1. Uvod

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo
in informatiko

Vsebina poglavja

- Načrtovanje strojne opreme
- Digitalni sistemi
- Načini predstavitve digitalnega sistema
- Hitra implementacija digitalnega sistema

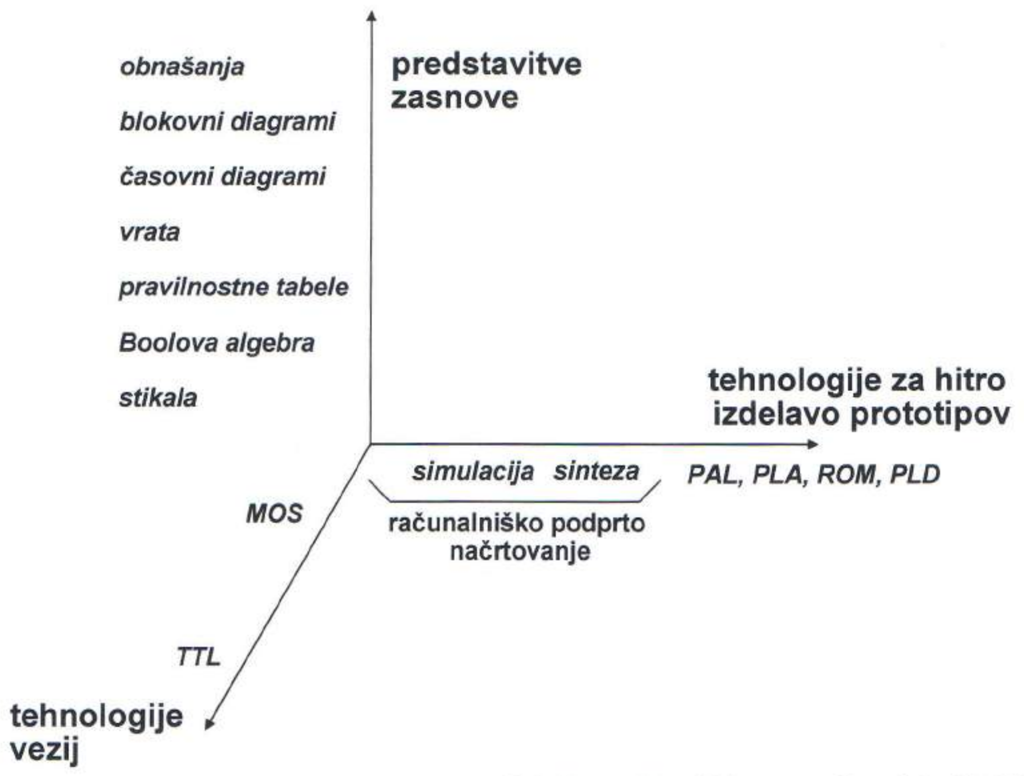
Načrtovanje strojne opreme

Značilnosti sodobnega načrtovanja strojne opreme

- ***Uporaba računalniško podprtih načrtovalskih orodij***
  - manjši poudarek na ročnih načrtovalskih metodah
  - poudarek na abstraktnih predstavitvah zasnove
  - načrtovanje strojne opreme podobno načrtovanju programov
- ***Prihod tehnologije vezij za hitro implementacijo zasnove***
  - programabilna logika namesto diskretne logike

Načrtovanje strojne opreme

Elementi sodobnega načrtovanja strojne opreme



Načrtovanje strojne opreme: Cilji

Cilji načrtovanja vsakega sistema so:

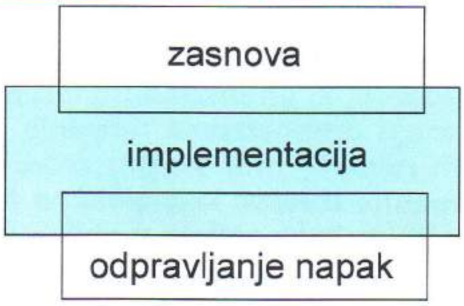
- **funkcionalna pravilnost,**
- **minimizacija cene**

  $C_T = C_D/N + C_M + C_S$, **kjer je**

  C_T - **skupna cena sistema,**
  C_D - **cena razvoja sistema,**
  N - **število izdelanih kopij sistema,**
  C_M - **cena izdelave ene kopije sistema in**
  C_S - **cena vzdrževanja ene kopije sistema,**

- **zmogljivost (zakasnitveni čas logičnih vrat, čas dostopa do pomnilnika, pasovna širina komunikacijske povezave, moč centralne procesne enote v MIPS-ih ipd.),**
- **kompatibilnost,**
- **poraba moči,**
- **velikost vezja,**
- **zanesljivost in**
- **testabilnost.**

Načrtovanje strojne opreme: Faze projekta



Zasnova

- **Nepopolne in neformalne zahteve in specifikacije, ki podajajo namen in funkcijo objekta, formaliziramo v natančne opise.**
- **Postavimo omejitve: hitrost, velikost vezja, poraba moči, cena.**
- **Abstraktne funkcionalne bloke razdelamo v konkretnejše realizacije.**

Implementacija

- **Uporabimo v fazi zasnove formaliziran opis za tvorbo fizičnega produkta.**
- **Izvedemo kompozicijo gradnikov z medsebojnim povezovanjem.**

Odpravljanje napak

- **Vzroki za nedelujoči sistem so: napake v zasnovi, napake pri povezovanju ali napake v komponentah.**
- **Načrtujemo tako, da bo odpravljanje napak čim lažje.**

Načrtovanje strojne opreme: Predstavitev sistema

Začetna specifikacija sistema, ki ga moramo razviti, je ponavadi podana v naravnem jeziku. Taka specifikacija je ponavadi premalo natančna in pogosto dvoumna. Moramo jo razdelati v natančnejši opis, ki predstavlja bodisi obnašanje bodisi strukturo sistema.

Opis obnašanja

Obnašanje sistema določa, *kaj* sistem dela oz. funkcije, ki jih sistem izvaja.

Obnašanje = (Vhodi, Izhodi)

Obnašanje sistema opišemo z enačbami, s tabelami, s časovnimi poteki vhodnih in izhodnih signalov, z diagrami poteka ali s programskimi stavki. Pogost sinonim za obnašanje sistema je funkcija sistema.

Opis strukture

Struktura sistema določa, *kako* je sistem sestavljen.

Struktura = (Komponente, Povezave)

Strukturo sistema podamo ponavadi z blokovnim diagramom. V njem vsak blok ali škatla predstavlja komponento sistema, črte (s puščico ali brez nje) pa komunikacijske povezave, ki združujejo komponente.

Načrtovanje strojne opreme: Načrtovalski nivoji

Sistem lahko obravnavamo na različnih načrtovalskih nivojih, določenih z množico komponent, ki jih imamo na teh nivojih za osnovne gradnike–*primitive* (nimajo prepoznavne notranje strukture). Na visokih (bolj abstraktnih nivojih) ima sistem relativno malo komponent, vsaka pa je sposobna izvajati kompleksne funkcije. Na nizkih (manj abstraktnih) nivojih se kaže sistem v obliki velikega števila primitivov z relativno preprostim obnašanjem.

Primitiv na kateremkoli načrtovalskem nivoju lahko na naslednjem nižjem načrtovalskem nivoju opišemo kot podsistem z več komponentami. Obratno pa lahko iz podsistema z abstrakcijo dobimo primitiv. Ta zveza med načrtovalskimi nivoji se imenuje hierarhija.

Digitalni sistemi imajo tri glavne načrtovalske nivoje: arhitekturni nivo, logični nivo in fizični nivo.

Hierarhični pristop k načrtovanju sistemov znižuje ceno in povečuje kvaliteto sistemov zaradi:

- zmanjšane kompleksnosti,
- razdelitve dela po nivojih,
- uporabe standardnih delov in
- večje robustnosti.

Načrtovanje strojne opreme: Stili načrtovanja hierarhičnih sistemov

Za hierarhične sisteme obstajata dva različna stila načrtovanja: načrtovanje od zgoraj navzdol in načrtovanje od spodaj navzgor.

Načrtovanje od zgoraj navzdol

Sistem opišemo najprej na višjem nivoju, nato pa vsako komponento na tem nivoju opišemo s primitivi na prvem nižjem nivoju in tako naprej.

Načrtovanje od zgoraj navzdol je v splošnem bolj zaželeno, posebej še v primerih, ko imamo za komponente na najnižjem nivoju na razpolago knjižnice s standardnimi elementi.

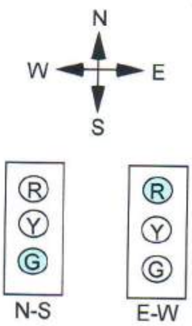
Načrtovanje od spodaj navzgor

Najprej načrtamo osnovne gradnike na najnižjem načrtovalskem nivoju. Z njihovo kompozicijo tvorimo osnovne gradnike za naslednji višji nivo in tako naprej.

Načrtovanje od spodaj navzgor je najbolj primerno pri razvoju ASIC vezij, kjer so komponente na vseh načrtovalskih nivojih narejene posebej za določen projekt in ne uporabljamo že razvitih vezij ali standardnih delov.

Načrtovanje strojne opreme: Razdelava predstavitev

Primer: **Krmilnik križiščnih semaforjev**



1. Funkcionalna specifikacija / Kaj sistem dela?

   V križišču so semaforji za smeri N, S, E, W.

   Na N gorijo iste luči kot na S, in na E iste kot na W.

   Zankaj v zaporedju ZELENA-RUMENA-RDEČA.

   Na N-S in E-W nikoli istočasno ne gorita ZELENA ali RUMENA.

   ZELENA luč gori 45 sekund, RUMENA 15, RDEČA 60.

2. Izpolnjene morajo biti naslednje omejitve oz. zahteve

   Hitrost: ni omejitev, saj so časovni intervali dolgi.

   Poraba moči: krmilnik naj troši manj kot 20 W.

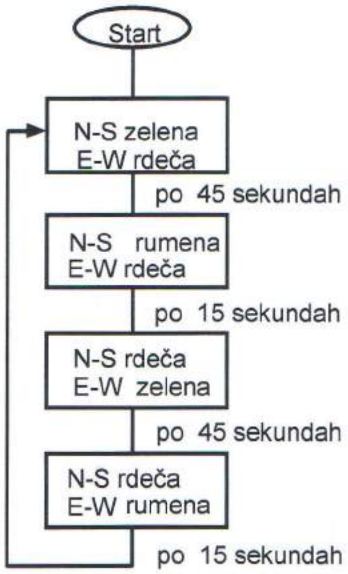
   Velikost: sestav naj ima površino manjšo od 20 cm^2.

   Cena: proizvodni stroški naj bodo manjši od 20 USD.

Načrtovanje strojne opreme

Primer: **Krmilnik križiščnih semaforjev**

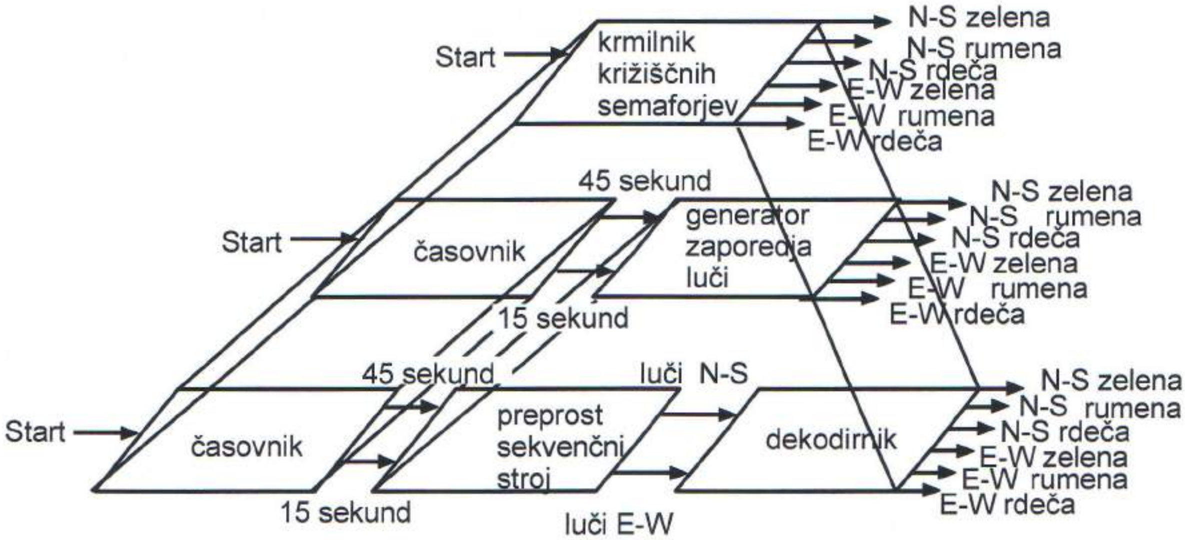
Predstavitev obnašanja krmilnika križiščnih semaforjev z diagramom poteka



Načrtovanje strojne opreme

Primer: **Krmilnik križiščnih semaforjev**

Predstavitev strukture krmilnika križiščnih semaforjev na treh hierarhičnih nivojih



Načrtovanje strojne opreme: Odpravljanje napak v sistemu

Kaj lahko gre narobe?

- ***Napake pri načrtovanju***
  - implementacija se ne ujema s funkcionalno specifikacijo
  - napačna zasnova logike (izvedena je napačna funkcija)
  - napačna interpretacija ali spregledani robni primeri
- ***Napake pri implementaciji zasnove***
  - posamezni moduli delujejo pravilno, njihova kompozicija pa ne
  - napačna razlaga vmesnika in časovnega obnašanja
  - napake pri povezovanju, električne napake
- ***Napake v komponentah***
  - sistem logično pravilen in pravilno povezan
  - ni zagotovljeno, da vse komponente pravilno delujejo (npr. pregorela komponenta)

Načrtovanje strojne opreme

Ugotavljanje napak s simulacijo pred implementacijo zasnove

Napotki za odkrivanje napak:

- izboljšati testabilnost zasnove
- izdelati načrt testiranja in izbirati testne primere
- postavljati hipoteze o vzroku problema
- testirati implementacijo po delih
- uporabljati laboratorijske instrumente za odkrivanje napak

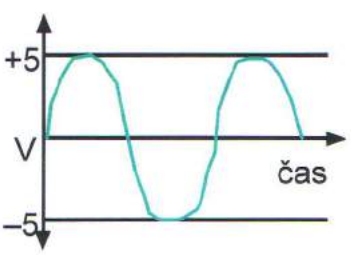
Digitalni sistemi

Primerjava digitalnih in analognih sistemov

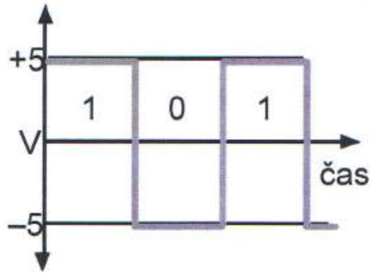
Informacijo lahko predstavimo na dva osnovna načina: analogno ali digitalno.

Analogne veličine lahko v danem območju zavzamejo kontinuirano vse vrednosti. Fizikalne veličine v naravi so večinoma analogne (npr. čas, dolžina, temperatura, teža, električna napetost).

Digitalna informacija je predstavljena z določenim številom diskretnih simbolov, imenovanih števke (tudi cifre ali digiti). Primer je 10 desetiških števk, s katerimi sestavljamo desetiška števila.



Analogni signal



Digitalni signal

Digitalni sistemi

Primerjava digitalnih in analognih sistemov

Analogni sistem je sistem, ki procesira informacijo v analogni obliki (npr. analogna ura) . Primer hranjenja informacije v analogni obliki je vinilna gramofonska plošča.

Digitalni sistem je sistem, ki procesira informacijo v digitalni obliki (npr. digitalna ura). Primer hranjenja informacije v digitalni obliki je CD plošča.

Mešani ali hibridni sistem je sistem, ki procesira tako digitalno kot analogno informacijo.

Današnji sistemi za procesiranje informacij so povečini digitalni, zato morajo za interakcijo z analognim okoljem uporabljati AD in DA pretvorbo. Primer AD pretvorbe je števec prevoženih kilometrov v avtu.

Velika prednost digitalnih sistemov je ta, da so za razliko od analognih sistemov sposobni iz do neke mere popačenih signalov dobiti pravilne izhodne vrednosti.

Digitalni sistemi so natančnejši in zanesljivejši.

Digitalni sistemi

Binarni digitalni sistemi

- *Dve diskretni vrednosti (bita):*
  da, vklop, 5 voltov, tok teče, magnetni sever, "1", true
  ne, izklop, 0 voltov, tok ne teče, magnetni jug, "0", false
- *Prednosti binarnih sistemov:*
  strogi matematični temelji, temelječi na logiki

IF garažna vrata so odprta
AND motor je prižgan
THEN avto lahko speljemo iz garaže

Vrata morajo biti odprta in motor mora teči, preden lahko speljemo.

IF N-S je zelena
AND E-W je rdeča
AND 45 sekund je preteklo od zadnje spremembe luči
THEN postavimo lahko naslednjo kombinacijo luči

Izpolnjeni morajo biti trije predpogoji, da se lahko izvede akcija.

Digitalni sistemi

Boolova algebra in logični operatorji

Algebra: spremenljivke, vrednosti, operacije

V Boolovi algebri sta vrednosti simbola 0 in 1.
Če je logična izjava nepravilna, ima vrednost 0.
Če je logična izjava pravilna, ima vrednost 1.

Operacije: AND, OR, NOT

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

Digitalni sistemi

Boolova algebra in logični operatorji

IF garažna vrata so odprta
AND motor je prižgan
THEN avto lahko speljemo iz garaže

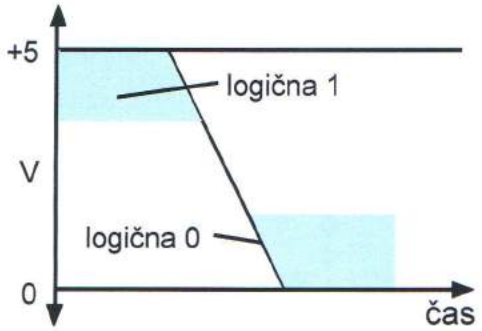
vrata odprta?	motor teče?	spelji avto?
false/0	false/0	false/0
false/0	true/1	false/0
true/1	false/0	false/0
true/1	true/1	true/1

Digitalni sistemi

Realni svet

Fizikalne elektronske komponente so zvezne, ne diskretne!

So osnovni gradniki vseh digitalnih komponent!



Prehod iz logične 1 v logično 0 se v realnih digitalnih sistemih ne zgodi v trenutku.

Za kratek čas lahko opazimo vmesne vrednosti.

Boolova algebra je koristna za opisovanje statičnega obnašanja digitalnih sistemov.

Zavedati se moramo tudi dinamičnega, časovno spremenljivega obnašanja!

Digitalni sistemi

Tehnologije digitalnih vezij

Tehnologija integriranih vezij

Na izbiro imamo prevodne, neprevodne in včasih prevodne (polprevodne) materiale.

Njihova interakcija lahko dopušča tok elektronov ali ne, kar tvori osnovo za električno krmiljena stikala.

Glavne tehnologije

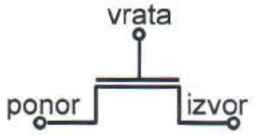
MOS: Metal-Oxide-Silicon

Bipolarna
- TTL: tranzistorsko-tranzistorska logika
- ECL: emitersko povezana logika

Digitalni sistemi

Tehnologija MOS

Tranzistor kot osnovno električno stikalo



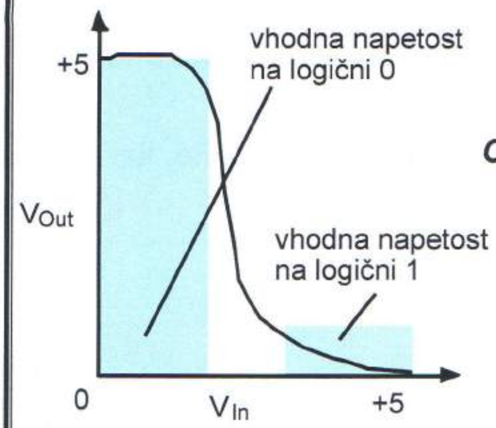
tropolno stikalo: vrata, izvor, ponor

Kadar napetost med vrati in izvorom presega prag, je stikalo sklenjeno ali prevodno, elektroni tečejo med izvorom in ponorom.

Kadar napetost odstranimo, je stikalo razklenjeno ali neprevodno, povezava med izvorom in ponorom je prekinjena.

Digitalni sistemi

Vezje, ki izvaja logično negacijo (NOT)



1 na vhodu daje 0 na izhodu.
0 na vhodu daje 1 na izhodu.

Obnašanje invertorja v odvisnosti od vhodne napetosti.

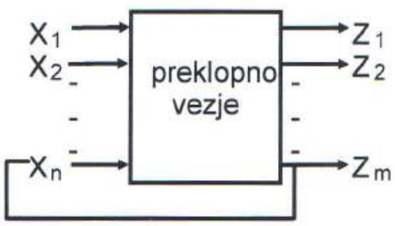
Napetost na vhodu naraste od 0V do 5V.

Napetost na izhodu ostane na 5V za določeno območje majhnih vhodnih napetosti, nato se hitro spremeni, vendar ne v trenutku!

Ne pozabimo na razliko med statičnim in dinamičnim obnašanjem.

Digitalni sistemi

Primerjava kombinacijskih in sekvenčnih vezij



Vezje je sestavljeno iz preklopnih elementov ali logičnih vrat. Prisotnost povratnih zank razloči med *sekvenčnimi* in *kombinacijskimi* vezji.

Kombinacijska vezja

Ni povratne vezave med izhodi in vhodi, izhodi so odvisni samo od vhodov.

***Primer:* Vezje za popolni seštevalnik**
Vhodi A, B, C_{in} se preslikajo v izhoda Sum, C_{out}.

Digitalni sistemi

Sekvenčna vezja

**V vezju obstajajo povratne vezave med izhodi in vhodi.
Izhodi so odvisni od vhodov in od celotne zgodovine izvajanja!**

**Vezja imajo samo omejeno število edinstvenih konfiguracij–*stanj*.
Npr. krmilnik semaforjev venomer zanka skozi štiri stanja.**

**V sekvenčnih logičnih vezjih zasledimo nove komponente:
pomnilni elementi za pomnjenje trenutnega stanja.**

**Izhod in novo stanje sta funkciji vhodov in starega stanja:
tj. vhodi vezja v povratnih zankah so stanja!**

Sinhroni sistemi

Periodični urin signal povzroča vpis novih vrednosti v pomnilne elemente in s tem spremembo stanja.

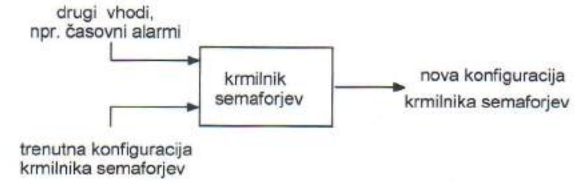
Asinhroni sistemi

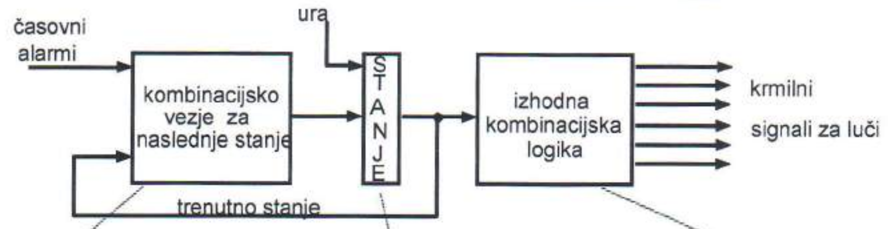
Ni nobene oznake o tem, kdaj spremeniti stanje.

Digitalni sistemi

Primerjava kombinacijskih in sekvenčnih vezij

Primer semaforja





logika za naslednje stanje
Preslika trenutno stanje in alarmne dogodke v naslednje stanje.

trenutno stanje
Pomnilni elementi zavzamejo naslednje stanje, ko pride urin signal.

izhodna logika
Trenutno stanje preslika v krmilne signale za spremembo luči in startanje časovnikov.

**IF krmilnik v stanju N-S zelena, E-W rdeča
AND aktiven časovnikov alarm za 45 sekund
THEN naslednje stanje postane N-S rumena,
E-W rdeča, ko se naslednjič aktivira urin signal**

Načini predstavitve digitalnega sistema: Stikala

Stikalo povezuje dve točki pod nadzorom kontrolnega signala.

normalno razklenjeno

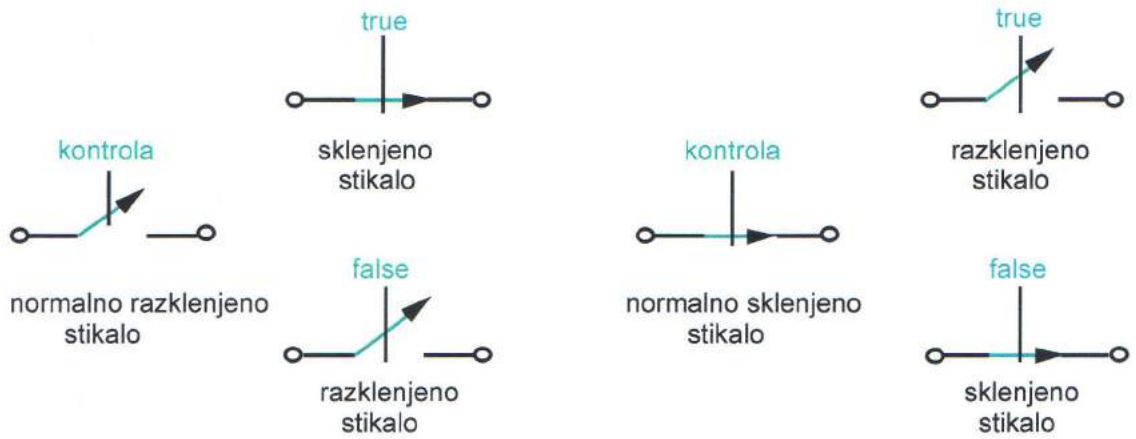
Kadar je kontrolni signal 0 (false), je stikalo razklenjeno.

Kadar je kontrolni signal 1 (true), je stikalo sklenjeno.

normalno sklenjeno

Kadar je kontrolni signal 0 (false), je stikalo sklenjeno.

Kadar je kontrolni signal 1 (true), je stikalo razklenjeno.



Načini predstavitve digitalnega sistema: Stikala

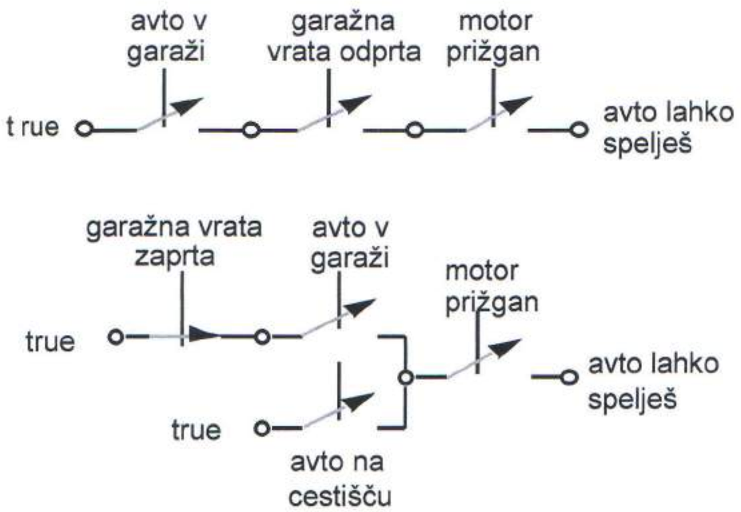
Primeri stikalnih vezij:

Primer:

IF avto v garaži
AND garažna vrata odprta
AND motor prižgan
THEN avto lahko odpelješ

Primer:

IF avto na cestišču
OR (avto v garaži
AND NOT garažna vrata zaprta)
AND motor prižgan
THEN avto lahko speljeroutes



Plavajoča vozlišča

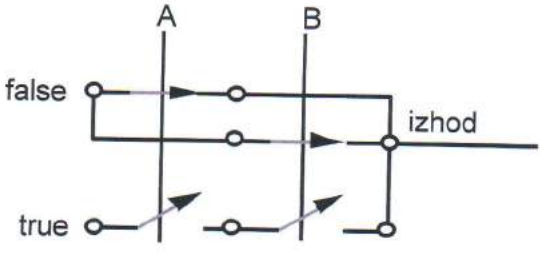
Kaj se zgodi, če motor ni prižgan?
Izhod plava, ker ni povezan na noben logični nivo.

Pri vseh mogočih nastavitvah kontrolnih signalov

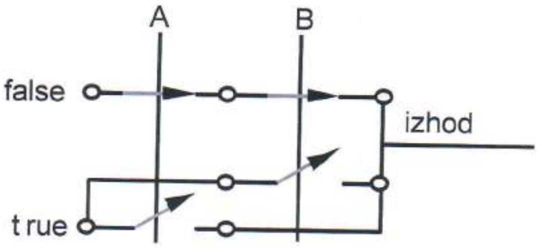
(1) mora biti vsak izhod po neki poti povezan na določen vhod,
(2) noben izhod po nobeni poti ne sme biti povezan na več kot en vhod.

Načini predstavitve digitalnega sistema: Stikala

Izvedba funkcij AND in OR s stikali



funkcija AND
serijska povezava na true



funkcija OR
paralelna povezava na true

Načini predstavitve digitalnega sistema

Pravilnostne tabele

V njih tabelarično uredimo vse možne vhodne kombinacije in njim prirejene izhodne vrednosti.

Primer: **polovični seštevalnik sešteje binarni števili A in B in daje vsoto Sum in prenos Carry.**

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Primer: **popolni seštevalnik sešteje binarni števili A in B in vhodni prenos C_{in} in daje vsoto Sum in izhodni prenos C_{out}.**

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

OPOMBA: binarno je 1 in 1 enako 0 s prenosom 1.

Načini predstavitve digitalnega sistema: Boolova algebra

vrednosti: 0, 1
spremenljivke: A, B, C, . . ., X, Y, Z
operacije: AND, OR, NOT

X AND Y označimo z X Y ali z X•Y
X OR Y označimo z X + Y
NOT X označimo z $\overline{X}$ ali z X'

Izpeljava Boolovih enačb iz pravilnostnih tabel

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum = $\overline{A}$ B + A $\overline{B}$

Z OR povežemo *produktne* člene za vsako vrstico pravilnostne tabele, kjer ima funkcija vrednost 1.

Če je vhodna spremenljivka 0, se pojavi v komplementirani obliki, če je 1, se pojavi nekomplementirana.

Carry = A B

Načini predstavitve digitalnega sistema: Boolova algebra

Drugi primer:

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum = $\overline{A}$ $\overline{B}$ Cin + $\overline{A}$ B $\overline{Cin}$ + A $\overline{B}$ $\overline{Cin}$ + A B Cin

Cout = $\overline{A}$ B Cin + A $\overline{B}$ Cin + A B $\overline{Cin}$ + A B Cin

Načini predstavitve digitalnega sistema: Boolova algebra

Reduciranje kompleksnosti Boolovih enačb

Na funkciji za izhodni prenos popolnega seštevalnika lahko uporabimo zakone Boolove algebre, da izpeljemo naslednji poenostavljeni izraz:

$$Cout = A\,Cin + B\,Cin + A\,B$$

A	B	Cin	Cout
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

B Cin

A Cin

A B

Preveri ekvivalenco z originalno pravilnostno tabelo za Cout:

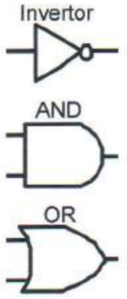
Postavi 1 v vsako vrstico pravilnostne tabele, kjer je produktni člen true.

Vsak produktni člen v zgornji enačbi pokriva natanko dve vrstici v pravilnostni tabeli; zadnja vrstica je "pokrita" s tremi členi.

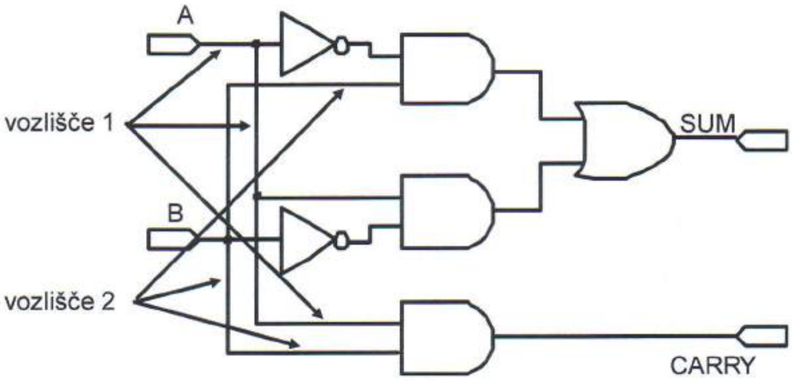
Načini predstavitve digitalnega sistema: Logična vrata

Logična vrata so najpogostejši osnovni gradnik pri načrtovanju digitalnih sistemov.

standardna predstavitev logičnih vrat



shema polovičnega seštevalnika

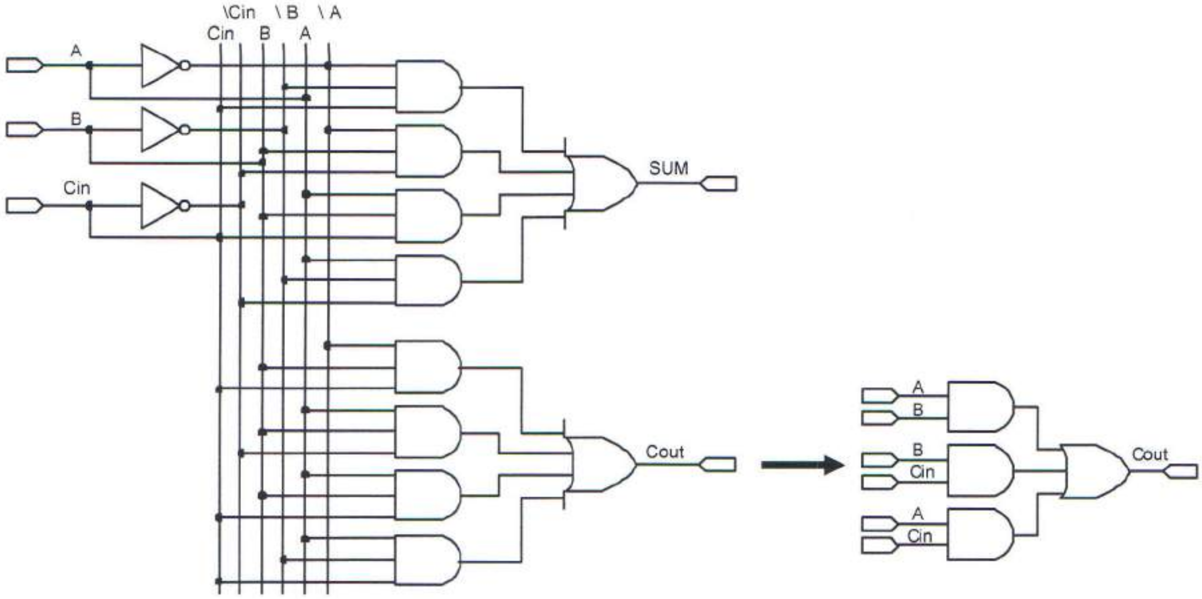


vozlišča: električno povezan skupek žic

seznam vozlišč: tabelarična ureditev vhodov in izhodov vrat ter vozlišč, s katerimi so povezani

Načini predstavitve digitalnega sistema: Vrata

Shema popolnega seštevalnika



Fan-in: število vhodov v dana vrata
Fan-out: število vseh na dani izhod povezanih vhodov vrat

Vsaka tehnologija postavlja meje za fan-in/fan-out.

Načini predstavitve digitalnega sistema: Časovni diagrami

Opazujemo dinamično obnašanje vezja.

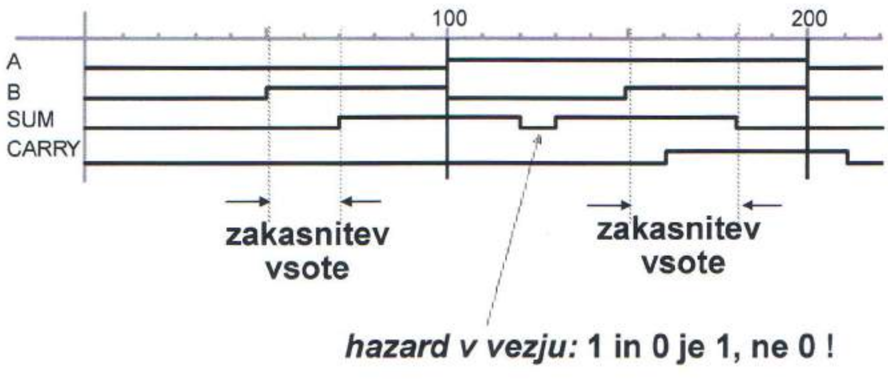
Realna vezja imajo od nič različne zakasnitve.

Spremembe na izhodih so zakasnjene glede na vhodne spremembe.

Zakasnitev je odvisna od poti v vezju.

Izhodi se lahko začasno spremenijo s pravilne vrednosti na napačno vrednost in spet nazaj na pravo vrednost–temu pravimo *hazard*.

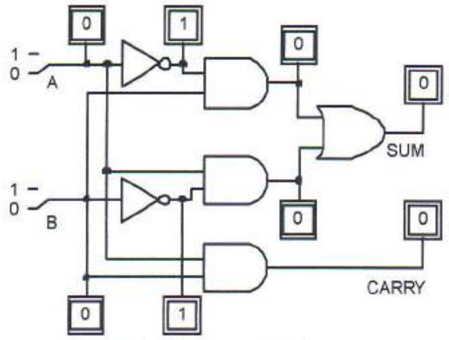
Časovni diagram polovičnega seštevalnika



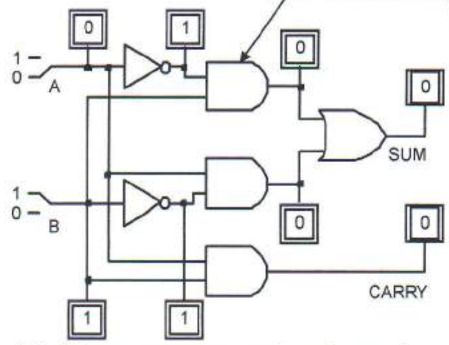
Načini predstavitve digitalnega sistema: Časovni diagrami

Sledenje zakasnitev: A=0,B=0 v A=0,B=1

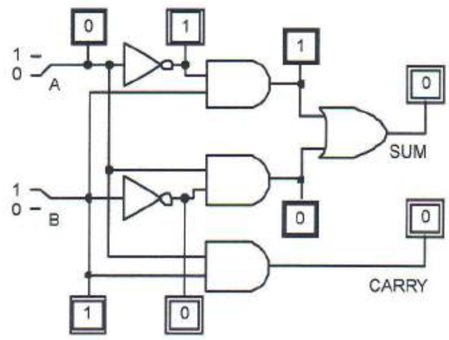
zakasnitev 10 časovnih enot



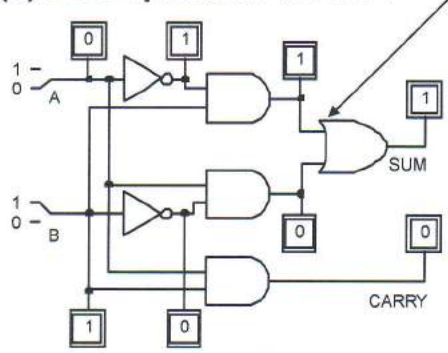
(i) začetno stanje



(ii) B se spremeni z 0 na 1



(iii) izhod zgornjih vrat AND se spremeni po 10 časovnih enotah

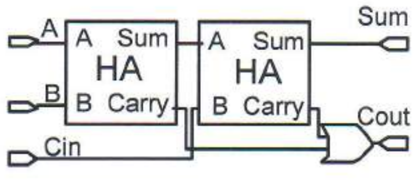


(iv) izhod vrat OR se spremeni po 10 časovnih enotah

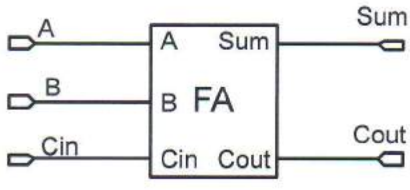
Načini predstavitve digitalnega sistema: Struktura

Blokovni diagram

- **strukturna organizacija zasnove**
- **črna škatla z vhodnimi in izhodnimi povezavami**
- **ustreza dobro definiranim funkcijam**
- **osredotočenje na to, kako povežemo komponente v sistem**



popolni seštevalnik realiziran s kompozicijo dveh polovičnih seštevalnikov

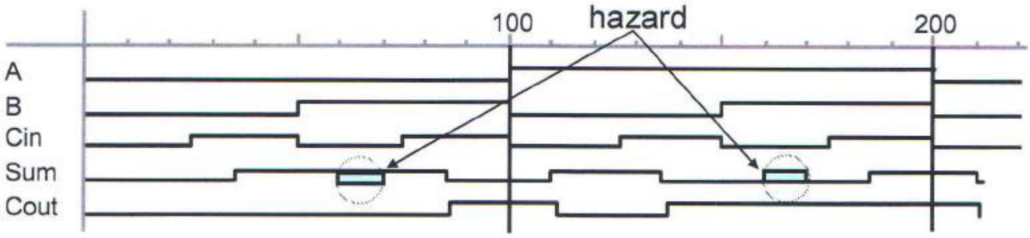


predstavitev popolnega seštevalnika z blokovnim diagramom

Načini predstavitve digitalnega sistema

Preverjanje časovnih diagramov

Ali se popolni seštevalnik, sestavljen iz dveh polovičnih seštevalnikov, obnaša enako kot neposredna izvedba popolnega seštevalnika s samimi vrati?



