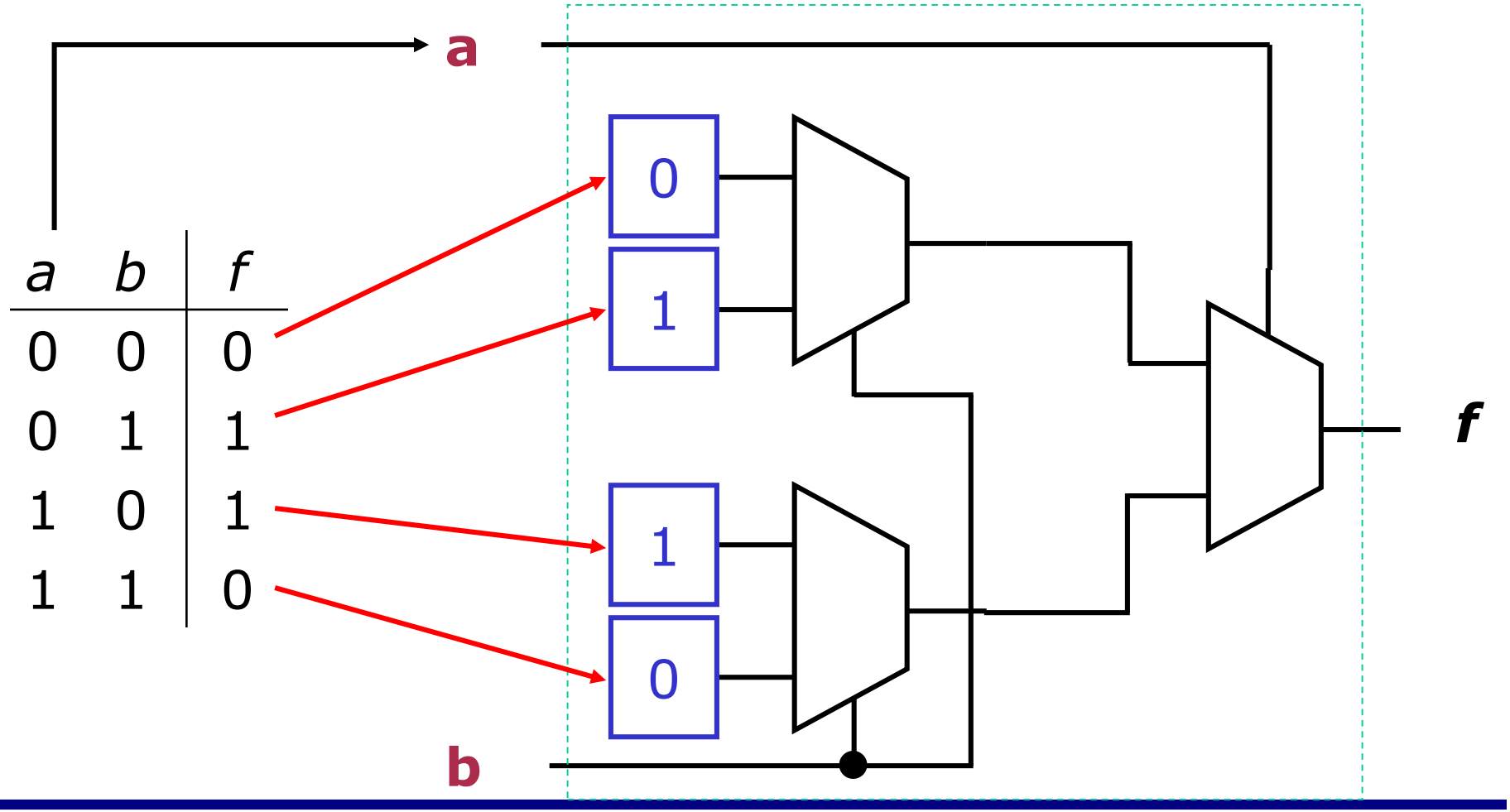
Simbol in zgradba LUT2





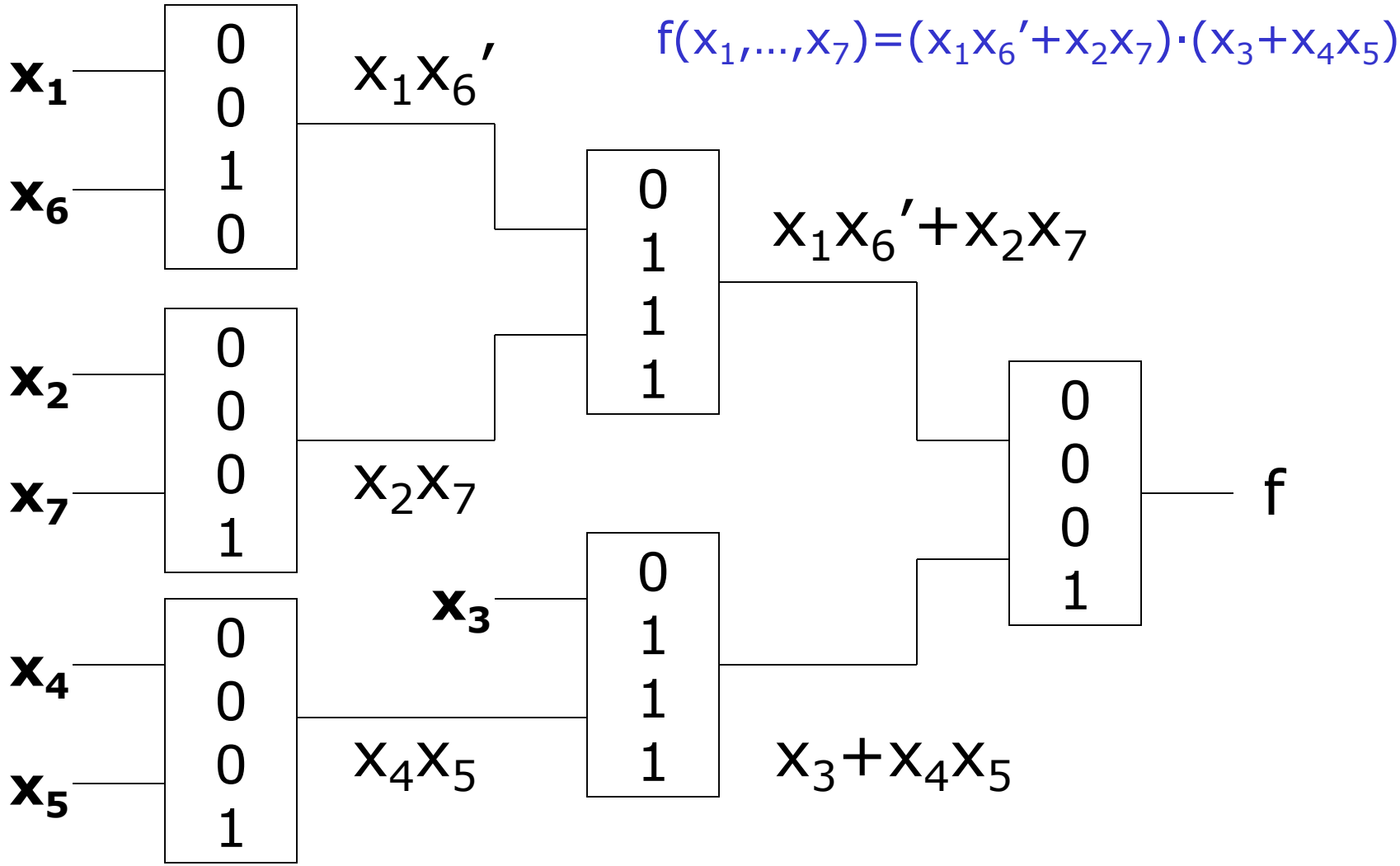
Programirana LUT (f=a'b+ab')

Najbolj pomembna spremenljivka (a) krmili zadnji izbiralnik

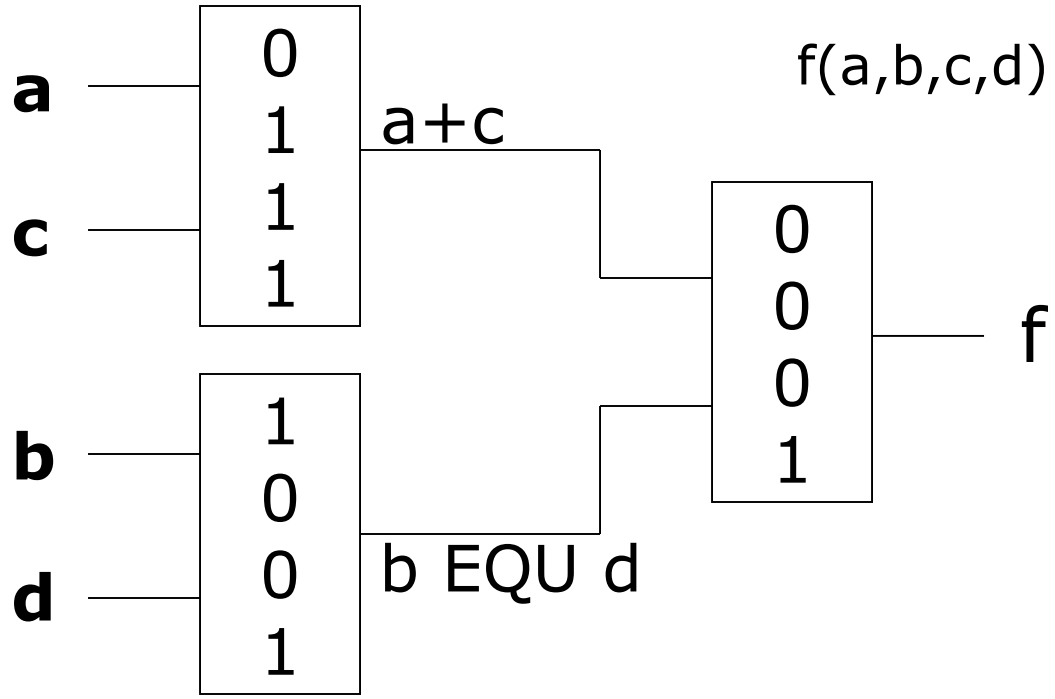


a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

Faktorski zapis funkcije v FPGA z LUT2

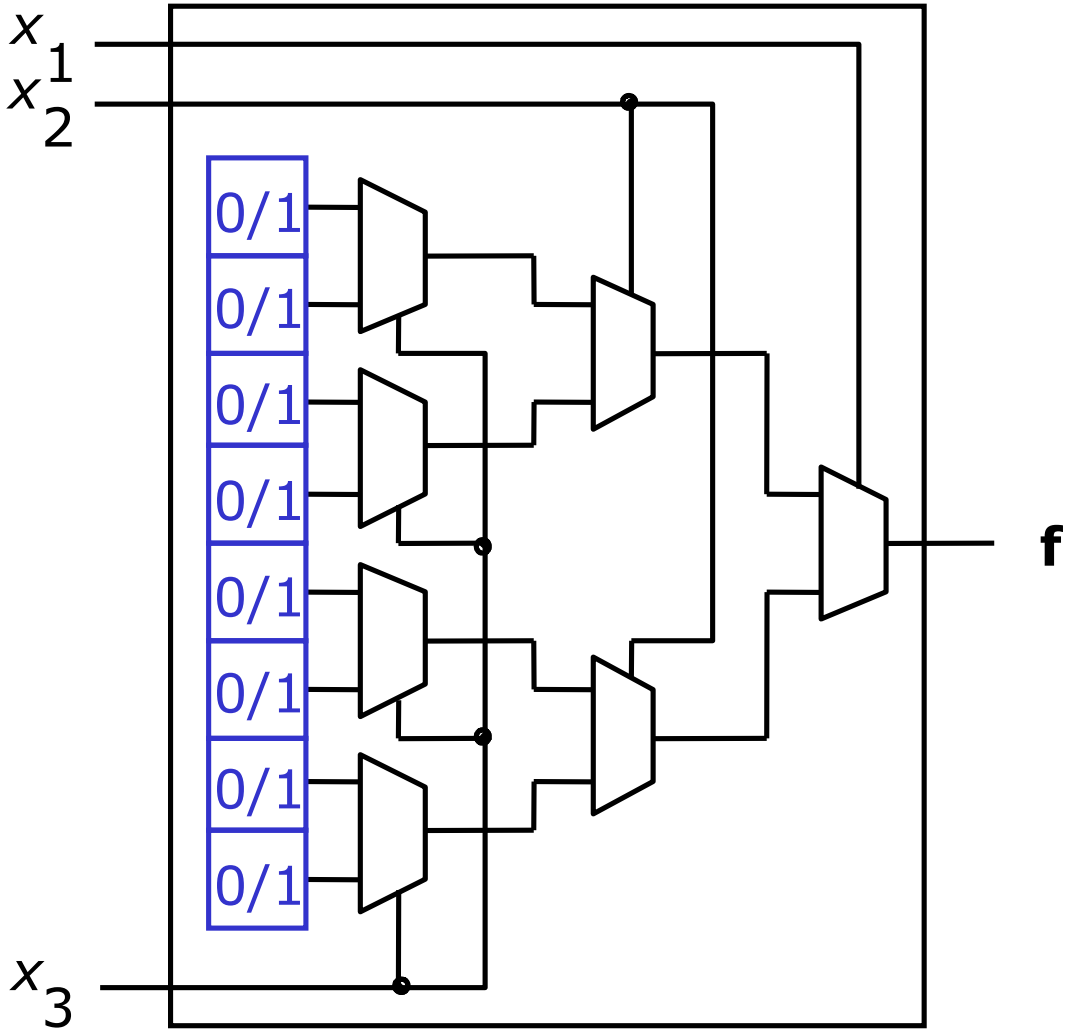


Realizacija zgleda večnivojske funkcije z FPGA, ki ima samo LUT2

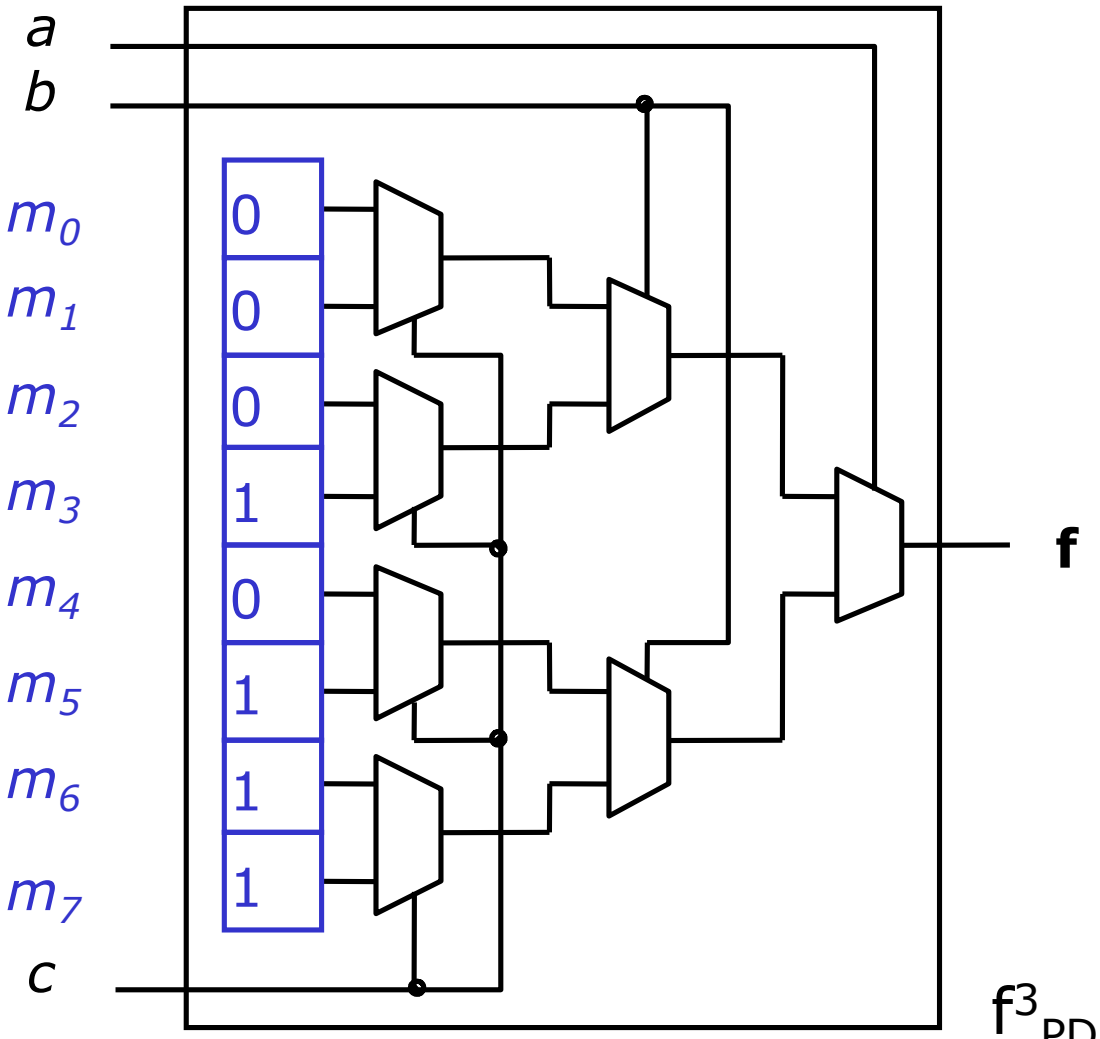


f(a,b,c,d)= (a+c)·(b equ d)
COST = (3, 6)

Trovhodna struktura LUT



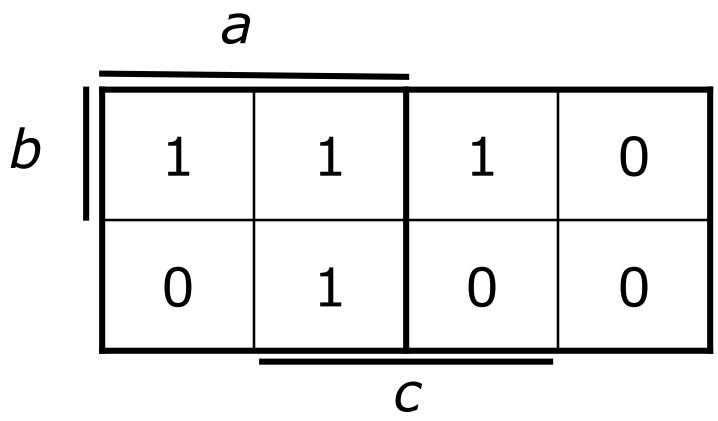
Zgled 3-vhodne strukture LUT



$f^3_{PDNO}=V(3,5\text{-}7)$

Zgled trovhodne strukture LUT

- Prikažite diagram programirane strukture LUT ki realizira funkcijo
- f(a, b, c)=a’·b·c+a·b·c’+a·c

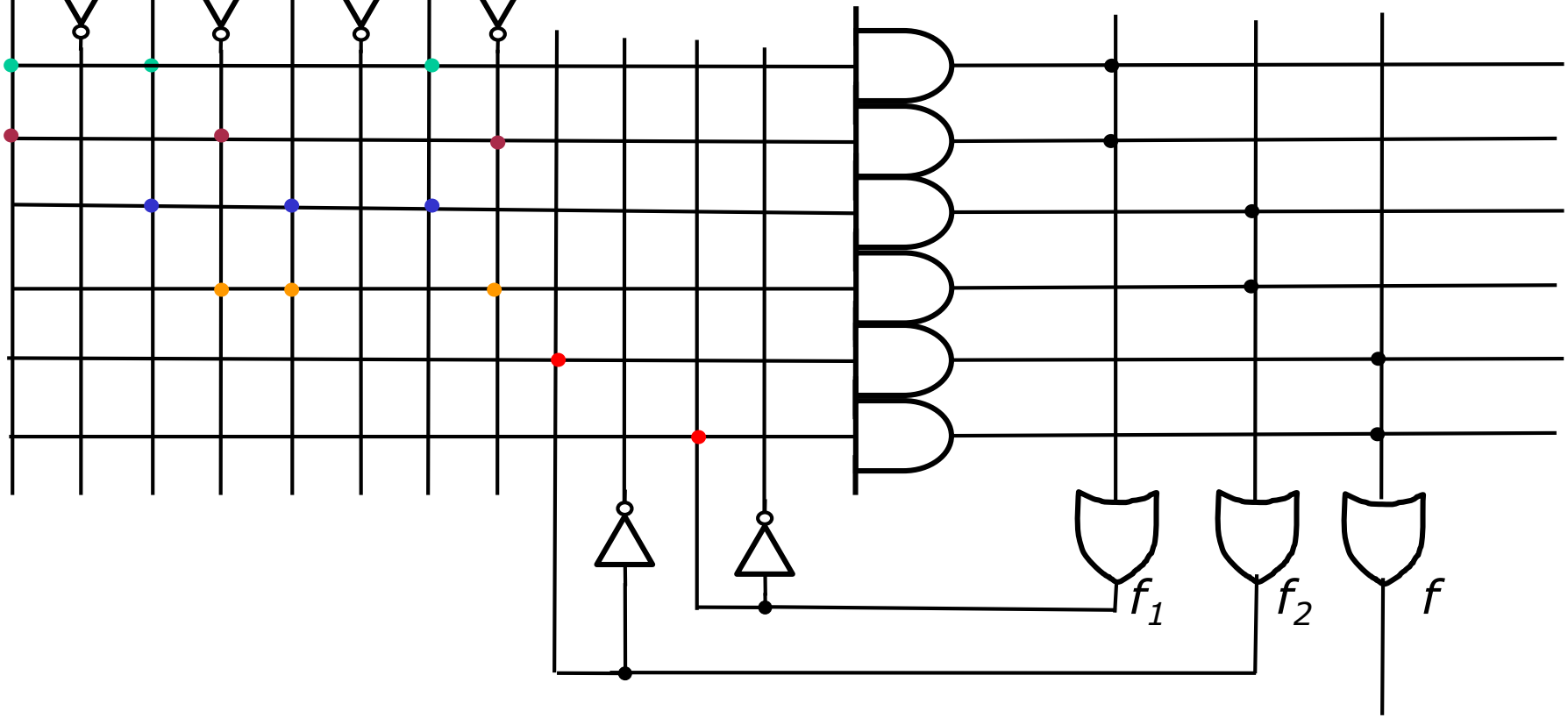


$f^3_{PDNO}=V(3,5\text{-}7)$

Realizacija zgleda večnivojske funkcije s PLD vezjem

$f(a,b,c,d)_{MDNO} = (abd + ab'd') + (bcd + b'cd')$

$= \quad f_1 \quad + \quad f_2$



Načrtovanje funkcij s PAL, PLA, ROM

$g_1 = x_1 + x_2'x_3'$

$g_2 = x_1'x_3 + x_1x_2$

$g_3 = x_2'x_3' + x_1x_2$

$g_4 = x_2'x_3 + x_1$

OR	$x_1x_2x_3$	$g_1g_2g_3g_4$
x_1x_2	1 1 -	0 1 1 0
$x_2'x_3$	- 0 1	0 0 0 1
$x_1'x_3$	1 - 0	0 1 0 0
$x_2'x_3'$	- 0 0	1 0 1 0
x_1	1 - -	1 0 0 1