Signala Sum in Cout zaostajata za spremembami na vhodu.

Koliko časovnih enot po spremembi vhoda so izhodi že veljavni?

Načini predstavitve digitalnega sistema: Obnašanja

Jezik ABEL za opis strojne opreme

Primer: **Polovični seštevalnik v ABEL-u**

specifikacija pravilnostne tabele

```
MODULE polovicni_sestevalnik;
a, b, sum, carry PIN 1, 2, 3, 4;
TRUTH_TABLE {[a, b] -> [sum, carry]}
   [0, 0] -> [0, 0];
   [0, 1] -> [1, 0];
   [1, 0] -> [1, 0];
   [1, 1] -> [0, 1];
END polovicni_sestevalnik;
```

specifikacija enačbe

```
MODULE polovicni_sestevalnik;
a, b, sum, carry PIN 1, 2, 3, 4;
EQUATIONS
   SUM = (A & !B) # (!A & B);
   CARRY = A & B;
END polovicni_sestevalnik;
```

OR **NOT** **AND**

Načini predstavitve digitalnega sistema: Obnašanja

Jezik VHDL za opis strojne opreme

Primer: **Polovični seštevalnik v VHDL-u**

```
-- ***** model invertorja *****
-- zunanji priključki
ENTITY invertor;
   PORT (a:  IN BIT;  z:  OUT BIT);
END invertor;

-- notranje obnašanje
ARCHITECTURE obnašanje OF invertor IS
BEGIN
    z <= NOT a AFTER 10 ns;
END obnašanje;


-- ***** model vrat AND *****
-- zunanji priključki
ENTITY vrata_AND;
   PORT (a, b: IN BIT;  z: OUT BIT);
END vrata_AND;

-- notranje obnašanje
ARCHITECTURE obnašanje OF vrata_AND IS
BEGIN
    z <= a AND b AFTER 10 ns;
END obnašanje;
```

invertor kot črna škatla

notranje obnašanje

Načini predstavitve digitalnega sistema: Obnašanja

```
-- ***** vrata OR *****
-- zunanji priključki
ENTITY vrata_OR;
   PORT (a, b: IN BIT;  z: OUT BIT);
END vrata_OR;

-- notranje obnašanje
ARCHITECTURE obnašanje OF vrata_OR IS
BEGIN
    z <= a OR b AFTER 10 ns;
END obnašanje;


-- ***** model polovičnega seštevalnika *****
-- zunanji priključki
ENTITY polovični_seštevalnik;
    PORT (a, b: IN BIT;  sum, c_out: OUT BIT);
END polovični_seštevalnik;

-- notranja struktura
ARCHITECTURE struktura OF polovični_seštevalnik IS
   -- potrebne komponente
   COMPONENT invertor
       PORT (a: IN BIT;  z: OUT BIT);  END COMPONENT;
   COMPONENT vrata_AND
       PORT (a, b: IN BIT;  z: OUT BIT);  END COMPONENT;
   COMPONENT vrata_OR
       PORT (a, b: IN BIT;  z: OUT BIT);  END COMPONENT;

   -- interni signali
   SIGNAL s1, s2, s3, s4: BIT;
```

Modeli za vrata AND, OR in NOT so navadno opisani v knjižnici.

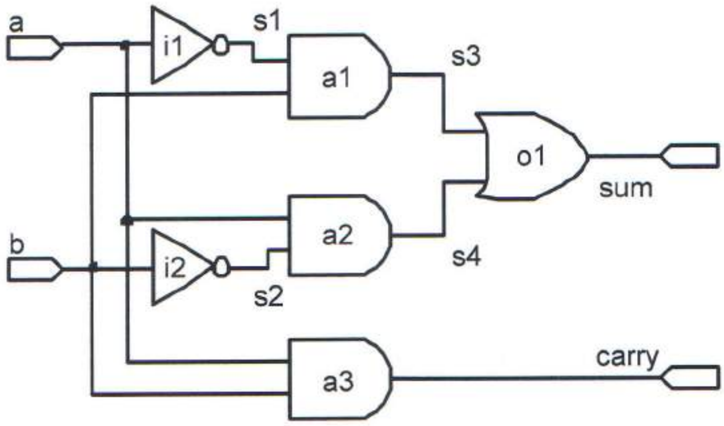
posamezne komponente uporabljene v modelu polovičnega seštevalnika

Načini predstavitve digitalnega sistema: Obnašanje

```
BEGIN
        -- ena vrstica za vsaka vrata, opiše njihov tip in povezave
        i1: invertor PORT MAP (a, s1);
        i2: invertor PORT MAP (b, s2);
        a1: vrata_AND PORT MAP (b, s1, s3);
        a2: vrata_AND PORT MAP (a, s2, s4);
        a3: vrata_AND PORT MAP (a, b, carry);
        o1: vrata_OR PORT MAP (s3, s4, sum);
END struktura;
```

tekstovni opis seznama povezav

Ta specifikacija v VHDL-u ustreza naslednji shemi z označenimi vrati in vozlišči.



Hitra implementacija digitalnega sistema

Cilji:

- **hitra izgradnja digitalnega sistema, da dokažemo koncept**
- **hitra raziskava alternativnih zasnov**
- **morebitno žrtvovanje nekaterih zmogljivosti (hitrost, poraba moči ali velikost) sistema na račun hitre izdelave prototipa**

Tehnike:

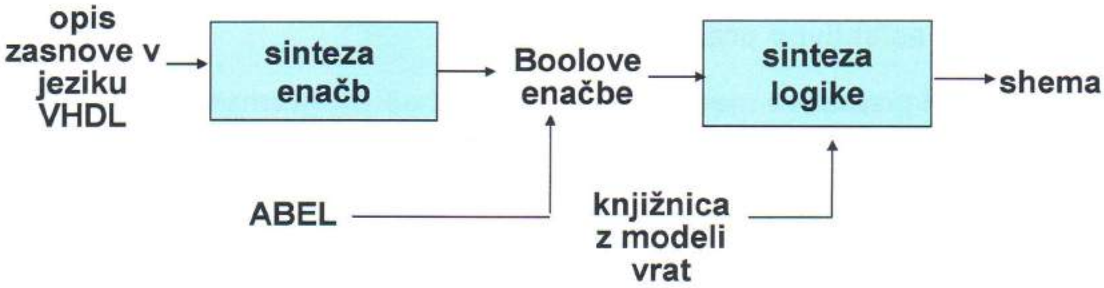
- **računalniško podprta načrtovalska orodja**
  - **simulacija: simulator odkrije, kako se bo sistem obnašal, preden ga zgradimo.**
  - **sinteza: iz opisa zasnove na visokem nivoju generira podrobne opise (npr. generiranje sheme iz Boolovih enačb).**
- **tehnologije za hitro implementacijo zasnove**
  - **programabilna logična vezja**

Hitra implementacija digitalnega sistema

Računalniško podprta načrtovalska orodja

Orodja za sintezo

- Ustvarijo eno predstavitev zasnove iz druge predstavitve.
- Ponavadi preslikajo abstraktnejše opise v bolj podrobne opise, ki so bližji končni obliki implementacije.
- Včasih orodja za sintezo preslikajo dano predstavitev v optimalnejšo obliko te predstavitve (npr. program *espresso*).



Hitra implementacija digitalnega sistema

Simulacija

- Simulator je program, ki dinamično izvaja abstraktni opis sistema.
- Verificira funkcionalno pravilnost in pridobi nekaj informacije o časovnih razmerah, preden je sistem fizično zgrajen.
- Lažje je preizkusiti in odpraviti napake v simulaciji kot pa v implementiranem sistemu.
- Simulacija ne more zagotoviti, da bo sistem delal.
  - Je samo tako dobra kot izbran testni primer.
  - Ne preverja električnih napak.
  - Izvede abstrakcijo nad nekaterimi dejstvi v realnem sistemu.

Logična simulacija

- Sistem je opisan z logičnimi vrati.
- Vrednosti signalov sta 0 in 1 (in tudi nekatere druge vrednosti, ki jih bomo predstavili kasneje).
- Je primerna za verifikacijo pravilnostne tabele.

Časovna simulacija

- Prikaže časovne poteke vhodnih in izhodnih signalov.
- Modelirajo se zakasnitve vrat.
- Ali je oblika signalov takšna, kot smo pričakovali?
- Odkrije ozka grla v zmogljivosti vezja.

Hitra implementacija digitalnega sistema

Tehnologije za hitro implementacijo

- Funkcijo in povezavo komponent naredimo “po lastni meri”.
- Je alternativa za diskretna logična vrata in ožičevanje.
- Reducira kompleksnost ožičenja in število čipov.
- Omogoča hitrejše spremembe in izboljšave sistema.

Programiranje

- Funkcija komponente se konfigurira po pravilnostni tabeli.
  - programabilne matrike z varovalkami
  - selektivno prežgane varovalke
- Tudi povezave med internimi moduli so programabilne.

Hitra implementacija digitalnega sistema

Primer: Bralni pomnilniki

- V strojni opremi izvedemo dvodimenzionalno polje.
- Vhodi tvorijo indeks v polje.
- Binarna beseda na naslovljeni pomnilniški lokaciji vsebuje izhodne vrednosti.
- Vsebina se programira enkrat, bere mnogokrat.

Polovični seštevalnik realiziran z ROM-om:

polovični seštevalnik

naslov (A B)	vsebina (SUM CARRY)
0 0	0 0
0 1	1 0
1 0	1 0
1 1	0 1

Popolni seštevalnik realiziran z ROM-om:

popolni seštevalnik

naslov (A B Cin)	vsebina (SUM Cout)
0 0 0	0 0
0 0 1	1 0
0 1 0	1 0
0 1 1	0 1
1 0 0	1 0
1 0 1	0 1
1 1 0	0 1
1 1 1	1 1

Povzetek poglavja

Predstavili smo:

- *proces načrtovanja:*
  - hierarhični pristop k načrtovanju
- *vrste sistemov, ki jih bomo načrtovali:*
  - binarni digitalni sistemi
  - kombinacijska in sekvenčna vezja
  - implementacija v tehnologiji MOS ali bipolarni tehnologiji
- *mnogo nivojev za predstavitev digitalnih sistemov:*
  - od nivoja stikal do opisov obnašanja
- *spremembe na področju tehnologije:*
  - hitra implementacija digitalnih sistemov
  - računalniška načrtovalska orodja (predvsem orodja za sintezo in simulacijo)
  - programabilna logična vezja

2. Številski sistemi in kodi

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo
in informatiko

Vsebina poglavja

- Številski sistemi
  - Pozicijska števila
  - Osnova številskega sistema
  - Pretvorbe med številskimi sistemi
- Binarna aritmetika
  - Nepredznačena števila
  - Predznačena števila
- Binarni kodi
  - BCD kod
  - Znakovni kodi
  - Kodi za odkrivanje napak
  - Števila s plavajočo vejico

Številski sistemi

Pozicijska števila

- Številski sistem je definiran z osnovnimi simboli, imenovanimi števke (cifre, digiti), in načini, kako se te števke kombinirajo, da predstavimo celoten obseg števil. Za človeka je najbolj naraven desetiški številski sistem, ki je sestavljen iz 10 arabskih števk (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
- Število v nekem številskem sistemu tvorimo tako, da zapišemo zaporedje števk, pri čemer pozicija števke določa njeno težo, to je faktor, s katerim se števka množi. Takemu zapisu pravimo pozicijski zapis števila. Pozicijska števila se veliko uporabljajo, ker so kompaktna in je z njimi lažje računati kot z nepozicijskimi števili (npr. rimskimi števili).
- Primer pozicijskega zapisa števila: Štiri števke v desetiškem številu 2979 predstavljajo od leve proti desni tisočice (2), stotice (prva 9), desetice (7) in enice (druga 9). Število lahko razstavimo na naslednji način:

  $$2979 = 2 \cdot 10^3 + 9 \cdot 10^2 + 7 \cdot 10^1 + 9 \cdot 10^0.$$

- Primer nepozicijskega zapisa števila: Rimske številke so sestavljene iz majhne množice črk, ki služijo kot števke: I (ena), V (pet), X(deset), L(petdeset), C(sto), D(petsto), M(tisoč). Število 2979 se tako zapiše z devetimi števkami kot MMCMLXXIX in ima naslednjo interpretacijo:

  MMCMLXXIX = M + M – C + M + L + X + X – I + X.

Številski sistemi

Osnova številskega sistema

- Število različnih števk, r, v številskem sistemu imenujemo osnova (baza, radiks ali koren) številskega sistema. Najpomembnejši so številski sistemi z osnovami $r = 2, 3, 8, 10, 16$.
- Primer: Celo število $N_r = a_{n-1}a_{n-2} \ldots a_1a_0$ v številskem sistemu z osnovo r lahko izrazimo kot

  $$N_r = a_{n-1}a_{n-2} \ldots a_1a_0 = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \ldots + a_1r^1 + a_0r^0 = \sum_{i=0}^{n-1} a_ir^i$$

  kjer so števke a_i elementi množice $\{0, 1, 2, \ldots, r - 1\}$

Ime številskega sistema	*Baza r*	*Števke številskega sistema (r števk)*
dvojiški (binarni)	2	0, 1
trojiški (ternarni)	3	0, 1, 2
osmiški (oktalni)	8	0, 1, 2, 3, 4, 5, 6, 7
desetiški (decimalni)	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
šestnajstiški (heksadecimalni)	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Številski sistemi

Število	$r = 2$	$r = 3$	$r = 8$	$r = 10$	$r = 16$
nič	0	0	0	0	0
ena	1	1	1	1	1
dve	10	2	2	2	2
tri	11	10	3	3	3
štiri	100	11	4	4	4
pet	101	12	5	5	5
šest	110	20	6	6	6
sedem	111	21	7	7	7
osem	1000	22	10	8	8
devet	1001	100	11	9	9
deset	1010	101	12	10	A
enajst	1011	102	13	11	B
dvanajst	1100	110	14	12	C
trinajst	1101	111	15	13	D
štirinajst	1110	112	16	14	E
petnajst	1111	120	17	15	F

Številski sistemi

Izračun optimalne osnove številskega sistema

- Prikladnost številskega sistema za fizično realizacijo ne ustreza njegovi prikladnosti za uporabo s strani človeka.
- Z večanjem osnove številskega sistema se zmanjšuje potrebno število števk za zapis nekega števila, povečuje pa se število števk v številskem sistemu.
- Elektronsko vezje, s katerim želimo realizirati števko v nekem številskem sistemu, mora imeti toliko različnih diskretnih stanj, kolikor je različnih števk v številskem sistemu, tako da se lahko vsakemu stanju priredi pomen ene števke. Za prikaz nekega števila z n števkami v številskem sistemu z osnovo r je torej potrebnih n vezij, od katerih ima vsako r diskretnih stanj. Na ta način se da prikazati toliko števil N, kolikor je različnih kombinacij:

  $$N = r^n$$

  Za število z n števkami znaša število različnih diskretnih stanj:

  $$v = r \cdot n$$

  Iz prejšnjega izraza sledi

  $$v = \ln N \cdot \frac{r}{\ln r}$$

Številski sistemi

Izračun optimalne osnove številskega sistema

- Ker je v skupno število diskretnih stanj, ki jih je z vezji potrebno realizirati, je optimalna rešitev prikazati število s čim manj skupnih stanj. Če izraz za v odvajamo po r in rezultat izenačimo z 0, dobimo pogoj za minimum

  $r = e \approx 2{,}71$

  Ker je r lahko samo celo število, je rešitev $r = 3$, ki je najbližji številu e. Izkaže se, da sta številska sistema z $r = 2$ in $r = 4$ samo za 5% manj učinkovita. Ker je z elektronskim vezjem najlažje predstaviti dve diskretni stanji (npr. tok teče, tok ne teče), je za osnovo številskega sistema v računalnikih izbran $r = 2$.

Pretvorba dvojiških števil v desetiška

- Ker ljudje uporabljamo desetiški številski sistem, digitalna vezja pa delujejo na osnovi dvojiškega, je potrebno vedeti, kako pretvoriti števila iz enega sistema v drugega.

  Pretvorba celih števil

  Nepredznačeno dvojiško celo število N_2 v splošni n-bitni obliki zapišemo kot

  $N_2 = b_{n-1}b_{n-2} \dots b_1b_0$, kjer je $b_i \in \{0,1\}$ in $0 \le i \le n-1$.

  N_{10} lahko zapišemo kot

  $N_{10} = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + \dots + b_1 2^1 + b_0 2^0$ ali krajše

  $$N_{10} = \sum_{i=0}^{n-1} b_i 2^i$$

- Primer: $11101_2 = 1\cdot2^4 + 1\cdot2^3 + 1\cdot2^2 + 0\cdot2^1 + 1\cdot2^0 = 16 + 8 + 4 + 0 + 1 = 29$

Pretvorba dvojiških števil v desetiška

- **Za računalnik je primernejše računanje v naslednji obliki:**

$$N_{10} = 2\cdot\Big(\sum_{i=1}^{n-1}(b_i 2^{i-1})\Big) + b_0 = 2\cdot\Big(2\cdot\Big(\sum_{i=2}^{n-1} b_i 2^{i-2}\Big) + b_1\Big) + b_0 =$$

$$\ldots$$

$$= 2\cdot\big(2\cdot \cdots (2\cdot(2\cdot b_{n-1} + b_{n-2}) \cdots) + b_1\big) + b_0$$

- **Primer:** $11101_2 = 2\cdot(2\cdot(2\cdot(2\cdot1 + 1) + 1) + 0) + 1 =$
  $= 2\cdot(2\cdot(2\cdot(3) + 1) + 0) + 1 =$
  $= 2\cdot(2\cdot(7) + 0) + 1 =$
  $= 2\cdot(14 + 0) + 1 =$
  $= 28 + 1 =$
  $= 29$

Pretvorba dvojiških števil v desetiška

Pretvorba ulomkov in mešanih števil

Če $N_2 = b_{n-1}b_{n-2} \ldots b_1b_0$ predstavlja ulomek, dobimo

$$N_{10} = b_{n-1}2^{-1} + b_{n-2}2^{-2} + \cdots + b_1 2^{1-n} + b_0 2^{-n} = \sum_{i=0}^{n-1} b_i 2^{i-n}$$

Če $N_2 = b_{n-1}b_{n-2} \ldots b_m, b_{m-1} \ldots b_1b_0$ predstavlja mešano število, kjer je "," binarna vejica, dobimo

$$N_{10} = \sum_{i=0}^{n-1} b_i 2^{i-m}\ .$$

Če izpostavimo faktor 2^{-m}, dobimo

$$N_{10} = 2^{-m} \cdot \sum_{i=0}^{n-1} b_i 2^{i} = 2^{-m} \cdot N'_{10},$$

kjer je N'_{10} desetiško celo število, ki ustreza binarnemu številu $N_2 = b_{n-1}b_{n-2} \ldots b_m b_{m-1} \ldots b_1b_0$. Člen 2^{-m} pomakne desetiško vejico števila N'_{10} za m mest v levo in ga tako spremeni v mešano število N_{10}.

Pretvorba dvojiških števil v desetiška

- Primer: Dvojiško mešano število $N_2 = 011001{,}01_2$ pretvorite v desetiško.

$m = 2$, ulomek dvojiškega mešanega števila je torej predstavljen z dvema veljavnima binarnima števkama – bitoma.

$N'_2 = 01100101$ je celo dvojiško število, ki ga dobimo iz mešanega dvojiškega števila N_2, če mu odstranimo binarno vejico.

Celo število N'_2 pretvorimo v celo število N'_{10} postopku za pretvorbo celih števil iz dvojiškega v desetiški številski sistem. Dobimo $N'_{10} = 101_{10}$.

Zdaj samo še pomnožimo N'_{10} s faktorjem 2^{-m} in dobimo

$$N_{10} = 2^{-2} \cdot N'_{10} = \tfrac{1}{4} \cdot 101_{10} = 25{,}25_{10}$$

Pretvorba desetiških števil v dvojiška

- Ideja pretvorbe je v tem, da želimo ugotoviti, koliko potenc števila 2 je vsebovanih v številu, tako da ga lahko v dvojiški obliki izrazimo kot

$$N_2 = b_{n-1}b_{n-2} \ldots b_1b_0 = \sum_{i=0}^{n-1} b_i 2^i$$

N_{10} delimo z 2 in dobimo kvocient Q_0 in ostanek R_0. Če je N_{10} lih, bo $R_0 = 1$, če je N_{10} sod, bo $R_0 = 0$. R_0 je torej enak b_0 – skrajnemu desnemu bitu števila N_2. Zdaj delimo Q_0 z 2, da dobimo nov kvocient Q_1 in ostanek R_1. V tem primeru imamo $R_1 = b_1$. Postopek končamo, ko kvocient doseže vrednost 0.

- Primer: $N_{10} = 249_{10}$

i=0	249 : 2 = 124	$b_0 = 1$
i=1	124 : 2 = 62	$b_1 = 0$
i=2	62 : 2 = 31	$b_2 = 0$
i=3	31 : 2 = 15	$b_3 = 1$
i=4	15 : 2 = 7	$b_4 = 1$
i=5	7 : 2 = 3	$b_5 = 1$
i=6	3 : 2 = 1	$b_6 = 1$
i=7	1 : 2 = 0	$b_7 = 1$

$N_2 = 11111001_2$

Pretvorba desetiških števil v dvojiška

Pretvorba ulomkov in mešanih števil

- Naj bo N_{10} splošno desetiško mešano število (s celim delom pred decimalno vejico in m veljavnimi desetiškimi števkami za njo) oblike

$N_{10} = d_{p-1}d_{p-2} \dots d_m, d_{m-1} \dots d_1d_0$

Pričakovali bi, da bo pretvorba potekala takole:

1. Pretvori N_{10} v N'_{10} z množenjem s faktorjem 10^m.
2. Po postopku za pretvorbo celih števil pretvori celo število N'_{10} v celo število N'_2.
3. Izračunaj N_2 s produktom $N'_2 \cdot 10^{-m}$.

Problem je v zadnjem koraku, saj bi za množenje morali uporabljati dvojiško aritmetiko. Zaradi tega uporabljamo za pretvorbo mešanih števil drugo pot.

Pretvorba desetiških števil v dvojiška

- Desetiško število N_{10} razdelimo v celoštevilčni del

$N^I_{10} = d_{p-1}d_{p-2} \dots d_{m+1}d_m,0$

in decimalni del za decimalno vejico

$N^F_{10} = 0,d_{m-1}d_{m-2} \dots d_1d_0.$

Pretvorbo N^I_{10} v N^I_2 opravimo po prejšnji proceduri, N^F_{10} v N^F_2 pa pretvarjamo po novi proceduri. Po pretvorbi dvojiška rezultata strnemo v končni rezultat

$N_2 = N^I_2,N^F_2.$

Nova procedura za pretvorbo dela za decimalno vejico

Naj bo $N^F_2 = 0,b_{-1}b_{-2} \dots$ želen binarni ekvivalent za N^F_{10}.

Pretvorba desetiških števil v dvojiška

- **Množenje N^F_{10} z 2 dâ desetiško število oblike $A = d_0,d_{-1}d_{-2}\cdots$, v katerem je števka enic d_0 bodisi 0 bodisi 1. $d_0 = 1$, če in samo če $N^F_{10} \geq 0{,}5$ in $d_0 = 0$, če in samo če $N^F_{10} < 0{,}5$. Če pogledamo N^F_2, vidimo, da je $b_{-1} = 1$, če in samo če $N^F_{10} \geq 0{,}5$ in $b_{-1} = 0$, če in samo če $N^F_{10} < 0{,}5$. Iz tega sledi, da je $d_0 = b_{-1}$. Celoštevilčni del rezultata množenja decimalnega dela desetiškega števila z 2 je prvi bit dvojiškega ulomka. Zdaj lahko vzamemo decimalni del A-ja in ga pomnožimo z 2, da dobimo naslednji bit b_{-2} za N^F_2, in tako naprej. V splošnem, zahteva se n množenj z 2, da dobimo prvih n bitov binarnega ulomka.**

- **Primer: Pretvorite število $N_{10} = 0{,}62705$ v 8-bitni rezultat $N_2 = 0,b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}b_{-7}b_{-8}$.**

i=1	0,62705 · 2 = 1,25410	$b_{-1} = 1$
i=2	0,25410 · 2 = 0,50820	$b_{-2} = 0$
i=3	0,50820 · 2 = 1,01640	$b_{-3} = 1$
i=4	0,01640 · 2 = 0,03280	$b_{-4} = 0$
i=5	0,03280 · 2 = 0,06560	$b_{-5} = 0$
i=6	0,06560 · 2 = 0,13120	$b_{-6} = 0$
i=7	0,13120 · 2 = 0,26240	$b_{-7} = 0$
i=8	0,26240 · 2 = 0,52480	$b_{-8} = 0$

Rezultat:
$N_2 = 0{,}10100000$

Bolj natančen je zaokrožen rezultat:
$N_2 = 0{,}10100001$

Pretvorba med števili v številskih sistemih z osnovo 2^k

- **Števila z osnovo 2^k, še posebej osmiška (k=3) in šestnajstiška (k=4), se uporabljajo za krajši zapis dvojiških podatkov.**
- **Primeri pretvorbe:**
  - **dvojiško v osmiško**

    $N_2 = \underbrace{100}_{4}\,\underbrace{110}_{6}\,\underbrace{101}_{5}$

    $N_8 = 4\ 6\ 5$
  - **osmiško v dvojiško**

    $N_8 = 1\ 0\ 7\ 2\ 5\ 1$

    $N_2 = \overbrace{001}^{1}\,\overbrace{000}^{0}\,\overbrace{111}^{7}\,\overbrace{010}^{2}\,\overbrace{101}^{5}\,\overbrace{001}^{1}$
  - **dvojiško v šestnajstiško**

    $N_2 = \underbrace{1000}_{8}\,\underbrace{1110}_{E}\,\underbrace{1010}_{A}\,\underbrace{1001}_{9}$

    $N_{16} = 8\ E\ A\ 9$
  - **šestnajstiško v dvojiško**

    $N_{16} = A\ B\ 8\ F$

    $N_2 = \overbrace{1010}^{A}\,\overbrace{1011}^{B}\,\overbrace{1000}^{8}\,\overbrace{1111}^{F}$
  - **šestnajstiško v osmiško**

    $N_{16} = A\ 1\ F\ 5\ 2\ 1\ 3,\ C$

    $N_2 = 1010000111110101001000010011,1100$

    $N_8 = 1\ 2\ 0\ 7\ 6\ 5\ 1\ 0\ 2\ 3,\ 6$

3. Boolova algebra in Boolove funkcije

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Vsebina poglavja

- Definicija Boolove algebre
  - Huntingtonovi postulati
  - Preklopna algebra
- Boolove funkcije
  - Definicija Boolove in preklopne funkcije
  - Pravilnostna tabela
  - Nepopolno specificirane funkcije
  - Boolovi izrazi
- Osnovne lastnosti Boolove algebre
  - Dualnost
  - Izreki Boolove algebre
  - Dokazovanje izrekov
- Druge Boolove algebre
  - Izjavni račun
  - Algebra množic
  - Boolova funkcijska algebra

Vsebina poglavja (nadaljevanje)

- **Preklopna vezja**
  - Logična vrata
  - Kombinacijska preklopna vezja
  - Pozitivna in negativna logika
- **Kanonične oblike preklopnih funkcij**
  - Mintermi
  - Popolna disjunktivna normalna oblika
  - Makstermi
  - Popolna konjunktivna normalna oblika
  - Nepopolno specificirane funkcije v kanoničnih oblikah
  - Shannonov izrek o razširitvi preklopne funkcije
  - Dvonivojska kombinacijska vezja
- **Funkcijsko polni sistemi preklopnih funkcij**
  - Zaprti razredi in polni sistemi
  - Shefferjev in Pierceov funkcijsko poln sistem

Definicija Boolove algebre: Huntingtonovi postulati

Boolova algebra predstavlja teoretično osnovo za načrtovanje logičnih vezij. Oče Boolove algebre je angleški matematik George Boole (1815-1864), ki je leta 1854 s knjigo "The Laws of Thought" postavil matematično teorijo o logičnem sklepanju.

Algebra $\mathscr{A}$ je matematična teorija $\mathscr{A} = (K, \Phi)$, ki vsebuje množico elementov K in množico operacij Φ, ki delujejo na elementih iz množice K.

Primera algeber sta

- $\mathscr{R} = (R, \Phi)$, kjer je R množica realnih števil in $\Phi = \{+, -, \times, /\}$ množica z osnovnimi računskimi operacijami, in
- $\mathscr{M} = (Z_m, \Phi)$, kjer je Z_m množica prvih m nenegativnih celih števil (0, 1, ..., m-1) in Φ množica osnovnih računskih operacij po modulu m.

Aksiomi ali postulati definirajo osnovne lastnosti algebre, iz katerih lahko izpeljemo vse druge lastnosti.

Boolovo algebro v popolnosti in na jedrnat način definirajo t.i. Huntingtonovi postulati oz. aksiomi, ki jih je leta 1904 objavil ameriški matematik Edward V. Huntington (1874-1952).

Definicija Boolove algebre: Huntingtonovi postulati

Definicija. **Boolova algebra $\mathscr{B}$ je dvojica (K, Φ), kjer je K = {a, b, c, ...} množica elementov in Φ = {•, +, ' } množica operacij* z naslednjimi lastnostmi:**

Št. postulata	Trditev postulata	Ime
P1a	$\forall a,b \in K : a+b \in K$	Zaprtost
P1b	$\forall a,b \in K : ab \in K$	
P2a	$\forall a \in K, \exists 0 \in K : a+0 = a$	Nevtralni element 0
P2b	$\forall a \in K, \exists 1 \in K : a1 = a$	Nevtralni element 1
P3a	$\forall a,b \in K : a+b = b+a$	Komutativnost
P3b	$\forall a,b \in K : ab = ba$	
P4a	$\forall a,b,c \in K : a+bc = (a+b)(a+c)$	Distributivnost
P4b	$\forall a,b,c \in K : a(b+c) = ab+ac$	
P5a	$\forall a \in K, \exists a' \in K : a+a' = 1$	Nasprotni element
P5b	$\forall a \in K, \exists a' \in K : aa' = 0$	
P6	V K sta vsaj dva različna elementa. ■	

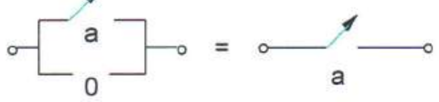
* Operacije •, + in ' imenujemo AND, OR in NOT. AND pogosto označujemo brez znaka • samo s pisanjem argumentov drug poleg drugega. NOT včasih označujemo s ‾.

Definicija Boolove algebre: Huntingtonovi postulati

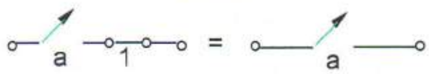
Interpretacija Huntingtonovih postulatov s stikali

Izrazu a+b ustreza vzporedna vezava stikal a in b, izrazu ab zaporedna vezava, izrazu a' pa normalno sklenjeno stikalo.

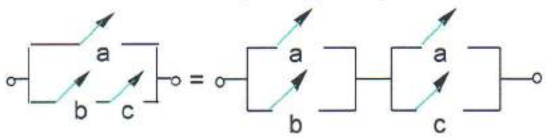
P2a: Nevtralni element 0
a+0=a



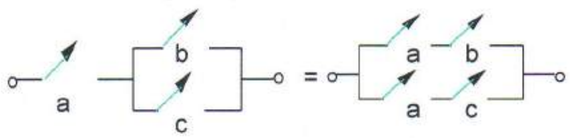
P2b: Nevtralni element 1
a1=a



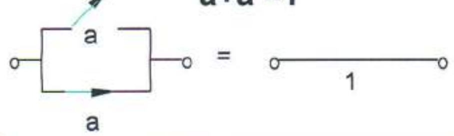
P4a: Distributivnost
a+bc=(a+b)(a+c)



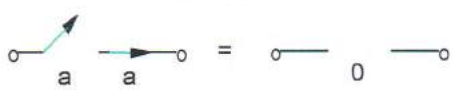
P4b: Distributivnost
a(b+c)=ab+ac



P5a: Nasprotni element
a+a'=1



P5b: Nasprotni element
aa'=0



Definicija Boolove algebre: Preklopna algebra

Boolova algebra je definirana z množico elementov K in z množico operacij $\Phi = \{\bullet, +, '\}$. Po postulatu P6 je najmanjša možna Boolova algebra tista, ki vsebuje samo dva elementa: $K = \{0, 1\}$. Označujemo jo z $\mathscr{B}_2$ in imenujemo *preklopna algebra*.

Operacije $\bullet$, + in ' v $\mathscr{B}_2$ so definirane z naslednjo pravilnostno tabelo:

a	b	ab	a+b	a'
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Hitro lahko preverimo, da $\mathscr{B}_2$ zadovoljuje vse Huntingtonove postulate in je zato Boolova algebra.

Definicija Boolove algebre

Primer: Ali je algebra ξ_3 z elementi 0, 1 in X (nedoločena vrednost) Boolova?

Pravilnostna tabela, ki definira operacije $\bullet$, + in ' v ξ_3:

a	b	ab	a+b	a'
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0
0	X	0	X	1
1	X	X	1	0
X	0	0	X	X
X	1	X	1	X
X	X	X	X	X

K= {0, 1, X}

Dokaz:

Hitro lahko dokažemo, da so v ξ_3 izpolnjeni vsi Huntingtonovi postulati razen postulatov P5a in P5b.

Definicija Boolove algebre

Dokaz, da postulata P5a in P5b ne držita:

Izberemo a=X in vstavimo v P5a in P5b.

P5a:	X+X'	=	
	X+X	=	(definicija operatorja ' iz tabele)
	X	=	(zadnja vrstica v tabeli)
	1		(P5a)

P5b:	XX'	=	
	XX	=	(definicija operatorja ' iz tabele)
	X	=	(zadnja vrstica v tabeli)
	0		(P5b)

Sklepati moramo, da X=0=1, torej ξ_3 vsebuje samo en element, kar je v protislovju s postulatom P6. ξ_3 torej ni Boolova algebra.

Boolove funkcije: Definicija Boolove in preklopne funkcije

Definicija. **Imejmo Boolovo algebro $\mathscr{B}$ = (K, Φ). Funkcijo $f(x_1, x_2, ..., x_n)$, ki preslikuje elemente $x_i \in K$, $1 \leq i \leq n$, v element $f(x_1, x_2, ..., x_n) \in K$, imenujemo *Boolova funkcija*. ■**

Za K = {0, 1} splošna Boolova algebra preide v preklopno algebro $\mathscr{B}_2$ in Boolove funkcije v *preklopne funkcije*. Sinonimi za preklopno funkcijo $f(x_1, x_2, ..., x_n)$ so:

- ***logična funkcija*** **(preklopna algebra je algebra logike),**
- ***dvojiška funkcija*** **(preklopna algebra sodi v dvojiško logiko),**
- ***odločitvena funkcija*** **(odločanje tipa "da-ne"),**
- ***izjavna funkcija*** **(spremenljivke x_i so izjave).**

V naslednjih poglavjih nas bodo zanimale predvsem preklopne funkcije.

Če je iz konteksta razvidno, da se obravnava omejuje zgolj na preklopne funkcije (torej Boolove funkcije iz $\mathscr{B}_2$), so v marsikateri literaturi preklopne funkcije imenovane kar Boolove funkcije.

Boolove funkcije: Pravilnostna tabela

Boolovo funkcijo $f(x_1, x_2, ..., x_n)$ velikokrat definiramo s *pravilnostno tabelo.* V njej so na levi strani navedene vse možne kombinacije vrednosti vhodnih spremenljivk $x_1, x_2, ..., x_n$, na desni strani pa so podane ustrezne vrednosti funkcije. Pravilnostna tabela za preklopno funkcijo iz $\mathscr{B}_2$ ima 2^n vrstic:

$x_1\ x_2\ ...\ x_n$	$f(x_1, x_2, ..., x_n)$
w_0	$f(w_0)$
w_1	$f(w_1)$
.	.
.	.
.	.
w_{2^n-2}	$f(w_{2^n-2})$
w_{2^n-1}	$f(w_{2^n-1})$

V i-ti vrstici pravilnostne tabele je na levi strani vektor (kodna beseda) $w_i = (w_{i1}, w_{i2}, ..., w_{in})$, kjer je $w_{ij} \in \{0, 1\}$, $1 \leq j \leq n$, vrednost spremenljivke x_j v i-ti vrstici, na desni strani pa funkcijska vrednost $f(w_i) \in \{0, 1\}$.

Značilnost preklopne funkcije je v tem, da množico vhodnih kombinacij oz. vhodnih vektorjev razdeli v dve podmnožici: za prvo (množica "OFF-set") je značilna funkcijska vrednost 0, za drugo (množica "ON-set") pa funkcijska vrednost 1.

Boolove funkcije: Nepopolno specificirane funkcije

Funkcija z n spremenljivkami ima 2^n možnih vhodnih kombinacij.

Za podano funkcijo ni nujno, da so vse vhodne kombinacije sploh možne.

To dejstvo izkoristimo pri minimizaciji vezja!

Primer: Krožni BCD števnik
BCD števke zakodirajo decimalne števke 0-9 v bitne vzorce 0000_2-1001_2.

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

množica "OFF-set" funkcije W

množica "ON-set" funkcije W

množica "don't care" (DC) funkcije W

Ti vhodni vzorci se v praksi nikoli ne pojavijo, zato so pripadajoče izhodne vrednosti nepomembne ("don't care"). Označimo jih z X.

Boolove funkcije: Boolovi izrazi

Definicija. Imejmo Boolovo algebro $\mathcal{B}$ = (K, Φ). Elementu množice K pravimo *Boolova konstanta* v $\mathcal{B}$. Simbolu, ki lahko predstavlja katerikoli element iz množice K, pravimo *Boolova spremenljivka.*

Izraz E imenujemo *Boolov izraz*, če je zgrajen po naslednjih pravilih:

1. Vsaka Boolova konstanta ali spremenljivka je Boolov izraz.
2. Če je E_1 Boolov izraz, je Boolov izraz tudi E'_1.
3. Če sta E_1 in E_2 Boolova izraza, sta Boolova izraza tudi $E_1 + E_2$ in E_1E_2.
4. Vsak izraz, ki ga lahko zgradimo s končnim številom aplikacij zgornjih pravil, in samo tak izraz, je Boolov izraz.

Prioriteto operatorjev določimo z oklepaji. Če oklepajev ni, se najprej izvede operacija ' , nato • in nazadnje +. ■

Boolov izraz je alternativni način za definiranje Boolove funkcije. Navadno je precej krajši od specifikacije Boolove funkcije s pravilnostno tabelo.

Vsak Boolov izraz definira edinstveno Boolovo funkcijo, Boolova funkcija pa je lahko definirana z mnogimi po zgradbi različnimi, vendar *ekvivalentnimi* Boolovimi izrazi.

Osnovne lastnosti Boolove algebre: Dualnost

Definicija. Naj bo E Boolov izraz. Izraz E^d, ki ga dobimo iz E, če vsak • zamenjamo s +, vsak + z •, vsako 0 z 1 in vsako 1 z 0, imenujemo *dualni* izraz E-ja:

$$\{f(x_1, x_2, ..., x_n, 0, 1, +, \cdot)\}^d = \{f(x_1, x_2, ..., x_n, 1, 0, \cdot, +)\} \blacksquare$$

Primer: $E = ((x_1+x_2)(x'_3))$, $E^d = ((x_1x_2)+(x'_3))$.

Izrek. Dualni izraz k E^d je spet originalni izraz E: $(E^d)^d = E$. ■

Definicija. Če velja $E^d = E$ (torej E in E^d definirata isto funkcijo), je Boolov izraz E *sebi dualen.* ■

Princip dualnosti

Izrek. Če je $E_1 = E_2$ veljavna enačba med Boolovima izrazoma, potem je tudi $E_1^d = E_2^d$ veljavna enačba Boolove algebre. ■

Večina obravnavanih postulatov in izrekov nastopa v dualnih parih.

Osnovne lastnosti Boolove algebre

Izreki Boolove algebre

V algebri so izreki veljavne izjave, ki jih lahko izpeljemo iz postulatov oz. aksiomov.