Načrtovanje funkcij s PLA

$g_1 = x_1 + x_2'x_3'$

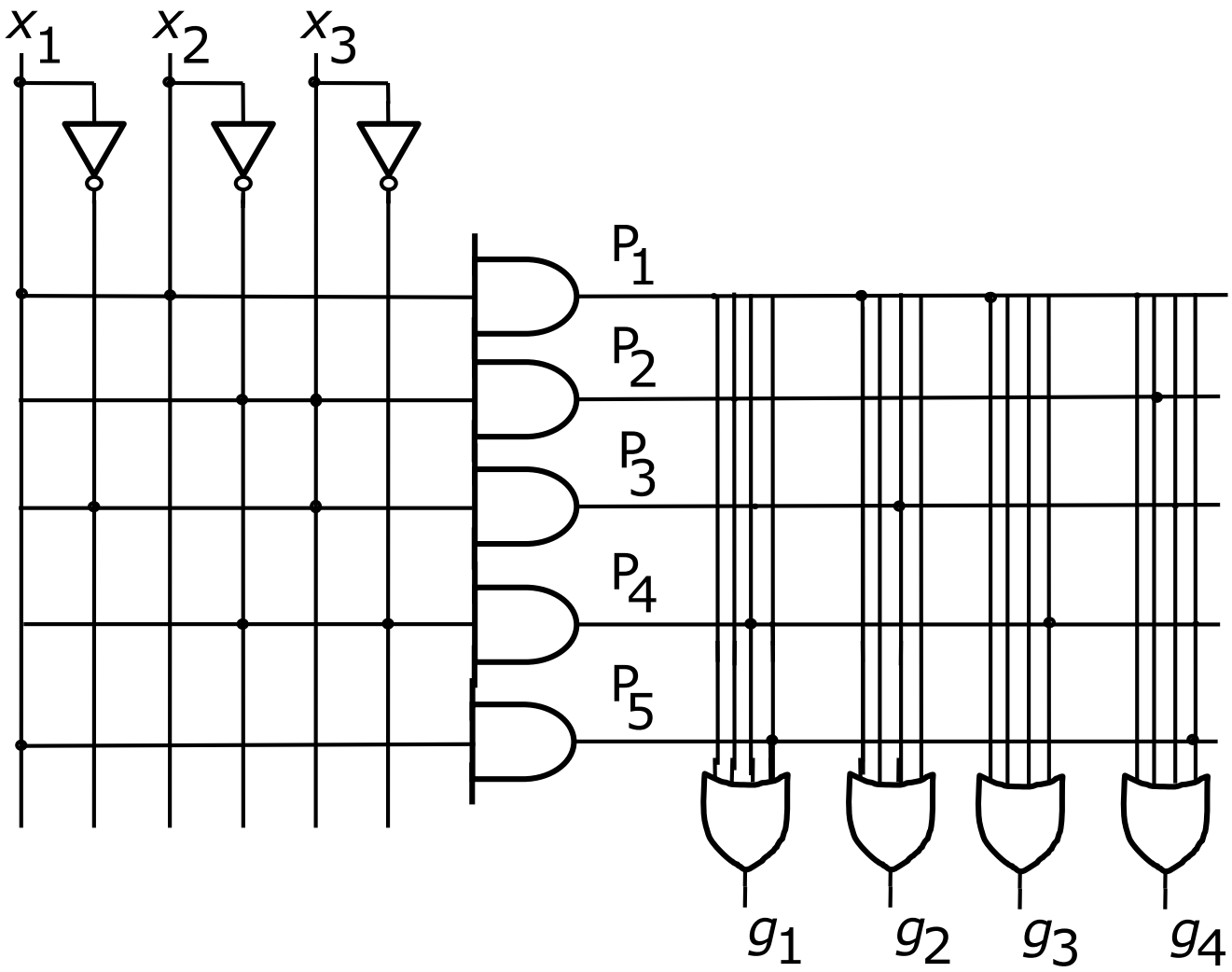
$g_2 = x_1'x_3 + x_1x_2$

$g_3 = x_2'x_3' + x_1x_2$

$g_4 = x_2'x_3 + x_1$

OR termi

$P_1 = x_1x_2$
$P_2 = x_2'x_3$
$P_3 = x_1'x_3$
$P_4 = x_2'x_3'$
$P_5 = x_1$



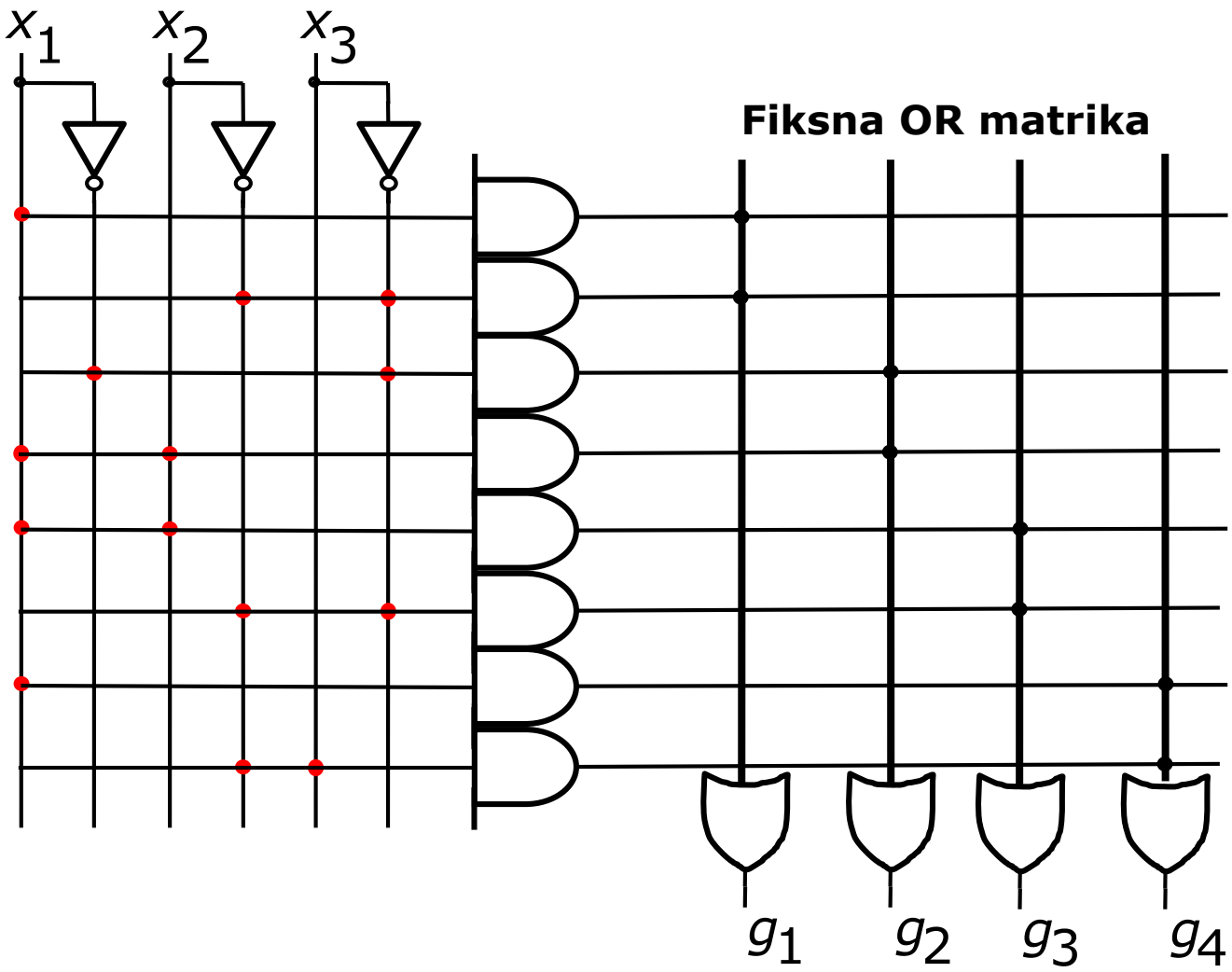
Načrtovanje funkcij s PAL

$g_1 = x_1 + x_2'x_3'$

$g_2 = x_1'x_3 + x_1x_2$

$g_3 = x_2'x_3' + x_1x_2$

$g_4 = x_2'x_3 + x_1$



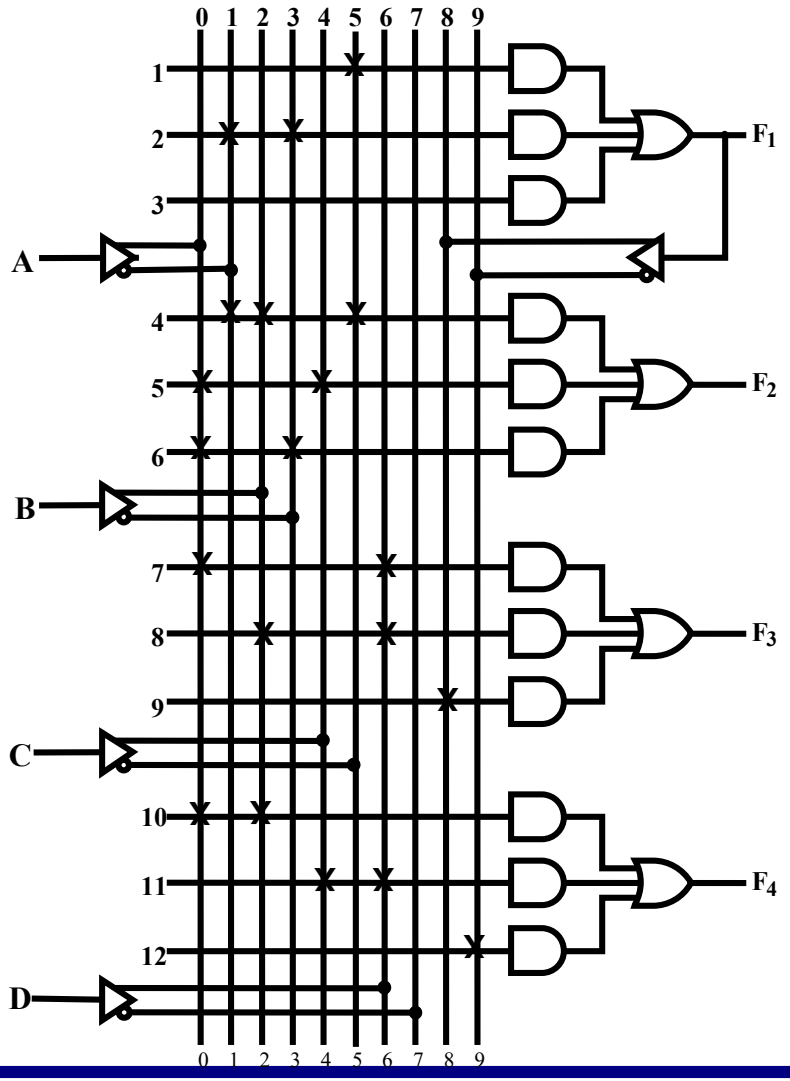
Načrtovanje funkcij s PAL

$F_1 = A' \cdot B' + C'$

$F_2 = A' \cdot B \cdot C' + A \cdot C + A \cdot B'$

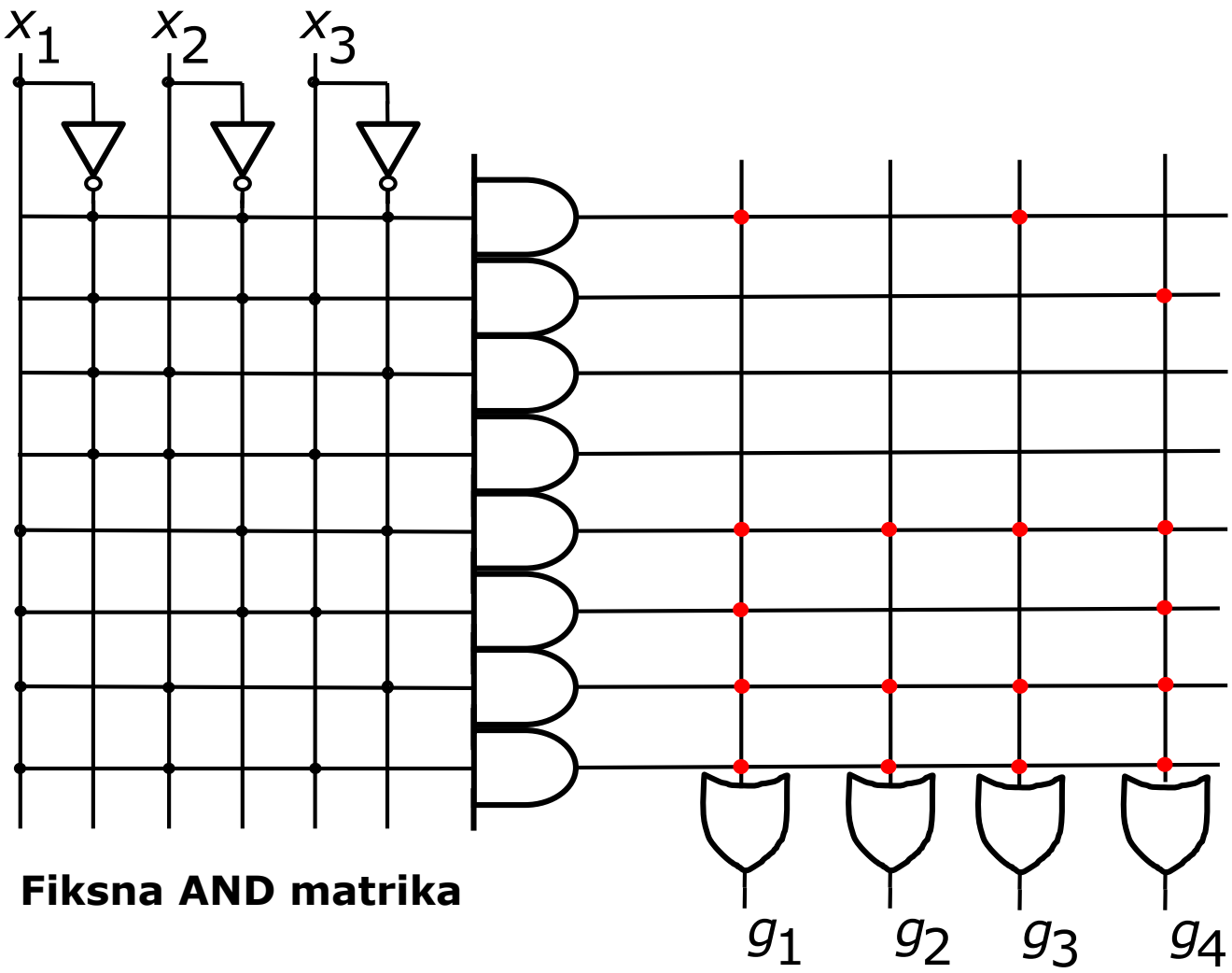
$F_3 = A \cdot D + B \cdot D + F_1$

$F_4 = A \cdot B + C \cdot D + F_1'$



Načrtovanje funkcij z ROM

$x_1x_2x_3$	$g_1g_2g_3g_4$
0 0 0	1 0 1 0
0 0 1	0 0 0 1
0 1 0	0 0 0 0
0 1 1	0 0 0 0
1 0 0	1 1 1 1
1 0 1	1 0 0 1
1 1 0	1 1 1 1
1 1 1	1 1 1 1



Fiksna AND matrika

Izbira PAL, PLA, ROM

Odvisno od strukture funkcij:

- Število OR izrazov za izvedbo logičnih funkcij
- Do katere stopnje se OR izrazi ponavljajo v različnih funkcijah
- PLA so primerni za manjše število različnih OR, ki se pojavljajo na številnih izhodnih funkcijah.
- PAL je tudi omejen z majhnim številom OR na izhodih.
- ROM je primeren za veliko število OR.

Razvoj digitalnih sistemov

Flip-flopi, registri in števci:
Zapah (ang. Latch)

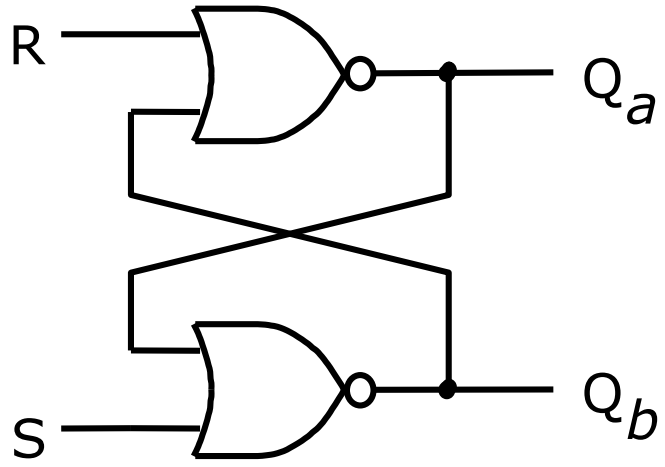
Spominski elementi

- Do sedaj ➔ kombinacijska vezja
  (izhod odvisen samo od vrednosti signalov na vhodih)
- Sekvenčna vezja
  (izhodi odvisni ne samo od trenutnih izhodov, tudi od preteklega obnašanja vezja)

Sekvenčna vezja

- Vsebina spominskih elementov predstavlja stanje vezja (ang. *state*)
- Spremembe vhodov pustijo vezje v istem stanju ali pa povzročijo njegovo spremembo
- Ko čas teče, vezje prehaja preko zaporedja stanj kot odziv na spremembo vhodov
- To so sekvenčna vezja (*sequential circuits*)

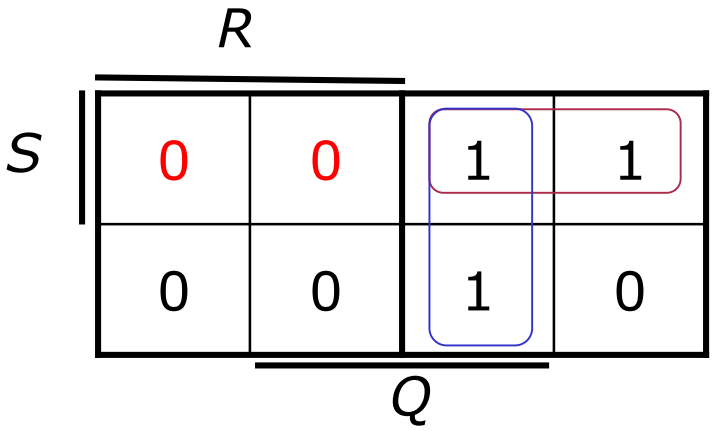
Osnovna izvedba RS zapaha



Karakteristična tabela:

S	R	Q_a	Q_b
0	0	0/1	1/0
0	1	0	1
1	0	1	0
1	1	0	0

S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	0



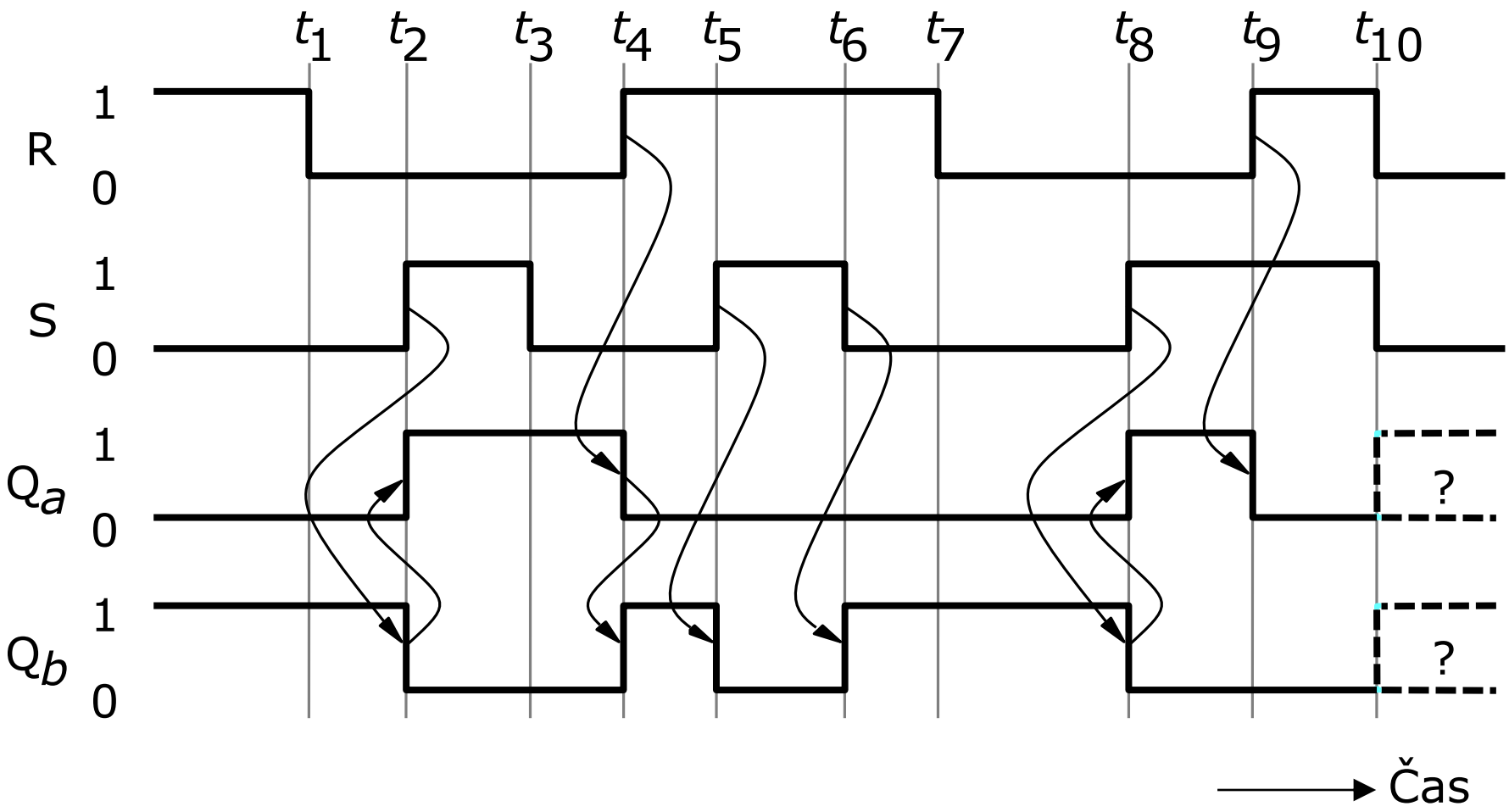
$$Q(t+1)=R'\cdot Q(t)+S\cdot R'$$

$$Q(t+1)=R'\cdot(Q(t)+S)$$

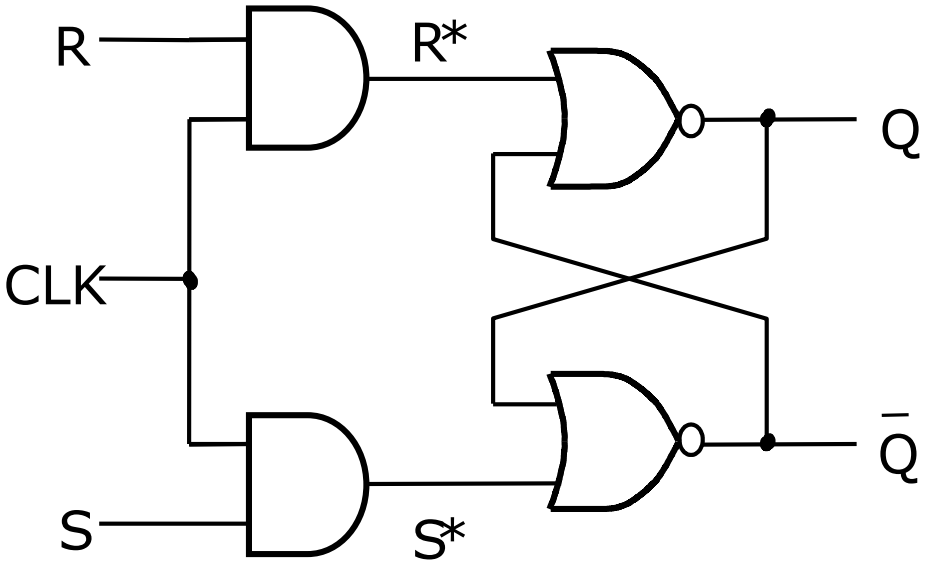
$$Q(t+1)=R'\cdot(Q'(t)\cdot S')'$$

$$Q(t+1)=(R+(Q(t)+S)')'$$

Časovni diagram RS zapaha

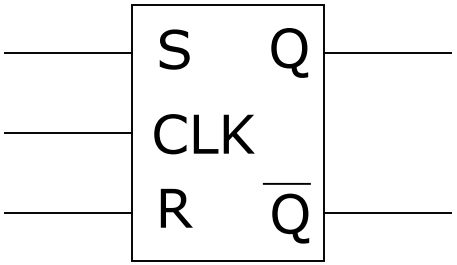


Vezje sinhronega RS zapaha



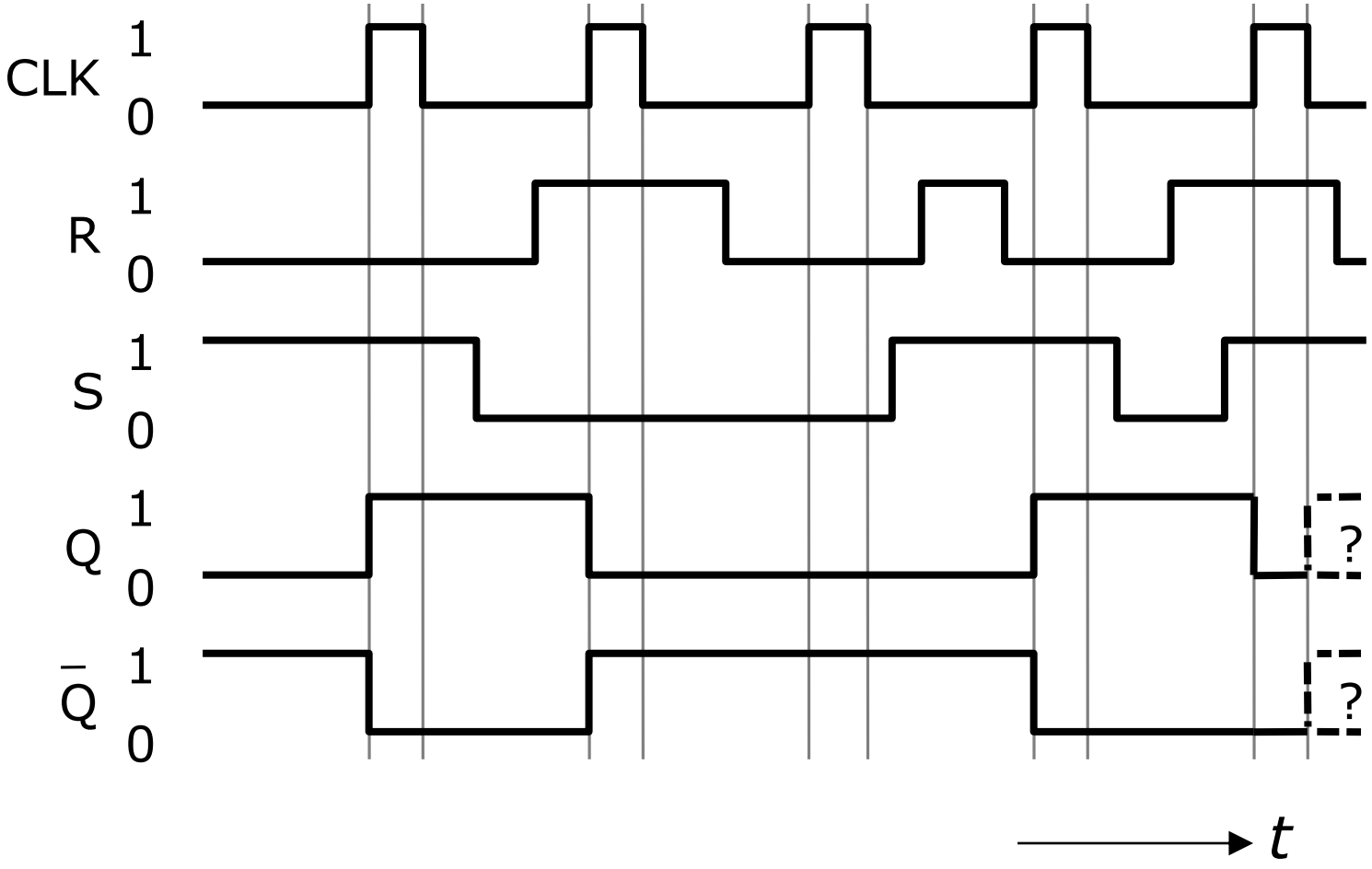
Karakteristična tabela

CLK	S	R	Q(t+1)
0	X	X	Q(t)
1	0	0	Q(t)
1	0	1	0
1	1	0	1
1	1	1	X

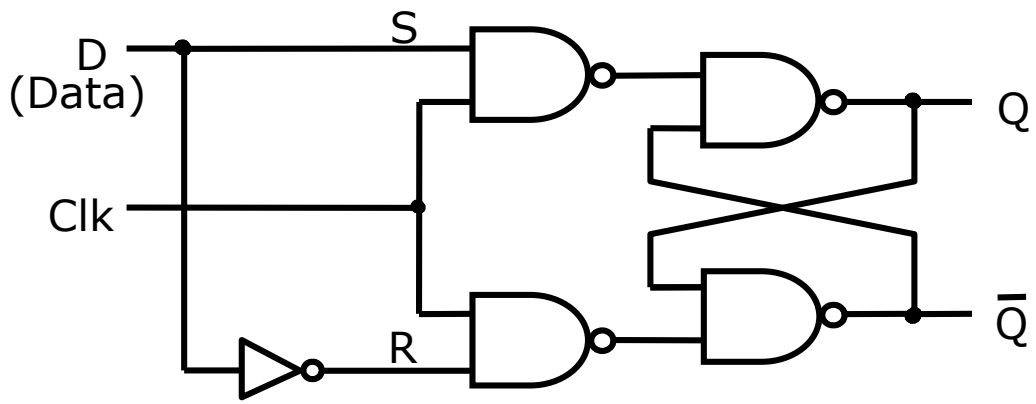


Q(t)=trenutno stanje
Q(t+1)=naslednje stanje

Časovni diagram sinhronega RS zapaha

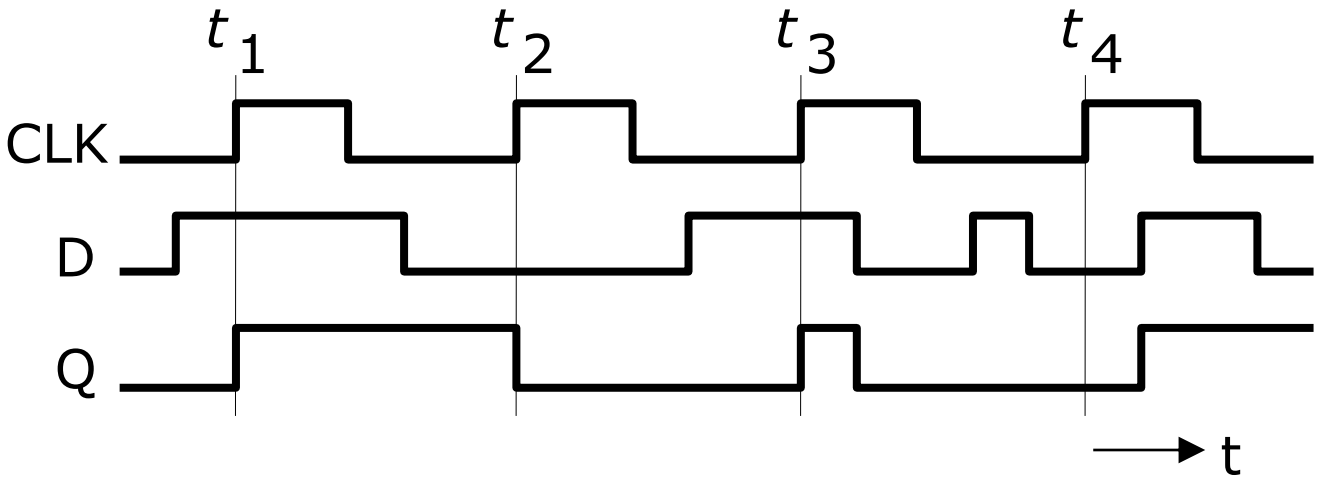


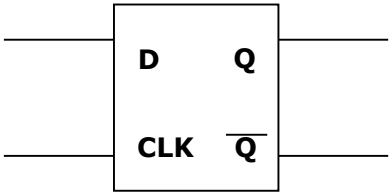
Sinhroni D zapah



Karakteristična tabela

CLK	D	Q(t+1)
0	X	Q(t)
1	0	0
1	1	1



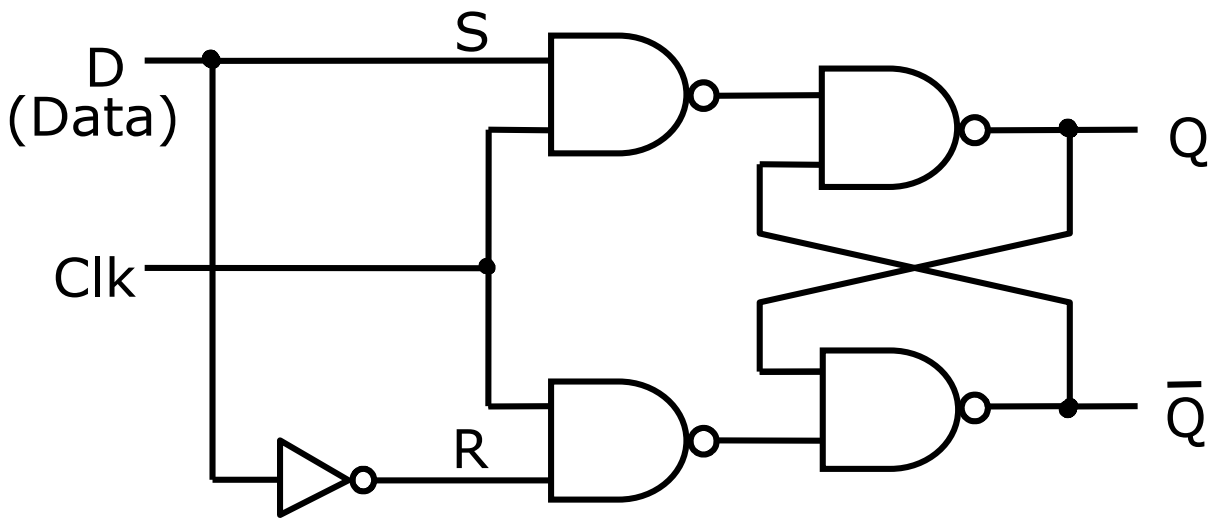


Vzbujevalna tabela

Q(t)	Q(t+1)	R	S	D
0	0	X	0	0
0	1	0	1	1
1	0	1	0	0
1	1	0	X	1

Sinhroni D zapah

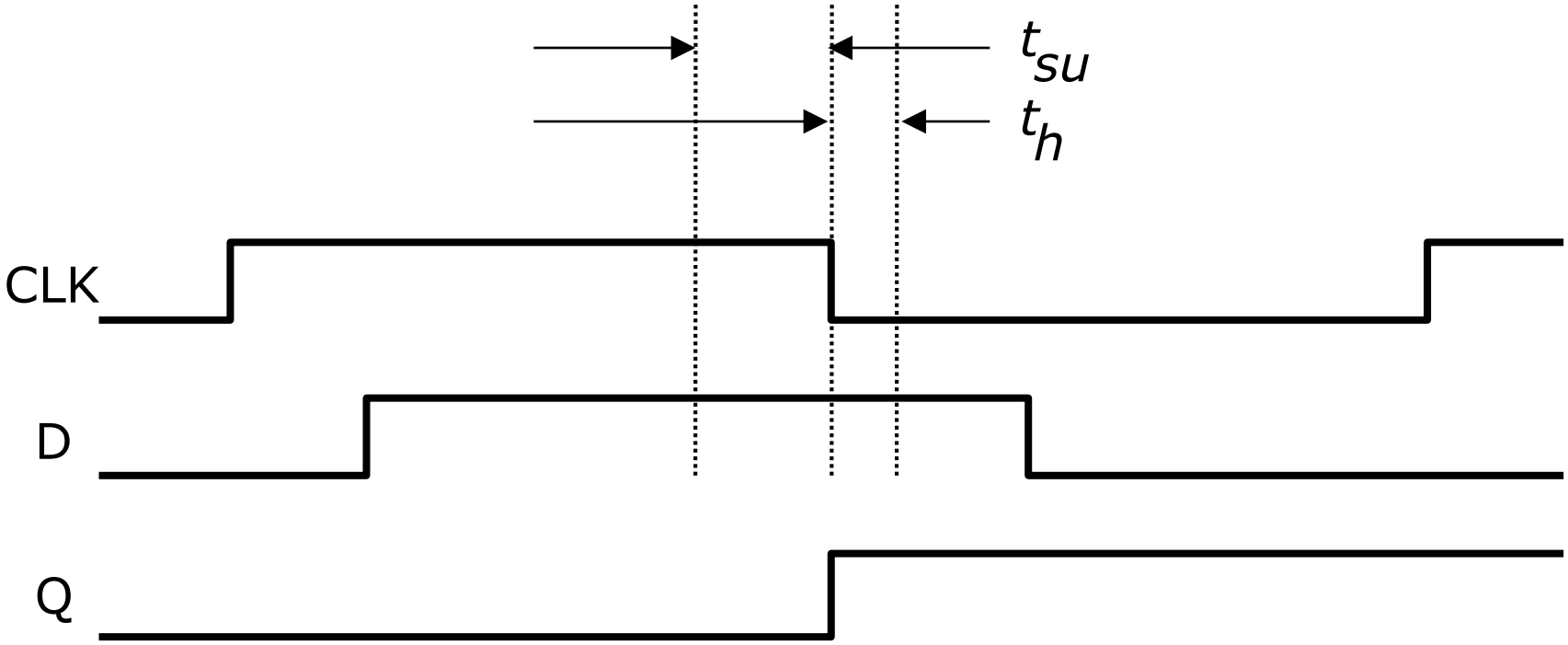
- R in S povežemo preko inverterja (R=S'),
- Dobimo **sinhroni D zapah** - zapah z enim vhodom, **D**, ki hrani vrednost tega vhoda skladno z signalom ure. (*gated D latch)*



Učinki zakasnitev širjenja

- Za sinhroni D zapah (in ostale) se D ne spreminja ob času ko gre signal ure iz 1 na 0
- čas vzpostavitve (***setup time*** (t_{su})):
  Minimalna zakasnitev, med katero mora biti signal D stabilen **pred** prehodom signala ure (1->0)
- čas zadrževanja (***hold time*** (t_h)):
  Minimalna zakasnitev, med katero mora biti signal stabilen **po** prehodu signala ure (1->0)
- Tipične zakasnitve za CMOS družino: t_{su}=3ns and t_h=2ns