Izrek. Nekateri koristni izreki Boolove algebre so:

Št. izreka	Trditev izreka	Ime izreka
I7a	Element 0 je edinstven.	Edinstvenost
I7b	Element 1 je edinstven.	elementov 0 in 1
I8a	$\forall a \in K : a+a = a$	Idempotenca
I8b	$\forall a \in K : aa = a$	
I9a	$\forall a \in K : a+1 = 1$	Identiteta
I9b	$\forall a \in K : a0 = 0$	
I10a	$\forall a,b \in K : a+ab = a$	Absorbcija
I10b	$\forall a,b \in K : a(a+b) = a$	
I11	$\forall a \in K$: Nasprotni element a' je edinstven.	Edinstvenost nasprotnega elementa

Osnovne lastnosti Boolove algebre

Št. izreka	Trditev izreka	Ime izreka
I12a	$\forall a,b,c \in K : (a+b)+c = a+(b+c)$	Asociativnost
I12b	$\forall a,b,c \in K : (ab)c = a(bc)$	
I13a	$\forall a,b \in K : (a+b)' = a'b'$	DeMorganova
I13b	$\forall a,b \in K : (ab)' = a'+b'$	izreka
I14	$\forall a \in K : (a')' = a$	Involucija
I15a	$\forall a \in K : a+a' = 1$	Komplementarnost
I15b	$\forall a \in K : aa' = 0$	
I16a	$\forall a,b \in K : ab+ab' = a$	Sosednost
I16b	$\forall a,b \in K : (a+b)(a+b') = a$	
I17a	$\forall a,b,c \in K : ab+a'c+bc = ab+a'c$	Izrek o konsenzu
I17b	$\forall a,b,c \in K : (a+b)(a'+c)(b+c) = (a+b)(a'+c)$	

Nekatere od zgornjih izrekov lahko posplošimo na večje število spremenljivk. Npr., DeMorganova izreka za n spremenljivk se glasita:

$(x_1 + x_2 + \ldots + x_n)' = x'_1 x'_2 \ldots x'_n$

$(x_1 x_2 \ldots x_n)' = x'_1 + x'_2 + \ldots + x'_n$ ■

Osnovne lastnosti Boolove algebre

Dokazovanje izrekov

Večina izrekov Boolove algebre ima obliko $E_1 = E_2$, pri čemer sta E_1 in E_2 Boolova izraza, sestavljena iz spremenljivk in konstant iz K in operatorjev iz Φ. Veljavnost takih izrekov lahko neformalno preverimo s primerjavo stikalnih vezij ali pravilnostnih tabel za E_1 in E_2.

Neformalno preverjanje DeMorganovih izrekov s pravilnostno tabelo

$(X + Y)' = X' \cdot Y'$

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X+Y}$	$\overline{X}\cdot\overline{Y}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

$(X \cdot Y)' = X' + Y'$

X	Y	$\overline{X}$	$\overline{Y}$	$\overline{X\cdot Y}$	$\overline{X}+\overline{Y}$
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Osnovne lastnosti Boolove algebre

Dokazovanje izrekov

Formalno dokažemo veljavnost izreka z algebrskim dokazovanjem. Tak dokaz ima obliko zaporedja korakov $T_1, T_2, ..., T_p$, kjer vsak korak T_i specificira trditev, katere veljavnost je odvisna bodisi od uporabe postulatov bodisi od prej dokazanega izreka na izrazu v koraku T_{i-1}.

Formalni dokaz izreka I7a:

I7a: Element 0 Boolove algebre je edinstven.

Dokaz: Dokazujemo z redukcijo do protislovja. Predpostavimo, da obstajata dva različna elementa 0: 0_1 in 0_2.

$a+0_1 = a$ (P2a)
$a+0_2 = a$ (P2a)

V prvo enačbo vstavimo $a = 0_2$ in v drugo $a = 0_1$.

$0_2+0_1 = 0_2$
$0_1+0_2 = 0_1$

Po postulatu P3a velja $0_2+0_1 = 0_1+0_2$, zato $0_1 = 0_2$, kar je v protislovju z začetno hipotezo, da sta 0_1 in 0_2 različna. V Boolovi algebri je lahko torej samo en element 0.

Osnovne lastnosti Boolove algebre

Dokazovanje izrekov z aksiomi Boolove algebre

Npr., dokažite izrek o absorbciji (I10a): $X + X \cdot Y = X$.

Npr., dokažite izrek o sosednosti (I16a): $X \cdot Y + X \cdot Y' = X$.

Osnovne lastnosti Boolove algebre

Dokazovanje izrekov z aksiomi Boolove algebre

Npr., dokažite izrek o absorbciji (I10a): $X + X \cdot Y = X$.

nevtralni element 1 (P2b)	$X + X \cdot Y = X \cdot 1 + X \cdot Y$
distributivnost (P4b)	$X \cdot 1 + X \cdot Y = X \cdot (1 + Y)$
komutativnost (P3a)	$X \cdot (1 + Y) = X \cdot (Y + 1)$
identiteta (I9a)	$X \cdot (Y + 1) = X \cdot (1)$
nevtralni element 1 (P2b)	$X \cdot (1) = X$

Npr., dokažite izrek o sosednosti (I16a): $X \cdot Y + X \cdot Y' = X$.

distributivnost (P4b)	$X \cdot Y + X \cdot Y' = X \cdot (Y + Y')$
komplementarnost (I15a)	$X \cdot (Y + Y') = X \cdot (1)$
nevtralni element 1 (P2b)	$X \cdot (1) = X$

Osnovne lastnosti Boolove algebre

Uporaba postulatov in izrekov za poenostavljanje Boolovih izrazov

Primer: Funkcija za izhodni prenos popolnega seštevalnika

Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin

Osnovne lastnosti Boolove algebre

Uporaba postulatov in izrekov za poenostavljanje Boolovih izrazov

Primer: Funkcija za izhodni prenos popolnega seštevalnika

idempotenca

Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin

= A' B Cin + A B' Cin + A B Cin' + $\boxed{\text{A B Cin + A B Cin}}$

= A' B Cin + A B Cin + A B' Cin + A B Cin' + A B Cin

= (A' + A) B Cin + A B' Cin + A B Cin' + A B Cin

= (1) B Cin + A B' Cin + A B Cin' + A B Cin

= B Cin + A B' Cin + A B Cin' + $\boxed{\text{A B Cin + A B Cin}}$

= B Cin + A B' Cin + A B Cin + A B Cin' + A B Cin

= B Cin + A (B' + B) Cin + A B Cin' + A B Cin

asociativnost

= B Cin + A (1) Cin + A B Cin' + A B Cin

= B Cin + A Cin + A B (Cin' + Cin)

= B Cin + A Cin + A B (1)

= B Cin + A Cin + A B

Druge Boolove algebre

Domena	Elementi	Operacije
Izjavni račun	K = vse pravilne ali nepravilne izjave (n spremenljivk) 0 = vedno nepravilna izjava (protislovje) 1 = vedno pravilna izjava (tavtologija)	• = veznik in ($\wedge$) + = veznik ali ($\vee$) ' = veznik ne ($\neg$)
Teorija množic	K = vseh 2^n podmnožic množice z n elementi 0 = prazna množica $\varnothing$ 1 = univerzalna množica U z vsemi n elementi	• = presek $\cap$ + = unija $\cup$ ' = komplement $\bar{\ }$
Logična vezja	K = vseh 2^{2^n} binarnih (logičnih) funkcij f(X) z n spremenljivkami 0 = funkcija konstante $f_Z(X) = 0$ za vsak X 1 = funkcija konstante $f_U(X) = 1$ za vsak X	• = funkcija AND + = funkcija OR ' = funkcija NOT

Druge Boolove algebre: Izjavni račun

Definicija. Boolova algebra $\mathscr{P} = (K, \Phi)$, v kateri je K množica vseh pravilnih ali nepravilnih izjav in Φ množica veznikov {in, ali, ne}, se imenuje *izjavni račun*. Element 0 (1) v izjavnem računu je izjava, ki je nepravilna (pravilna) pri vseh kombinacijah vhodnih spremenljivk (osnovnih izjav). V matematičnih izrazih izjave označujemo s spremenljivkami (a, b, c, ...), veznike (in, ali, ne) pa z operacijami $\wedge$ ***(konjunkcija),*** $\vee$ ***(disjunkcija)*** in $\neg$ ***(negacija).*** ■

S pomočjo operacij konjunkcije, disjunkcije in negacije lahko iz danih osnovnih izjav dobimo sestavljene izjave.

Primer:

a: lačen sem   b: žejen sem

$z_1 = a \wedge b$: lačen sem in žejen sem

$z_2 = a \vee b$: lačen sem ali žejen sem

$z_3 = \neg a$: nisem lačen

$z_4 = \neg b$: nisem žejen

Izjava z_1 je pravilna natanko tedaj, kadar je pravilna tako izjava a kot tudi izjava b. V vseh drugih primerih je izjava z_1 nepravilna.

Izjava z_2 je nepravilna natanko tedaj, kadar je nepravilna tako izjava a kot tudi izjava b. Če je pravilna vsaj ena od izjav a, b, je izjava z_2 pravilna.

Izjava z_3 (z_4) je pravilna, če je izjava a (b) nepravilna.

Druge Boolove algebre: Izjavni račun

Tudi v izjavnem računu uporabimo pravilnostne tabele za predstavitev sestavljenih izjav. Vsaka izjava (osnovna ali sestavljena) ima lahko vrednost false (izjava je nepravilna) ali vrednost true (izjava je pravilna).

a	b	$a \wedge b$	$a \vee b$	$\neg a$	$\neg b$
false	false	false	false	true	true
false	true	false	true	true	false
true	false	false	true	false	true
true	true	true	true	false	false

Poleg osnovnih operacij v izjavnem računu veliko uporabljamo tudi operacijo $\rightarrow$ (implikacija) in $\equiv$ (ekvivalenca).

a	b	$a \rightarrow b$	$a \equiv b$
false	false	true	true
false	true	true	false
true	false	false	false
true	true	true	true

$a \rightarrow b = \neg a \vee b$

$a \equiv b = (\neg a \wedge \neg b) \vee (a \wedge b)$

$(a \rightarrow b) \wedge (b \rightarrow a) = a \equiv b$

Primer sestavljene izjave z implikacijo: a: sin opravi izpit
b: oče kupi sinu kolo

Sestavimo novo izjavo $z = a \rightarrow b$: če sin opravi izpit, mu oče kupi kolo. Izjava z je nepravilna samo v primeru, ko sin opravi izpit, oče pa mu ne kupi kolesa (če oče ni mož beseda).

Druge Boolove algebre: Algebra množic

Dokažemo lahko, da množica K vseh mogočih podmnožic U-ja z operacijami $\cap$, $\cup$ in $^-$ zadovoljuje Huntingtonove postulate. To Boolovo algebro označujemo s $\mathscr{S}_{2^n}$ in jo imenujemo *algebra množic.*

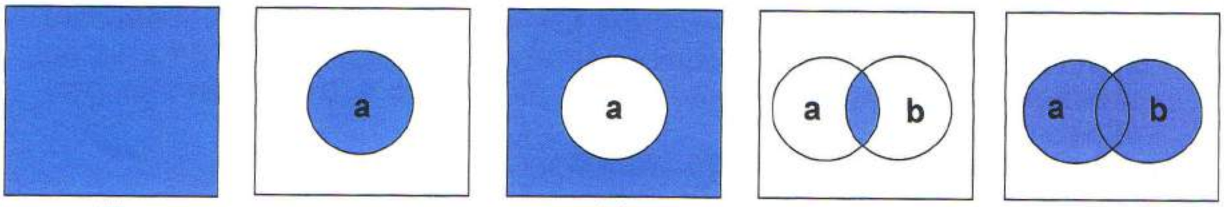
Množica $Z_n = \{0, 1, ..., n-1\}$ z n elementi ima 2^n podmnožic, torej ima vsaka algebra množic natanko 2^n elementov. Npr., $Z_2 = \{0, 1\}$ ima naslednje štiri podmnožice: $\varnothing$, $\{0\}$, $\{1\}$, $\{0, 1\} = U$ in definira algebro množic $\mathscr{S}_4$ s štirimi elementi.

Vsaka Boolova algebra $\mathscr{B}$ je ekvivalentna neki algebri množic, zato mora biti število elementov |K| v $\mathscr{B}$ enako potenci števila 2. Edine možne vrednosti za |K| so torej 2, 4, 8, 16 itn. To dejstvo takoj pomeni, da algebra ξ_3 s tremi elementi ne more biti Boolova.

Predstavitev Boolove algebre z Vennovimi diagrami

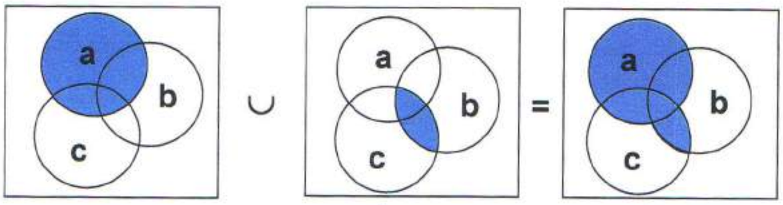
Algebre množic in druge Boolove algebre lahko predstavimo grafično s t.i. *Vennovimi diagrami* (po angleškem logiku Johnu Vennu (1834-1923)).

Druge Boolove algebre: Algebra množic

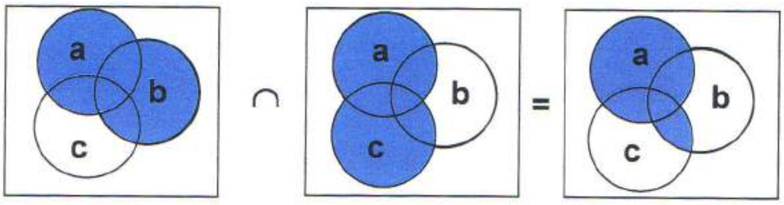


U | a | $\overline{a}$ | $a \cap b$ | $a \cup b$

Vennovi diagrami za predstavitev postulata P4a



a | $b \cap c$ | $a \cup (b \cap c)$



$a \cup b$ | $a \cup c$ | $(a \cup b) \cap (a \cup c)$

Druge Boolove algebre: Boolova funkcijska algebra

Naj bo F_n množica vseh Boolovih funkcij nad n spremenljivkami. Vsako tako funkcijo lahko definiramo z edinstveno 2^n-vrstično pravilnostno tabelo. Obstaja 2^{2^n} različnih izhodnih stolpcev za 2^n-vrstično pravilnostno tabelo, torej ima F_n 2^{2^n} elementov (funkcij): $\{f_0, f_1, f_2, ..., f_{2^{2^n}-1}\}$.

Definicija. Na poljubnih funkcijah $f_1(X)$ in $f_2(X)$, $X = (x_1, x_2, ..., x_n)$, iz množice F_n definiramo operacije iz množice $\Phi = \{\bullet, +, '\}$ na naslednji način:

$f_1(X) \bullet f_2(X) = 1$, če in samo če $f_1(X) = 1$ in $f_2(X) = 1$,

$f_1(X) + f_2(X) = 1$, če in samo če $f_1(X) = 1$ ali $f_2(X) = 1$,

$f'_1(X) = 1$, če in samo če $f_1(X) = 0$. ■

Preprosto se da dokazati, da algebra $\mathscr{F}_n = (F_n, \Phi)$ izpolnjuje vse Huntingtonove postulate. Imenujemo jo *Boolova funkcijska algebra*.

Element 0 v $\mathscr{F}_n$ je funkcija f_Z, ki vsak X preslika v 0: $f_Z(X) = 0$.

Element 1 v $\mathscr{F}_n$ je funkcija f_U, ki vsak X preslika v 1: $f_U(X) = 1$.

Druge Boolove algebre: Boolova funkcijska algebra

n=0: $\mathscr{F}_0 = (F_0, \Phi)$

$F_0 = \{f_Z, f_U\} = \{0, 1\}$

$\mathscr{F}_0$ je preklopna algebra $\mathscr{B}_2$, v kateri sta samo dve funkciji brez spremenljivk (konstanti 0 in 1).

n=1: $\mathscr{F}_1 = (F_1, \Phi)$

$F_1 = \{f_Z = f_0, f_1, f_2, f_3 = f_U\}$

x_1	0 1	ime in oznaka funkcije	
f_0	0 0	konstanta 0	0
f_1	0 1	identiteta	x_1
f_2	1 0	negacija (NOT)	x'_1
f_3	1 1	konstanta 1	1

Opomba: Ker ime in oznako funkcije lažje zapišemo v vrstici kot v stolpcu, je glede na običajni zapis pravilnostne tabele v zgornji pravilnostni tabeli vloga vrstic in stolpcev zamenjana.

Druge Boolove algebre: Boolova funkcijska algebra

n=2: $\mathscr{F}_2 = (F_2, \Phi)$

$F_2 = \{f_Z = f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, f_{15} = f_U\}$

x_1 x_2	0 0 1 1 0 1 0 1	ime in oznaka funkcije		izražava z osnovnimi operatorji
f_0	0 0 0 0	konstanta 0	0	0
f_1	0 0 0 1	konjunkcija (AND)	x_1x_2	x_1x_2
f_2	0 0 1 0	negacija implikacije	$(x_1 \rightarrow x_2)'$	$x_1\, x'_2$
f_3	0 0 1 1	projekcija	x_1	x_1
f_4	0 1 0 0	negacija implikacije	$(x_2 \rightarrow x_1)'$	x'_1x_2
f_5	0 1 0 1	projekcija	x_2	x_2
f_6	0 1 1 0	ekskluzivni OR (XOR)	$x_1 \oplus x_2$	$x'_1x_2 + x_1\, x'_2$
f_7	0 1 1 1	disjunkcija (OR)	$x_1 + x_2$	$x_1 + x_2$
f_8	1 0 0 0	Pierce (NOR)	$x_1 \downarrow x_2$	$(x_1 + x_2)'$
f_9	1 0 0 1	ekvivalenca (XNOR)	$x_1 \equiv x_2$	$x'_1x'_2 + x_1x_2$
f_{10}	1 0 1 0	negacija (NOT)	x'_2	x'_2
f_{11}	1 0 1 1	implikacija	$x_2 \rightarrow x_1$	$x'_2 + x_1$
f_{12}	1 1 0 0	negacija (NOT)	x'_1	x'_1
f_{13}	1 1 0 1	implikacija	$x_1 \rightarrow x_2$	$x'_1 + x_2$
f_{14}	1 1 1 0	Scheffer (NAND)	$x_1 \mid x_2$	$(x_1x_2)'$
f_{15}	1 1 1 1	konstanta 1	1	1

Druge Boolove algebre: Boolova funkcijska algebra

Funkcije f_0, f_1, f_3, f_5, f_7, f_{10}, f_{12} in f_{15} iz $\mathcal{F}_2$ so *osnovne*, saj eksplicitno nastopajo v Huntingtonovih postulatih, vse druge funkcije so glede na postulate implicitne, zato jih imenujemo *psevdoosnovne*.

Vsako psevdoosnovno funkcijo iz $\mathcal{F}_2$ lahko izrazimo z osnovnimi funkcijami, kar je razvidno iz zadnjega stolpca v pravilnostni tabeli za $\mathcal{F}_2$.

***Definicija.* Če je preklopna funkcija $f(x_1, x_2, ..., x_n)$ odvisna od vseh spremenljivk x_i, $1 \le i \le n$, ji pravimo *nedegenerirana* funkcija, če pa ni odvisna od vseh spremenljivk, ji pravimo *degenerirana* funkcija. ■**

Funkcije f_0, f_3, f_5, f_{10}, f_{12} in f_{15} iz $\mathcal{F}_2$ so degenerirane funkcije dveh spremenljivk, ker so odvisne od manj kot dveh spremenljivk. Preostale funkcije so nedegenerirane funkcije dveh spremenljivk, ker so od obeh spremenljivk dejansko odvisne.

Število funkcij z večanjem števila vhodnih spremenljivk (n) izredno hitro raste. Npr., za tri vhodne spremenljivke obstaja 256 funkcij, za štiri 65 536, za pet pa že kar astronomskih 4 294 967 296 funkcij.

Preklopna vezja: Logična vrata

Mnoge funkcije iz algebre $\mathcal{F}_n$ v elektroniki implementiramo z osnovnimi gradniki–n-vhodnimi *logičnimi vrati*. Obstajajo naslednji tipi logičnih vrat: neinvertirajoči vmesnik, NOT, AND, OR, NAND, NOR, XOR in XNOR.

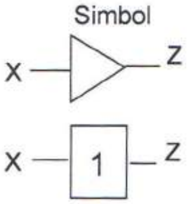
Zaradi preprostosti bomo predstavili logična vrata, ki realizirajo funkcije iz algebre $\mathcal{F}_2$ (največ dva vhoda). Uporabljajo se tudi vrata z večjim številom vhodov, ki realizirajo zgoraj omenjene funkcije v $\mathcal{F}_n$, $n > 2$. Tehnološke omejitve pri izdelavi vrat postavljajo za n določeno zgornjo mejo.

Za vsaka logična vrata podajamo njihovo ime, opis delovanja, grafični simbol (ameriški simbol in škatlasti simbol po standardu IEC), pravilnostno tabelo in ekvivalentno stikalno vezje. V nadaljevanju bomo uporabljali ameriške grafične simbole za logična vrata, ker se ti najpogosteje uporabljajo v strokovni literaturi in v orodjih za računalniško podprto načrtovanje.

Ime: **Neinvertirajoči vmesnik**, $f_3, f_5 \in F_2$

Opis: Z je enak X-u.

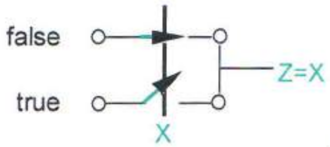
Simbol:



Pravilnostna tabela:

X	Z
0	0
1	1

Stikalno vezje:



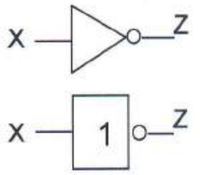
Preklopna vezja: Logična vrata

Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

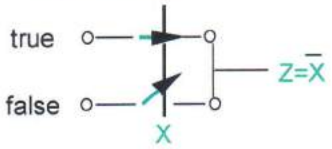
NOT

$f_{10}, f_{12} \in F_2$

Če X=0, potem Z=1.
Če X=1, potem Z=0.



X	Z
0	1
1	0

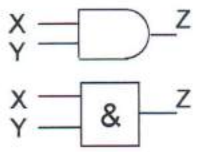


Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

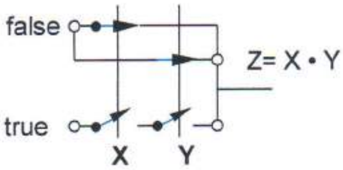
AND

$f_1 \in F_2$

Z = 1, če sta X in Y oba 1.



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

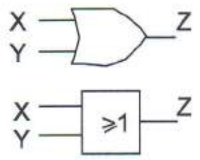


Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

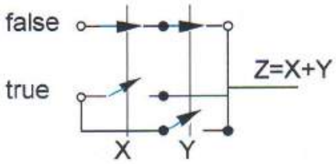
OR

$f_7 \in F_2$

Z = 1, če X=1 ali Y=1.



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



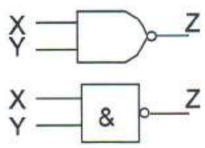
Preklopna vezja: Logična vrata

Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

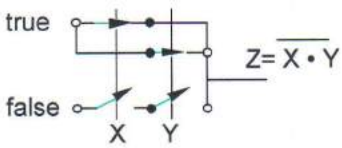
NAND

$f_{14} \in F_2$

Z = 1, če X=0 ali Y=0.



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

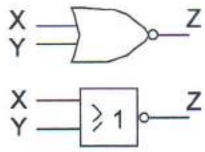


Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

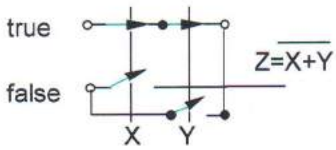
NOR

$f_8 \in F_2$

Z = 1, če je X=0 in Y=0.



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



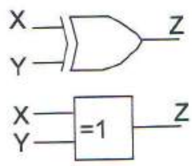
Preklopna vezja: Logična vrata

Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

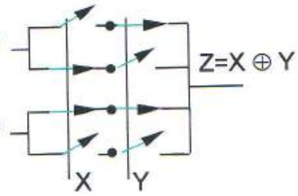
XOR

$f_6 \in F_2$

Z = 1, če ima X drugačno vrednost kot Y.



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

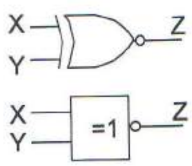


Ime | Opis | Simbol | Pravilnostna tabela | Stikalno vezje

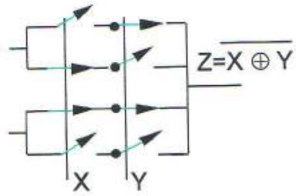
XNOR

$f_9 \in F_2$

Z = 1, če ima X enako vrednost kot Y.

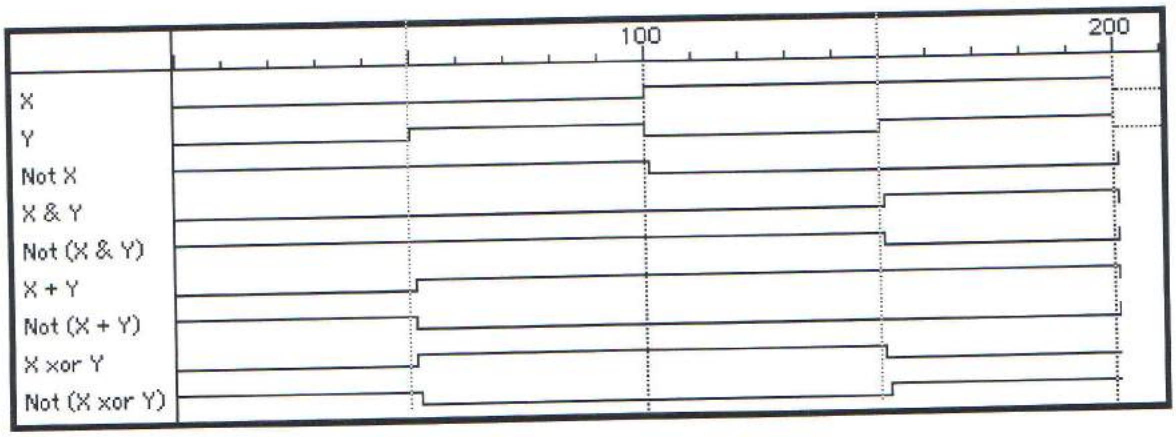


X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1



Preklopna vezja: Logična vrata

Časovni poteki signalov za logična vrata



Preklopna vezja: Kombinacijska preklopna vezja

Definicija. Kombinacijsko vezje z n vhodi in m izhodi je preklopno vezje, sestavljeno iz več logičnih vrat, ki realizira m-terico funkcij ($m \geq 1$):

$$F = (f_1(X), f_2(X), ..., f_m(X)),$$

kjer je $X = (x_1, x_2, ..., x_n)$ in vsak f_i , $1 \leq i \leq m$, preklopna funkcija, ki preslikuje

$$f_i\colon \{0,1\}^n \rightarrow \{0,1\}. \blacksquare$$

Idealna kombinacijska vezja na vhodne spremembe odgovarjajo trenutno. V realnih kombinacijskih vezjih obstaja zmeraj neka kratka zakasnitev, preden se pojavi odziv.

Pri sestavljanju logičnih vrat v kombinacijska logična vezja moramo upoštevati določena pravila, da dobimo dobro sestavljena vezja. Dobro sestavljena kombinacijska vezja ne smejo vsebovati kratko povezanih izhodov med logičnimi vrati niti sklenjenih zank.

Osrednji problem načrtovanja logike je sestaviti kombinacijsko vezje, ki realizira podano m-terico funkcij, uporablja podane tipe vrat in ima najnižjo možno ceno.

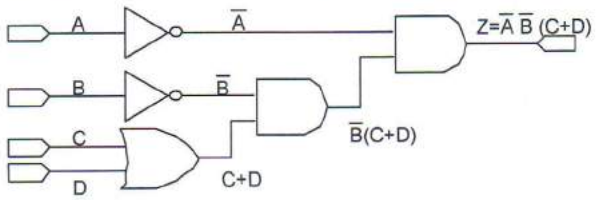
Preklopna vezja: Kombinacijska preklopna vezja

Algebra $\mathscr{F}_n$ je osnovna algebra za načrtovanje logičnih vezij.

Z n-vhodnim kombinacijskim logičnim vezjem, sestavljenim iz logičnih vrat AND, OR in NOT, direktno implementiramo množico preklopnih funkcij z n spremenljivkami. Dejansko vsako vozlišče v vezju realizira neko funkcijo iz Boolove algebre $\mathscr{F}_n$.

Vsaka vrata kombinirajo preklopne funkcije, ki se pojavijo na vhodih vrat, v novo preklopno funkcijo, ki se pojavi na izhodu vrat. Signale v vezju lahko torej poistovetimo s preklopnimi funkcijami, ki prehajajo skozi vezje od primarnih vhodov k primarnim izhodom in se na tej poti transformirajo v nove funkcije.

Primer kombinacijskega vezja s štirimi spremenljivkami:



Namesto z logičnimi vrati AND, OR in NOT preklopne funkcije pogosto realiziramo samo z vrati NAND ali samo z vrati NOR.

Preklopna vezja: Kombinacijska preklopna vezja

Razlogi za poenostavljanje preklopnih vezij

Minimizacija logike zmanjša kompleksnost preklopnega vezja, ker

- zmanjša število literalov (številov vhodov logičnih vrat),
- zmanjša število logičnih vrat in
- zmanjša število nivojev logičnih vrat v vezju.

Manjše število vhodov v nekaterih tehnologijah implicira hitrejša vrata.

Število vhodov vrat (fan-in) je v nekaterih tehnologijah omejeno.

Manj nivojev vrat implicira manjše propagacijske zakasnitve signala.

Konfiguracija z minimalnimi zakasnitvami signalov zahteva več vrat.

Število vrat (ali čipov z vrati) vpliva na proizvodne stroške.

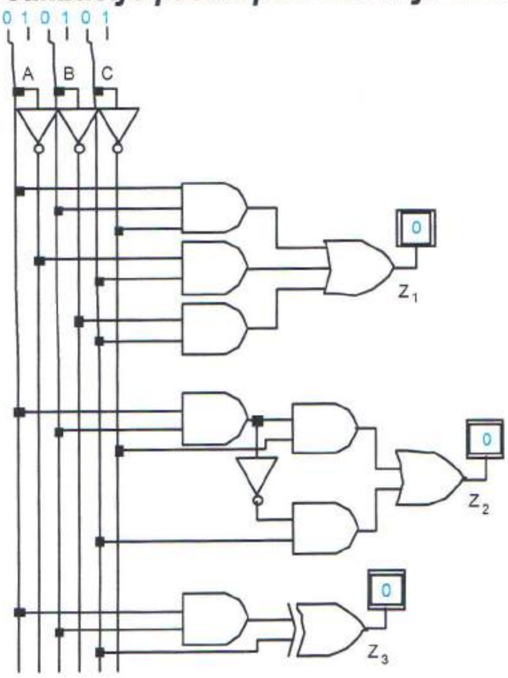
Tradicionalne metode načrtovanja zmanjšujejo zakasnitve na račun dodajanja vrat.

Nove metode načrtovanja ponujajo kompromis med povečanimi zakasnitvami vezja in zmanjšanim številom vrat.

Preklopna vezja: Kombinacijska preklopna vezja

Alternativne realizacije preklopne funkcije z vrati

A	B	C	Z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Dvonivojska realizacija (invertorjev ne štejemo)

Večnivojska realizacija
Prednost: zmanjšano število vhodov vrat

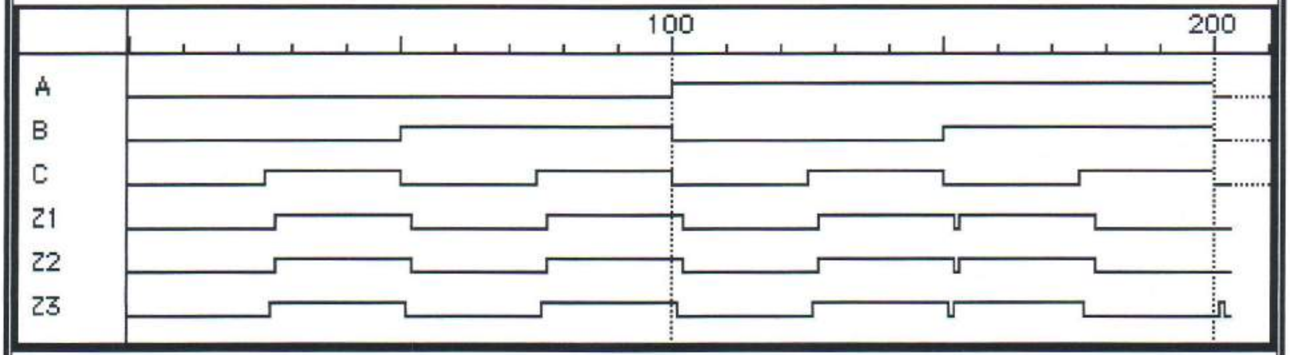
Realizacija z vrati XOR
Prednost: najmanj vrat

Število TTL čipov:

- Z1 - trije čipi (1x 6-invertorjev, 1x 3-vhodni AND, 1x 3-vhodni OR)
- Z2 - trije čipi (1x 6-invertorjev, 1x 2-vhodni AND, 1x 2-vhodni OR)
- Z3 - dva čipa (1x 2-vhodni AND, 1x 2-vhodni XOR)

Preklopna vezja: Kombinacijska preklopna vezja

Primerjava časovnih potekov izhodnih signalov



Pri enakih vhodnih signalih imajo tri alternativne implementacije v bistvu enako izhodno obnašanje.

Malenkostne razlike so posledica različnega števila nivojev logičnih vrat.

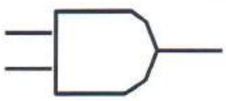
Vse tri implementacije so ekvivalentne.

Preklopna vezja: Pozitivna in negativna logika

Logična vrata so implementirana z elektronskim vezjem, ki dela z napetostnimi, ne pa z logičnimi nivoji. Vsako pravilnostno tabelo, ki opisuje delovanje vrat, lahko zmeraj interpretiramo na dva načina–s *pozitivno logiko* ali *z negativno logiko*.

Pri pozitivni logiki je aktiven visok nivo: **nizka napetost (low) = 0, visoka napetost (high) = 1.**

Pri negativni logiki je aktiven nizek nivo: **nizka napetost (low) = 1, visoka napetost (high) = 0.**



Napetostna pravilnostna tabela

A	B	F
low	low	low
low	high	low
high	low	low
high	high	high

Pozitivna logika

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Negativna logika

A	B	F
1	1	1
1	0	1
0	1	1
0	0	0

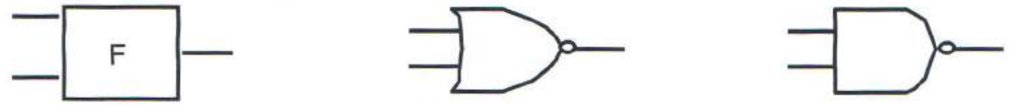
Alternativni interpretaciji

Obnašanje izraženo z električnimi nivoji

Vrata F se interpretirajo kot vrata AND v pozitivni logiki in kot vrata OR v negativni logiki.

Preklopna vezja: Pozitivna in negativna logika

Pretvorba iz pozitivne logike v negativno logiko



Napetostna pravilnostna tabela

A	B	F
low	low	high
low	high	low
high	low	low
high	high	low

Pozitivna logika

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Negativna logika

A	B	F
1	1	0
1	0	1
0	1	1
0	0	1

Vrata F se interpretirajo kot vrata NOR v pozitivni logiki: $\overline{A+B} = \overline{A}\cdot\overline{B}$

Vrata F se interpretirajo kot vrata NAND v negativni logiki: $\overline{A\cdot B} = \overline{A}+\overline{B}$

Pri pretvorbi iz pozitivne logike v negativno logiko medsebojno zamenjamo dualni operaciji AND in OR, komplementi pa ostanejo nespremenjeni.

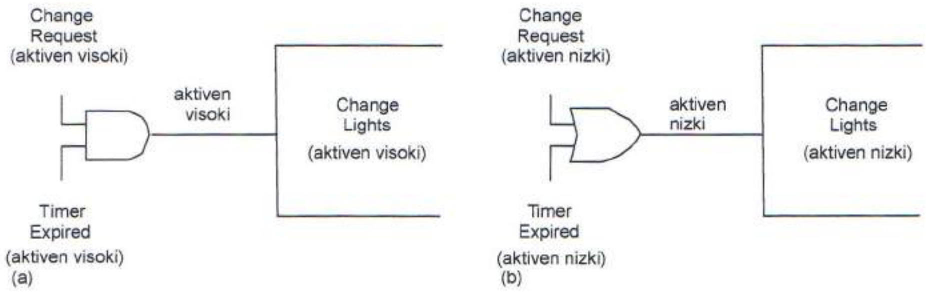
Mešanje pozitivne in negativne logike v vezju lahko pripelje do zmešnjave, zato raje sprejmemo označevanje, ki predpostavlja, da delujejo vsa vrata v pozitivni logiki, eksplicitno pa označimo, ali so signali aktivni visoki ali nizki. Na vhod ali izhod, ki je aktiven nizek, postavimo krožec.

Preklopna vezja: Pozitivna in negativna logika

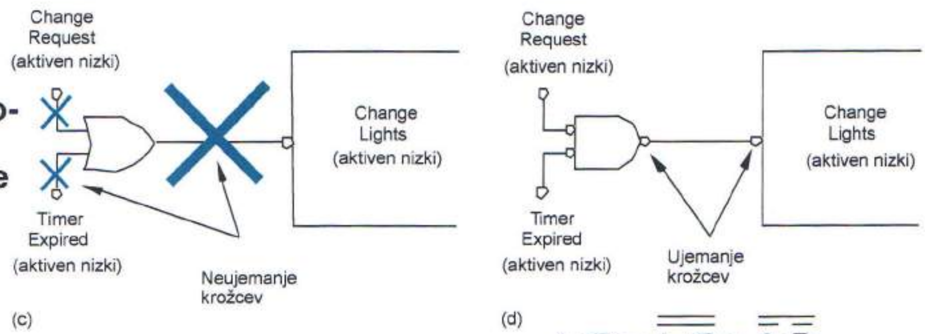
Praktični primer

Če so signali aktivni nizki, uporabimo vrata OR.

Če so signali aktivni visoki, uporabimo vrata AND.



Vhodne in izhodne polaritete signalov se ne ujemajo.



$$A+B = \overline{\overline{A+B}} = \overline{\overline{A}\cdot\overline{B}}$$

Kanonične oblike preklopnih funkcij

Vsaka preklopna funkcija ima edinstveno pravilnostno tabelo, opišemo pa jo lahko z mnogimi alternativnimi (vendar ekvivalentnimi) Boolovimi izrazi.

Koristno je imeti standardno obliko Boolovih izrazov za predstavitev preklopne funkcije. Ta standardna oblika se imenuje *kanonična oblika* in predstavlja edinstven algebrski zapis funkcije.

Obstajata dve glavni kanonični obliki Boolovih izrazov za predstavitev preklopne funkcije:

- *kanonična disjunktivna oblika* ali *oblika vsote mintermov* ali *popolna disjunktivna normalna oblika* (PDNO) in
- *kanonična konjunktivna oblika* ali *oblika produkta makstermov* ali *popolna konjunktivna normalna oblika* (PKNO).

Pred obravnavo kanoničnih oblik preklopne funkcije si definiramo pojem *literal*.

Definicija. Literal Boolove spremenljivke x je bodisi sama spremenljivka x bodisi njen komplement x'. ■

Kanonične oblike preklopnih funkcij: Mintermi

Definicija. Naj bo $m_i(x_1, x_2, ..., x_n)$ preklopna funkcija n spremenljivk, katere pravilnostna tabela ima vrednost 1 v vrstici i, $0 \le i \le 2^n-1$, v vseh ostalih vrsticah pa vrednost 0. Funkciji m_i pravimo i-ta *mintermska funkcija* ali krajše i-ti *minterm*. ■

V i-ti vrstici pravilnostne tabele je na levi strani vhodni vektor

$$w_i = (w_{i1}, w_{i2}, \dots, w_{in}),$$

kjer je $w_{ij} \in \{0, 1\}$ vrednost spremenljivke x_j, $1 \le j \le n$, v i-ti vrstici.

Minterm m_i lahko sedaj zapišemo v obliki produkta literalov

$$m_i(x_1, x_2, \dots, x_n) = \dot{x}_1\dot{x}_2\dots\dot{x}_n,$$

kjer je $\dot{x}_j = \begin{cases} x'_j, & \text{če } w_{ij} = 0 \\ x_j, & \text{če } w_{ij} = 1 \end{cases}, \quad 1 \le j \le n.$

Minterm je torej preklopna funkcija n spremenljivk, ki jo lahko zapišemo kot produkt *(konjunkcijo)* n različnih literalov.

Kanonične oblike preklopnih funkcij: Mintermi

Za n spremenljivk obstaja 2^n mintermov.

Primer: Mintermi za tri spremenljivke.

i	x_1	x_2	x_3	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	
0	0	0	0	1	0	0	0	0	0	0	0	$m_0(x_1,x_2,x_3) = x'_1x'_2x'_3$
1	0	0	1	0	1	0	0	0	0	0	0	$m_1(x_1,x_2,x_3) = x'_1x'_2x_3$
2	0	1	0	0	0	1	0	0	0	0	0	$m_2(x_1,x_2,x_3) = x'_1x_2x'_3$
3	0	1	1	0	0	0	1	0	0	0	0	$m_3(x_1,x_2,x_3) = x'_1x_2x_3$
4	1	0	0	0	0	0	0	1	0	0	0	$m_4(x_1,x_2,x_3) = x_1x'_2x'_3$
5	1	0	1	0	0	0	0	0	1	0	0	$m_5(x_1,x_2,x_3) = x_1x'_2x_3$
6	1	1	0	0	0	0	0	0	0	1	0	$m_6(x_1,x_2,x_3) = x_1x_2x'_3$
7	1	1	1	0	0	0	0	0	0	0	1	$m_7(x_1,x_2,x_3) = x_1x_2x_3$

Kanonične oblike preklopnih funkcij: Popolna disjunktivna normalna oblika

Izrek. Vsako preklopno funkcijo $f(x_1, x_2, ..., x_n)$ lahko izrazimo v obliki vsote mintermov:

$$\begin{aligned} f(x_1, x_2, ..., x_n) &= f(w_0)m_0(x_1, x_2, ..., x_n) + \\ &\quad f(w_1)m_1(x_1, x_2, ..., x_n) + \\ &\quad \cdot \;\cdot \;\cdot \quad + \\ &\quad f(w_{2^n-1})m_{2^n-1}(x_1, x_2, ..., x_n) \\ &= \sum_{i=0}^{2^n-1} f(w_i)m_i(x_1, x_2, ..., x_n) \end{aligned}$$

Ker je funkcijska vrednost $f(w_i)$ lahko samo 0 ali 1, se zgornja enačba poenostavi v:

$$\boxed{f(x_1, x_2, ..., x_n) = \sum_{\substack{i=0 \\ f(w_i)=1}}^{2^n-1} m_i(x_1, x_2, ..., x_n)} \quad \blacksquare$$

To je popolna disjunktivna normalna oblika (PDNO) preklopne funkcije f. Mintermi, ki nastopajo v PDNO, funkcijo f enolično določajo. Imenujemo jih ***mintermi funkcije*** f.

Kanonične oblike preklopnih funkcij: Makstermi

Definicija. Naj bo $M_i(x_1, x_2, ..., x_n)$ preklopna funkcija n spremenljivk, ki ima v svoji pravilnostni tabeli vrednost 0 v i-ti vrstici, v vseh ostalih vrsticah pa vrednost 1. Funkciji M_i pravimo i-ta *makstermska funkcija* ali krajše i-ti *maksterm*. ■

V i-ti vrstici pravilnostne tabele je na levi strani vhodni vektor

$$w_i = (w_{i1}, w_{i2}, ... , w_{in}),$$

kjer je $w_{ij} \in \{0, 1\}$ vrednost spremenljivke x_j, $1 \le j \le n$, v i-ti vrstici.

Maksterm M_i lahko sedaj zapišemo v obliki vsote literalov

$$M_i(x_1, x_2, \ldots, x_n) = \dot{x}_1 + \dot{x}_2 + \ldots + \dot{x}_n,$$

kjer je $\dot{x}_j = \begin{cases} x'_j, \text{ če } w_{ij} = 1 \\ x_j, \text{ če } w_{ij} = 0 \end{cases}$, $1 \le j \le n$.

Maksterm je torej preklopna funkcija n spremenljivk, ki jo lahko zapišemo kot vsoto *(disjunkcijo)* n različnih literalov.

Kanonične oblike preklopnih funkcij: Makstermi

Za n spremenljivk obstaja 2^n makstermov.

Primer: Makstermi za tri spremenljivke.

i	x_1	x_2	x_3	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	
0	0	0	0	0	1	1	1	1	1	1	1	$M_0(x_1,x_2,x_3) = x_1+x_2+x_3$
1	0	0	1	1	0	1	1	1	1	1	1	$M_1(x_1,x_2,x_3) = x_1+x_2+x'_3$
2	0	1	0	1	1	0	1	1	1	1	1	$M_2(x_1,x_2,x_3) = x_1+x'_2+x_3$
3	0	1	1	1	1	1	0	1	1	1	1	$M_3(x_1,x_2,x_3) = x_1+x'_2+x'_3$
4	1	0	0	1	1	1	1	0	1	1	1	$M_4(x_1,x_2,x_3) = x'_1+x_2+x_3$
5	1	0	1	1	1	1	1	1	0	1	1	$M_5(x_1,x_2,x_3) = x'_1+x_2+x'_3$
6	1	1	0	1	1	1	1	1	1	0	1	$M_6(x_1,x_2,x_3) = x'_1+x'_2+x_3$
7	1	1	1	1	1	1	1	1	1	1	0	$M_7(x_1,x_2,x_3) = x'_1+x'_2+x'_3$

V splošnem velja:

$$M_i(x_1, x_2, ..., x_n) = m'_i(x_1, x_2, ..., x_n),$$
$$m_i(x_1, x_2, ..., x_n) = M'_i(x_1, x_2, ..., x_n).$$

Primer:

$$m_1(x_1, x_2, x_3) = x'_1x'_2x_3$$
$$M_1(x_1, x_2, x_3) = m'_1(x_1, x_2, x_3) = (x'_1x'_2x_3)' = x_1+x_2+x'_3$$

Kanonične oblike preklopnih funkcij: Popolna konjunktivna normalna oblika

Izrek. Vsako preklopno funkcijo $f(x_1, x_2, ..., x_n)$ lahko izrazimo v obliki produkta makstermov:

$$\begin{aligned} f(x_1, x_2, ..., x_n) = & (f(w_0) + M_0(x_1, x_2, ..., x_n)) \bullet \\ & (f(w_1) + M_1(x_1, x_2, ..., x_n)) \bullet \\ & \quad \cdot \quad \cdot \quad \cdot \quad \bullet \\ & (f(w_{2^n-1}) + M_{2^n-1}(x_1, x_2, ..., x_n)) \\ = & \prod_{i=0}^{2^n-1} (f(w_i) + M_i(x_1, x_2, ..., x_n)) \end{aligned}$$

Ker je funkcijska vrednost $f(w_i)$ lahko samo 0 ali 1, se zgornja enačba poenostavi v:

$$\boxed{f(x_1, x_2, ..., x_n) = \prod_{\substack{i=0 \\ f(w_i)=0}}^{2^n-1} M_i(x_1, x_2, ..., x_n)} \quad \blacksquare$$

To je popolna konjunktivna normalna oblika (PKNO) preklopne funkcije f. Makstermi, ki nastopajo v PKNO, funkcijo f enolično definirajo. Imenujemo jih *makstermi funkcije* f.

Kanonične oblike preklopnih funkcij

Primer: Zapišimo PDNO in PKNO preklopne funkcije, ki je specificirana z naslednjo pravilnostno tabelo:

$x_1x_2x_3$	f
0 0 0	0
0 0 1	0
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

PDNO: $f(x_1, x_2, x_3) = m_3 + m_4 + m_5 + m_6 + m_7 = \sum(3,4,5,6,7)$
$= x'_1x_2x_3 + x_1x'_2x'_3 + x_1x'_2x_3 + x_1x_2x'_3 + x_1x_2x_3$

PKNO: $f(x_1, x_2, x_3) = M_0M_1M_2 = \prod(0,1,2)$
$= (x_1 + x_2 + x_3) \bullet (x_1 + x_2 + x'_3) \bullet (x_1 + x'_2 + x_3)$

Kanonične oblike preklopnih funkcij

Pretvorbe med kanoničnimi oblikami

1. Pretvorba iz PDNO v PKNO:

   Oznako Σ za minterme zamenjamo z oznako Π za maksterme.
   Zamenjamo mintermske indekse z indeksi, ki v zapisu ne nastopajo.

   Npr., $F(A,B,C) = \Sigma(3,4,5,6,7) = \Pi(0,1,2)$

2. Pretvorba iz PKNO v PDNO:

   Oznako Π za maksterme zamenjamo z oznako Σ za minterme.
   Zamenjamo makstermske indekse z indeksi, ki v zapisu ne nastopajo.

   Npr., $F(A,B,C) = \Pi(0,1,2) = \Sigma(3,4,5,6,7)$

3. Pretvorba iz PDNO(PKNO) funkcije F v PKNO(PDNO) funkcije F':
   V PDNO(PKNO) funkcije F' nastopajo indeksi, ki ne nastopajo v PDNO(PKNO) funkcije F:

   Npr., $F(A,B,C) = \Sigma(3,4,5,6,7) \longrightarrow F'(A,B,C) = \Sigma(0,1,2)$
   $= \Pi(0,1,2) \longrightarrow = \Pi(3,4,5,6,7)$

Kanonične oblike preklopnih funkcij

Nepopolno specificirane funkcije v kanoničnih oblikah

Kanonični predstavitvi krožnega BCD števnika:

$$Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15$$

$$Z = \Sigma(0, 2, 4, 6, 8) + \Delta(10, 11, 12, 13, 14, 15)$$

$$Z = M1 \cdot M3 \cdot M5 \cdot M7 \cdot M9 \cdot D10 \cdot D11 \cdot D12 \cdot D13 \cdot D14 \cdot D15$$

$$Z = \Pi(1, 3, 5, 7, 9) \cdot \Delta(10, 11, 12, 13, 14, 15)$$

Z d označimo vrednost X v pravilnostni tabeli, če zapišemo funkcijo v obliki vsote mintermov.

Z D označimo vrednost X v pravilnostni tabeli, če zapišemo funkcijo v obliki produkta makstermov.

V okrajšanem decimalnem zapisu funkcije bodisi v obliki vsote mintermov bodisi v obliki produkta makstermov označujemo vrednosti X s črko Δ.

Kanonične oblike preklopnih funkcij

Shannonov izrek o razširitvi preklopne funkcije

Izrek. Vsako preklopno funkcijo $f(x_1, x_2, ..., x_i, ..., x_n)$ lahko izrazimo kot:

1. $f(x_1, x_2, ..., x_i, ..., x_n) = x_i f(x_1, x_2, ..., 1, ..., x_n) + x'_i f(x_1, x_2, ..., 0, ..., x_n)$
2. $f(x_1, x_2, ..., x_i, ..., x_n) = (x_i + f(x_1, x_2, ..., 0, ..., x_n))\bullet(x'_i + f(x_1, x_2, ..., 1, ..., x_n))$

Pravimo, da smo funkcijo f *razširili po spremenljivki* x_i, $1 \leq i \leq n$.

Dokaz: Izrek dokažemo s popolno indukcijo.

Naj bo $x_i = 1$, torej $x'_i = 0$. To vstavimo v enačbi 1 in 2. Dobimo:

$$\begin{aligned} f(x_1, x_2, ..., 1, ..., x_n) &= 1\bullet f(x_1, x_2, ..., 1, ..., x_n) + 0\bullet f(x_1, x_2, ..., 0, ..., x_n) \\ &= f(x_1, x_2, ..., 1, ..., x_n) \\ f(x_1, x_2, ..., 1, ..., x_n) &= (1 + f(x_1, x_2, ..., 0, ..., x_n))\bullet(0 + f(x_1, x_2, ..., 1, ..., x_n)) \\ &= f(x_1, x_2, ..., 1, ..., x_n) \end{aligned}$$

Zdaj naj bo $x_i = 0$, torej $x'_i = 1$. To vstavimo v enačbi 1 in 2. Dobimo:

$$\begin{aligned} f(x_1, x_2, ..., 0, ..., x_n) &= 0\bullet f(x_1, x_2, ..., 1, ..., x_n) + 1\bullet f(x_1, x_2, ..., 0, ..., x_n) \\ &= f(x_1, x_2, ..., 0, ..., x_n) \\ f(x_1, x_2, ..., 0, ..., x_n) &= (0 + f(x_1, x_2, ..., 0, ..., x_n))\bullet(1 + f(x_1, x_2, ..., 1, ..., x_n)) \\ &= f(x_1, x_2, ..., 0, ..., x_n) \ \blacksquare \end{aligned}$$

Kanonične oblike preklopnih funkcij

Izpeljava PDNO preklopne funkcije z uporabo Shannonovega izreka

Če po Shannonovem izreku razširimo preklopno funkcijo $f(x_1, x_2, ..., x_n)$ po vsaki spremenljivki x_i, $1 \leq i \leq n$, dobimo njeno PDNO oz. PKNO.

$$\begin{aligned} f(x_1, x_2, ..., x_n) &= x_1 f(1, x_2, ..., x_n) + x'_1 f(0, x_2, ..., x_n) \\ &= x_1(x_2 f(1, 1, ..., x_n) + x'_2 f(1, 0, ..., x_n)) + \\ &\quad x'_1(x_2 f(0, 1, ..., x_n) + x'_2 f(0, 0, ..., x_n)) \\ &= x_1 x_2 f(1, 1, ..., x_n) + x_1 x'_2 f(1, 0, ..., x_n)) + \\ &\quad x'_1 x_2 f(0, 1, ..., x_n) + x'_1 x'_2 f(0, 0, ..., x_n)) \\ &= \quad ... \\ &= x_1 x_2 ... x_n f(1, 1, ..., 1) + \\ &\quad x_1 x_2 ... x'_n f(1, 1, ..., 0) + \\ &\quad ... + \\ &\quad x'_1 x'_2 ... x_n f(0, 0, ..., 1) + \\ &\quad x'_1 x'_2 ... x'_n f(0, 0, ..., 0) \\ &= \sum_{\substack{i=0 \\ f(w_{i1}, w_{i2}, ..., w_{in}) = 1}}^{2^n-1} m_i(x_1, x_2, ..., x_n) \\ &= \sum_{\substack{i=0 \\ f(w_i) = 1}}^{2^n-1} m_i(x_1, x_2, ..., x_n) \end{aligned}$$

Kanonične oblike preklopnih funkcij: Dvonivojska kombinacijska vezja

Obravnavani kanonični obliki (PDNO in PKNO) preklopnih funkcij sta *normalni*. Normalnost pomeni, da imamo med vhodnimi literali in izhodno funkcijo največ dva nivoja logičnih operatorjev. PDNO (PKNO) preklopne funkcije ima na prvem nivoju konjunkcije (disjunkcije), na drugem nivoju pa disjunkcije (konjunkcije).

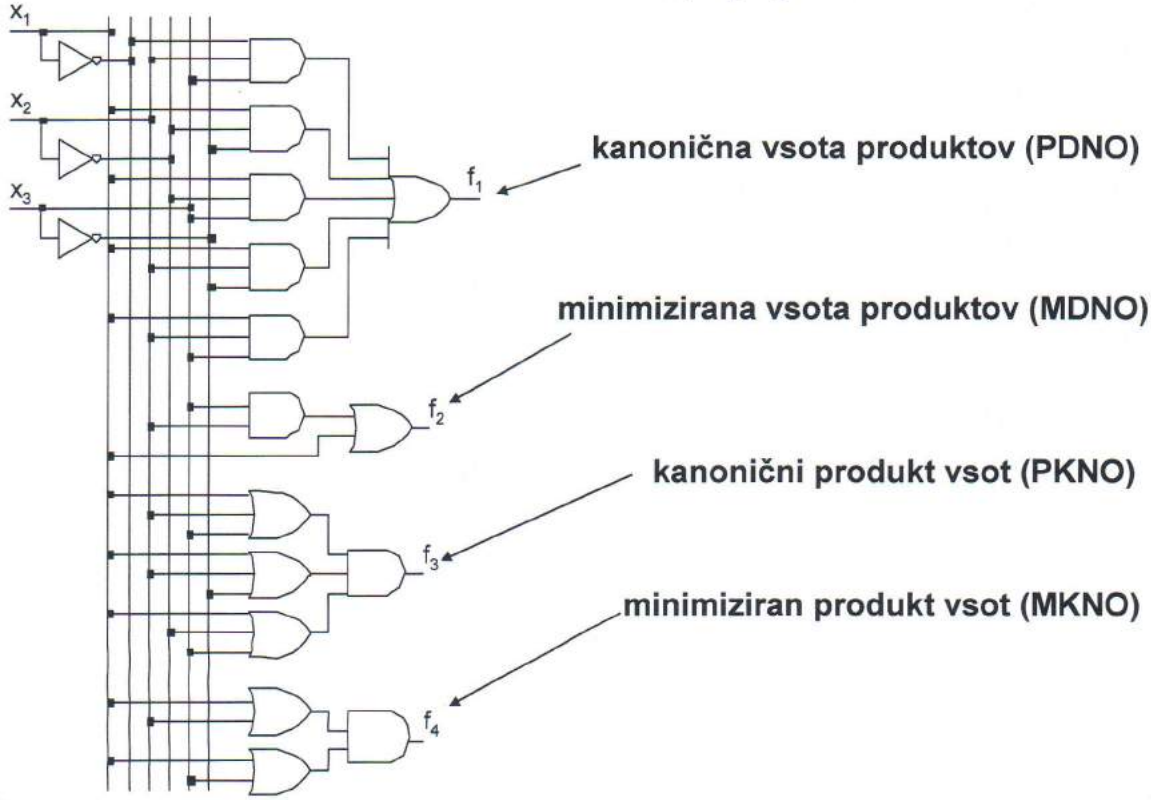
Normalne preklopne funkcije lahko realiziramo z *dvonivojskimi kombinacijskimi logičnimi vezji*. Globina (število vrat med vhodi in izhodom) takih vezij znaša dva, če ne štejemo vhodnih invertorjev. Ker lahko vsako preklopno funkcijo izrazimo v kanoničnih oblikah, ima vsaka funkcija dvonivojsko realizacijo, sestavljeno iz vrat AND in OR.

Dvonivojska realizacija preklopne funkcije je najboljša realizacija glede na zakasnitve ali hitrost delovanja vezja, vendar daleč od najboljše glede na število potrebnih vrat in povezav.

Če ima funkcija f z n spremenljivkami k mintermov, ima 2^n-k makstermov, zato lahko število vrat za kanonično realizacijo f-ja doseže vrednost 2^n, ki je lahko zelo velika. Taka realizacija lahko postavlja za fan-in in fan-out zahteve, ki presegajo tehnološke omejitve. Preklopne funkcije v praksi zato ne realiziramo s kanoničnimi ampak z minimiziranimi dvonivojskimi vezji, ki imajo minimalno možno ceno.

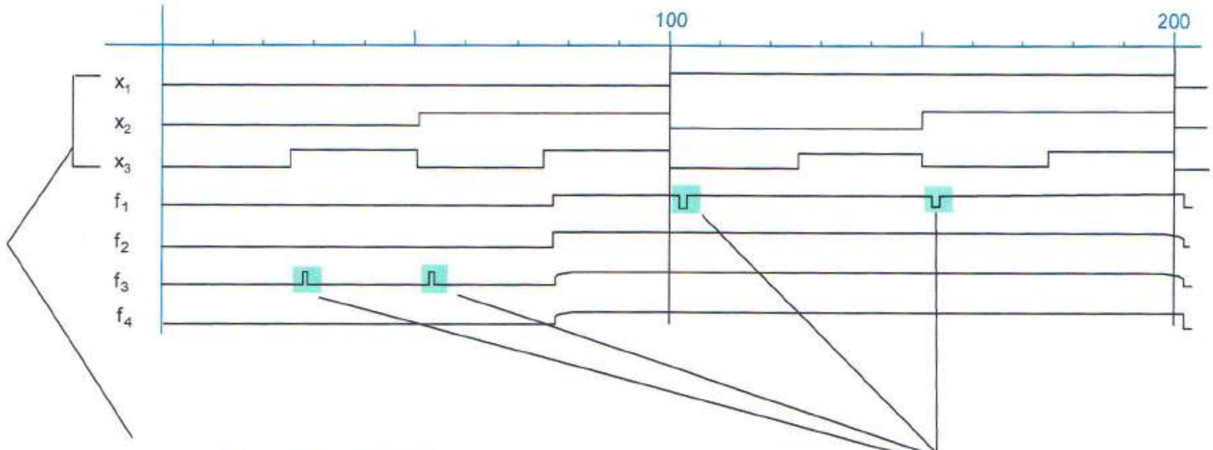
Kanonične oblike preklopnih funkcij: Dvonivojska kombinacijska vezja

Štiri alternativne implementacije funkcije $f(x_1, x_2, x_3) = \Sigma(3,4,5,6,7) = \Pi(0,1,2)$



Kanonične oblike preklopnih funkcij: Dvonivojska kombinacijska vezja

Primerjava časovnih potekov štirih alternativnih implementacij funkcije f



vse kombinacije vhodov x_1, x_2 in x_3

Razen kratkotrajnih napetostnih konic so časovni poteki signalov na izhodih vseh štirih implementacij v bistvu identični.

Funkcijsko polni sistemi preklopnih funkcij

Zaprti razredi in polni sistemi

Definicija. **Imejmo Boolovo funkcijsko algebro $\mathscr{F}_n = (F_n, \Phi)$. Množica funkcij $C \subset F_n$ je *zaprt razred*, če s funkcijo $f(x_1,x_2,\ldots,x_n) \in C$ ne moremo realizirati nobene funkcije, ki ne bi bila vsebovana v množici C. ■**

Pojem zaprtega razreda bomo potrebovali pri definiranju funkcijsko polnih sistemov preklopnih funkcij.

Dokazano je, da obstaja v logiki le pet *osnovnih* zaprtih razredov: T_0, T_1, S, L in M. Zaprti razred $C = F_n$ nas ne zanima.

Definirajmo vseh pet osnovnih zaprtih razredov.

Funkcijsko polni sistemi preklopnih funkcij

1. T_0 – razred ohranjanja konstante 0

Definicija. $f \in T_0$: $f(0, 0, ..., 0) = 0$ ■

Če vse vhodne preklopne spremenljivke zavzamejo vrednost 0, funkcije $f \in T_0$ vrnejo rezultat 0, torej ohranjajo konstanto 0.

Če je $f \in T_0$, v PDNO funkcije f ne nastopa minterm m_0, v PKNO funkcije f pa nastopa maksterm M_0.

Izrek. V razredu T_0 obstaja $2^{2^n}/2$ funkcij. ■

Npr., za n = 2 je to naslednjih osem funkcij:

$f_0 = 0$
$f_1 = x_1x_2$
$f_2 = (x_1 \rightarrow x_2)'$
$f_3 = x_1$
$f_4 = (x_2 \rightarrow x_1)'$
$f_5 = x_2$
$f_6 = x_1 \oplus x_2$
$f_7 = x_1 + x_2$

Funkcijsko polni sistemi preklopnih funkcij

2. T_1 – razred ohranjanja konstante 1

Definicija. $f \in T_1$: $f(1, 1, ..., 1) = 1$ ■

Če vse vhodne preklopne spremenljivke zavzamejo vrednost 1, funkcije $f \in T_1$ vrnejo rezultat 1, torej ohranjajo konstanto 1.

Če je $f \in T_1$, v PDNO funkcije f nastopa minterm m_{2^n-1}, v PKNO funkcije f pa ne nastopa maksterm M_{2^n-1}.

Izrek. V razredu T_1 obstaja $2^{2^n}/2$ funkcij. ■

Npr., za n = 2 je to naslednjih osem funkcij:

$f_1 = x_1x_2$
$f_3 = x_1$
$f_5 = x_2$
$f_7 = x_1 + x_2$
$f_9 = x_1 \equiv x_2$
$f_{11} = x_2 \rightarrow x_1$
$f_{13} = x_1 \rightarrow x_2$
$f_{15} = 1$

Funkcijsko polni sistemi preklopnih funkcij

3. S – razred sebi dualnih preklopnih funkcij

Definicija. $f \in S : f'(x'_1, x'_2, ..., x'_n) = f(x_1, x_2, ..., x_n)$ ■

Za funkcije $f \in S$ komplementirana funkcija f' preslikuje komplementirane vhodne preklopne spremenljivke x'_i, $1 \le i \le n$, natanko tako kot funkcija f vhodne preklopne spremenljivke x_i.

Izrek. **V razredu S obstaja $2^{2^{n-1}}$ funkcij.** ■

Npr., za n = 2 so to naslednje štiri funkcije:

$$f_3 = x_1$$
$$f_5 = x_2$$
$$f_{10} = x'_2$$
$$f_{12} = x'_1$$

Funkcijsko polni sistemi preklopnih funkcij

4. L – razred linearnih preklopnih funkcij

Definicija. $f \in L : f(x_1, x_2, ..., x_n) = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus ... \oplus a_nx_n$, $a_i \in \{0,1\}$, $0 \le i \le n$. ■

Funkcije $f \in L$ imajo v pravilnostni tabeli sodo število enic.

Izrek. **V razredu L obstaja 2^{n+1} funkcij.** ■

Npr., za n = 2 je to naslednjih osem funkcij:

$$f_0 = 0 = 0 \oplus 0x_1 \oplus 0x_2$$
$$f_3 = x_1 = 0 \oplus 1x_1 \oplus 0x_2$$
$$f_5 = x_2 = 0 \oplus 0x_1 \oplus 1x_2$$
$$f_6 = x_1 \oplus x_2 = 0 \oplus 1x_1 \oplus 1x_2$$
$$f_9 = x_1 \equiv x_2 = (x_1 \oplus x_2)' = 1 \oplus 1x_1 \oplus 1x_2$$
$$f_{10} = x'_2 = 1 \oplus 0x_1 \oplus 1x_2$$
$$f_{12} = x'_1 = 1 \oplus 1x_1 \oplus 0x_2$$
$$f_{15} = 1 = 1 \oplus 0x_1 \oplus 0x_2$$

Funkcijsko polni sistemi preklopnih funkcij

5. M – razred monotono naraščajočih preklopnih funkcij

Najprej si definirajmo relacijo $\leq$.

Definicija. $w_i \leq w_j$, če in samo če $w_{ik} \leq w_{jk}$, $1 \leq k \leq n$, kjer sta $w_i = (w_{i1}, w_{i2}, \dots, w_{in})$ in $w_j = (w_{j1}, w_{j2}, \dots, w_{jn})$ n-bitni kodni besedi in $w_{ik}, w_{jk} \in \{0,1\}$, $1 \leq k \leq n$. ■

Primer: $w_1 = (1,0,0,0)$, $w_2 = (1,0,1,0)$, $w_3 = (0,1,1,1)$.

$w_1 \leq w_2$, $w_1 \nleq w_3$, $w_3 \nleq w_1$, $w_2 \nleq w_3$, $w_3 \nleq w_2$

Definicija. $f \in M$: $w_i \leq w_j \rightarrow f(w_i) \leq f(w_j)$ ■

Npr., za n = 2 obstaja v razredu M naslednjih šest funkcij:

$f_0 = 0$
$f_1 = x_1 x_2$
$f_3 = x_1$
$f_5 = x_2$
$f_7 = x_1 + x_2$
$f_{15} = 1$

Funkcijsko polni sistemi preklopnih funkcij

Definicija. Množico $F = \{f_1, f_2, \dots, f_q\}$ imenujemo funkcijsko poln sistem, če s funkcijami $f_1, f_2, \dots, f_q$ odpremo vseh pet osnovnih zaprtih razredov, tj., če velja:

1. $\exists f \in F : f \notin T_0$
2. $\exists f \in F : f \notin T_1$
3. $\exists f \in F : f \notin S$
4. $\exists f \in F : f \notin L$
5. $\exists f \in F : f \notin M$ ■

Z logičnimi vrati, ki ustrezajo elementom $f \in F$ funkcijsko polnega sistema, lahko realiziramo poljubno Boolovo funkcijo.

Primer: Ali je množica funkcij F ={+, →, 1} funkcijsko poln sistem?

F \ C	T_0	T_1	S	L	M
+	$\in$	$\in$	$\notin$	$\notin$	$\in$
→	$\notin$	$\in$	$\notin$	$\notin$	$\notin$
1	$\notin$	$\in$	$\notin$	$\in$	$\in$
0	$\in$	$\notin$	$\notin$	$\in$	$\in$

Množica funkcij F ={+, →, 1} ni funkcijsko poln sistem, ker s funkcijami iz F nismo odprli razreda T_1. Funkcijsko polna pa je enostavnejša množica F* = { →, 0}, ki vsebuje implikacijo in konstanto 0.

Funkcijsko polni sistemi preklopnih funkcij

Osnovni funkcijsko poln sistem **je {+, •, ' }, saj se nanj nanašajo postulati Boolove algebre. Funkcijsko polnost nekega sistema lahko preverimo tudi tako, da ga prevedemo na omenjeni osnovni funkcijsko poln sistem. Če lahko s funkcijami (operatorji) iz množice F izrazimo disjunkcijo, konjunkcijo in negacijo, je preverjani sistem poln, sicer pa ne. V tabeli je prikazanih nekaj funkcijsko polnih sistemov.**

F	x_1+x_2	x_1x_2	x'_1
{+, •, ' }	x_1+x_2	x_1x_2	x'_1
{+, ' }	x_1+x_2	$(x'_1+x'_2)'$	x'_1
{•, ' }	$(x'_1x'_2)'$	x_1x_2	x'_1
{↓}	$(x_1{\downarrow}x_2){\downarrow}(x_1{\downarrow}x_2)$	$(x_1{\downarrow}x_1){\downarrow}(x_2{\downarrow}x_2)$	$x_1{\downarrow}x_1$
{|}	$(x_1 \mid x_1) \mid (x_2 \mid x_2)$	$(x_1 \mid x_2) \mid (x_1 \mid x_2)$	$x_1 \mid x_1$
{→, 0}	$(x_1\to 0)\to x_2$	$(x_1\to(x_2\to 0))\to 0$	$x_1\to 0$
{≡, +, 0}	x_1+x_2	$((x_1\equiv 0)+(x_2\equiv 0))\equiv 0$	$x_1\equiv 0$

Najprimernejša funkcijsko polna sistema sta {↓} in { | }, kjer je ↓ Pierceov operator (NOR), | pa Shefferjev operator (NAND). Poljubno preklopno funkcijo lahko realiziramo samo z enim tipom logičnih vrat (NOR ali NAND). Poleg tega njuna implementacija zahteva manj tranzistorjev od implementacije operatorjev AND in OR.

Funkcijsko polni sistemi preklopnih funkcij

Shefferjev in Pierceov funkcijsko poln sistem

Zaradi naštetih prednosti Shefferjevega in Pierceovega operatorja se pogosto srečamo z nalogo, da moramo pretvoriti DNO ali KNO preklopne funkcije v Shefferjevo ali Pierceovo obliko ali povedano z drugimi besedami, vezje AND/OR ali OR/AND moramo pretvoriti v vezje NAND/NAND ali NOR/NOR.

Pretvorba temelji na DeMorganovem izreku.

DeMorganov izrek: $(A + B)' = A' \bullet B'$; $(A \bullet B)' = A' + B'$

Drugačen zapis: $A + B = (A' \bullet B')'$; $A \bullet B = (A' + B')'$

Z drugimi besedami:

OR je enak kot NAND s komplementiranimi vhodi.

AND je enak kot NOR s komplementiranimi vhodi.

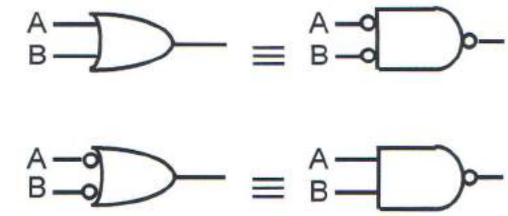
NAND je enak kot OR s komplementiranimi vhodi.

NOR je enak kot AND s komplementiranimi vhodi.

Funkcijsko polni sistemi preklopnih funkcij

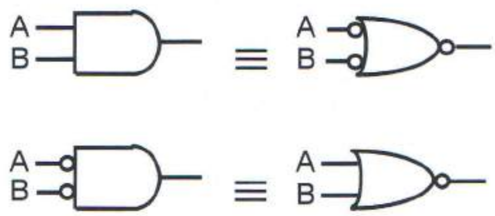
Ekvivalenca OR/NAND

A	$\overline{A}$	B	$\overline{B}$	$A+B$	$\overline{\overline{A}\cdot\overline{B}}$	$\overline{A}+\overline{B}$	$\overline{A\cdot B}$
0	1	0	1	0	0	1	1
0	1	1	0	1	1	1	1
1	0	0	1	1	1	1	1
1	0	1	0	1	1	0	0



Ekvivalenca AND/NOR

A	$\overline{A}$	B	$\overline{B}$	$A\cdot B$	$\overline{\overline{A}+\overline{B}}$	$\overline{A}\cdot\overline{B}$	$\overline{A+B}$
0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0



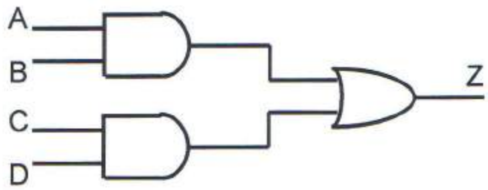
Vezja z vrati AND in OR je mogoče pretvoriti v vezja z vrati NAND in NOR tako, da vpeljemo ustrezne krožce za komplementiranje signalov.

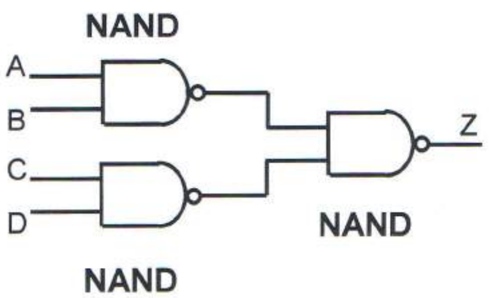
Sheme z vrati NAND in NOR so lažje razumljive, če se v vozliščih vezja izhodni in vhodni krožci ujemajo.

Funkcijsko polni sistemi preklopnih funkcij

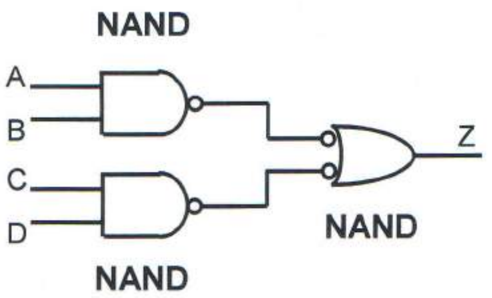
Primer: Pretvorba vezja AND/OR v vezje NAND/NAND

$$Z = AB + CD = (AB)' \mid (CD)' = (A \mid B) \mid (C \mid D)$$





a) Shema s konvencionalnimi vrati NAND

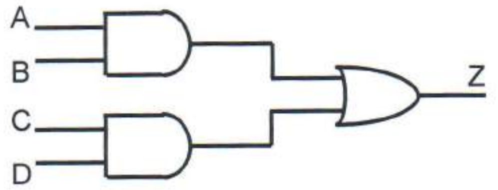


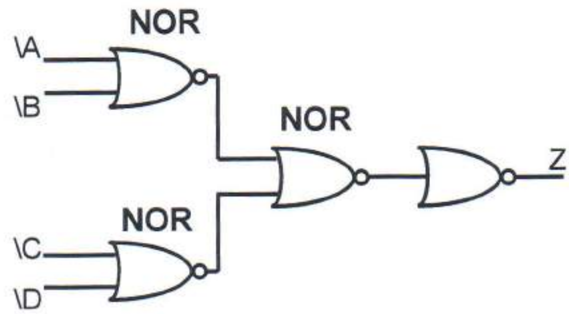
b) Shema z ujemanjem krožcev

Funkcijsko polni sistemi preklopnih funkcij

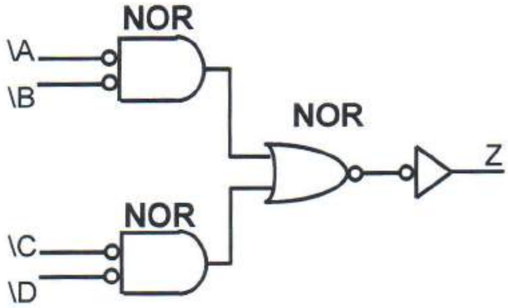
Primer: Pretvorba vezja AND/OR v vezje NOR/NOR

$Z = AB + CD = [(AB + CD)']' = [(AB)\downarrow(CD)]' = [(A'\downarrow B')\downarrow(C'\downarrow D')]'$





a) shema s konvencionalnimi vrati NOR



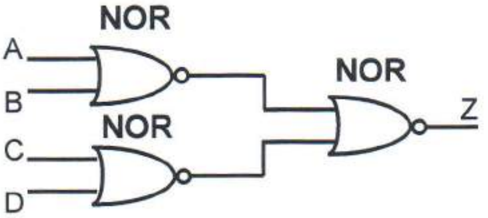
b) Shema z ujemanjem krožcev

Zaradi potrebe po izhodnem invertorju je to vezje trinivojsko. Za vezje AND/OR (DNO preklopne funkcije) je bolj naravna pretvorba v vezje NAND/NAND.

Funkcijsko polni sistemi preklopnih funkcij

Primer: Pretvorba vezja OR/AND v vezje NOR/NOR

$Z = (A + B)\cdot(C + D) = (A + B)'\downarrow(C + D)' = (A\downarrow B)\downarrow(C\downarrow D)$



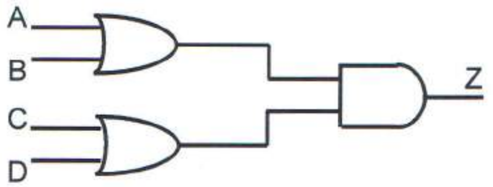
a) Shema s konvencionalnimi vrati NOR

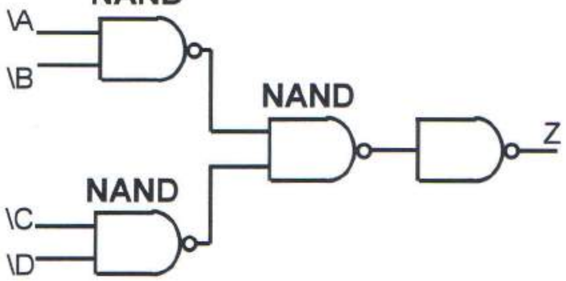
b) Shema z ujemanjem krožcev

Funkcijsko polni sistemi preklopnih funkcij

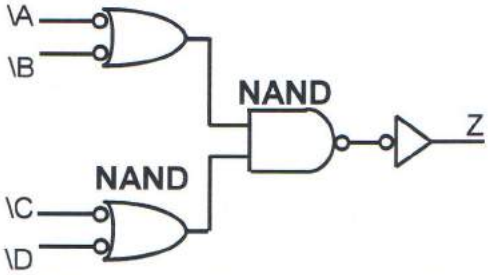
Primer: Pretvorba vezja OR/AND v vezje NAND/NAND

$$Z = (A + B)\cdot(C + D) = \{[(A + B)\cdot(C + D)]'\}' = [(A + B) \mid (C + D)]'$$
$$= [(A' \mid B') \mid (C' \mid B')]'$$





a) Shema s konvencionalnimi vrati NAND



b) Shema z ujemanjem krožcev

Zaradi potrebe po izhodnem invertorju je to vezje trinivojsko. Za vezje OR/AND (KNO preklopne funkcije) je bolj naravna pretvorba v vezje NOR/NOR.

Povzetek poglavja

Predstavili smo:

- **Boolovo algebro**
- **Boolove funkcije**
- **Kanonične oblike Boolovih funkcij**
- **Osnovne logične gradnike–vrata**

4. Minimizacija preklopnih funkcij

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Vsebina poglavja

- Osnove minimizacije
- Algebrsko poenostavljanje
- Metoda minimizacije z Boolovimi kockami
- Karnaughova metoda minimizacije
- Veitcheva metoda minimizacije
- Quine-McCluskeyeva metoda minimizacije
- CAD orodja za minimizacijo

Osnove minimizacije: Minimizacijski izrek

Minimizacija preklopne funkcije privede DNO (KNO) v njeno minimalno disjunktivno normalno obliko–MDNO (minimalno konjunktivno normalno obliko–MKNO). MDNO (MKNO) vsebuje najmanjše število konjunktivnih (disjunktivnih) členov in minimalno število literalov.

Minimizirano preklopno funkcijo lahko implementiramo z logičnim vezjem, ki ima največ dva nivoja in vsebuje najmanjše možno število vrat, kar minimizira ceno komponent vezja, in najmanjše možno število literalov, kar minimizira zahteve za fan-in vrat in s tem število potrebnih povezav. Osnova za minimizacijo je naslednji *minimizacijski izrek*:

Izrek. Naj bo E poljubni Boolov izraz in x Boolova spremenljivka.
Velja: $Ex + Ex' = E$ in $(E + x)\bullet(E + x') = E$. ■

Minimizacijski izrek je posplošitev izreka o sosednosti (I16a in I16b). Ključna procedura minimizacijskega postopka je poenostavitev dveh konjunktivnih ali disjunktivnih členov, ki sta *logično sosedna*.

Logično sosedna sta člena, ki se razlikujeta samo v enem literalu (v enem členu nastopa ena spremenljivka komplementirana, v drugem pa nekomplementirana).

Z zaporedno uporabo minimizacijskega izreka odstranjujemo nepotrebne literale ali celotne konjunktivne ali disjunktivne člene.

Osnove minimizacije: Vsebovalnik funkcije

Definicija. Preklopna funkcija $f(x_1, x_2, ..., x_n)$ *pokriva* funkcijo $g(x_1, x_2, ..., x_n)$, če zavzame f vrednost 1, kadarkoli jo zavzame g. Torej, če f pokriva g, potem ima f vrednost 1 v vsaki vrstici pravilnostne tabele, v kateri ima g vrednost 1. ■

Definicija. Če f pokriva g in hkrati g pokriva f, sta funkciji f in g *ekvivalentni.* ■

Naj bo $f(x_1, x_2, ..., x_n)$ preklopna funkcija in $p(x_1, x_2, ..., x_n)$ produktni člen (produkt literalov). Če f pokriva p, potem pravimo, da p *implicira* f. p-ju pravimo *vsebovalnik* (implicant) f-ja.

Primer: $f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x'_2x_3$

Vsebovalniki f-ja so: x_1x_2, x_2x_3, $x_1x'_2x_3$, $x'_1x_2x_3$, $x_1x_2x'_3$, $x_1x_2x_3$, x_1x_3, ...

Osnove minimizacije: Glavni in potrebni glavni vsebovalnik funkcije

Definicija. Vsebovalnik p funkcije f imenujemo *glavni vsebovalnik* (prime implicant), če z brisanjem kateregakoli literala iz p dobimo produktni člen, ki ni več vsebovalnik f-ja. ■

Primer: $f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_1x'_2x_3$

Npr., x_1x_2 je glavni vsebovalnik f-ja, ker niti sam x_1 niti sam x_2 ni vsebovalnik f-ja. Vsebovalnik $x_1x'_2x_3$ ni glavni vsebovalnik, ker dobimo z brisanjem literala x'_2 člen x_1x_3, ki je vsebovalnik f-ja.

Definicija. Glavni vsebovalnik p funkcije f je *potrebni glavni vsebovalnik* (essential prime implicant), če pokriva vsaj en minterm funkcije f, ki ni pokrit z nobenim drugim glavnim vsebovalnikom. ■

V MDNO nastopajo vsi potrebni glavni vsebovalniki in minimalna množica tistih glavnih vsebovalnikov, ki pokrijejo vse s potrebnimi glavnimi vsebovalniki še nepokrite minterme.