Časa vzpostavitve in zadrževanja



Razvoj digitalnih sistemov

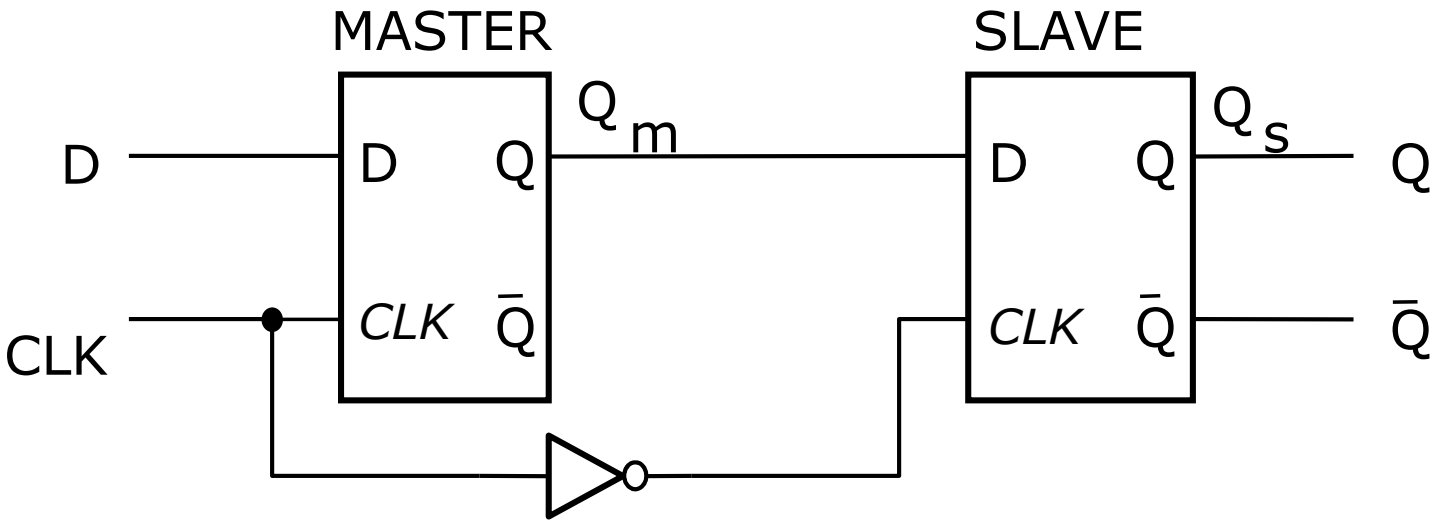
Flip-flopi, registri in števci:
Flip-flopi

Flip-flopi (FF)

- Sinhrona vezja zapahov, ki smo jih predstavili, lahko spreminjajo stanja **več kot enkrat** med "aktivno" periodo signala ure
- Spominski elementi, ki lahko spreminjajo stanje **največ enkrat** med periodo signala ure (CLK) omogočajo lažji nadzor nad vezjem:
  - Rob proženja:
    - sprednji ali pozitivni rob proženja (CLK = 0➔1)
    - zadnji ali negativni rob proženja (CLK = 1➔0)
- Predstavili bomo dva tipa takšnih vezij
  - Master-slave FF
  - Robno prožen (edge triggered) FF

Master-Slave D-FF*

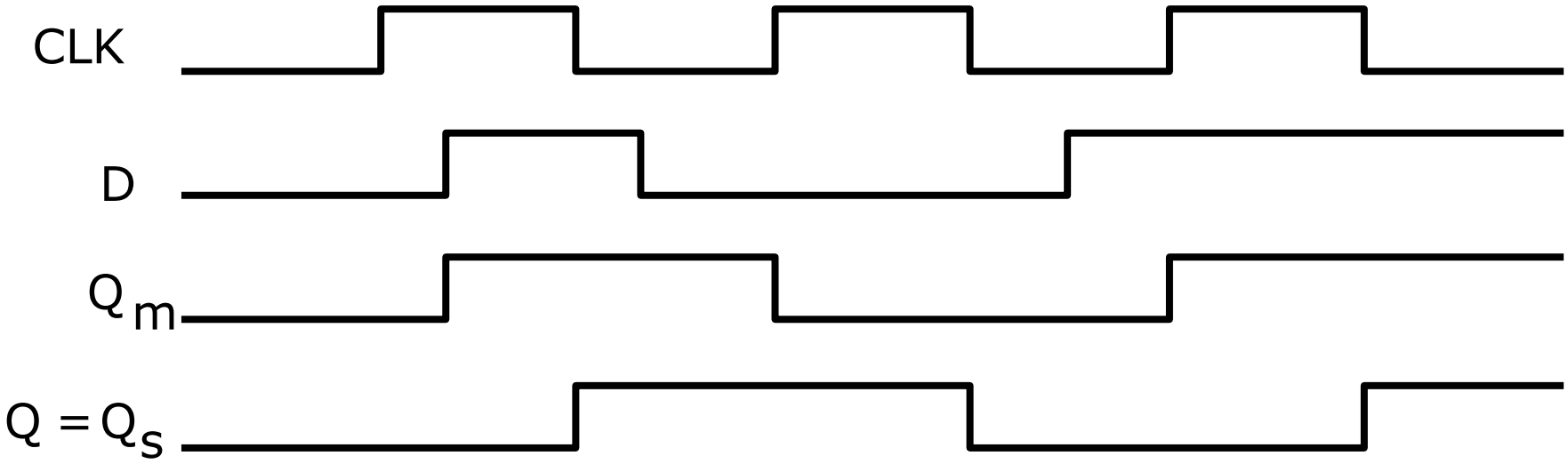
- Sestavljen iz 2 **sinhronih** D zapahov
  - Glavna celica, ***master***, spreminja stanje samo ko je CLK = 1
  - Delovna celica, ***slave***, spreminja stanje samo ko je CLK = 0

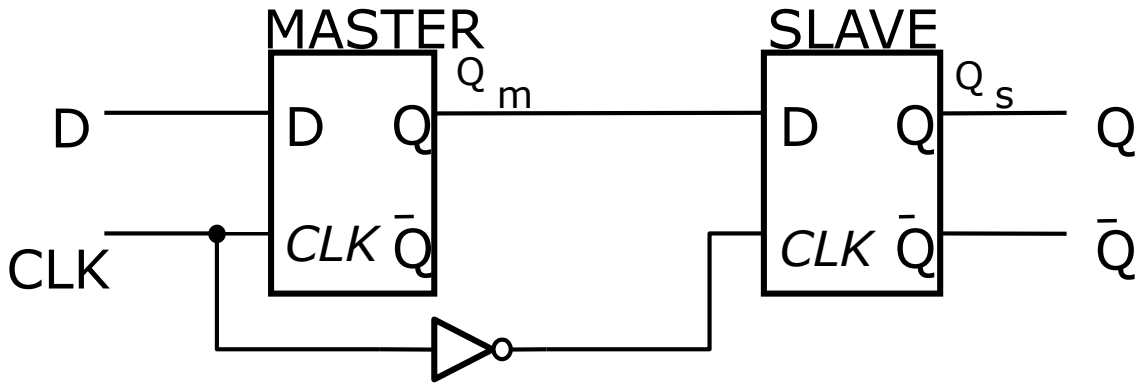


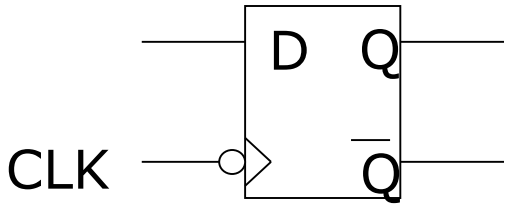
*Celica s predpomnjenjem.

38 tranzistorjev

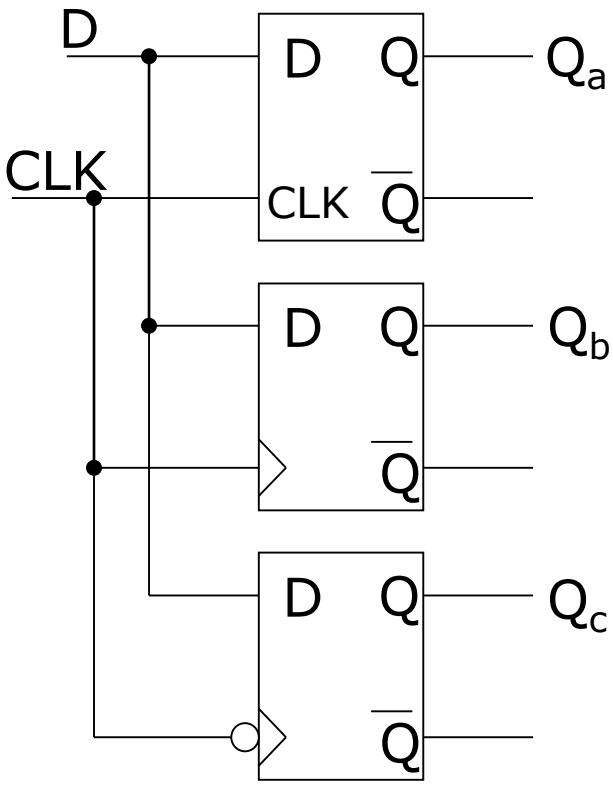
Master-Slave D-FF

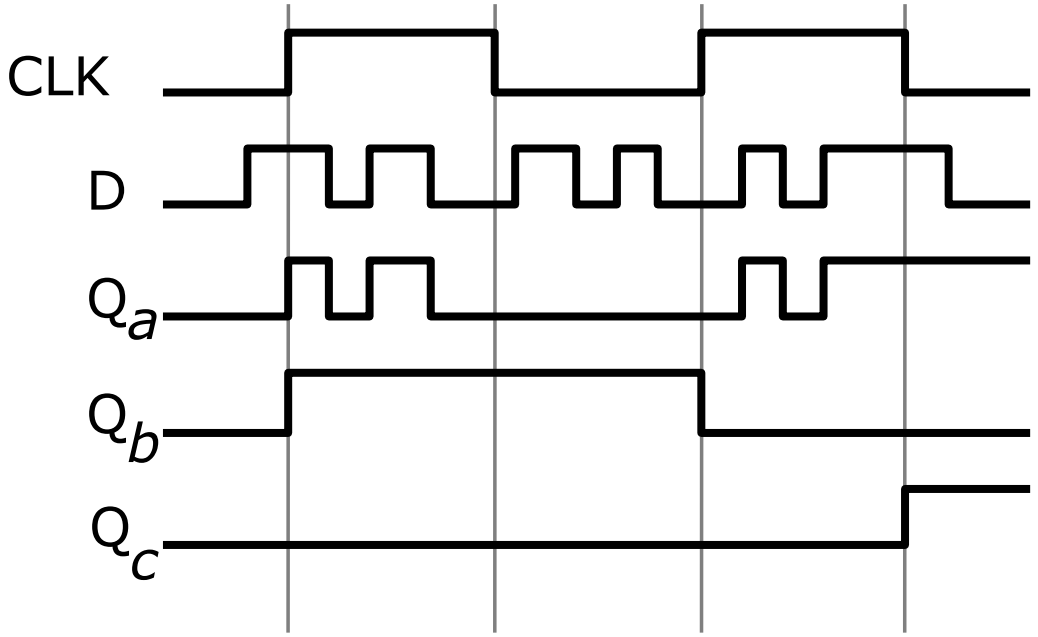






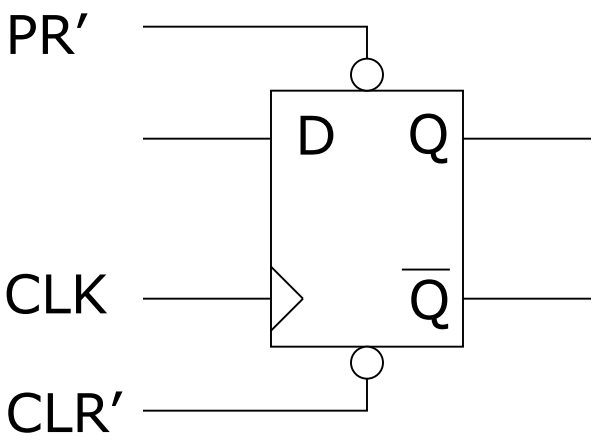
Primerjava D spominskih elementov





Vhodi za brisanje in prednastavitev

- Če želimo vsiliti stanje v (Q=1) ali (Q=0)
- Praktične izvedbe FF imajo asinhrona vhoda ***preset*** (prednastavitev) in ***clear*** (brisanje)



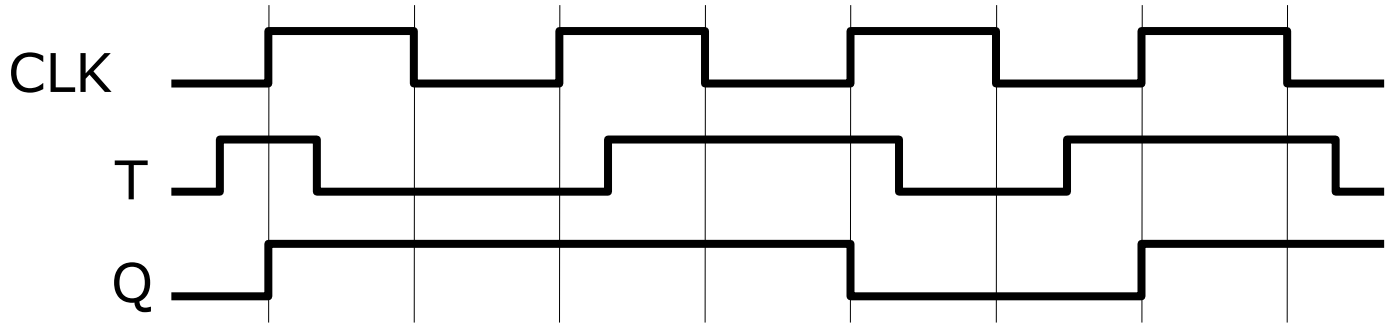
PR'=0 ➔ Q=1

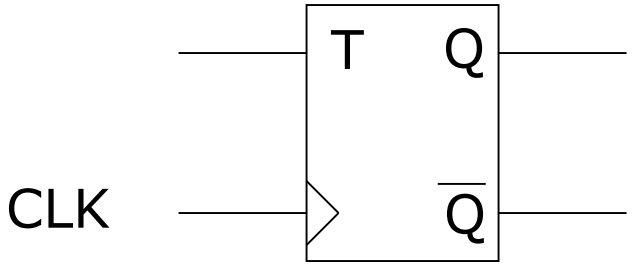
CLR'=0 ➔ Q=0

T-FF

- T ➔ ‘zamenja’ (‘toggle’) stanje, ko je T=1

T	Q(t+1)
0	Q(t)
1	Q’(t)





Prožen na pozitivni rob signala ure (CLK prehod 0➔1)

T-FF

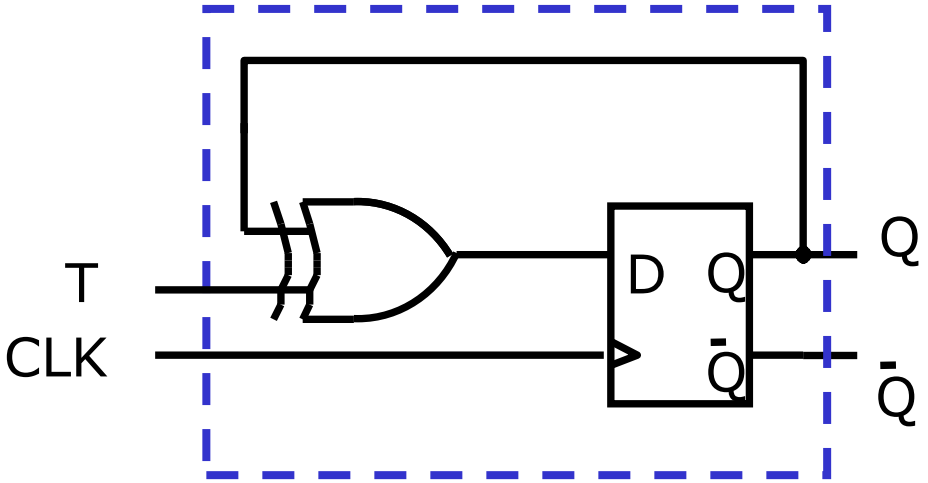
T	Q(t)	Q(t+1)	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

$D = T \cdot Q' + T' \cdot Q$

$D = T \oplus Q$

Karakteristična tabela

T	Q(t+1)
0	Q(t)
1	Q'(t)

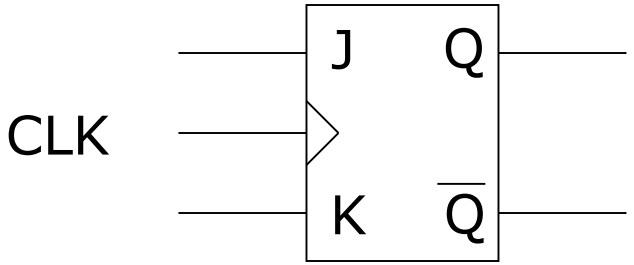


JK-FF

- JK-FF združuje lastnosti RS in T-FF

Karakteristična tabela

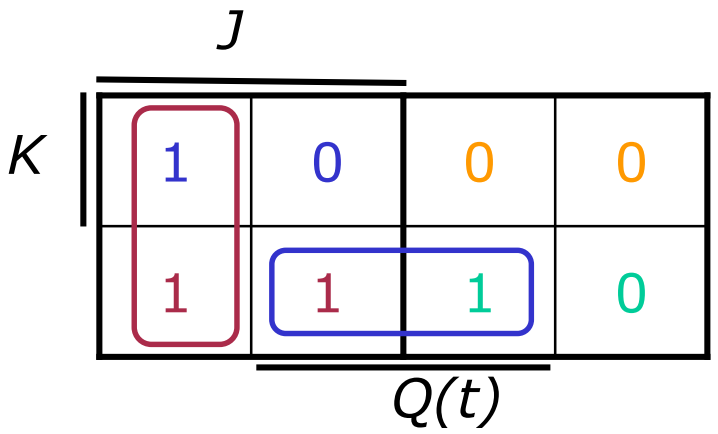
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q’(t)



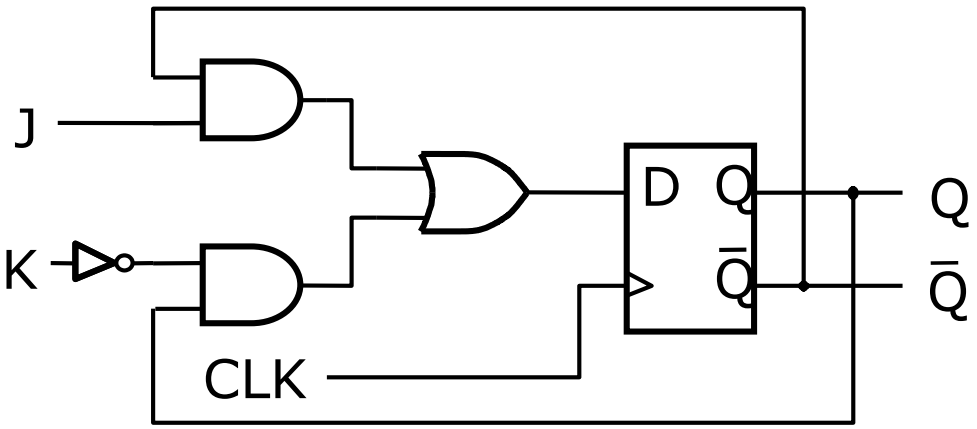
Robno prožen na pozitivni rob signala ure

JK-FF z D-FF

J	K	Q(t)	Q(t+1)	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

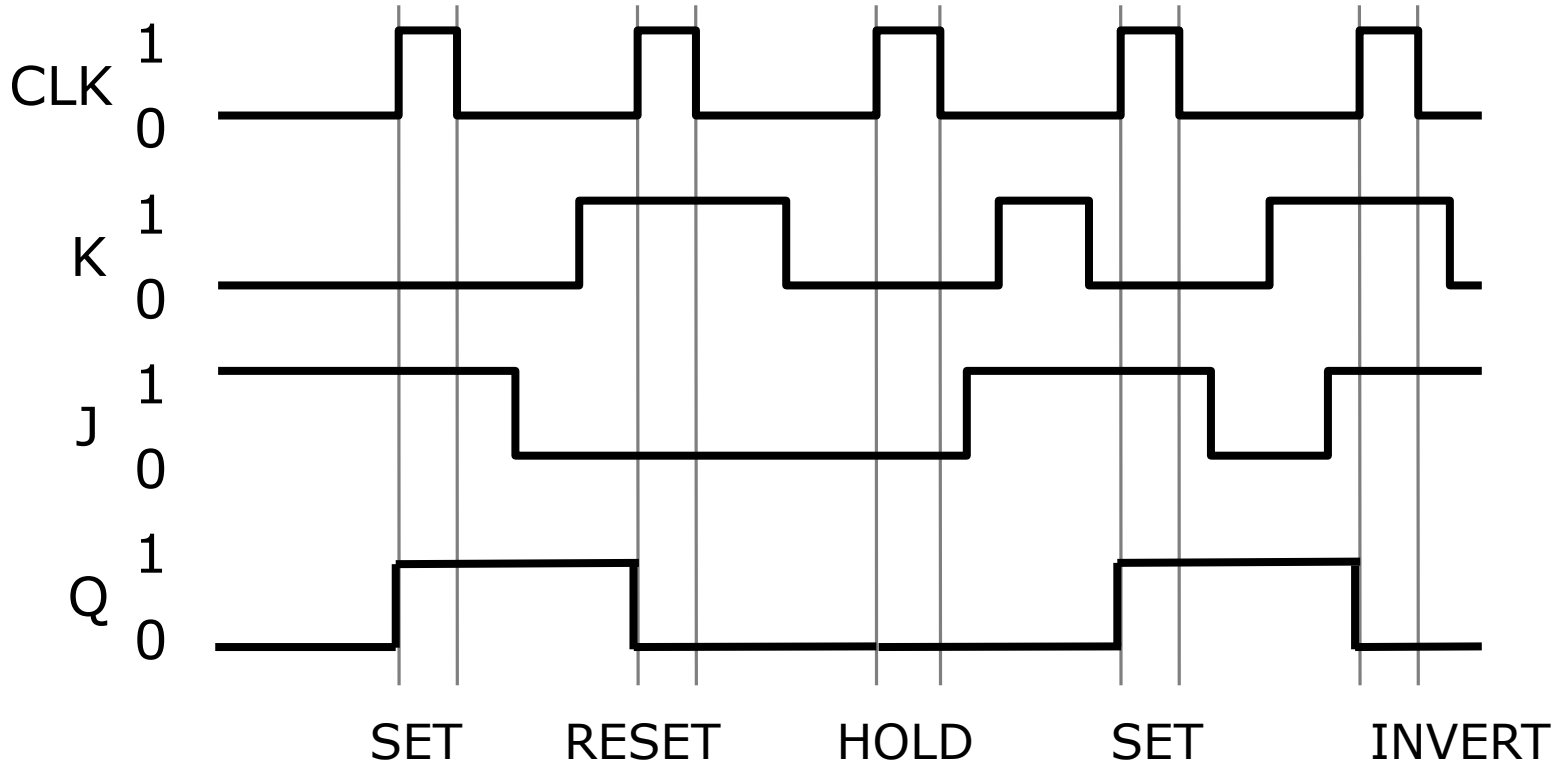


$$Q(t+1)=D=K'\cdot Q(t)+J\cdot Q(t)'$$



Časovni diagram JK-FF

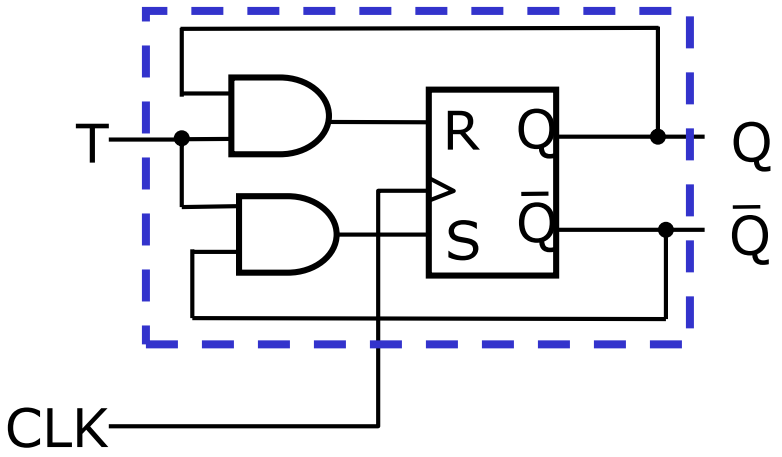
Dopolnite spodnji časovni diagram JK-FF



T-FF z RS-FF

T	Q(t)	Q(t+1)	R	S
0	0	0	X	0
0	1	1	0	X
1	0	1	0	1
1	1	0	1	0

$R = T \cdot Q(t)$ $S = T \cdot Q(t)'$



D zapah v VHDL (ang. D-latch)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY latch IS
PORT (D, Clk : IN STD_LOGIC;
      Q : OUT STD_LOGIC);
END latch;
ARCHITECTURE arch OF latch IS
BEGIN
PROCESS ( D, Clk )
   BEGIN
     IF Clk = '1' THEN
      Q <= D;
     END IF;
   END PROCESS;
END arch;
```

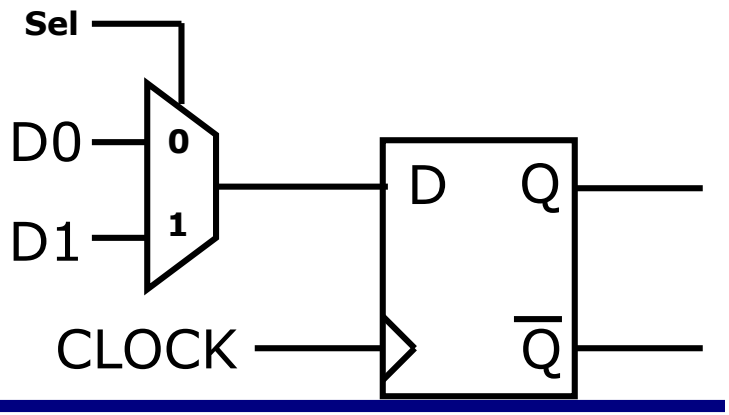
Uporablja implicitni spomin!

D-FF z asinhronima signaloma za brisanje (clear) in postavljanje (preset)

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dff IS
PORT (D, Clock, nClr, nPreset : IN STD_LOGIC;
              Q : OUT STD_LOGIC);
END dff ;
ARCHITECTURE Behavior OF dff IS
BEGIN
PROCESS ( Clock, nClr, nPreset )
   BEGIN
      IF nClr = '0' THEN
            Q <= '0';          --brisi
      ELSIF nPreset = '0' THEN
            Q <= '1';          --postavi
      ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D;            --trenutno stanje
      END IF;
   END PROCESS;
END Behavior;
```