Osnove minimizacije: Dualni pojmi za iskanje MKNO

Po pričakovanju imamo k vsebovalnikom, glavnim vsebovalnikom in potrebnim glavnim vsebovalnikom tudi dualne pojme, ki se nanašajo na vsotne člene.

Angleški izrazi za dualne pojme so: *implicate*, *prime implicate* in *essential prime implicate*. Ustreznih prevodov za te izraze v slovenščini ni.

Zaradi skrajšane obravnave minimizacijskih metod bomo vsako metodo podrobno razložili pri iskanju MDNO preklopne funkcije. Na koncu predstavitve posamezne metode bomo na kratko povedali tudi, kako pridemo do MKNO.

Osnove minimizacije: Vrste in značilnosti minimizacijskih metod

Obstaja veliko metod za minimizacijo preklopnih funkcij. Najpomembnejše so: algebrsko poenostavljanje, ročne grafične metode (metoda z Boolovimi kockami, Karnaughova metoda, Veitcheva metoda), ročne tabelarične metode (Quine-McCluskeyeva metoda) in CAD orodja za minimizacijo (Espresso).

Algebrsko poenostavljanje

- V ad-hoc zaporedju uporabljamo minimizacijski izrek.
- Procedura ni algoritmična in zato ni sistematična.
- Težko ugotovimo, ali smo že dosegli minimalno obliko.

CAD orodja za minimizacijo

- Eksaktne rešitve zahtevajo zelo veliko procesorskega časa, še posebej za funkcije z velikim številom vhodnih spremenljivk ($n > 10$).
- Za zmanjšanje kompleksnosti računanja se zatečemo k uporabi hevrističnih metod, ki dajo dobre, ne pa tudi najboljše rešitve.

Ročne metode

- Praktično so uporabne samo za minimizacijo funkcij z manjšim številom vhodnih spremenljivk ($n \leq 6$), ki pa so pri delu v laboratoriju najpogostejše.
- Poznavanje ročnih metod nam daje vpogled v delovanje CAD orodij za minimizacijo in možnost, da vsaj na manjših primerih preverimo njihove rezultate.

Algebrsko poenostavljanje

Primer: Podana je PDNO preklopne funkcije F: $F(A,B,C)=\sum(0,1,2,3,7)$. Izpeljimo MDNO funkcije F.

$$F(A,B,C) = A'B'C' + A'B'C + A'BC' + A'BC + ABC = A' + BC$$

A'B' A'B BC

A'

Zdaj izpeljimo še MKNO funkcije F.

$$F(A,B,C) = \sum(0,1,2,3,7) = \prod(4,5,6) =$$

$$= (A' + B + C)\bullet(A' + B + C')\bullet(A' + B' + C) = (A' + B)\bullet(A' + C)$$

(A' + B) (A' + C)

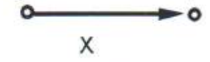
Pokažimo, da je Boolov izraz za MKNO ekvivalenten izrazu za MDNO:

$$(A' + B)\bullet(A' + C) = A' + A'B + A'C + BC = A'(1 + B + C) + BC = A' + BC \checkmark$$

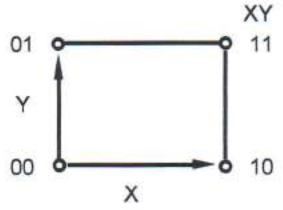
Metoda minimizacije z Boolovimi kockami

Boolove kocke

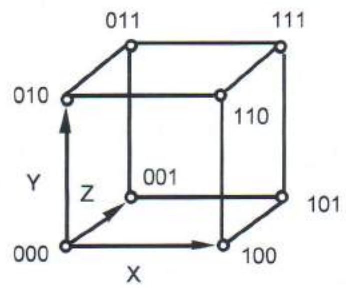
Boolove kocke so grafične sheme za ugotavljanje, kje lahko uporabimo minimizacijski izrek. Za n vhodnih spremenljivk dobimo n-dimenzionalno "kocko".



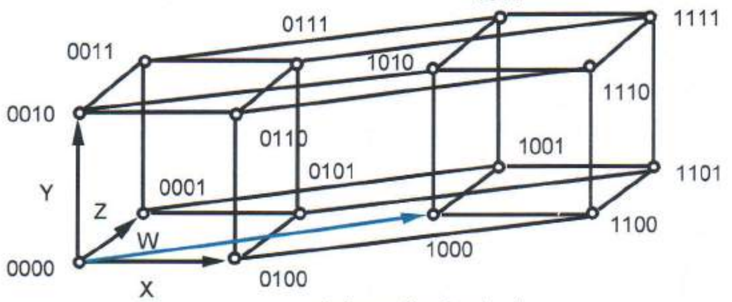
1-dimenzionalna kocka



2-dimenzionalna kocka



3-dimenzionalna kocka



4-dimenzionalna kocka

Metoda minimizacije z Boolovimi kockami

Preslikava pravilnostnih tabel v Boolove kocke

Množico "ON-set" predstavljajo polna vozlišča.

Množico "OFF-set" predstavljajo prazna vozlišča.

Množico "DC-set" predstavljajo vozlišča X.

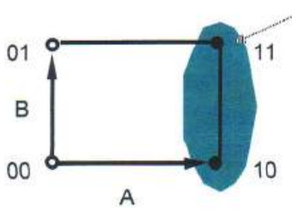
Primer:

AB	FG
00	01
01	00
10	11
11	10

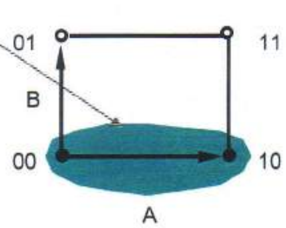
Kocka z dimenzijo n-1

Reduciran izraz vsebuje n-1 spremenljivk.

F



G



A ima vrednost 1.
B ima vrednost 0 ali 1.

A ima vrednost 0 ali 1.
B ima vrednost 0.

Metoda minimizacije z Boolovimi kockami

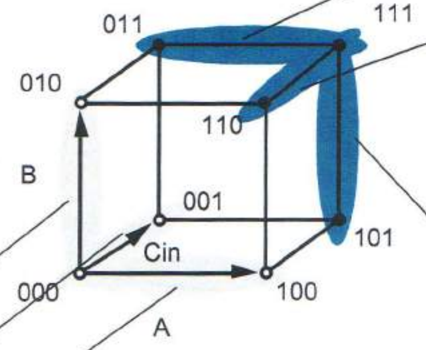
Primer s tremi spremenljivkami: Izhodni prenos popolnega seštevalnika

A	B	Cin	Cout
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

(A' + A) B Cin = BCin

A B (Cin' + Cin) = AB

Množica "OFF-set" je pokrita s konjunkcijami treh svetlo šrafiranih podkock z dimenzijo 1.



Množica "ON-set" je pokrita z disjunkcijami treh temno šrafiranih podkock z dimenzijo 1.

A (B + B') Cin = ACin

$A'C'_{in}$

$A'B'$

$B'C'_{in}$

MDNO: Cout = B Cin + A B + A Cin

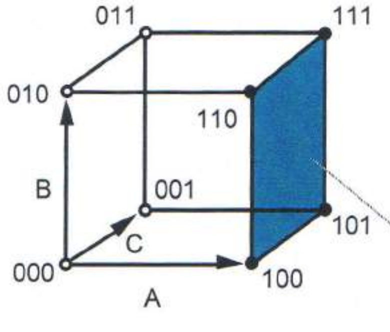
MKNO: Cout' = B'Cin' + A'B' + A'Cin'

(Cout')' = [B'Cin' + A'B' + A'Cin']'

Cout = (B + Cin)•(A + B)•(A+ Cin)

Metoda minimizacije z Boolovimi kockami

Podkocke z dimenzijami večjimi od 2



F(A,B,C) = Σ(4,5,6,7)

Množica "ON-set" tvori kvadrat, (2-dimenzionalno kocko). Predstavlja izraz z eno spremenljivko (3 - 2 = 1).

A ima vrednost 1.
B in C imata vrednosti 0 ali 1.

Ta podkocka predstavlja literal A.

Metoda minimizacije z Boolovimi kockami

V 3-dimenzionalni kocki imamo:

- **0-dimenzionalne kocke–posamezna vozlišča, ki dajejo člen s tremi literali,**
- **1-dimenzionalne kocke–stranice z dvema vozliščema, ki dajejo člen z dvema literaloma,**
- **2-dimenzionalne kocke–ravnine s štirimi vozlišči, ki dajejo člen z enim literalom in**
- **3-dimenzionalno kocko–kocko z osmimi vozlišči, ki daje konstantni člen "1".**

V splošnem velja, da k-dimenzionalna kocka v n-dimenzionalni kocki (k < n) daje člen z n - k literali.

Metoda minimizacije z Boolovimi kockami je uporabna le do dimenzije n = 4, ker je težko risati kocke z večjimi dimenzijami.

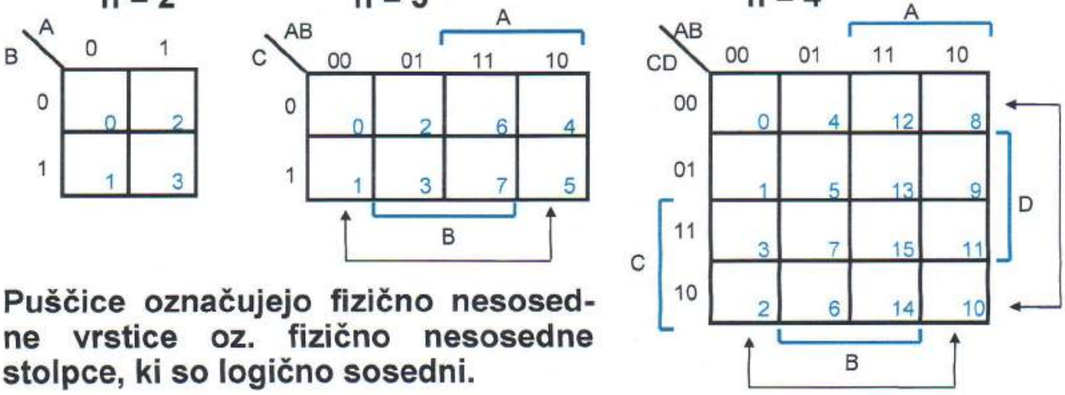
Karnaughova metoda minimizacije

Karnaughov diagram

***Karnaughov diagram* (krajše *K-diagram*) je matrična logična shema za predstavitev pravilnostne tabele, iz katere lahko z opazovanjem dokaj enostavno odčitamo sosednosti med Boolovimi kockami do 6 dimenzij.**

K-diagram ni v bistvu nič drugega kot modificiran Vennov diagram, če Vennove kroge spremenimo v kvadrate oz. pravokotnike in njihovo prekrivanje sistematično uredimo.

Oglejmo si splošne oblike K-diagramov za preklopne funkcije z 2, 3, 4, 5 in 6 spremenljivkami.



Puščice označujejo fizično nesosedne vrstice oz. fizično nesosedne stolpce, ki so logično sosedni.

Karnaughova metoda minimizacije

Karnaughov diagram

Karnaughov diagram za preklopno funkcijo z n spremenljivkami ima 2^n kvadratnih *celic*. Vsaka vrstica i v pravilnostni tabeli ustreza celici i v Karnaughovem diagramu. Celice so označene z desetiškim ekvivalentom i-ja v desnem spodnjem kotu.

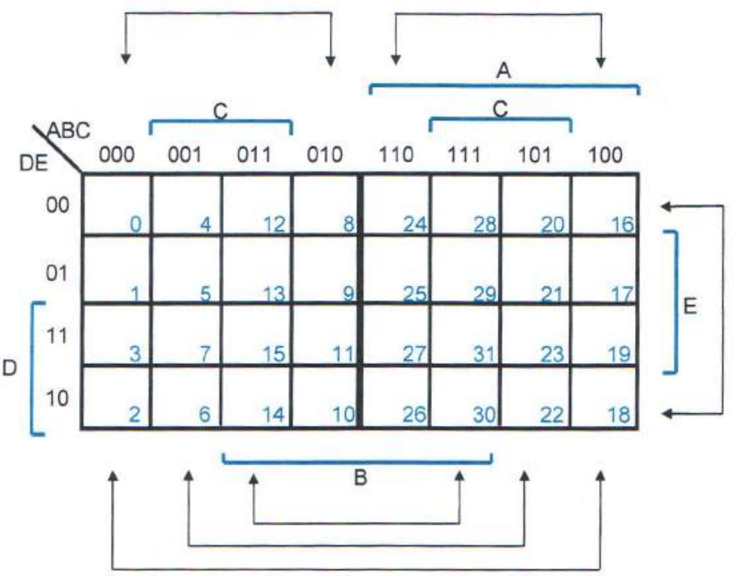
Vsaka celica i je naslovljena z binarno koordinato stolpca in binarno koordinato vrstice, ki ju tvori kombinacija vrednosti vhodnih spremenljivk. Koordinate stolpcev in vrstic so zapisane v Grayevem kodu–sosedni kodni besedi se razlikujeta samo v enem bitu.

Glavna ideja K-diagrama je ta, da so v njem logično sosedne celice tudi fizično sosedne. Na ta način je lahko prepoznati člene, ki jih smemo združiti v en sam, preprostejši člen. Upoštevati moramo, da so si v K-diagramu tudi fizično nesosedne celice logično sosedne, če ležijo zrcalno glede na *simetrali K-diagrama*. Npr., v K-diagramu za 4 spremenljivke so celici 8 logično sosedne celice 9, 12, 0 in 10.

Karnaughova metoda minimizacije

Karnaughov diagram

n = 5



Puščice označujejo fizično nesosedne vrstice oz. fizično nesosedne stolpce, ki so logično sosedni.

Karnaughova metoda minimizacije

Karnaughov diagram

n = 6

DEF \ ABC	000	001	011	010	110	111	101	100
000	0	8	24	16	48	56	40	32
001	1	9	25	17	49	57	41	33
011	3	11	27	19	51	59	43	35
010	2	10	26	18	50	58	42	34
110	6	14	30	22	54	62	46	38
111	7	15	31	23	55	63	47	39
101	5	13	29	21	53	61	45	37
100	4	12	28	20	52	60	44	36

Puščice označujejo fizično nesosedne vrstice oz. fizično nesosedne stolpce, ki so logično sosedni.

Karnaughova metoda minimizacije

Grupiranje celic

V celico i K-diagrama vpišemo vrednost funkcije v i-ti vrstici pravilnostne tabele.

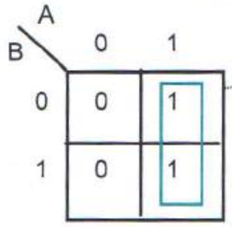
Vsaka celica z vrednostjo 1 (0) ustreza mintermu (makstermu) funkcije.

Če lahko množico 2^k mintermov preklopne funkcije f z n spremenljivkami predstavimo z enim samim produktnim členom (ta vsebuje n - k literalov), potem ustrezno grupo 2^k celic 1 v K-diagramu za f združimo v Boolovo podkocko in jo označimo s pravokotnikom ali kvadratom.

Cilj iskanja MDNO v K-diagramu je, da poiščemo najmanjše število največjih možnih podkock, s katerimi pokrijemo množico "ON-set".

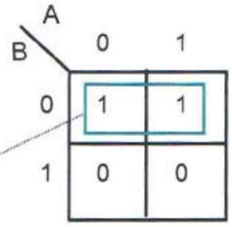
Karnaughova metoda minimizacije

Primeri K-diagramov



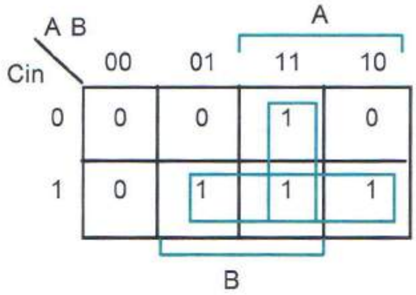
A ima vrednost 1.
B ima vrednost 0 ali 1.

F =

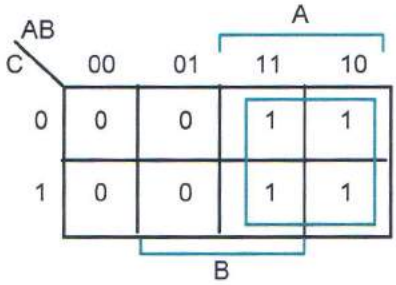


B ima vrednost 0.
A ima vrednost 0 ali 1.

G =



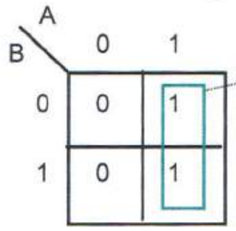
Cout =



F(A,B,C) =

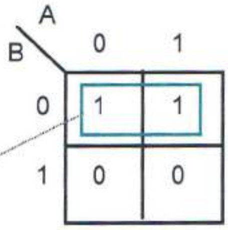
Karnaughova metoda minimizacije

Primeri K-diagramov



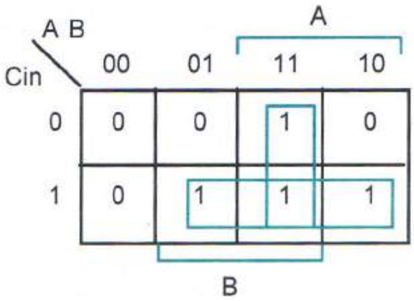
A ima vrednost 1.
B ima vrednost 0 ali 1.

F = A

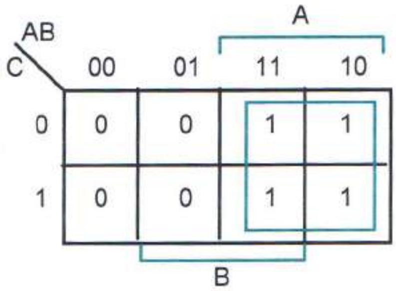


B ima vrednost 0.
A ima vrednost 0 ali 1.

G = B'



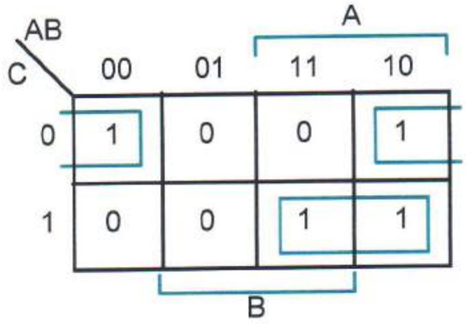
Cout = A B + B Cin + A Cin



F(A,B,C) = A

Karnaughova metoda minimizacije

Primer K-diagrama s tremi spremenljivkami

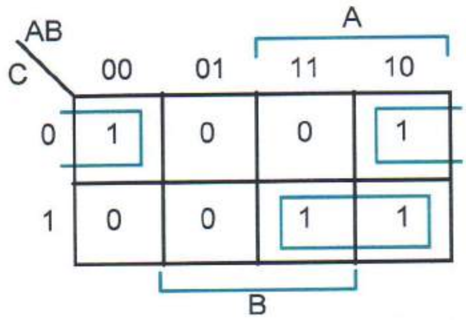


$F(A,B,C) = \Sigma(0,4,5,7)$

$F =$

Karnaughova metoda minimizacije

Primer K-diagrama s tremi spremenljivkami



$F(A,B,C) = \Sigma(0,4,5,7)$

$F = B'C' + AC$

Stolpca na robovih K-diagrama sta logično sosedna.

Karnaughova metoda minimizacije

Primer K-diagrama s štirimi spremenljivkami

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

A: stolpca 11, 10; B: stolpca 01, 11; C: vrstici 11, 10; D: vrstici 01, 11

F(A,B,C,D) = Σ(0,2,3,5,6,7,8,10,11,14,15)

F =

Karnaughova metoda minimizacije

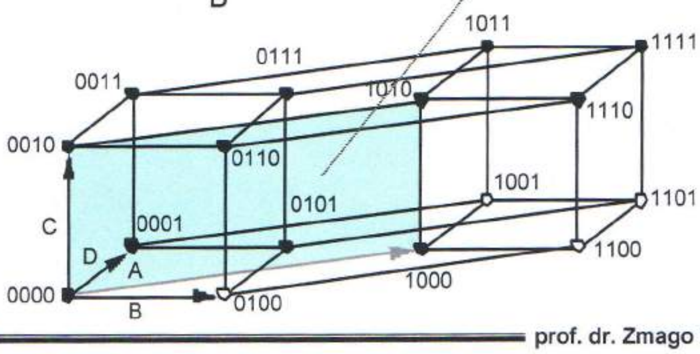
Primer K-diagrama s štirimi spremenljivkami

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

A: stolpca 11, 10; B: stolpca 01, 11; C: vrstici 11, 10; D: vrstici 01, 11

F(A,B,C,D) = Σ(0,2,3,5,6,7,8,10,11,14,15)

F = C + A' B D + B' D'



Ilustriracija logične sosednosti celic 1 v kotih K-diagrama s 4-dimenzionalno kocko.

Karnaughova metoda minimizacije

Primer K-diagrama z nedoločenimi vrednostmi

Nedoločene vrednosti lahko jemljemo kot 1 ali kot 0.

CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

$F(A,B,C,D) = \Sigma(1,3,5,7,9) + \Delta(6,12,13)$

Brez upoštevanja nedoločenih vrednosti: F =

Z upoštevanjem nedoločenih vrednosti: F =

Karnaughova metoda minimizacije

Primer K - diagrama z nedoločenimi vrednostmi

Nedoločene vrednosti lahko jemljemo kot 1 ali kot 0.

CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

$F(A,B,C,D) = \Sigma(1,3,5,7,9) + \Delta(6,12,13)$

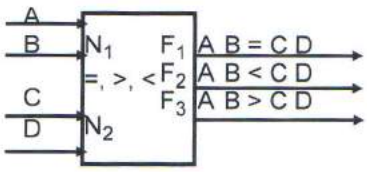
Z jemanjem tega X-sa za "1" lahko minterm 9 pokrijemo z 2-dimenzionalno kocko namesto z 1-dimenzionalno.

Brez upoštevanja nedoločenih vrednosti: F = A'D + B' C' D

Z upoštevanjem nedoločenih vrednosti: F = A' D + C' D

Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni primerjalnik



Blokovni diagram

A	B	C	D	F_1	F_2	F_3
0	0	0	0	1	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	1	0	0
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	1	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	1	0	0

Pravilnostna tabela

Za vsako od treh izhodnih funkcij potrebujemo K-diagram s štirimi spremenljivkami.

Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni primerjalnik

CD \ AB	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

K-diagram za F_1

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

K-diagram za F_2

CD \ AB	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

K-diagram za F_3

F1 =

F2 =

F3 =

Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni primerjalnik

CD \ AB	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

K-diagram za F_1

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

K-diagram za F_2

CD \ AB	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

K-diagram za F_3

$$F1 = A'B'C'D' + A'BC'D + ABCD + AB'CD'$$

$$F2 = A'B'D + A'C + B'CD$$

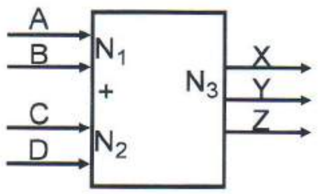
$$F3 = BC'D' + AC' + ABD'$$

$$F1 = A'C'(B'D' + BD) + AC(B'D' + BD) = (A'C' + AC)\bullet(B'D' + BD)$$

$$= (A \equiv C)\bullet(B \equiv D) = \overline{(A \oplus C)}\bullet\overline{(B \oplus D)}$$

Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni seštevalnik



Blokovni diagram

A	B	C	D	X	Y	Z
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Pravilnostna tabela

Za vsako od treh izhodnih funkcij potrebujemo K-diagram s štirimi spremenljivkami.

Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni seštevalnik

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	1

K-diagram za X

CD \ AB	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

K-diagram za Y

CD \ AB	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

K-diagram za Z

X =

Z =

Y =

Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni seštevalnik

CD \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	1	1	1
10	0	0	1	1

K-diagram za X

CD \ AB	00	01	11	10
00	0	0	1	1
01	0	1	0	1
11	1	0	1	0
10	1	1	0	0

K-diagram za Y

CD \ AB	00	01	11	10
00	0	1	1	0
01	1	0	0	1
11	1	0	0	1
10	0	1	1	0

K-diagram za Z

Celice 1 po diagonali sugerirajo uporabo vrat XOR.

$$X = A\,C + B\,C\,D + A\,B\,D$$

$$Z = B\,D' + B'\,D = B \oplus D$$

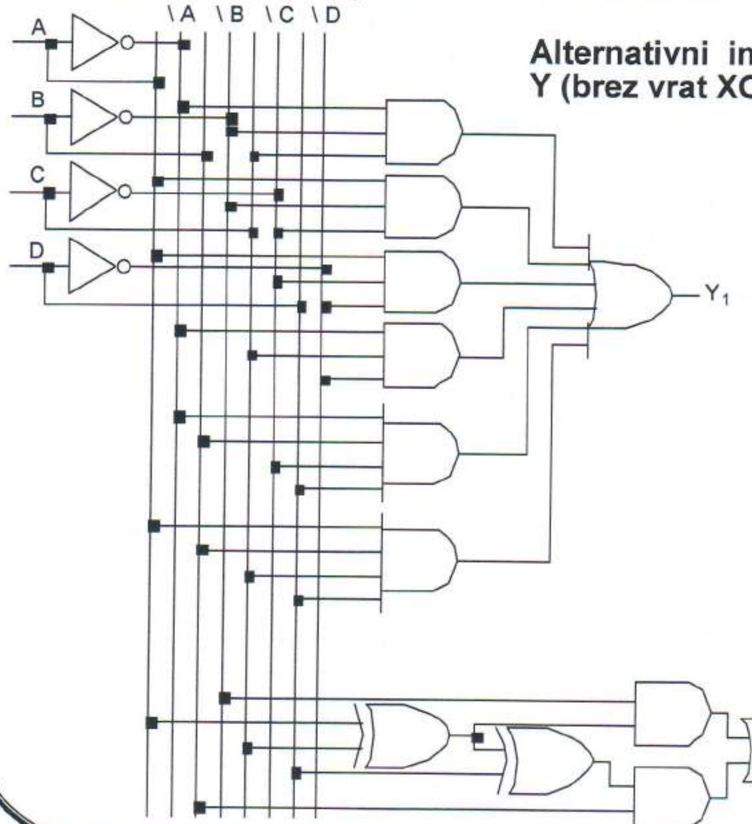
$$Y = A'\,B'\,C + A\,B'\,C' + A'\,C\,D' + A\,C'\,D' + A'\,B\,C'\,D + A\,B\,C\,D$$

Če uporabimo vrata XOR, zmanjšamo število potrebnih vrat in literalov.

$$Y = A'\,B'\,C + A\,B'\,C' + A'\,B\,C'\,D + A'\,B\,C\,D' + A\,B\,C'\,D' + A\,B\,C\,D$$
$$= B'\,(A \oplus C) + A'\,B\,(C \oplus D) + A\,B\,(\overline{C \oplus D})$$
$$= B'\,(A \oplus C) + B(A'(C \oplus D) + A(\overline{C \oplus D}))$$
$$= B'\,(A \oplus C) + B\,(A \oplus C \oplus D)$$

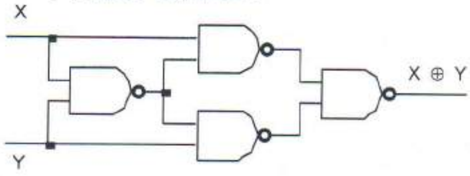
Karnaughova metoda minimizacije

Primer načrtovanja: Dvobitni seštevalnik



Alternativni implementaciji funkcije Y (brez vrat XOR in z vrati XOR).

Opomba: **implementacija vrat XOR tipično zahteva 4 vrata NAND.**



Karnaughova metoda minimizacije

Krožni BCD števnik

W

CD \ AB	00	01	11	10
00	0	0	X	1
01	0	0	X	0
11	0	1	X	X
10	0	0	X	X

X

CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	0	X	X
10	0	1	X	X

Y

CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	0
11	0	0	X	X
10	1	1	X	X

Z

CD \ AB	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

W =

X =

Y =

Z =

Karnaughova metoda minimizacije

Krožni BCD števnik

W

CD \ AB	00	01	11	10
00	0	0	X	1
01	0	0	X	0
11	0	1	X	X
10	0	0	X	X

X

CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	0	X	X
10	0	1	X	X

Y

CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	0
11	0	0	X	X
10	1	1	X	X

Z

CD \ AB	00	01	11	10
00	1	1	X	1
01	0	0	X	0
11	0	0	X	X
10	1	1	X	X

W = B C D + A D' **Y = A' C' D + C D'**

X = B C' + B D' + B' C D **Z = D'**

Karnaughova metoda minimizacije

Vsebovalniki, glavni vsebovalniki in potrebni glavni vsebovalniki v K-diagramu

Vsebovalniki v K-diagramu so posamezne celice 1 ali katerekoli grupe celic 1, ki jih lahko združimo v podkocko.

Glavni vsebovalniki v K-diagramu so največje grupe celic 1, ki jih lahko združimo v podkocko.

Potrebni glavni vsebovalniki v K-diagramu so tisti glavni vsebovalniki, ki sami pokrivajo kakšno celico 1.

Cilj minimizacije

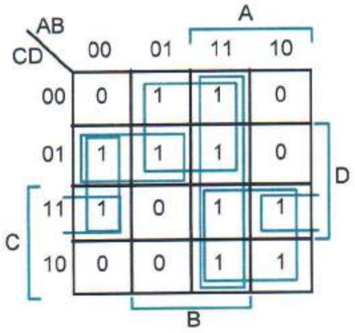
Vsebovalnike povečujemo do glavnih vsebovalnikov.

Celice 1 pokrijemo s čim manjšim številom glavnih vsebovalnikov.

Potrebni glavni vsebovalniki nastopajo v *vseh* možnih pokritjih.

Karnaughova metoda minimizacije

Primeri za ilustracijo glavnih in potrebnih glavnih vsebovalnikov

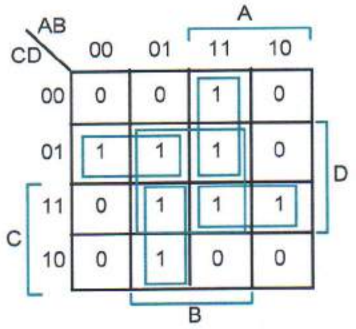


6 glavnih vsebovalnikov:

A' B' D, B C', A C, A' C' D, A B, B' C D

potrebna

MDNO: F = B C' + A C + A' B' D



5 glavnih vsebovalnikov:

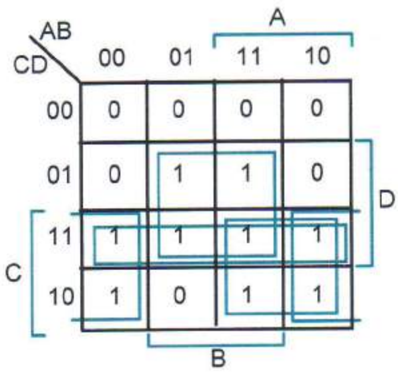
B D, A B C', A C D, A' B C, A' C' D

potrebni

MDNO: F = A B C' + A C D + A' B C + A' C' D

Karnaughova metoda minimizacije

Primeri za ilustracijo glavnih in potrebnih glavnih vsebovalnikov



4 glavni vsebovalniki:

B D, C D, A C, B' C

potrebni

MDNO: F = B D + A C + B' C

Karnaughova metoda minimizacije

Algoritem za iskanje MDNO s pomočjo K-diagrama

Korak 1: Za funkcijo narišemo K-diagram in vanj vnesemo funkcijske vrednosti (0, 1, X).

Korak 2: Izberemo neki element iz množice "ON-set". Poiščemo vse "maksimalne" grupe celic 1 in celic X (število celic v grupi je enako potenci števila 2), ki pokrijejo ta element. Te grupe so glavni vsebovalniki. Korak 2 ponovimo za vsak element iz množice "ON-set", da najdemo vse glavne vsebovalnike.

Korak 3: Pregledamo celice 1 v K-diagramu. Če je celica 1 pokrita z enim samim glavnim vsebovalnikom, je ta potreben, in nastopa v končnem pokritju. Celic 1 v potrebnem glavnem vsebovalniku ni treba več pregledovati.

Korak 4: Če v K-diagramu ostanejo celice 1, ki niso pokrite s potrebnimi glavnimi vsebovalniki, izberemo najmanjšo množico glavnih vsebovalnikov , ki jih pokrijejo. Če obstaja več takih množic, izberemo tisto, ki ima najmanjše možno število literalov.

Opomba: Preklopna funkcija ima lahko več enakovrednih minimalnih oblik.

Karnaughova metoda minimizacije

Primer: F(A,B,C,D) = $\Sigma(4,5,6,8,9,10,13) + \Delta(0,7,15)$

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Začetni K-diagram

Karnaughova metoda minimizacije

Primer: $F(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + \Delta(0,7,15)$

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Začetni K-diagram

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Glavni vsebovalniki okrog A' B C' D'

Karnaughova metoda minimizacije

Primer: $F(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + \Delta(0,7,15)$

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Začetni K-diagram

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

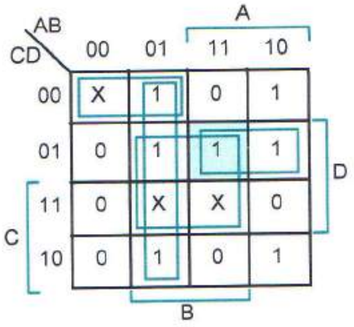
Glavni vsebovalniki okrog A' B C' D'

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Glavni vsebovalniki okrog A' B C' D

Karnaughova metoda minimizacije

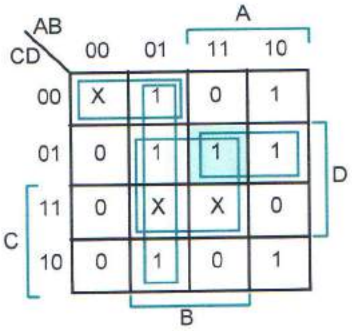
Nadaljevanje primera

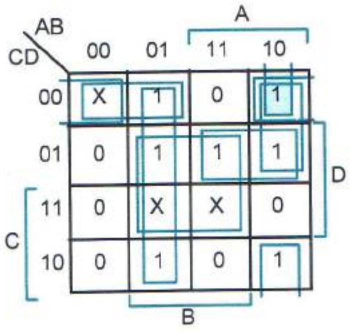


Glavni vsebovalniki okrog A B C' D

Karnaughova metoda minimizacije

Nadaljevanje primera





Glavni vsebovalniki okrog A B C' D

Glavni vsebovalniki okrog A B' C' D'

Karnaughova metoda minimizacije

Nadaljevanje primera

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Glavni vsebovalniki okrog A B C' D

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Glavni vsebovalniki okrog A B' C' D'

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Potrebni glavni vsebovalniki in minimalno pokritje

MDNO: F(A, B, C, D) = A'B + AB'D' + AC'D

Karnaughova metoda minimizacije

Primer K-diagrama s petimi spremenljivkami

F(A, B, C, D, E) = Σ(2,5,7,8,10,13,15,17,19,21,23,24,29,31)

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

MDNO: F(A, B, C, D, E) =

Karnaughova metoda minimizacije

Primer K-diagrama s petimi spremenljivkami

F(A, B, C, D, E) = Σ(2,5,7,8,10,13,15,17,19,21,23,24,29,31)

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

MDNO: F(A, B, C, D, E) = CE + AB'E + BC'D'E' + A'C'DE'

Karnaughova metoda minimizacije

Primer K-diagrama s šestimi spremenljivkami

F(A, B, C, D, E, F) = Σ(2,8,10,18,24,26,34,37,42,45,50,53,58,61)

DEF \ ABC	000	001	011	010	110	111	101	100
000	0	1	1	0	0	0	0	0
001	0	0	0	0	0	0	0	0
011	0	0	0	0	0	0	0	0
010	1	1	1	1	1	1	1	1
110	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	0	0
101	0	0	0	0	1	1	1	1
100	0	0	0	0	0	0	0	0

F(A, B, C, D, E, F) =

Karnaughova metoda minimizacije

Primer K-diagrama s šestimi spremenljivkami

F(A, B, C, D, E, F) = Σ(2,8,10,18,24,26,34,37,42,45,50,53,58,61)

DEF \ ABC	000	001	011	010	110	111	101	100
000	0	1	1	0	0	0	0	0
001	0	0	0	0	0	0	0	0
011	0	0	0	0	0	0	0	0
010	1	1	1	1	1	1	1	1
110	0	0	0	0	0	0	0	0
111	0	0	0	0	0	0	0	0
101	0	0	0	0	1	1	1	1
100	0	0	0	0	0	0	0	0

F(A, B, C, D, E, F) = D'EF' + ADE'F + A'CD'F'

Karnaughova metoda minimizacije

Primer iskanja MKNO s pomočjo K-diagrama

F(A,B,C,D) = Σ(0,2,3,5,6,7,8,10,11,14,15)

Korak 1: Poiščemo MDNO za funkcijo F' (pokrivamo ničle).

Korak 2: Obe strani izraza za F' negiramo in tako dobimo MKNO za funkcijo F.

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	0
11	1	1	1	1
10	1	1	1	1

F = (B' + C + D)•(A' + C + D')•(B + C + D')

F' = BC'D' + AC'D + B'C'D

(F')' = (BC'D' + AC'D + B'C'D)'

MKNO: F = (B' + C + D)•(A' + C + D')•(B + C + D')

Karnaughova metoda minimizacije

Primer iskanja MKNO nepopolno določene funkcije s pomočjo K-diagrama

F(A,B,C,D) = Σ(1,3,5,7,9) + Δ(6,12,13)

CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

(A: columns 11, 10; B: columns 01, 11; C: rows 11, 10; D: rows 01, 11)

F' =

(F')'=

MKNO: F =

Karnaughova metoda minimizacije

Primer iskanja MKNO nepopolno določene funkcije s pomočjo K-diagrama

F(A,B,C,D) = Σ(1,3,5,7,9) + Δ(6,12,13)

CD \ AB	00	01	11	10
00	0	0	X	0
01	1	1	X	1
11	1	1	0	0
10	0	X	0	0

(A: columns 11, 10; B: columns 01, 11; C: rows 11, 10; D: rows 01, 11)

F' = D' + AC

(F')' = (D' + AC)'

MKNO: F = D(A' + C')

Veitcheva metoda minimizacije

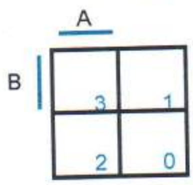
Veitchev diagram

Veitchev diagram je matrična logična shema, na kateri temelji grafična metoda za ročno minimizacijo preklopnih funkcij do 6 spremenljivk.

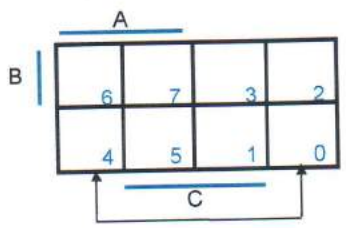
Veitcheva metoda minimizacije preklopnih funkcij je enaka kot Karnaughova. Razlika je le v tem, da so celice v Veitchevem diagramu drugače razmeščene kot v Karnaughovem, zato veljajo (za n > 4) drugačna pravila o logični sosednosti stolpcev in vrstic.

Oglejmo si splošne oblike Veitchevih diagramov za preklopne funkcije z 2, 3, 4, 5 in 6 spremenljivkami.

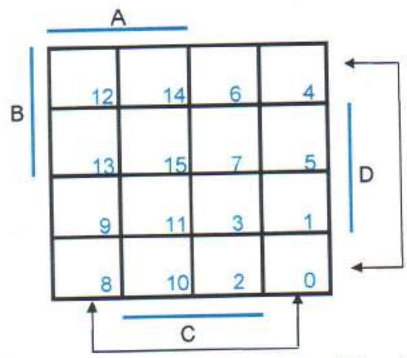
n = 2



n = 3



n = 4

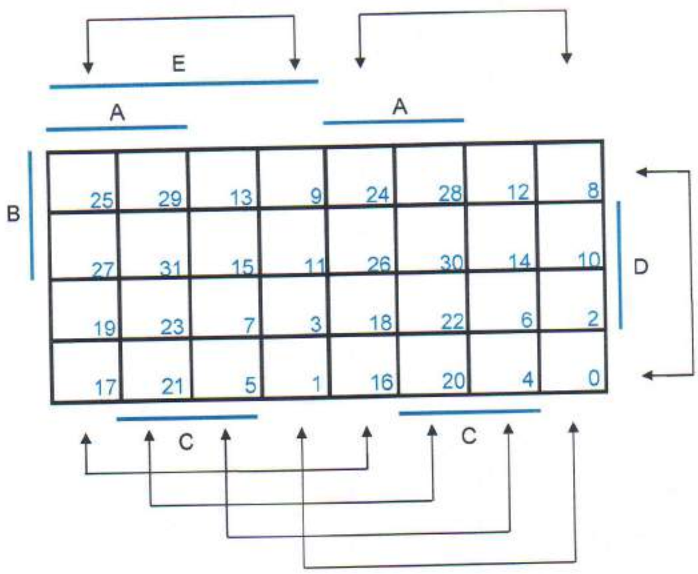


Puščice označujejo fizično nesosedne vrstice oz. fizično nesosedne stolpce, ki so logično sosedni.

Veitcheva metoda minimizacije

Veitchev diagram

n = 5



Puščice označujejo fizično nesosedne vrstice oz. fizično nesosedne stolpce, ki so logično sosedni.