D-FF z MUX vhodom - ULM

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY muxdff IS
PORT ( D0, D1, Sel, Clock : IN STD_LOGIC ;
             Q : OUT STD_LOGIC ) ;
END muxdff ;
ARCHITECTURE Behavior OF muxdff IS
BEGIN
PROCESS
   BEGIN
     WAIT UNTIL Clock'EVENT AND Clock = '1' ;
     IF Sel = '0' THEN
      Q <= D0;
     ELSE
      Q <= D1;
     END IF;
   END PROCESS;
END Behavior;
```



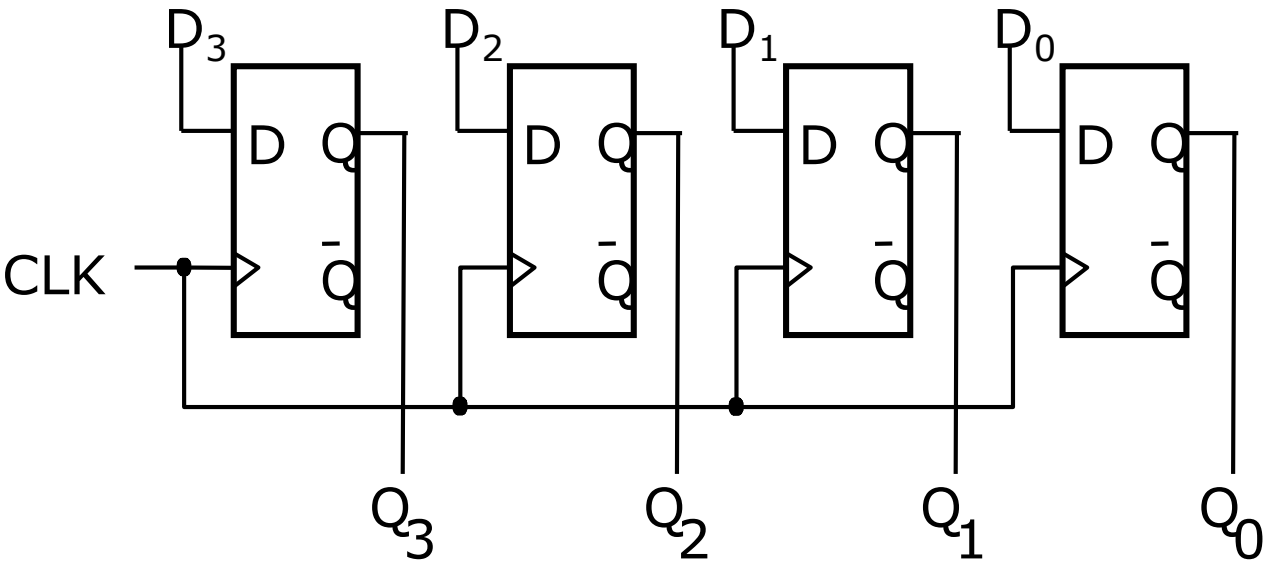
Razvoj digitalnih sistemov

Flip-flopi, registri in števci:
Registri

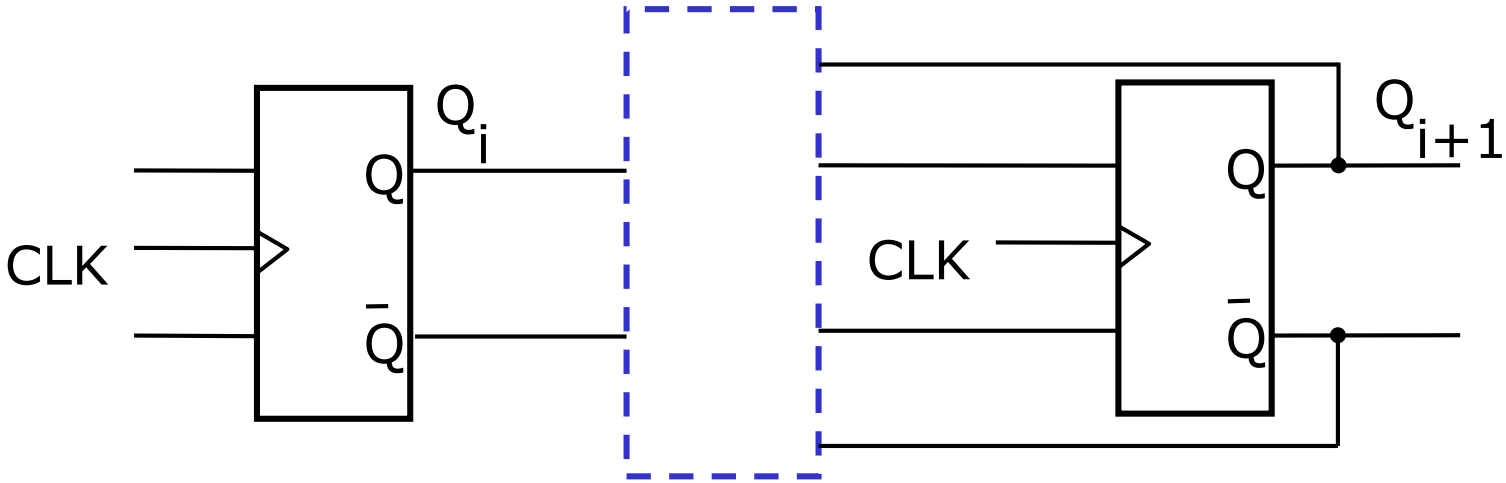
Registri

- Flip-flop hrani 1 bit informacije
- ***Register*** je niz *n* flip-flopov, v katerem shranimo *n* bitov podatkov
- Vrste registrov:
  - Shranjevalni
  - Pomikalni
  - Splošni ali univerzalni
- Razdelitev glede na možnost vzporednega vpisa in dostopnost izhodov celic:
  - SISO (zaporedni vhod, zaporedni izhod),
  - SIPO (zaporedni vhod, vzporedni izhod)
  - PISO (vzporedni vhod, zaporedni izhod)
  - PIPO (vzporedni vhod, vzporedni izhod)

Shranjevalni register



Pomik vsebine desno za eno mesto



$Q_i(t)$	$Q_{i+1}(t)$	$Q_{i+1}(t+1)$	D	R	S	J	K	T
0	0	0	0	X	0	0	X	0
0	1	0	0	1	0	X	1	1
1	0	1	1	0	1	1	X	1
1	1	1	1	0	X	X	0	0

Pomikalni register s paralelnim dostopom

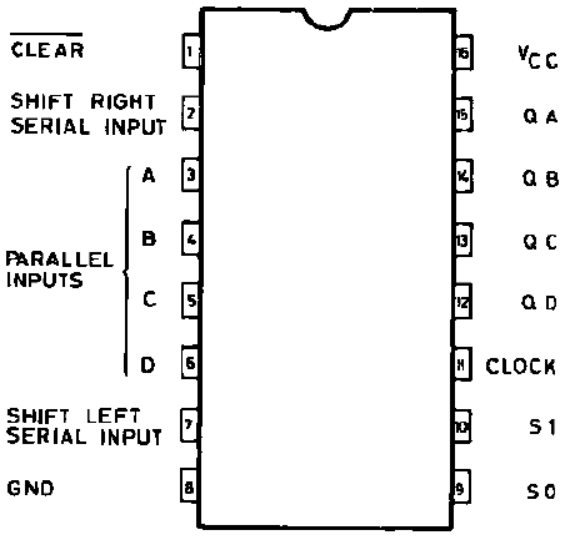
- Prenos *n*-bitov naenkrat ➔ vzporedni (***paralelni***) prenos
- Prenos *1*-bit naenkrat ➔ zaporedni (***serijski***) prenos
- Vzporedno-zaporedna pretvorba podatkov (PISO)
  - Podatke najprej naložimo v register vzporedno (v enem ciklu signala ure) in nato vsebino registra pomikamo zaporedno (sinhroni serijski oddajnik - SPI)
- Zaporedno-vzporedna pretvorba podatkov (SIPO)
  - Podatki so sprejeti zaporedno, potem po *n* ciklih signala ure lahko paralelno preberemo vsebino registra kot *n*-bitno informacijo (sinhroni serijski sprejemnik - SPI)

Razvoj digitalnih sistemov

Flip-flopi, registri in števci:
TTL univerzalni register

Univerzalni register MSI 74194

PRIKLJUČEK	FUNKCIJA
CLEAR	Asinhrono brisanje (negativna logika)
SR	Zaporedni vhod (ob pomikanju desno)
A,B,C,D	Vzporedni vhodi
SL	Zaporedni vhod (ob pomikanju levo)
S0, S1	vhoda za določanje načina delovanja
CLOCK	Clock Input (negative edge triggered)
QA,QB,QC,QD	Vzporedni izhodi



Delovanje 74194

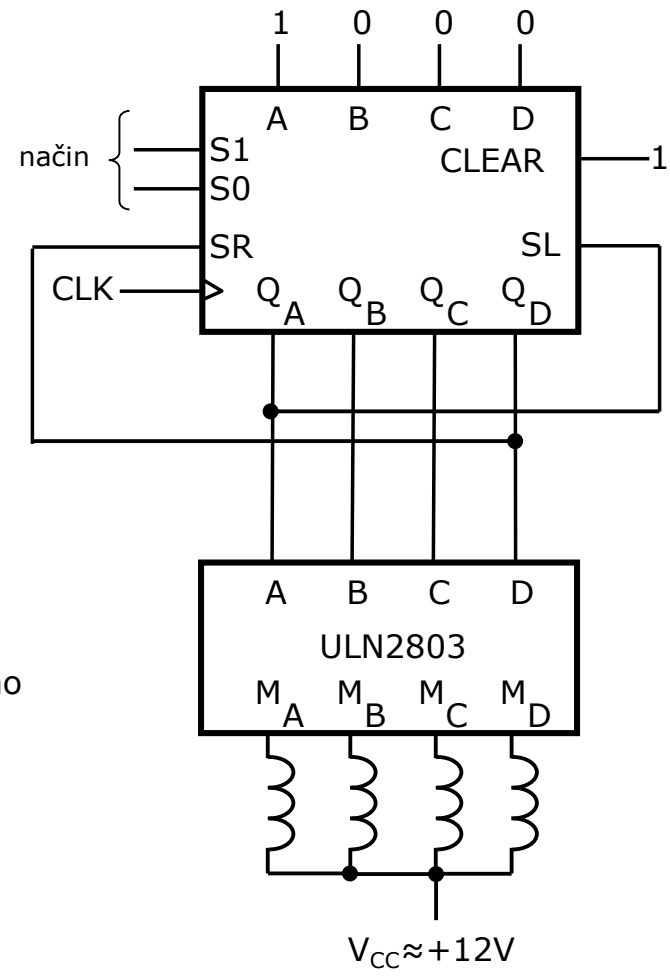
VHODI										IZHODI			
CLEAR	NAČIN		CLOCK	ZAPOREDNI		VZPOREDNI				QA	QB	QC	QD
	S1	S0		LEVO	DESNO	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	X	X	↓	X	X	X	X	X	X	QA0	QB0	QC0	QD0
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	QAn	QBn	QCn
H	L	H	↑	X	L	X	X	X	X	L	QAn	QBn	QCn
H	H	L	↑	H	X	X	X	X	X	QBn	QCn	QDn	H
H	H	L	↑	L	X	X	X	X	X	QBn	QCn	QDn	L
H	L	L	X	X	X	X	X	X	X	QA0	QB0	QC0	QD0

↓ - zadnji rob signala CLOCK
↑ - prednji rob signala CLOCK
4 operacije:

- S0='H' S1='H':Vzporedno nalaganje (a,b,c,d)➔ (Qa, Qb, Qc, Qd)
- S0='H' S1='L':Pomikanje levo (v smeri Qd➔Qa)
- S0='L' S1='H':Pomikanje desno (v smeri Qa➔Qd)
- S0='L' S1='L':Držanje vsebine

Uporaba MSI pomikalnih registrov: Krmilnik unipolarnega koračnega motorja.

NAČIN		trenutno				naslednje				
S1	S0	Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	
0	0									naloži
0	0									
0	0									
0	0									
0	1	1	0	0	0	0	1	0	0	motor levo
0	1	0	1	0	0	0	0	1	0	
0	1	0	0	1	0	0	0	0	1	
0	1	0	0	0	1	1	0	0	0	
1	0	1	0	0	0	0	1	0	0	motor desno
1	0	0	1	0	0	0	0	1	0	
1	0	0	0	1	0	0	0	0	1	
1	0	0	0	0	1	1	0	0	0	
1	1									motor stoji
1	1									
1	1									
1	1									

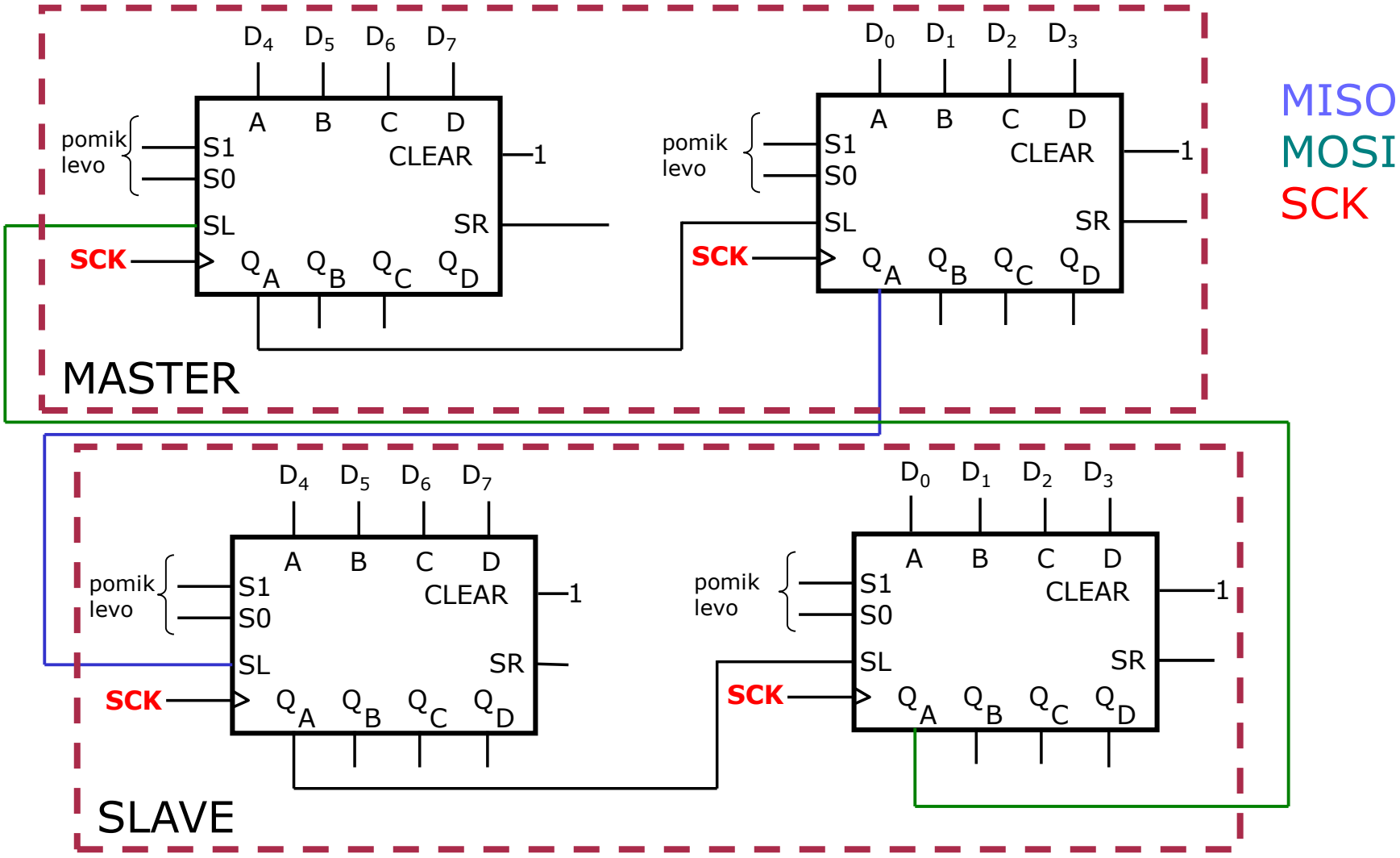


Vir: http://www.me.gatech.edu/mechatronics_lab/LabMaterials/EExercise3.pdf

Uporaba pomikalnih registrov

- Tvorba zakasnitev
- Vzporedno/zaporedna pretvorba in obratno
- Števci na osnovi pomikalnih registrov
- FIFO in LIFO pomnilnik
- LFSR (Linear Feedback Shift Register)
  - psevdonaključni generator
  - tvorba CRC (ostanek v XOR deljenju)

Vzporedno/zaporedna pretvorba – SPI protokol



Razvoj digitalnih sistemov

Flip-flopi, registri in števci:
Števci

Števci

- Števci so poseben primer aritmetičnih vezij, katerim se izhod povečuje/zmanjšuje za 1
- Števci štejejo število sprememb številnega signala (signala ure)!
- Vezja števcev se uporabljajo za več namenov
  - Štetje dogodkov (Counter)
  - Sekvenčno naslavljanje pomnilnika (DMA)
  - Časovnik:
    - Tvorba časovnih zakasnitev (timer)
    - Spremljanje pretečenega časa med dogodki (time delay)
- Modul štetja (M=8, pomeni da ima števec na izhodu vsa stanja od 0 do 7)
- Vrste števcev (asinhroni, sinhroni)

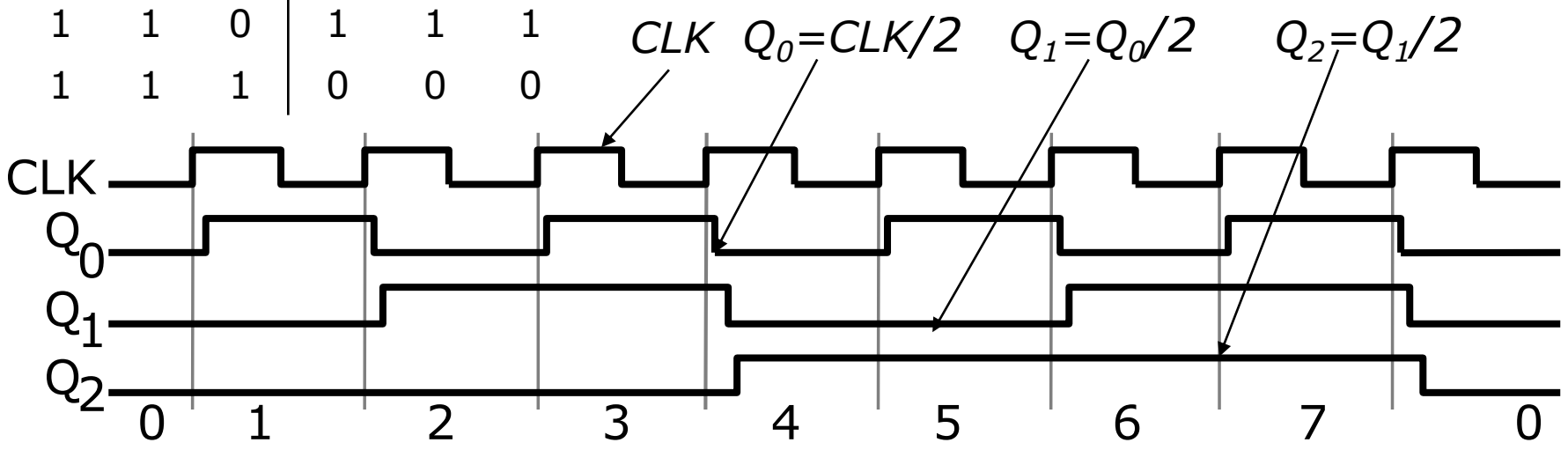
Asinhroni števci

- Asinhronim števcem pravimo tudi ***ripple*** števci. Zakaj?
  - *Vhodna* ura je vedno povezana samo na *en* FF
  - Signali ure za druge FF so izvedeni iz izhodov predhodnih FF
- Asinhrona oblika števca je počasna, zaradi kaskadne povezave signala ure
  - Izvor signala ure valovi (ripple) od stopnje do stopnje
  - Efekt valovanja (ripple effect) je podoben kot pri ripple carry seštevalniku
- Dobra lastnost teh števcev je, da so enostavni: ne potrebujejo dodatne kombinacijske logike.

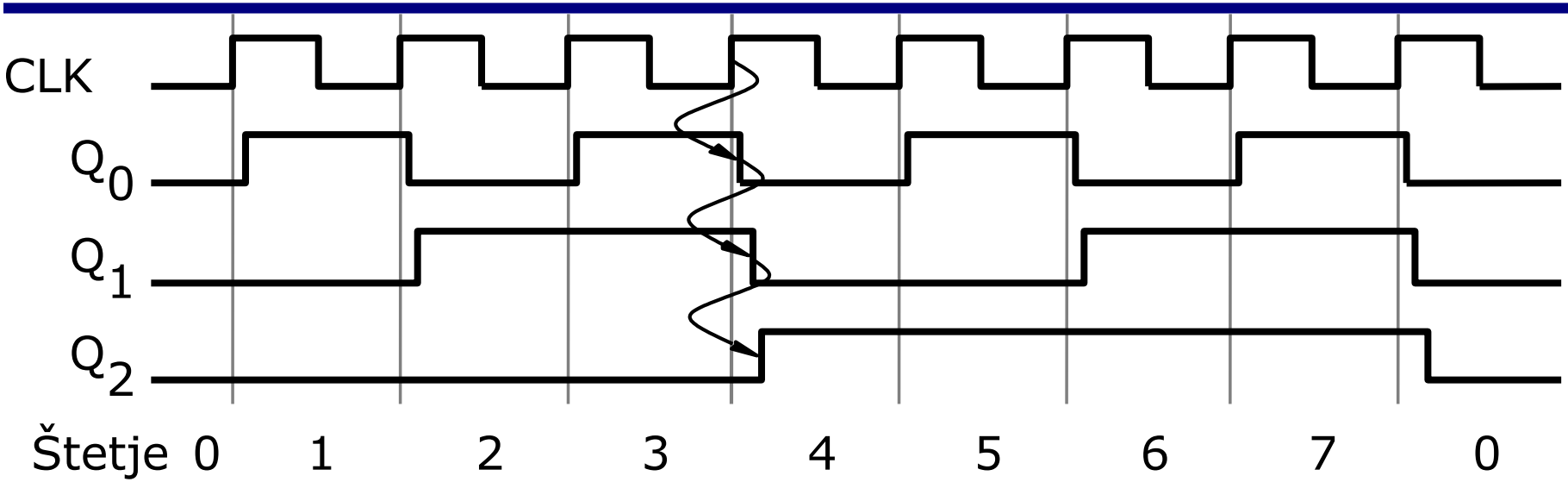
Asinhrono štetje s T-FF