***Opozorilo:* Četrti in peti stolpec sta fizično sosedna, vendar *nista* logično sosedna (razlikujeta se v spremenljivkah A in E).**

Veitcheva metoda minimizacije

Veitchev diagram

n = 6

Opozorilo: **Četrti in peti stolpec sta fizično sosedna, vendar *nista* logično sosedna (razlikujeta se v spremenljivkah A in E). Četrta in peta vrstica sta fizično sosedni, vendar nista logično sosedni (razlikujeta se v spremenljivkah B in F).**

		E	E	E	E					
		A	A			A	A			
B	F	51	59	27	19	49	57	25	17	
B	F	55	63	31	23	53	61	29	21	D
	F	39	47	15	7	37	45	13	5	D
	F	35	43	11	3	33	41	9	1	
B		50	58	26	18	48	56	24	16	
B		54	62	30	22	52	60	28	20	D
		38	46	14	6	36	44	12	4	D
		34	42	10	2	32	40	8	0	
			C	C			C	C		

Puščice označujejo fizično nesosedne vrstice oz. fizično nesosedne stolpce, ki so logično sosedni.

Veitcheva metoda minimizacije

Primer Veitchevega diagrama s petimi spremenljivkami

$F(A, B, C, D, E) = \Sigma(2,7,12,13,18,20,24,25,27) + \Delta(0,3,8,10,16,22,26,28,29,30)$

MDNO:

F =

	E	E	E	E					
	A	A			A	A			
B	1	X	1	0	1	X	1	X	
B	1	0	0	0	X	X	0	X	D
	0	0	1	X	1	X	0	1	D
	0	0	0	0	X	1	0	X	
		C	C			C	C		

	E	E	E	E					
	A	A			A	A			
B	1	X	1	0	1	X	1	X	
B	1	0	0	0	X	X	0	X	D
	0	0	1	X	1	X	0	1	D
	0	0	0	0	X	1	0	X	
		C	C			C	C		

MKNO:

F' =

F =

Veitcheva metoda minimizacije

Primer Veitchevega diagrama s petimi spremenljivkami

$$F(A, B, C, D, E) = \Sigma(2,7,12,13,18,20,24,25,27) + \Delta(0,3,8,10,16,22,26,28,29,30)$$

MDNO:

$$F = AE' + C'E' + ABC' + BCD' + A'B'DE$$

1	X	1	0	1	X	1	X
1	0	0	0	X	X	0	X
0	0	1	X	1	X	0	1
0	0	0	0	X	1	0	X

1	X	1	0	1	X	1	X
1	0	0	0	X	X	0	X
0	0	1	X	1	X	0	1
0	0	0	0	X	1	0	X

MKNO:

$$F' = A'B'D' + A'BC' + AB'E + CDE' + BCD$$

$$F = (A + B + D)\bullet(A + B' + C)\bullet(A' + B + E')\bullet(C' + D' + E)\bullet(B' + C' + D')$$

Quine-McCluskeyeva metoda minimizacije

Je tabelarična metoda za minimizacijo preklopnih funkcij. Izpeljemo jo v dveh delih.

Primer: $F(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + \Delta(0,7,15)$

<u>*1. del:*</u> Poiščemo vse glavne vsebovalnike.

Korak 1: **V prvi stolpec vpišemo kodne besede mintermov iz množice "ON-set" in množice "DC-set". Grupiramo jih po številu enic.**

Tabela vsebovalnikov

Stolpec 1	
0000	
0100	
1000	
0101	
0110	
1001	
1010	
0111	
1101	
1111	

Quine-McCluskeyeva metoda minimizacije

Je tabelarična metoda za minimizacijo preklopnih funkcij. Izpeljemo jo v dveh delih.

Primer: **F(A,B,C,D) = Σ(4,5,6,8,9,10,13) + Δ(0,7,15)**

1. del: **Poiščemo vse glavne vsebovalnike.**

Korak 1: **V prvi stolpec vpišemo kodne besede mintermov iz množice "ON-set" in množice "DC-set". Grupiramo jih po številu enic.**

Korak 2: **Uporabimo minimizacijski izrek. Elemente grupe z N enicami primerjamo z elementi grupe z N+1 enicami. Razlika v enem bitu pomeni sosednost. Odstranimo spremenljivko in ostanek vpišemo v naslednji stolpec. Npr., 0000 in 0100 dajeta 0-00, 0000 in 1000 dajeta -000. Vse elemente, ki imajo sosede, označimo s kljukicami. Preostali elementi so glavni vsebovalniki. Označimo jih z zvezdicami. Korak 2 ponavljamo, dokler obstajajo sosedni elementi.**

Tabela vsebovalnikov

Stolpec 1	Stolpec 2	
0000 ✓	0-00	
	-000	
0100 ✓		
1000 ✓	010-	
	01-0	
0101 ✓	100-	
0110 ✓	10-0	
1001 ✓		
1010 ✓	01-1	
	-101	
0111 ✓	011-	
1101 ✓	1-01	
1111 ✓	-111	
	11-1	

Quine-McCluskeyeva metoda minimizacije

Je tabelarična metoda za minimizacijo preklopnih funkcij. Izpeljemo jo v dveh delih.

Primer: **F(A,B,C,D) = Σ(4,5,6,8,9,10,13) + Δ(0,7,15)**

1. del: **Poiščemo vse glavne vsebovalnike.**

Korak 1: **V prvi stolpec vpišemo kodne besede mintermov iz množice "ON-set" in množice "DC-set". Grupiramo jih po številu enic.**

Korak 2: **Uporabimo minimizacijski izrek. Elemente grupe z N enicami primerjamo z elementi grupe z N+1 enicami. Razlika v enem bitu pomeni sosednost. Odstranimo spremenljivko in ostanek vpišemo v naslednji stolpec. Npr., 0000 in 0100 dajeta 0-00, 0000 in 1000 dajeta -000. Vse elemente, ki imajo sosede, označimo s kljukicami. Preostali elementi so glavni vsebovalniki. Označimo jih z zvezdicami. Korak 2 ponavljamo, dokler obstajajo sosedni elementi.**

Tabela vsebovalnikov

Stolpec 1	Stolpec 2	Stolpec 3
0000 ✓	0-00 *	01-- *
	-000 *	
0100 ✓		-1-1 *
1000 ✓	010- ✓	
	01-0 ✓	
0101 ✓	100- *	
0110 ✓	10-0 *	
1001 ✓		
1010 ✓	01-1 ✓	
	-101 ✓	
0111 ✓	011- ✓	
1101 ✓	1-01 *	
1111 ✓	-111 ✓	
	11-1 ✓	

Quine-McCluskeyeva metoda minimizacije

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Glavni vsebovalniki:

0-00 = A' C' D'   **-000 = B' C' D'**

100- = A B' C'   **10-0 = A B' D'**

1-01 = A C' D   **01-- = A' B**

-1-1 = B D

Quine-McCluskeyeva metoda minimizacije

CD \ AB	00	01	11	10
00	X	1	0	1
01	0	1	1	1
11	0	X	X	0
10	0	1	0	1

Glavni vsebovalniki:

0-00 = A' C' D'   **-000 = B' C' D'**

100- = A B' C'   **10-0 = A B' D'**

1-01 = A C' D   **01-- = A' B**

-1-1 = B D

2. del: **Poiščemo najmanjšo množico glavnih vsebovalnikov, ki pokrijejo vse minterme.**

Potrebni glavni vsebovalniki morajo nastopati v vseh pokritjih.

Drugi del metode izvedemo s pomočjo *tabele pokritja*.

Quine-McCluskeyeva metoda minimizacije

Tabela pokritja

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

Vrstice so glavni vsebovalniki. Stolpci so mintermi. Na presečišče vrstice i in stolpca j zapišemo znak X, če glavni vsebovalnik v vrstici i pokriva minterm v stolpcu j.

Quine-McCluskeyeva metoda minimizacije

Tabela pokritja

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				~~X~~		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	~~X~~	~~X~~	X				
5,7,13,15(–1–1)		X					X

Vrstice so glavni vsebovalniki. Stolpci so mintermi. Na presečišče vrstice i in stolpca j zapišemo znak X, če glavni vsebovalnik v vrstici i pokriva minterm v stolpcu j.

Če je v stolpcu en sam znak X, je glavni vsebovalnik, ki ustreza vrstici z znakom X, potreben in mora nastopati v minimalnem pokritju.

Quine-McCluskeyeva metoda minimizacije

Tabela pokritja

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

Odstranimo vse stolpce, ki so pokriti s potrebnimi glavnimi vsebovalniki.

Quine-McCluskeyeva metoda minimizacije

Tabela pokritja

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

Odstranimo vse stolpce, ki so pokriti s potrebnimi glavnimi vsebovalniki.

	4	5	6	8	9	10	13
0,4(0–00)	X						
0,8(–000)				X			
8,9(100–)				X	X		
8,10(10–0)				X		X	
9,13(1–01)					X		X
4,5,6,7(01– –)	X	X	X				
5,7,13,15(–1–1)		X					X

Poiščemo minimalno množico vrstic, ki pokrijejo preostale stolpce.

MDNO: F = A B' D' + A C' D + A' B

Quine-McCluskeyeva metoda minimizacije

Dominacija vrstic in dominacija stolpcev v tabeli pokritja

	m_a	m_b	m_c	m_d	m_e	m_f	m_g	m_h
P_A		X			X		X	X
P_B	X	X				X		X
P_C	X				X		X	X
P_D		X						X
P_E	X	X		X	X	X	X	X

$P_A \supseteq P_D$

$P_E \supseteq P_D$

P_E dejansko dominira nad vsemi vrsticami.

$m_a \supseteq m_d$

$m_e = m_g$

m_h dominira vsem stolpcem.

$m_a \supseteq m_f$

Definicija. ***Vrstica*** P_i ***dominira vrstici*** P_j, **označimo** $P_i \supseteq P_j$, **če vsebuje vrstica** P_i **križec X v vsakem stolpcu, v katerem vsebuje križec X vrstica** P_j. ■

Definicija. ***Stolpec*** m_i ***dominira stolpcu*** m_j, **označimo** $m_i \supseteq m_j$, **če vsebuje stolpec** m_i **križec X v vsaki vrstici, v kateri vsebuje križec X stolpec** m_j. ■

Quine-McCluskeyeva metoda minimizacije

Dominacija vrstic in dominacija stolpcev

Izrek. **Z upoštevanjem dominacije vrstic in stolpcev se lahko pri reduciranju tabele pokritja ravnamo po naslednjih pravilih:**

1. **Če stolpec** m_i **dominira nad stolpcem** m_j ($m_i \supseteq m_j$), **lahko** ***dominirajoči stolpec*** m_i **zbrišemo iz tabele pokritja.**
2. **Če vrstica** P_i **dominira nad vrstico** P_j ($P_i \supseteq P_j$), **lahko** ***dominirano vrstico*** P_j **zbrišemo iz tabele pokritja, toda le tedaj, če vsebuje** P_j **enako ali več literalov kot** P_i. ■

Lahko se zgodi, da v tabeli pokritja ni potrebnih glavnih vsebovalnikov, dominiranih vrstic ali dominirajočih stolpcev. V takem primeru moramo uporabiti novo metodo–*metodo vejanja*.

Pri metodi vejanja najprej naključno izberemo eno vrstico in nato izpeljemo reduciranje. Celoten proces moramo ponavljati za vsako možno začetno izbiro vrstice in med vsemi rešitvami izbrati tisto, ki je minimalna.

Če izbira ene vrstice po zgornjem načinu ne vodi do celotne rešitve z zagotovljeno minimalno ceno - to je, če imamo še zmeraj tabele pokritja, ki se ne dajo reducirati - izberemo še eno vrstico in preizkušamo možne rešitve z obema zahtevanima vrsticama.

Proces se lahko nadaljuje, tako da izberemo več zahtevanih vrstic.

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
P_A = -0-0	X	X				
P_B = 0-00	X			X		
P_C = 010-				X	X	
P_D = 01-1					X	X
P_E = --11			X			X
P_F = 001-		X	X			

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
$\boxed{P_A = \text{-0-0}}$	X	X				
P_B = 0-00	X			X		
P_C = 010-				X	X	
P_D = 01-1					X	X
P_E = --11			X			X
P_F = 001-		X	X			

Izberemo P_A.

	m_c	m_d	m_e	m_f
~~P_B = 0-00~~		X		
P_C = 010-		X	X	
P_D = 01-1			X	X
P_E = --11	X			X
~~P_F = 001-~~	X			

$P_C \supseteq P_B$

$P_E \supseteq P_F$

	m_c	m_d	m_e	m_f
$\boxed{P_C = \text{010-}}$		X	X	
P_D = 01-1			X	X
$\boxed{P_E = \text{--11}}$	X			X

$\{P_A, P_C, P_E\}$ = {-0-0, 010-, --11}; 3 členi, 7 literalov

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
P_A = -0-0	X	X				
$\boxed{P_B = 0\text{-}00}$	X			X		
P_C = 010-				X	X	
P_D = 01-1					X	X
P_E = --11			X			X
P_F = 001-		X	X			

Izberemo P_B.

	m_b	m_c	m_e	m_f
P_A = -0-0	X			
~~P_C = 010-~~			~~X~~	
P_D = 01-1			X	X
P_E = --11		X		X
P_F = 001-	X	X		

$P_D \supseteq P_C$

	m_b	m_c	m_e	m_f
P_A = -0-0	X			
$\boxed{P_D = 01\text{-}1}$			X	X
P_E = --11		X		X
$\boxed{P_F = 001\text{-}}$	X	X		

$\{P_B, P_D, P_F\}$ = {0-00, 01-1, 001-}; 3 členi, 9 literalov

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
P_A = -0-0	X	X				
P_B = 0-00	X			X		
$\boxed{P_C = 010\text{-}}$				X	X	
P_D = 01-1					X	X
P_E = --11			X			X
P_F = 001-		X	X			

Izberemo P_C.

	m_a	m_b	m_c	m_f
P_A = -0-0	X	X		
~~P_B = 0-00~~	~~X~~			
~~P_D = 01-1~~				~~X~~
P_E = --11			X	X
P_F = 001-		X	X	

$P_A \supseteq P_B$

$P_E \supseteq P_D$

	m_a	m_b	m_c	m_f
$\boxed{P_A = \text{-}0\text{-}0}$	X	X		
$\boxed{P_E = \text{--}11}$			X	X
P_F = 001-		X	X	

$\{P_A, P_C, P_E\}$ = {-0-0, 010-, --11}; 3 členi, 7 literalov

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
P_A = -0-0	X	X				
P_B = 0-00	X			X		
P_C = 010-				X	X	
$\boxed{P_D = 01\text{-}1}$					X	X
P_E = --11			X			X
P_F = 001-		X	X			

Izberemo P_D.

	m_a	m_b	m_c	m_d
P_A = -0-0	X	X		
P_B = 0-00	X			X
~~P_C = 010-~~				X
P_E = --11			X	
P_F = 001-		X	X	

$P_B \supseteq P_C$

	m_a	m_b	m_c	m_d
P_A = -0-0	X	X		
$\boxed{P_B = 0\text{-}00}$	X			X
P_E = --11			X	
$\boxed{P_F = 001\text{-}}$		X	X	

$\{P_B, P_D, P_F\}$ = {0-00, 01-1, 001-}; 3 členi, 9 literalov

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
P_A = -0-0	X	X				
P_B = 0-00	X			X		
P_C = 010-				X	X	
P_D = 01-1					X	X
$\boxed{P_E = \text{--}11}$			X			X
P_F = 001-		X	X			

Izberemo P_E.

	m_a	m_b	m_d	m_e
P_A = -0-0	X	X		
P_B = 0-00	X		X	
P_C = 010-			X	X
~~P_D = 01-1~~				X
~~P_F = 001-~~		X		

$P_A \supseteq P_F$

$P_C \supseteq P_D$

	m_a	m_b	m_d	m_e
$\boxed{P_A = \text{-}0\text{-}0}$	X	X		
P_B = 0-00	X		X	
$\boxed{P_C = 010\text{-}}$			X	X

$\{P_A, P_C, P_E\}$ = {-0-0, 010-, --11}; 3 členi, 7 literalov

Quine-McCluskeyeva metoda minimizacije

Primer uporabe metode vejanja

	m_a	m_b	m_c	m_d	m_e	m_f
P_A = -0-0	X	X				
P_B = 0-00	X			X		
P_C = 010-				X	X	
P_D = 01-1					X	X
P_E = --11			X			X
$\boxed{P_F = 001-}$		X	X			

Izberemo P_F.

	m_a	m_d	m_e	m_f
P_A = -0-0	X			
P_B = 0-00	X	X		
P_C = 010-		X	X	
P_D = 01-1			X	X
P_E = --11				X

Tabele pokritja ne moremo reducirati, zato izberemo poleg P_F še en zahtevan glavni vsebovalnik in iščemo možne rešitve. Ugotovili bi, da če je med izbranimi glavnimi vsebovalniki tudi P_F, v nobeni veji ne dobimo minimalnega pokritja.

Končni rezultat: Minimalno pokritje je množica glavnih vsebovalnikov $\{P_A, P_C, P_E\}$ = {-0-0, 010-, --11}, ki jo dobimo z vejanjem na P_A, P_C ali P_E .

Quine-McCluskeyeva metoda minimizacije

Iskanje MKNO s Quine-McCluskeyevo metodo

Za iskanje MKNO s Quine-McCluskeyevo metodo so potrebne enake modifikacije kot pri že obravnavanih pristopih s Karnaughovim ali Veitchevim diagramom.

Namesto z mintermi v proceduri delamo z makstermi in namesto z glavnimi vsebovalniki s "prime implicates".

Uporabljamo enak zapis kodnih besed, le da se kodne besede interpretirajo kot vsote, ne pa kot produkti, in vrednost 0 ustreza nekomplementirani spremenljivki ter vrednost 1 komplementirani spremenljivki.

Primer: S Quine-McCluskeyevo metodo poiščite MKNO funkcije F:

$$F(A, B, C, D, E) = \Pi(1,4,5,6,9,11,14,15,17,19,21,23,31)\bullet$$
$$\bullet\Delta(0,3,8,10,16,22,26,28,29,30)$$

Rešitev:

$$F = (A + B + D)\bullet(A' + B + E')\bullet(C' + D' + E)\bullet(B' + C' + D')\bullet(A + C + E')$$

CAD orodja za minimizacijo

Program Espresso

Problem pri Quine-McCluskeyevi metodi minimizacije je ta, da število glavnih vsebovalnikov zelo hitro raste s številom vhodov.

Zgornja meja števila glavnih vsebovalnikov je $3^n/n$, kjer je n število vhodov.

Iskanje minimalnega pokritja je NP-težak problem, tj., računsko zahteven proces, zato ni verjetno, da bi zanj obstajal učinkovit algoritem.

Espresso daje prednost hitri rešitvi pred minimalnostjo rešitve.

Ne generira *vseh* glavnih vsebovalnikov, kot to počnemo v prvem delu Quine-McCluskeyeve metode.

Tehtno izbere podmnožico glavnih vsebovalnikov, ki še vedno pokrivajo vse minterme.

Deluje podobno kot ljudje pri iskanju glavnih vsebovalnikov v K-diagramu.

CAD orodja za minimizacijo

Osnove programa Espresso

1. Razširi vsebovalnike do njihove maksimalne velikosti. Vsebovalnikov, pokritih z razširjenim vsebovalnikom, več ne obravnava. Kvaliteta rezultata je odvisna od vrstnega reda razširjanja vsebovalnikov. Za določanje vrstnega reda uporablja hevristične metode. Korak se imenuje EXPAND.

2. Iz razširjenih glavnih vsebovalnikov se dobi neredundantno pokritje (tj., nobena prava podmnožica glavnih vsebovalnikov ni pokritje), podobno kot v 2. delu Quine-McCluskeyeve metode. Korak se imenuje IRREDUNDANT COVER.

3. Rešitev je navadno kar dobra, toda včasih se lahko še izboljša. Obstaja lahko drugo pokritje z manjšim številom členov ali literalov. Skrči glavne vsebovalnike na najmanjšo velikost, ki še zmeraj pokrije vse minterme. Korak se imenuje REDUCE.

4. Ponavlja zaporedje REDUCE/EXPAND/IRREDUNDANT COVER, da najde alternativne glavne vsebovalnike. To ponavlja tako dolgo, dokler nova pokritja izboljšujejo zadnjo rešitev.

5. Uporablja še številne druge strategije (npr. zgodnje ugotavljanje in odstranitev potrebnih glavnih vsebovalnikov).

CAD orodja za minimizacijo

Vhod in izhod v programu Espresso

$$F(A,B,C,D) = \Sigma(4,5,6,8,9,10,13) + \Delta(0,7,15)$$

Vhod v Espresso

```
.i 4        -- štev. vhodov
.o 1        -- štev. izhodov
.ilb a b c d -- imena vhodov
.ob f       -- ime izhoda
.p 10       -- število produktnih členov
0100  1     -- A'BC'D'
0101  1     -- A'BC'D
0110  1     -- A'BCD'
1000  1     -- AB'C'D'
1001  1     -- AB'C'D
1010  1     -- AB'CD'
1101  1     -- ABC'D
0000  -     -- A'B'C'D' nepomembno
0111  -     -- A'BCD nepomembno
1111  -     -- ABCD nepomembno
.e          -- konec seznama
```

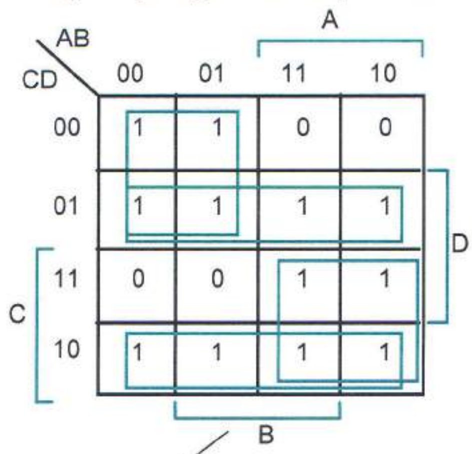
Izhod iz Espressa

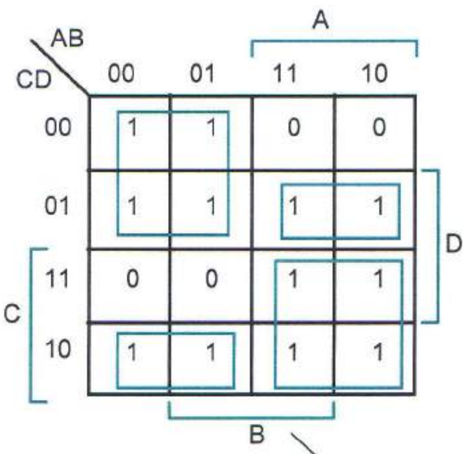
```
.i 4
.o 1
.ilb a b c d
.ob f
.p 3
1-01  1
10-0  1
01--  1
.e
```

$$F = A\,C'\,D + A\,B'\,D' + A'\,B$$

CAD orodja za minimizacijo

Iteracije v programu Espresso





Začetna množica glavnih vsebovalnikov, ki jo najdeta 1. in 2. korak v Espressu.

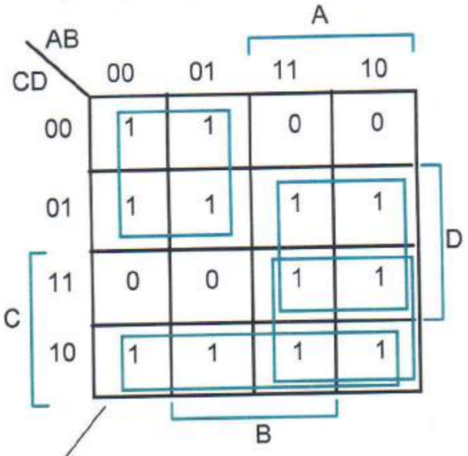
Dobi glavne vsebovalnike in neredundantno pokritje, ki pa ni minimalno.

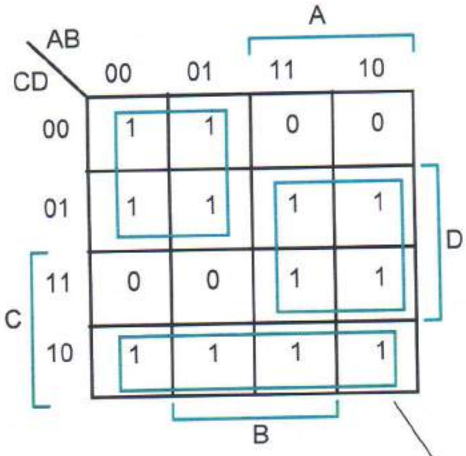
Rezultat koraka REDUCE: Skrči glavne vsebovalnike tako, da so še zmeraj pokriti vsi mintermi.

Izbira vrstnega reda, v katerem se izvede skrčenje, je pomembna.

CAD orodja za minimizacijo

Iteracije v programu Espresso





Drugi korak EXPAND generira drugačno množico glavnih vsebovalnikov.

Drugi korak IRREDUNDANT COVER najde minimalno pokritje, ki vsebuje samo tri glavne vsebovalnike.

Povzetek poglavja

Predstavili smo:

- **minimizacijo logike**

Naš cilj je bil dobiti dvonivojsko realizacijo logike z najmanjšim številom vrat in najmanjšim številom vhodov v vrata.

Dosegli smo ga:

- **z algebrskim poenostavljanjem s postulati in izreki Boolove algebre ali**
- **z Boolovimi kockami (do 4 spremenljivke) ali**
- **s Karnaughovimi diagrami (do 6 spremenljivk) ali**
- **z Veitchevimi diagrami (do 6 spremenljivk) ali**
- **s Quine-McCluskeyevim algoritmom ali**
- **s CAD orodjem Espresso.**

5. Večnivojska kombinacijska vezja

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Vsebina poglavja

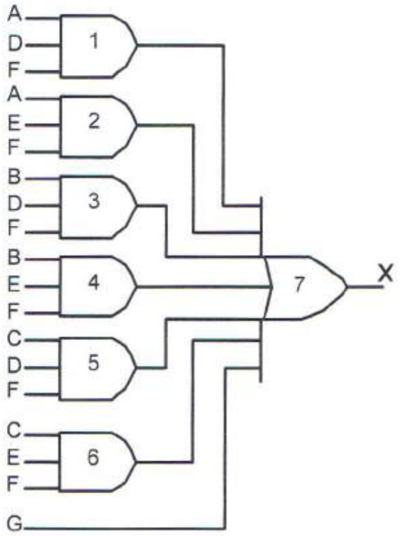
- Večnivojska logika
  - Gradnik AOI
  - Sinteza večnivojske logike
  - CAD orodje misII
- Časovni odziv v kombinacijskih vezjih
  - Zakasnitev vrat
  - Hazardi

Večnivojska logika: Prednosti

Funkcija v MDNO:

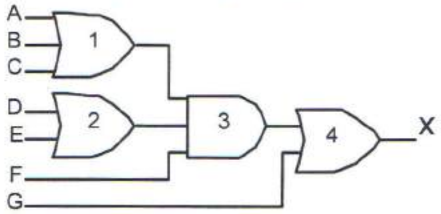
$$X = ADF + AEF + BDF + BEF + CDF + CEF + G$$

Za implementacijo potrebujemo: šest 3-vhodnih vrat AND in ena 7-vhodna vrata OR (mogoče sploh ne obstajajo). Vezje ima 25 povezav (19 literalov in 6 internih povezav).



Funkcija v faktorizirani obliki:

$$X = (A + B + C)(D + E)F + G$$

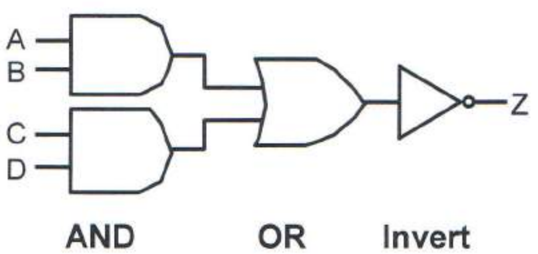


Za implementacijo potrebujemo: ena 3-vhodna vrata OR, dvoje 2-vhodnih vrat OR in ena 3-vhodna vrata AND. Vezje ima 10 povezav (7 literalov in 3 interne povezave).

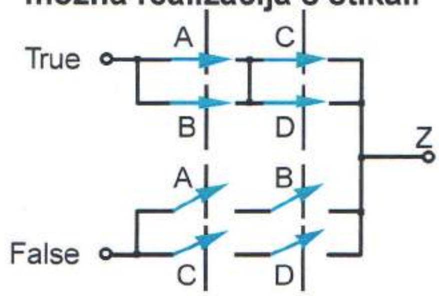
Večnivojska logika: Gradnik AOI

Gradnik AOI (AND-OR-Invert) je trinivojsko logično vezje, sestavljeno iz več vrat AND, enih vrat OR in invertorja.

logična shema

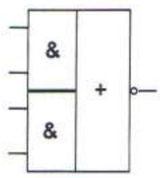


možna realizacija s stikali

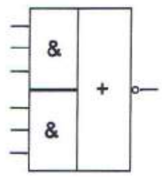


2-vhodni 2-skladovni gradnik AOI

simbol gradnika 2x2 AOI



simbol gradnika 3x2 AOI



Dualni gradnik k AOI se imenuje OAI (OR-AND-Invert).

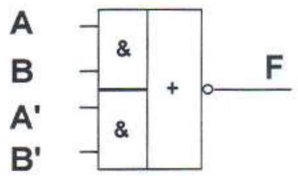
Večnivojska logika: Gradnik AOI

Če želimo funkcijo realizirati z gradnikom AOI, zapišemo komplement funkcije v obliki vsote produktov, iz katere odčitamo literale, ki jih moramo pripeljati na vhode gradnika.

Primer: Implementacija funkcije XOR

$$F = A \oplus B = A'B + AB'$$

$$F' = (A'B + AB')' = (A+B')\,(A'+B) = AB + A'B'$$



Večnivojska logika: Gradnik AOI

Primer:

C \ AB	00	01	11	10
0	0	1	1	1
1	0	0	1	0

A (stolpca 11, 10); B (stolpca 01, 11)

K-diagram za F

$F = \Sigma(2,4,6,7) = B\,C' + A\,C' + A\,B$ **MDNO za F**

$F' = A'\,B' + A'\,C + B'\,C$ **MDNO za F'**

Implementacija z 2-vhodnim 3-skladovnim gradnikom AOI.

$F = (A + B)\,(A + C')\,(B + C')$ **MKNO za F**

$F' = (B' + C)\,(A' + C)\,(A' + B')$ **MKNO za F'**

Implementacija z 2-vhodnim 3-skladovnim gradnikom OAI.

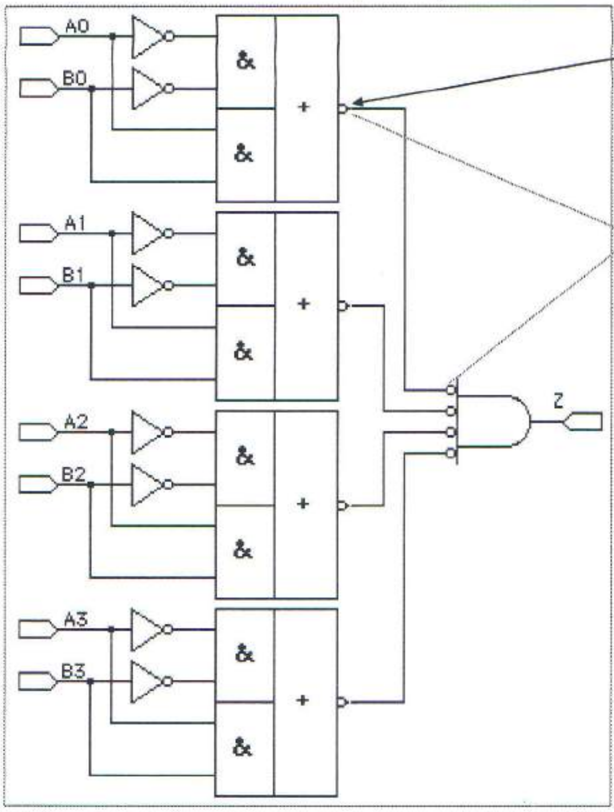
Primer: **Funkcija za enakost dveh 4-bitnih števil**

$$Z = (A0\,B0 + A0'\,B0')\,(A1\,B1 + A1'\,B1')\,(A2\,B2 + A2'\,B2')\,(A3\,B3 + A3'\,B3')$$

Vsak izraz v oklepaju je implementiran z enim gradnikom 2x2 AOI.

Večnivojska logika: Gradnik AOI

Primer: Implementacija vezja za preizkušanje enakosti dveh 4-bitnih števil



Izhod je 0, če A0 = B0.

Izhod je 1, če A0 ≠ B0.

ujemanje krožcev

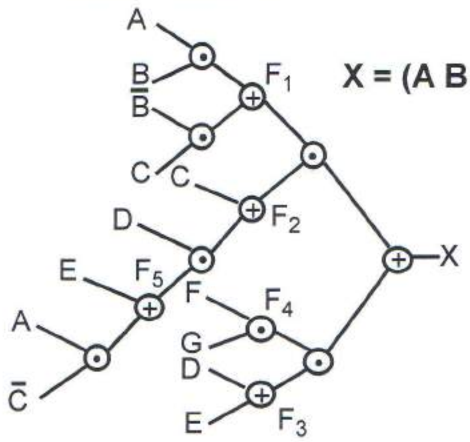
Če so vsi vhodi enaki 0, (aktivni v negativni logiki), je Ai = Bi, 0 ≤ i ≤ 3, in izhod Z je 1.

Večnivojska logika: Sinteza večnivojske logike

Večnivojska optimizacija:

1. **Izpostavimo skupni del logike (zmanjšamo fan-in, povečamo število nivojev vrat) ob upoštevanju časovnih omejitev.**
2. **Faktorizirano obliko implementiramo z razpoložljivimi vrati v knjižnici.**
3. **Minimiziramo število literalov in s tem število povezav.**

Preklopna funkcija v faktorizirani obliki je dejansko vsota produktov vsote produktov ...



X = (A B + B' C) (C + D (E + A C')) + (D + E) (F G)

Večnivojska logika: Sinteza večnivojske logike

Osnovne operacije za manipulacijo večnivojskih vezij:

- **Dekompozicija**
- **Faktorizacija**
- **Ekstrakcija**
- **Substitucija**
- **Sesedanje**

Z vezji manipuliramo tako, da nad njimi interaktivno izvajamo primerne operacije.

Ne obstaja noben algoritem, ki bi zagotavljal, da bomo dobili optimalno večnivojsko vezje.

Večnivojska logika: Sinteza večnivojske logike

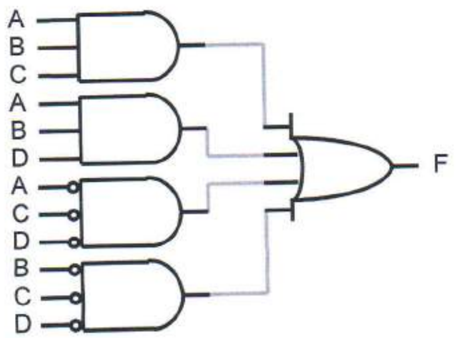
Dekompozicija: **Vzamemo en Boolov izraz in ga zamenjamo z naborom novih izrazov.**

$F = A\,B\,C + A\,B\,D + A'\,C'\,D' + B'\,C'\,D'$ **(12 literalov)**

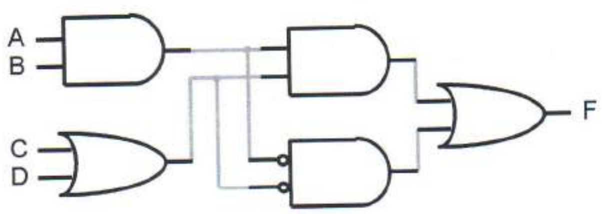
F pretvorimo v naslednjo obliko:

(8 literalov)

$F = X\,Y + X'\,Y'$
$X = A\,B$
$Y = C + D$



pred dekompozicijo (9 vrat)



po dekompoziciji (7 vrat)

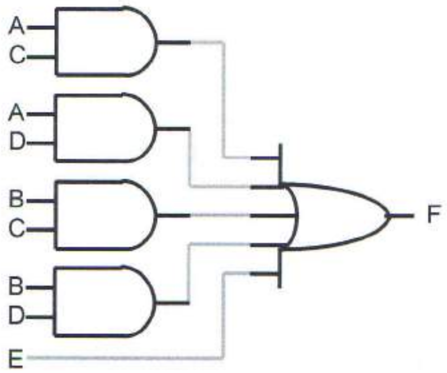
Večnivojska logika: Sinteza večnivojske logike

Faktorizacija: **Vzamemo Boolov izraz v dvonivojski obliki in ga pretvorimo v večnivojsko (faktorizirano) obliko, ne da bi pri tem vnašali kakšne vmesne podfunkcije.**

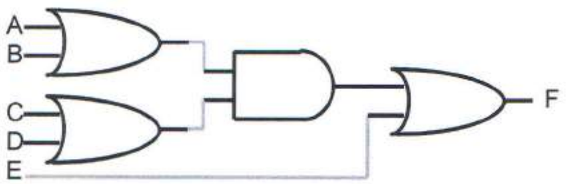
$F = AC + AD + BC + BD + E$ **(9 literalov)**

F pretvorimo v naslednjo obliko:

$F = (A + B)(C + D) + E$ **(5 literalov)**



pred faktorizacijo (5 vrat)



po faktorizaciji (4 vrata)

Večnivojska logika: Sinteza večnivojske logike

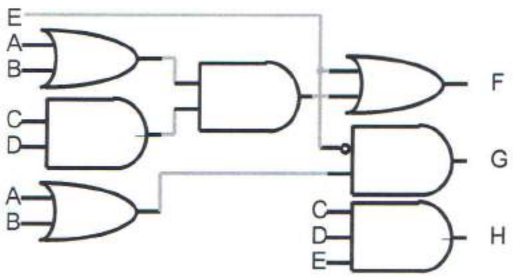
Ekstrakcija: **Odkrijemo skupne podizraze v naboru funkcij, ki so zapisane v faktorizirani obliki.**

$F = (A + B)\,C\,D + E$ **(11 literalov)**
$G = (A + B)\,E'$
$H = C\,D\,E$

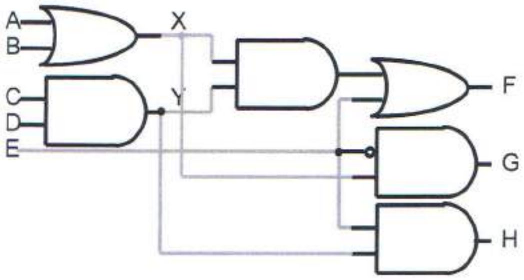
Funkcije pretvorimo v naslednjo obliko:

$F = X\,Y + E$ **(11 literalov)**
$G = X\,E'$
$H = Y\,E$
$X = A + B$
$Y = C\,D$

"jedri": glavna delitelja



pred ekstrakcijo (8 vrat)



po ekstrakciji (7 vrat)

Večnivojska logika: Sinteza večnivojske logike

Substitucija: S substitucijo funkcije G v funkcijo F izrazimo funkcijo F s funkcijo G.

$$F = A + B\,C$$
$$G = A + B$$

F izrazimo z G:

$$F = G\,(A + C)$$

Sesedanje: Je nasprotna operacija od substitucije. Uporablja se za zmanjšanje števila logičnih nivojev, da zadostimo časovnim omejitvam.

$$\begin{aligned} F &= G\,(A + C) \\ &= (A + B)\,(A + C) \\ &= A\,A + A\,B + A\,C + B\,C \\ &= A + B\,C \checkmark \end{aligned}$$

Večnivojska logika: Sinteza večnivojske logike

Ključ za izvedbo teh operacij je "deljenje" Boolovih funkcij.

$$F = \underbrace{P}_{\text{delitelj}}\,\underbrace{Q}_{\text{kvocient}} + \underbrace{R}_{\text{ostanek}}$$

Primer:

$$X = A\,C + A\,D + B\,C + B\,D + E$$
$$Y = A + B$$

X "deljeno" z Y lahko zapišemo kot:

$$X = Y\,(C + D) + E$$

Primer:

$$F = A\,D + B\,C\,D + E$$
$$G = A + B$$

G ne deli F-ja v okviru algebrskih pravil deljenja. Tako imenovana algebrska delitelja F-ja sta D in (A+BC).
G deli F v okviru Boolovih pravil, ki jih je zelo veliko.

$F/G = (A + C)\,D$

F je zapisan kot G Q + R.

$$\begin{aligned} F &= [G\,(A + C)\,D] + E \\ &= (A + B)\,(A + C)\,D + E \\ &= (A\,A + A\,C + A\,B + B\,C)\,D + E \\ &= (A + B\,C)\,D + E \\ &= A\,D + B\,C\,D + E \checkmark \end{aligned}$$

Večnivojska logika: CAD orodje misll

Sinteza popolnega seštevalnika z misll

```
% misll
UC Berkeley, MIS Release #2.1 (compiled 3-Mar-89 at 5:32 PM)
misll> re full.adder
misll> p
   {co} = a b ci + a b ci' + a b' ci + a' b ci
   {sum} = a b ci + a b' ci' + a' b ci' + a' b' ci
misll> pf
   {co} = a b' ci + b (ci (a' + a) + a ci')
   {sum} = ci (a' b' + a b) + ci' (a b' + a' b)
misll> sim1 *
misll> p
   {co} = a b + a ci + b ci
   {sum} = a b ci + a b' ci' + a' b ci' + a' b' ci
misll> pf
   {co} = ci (b + a) + a b
   {sum} = ci (a' b' + a b) + ci' (a b' + a' b)
misll> gd *
misll> pf
   {co} = a [2] + b ci
   {sum} = a' [3]' + a [3]
   [2] = ci + b
   [3] = b' ci' + b ci
```

včita enačbe

dvonivojska minimizacija

dobra dekompozicija

Do te točke je sinteza tehnološko neodvisna.

Večnivojska logika: CAD orodje misll

```
misll> rlib msu.genlib
misll> map
misll> pf
   [361] = b' ci' + a'
   [328] = b'
   [329] = ci'
   {co} = [328]' [329]' + [361]'
   [3] = b ci' + b' ci
   {sum} = [3] a' + [3]' a
misll> pg
[361]    1890:physical 32.00
[328]    1310:physical 16.00
[329]    1310:physical 16.00
{co}     1890:physical 32.00
[3]      2310:physical 40.00
{sum}    2310:physical 40.00
misll> pat
... uporabi model zakasnitev
   iz knjižnice
{sum}   : arrival=( 2.2  2.2)
{co}    : arrival=( 2.2  2.2)
[328]   : arrival=( 1.2  1.2)
[361]   : arrival=( 1.2  1.2)
[329]   : arrival=( 1.2  1.2)
[3]     : arrival=( 1.2  1.2)
ci      : arrival=( 0.0  0.0)
b       : arrival=( 0.0  0.0)
a       : arrival=( 0.0  0.0)
misll> quit
%
```

Včita knjižnico in preslika trenutno tehnološko neodvisno vezje v vezje z vrati iz knjižnice.

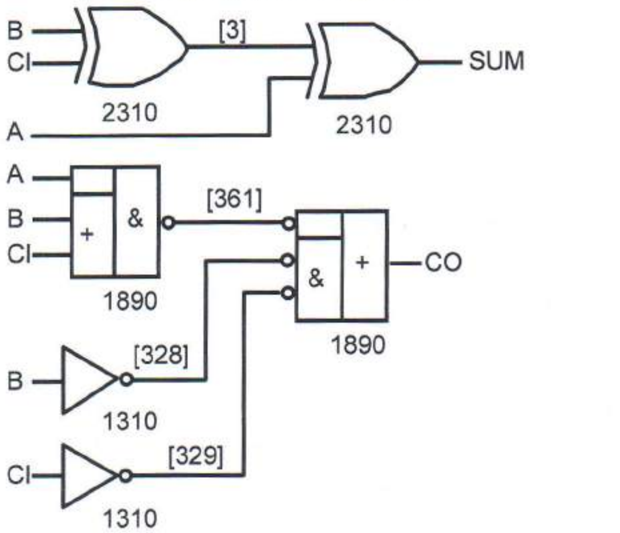
Številke logičnih vrat v vozliščih vezja in njihova relativna površina.

Časovna simulacija: zakasnitev ena časovna enota plus 0.2 časovni enoti na fan-out.

Večnivojska logika: CAD orodje misII

misII in knjižnica logičnih vrat MSU

VLSI standardne celice



Opomba: **Gradnik OR-AND-Invert je ekvivalenten gradniku Invert-AND-OR.**

Številka	Ime	Funkcija
1310	inv	A'
1120	nor2	(A+B)'
1130	nor3	(A+B+C)'
1140	nor4	(A+B+C+D)'
1220	nand2	(A•B)'
1230	nand3	(A•B•C)'
1240	nand4	(A•B•C•D)'
1660	and2/nand2	[A•B, (A•B)']
1670	and3/nand3	[A•B•C, (A•B•C)']
1680	and4/nand4	[A•B•C•D, (A•B•C•D)']
1760	or2/nor2	[A+B, (A+B)']
1770	or3/nor3	[A+B+C, (A+B+C)']
1780	or4	(A+B+C+D)
1870	aoi22	(A•B + C•D)'
1880	aoi21	(A + B•C)'
1860	oai22	[(A + B)(C + D)]'
1890	oai21	[A (B + C)]'
1970	ao22	A•B + D•E
1810	ao222	A•B + C•D + E•F
1910	ao2222	A•B + C•D + E•F + G•H
1930	ao33	A•B•C + D•E•F
2310	xor2	A•B' + A'•B
2350	xnor2	A•B + A'•B'

Večnivojska logika: CAD orodje misII

Dodatni primeri

Standardni klic programa misII

```
misII -f script -t pla <datoteka s pravilnostno tabelo za espresso>
```

Popolni seštevalnik:

```
.model full.adder
.inputs a b ci
.outputs sum co
.names a b ci co sum
1--0 1
-1-0 1
--10 1
111- 1
.names a b ci co
11- 1
1-1 1
-11 1
.end
```

izhod iz programa misII v formatu esspreso

izhodna spremenljivka

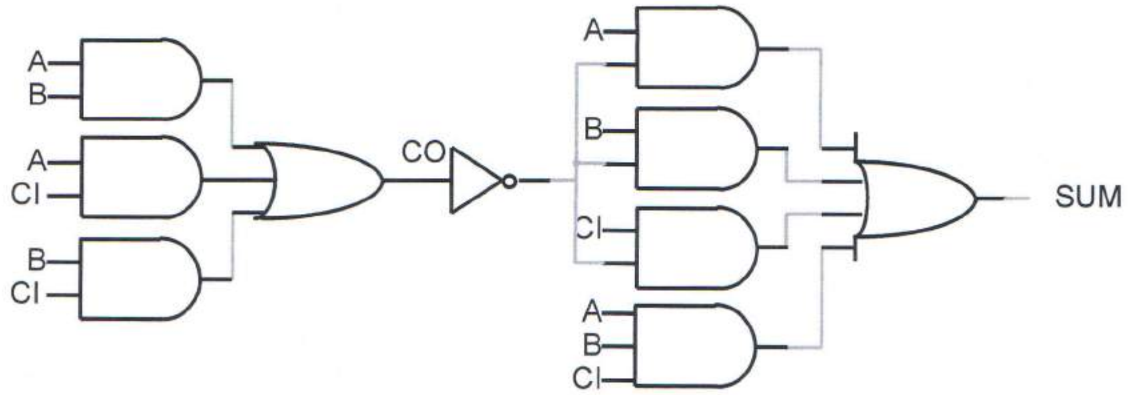
vhodne spremenljivke

SUM = A CO' + B CO' + CI CO' + A B CI (9 literalov)

CO = A B + A CI + B CI (6 literalov)

Opomba: **A ⊕ B ⊕ CI = A' B' CI + A B' CI' + A' B CI' + A B CI (12 literalov!)**

Večnivojska logika: CAD orodje misll



Večnivojska implementacija popolnega seštevalnika: 5 logičnih nivojev!

Večnivojska logika: CAD orodje misll

Dvobitni seštevalnik

```
.inputs a b c d
.outputs x y z
.names a c z [22] x
---1 1
11-- 1
-10- 1
.names a b c d x z [22] y
1---0-- 1
--1---1 1
-11-0-- 1
--110-- 1
---100- 1
.names a b c d z
-0-1 1
-1-0 1
0-10 1
.names a d z [22]
110 1
.end
```

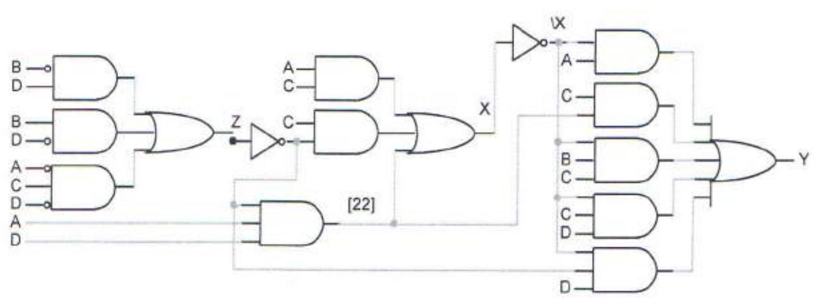
izhod iz programa misll

Z = B' D + B D' + A' C D'

[22] = A D Z'

X = [22] + A C + C Z'

Y = A X' + C [22] + B C X' + C D X' + D X' Z'



8 logičnih nivojev!

Večnivojska logika: CAD orodje misII

Krožni BCD števnik

```
.model bcd.increment
.inputs a b c d
.outputs w x y z
.names a b c d z w
1---1 1
0111- 1
.names a b c w z x
01-0- 1
0-100 1
.names a c z y
-11 1
000 1
.names a b c d z
0--0 1
-000 1
.end
```

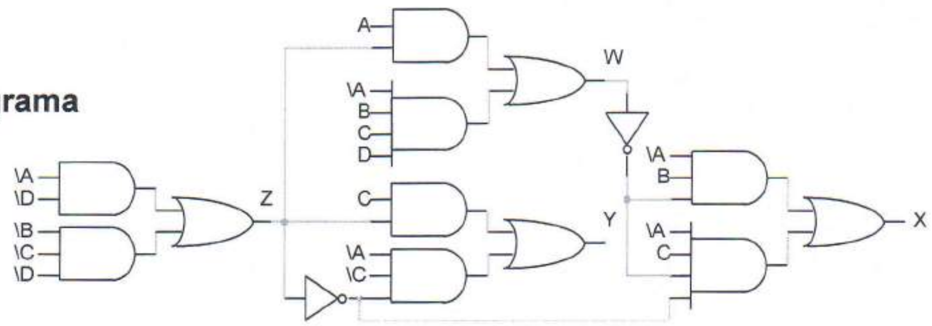
$$Z = A'D' + B'C'D'$$

$$Y = CZ + A'C'Z'$$

$$W = AZ + A'BCD$$

$$X = A'BW' + A'CW'Z'$$

izhod iz programa misII



7 logičnih nivojev!

Časovni odziv v kombinacijskih vezjih: Zakasnitev vrat

Kakšno je časovno obnašanje kombinacijskih vezij?

Časovni poteki signalov nam prikažejo, kaj se v vezju dogaja.

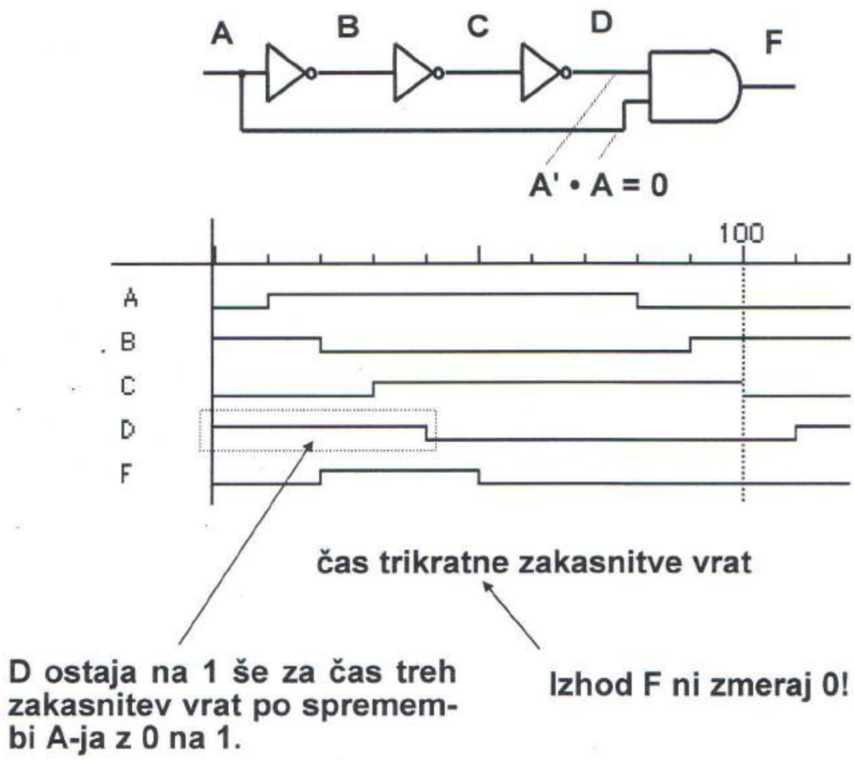
Časovne poteke signalov generiramo s simulacijo.

Pri obravnavi časovnih odzivov v kombinacijskih vezjih so pomembni naslednji parametri:

- ***Zakasnitev vrat***–**čas, ki preteče, da sprememba na vhodu povzroči spremembo na izhodu. Proizvajalci podajajo minimalno (najboljši primer), tipično (povprečje) in maksimalno (najslabši primer) zakasnitev vrat. Vezje moramo načrtovati zmeraj za najslabši primer!**
- ***Čas vzpona***–**čas, ki preteče za prehod izhodnega signala z nizkega na visok logični nivo.**
- ***Čas padca***–**čas, ki preteče za prehod izhodnega signala z visokega na nizek logični nivo.**

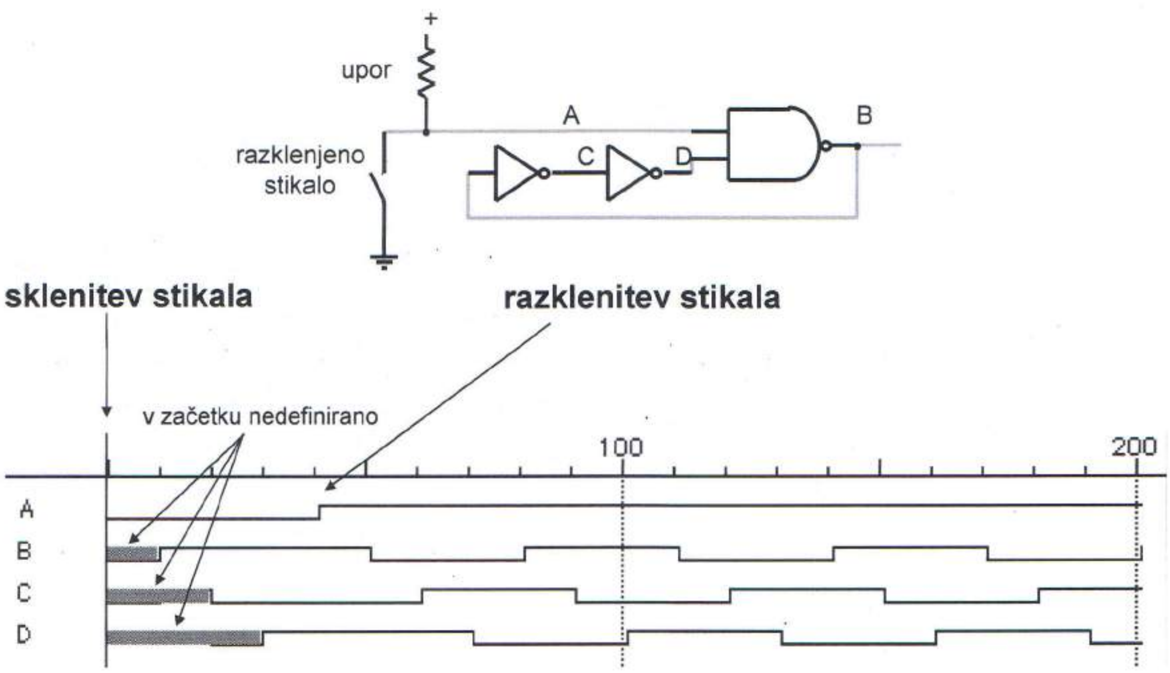
Časovni odziv v kombinacijskih vezjih: Zakasnitev vrat

Vezje za oblikovanje impulzov



Časovni odziv v kombinacijskih vezjih: Zakasnitev vrat

Drugo vezje za oblikovanje impulzov



Časovni odziv v kombinacijskih vezjih: Hazardi

Napetostna konica **(glitch) je nezaželen kratkotrajen impulz na izhodu kombinacijskega vezja.**

Vezje, v katerem obstaja možnost pojava napetostne konice, imenujemo vezje s *hazardom.*

Hazardi so lahko koristni (npr. v vezju za oblikovanje impulzov), navadno pa so nezaželeni (napetostne konice lahko povzročijo nepravilno delovanje vezja).

Napetostna konica se pojavi zaradi tega, ker se signali skozi vezje razširjajo po različnih poteh z različnimi zakasnitvami.

Nevarno je, če logika "opravi odločitev", ko je izhod nestabilen, ali če hazardni izhod krmili *asinhroni* vhod (ta reagira na spremembe takoj, ne da bi čakal na sinhronizacijski signal, imenovan *ura*).

Rešitve:

Počakamo, da se signali stabilizirajo (uporabimo uro).

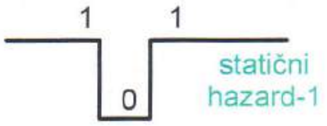
Izogibamo se vezij z asinhronimi vhodi.

Če že moramo načrtovati vezje, ki naj krmili asinhrone vhode, je treba načrtati vezje tako, da bo brez hazardov.

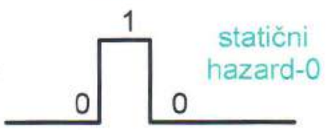
Časovni odziv v kombinacijskih vezjih: Hazardi

Vrste hazardov

Statični hazard **v vezju obstaja, če se na izhodu lahko pojavi napetostna konica, čeprav pričakujemo, da bo izhod nespremenjen.**

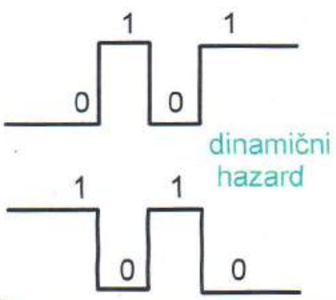


Vhodna sprememba povzroči, da gre izhod z 1 na 0 in nazaj na 1.



Vhodna sprememba povzroči, da gre izhod z 0 na 1 in nazaj na 0.

Dinamični hazard **v vezju obstaja, če se izhodni signal lahko spremeni več kot enkrat, kadar pričakujemo en sam prehod z 0 na 1 ali z 1 na 0.**



Vhodna sprememba povzroči dvakratno spremembo:

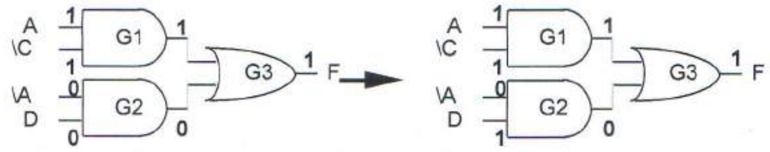
z 0 na 1 na 0 na 1

ali

z 1 na 0 na 1 na 0.

Časovni odziv v kombinacijskih vezjih: Hazardi

Primer napetostne konice

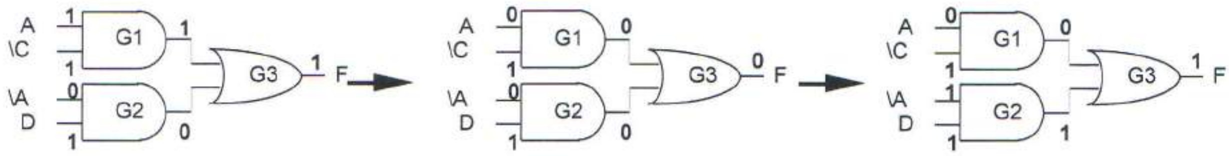


ABCD = 1100 → ABCD = 1101

CD \ AB	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	0	0
10	0	0	0	0

Spremeni se vhod znotraj glavnega vsebovalnika, zato hazarda ni.

F = A' D + A C'



ABCD = 1101 → ABCD = 0101 (A' je še 0) → ABCD = 0101 (A' je 1)

Spremeni se vhod med dvema glavnima vsebovalnikoma, zato se na izhodu F pojavi statični hazard-1.

Časovni odziv v kombinacijskih vezjih: Hazardi

Primer napetostne konice

Splošna strategija za odstranitev hazarda je ta, da Boolovemu izrazu dodamo redundantne glavne vsebovalnike, ki zagotavljajo, da so vse enobitne vhodne spremembe pokrite z enim takim vsebovalnikom.

Primer: **F = A' D + A C' postane F = A' D + A C' + C' D .**

S tem smo odstranili statični hazard-1. Kaj pa statični hazard-0?

Izrazimo F še v MKNO:

F = (A' + C') (A + D)

Obstaja statični hazard-0 pri vhodni spremembi ABCD = 0110 v 1110 ali 0010 v 1010.

Dodamo člen (C' + D).

Dobljeni izraz je ekvivalenten DNO funkcije F, ki odstrani statični hazard-1:

CD \ AB	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	0	0
10	0	0	0	0

F = (A' + C') (A + D) (C' + D) = (A C' + A' D + C' D) (C' + D)
= A C' + A' D + C' D

Časovni odziv v kombinacijskih vezjih: Hazardi

Primer napetostne konice

Analizo funkcije za statični hazard-0 lahko napravimo tudi drugače. Začnemo z izrazom, ki nima statičnih hazardov-1:

$$F = A\,C' + A'\,D + C'\,D$$

Izrazimo komplement:

$$F' = (A\,C' + A'\,D + C'\,D)'$$

$$= (A' + C)\,(A + D')\,(C + D')$$

$$= A\,C + A'\,C\,D' + C\,D' + A\,C\,D' + A'\,D'$$

$$= A\,C + C\,D' + A'\,D'$$

Dobljeni izraz pokriva vse sosedne celice 0 v K-diagramu.

Izraz za funkcijo F je tako brez statičnih hazardov-1 in brez statičnih hazardov-0.

Časovni odziv v kombinacijskih vezjih: Hazardi

Odkrivanje statičnih hazardov v večnivojskih vezjih

Večnivojsko funkcijo pretvorimo v dvonivojsko obliko, imenovano *izhodna funkcija prehodnega pojava.* Pri tem obravnavamo spremenljivke in njihove komplemente kot neodvisne spremenljivke, zato postulatov X + X' = 1 ali X • X' = 0 ne moremo uporabiti za poenostavljanje.

Primer:

$$F = A\,B\,C + (A + D)\,(A' + C')$$

$$F_1 = A\,B\,C + A\,A' + A\,C' + A'\,D + C'\,D$$ dvonivojska oblika

CD \ AB	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

Sprememba ABCD = 1111 v 1110 je pokrita s členom ABC, zato ni hazarda-1.

V vezju obstajajo trije statičnih hazardi - 1: pri vhodni spremembi ABCD = 1111 v 0111 ali 1111 v 1101 ali 1110 v 1100.

Časovni odziv v kombinacijskih vezjih: Hazardi

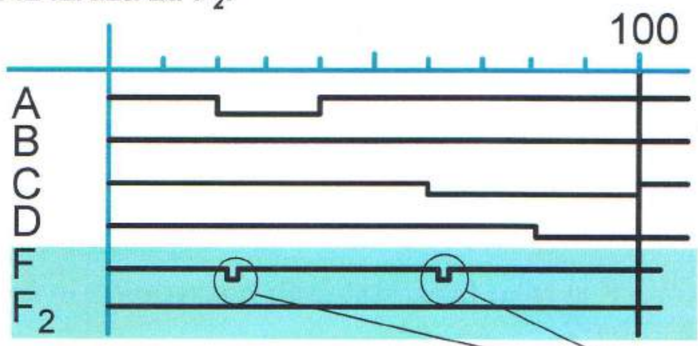
Statični hazardi-1

Rešitev:

Dodamo redundantne glavne vsebovalnike, da zagotovimo, da so vse sosedne celice 1 pokrite s skupnim vsebovalnikom.

$F_2 = A C' + A' D + C' D + \boxed{A B + B D}$

Ker člen AB popolnoma pokrije člen ABC, smo slednjega odstranili iz izraza za F_2.



Hazarda-1 v F sta v F_2 odpravljena.

Časovni odziv v kombinacijskih vezjih: Hazardi

Statični hazardi-0

Delamo podobno kot v prejšnjem primeru, vendar s komplementom funkcije F.

Če členi izhodne funkcije prehodnega pojava pokrivajo vse celice 0, potem v vezju ne obstaja noben hazard-0.

$F' = [A B C + (A + D)(A' + C')]'$

$= (A' + B' + C')(A' D' + A C)$

$= A' D' + A' B' D' + A' C' D' + A B' C$

$= A' D' + A B' C$

CD \ AB	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	1	1	1	0
10	0	0	1	0

Obstaja hazard-0 pri vhodni spremembi ABCD = 0010 v 1010, zato dodamo člen B'C D'.

$F' = A' D' + A B' C + B' C D'$

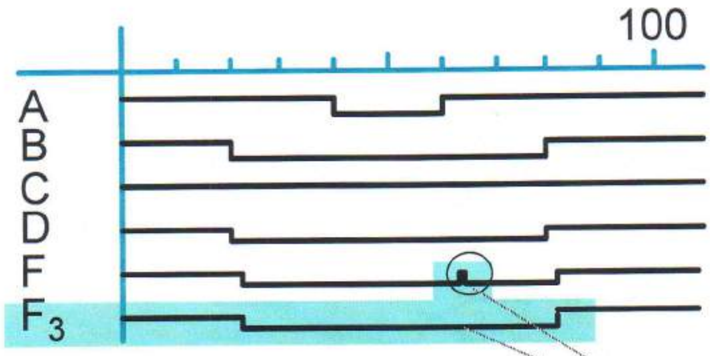
$F_3 = (A + D)(A' + B + C')(B + C' + D)$

Novi izraz za F ne vsebuje statičnih hazardov-0.

Izraza za F_3 in F_2 sta ekvivalentna. Oba izraza sta hkrati brez statičnih hazardov-0 in statičnih hazardov-1.

Časovni odziv v kombinacijskih vezjih: Hazardi

Statični hazardi-0

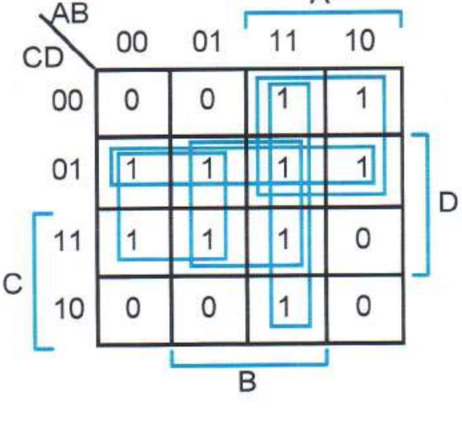


Hazard-0 v F je v F_3 odpravljen.

Časovni odziv v kombinacijskih vezjih: Hazardi

Načrtovanje vezij brez statičnih hazardov

Funkcijo pretvorimo v tako obliko, da njena izhodna funkcija prehodnega pojava zagotavlja, da so vse logično sosedne celice v K-diagramu pokrite s členom (ta pogoj odstrani hazarde-1) in da noben člen v izhodni funkciji prehodnega pojava ne vsebuje hkrati spremenljivke in njenega komplementa (ta pogoj odstrani hazarde-0).



$$F(A,B,C,D) = \sum(1,3,5,7,8,9,12,13,14,15)$$

$$F = A B + A' D + B D + A C' + C' D$$

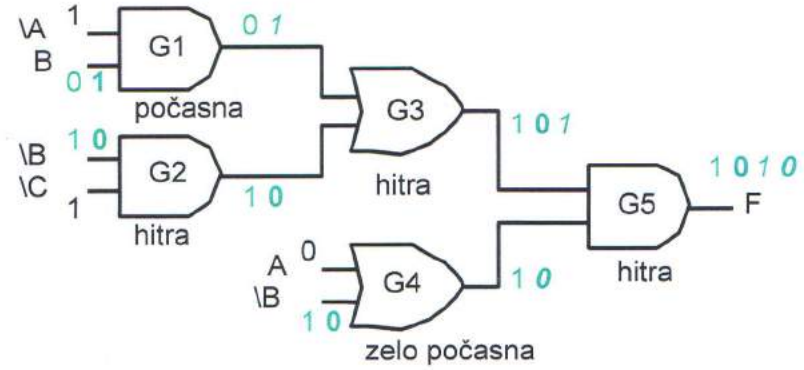
$$= (A' + B + C') D + A (B + C')$$

Zadnji izraz smo dobili s faktorizacijo po postulatu o distributivnosti, ki ne vnaša hazardov, ker njegova veljavnost ni odvisna od postulata P5.

Časovni odziv v kombinacijskih vezjih: Hazardi

Dinamični hazardi

Primer vezja z dinamičnim hazardom



Obstajajo tri različne poti od B ali B' na izhod.

Opazujemo prehod: ABC = 000, F = 1, v ABC = 010, F = 0.

Posamezne poti imajo različne zakasnitve. Vrata G1 so počasna, G4 zelo počasna, druga vrata so hitra. Na izhodu se pojavi dinamični hazard.

Odpravljanje dinamičnih hazardov je kompleksen problem.

Pregled poglavja

Predstavili smo:

- **Prehod od preprostih vrat do kompleksnejših gradnikov iz vrat**
- **Večnivojsko logiko: zmanjšano število vrat in vhodov vrat, vendar povečana zakasnitev večnivojske logike**
- **Uporabo programa misII za optimizacijo večnivojske logike in njeno preslikavo v izbrano tehnologijo**
- **Časovne odzive v kombinacijskih vezjih:**
  - **zakasnitev vrat, čas vzpona, čas padca**
  - **hazarde in načrtovanje vezij brez hazardov**

6. Strukturalna preklopna vezja

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

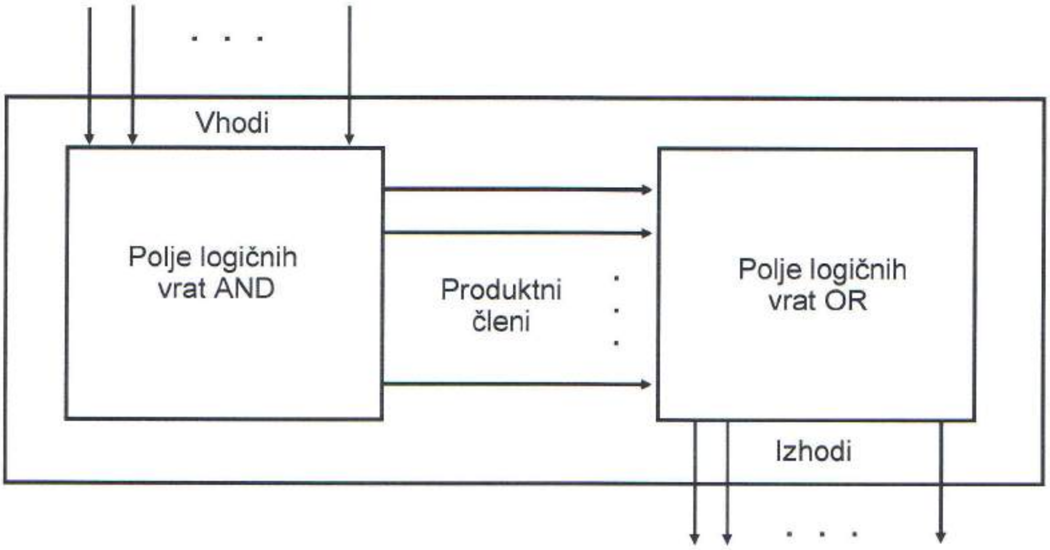
Vsebina poglavja

- ***Programabilna polja logičnih vrat - vezja PAL in PLA***
- ***Napetostno krmiljena stikala***
- ***Multipleksor***
- ***Demultipleksor***
- ***Dekodirnik***
- ***Tristanjski izhod in izhod z odprtim kolektorjem***
- ***Bralni pomnilnik***
- ***Primeri načrtovanja kombinacijske logike***
    - ***Krmiljenje industrijskega procesa***
    - ***Dekodirnik koda BCD za 7-segmentni LED prikazovalnik***
    - ***Logična funkcijska enota***
    - ***8-vhodni barrel pomikalnik***

Programabilna polja logičnih vrat - vezja PAL in PLA

Programabilna polja logičnih vrat so tovarniško že izdelani gradniki z velikim številom vrat AND in OR, ki jim lahko z vzpostavljanjem ali prekinjanjem povezav med vrati sprogramiramo želeno funkcijo. Takšni splošnonamenski logični gradniki so PAL-i (Programmable Array Logic) in PLA-ji (Programmable Logic Arrays).

Blokovni diagram programabilnega polja za obliko vsote produktov



Programabilna polja logičnih vrat - vezja PAL in PLA

Ključ do uspeha vezij PLA so skupni produktni členi v več funkcijah.

Primer: ***Enačbe***

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

Matrika produktnih členov

produktni členi	vhodi A	B	C	izhodi F_0	F_1	F_2	F_3
A B	1	1	-	0	1	1	0
$\overline{B}$ C	-	0	1	0	0	0	1
A $\overline{C}$	1	-	0	0	1	0	0
$\overline{B}$ $\overline{C}$	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

skupni produktni členi

Vhodna stran:

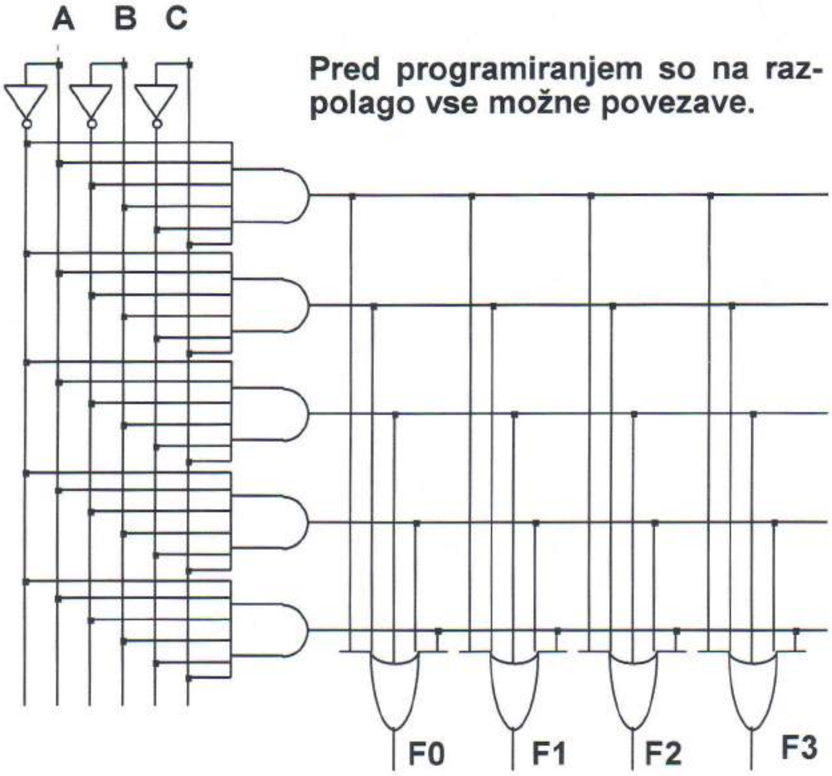
1 = spremenljivka v členu nenegirana
0 = spremenljivka v členu negirana
- = spremenljivka v členu ne nastopa

Izhodna stran:

1 = člen povezan na izhod
0 = ni povezave na izhod

Programabilna polja logičnih vrat - vezja PAL in PLA

Nadaljevanje primera

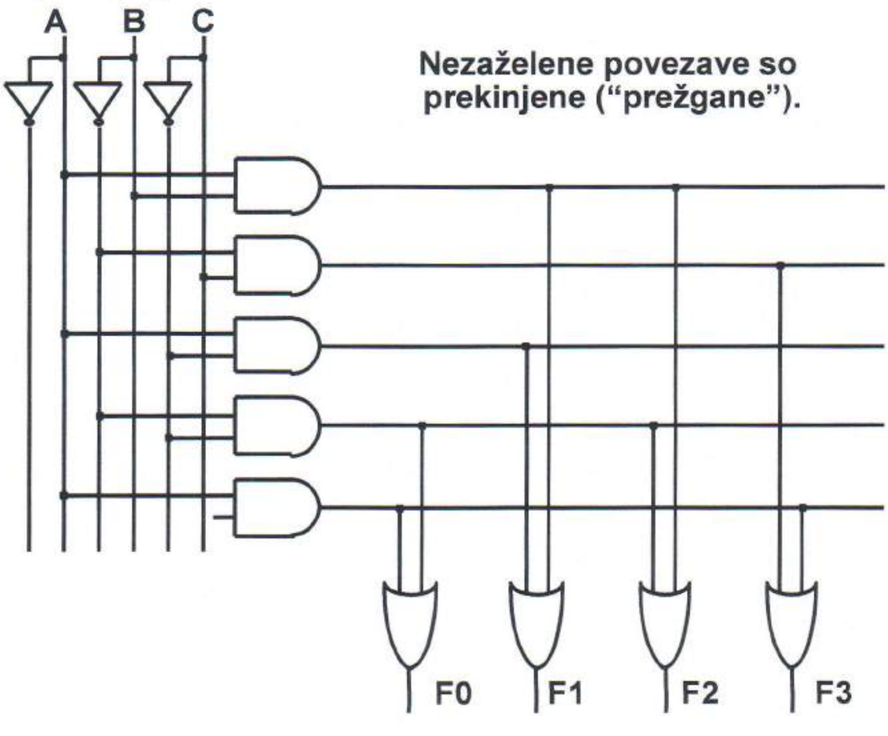


Pred programiranjem so na razpolago vse možne povezave.

Digitalna tehnika *Strukturalna preklopna vezja*

Programabilna polja logičnih vrat - vezja PAL in PLA

Nadaljevanje primera

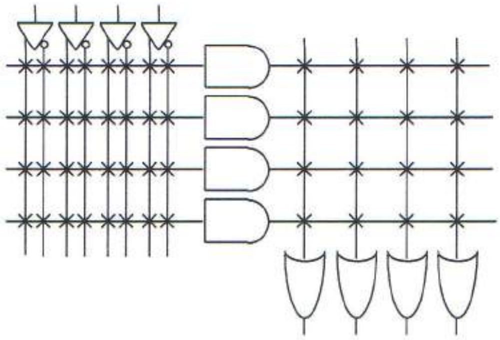


Nezaželene povezave so prekinjene ("prežgane").

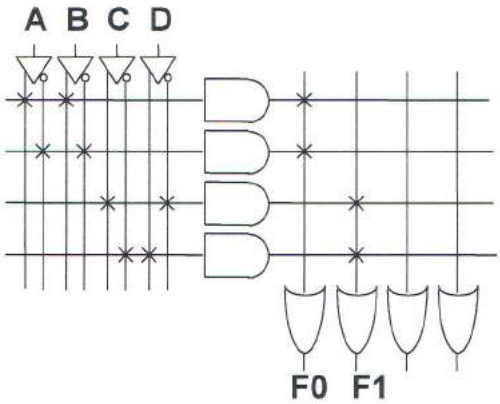
Opomba: Nekatera polja logičnih vrat delajo tako, da se povezave vzpostavljajo, ne pa prekinjajo.

Programabilna polja logičnih vrat - vezja PAL in PLA

Poenostavljena predstavitev logičnih vrat



S križcem označimo spoj med linijama v logičnem polju.



Shema za implementacijo funkcije
F0 = A B + A' B' in
F1 = C D' + C' D .

Programabilna polja logičnih vrat - vezja PAL in PLA

Primer implementacije funkcij z vezjem PLA

Več funkcij spremenljivk A, B in C:

$F1 = A\,B\,C$

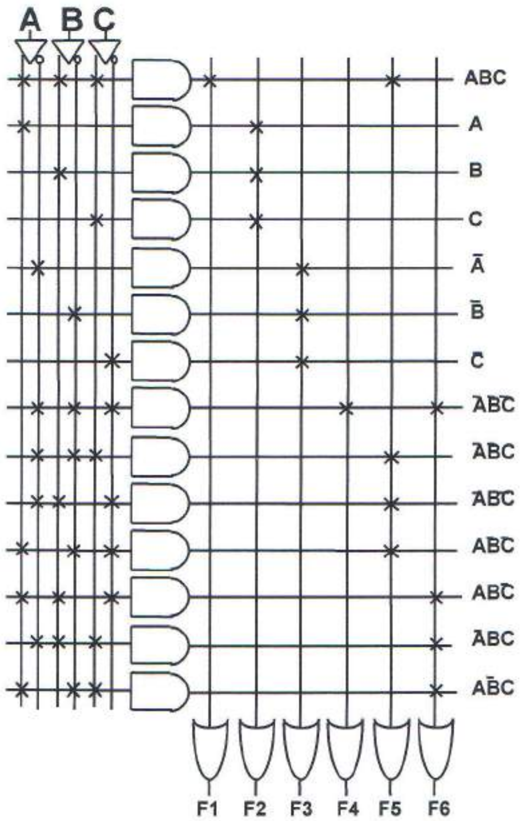
$F2 = A + B + C$

$F3 = \overline{A\,B\,C}$

$F4 = \overline{A + B + C}$

$F5 = A \oplus B \oplus C$

$F6 = \overline{A \oplus B \oplus C}$

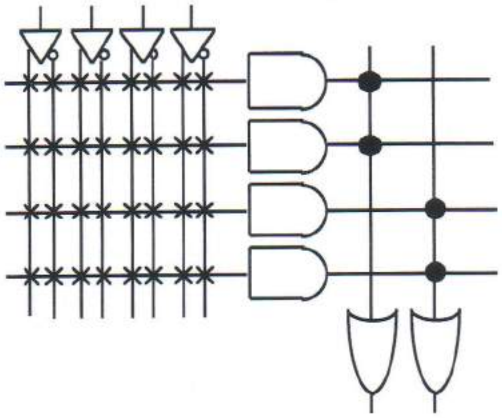


Programabilna polja logičnih vrat - vezja PAL in PLA

Razlika med vezjema PAL in PLA

Kakšna je razlika med vezjem PAL (Programmable Array Logic) in PLA (Programmable Logic Array)?

PAL vezje ima programabilno samo vhodno polje z vrati AND, izhodno polje z vrati OR pa ima že tovarniško določeno topologijo.



Dani stolpec v polju OR ima dostop samo do podmnožice mogočih produktnih členov.

PLA vezje ima programabilno vhodno polje z vrati AND in tudi programabilno izhodno polje z vrati OR.