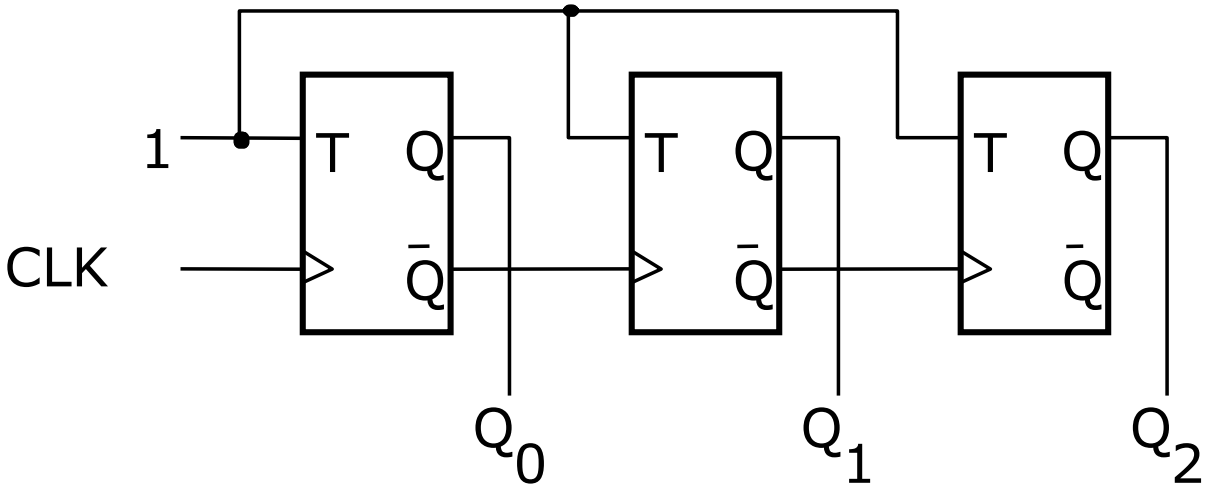
trenutno			*naslednje*		
Q_2	**Q_1**	**Q_0**	**Q_2**	**Q_1**	**Q_0**
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

T-FF že po svoji naravi delovanja deli frekvenco signala ure z 2, čim je vhod T='1'!

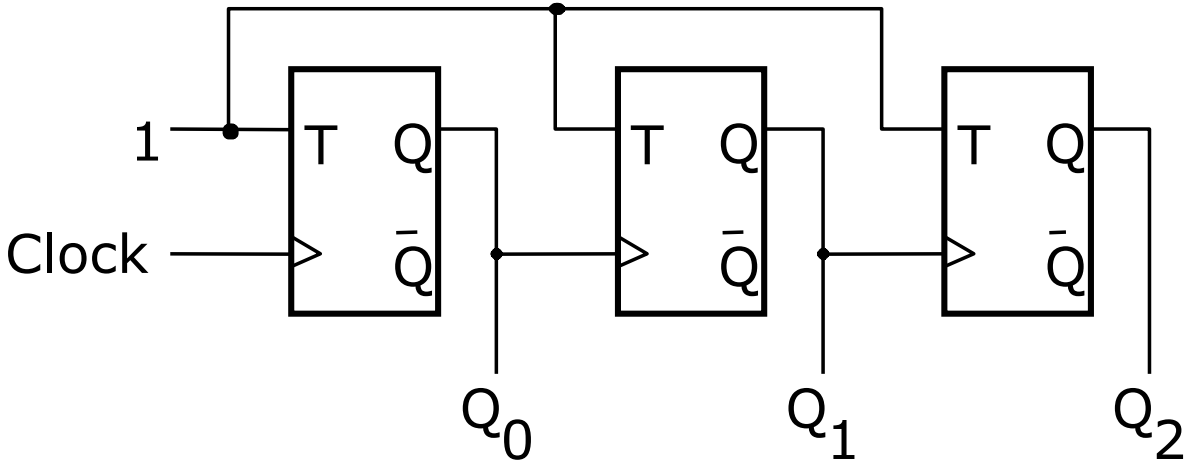


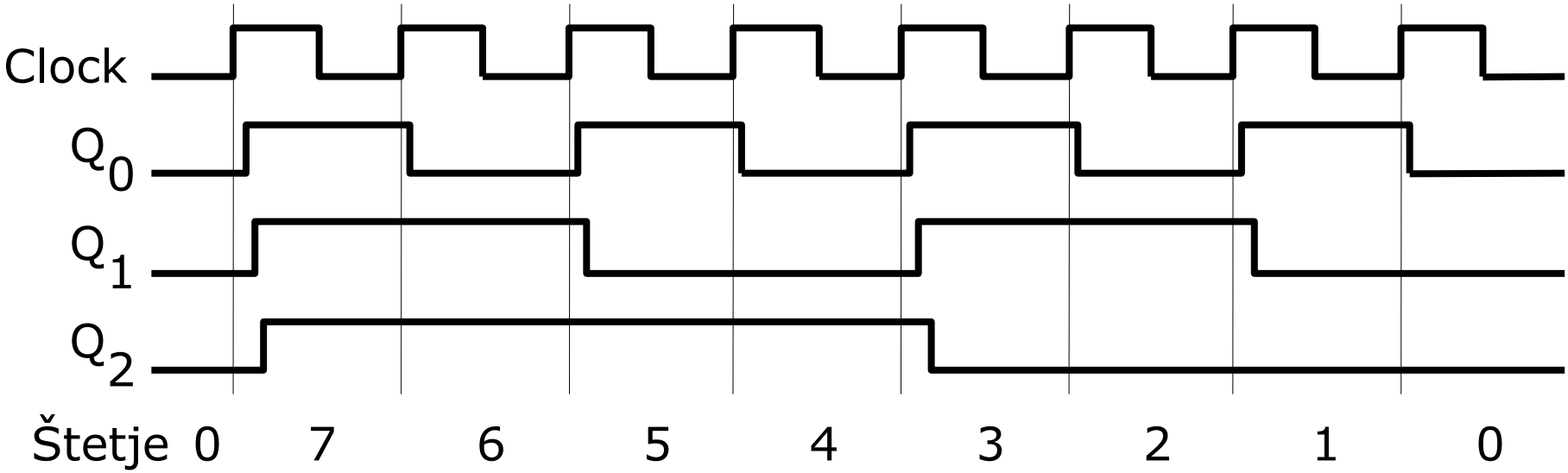
Naraščajoče štetje s T-FF



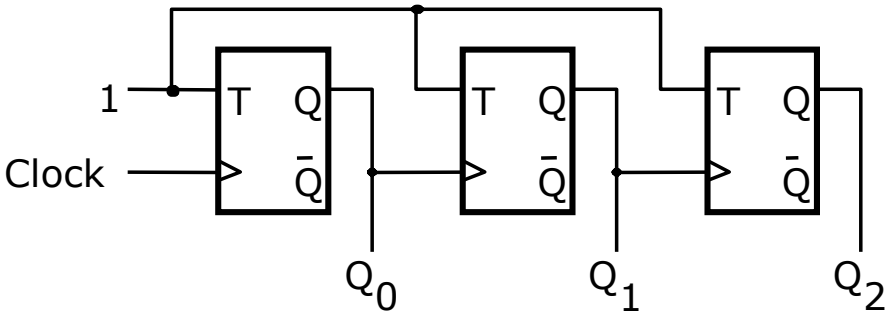


Padajoče štetje s T-FF



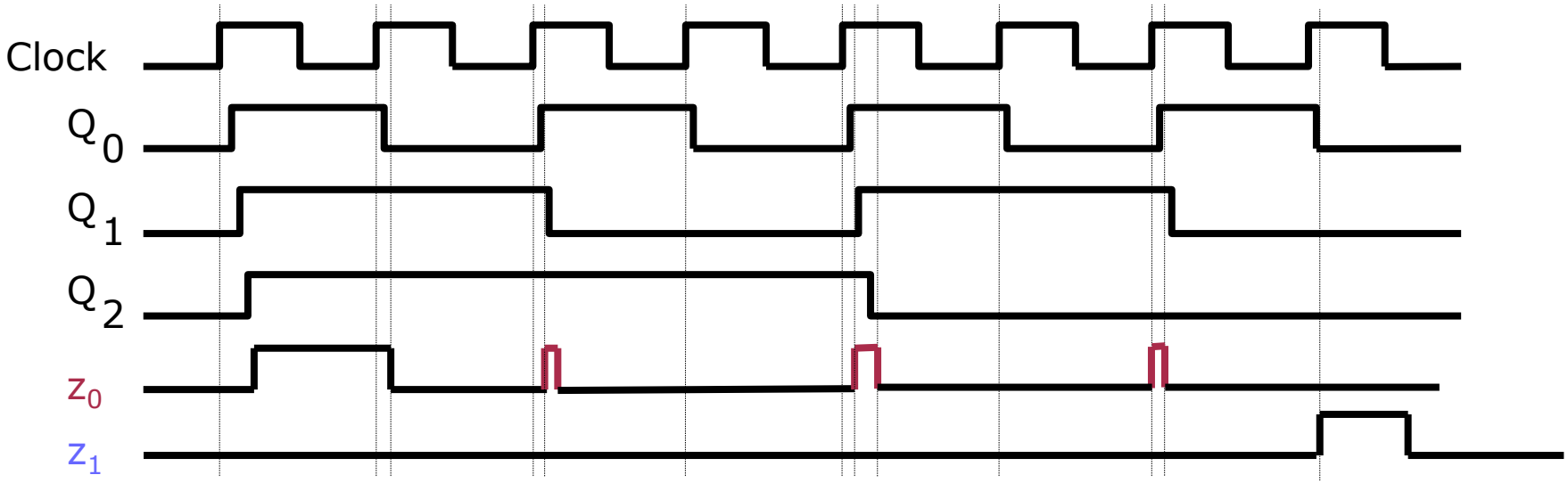


Problem detekcije izbranega stanja pri asinhronih števcih



$z_0 = Q_2 \cdot Q_1 \cdot Q_0$

$z_1 = Q_2' \cdot Q_1' \cdot Q_0'$



Na signalu z_0 se pojavijo motnje ➔ trava, zato je signal uporaben samo kot statičen.

Sinhroni števec navzgor s T-FF

trenutno			*naslednje*					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

T_2:

Q_1 \ Q_2				
	0	1	1	0
	0	0	0	0

Q_0

$T_0=1$

$T_1=Q_0$

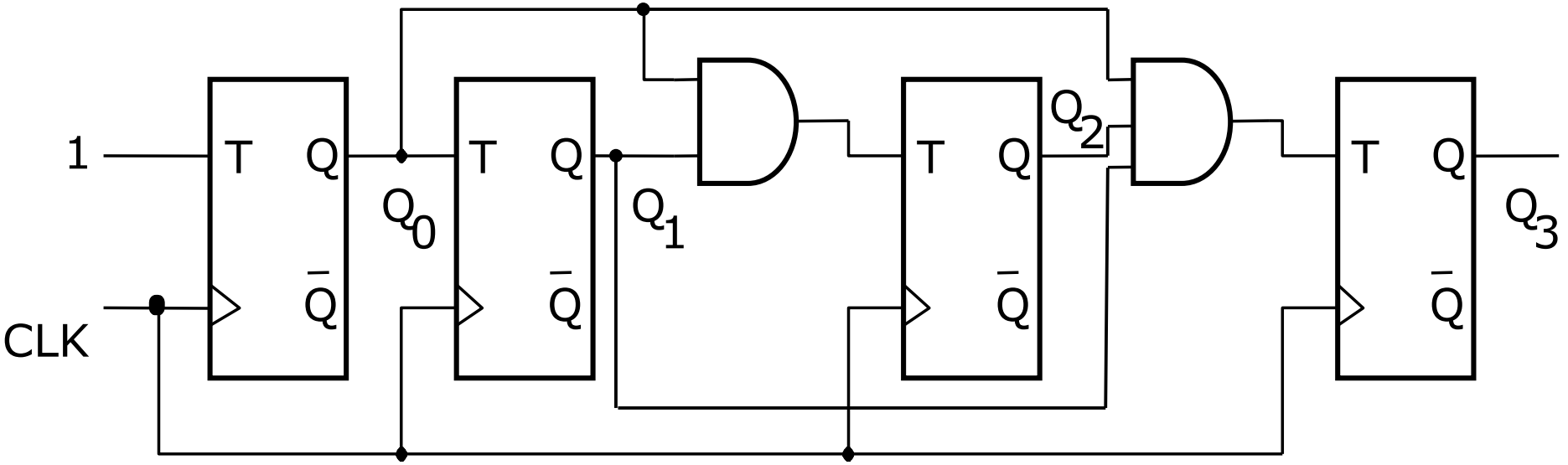
$T_2=Q_1 \cdot Q_0$

$T_n=Q_{n-1} \cdot \ldots \cdot Q_1 \cdot Q_0$

Enonivojska realizacija sinhronega števca navzgor s T-FF

$T_0=1$ $T_1=Q_0$ $T_2=Q_1 \cdot Q_0$ $T_3=Q_2 \cdot Q_1 \cdot Q_0$

$T_0=1$ $T_1=Q_0 \cdot T_0$ $T_2=Q_1 \cdot T_1$ $T_3=Q_2 \cdot T_2$

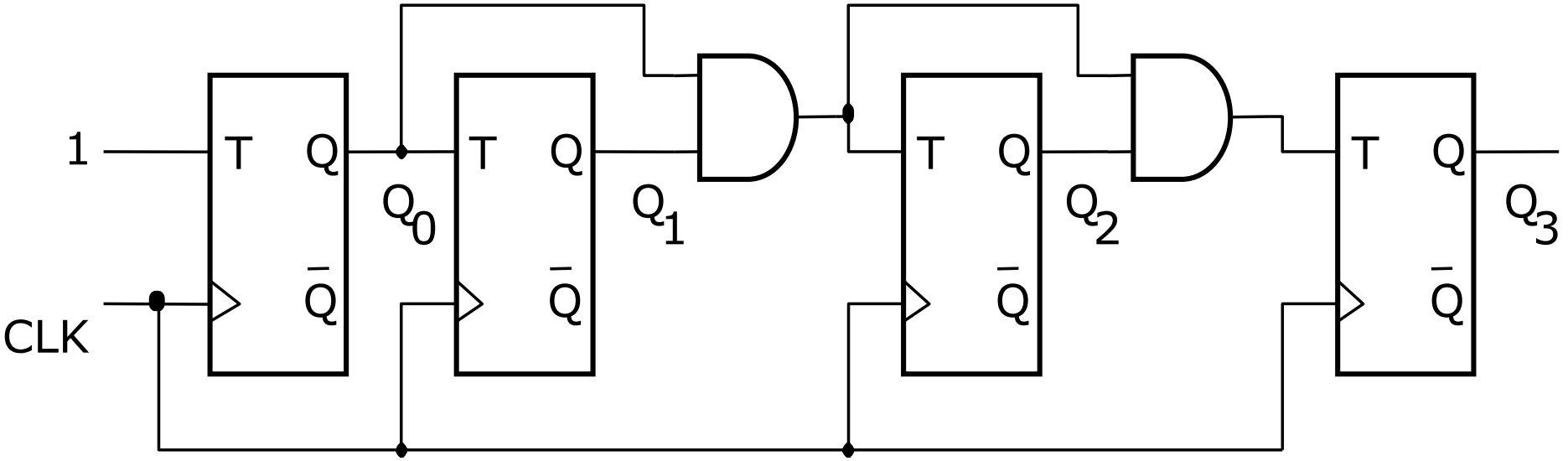


4-bitni sinhronski števec (naraščajoče štetje)
Izhodi števca so $Q_3Q_2Q_1Q_0$.
Problem: FAN-IN za n-vhodna AND vrata, zato števec raje realiziramo večnivojsko.

Večnivojska realizacija sinhronega števca navzgor s T-FF

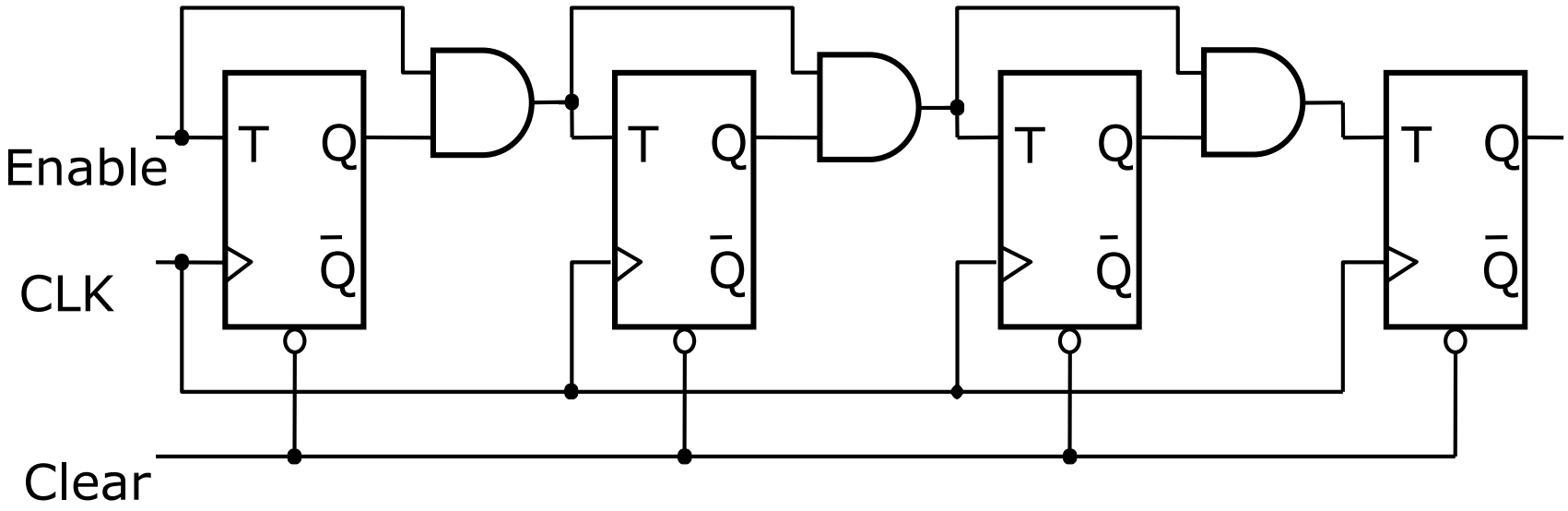
$T_0=1$ $T_1=Q_0$ $T_2=Q_1 \cdot Q_0$ $T_3=Q_2 \cdot Q_1 \cdot Q_0$

$T_0=1$ $T_1=Q_0 \cdot T_0$ $T_2=Q_1 \cdot T_1$ $T_3=Q_2 \cdot T_2$



- Vsi FF zamenjajo stanja z eno zakasnitvijo širjenja vrat (propagation delay) ➔ Ta realizacija ne upočasnjuje števca.
- Hitrost štetja je omejena s hitrostjo širjenja sprememb Q_0, katerega sprememba se mora širiti preko vseh stopenj.
- Čas širjenja ne sme biti daljši od ene periode signala ure.
- Problem so veliki moduli štetja

Omogočanje štetja in brisanje števca



Sinhroni števci z D-FF

Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	1	1
1	0	1	1	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1	1	1	0	1
1	1	0	1	1	1	1	0	1	1	1	0
1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0

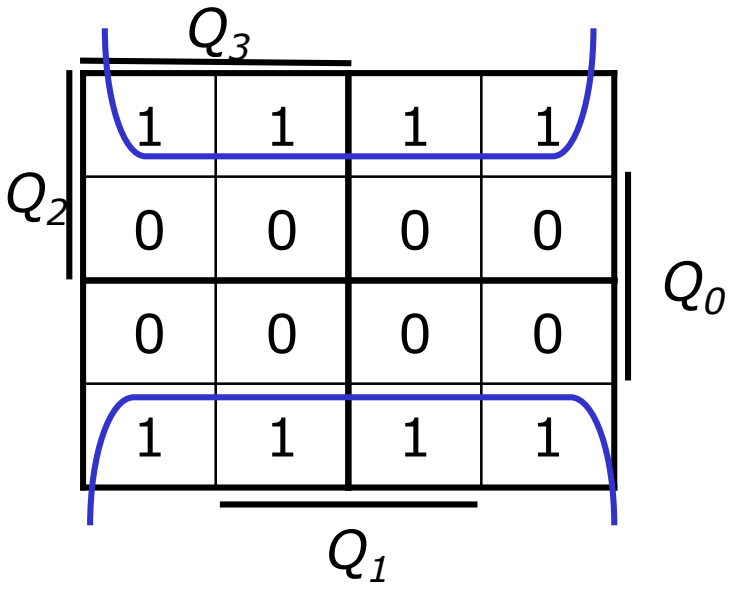
$D_0 = Q_0'$

$D_1 = Q_1 \oplus Q_0$

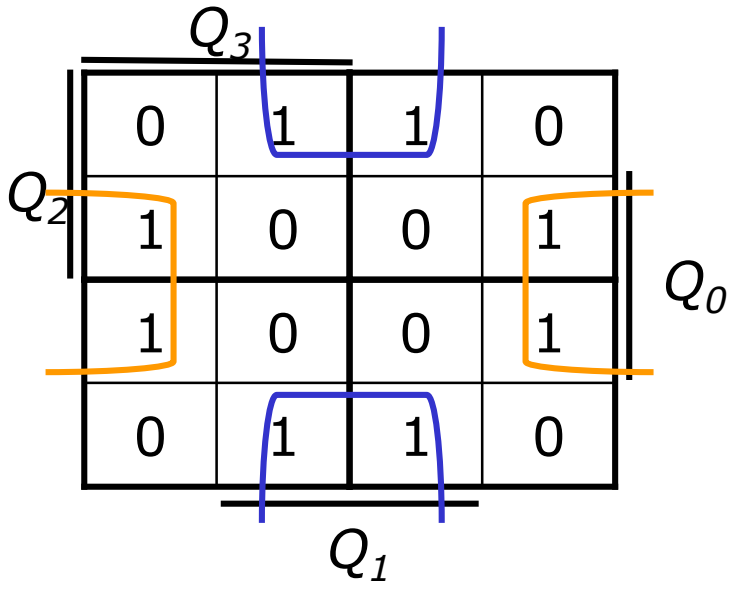
D_2 = ? ➔ Veitchev diagram

D_3 = ? ➔ Veitchev diagram

Sinhroni števci z D-FF



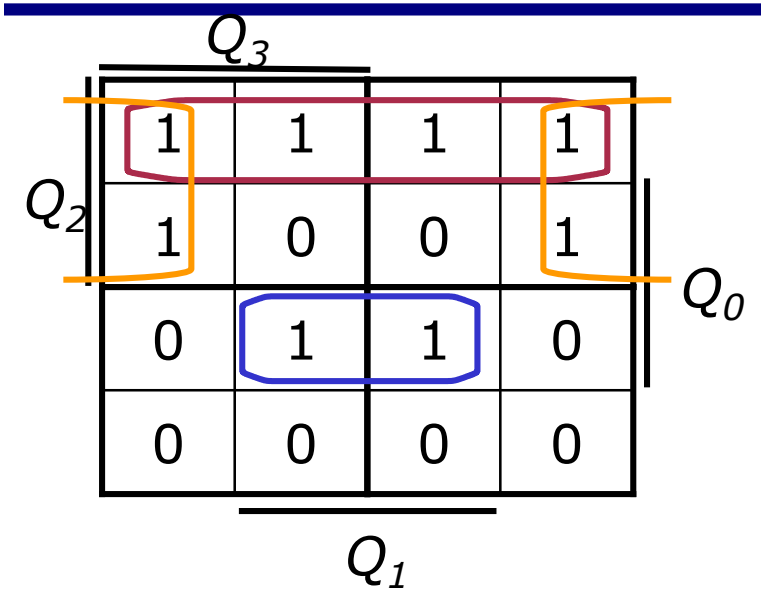
$D_0=Q_0'$



$D_1=Q_1\cdot Q_0'+Q_1'\cdot Q_0$

$D_1=Q_1 \oplus Q_0$

Sinhroni števci z D-FF

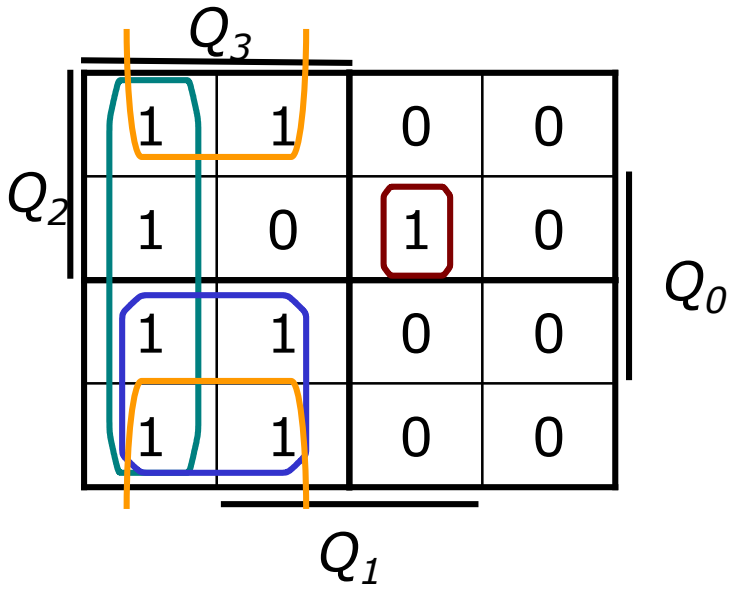


$D_2=Q_2\cdot Q_0'+Q_2\cdot Q_1'+Q_2'\cdot Q_1\cdot Q_0$

$D_2=Q_2\cdot (Q_0'+Q_1')+Q_2'\cdot Q_1\cdot Q_0$

$D_2=Q_2\cdot (Q_0\cdot Q_1)'+Q_2'\cdot (Q_1\cdot Q_0)$

$D_2= Q_2 \oplus Q_1\cdot Q_0$



$D_3=Q_3\cdot Q_1'+Q_3\cdot Q_2'+Q_3\cdot Q_0'+Q_3'Q_2Q_1Q_0$

$D_3=Q_3\cdot (Q_1'+Q_2'+Q_0')+Q_3'\cdot Q_2\cdot Q_1\cdot Q_0$

$D_3=Q_3\cdot (Q_1\cdot Q_2\cdot Q_0)'+Q_3'\cdot (Q_2\cdot Q_1\cdot Q_0)$

$D_3= Q_3 \oplus Q_2\cdot Q_1\cdot Q_0$

Sinhroni števci z D-FF

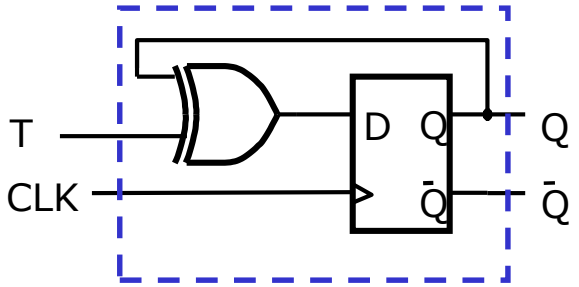
- 4-bitni števec gor šteje v sekvenci
  0, 1, 2,…, 15, 0, 1…
- Štetje je predstavljeno na 4 izhodih FF
  $Q_3Q_2Q_1Q_0$
- D vhodi so podani kot:

$$D_0 = Q_0 \oplus \text{Enable}$$

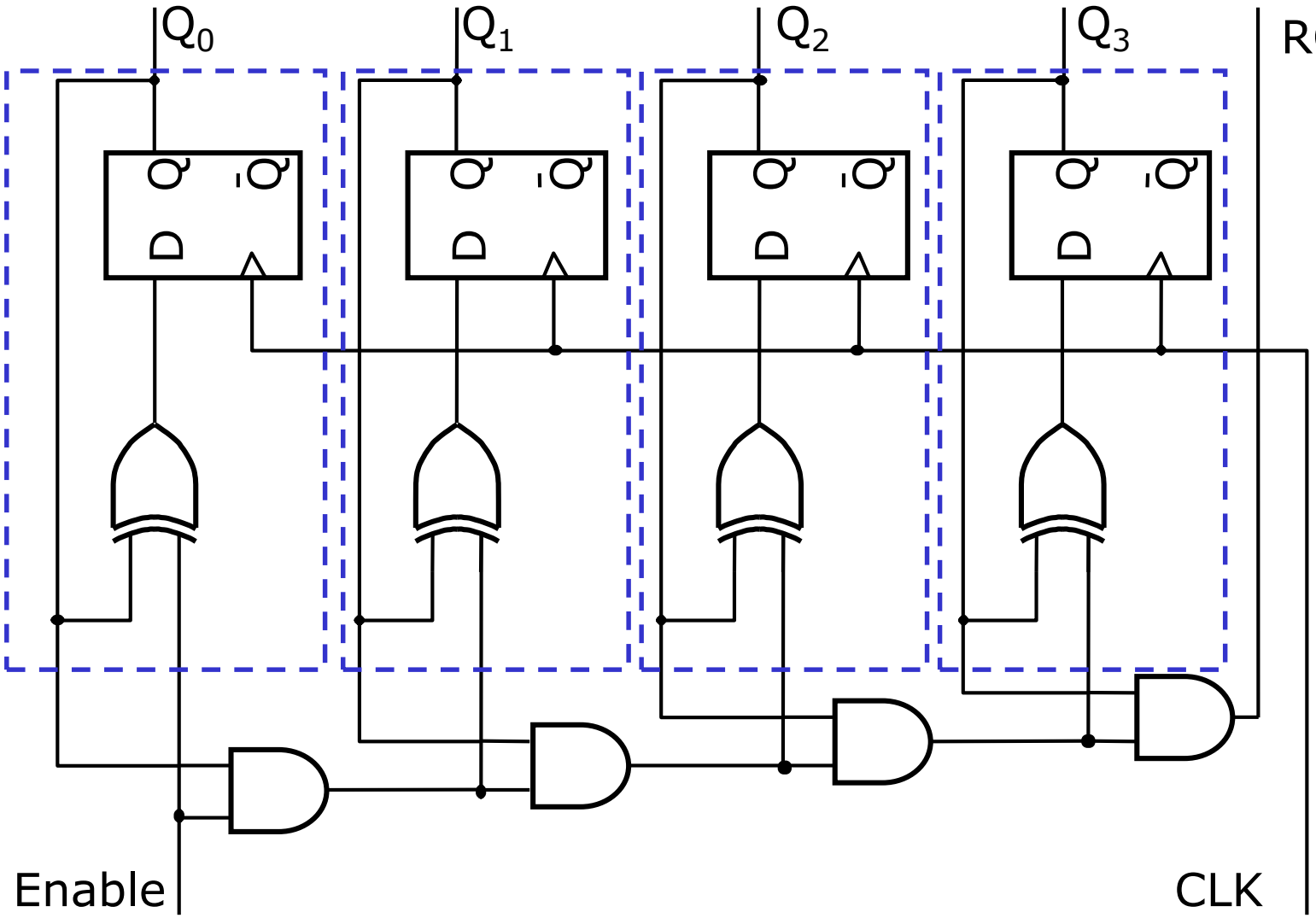
$$D_1 = Q_1 \oplus Q_0 \cdot \text{Enable}$$

$$D_2 = Q_2 \oplus Q_1 \cdot Q_0 \cdot \text{Enable}$$

$$D_3 = Q_3 \oplus Q_2 \cdot Q_1 \cdot Q_0 \cdot \text{Enable}$$



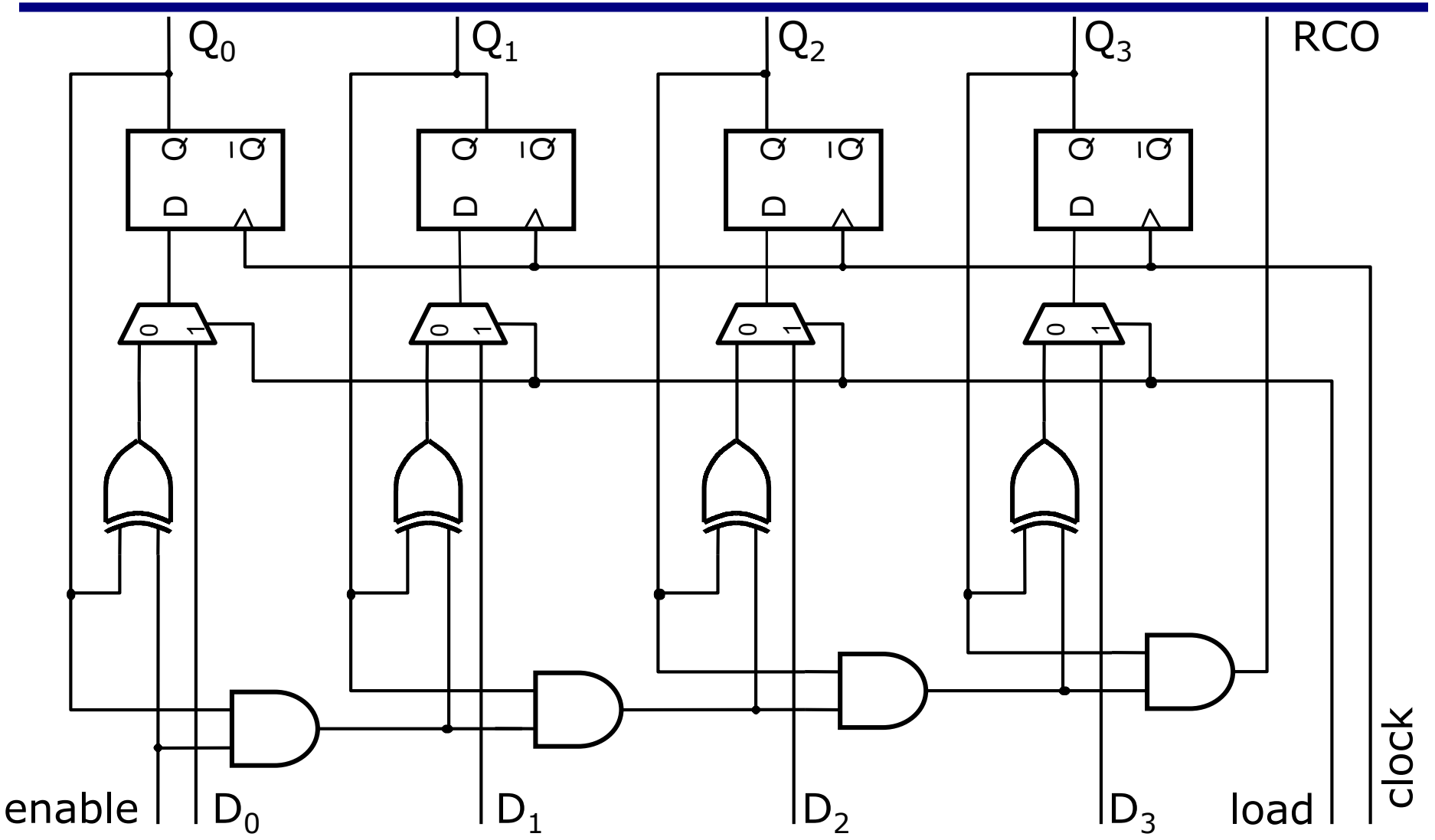
4 bitni števec navzgor z D-FF



Števci z vzporednim vpisom

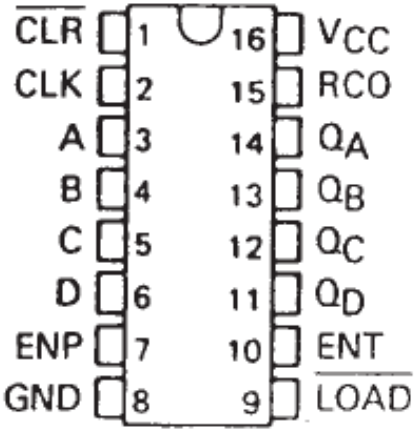
- Običajno števci štejejo od vrednosti 0 dalje
  - Postavljanje števca na 0 NE IZVAJAMO z uporabo *clear* vhoda na FF!
- Štejemo pa lahko tudi *od* vrednosti, ki ni 0.
- Dodati moramo vezje, ki služi **vzporednemu nalaganju** (*parallel load*)
- Kontrolni vhod, *load*, uporabljamo za določanje **načina** (*mode*) delovanja
  - Load=0, štej
  - Load=1, naloži vzporedno vrednost v števec

Števci z vzporednim vpisom



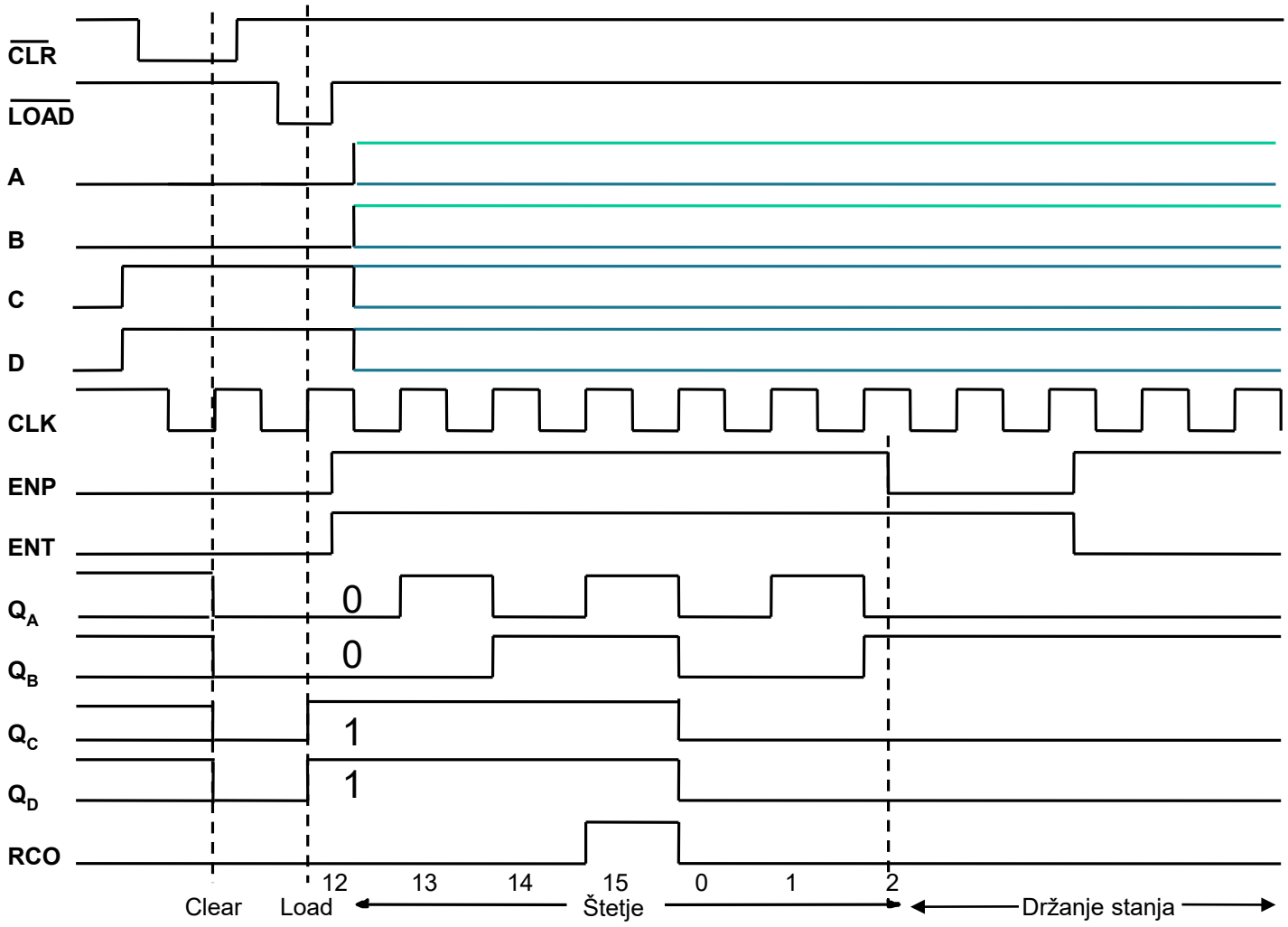
4-bitni sinhroni MSI števec 74163

- MSI števec 74163 ➔ 4-bitni sinhroni števec z vzporednim vpisom :
  - CLR='0' ➔ zbriše števec
    (iz vhodov DCBA na izhode QD, QC, QB, QA)
  - LOAD ='0' ➔ naloži stanje
    (iz vhodov DCBA na izhode QD, QC, QB, QA)
  - ENP in ENT ='1' ➔ šteje
  - Če ni pozitivnega roba CLK ➔ ohranja stanje
  - RCO postane '1'
    pri prehodu iz stanja 1111➔0000

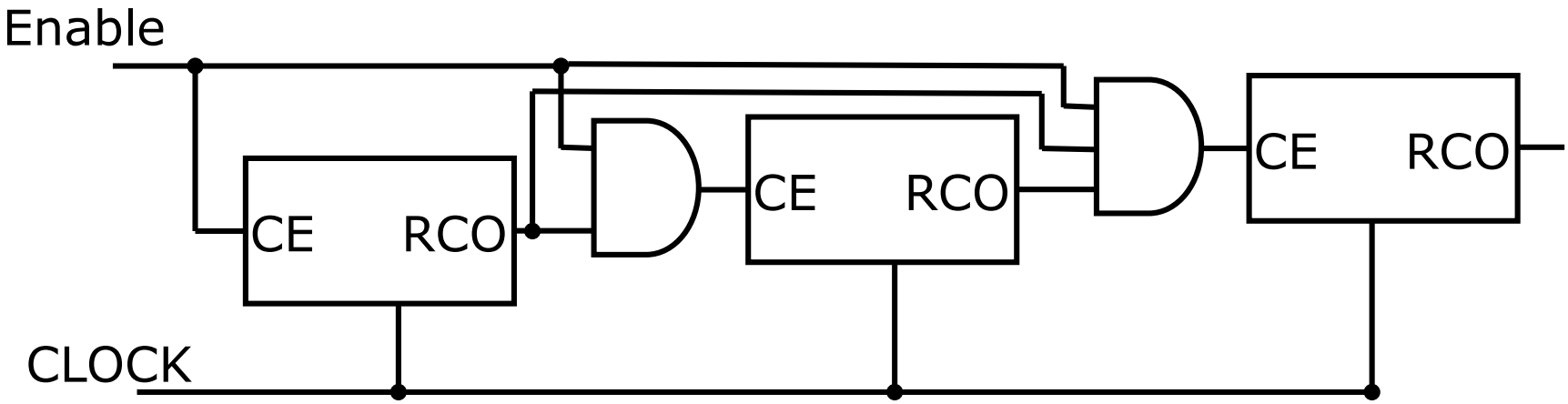


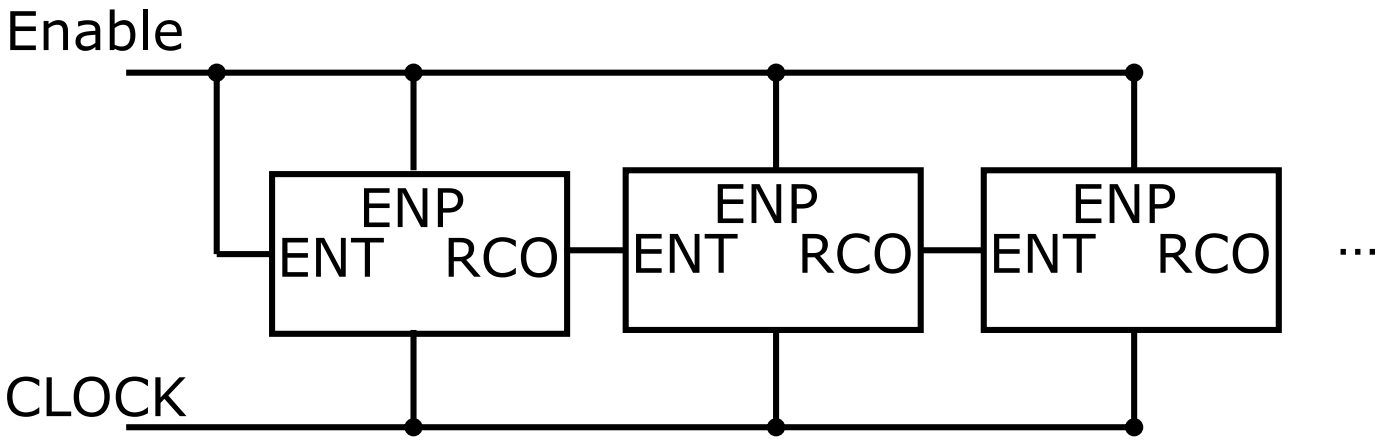
CLR	LOAD	ENP	ENT	CLK	A	B	C	D	QA	QB	QC	QD	RCO
0	X	X	X	POS	X	X	X	X	0	0	0	0	0
1	0	0	0	POS	X	X	X	X	A	B	C	D	*1
1	1	1	1	POS	X	X	X	X	števec šteje				*1
1	1	1	X	X	X	X	X	X	QA0	QB0	QC0	QD0	*1
1	1	X	1	X	X	X	X	X	QA0	QB0	QC0	QD0	*1

74163: Časovni diagram

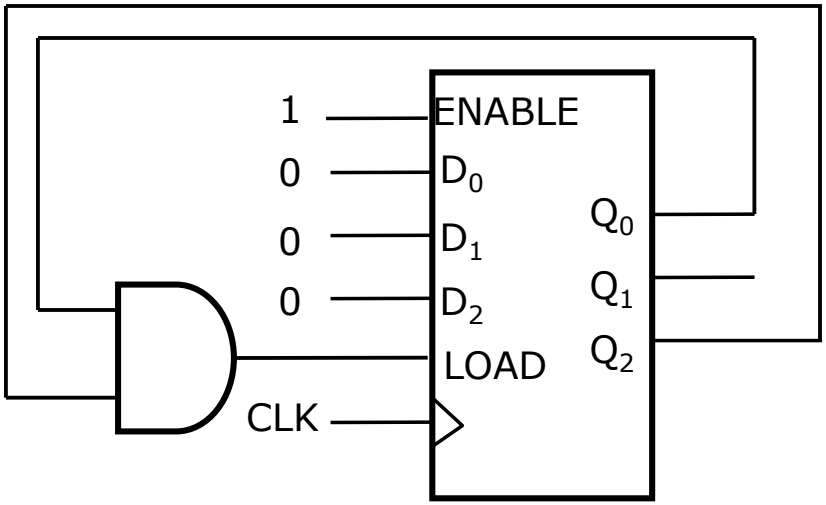


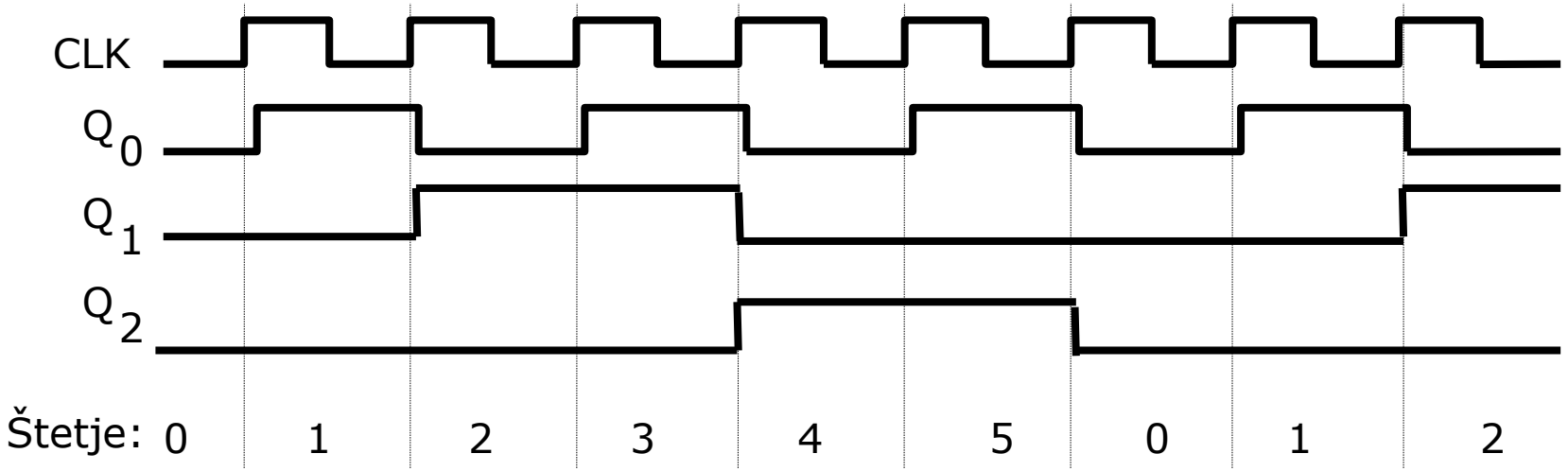
Kaskadna vezava MSI števcev



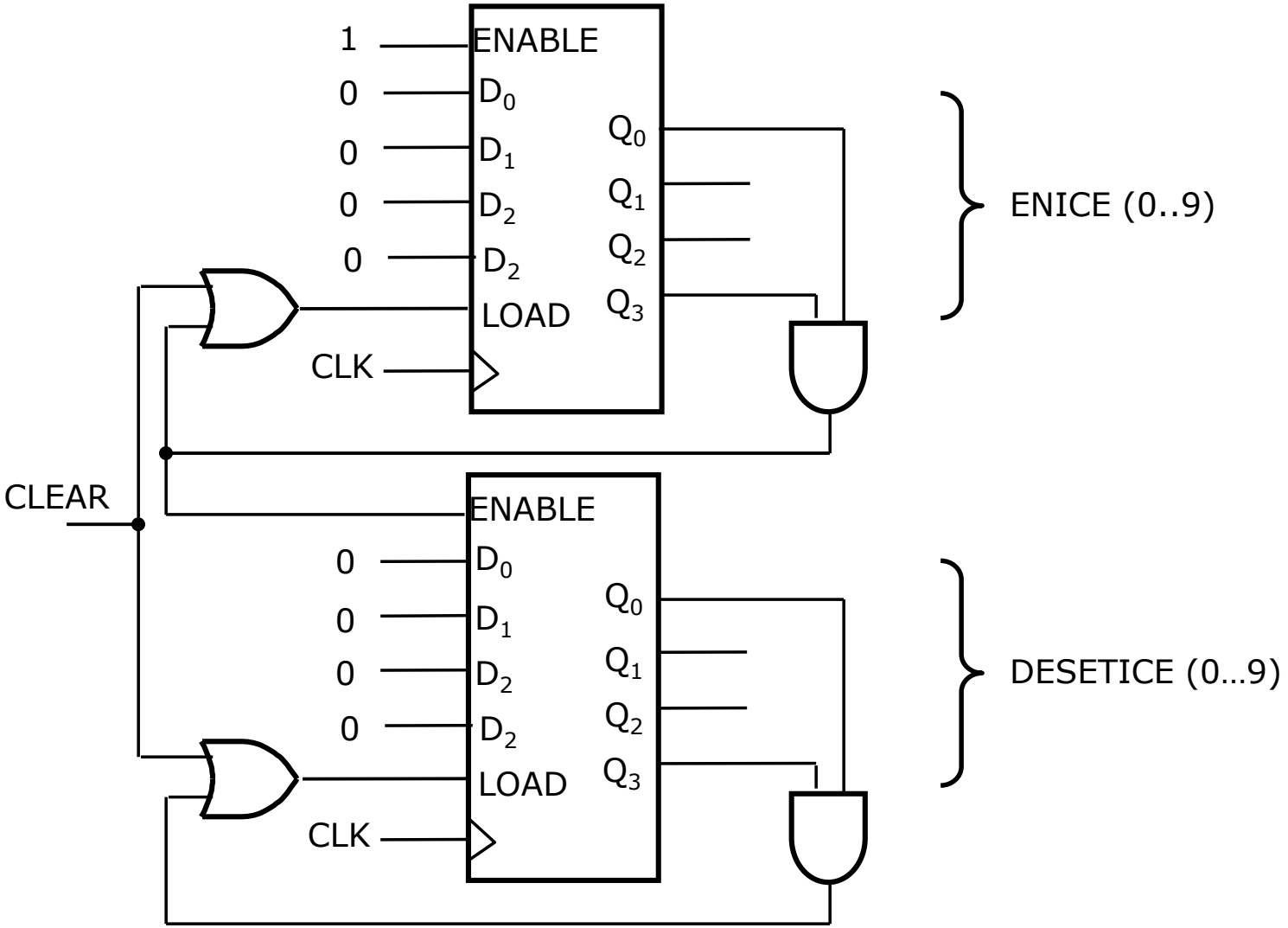


Štetje po modulu, ki ni 2^n





Drugi tipi števcev – BCD števec



74163 v VHDL

```
entity counter163 is
    port( nLOAD, nCLEAR, ENP, ENT, clk :   in std_logic;
        D: in std_logic_vector(3 downto 0);
        Q: out std_logic_vector(3 downto 0);
        RCO :   out std_logic);
end entity counter163;
architecture arch of counter163 is
begin
process (clk, nCLEAR, nLOAD)
begin
    if (clk'EVENT AND clk = '1') then
        if ( nCLEAR = '0' ) then  Q <= "0000"; -- asinhroni reset
        elsif ( nLOAD = '0' ) then    Q <= D; -- vzporedno nalaganje
        elsif ( ENP = '1' and ENT = '1' ) then Q <= Q + 1;   -- stetje
        end if;
    end if;
   RCO <= Q[3] and Q[2] and Q[1] and Q[0] and ENT; --tvorba RCO
end process;
end architecture arch;
```

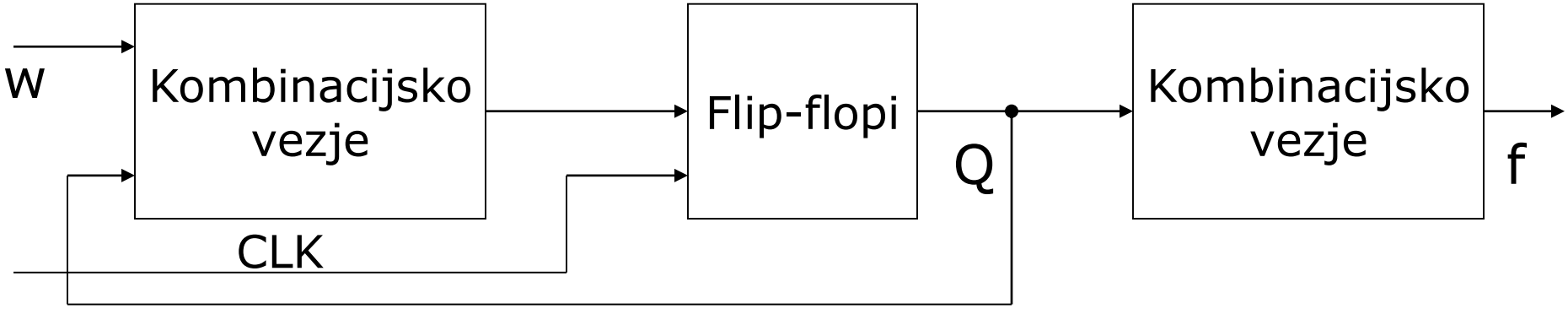
Razvoj digitalnih sistemov

Sinhronska sekvenčna vezja:
Diagrami stanj, tabele stanj

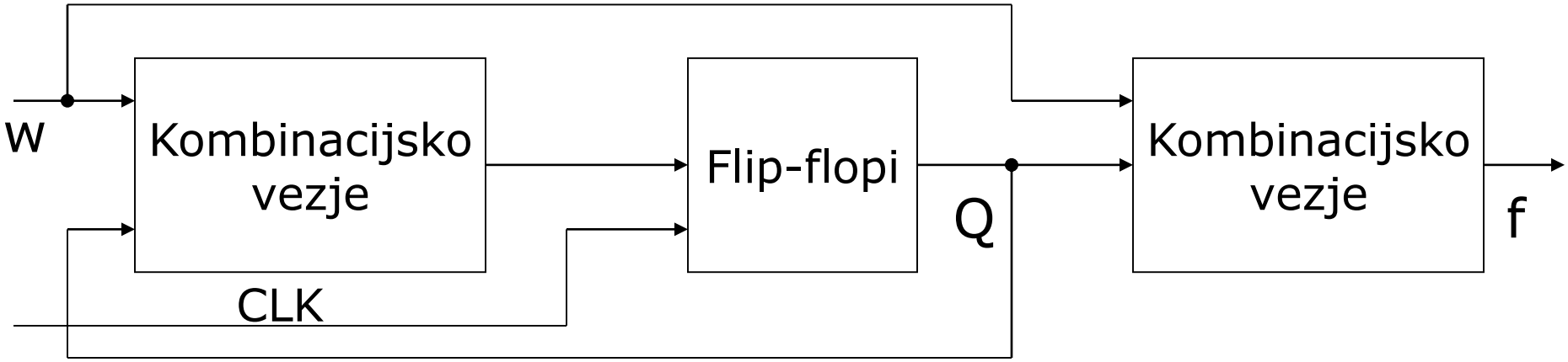
Sinhronska sekvenčna vezja

- Vezja pri katerih je signal ure uporabljen za nadzor delovanja imenujemo sinhrona sekvenčna vezja
  - Izraz aktivni rob signala ure (***active clock edge***) se nanaša na rob signala ure, ki povzroča spremembe stanja (pozitivni, negativni)
- Realizirana so s pomočjo kombinacijske logike in enega ali več FF
- Poznamo dva modela sinhronskih sekvenčnih vezij
  - **Moore-ov model**: Izhodi vezja so odvisni samo od trenutnega stanja vezja
  - **Mealy-ev model**: Izhodi vezja so odvisni od trenutnega stanja vezja in glavnih vhodov
- Sekvenčnim vezjem pravimo tudi **avtomati končnih stanj** (*finite state machines*) (FSM)

Avtomata Moore in Mealy



Moore-ov avtomat končnih stanj



Mealy-ev avtomat končnih stanj

Osnovni koraki načrtovanja

- Korake načrtovanja bomo predstavili ob zgledu detektorja vhodnega zaporedja **11**:
  - Vezje ima en vhod, ***w*** in en izhod ***z***
  - Vse spremembe v vezju se odvijajo ob pozitivnem robu signala ure
  - Izhod postane ***z***=1 če je bil vhod ***w***=1 *neposredno* dva prejšnja cikla signala ure
- Iz zahtev sledi, da ***z*** ni odvisen samo od ***w***!

CLK	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w	0	1	0	1	1	0	1	1	1	0	1
z	0	0	0	0	0	1	0	0	1	1	0

Diagram prehajanja stanj

- Za naš primer začetno stanje imenujmo A
- Dokler je w=0 bo vezje ostajalo v tem stanju in izhod z=0

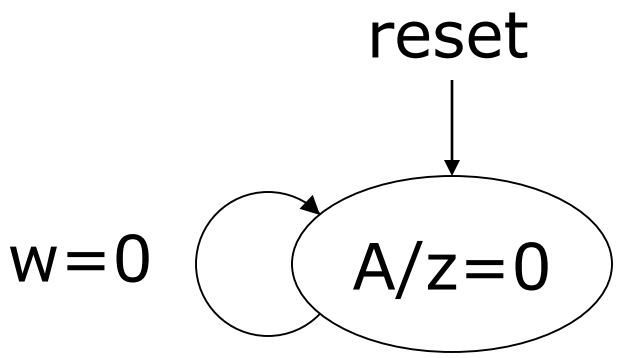


Diagram prehajanja stanj

- Ko postane w='1', si mora vezje to ‘zapomniti’. To stori tako, da preide v novo stanje (**B**).
- Tak prehod se zgodi ob naslednjem pozitivnem robu signala ure

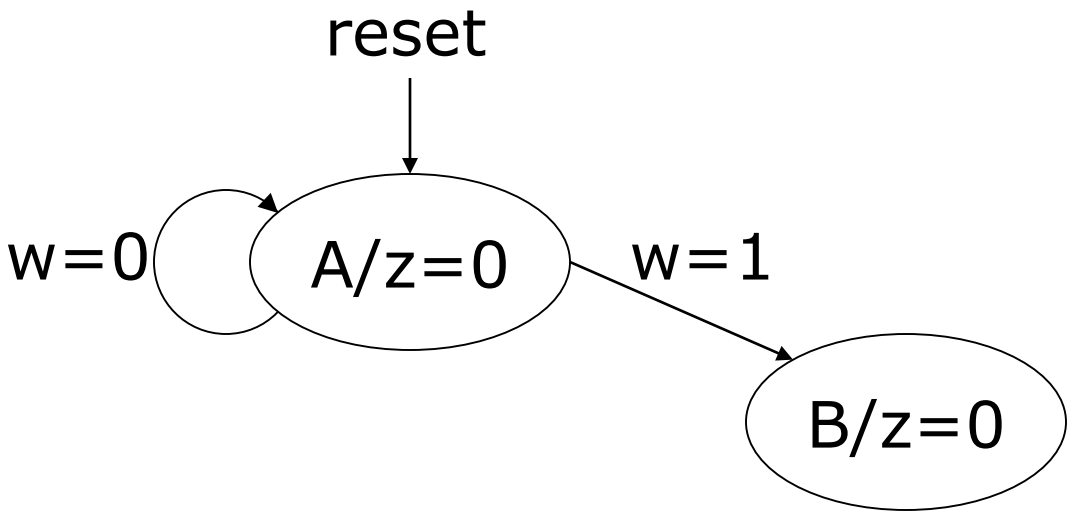
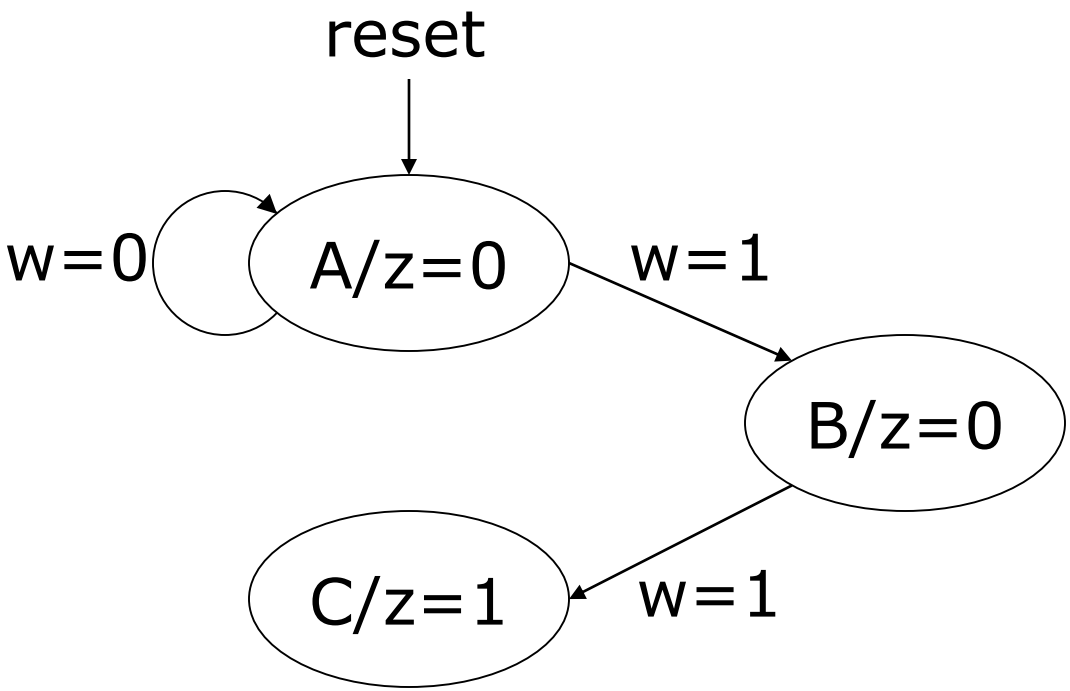
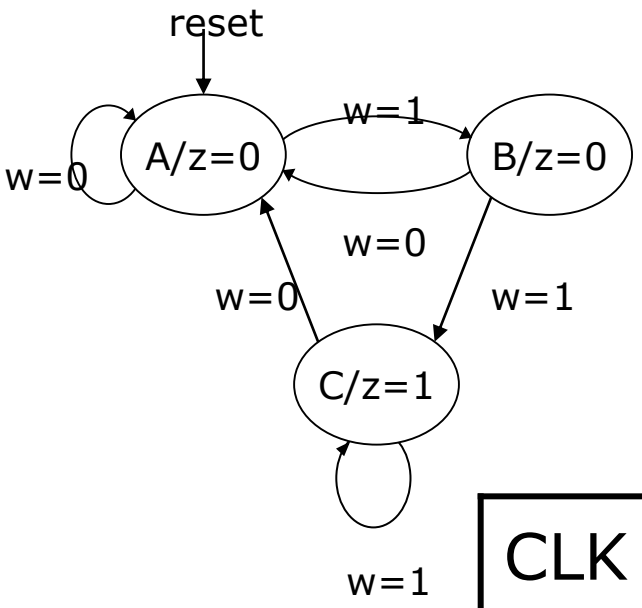


Diagram prehajanja stanj

- Ko se vezje nahaja v stanju **B** in postane w=1, si mora vezje to ponovno zapomniti, tako da preide v stanje (**C**)



Popoln diagram prehajanja stanj

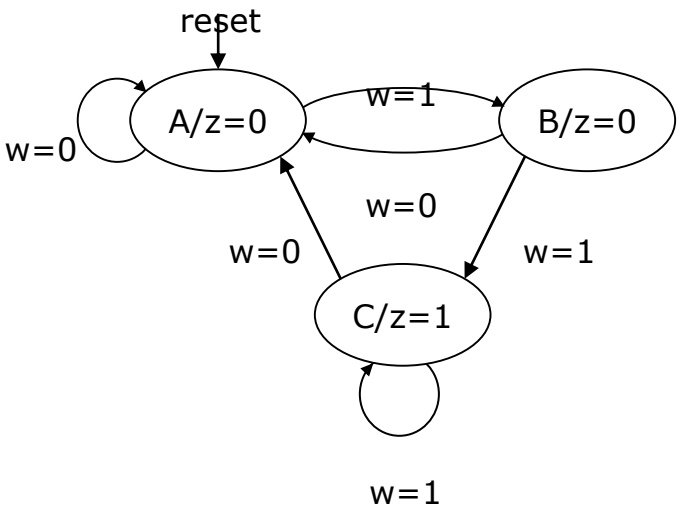


CLK	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w	0	1	0	1	1	0	1	1	1	0	1
stanje	-	A	B	A	B	C	A	B	C	C	A
z	0	0	0	0	0	1	0	0	1	1	0

Moore-ov model diagrama stanj

Tabela prehajanja stanj

- Tabela prehajanja stanj (***state table***) vsebuje
  - Vse prehode iz trenutnega stanja v naslednje za vse vrednosti vhodnih signalov
  - Izhod ***z***, ki je določen glede na trenutno stanje



Trenutno stanje	Naslednje stanje		Izhod z
	w=0	w=1	
A	A	B	0
B	A	C	0
C	A	C	1

Kodiranje stanj

- Stanja so določena s črkami (**A**, **B** in **C**)
- Vsako stanje je predstavljeno z določeno kombinacijo **spremenljivk stanja** (*state variables*)
- Vsaka spremenljivka stanja je izvedena s FF
- Realizirati moramo **3** stanja, kar lahko storimo z **dvema** spremenljivkama stanja
  - Trenutno stanje kodiramo z $Q_2(t)$ $Q_1(t)$ (spremenljivki trenutnega stanja)
  - Naslednje stanje kodiramo z $Q_2(t+1)$ $Q_1(t+1)$ (spremenljivki naslednjega stanja)

Tabela kodiranja stanj

	Trenutno stanje $Q_2(t)\ Q_1(t)$	Naslednje stanje $Q_2(t+1)\ Q_1(t+1)$ w=0	Naslednje stanje $Q_2(t+1)\ Q_1(t+1)$ w=1	Izhod z
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	XX	XX	X

Zaradi popolnosti smo dodali še stanje $Q_2(t)\ Q_1(t)=11$.

Diagrami naslednjih stanj in izhodov

- Iz tabele kodiranja stanj sestavimo Veitcheve diagrame za:
  - Izhode vezja (**z**)
  - Vhode za FF (V-diagrami naslednjih stanj)
- Tvorba diagramov naslednjih stanj je odvisna od uporabljenih tipov FF (D, T, JK)
  - D-FF je najbolj enostaven za realizacijo: Diagram naslednjih stanj lahko kar preberemo iz tabele kodiranja stanj, ker je: $Q(t+1)=D$
  - T in JK realizacijo bomo obdelali pozneje

Diagrami naslednjih stanj

	Trenutno stanje	Naslednje stanje				Izhod
		w=0		w=1		z
	$Q_2(t)$ $Q_1(t)$	**$Q_2(t+1)$**	**$Q_1(t+1)$**	**$Q_2(t+1)$**	**$Q_1(t+1)$**	
A	00	0	0	0	1	0
B	01	0	0	1	0	0
C	10	0	0	1	0	1
	11	X	X	X	X	X

$Q_2(t+1)=w\cdot(Q_1(t)+Q_2(t))$

$Q_2(t)$ \ $Q_1(t)$				
	X	X	1	0
	0	1	0	0

w

$Q_1(t+1)=w\cdot Q_1(t)'\cdot Q_2(t)'$

$Q_2(t)$ \ $Q_1(t)$				
	X	X	0	0
	0	0	1	0

w

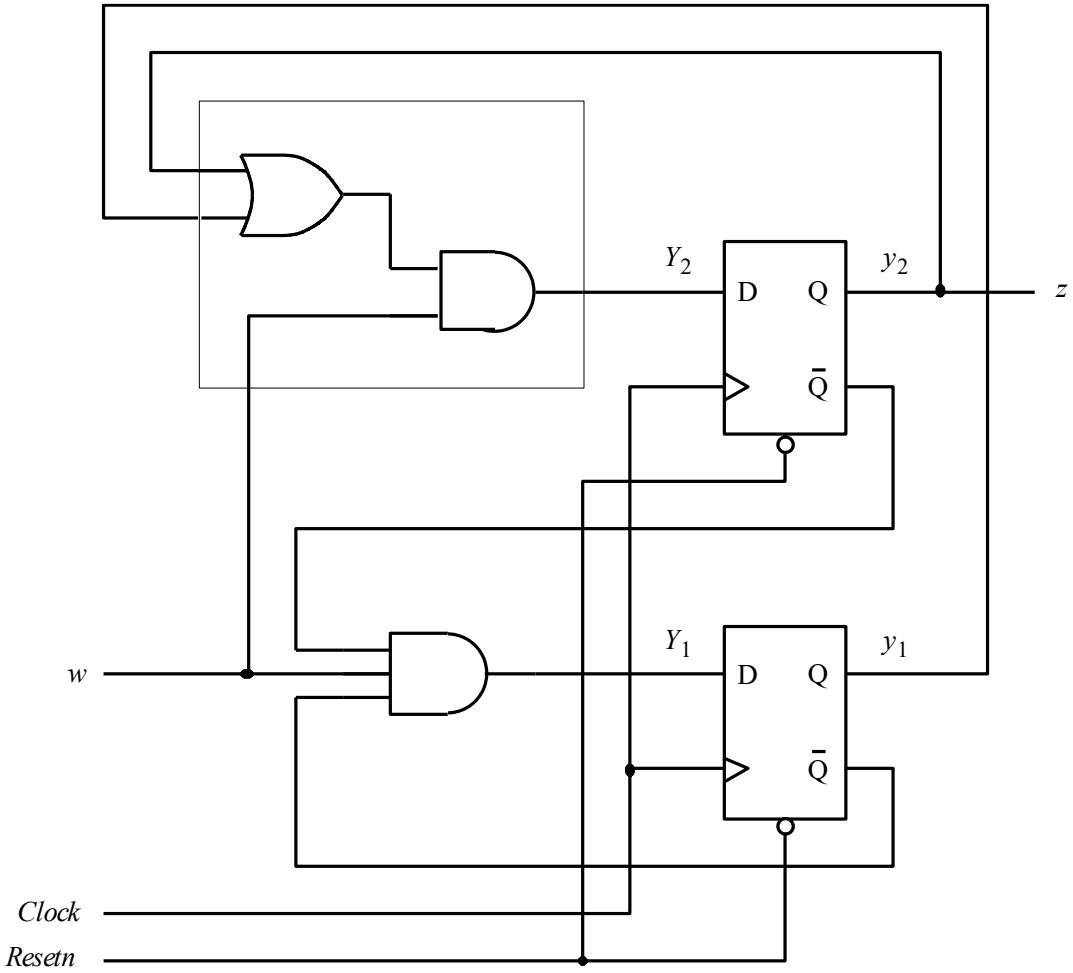
Diagram izhoda

	Trenutno stanje	Naslednje stanje		Izhod
		w=0	w=1	z
	$Q_2(t)\ Q_1(t)$	**$Q_2(t+1)\ Q_1(t+1)$**	**$Q_2(t+1)\ Q_1(t+1)$**	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	XX	XX	X

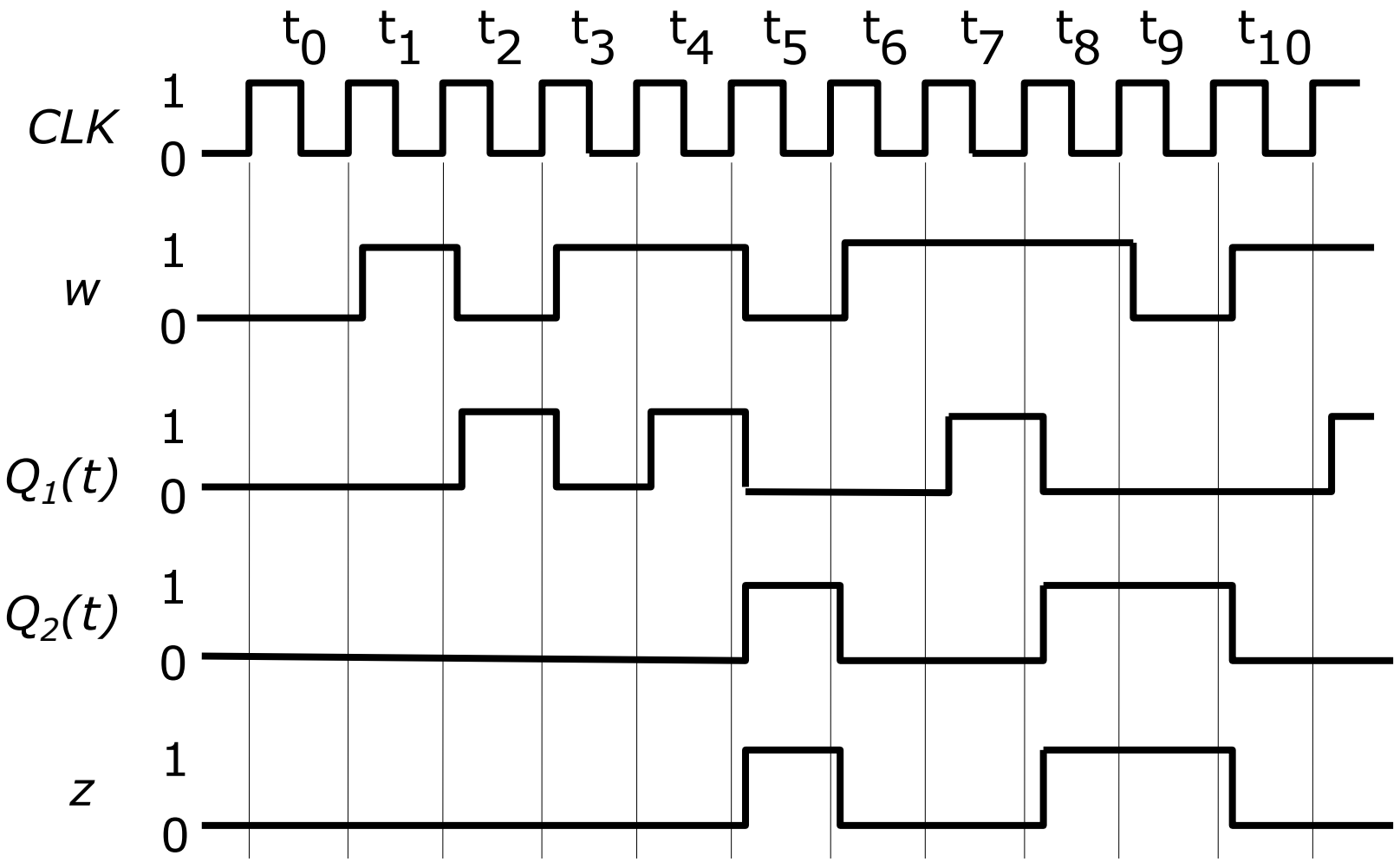
$Q_2(t)$ \ $Q_1(t)$	X	1
0	0	0

$$z=Q_2(t)$$

Vezje avtomata



Časovni diagram



Razvoj digitalnih sistemov

Sinhrona sekvenčna vezja:

Realizacije z uporabo
T in JK FF

Primer načrtovanja števca

- Načrtajte 2-bitni števec, ki šteje
  - V sekvenci 0,1,2,3,0,… če je kontrolni signal U=1, oziroma
  - V sekvenci 0,3,2,1,0,… če je kontrolni signal U=0
- To je števec po modulu 4 gor/dol
  - Vsebuje vhod U ki nadzira smer štetja (U=up) gor
  - RESET vhod s katerim ponastavimo števec na 0
  - Dva izhoda (Z_1Z_0) ki predstavljata vsebino štetja (0-3)
  - Števec šteje ob pozitivnih spremembah skupnega signala ure
- Načrtajte števec kot sinhronski sekvenčni avtomat z uporabo T in JK FF

Diagram prehajanja stanj števca

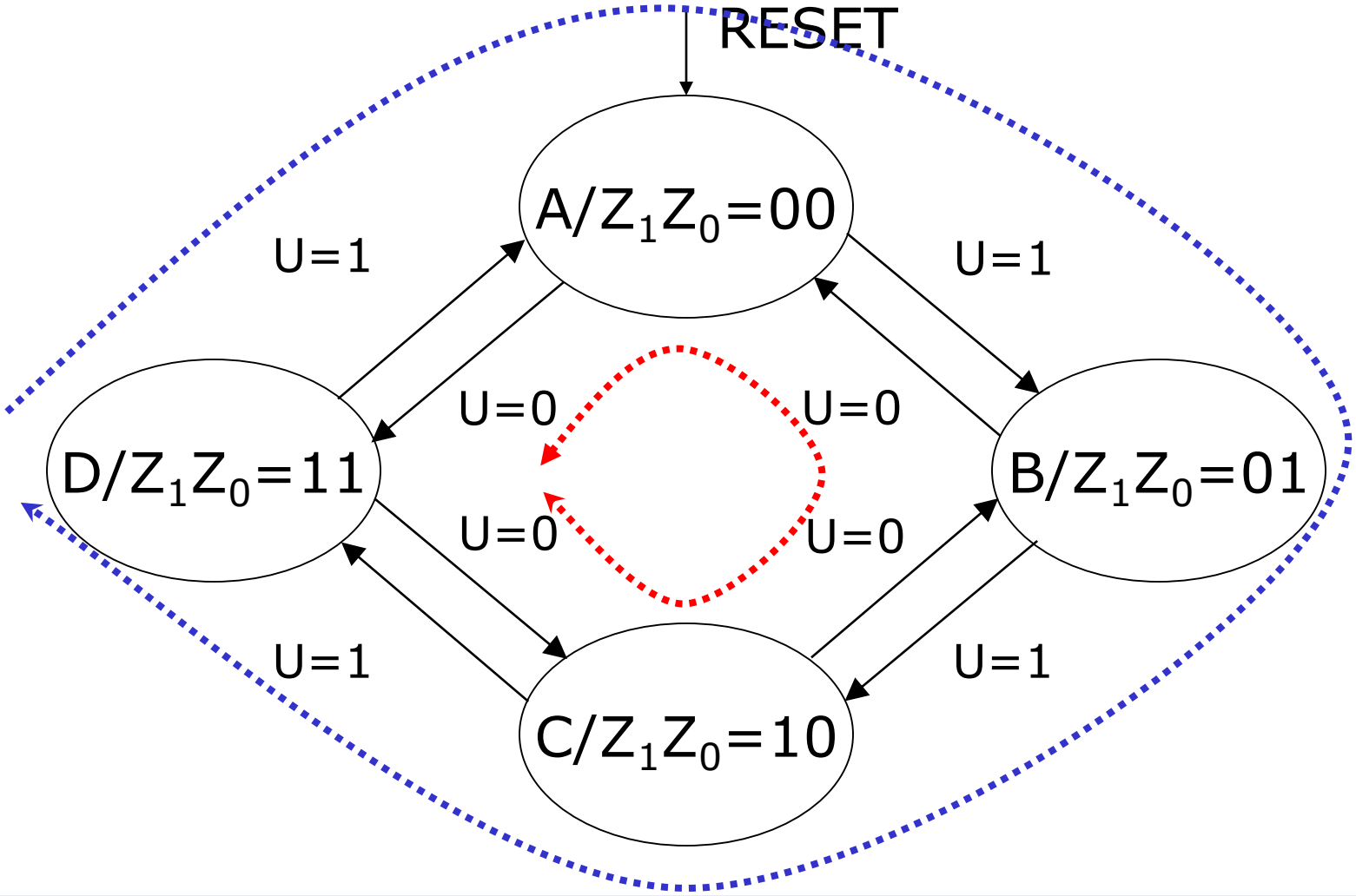


Tabela prehajanja stanj števca

Trenutno stanje	Naslednje stanje		Izhod Z_2Z_1
	U=0	U=1	
A	D	B	00
B	A	C	01
C	B	D	10
D	C	A	11

Kodiranje stanj števca

	Trenutno stanje **(t)** Q_2Q_1	Naslednje stanje **(t+1)**		Izhod Z_2Z_1
		U=0	U=1	
		Q_2Q_1	**Q_2Q_1**	
A	00	11	01	00
B	01	00	10	01
C	10	01	11	10
D	11	10	00	11

Diagrami naslednjih stanj in izhodov

	Trenutno stanje	Naslednje stanje				Izhod	
		U=0		U=1		Z_2	Z_1
	Q_2Q_1	Q_2	Q_1	Q_2	Q_1		
A	00	1	1	0	1	0	0
B	01	0	0	1	0	0	1
C	10	0	1	1	1	1	0
D	11	1	0	0	0	1	1

$Q_1(t+1)=Q_1(t)'$

Q_2 \ Q_1	0	0	1	1
0	0	0	1	1
0	0	0	1	1

U

$Q_2(t+1)=(Q_2(t)\oplus Q_1(t)\oplus U)'$

Q_2 \ Q_1	0	0	1	1
0	1	0	1	0
0	0	1	0	1

U

$Z_1=Q_1(t)$

$Z_2=Q_2(t)$

Načrtovanje s T oz. JK-FF

- Za T oz. JK FF moramo njihove vhode določiti *posebej.*
- Začnemo s konstrukcijo tabele prehajanja stanj za uporabljen tip FF
  - Tabela podaja število vhodov za dano spremembo stanja
- Tabelo prehajanja stanj uporabljamo skupaj s tabelo kodiranja stanj, tako da naredimo **vzbujalno tabelo** (*excitation table*)
  - Vzbujalna tabela vsebuje vse potrebne vhode FF, ki jih moramo 'vzbuditi' da bi se zgodil prehod v naslednje stanje

Tabela prehajanja stanj

J	K	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK Tabela prehajanja stanj

T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

T Tabela prehajanja stanj

Q^+ je naslednje stanje

Izvedba s T FF

Q	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Uporabljamo tabelo prehajanja stanj za tvorbo funkcij vhodov FF v tabeli kodiranja stanj

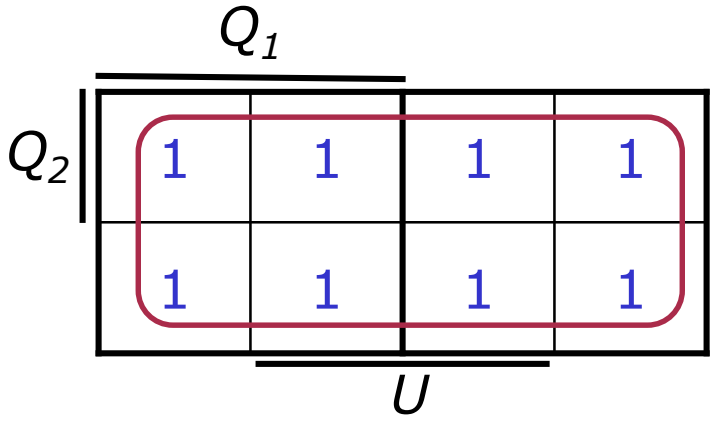
Vzbujalna tabela

Trenutno stanje Q_2Q_1	Naslednje stanje				Izhod Z_2Z_1
	U=0		U=1		
	Q_2Q_1	$\mathbf{T_2T_1}$	Q_2Q_1	$\mathbf{T_2T_1}$	
00	11	11	01	01	00
01	00	01	10	11	01
10	01	11	11	01	10
11	10	01	00	11	11

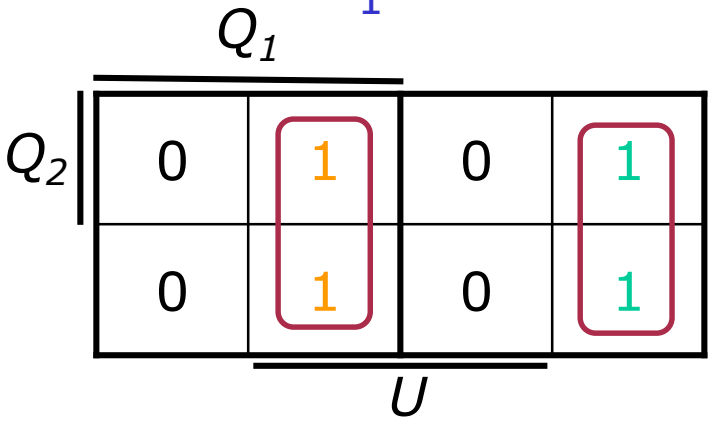
Vzbujalna tabela in V-diagrami

Trenutno stanje Q_2Q_1	Naslednje stanje		Izhod Z_2 Z_1	
	U=0	U=1		
	T_2T_1	T_2T_1		
00	11	01	0	0
01	01	11	0	1
10	11	01	1	0
11	01	11	1	1

$Z_2=Q_2 \quad Z_1=Q_1$

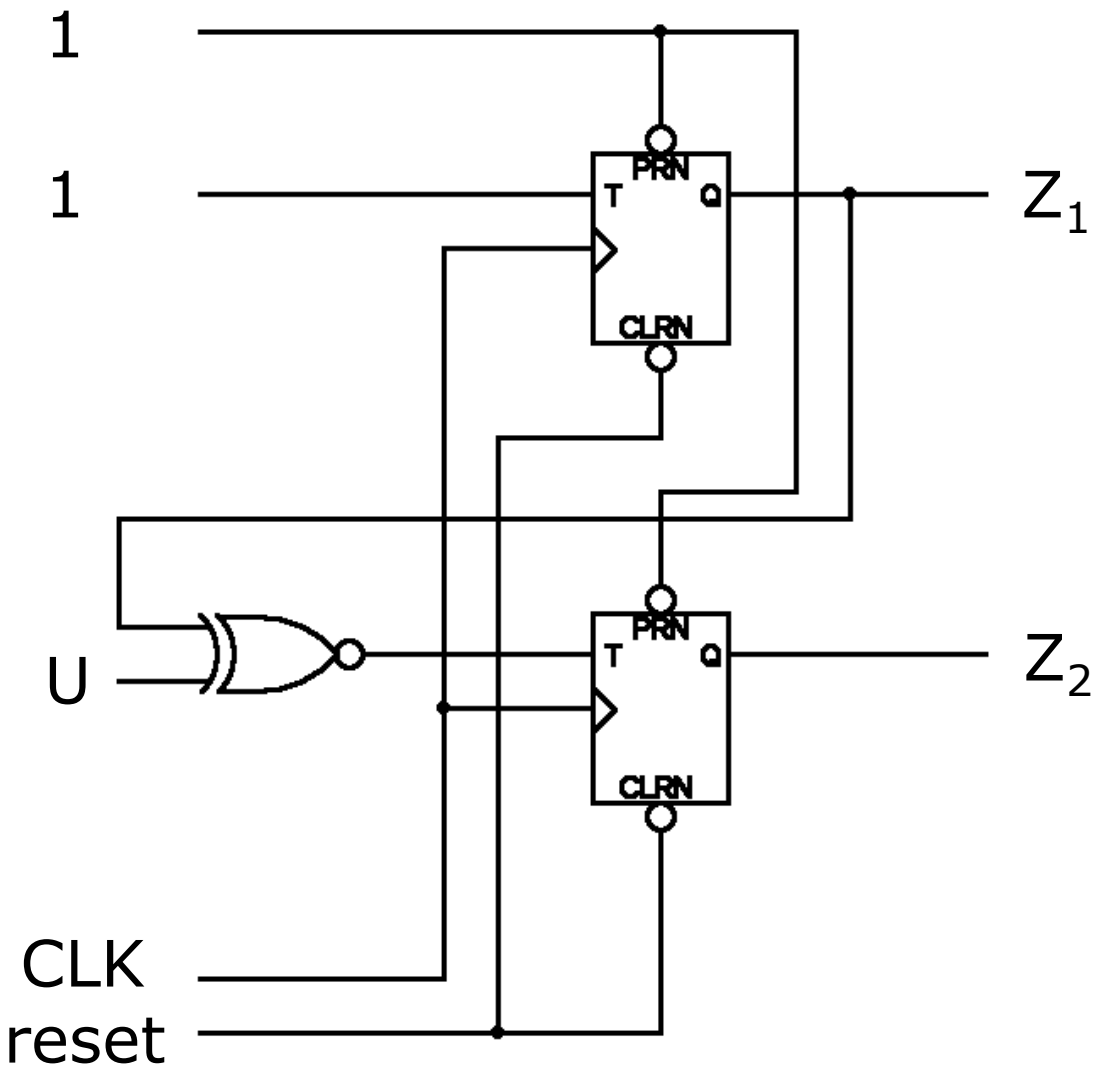


$T_1=1$



$T_2=Q_1U+Q_1'U'=(Q_1 \oplus U)'$

Realizacija vezja s T-FF



Realizacija vezja z JK-FF

- Uporabljamo tabelo prehajanja stanj za tvorbo funkcij vhodov FF v tabeli kodiranja stanj:
  Vendar tokrat za vsak vhod FF posebej (J in K)

Trenutno stanje Q_2Q_1	Naslednje stanje		Izhod Z_2Z_1
	U=0	U=1	
	Q_2Q_1	Q_2Q_1	
00	11	01	00
01	00	10	01
10	01	11	10
11	10	00	11

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK tabela prehajanja stanj

Realizacija vezja z JK-FF

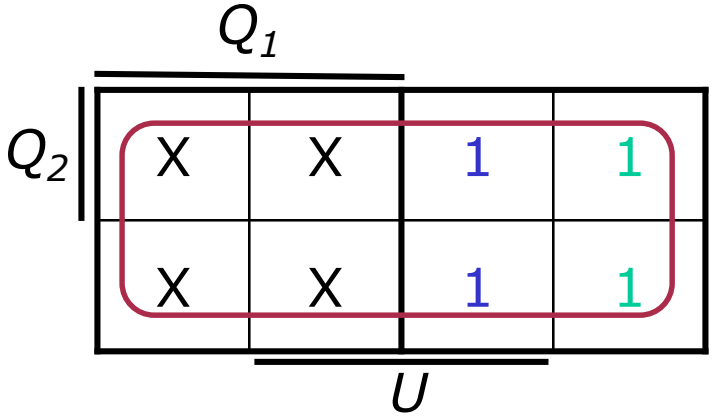
Q	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK tabela prehajanja stanj

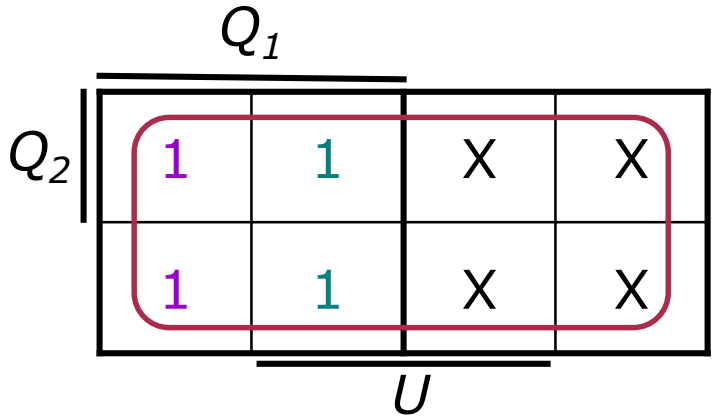
Trenutno stanje Q_2Q_1	Naslednje stanje						Izhod Z_2Z_1
	U=0			U=1			
	Q_2Q_1	J_2K_2	J_1K_1	Q_2Q_1	J_2K_2	J_1K_1	
00	11	1X	1X	01	0X	1X	00
01	00	0X	X1	10	1X	X1	01
10	01	X1	1X	11	X0	1X	10
11	10	X0	X1	00	X1	X1	11

Vzbujalna tabela in V-diagrami

Trenutno stanje Q_2Q_1	Naslednje stanje						Izhod Z_2Z_1
	U=0			U=1			
	Q_2Q_1	J_2K_2	J_1K_1	Q_2Q_1	J_2K_2	J_1K_1	
00	11	1X	1X	01	0X	1X	00
01	00	0X	X1	10	1X	X1	01
10	01	X1	1X	11	X0	1X	10
11	10	X0	X1	00	X1	X1	11



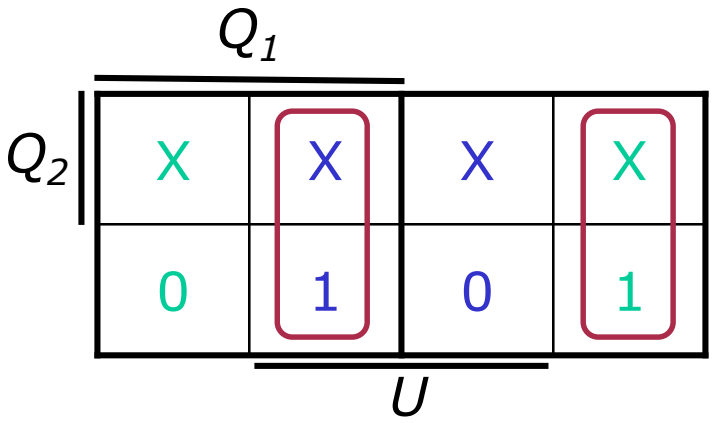
$J_1 = 1$



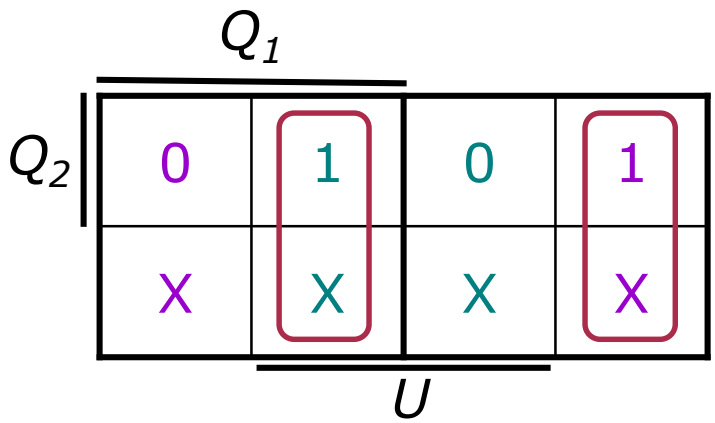
$K_1 = 1$

Vzbujalna tabela in V-diagrami

Trenutno stanje Q_2Q_1	Naslednje stanje						Izhod Z_2Z_1
	U=0			U=1			
	Q_2Q_1	J_2K_2	J_1K_1	Q_2Q_1	J_2K_2	J_1K_1	
00	11	1X	1X	01	0X	1X	00
01	00	0X	X1	10	1X	X1	01
10	01	X1	1X	11	X0	1X	10
11	10	X0	X1	00	X1	X1	11

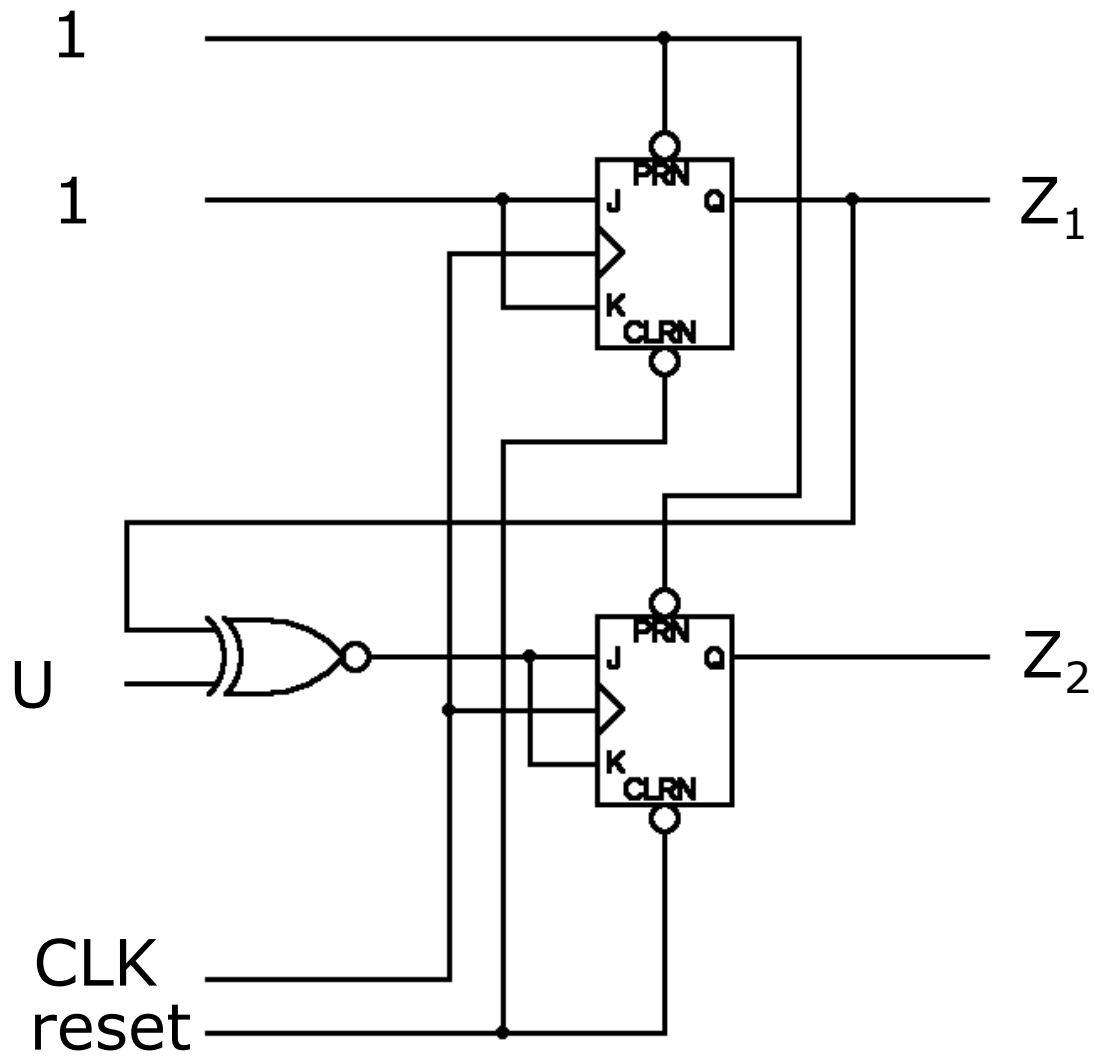


$J_2 = (Q_1 \oplus U)'$



$K_2 = (Q_1 \oplus U)'$

Vezje z JK-FF



Razvoj digitalnih sistemov

Sinhronska sekvenčna vezja:

Problem kodiranja stanj,
Mealy-evi avtomati

Problem kodiranja stanj

- V splošnem so vezja veliko večja kot predstavljeni zgled in drugačno kodiranje stanj ima lahko velik vpliv na ceno končne realizacije vezja
- Običajno je nepraktično iskati optimalno določitev kodiranja stanj, saj imamo opravka z velikim številom stanj
- CAD orodja izvajajo kodiranje stanj s hevrističnimi pristopi

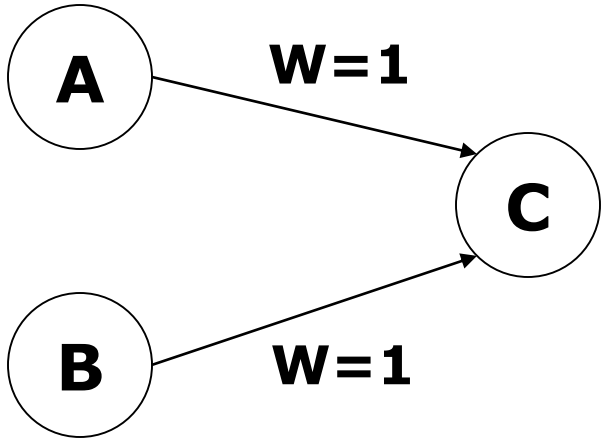
Napotki za kodiranje stanj

- NE zagotavljajo minimalne rešitve
- Stanja kodiramo kot sosedna, če se njihove kode razlikujejo samo v eni spremenljivki stanja

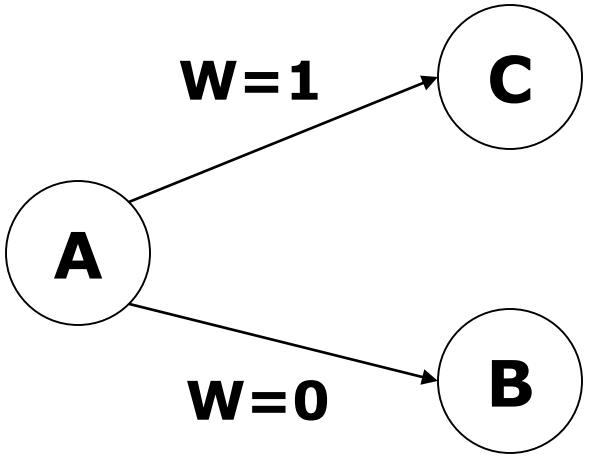
1. Stanja, ki imajo **enako naslednje stanje za dani vhod**, kodiramo kot sosedna
2. **Naslednja stanja, ki izvirajo iz istega trenutnega stanja**, kodiramo kot sosedna
3. Stanja, ki so naslednja stanja istega stanja kodiramo kot sosedna
4. Stanja, ki imajo enak izhod za dani vhod kodiramo kot sosedna (skupine ‘1’ v V-diagramu izhoda)

Navodila za kodiranje stanj

1. Stanja, ki imajo enako naslednje stanje za dani vhod, kodiramo kot sosedna



2. Naslednja stanja, ki izvirajo iz istega trenutnega stanja, kodiramo kot sosedna



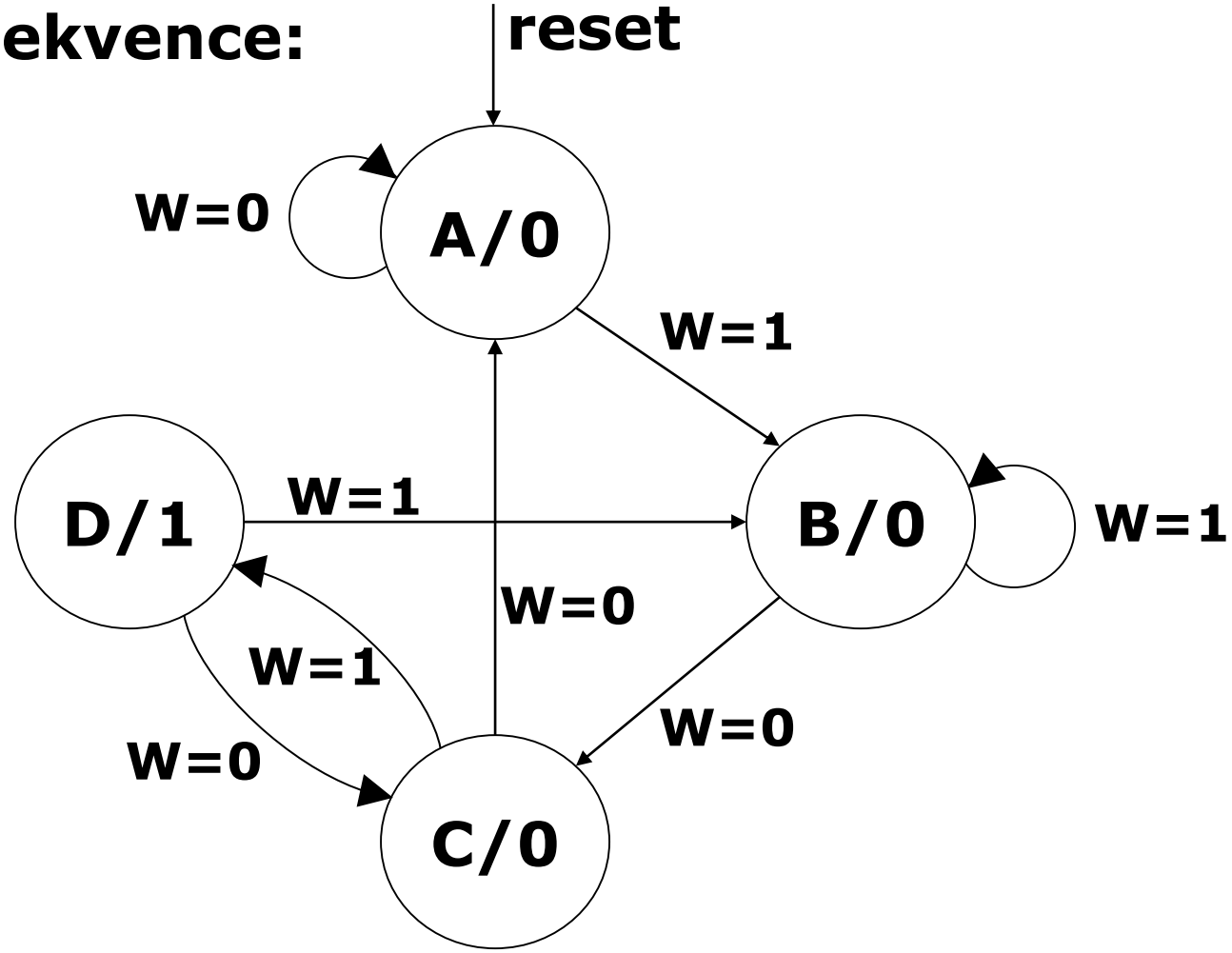
Navodila za kodiranje stanj

Narišite tabelo prehajanja stanj

1. Izhodiščnemu stanju postavite številko ‘0’ in ga dajte v celico 0
   (začetno stanje ima izhode vseh FF 0)
2. Upoštevajte navodili o izvoru in ponoru stanj (prejšnja prosojnica) in stanja kodirajte
3. Če naletite na skupino 3 ali 4 sosednih stanj, jih postavite skupaj v 4 sosedna polja v tabeli stanj
4. Prejšnji napotek je manj pomemben kot prvi in drugi, razen če vezje nima več izhodov

Zgled:
Moore-ov diagram prehajanja stanj

Detektor 101 sekvence:



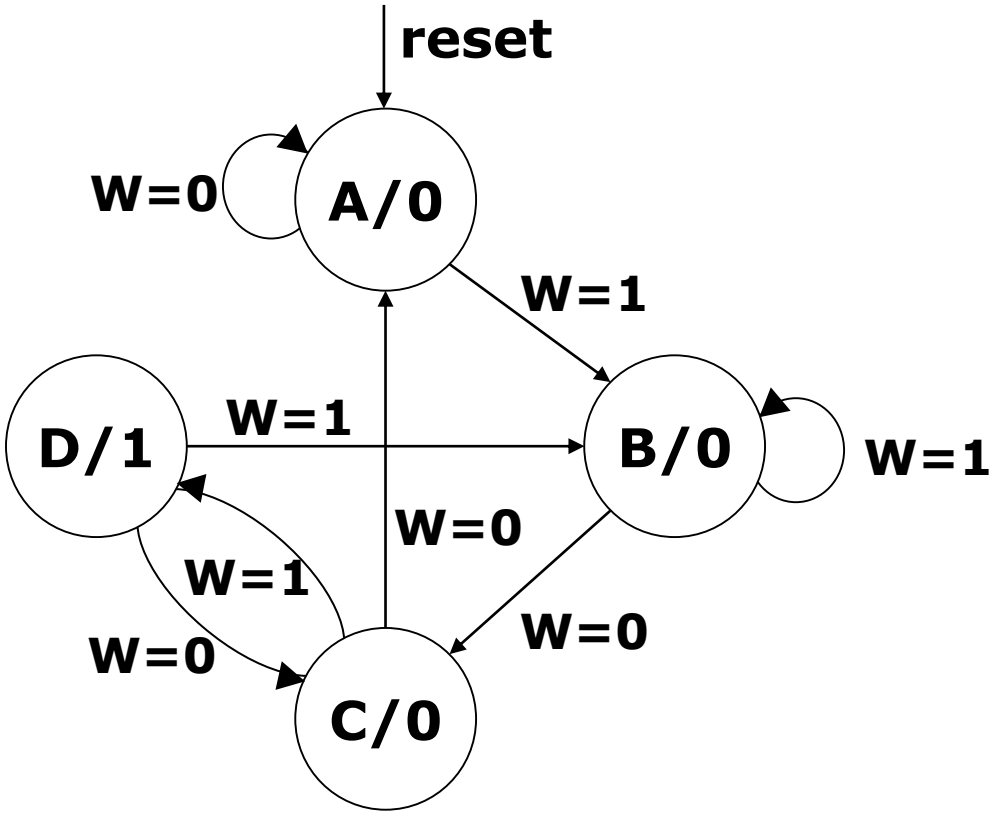
Primer kodiranja stanj

- Dve spremenljivki stanj Q_1Q_0
- Če sledimo napotke
- A=00

1. {A,D}, {B,D}, {A,C}
2. {A,D}, {B,C}
3. {A,B,C}

Q_1 \ Q_0		
	B	D
	C	A

A=00
B=11
C=01
D=10

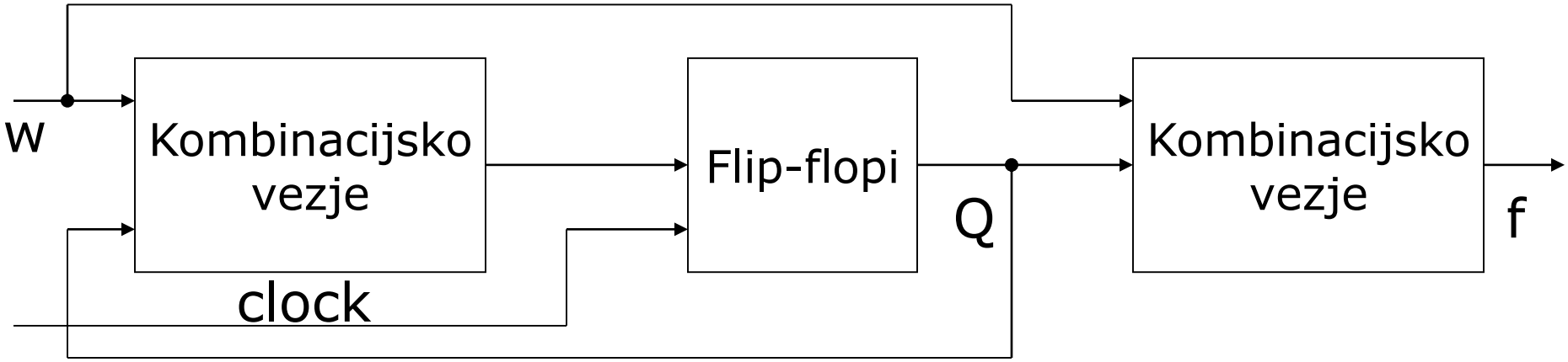


Kodiranje stanj "ena naenkrat"

- Ena možnost kodiranja stanj je, da izberemo toliko spremenljivk stanja kot je stanj v sekvenčnem vezju
- Za vsako stanje so vse spremenljivke stanja razen ene enake 0
- Spremenljivka, katere vrednost je 1 je aktivna
  - ***Od tod ime "ena naenkrat"***
- Poveča število FF, ki se rabijo za realizacijo, vendar zagotavlja enostavnejše izhodne izraze
  - Enostavnejši izhodni izrazi omogočajo hitrejše delovanje vezja, saj je zakasnitev širjenja manjša od izhodov FF do končnih izhodov sekvenčnega vezja

Mealy-ev avtomat

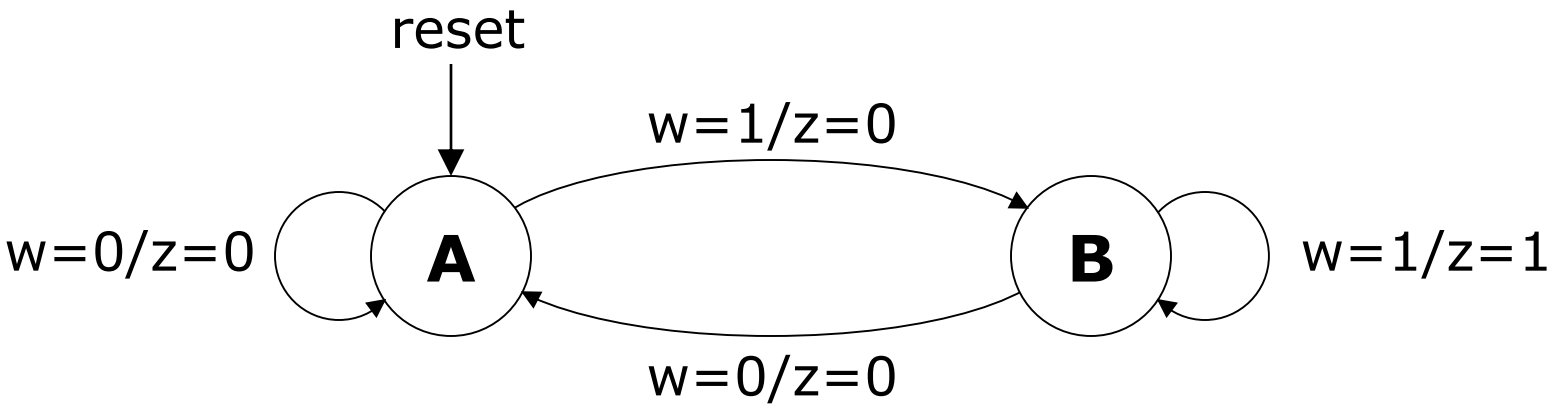
- **Mealy-ev pristop**: izhodi vezja so odvisni od trenutnega stanja vezja in glavnih vhodov vezja, kar omogoča večjo prilagodljivost pri načrtovanju sekvenčnih vezij
- Večja prilagodljivost pomeni enostavnejša vezja



Mealy-ev avtomat

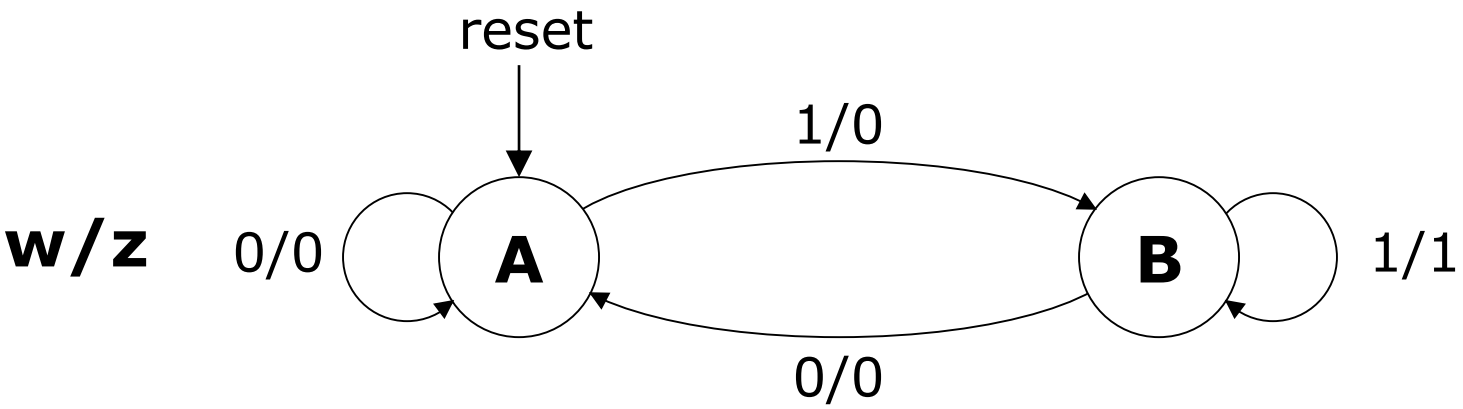
Mealy-ev diagram stanj

- V Mealy-evem pristopu izhodi niso več povezani z določenim stanjem
  - Izhodi so povezani s prehodi med stanji.
  - Posledica: Stanja nimajo več samo ene funkcije.
- Tipični Mealy-ev diagram stanj
  - Detektor w=11 sekvence



Mealy-eva tabela stanj

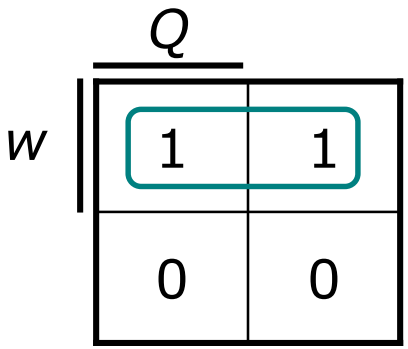
- Mealy-eva tabela stanj se razlikuje od Moore-ove samo s stališča izhodov.



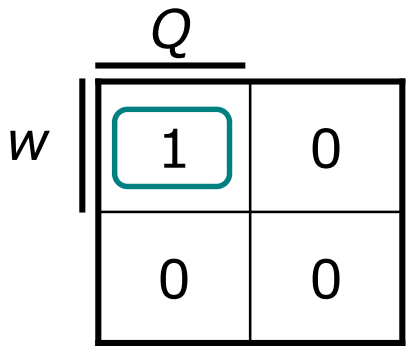
Trenutno stanje	Naslednje stanje		Izhod *z*	
	w = 0	*w* = 1	*w* = 0	*w* = 1
A	A	B	0	0
B	A	B	0	1

Kodiranje stanj

	Trenutno stanje	Naslednje stanje		Izhod	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	Q	Q	Q	z	z
A	0	0	1	0	0
B	1	0	1	0	1



$Q(t+1)=w$



$z=w \cdot Q(t)$

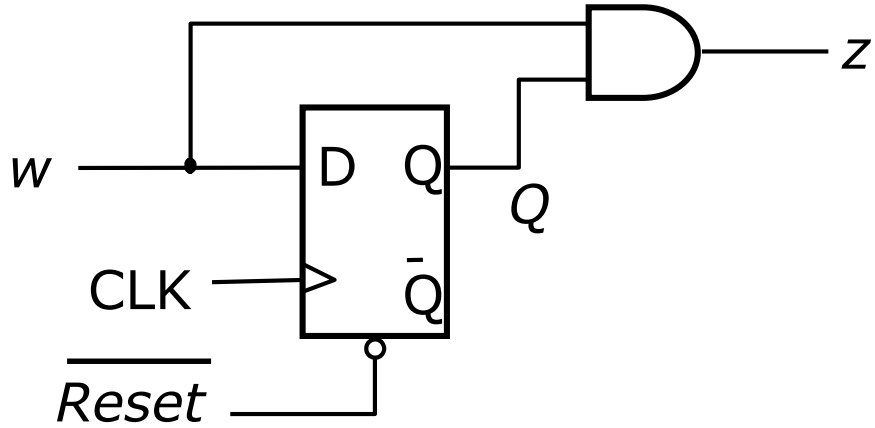


Diagram stanj za detektor zaporedja w=101 kot Mealy-ev avtomat

CLK	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
w	0	0	1	1	0	0	1	1	0	1	1	1	0	1	1	1
z	-	0	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	0
z	A	A	B	B	C	A	B	B	C	A	B	B	C	A	B	B

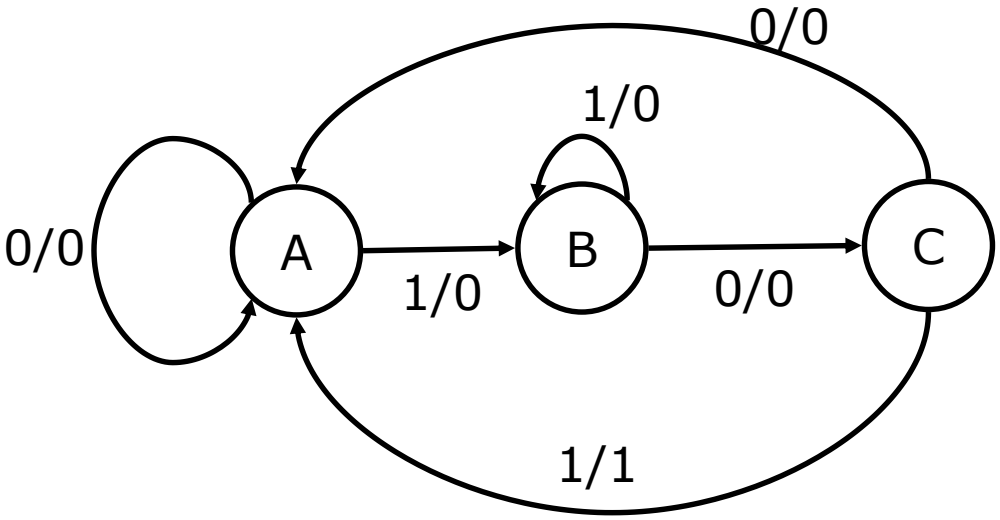
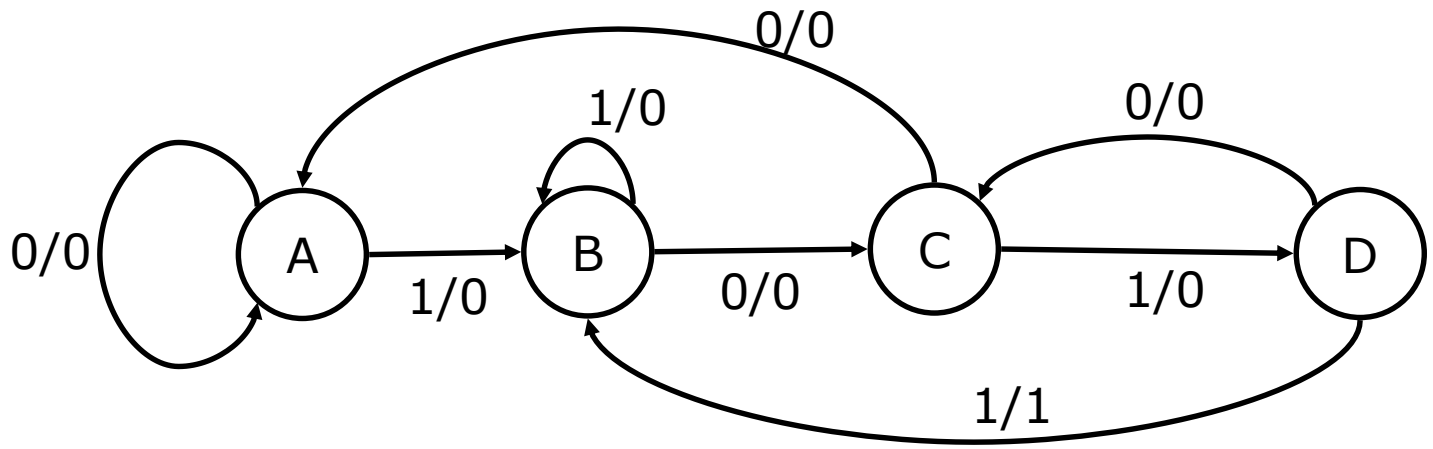
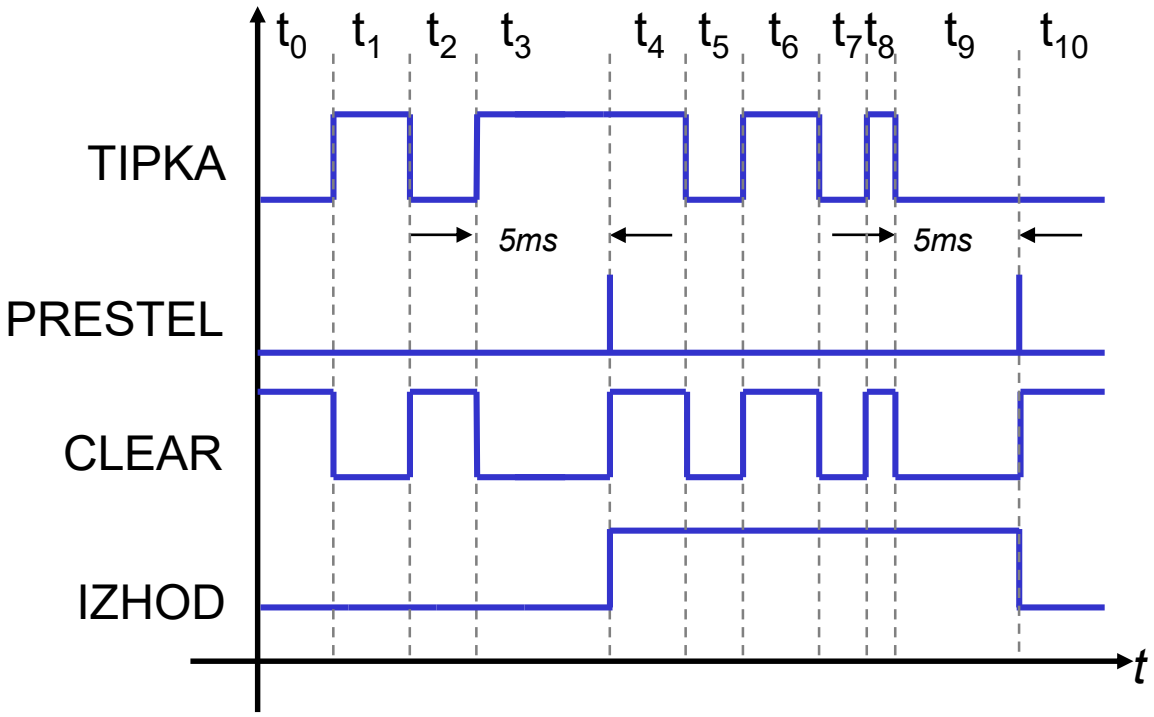


Diagram stanj za detektor zaporedja w=1011 kot Mealy-ev avtomat

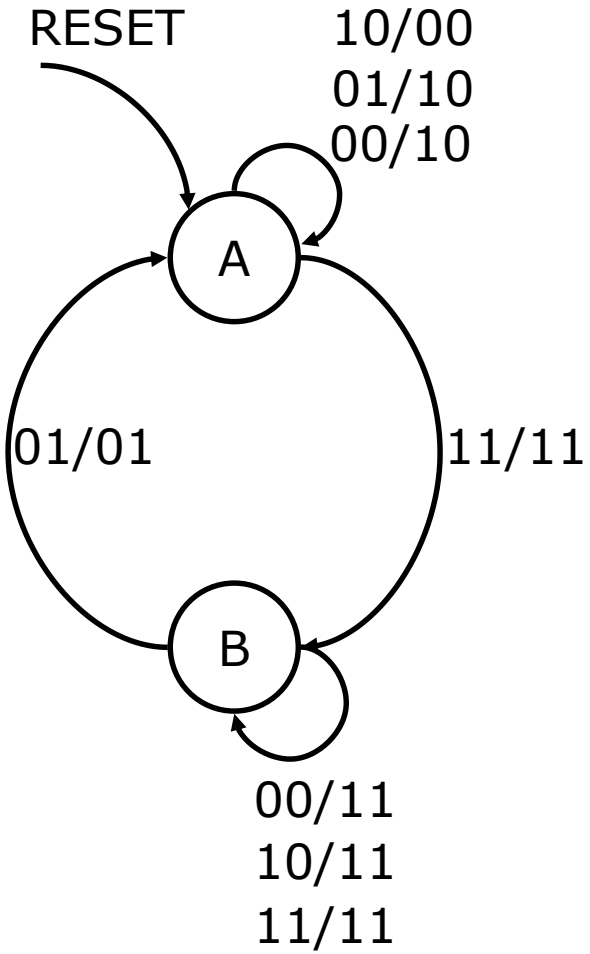
CLK	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}
w	0	1	0	1	1	0	1	1	0	1	0	1	1	1	1	1
z	-	0	0	0	**1**	0	0	**1**	0	0	0	0	**1**	0	0	0
z	A	B	C	D	B	C	D	B	C	D	C	D	A	B	B	B



Odskakovanja tipk kot Mealy-ev avtomat



Tipka	Prestel / Clear	Izhod
0	0 / 0	0



Pretvorbe avtomatov

- Dva avtomata sta **ekvivalentna**, če se pri enakem vhodnem nizu odzivata z enakim izhodnim nizom.
- Vsak FSM lahko pretvorimo iz tipa Moore v tip Mealy in obratno.

Moore➔Mealy

Število stanj Mealy-evega avtomata ostane nespremenjeno!

Izhod v Mealy-evem avtomatu določimo tako, da v naslednjih stanjih dodamo izhod stanja v Moore-ovem avtomatu.

Trenutno stanje	Naslednje stanje					
	TIPKA/PRESTEL				IZHODA	
	00	01	10	11	IZHOD/CLEAR	
S_0	S_0	S_0	S_1	S_1	0	1
S_1	S_0	S_0	S_1	S_2	0	0
S_2	S_3	S_3	S_2	S_2	1	1
S_3	S_3	S_0	S_2	S_2	1	0

Trenutno stanje	Naslednje stanje			
	TIPKA/PRESTEL			
	00	01	10	11
S_0	S_0/01	S_0/01	S_1/00	S_1/00
S_1	S_0/01	S_0/01	S_1/00	S_2/11
S_2	S_3/10	S_3/10	S_2/11	S_2/11
S_3	S_3/10	S_0/01	S_2/11	S_2/11

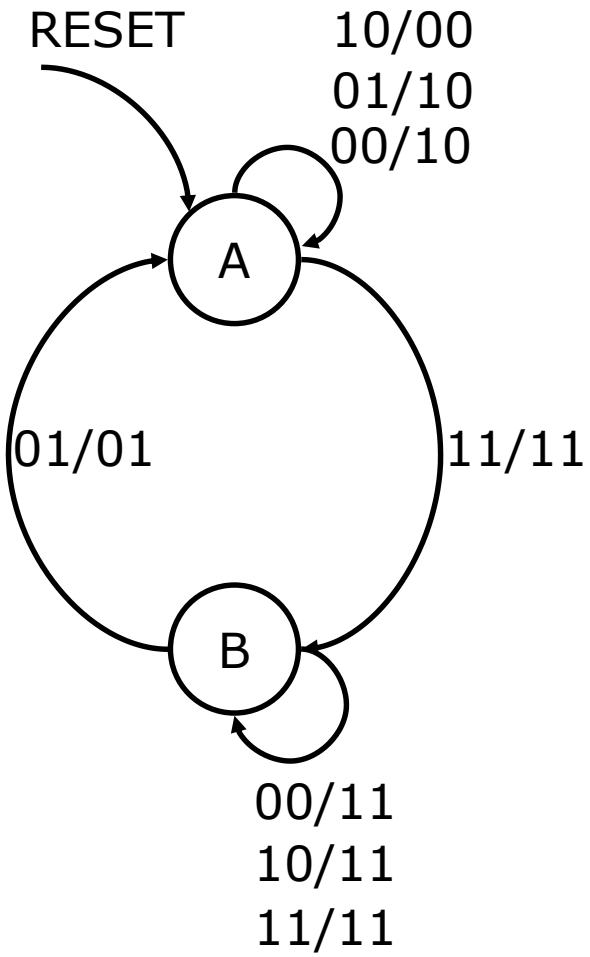
Mealy➔Moore

Število stanj v Moore-ovega avtomata, ki izvirajo iz enega stanja Mealy-evega avtomata je odvisno od števila različnih izhodov Mealy-evega avtomata!

	x=0	x=1
S_1	$S_2/1$	$S_3/0$
S_2	$S_5/1$	$S_3/1$
S_3	$S_1/0$	$S_5/0$
S_4	$S_2/1$	$S_4/1$
S_5	$S_3/0$	$S_2/1$

	x=0	x=1	izhod
S_{10}	S_{21}	S_{30}	0
S_{21}	S_{51}	S_{31}	1
S_{30}	S_{10}	S_{50}	0
S_{31}	S_{10}	S_{50}	1
S_{41}	S_{21}	S_{41}	1
S_{50}	S_{30}	S_{21}	0
S_{51}	S_{30}	S_{21}	1

Mealy➔Moore



Trenutno stanje	Naslednje stanje			
	Tipka Prestel			
	00	01	10	11
A	A/10	A/10	A/00	B/11
B	B/11	A/01	B/11	B/11

Trenutno stanje	Naslednje stanje				
	Tipka Prestel				clear/izhod
	00	01	10	11	
A_{00}	A_{10}	A_{10}	A_{00}	B_{11}	00
A_{01}	A_{10}	A_{10}	A_{00}	B_{11}	01
A_{10}	A_{10}	A_{10}	A_{00}	B_{11}	10
B_{11}	B_{11}	A_{01}	B_{11}	B_{11}	11

Razvoj digitalnih sistemov

Minimizacija stanj

Minimizacija stanj

- Za enostavne FSM minimalno število stanj razberemo kar iz njihovega diagrama stanj
  - Primer enostavnih FSM so števci
- Za kompleksnejše FSM je zelo verjetno, da bo začetni diagram stanj vseboval več stanj kot je potrebno za izvedbo funkcije
- Minimizacija stanj je potrebna, saj manj stanj pomeni manj FF za realizacijo vezja
  - Ponavadi se zmanjša tudi kompleksnost kombinacijskih vezij
- Namesto, da bi iskali ekvivalentna stanja, je praviloma lažje iskati **neekvivalentna** stanja
  - Slednje izkoriščamo pri procesu minimizacije stanj

Ekvivalenca stanj

- **Definicija**: Stanji S_i *in* S_j *sta ekvivalentni takrat in samo takrat, ko se za vsako vhodno sekvenco tvori enaka izhodna sekvenca, ne glede na to ali je izhodiščno stanje* S_i *ali* S_j
- Če damo vhod w=0 na FSM v stanju S_i in FSM preide v stanje S_u, potem je S_u označimo kot ***naslednje stanje pri w=0*** stanja S_i
- Podobno, če je vhod w=1 in FSM preide v stanje S_y, potem je S_y ***naslednje stanje pri w=1*** stanja S_i
- Vsa naslednja stanja S_i so njegova ***naslednja stanja pri w=k***
- Velja dokler imamo en vhod ***k***, ki je lahko samo 0 ali 1. Če imamo več vhodov, potem ***k*** predstavlja vsa vrednotenja vhodov

Minimizacija z razdelki

- Iz definicije ekvivalentnosti stanj sledi, če sta stanji S_i and S_j ekvivalentnim potem so ekvivalentni tudi ustrezni ***k-nasledniki***
- Ob tem lahko zasnujemo postopek minimizacije, ki jemlje FSM kot niz. V postopku minimizacije tak niz razdelimo v ***razdelke,*** ki so sestavljeni iz neekvivalentnih podnizov
- **Definicija**: **Razdelek** sestoji iz enega ali več blokov, kjer vsak blok sestoji iz podniza stanj, ki so lahko ekvivalentna, vendar so stanja v enega bloka neekvivalentna stanjem v drugih blokih

Minimizacija z razdelki

Trenutno stanje	Naslednje stanje		Izhod 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

Začetni razdelek vsebuje vsa stanja v eni grupi

$$P_1=(ABCDEFG)$$

Zgled minimizacije z razdelki

- Naslednji razdelek loči stanja, ki imajo različne izhode: P_2=(ABD)(CEFG)
- Pregledamo vsa naslednja stanja pri vhodu 0 in 1 v vsakem bloku:
  - Blok (ABD):
    - Naslednja stanja pri w=0 (BDB)
    - Naslednja stanja pri w=1 (CFG)
  - Blok (CE**F**G):
    - Naslednja stanja pri w=0 (FFEF)
    - Naslednja stanja pri w=1 (EC**D**G) Vsa stanja niso v enem bloku. Problem je pri stanju **F**, ki ima naslednje stanje **D**. Zato bo stanje **F** NEEKVIVALENTNO ostalim CEG.
    - Novo stanje **F** zato postavimo v svojo skupino.

Trenutno stanje	Naslednje stanje		Izhod 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

P_3=(ABD)(CEG)(F)

Zgled minimizacije z razdelki

- P_3=(ABD)(CEG)(F)
- Blok (ABD):
  - Naslednja stanja pri w=0 (BDB) So vsa v istem bloku
  - Naslednja stanja pri w=1 (C**F**G) Niso v istem bloku, ker je **F** v drugem bloku kot C in G. Zato bo stanje B v novem bloku.
- Blok (CEG):
  - Naslednja stanja pri w=0 (FFF)
  - Naslednja stanja pri w=1 (ECG) C, E in G imamo lahko še vedno za ekvivalentna

Trenutno stanje	Naslednje stanje		Izhod 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

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

Zgled minimizacije z razdelki

- P_4=(AD)(B)(CEG)(F)
- Blok (AD)
  - Naslednja stanja pri w=0 (BB)
  - Naslednja stanja pri w=1 (CG)
    So vsa v istem bloku.
- Blok (CEG)
  - Naslednja stanja pri w=0 (FFF)
  - Naslednja stanja pri w=1 (ECG)
    So vsa v istem bloku.

Trenutno stanje	Naslednje stanje		Izhod 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

Sledi torej, da bo P_5=P_4 ➔ zato se postopek zaključi.

Stanji A in D sta ekvivalentni
Stanja C, E in G so ekvivalentna

P_5=(AD)(B)(CEG)(F)

Zgled minimizacije z razdelki

- Tabelo stanj zapišemo na novo
- Izbrišemo vrstice za D, E in G
- Zamenjamo stanja:

D➔ A in vse E➔ C ter G ➔ C

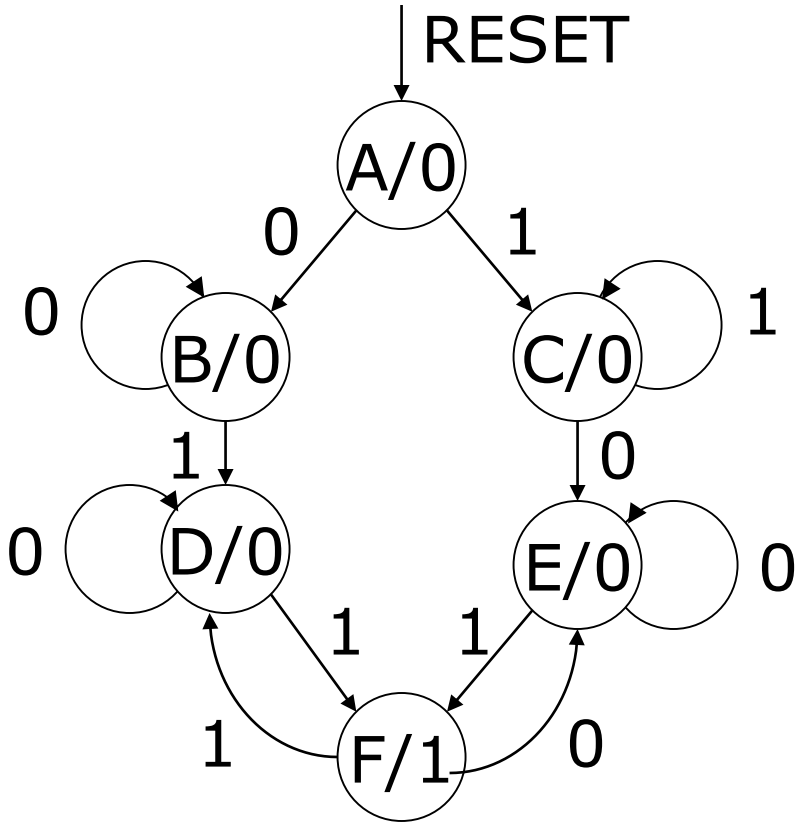
Trenutno stanje	Naslednje stanje		Izhod z
	w=0	w=1	
A	B	C	1
B	A	**F**	1
C	F	C	0
F	*C*	*A*	*0*

Trenutno stanje	Naslednje stanje		Izhod 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

$P_5=(AD)(B)(CEG)(F)$

Minimizacija z razdelki

- Določite katera stanja so ekvivalentna v podanem diagramu stanj



Trenutno stanje	Naslednje stanje		Izhod z
	w=0	w=1	
A	B	C	0
B	B	D	0
C	E	C	0
D	D	F	0
E	E	F	0
F	E	D	1

Minimizacija Mealy-evih avtomatov

- Stanji Mealy-evega FSM, ki imata:
  - enak izhod ob isti vhodni sekvenci in
  - prehajata ***v enaki stanji*** pod istimi vhodnimi pogoji
  - sta enaki in ju lahko združimo v eno stanje.
- Ta proces ponavljamo, dokler ni več ekvivalentnih stanj.