Programabilna polja logičnih vrat - vezja PAL in PLA

Pretvornik iz koda BCD v Grayev kod

Pravilnostna tabela

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

K-diagrami

CD \ AB	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

K-diagram za W

CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	0	0	X	X
10	0	0	X	X

K-diagram za X

CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	1	X	X
10	1	1	X	X

K-diagram za Y

CD \ AB	00	01	11	10
00	0	0	X	1
01	1	0	X	0
11	0	1	X	X
10	1	0	X	X

K-diagram za Z

Minimizirane funkcije:

$W = A + B\,D + B\,C$

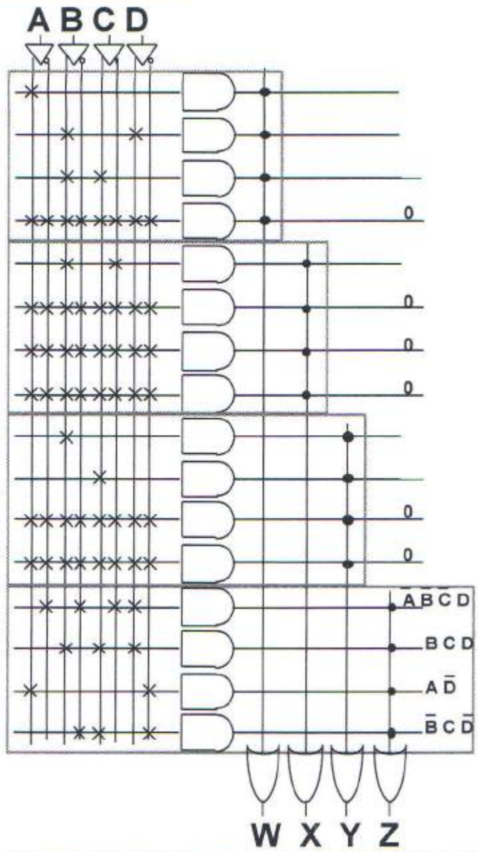
$X = B\,C'$

$Y = B + C$

$Z = A'B'C'D + B\,C\,D + A\,D' + B'\,C\,D'$

Programabilna polja logičnih vrat - vezja PAL in PLA

Pretvornik iz koda BCD v Grayev kod



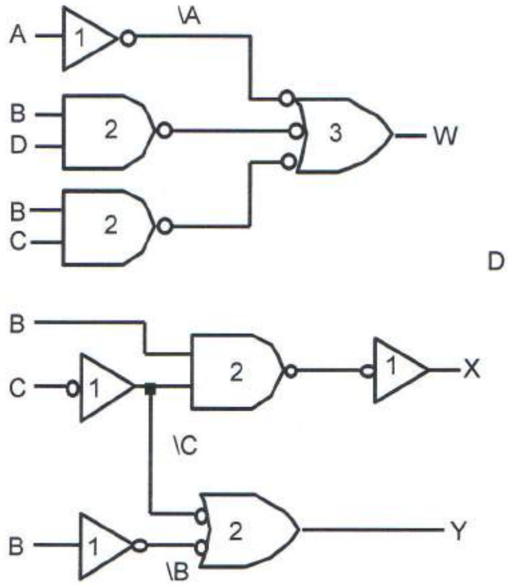
Sprogramiran PAL

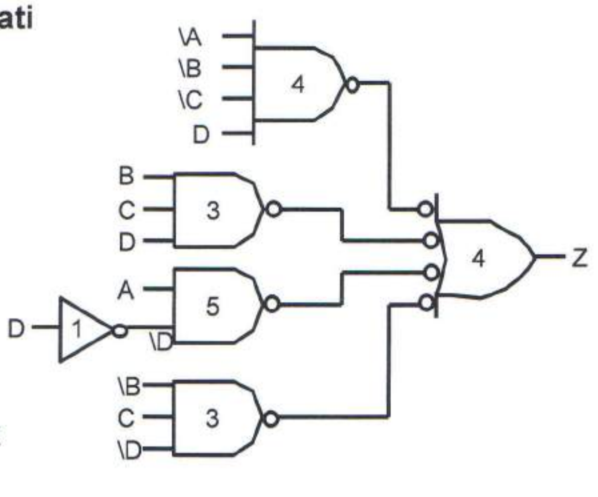
4 produktni členi na vsaka vrata OR

Programabilna polja logičnih vrat - vezja PAL in PLA

Pretvornik iz koda BCD v Grayev kod

Implementacija z diskretnimi vrati





1: 7404 — šest invertorjev
2,5: 7400 — štiri 2-vhodna vrata NAND
3: 7410 — tri 3-vhodna vrata NAND
4: 7420 — dvoje 4-vhodnih vrat NAND

5 čipov SSI proti 1 čipu PAL!

Programabilna polja logičnih vrat - vezja PAL in PLA

Primerjalnik dveh 2-bitnih števil

CD \ AB	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

K-diagram za EQ

CD \ AB	00	01	11	10
00	0	1	1	1
01	1	0	1	1
11	1	1	0	1
10	1	1	1	0

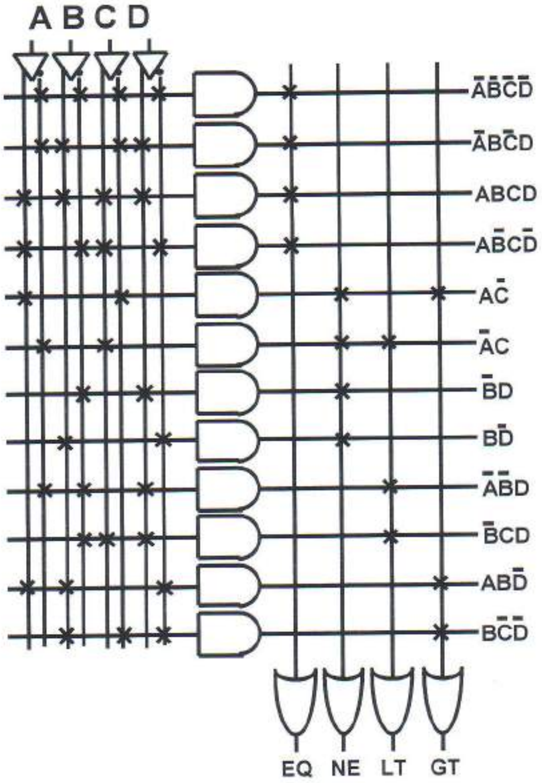
K-diagram za NE

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

K-diagram za LT

CD \ AB	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

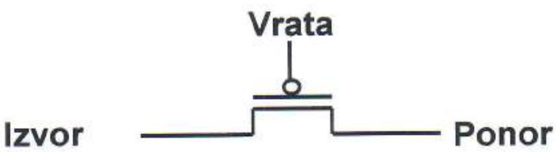
K-diagram za GT



Napetostno krmiljena stikala

NMOS tranzistor

Logična "1" na vhodu Vrata poveže priključka Izvor in Ponor.



PMOS tranzistor

Logična "0" na vhodu Vrata poveže priključka Izvor in Ponor.

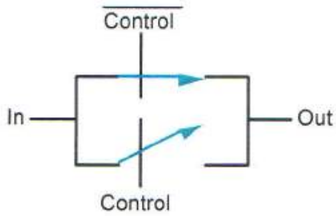
Napetostno krmiljena stikala

CMOS prenosna vrata

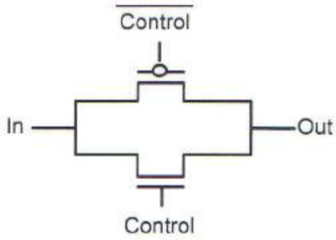
NMOS tranzistorji dobro prenašajo vrednost "0", slabo pa "1".

PMOS tranzistorji dobro prenašajo vrednost "1", slabo pa "0".

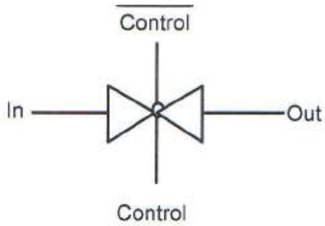
"Prenosna" vrata sestavljajo vzporedno vezana tranzistorja NMOS in PMOS:



Stikala



Tranzistorji

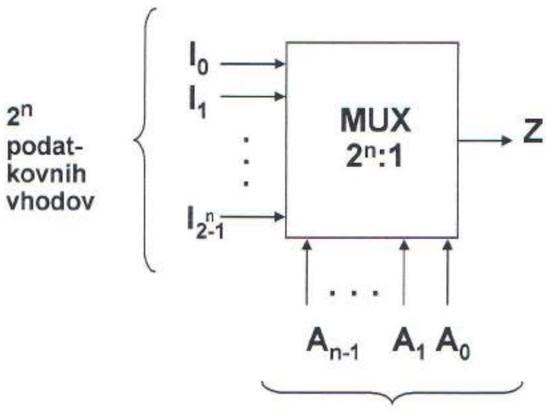


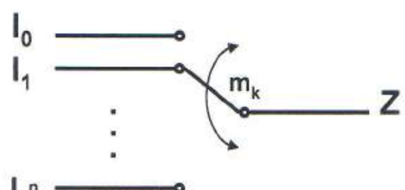
Prenosna ali "metuljčna" vrata

Multipleksor

Multipleksor **ali** ***selektor*** **je strukturalno preklopno vezje z 2^n podatkovnimi vhodi, n naslovnimi vhodi in enim podatkovnim izhodom.**

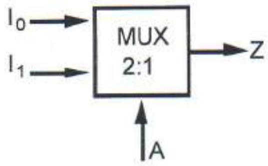
V vsakem trenutku povezuje enega izmed 2^n podatkovnih vhodov na izhod. Vrednosti na naslovnih vhodih tvorijo binarni indeks vhoda, ki se poveže na izhod.



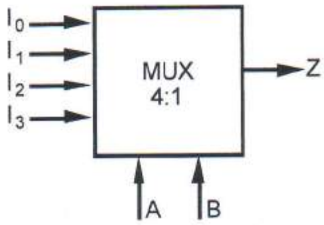


$$Z = \sum_{k=0}^{2^n-1} m_k I_k \quad , \qquad m_k = \dot{A}_0 \dot{A}_1 \dots \dot{A}_{n-1}$$

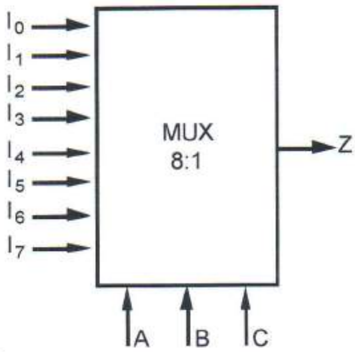
Multipleksor



$$Z = A'\,I0 + A\,I1$$



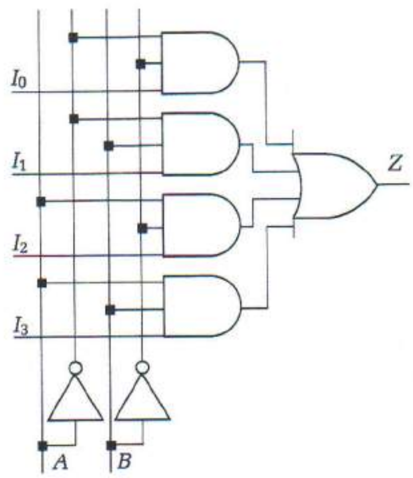
$$Z = A'\,B'\,I0 + A'\,B\,I1 + A\,B'\,I2 + A\,B\,I3$$

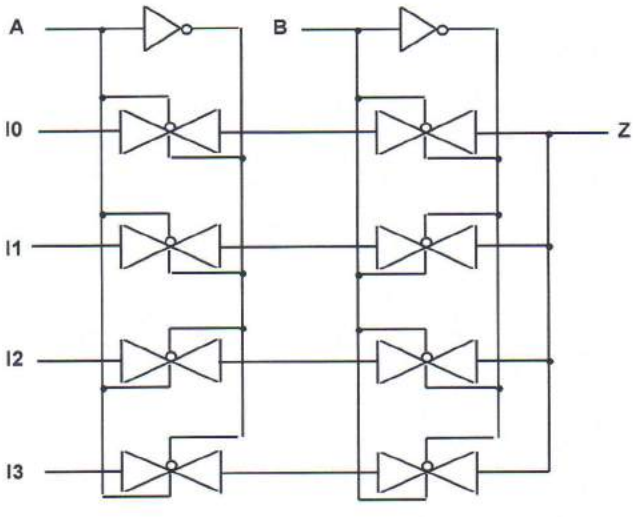


$$Z = A'\,B'\,C'\,I0 + A'\,B'\,C\,I1 + A'\,B\,C'\,I2 + A'\,B\,C\,I3 + A\,B'\,C'\,I4 + A\,B'\,C\,I5 + A\,B\,C'\,I6 + A\,B\,C\,I7$$

Multipleksor

Alternativni implementaciji multipleksorja 4:1





Implementacija multipleksorja 4:1 z logičnimi vrati

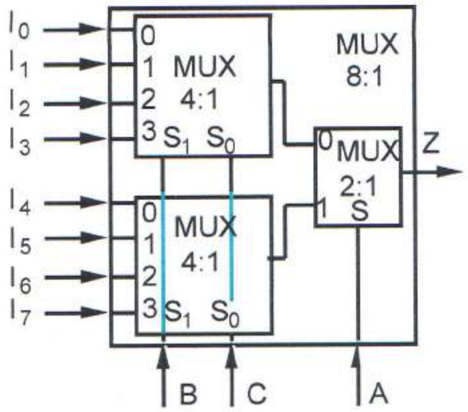
36 tranzistorjev

Implementacija multipleksorja 4:1 s prenosnimi vrati

20 tranzistorjev

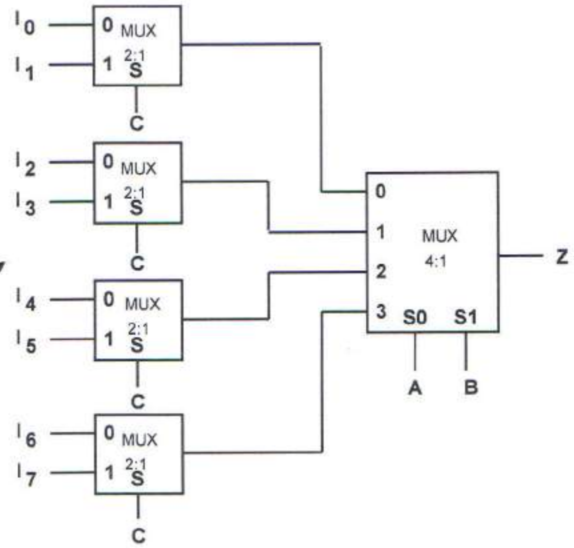
Multipleksor

Velike multipleksorje lahko implementiramo s kaskadno vezavo manjših multipleksorjev.



Naslovna signala B in C hkrati izbereta enega izmed I0-I3 in I4-I7.

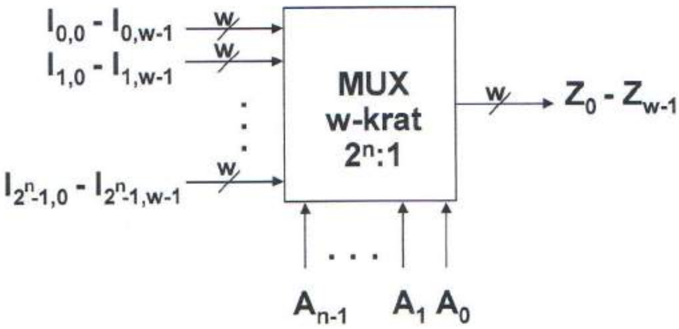
Naslovni signal A izbere, ali poveže na izhod Z izhod zgornjega ali spodnjega multipleksorja 4:1.

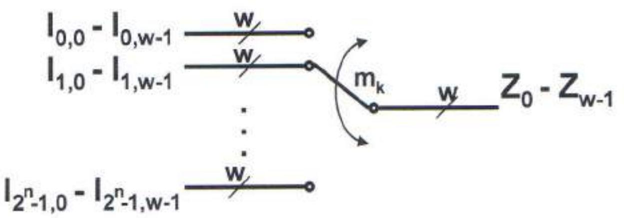


Alternativna implementacija multipleksorja 8:1

Multipleksor

Podatkovne poti od vhodov na izhod multipleksorja so lahko tudi večbitne (npr. s širino besede w). Takšnim multipleksorjem pravimo *vektorski multipleksorji*.





$$Z_j = \sum_{k=0}^{2^n-1} m_k I_{k,j}\ , \quad 0 \le j \le w-1\ , \quad m_k = \dot{A}_0\,\dot{A}_1 \ldots \dot{A}_{n-1}$$

Multipleksor

Realizacija preklopnih funkcij z multipleksorjem

Imejmo preklopno funkcijo $f(x_1, x_2, ..., x_n)$. Množico neodvisnih spremenljivk $X=\{x_1, x_2, ..., x_n\}$ razdelimo na množico *naslovnih spremenljivk* X_a in množico *podatkovnih spremenljivk* X_d tako, da velja:

$X_a=\{x_{a1}, x_{a2}, ..., x_{as}\} \subseteq X,$

$X_d=\{x_{d1}, x_{d2}, ..., x_{d(n-s)}\} \subset X,$

$X_a \cup X_d = \{x_1, x_2, ..., x_n\}$ in

$X_a \cap X_d = \varnothing .$

Funkcijo $f(x_1, x_2, ..., x_n)$ po Shannonovem izreku razširimo po vseh naslovnih spremenljivkah. Dobimo naslednjo DNO funkcije f:

$$f(x_1, x_2, ..., x_n) = \sum_{i=0}^{2^s-1} \dot{x}_{a1} \dot{x}_{a2} ... \dot{x}_{as} f(w_{i1}, w_{i2}, ..., w_{is}, x_{d1}, x_{d2}, ..., x_{d(n-s)}).$$

V zadnjem izrazu smo privzeli $x_{a1} = x_1, x_{a2} = x_2, ..., x_{as} = x_s$, kar naj ne zmanjšuje splošnosti.

Multipleksor

Realizacija preklopnih funkcij z multipleksorjem

V dobljeni DNO funkciji f so konjunktivni členi sestavljeni iz literalov naslovnih spremenljivk in funkcijskih ostankov

$$f(w_{i1}, w_{i2}, ..., w_{is}, x_{d1}, x_{d2}, ..., x_{d(n-s)}),$$

ki so odvisni le od podatkovnih spremenljivk.

Po primerjavi tega zapisa z multipleksorsko enačbo ugotovimo, da če na naslovne vhode multipleksorja priključimo naslovne spremenljivke, na podatkovne vhode multipleksorja pa omenjene funkcijske ostanke, zadošča izhod Z multipleksorja dani preklopni funkciji $f(x_1, x_2, ..., x_n)$.

Vsak funkcijski ostanek na vhodu multipleksorja si lahko predstavljamo kot originalno funkcijo, ki jo lahko zopet realiziramo z multipleksorjem. Ker lahko po Shannonovem izreku razširimo poljubno preklopno funkcijo, jo lahko tudi realiziramo le z multipleksorji (multipleksorjem).

Najpogosteje preklopno funkcijo z n spremenljivkami realiziramo tako, da n-1 spremenljivk povežemo na naslovne vhode multipleksorja, eno pa uporabimo za realizacijo funkcijskih ostankov na podatkovnih vhodih.

Multipleksor

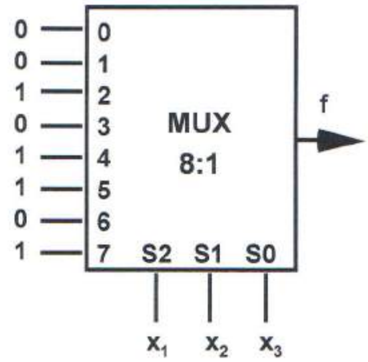
Realizacija preklopnih funkcij z multipleksorjem

Primer: Preklopno funkcijo $f(x_1, x_2, x_3) = \Sigma(2,4,5,7)$ realiziramo z multipleksorjem (multipleksorji).

1. $X_a = \{x_1, x_2, x_3\}$, $X_d = \varnothing$

$$\begin{aligned} f(x_1, x_2, x_3) = {} & x'_1x'_2x'_3f(0,0,0) + x'_1x'_2x_3f(0,0,1) + x'_1x_2x'_3f(0,1,0) + \\ & x'_1x_2x_3f(0,1,1) + x_1x'_2x'_3f(1,0,0) + x_1x'_2x_3f(1,0,1) + \\ & x_1x_2x'_3f(1,1,0) + x_1x_2x_3f(1,1,1) = \\ = {} & x'_1x'_2x'_3 \bullet 0 + x'_1x'_2x_3 \bullet 0 + x'_1x_2x'_3 \bullet 1 + x'_1x_2x_3 \bullet 0 + \\ & x_1x'_2x'_3 \bullet 1 + x_1x'_2x_3 \bullet 1 + x_1x_2x'_3 \bullet 0 + x_1x_2x_3 \bullet 1 \end{aligned}$$

i	x_1	x_2	x_3	$f = I_i$
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



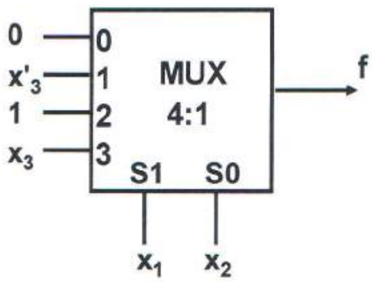
Multipleksor

Realizacija preklopnih funkcij z multipleksorjem

2. $X_a = \{x_1, x_2\}$, $X_d = \{x_3\}$

$$\begin{aligned} f(x_1, x_2, x_3) & = x'_1x'_2f(0,0,x_3) + x'_1x_2f(0,1,x_3) + x_1x'_2f(1,0,x_3) + \\ & \quad x_1x_2f(1,1,x_3) = \\ & = x'_1x'_2 \bullet 0 + x'_1x_2 \bullet x'_3 + x_1x'_2 \bullet 1 + x_1x_2 \bullet x_3 \end{aligned}$$

i	x_1	x_2	x_3	f	I_i
0	0	0	0	0	0
	0	0	1	0	
1	0	1	0	1	x'_3
	0	1	1	0	
2	1	0	0	1	1
	1	0	1	1	
3	1	1	0	0	x_3
	1	1	1	1	



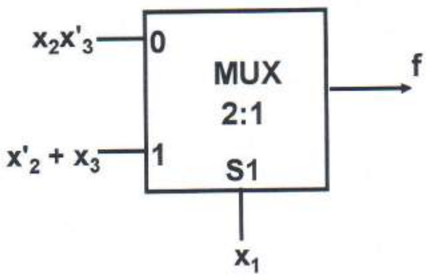
Multipleksor

Realizacija preklopnih funkcij z multipleksorjem

3. $X_a = \{x_1\}$, $X_d = \{x_2, x_3\}$

$$f(x_1, x_2, x_3) = x'_1 f(0, x_2, x_3) + x_1 f(1, x_2, x_3) =$$
$$= x'_1 \bullet x_2 x'_3 + x_1 \bullet (x'_2 + x_3)$$

i	x_1	x_2	x_3	f	I_i
0	0	0	0	0	$x_2x'_3$
	0	0	1	0	
	0	1	0	1	
	0	1	1	0	
1	1	0	0	1	$x'_2 + x_3$
	1	0	1	1	
	1	1	0	0	
	1	1	1	1	



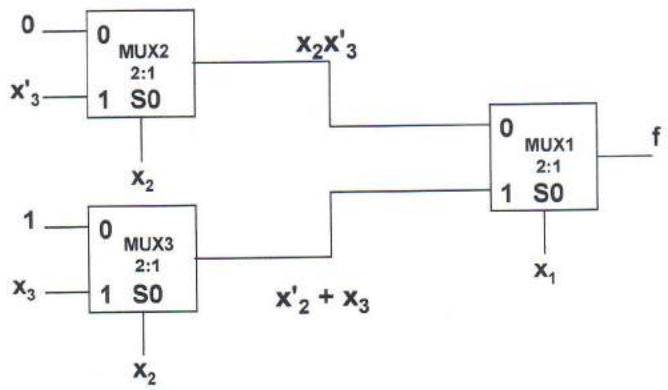
Funkcijska ostanka $x_2x'_3$ in $x'_2 + x_3$ realiziramo z multipleksorjema MUX2 in MUX3. Za naslovno spremenljivko izberemo x_2.

Multipleksor

Realizacija preklopnih funkcij z multipleksorjem

i	x_2	x_3	$x_2x'_3$	I_i	$x'_2 + x_3$	I_i
0	0	0	0	0	1	1
	0	1	0		1	
1	1	0	1	x'_3	0	x_3
	1	1	0		1	

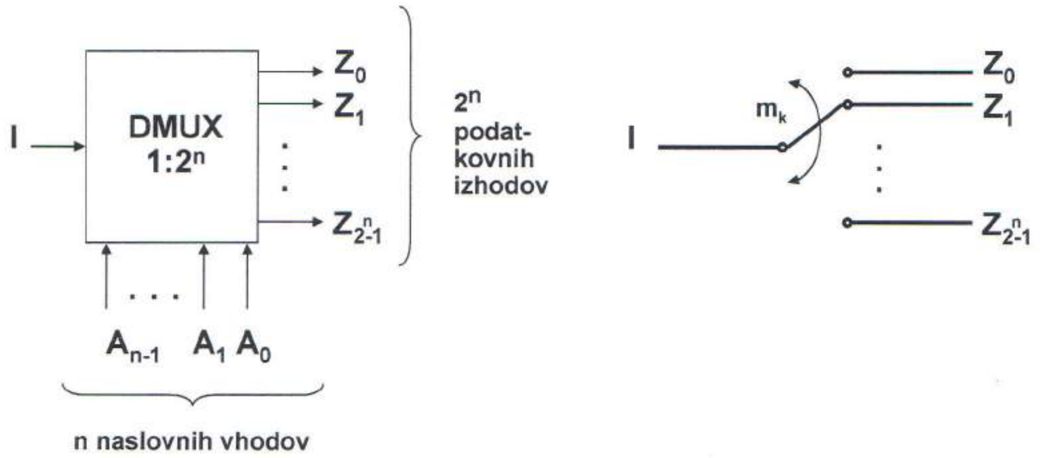
$$x_2x'_3 = x'_2 \bullet 0 + x_2 \bullet x'_3$$
$$x'_2 + x_3 = x'_2 \bullet 1 + x_2 \bullet x_3$$



To vezje je primer *univerzalnega multipleksorskega drevesa*, ki je sestavljeno samo iz multipleksorjev 2:1. Poljubno funkcijo n spremenljivk lahko realiziramo z N = n-1 nivoji.

Demultipleksor

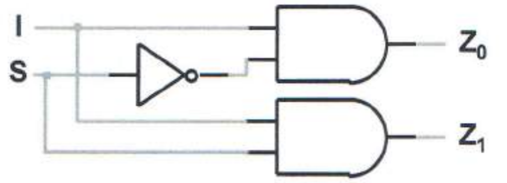
Demultipleksor **je strukturalno preklopno vezje, ki izvaja obratno nalogo kot multipleksor. Ima en podatkovni vhod, n naslovnih vhodov in 2^n podatkovnih izhodov. Demultipleksor v vsakem trenutku povezuje podatkovni vhod na tisti podatkovni izhod, ki ga izberemo z naslovnimi vhodi.**

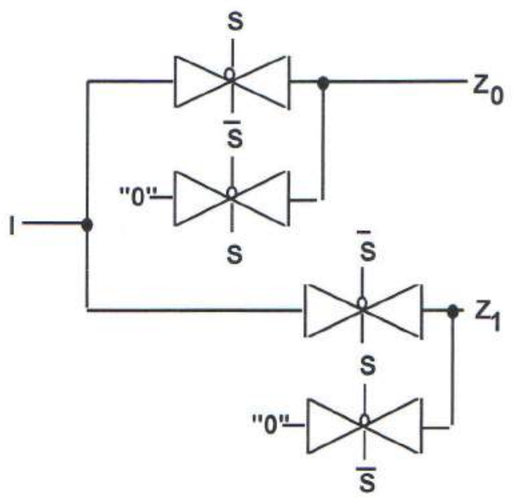


$$Z_k = m_k I \ , \quad 0 \le k \le 2^n-1 \ , \quad m_k = \dot{A}_0 \dot{A}_1 \ldots \dot{A}_{n-1}$$

Demultipleksor

Alternativni implementaciji demultipleksorja 1:2





Implementacija demultipleksorja 1:2 z logičnimi vrati

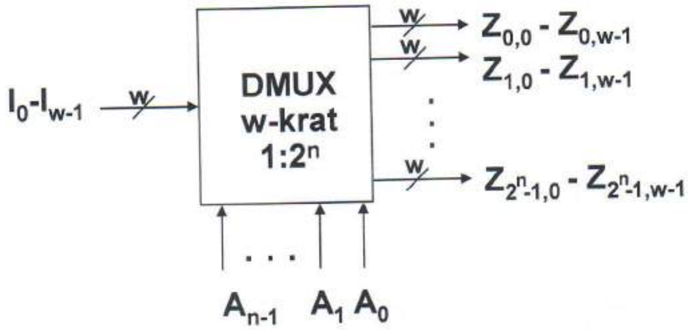
10 tranzistorjev

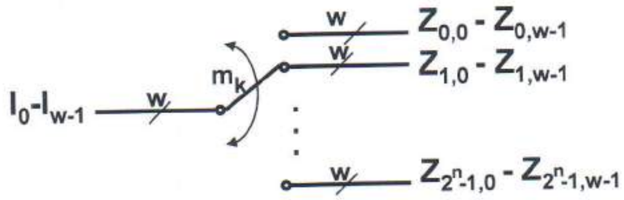
Implementacija demultipleksorja 1:2 s prenosnimi vrati

8 tranzistorjev

Demultipleksor

Podatkovne poti od vhoda na izhode multipleksorja so lahko tudi večbitne (npr. širina besede w). Takšnim demultipleksorjem pravimo ***vektorski demultipleksorji.***

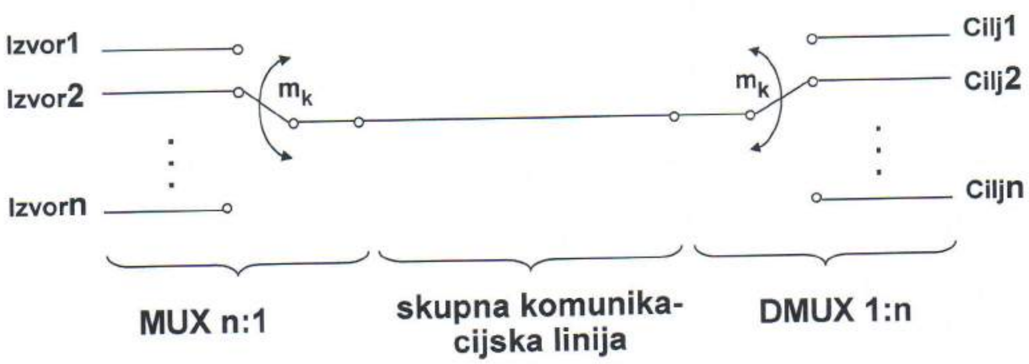




$Z_{k,j} = m_k I_j \quad , 0 \le k \le 2^n-1 \quad , 0 \le j \le w-1, \quad m_k = \dot{A}_0 \dot{A}_1 \ldots \dot{A}_{n-1}$

Demultipleksor

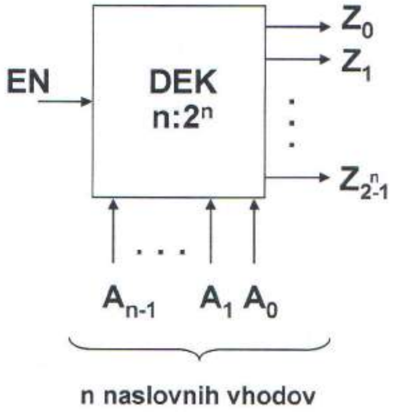
Kombinacijo multipleksor-demultipleksor pogosto uporabljamo za časovno multipleksiran prenos več signalov po skupni komunikacijski liniji (linijah).



Dekodirnik

Dekodirnik je strukturalno preklopno vezje, identično demultipleksorju. Ima n naslovnih vhodov in 2^n podatkovnih izhodov, razlika je le pri pomenu podatkovnega vhoda, ki ima pri dekodirniku vlogo omogočitvenega vhoda.

Če je dekodirnik omogočen, generira na izhodih mintermske funkcije. Zato mu pravimo tudi *generator mintermov*.



EN	A_{n-1}	A_{n-2}	. . .	A_1	A_0	Z_0	Z_1	Z_2	Z_3	. . .	Z_{2^n-1}
0	x	x		x	x	0	0	0	0		0
1	0	0		0	0	1	0	0	0		0
1	0	0		0	1	0	1	0	0		0
1	0	0		1	0	0	0	1	0		0
1	0	0		1	1	0	0	0	1		0
		. . .						. . .			
1	1	1		1	1	0	0	0	0		1

$$Z_k = m_k EN \,,\; 0 \le k \le 2^n-1 \,,\quad m_k = \dot{A}_0 \dot{A}_1 \ldots \dot{A}_{n-1}$$

Če so izhodi dekodirnika aktivni nizki (označeni s krožcem), dobimo na izhodih makstermske funkcije.

Dekodirnik

Realizacija preklopnih funkcij z dekodirnikom

Z dekodirnikom n:2^n lahko realiziramo poljubno preklopno funkcijo z n spremenljivkami, saj imamo na izhodu dekodirnika na voljo vse minterme.

Za vsako funkcijo, ki jo želimo realizirati, moramo dodati samo ena vrata OR, na katera povežemo tiste minterme iz dekodirnika, ki v funkciji nastopajo.

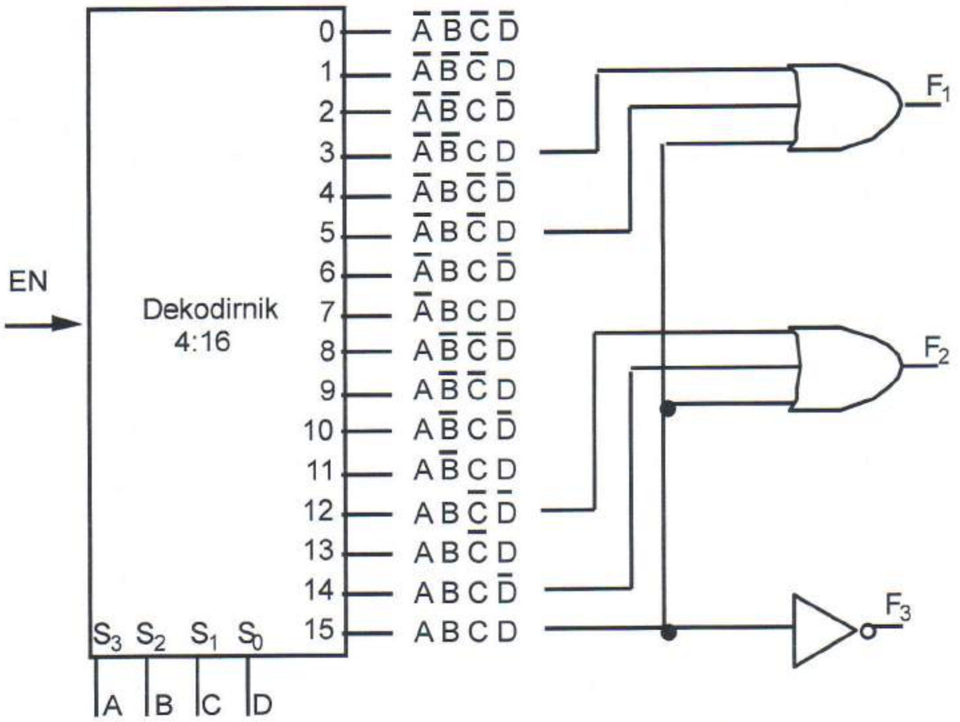
Realizacija preklopne funkcije z dekodirnikom je primerna, če moramo realizirati veliko število funkcij.

Primer: Z dekodirnikom in vrati OR realizirajmo funkcije:

F1 = A' B C' D + A' B' C D + A B C D
F2 = A B C' D' + A B C
F3 = (A' + B' + C' + D')

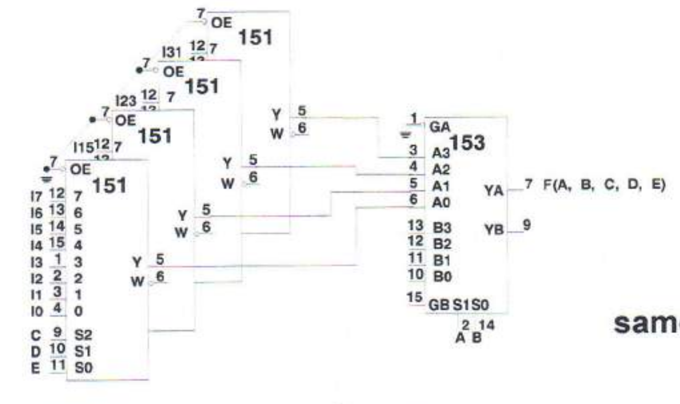
Dekodirnik

Realizacija preklopnih funkcij z dekodirnikom

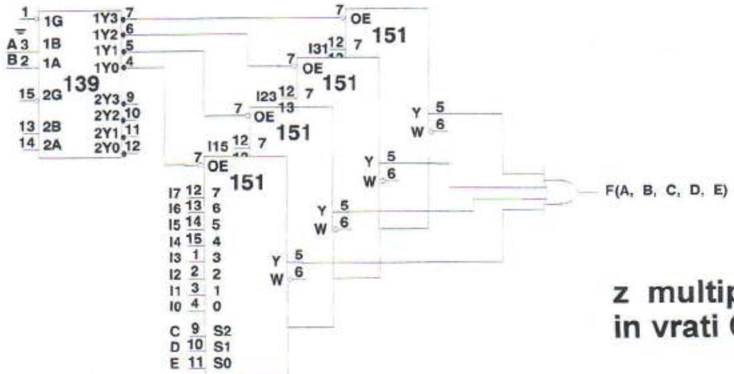


Dekodirnik

Alternativni implementaciji multipleksorja 32:1



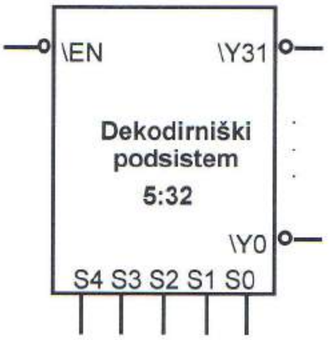
samo z multipleksorji

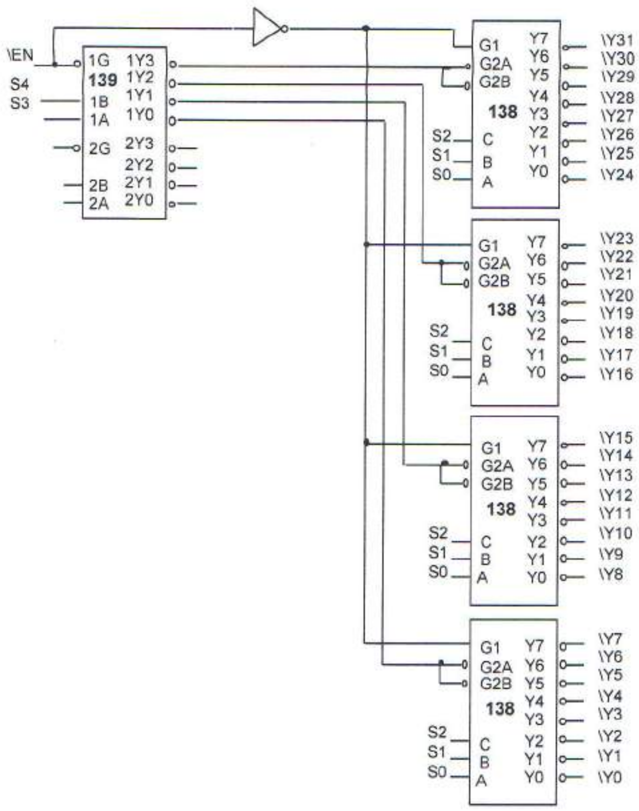


z multipleksorji, dekodirnikom in vrati OR

Dekodirnik

Realizacija dekodirnika 5:32





Tristanjski izhod in izhod z odprtim kolektorjem

Tretje stanje

Logični stanji: "0", "1".

Nepomembno, nepoznano stanje: "X"(v realnem vezju mora biti neka vrednost!)

Tretje stanje: "Z"— visoka impedanca (neskončna upornost, ni povezave).

Vrata s tristanjskim izhodom: izhodne vrednosti so "0", "1" in "Z".

Dodatni vhod je vhod OE za omogočitev izhoda.

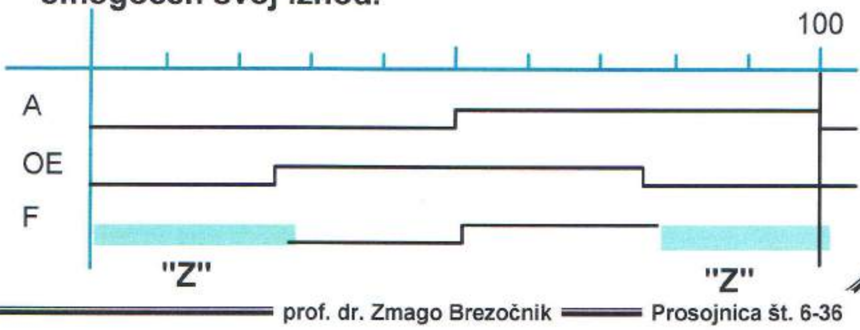
A	OE	F
X	0	Z
0	1	0
1	1	1

Če je OE visok, so ta vrata neinvertirajoči vmesnik.

Če je OE nizek, je tako, kot da bi bila povezava vrat z izhodom prekinjena.

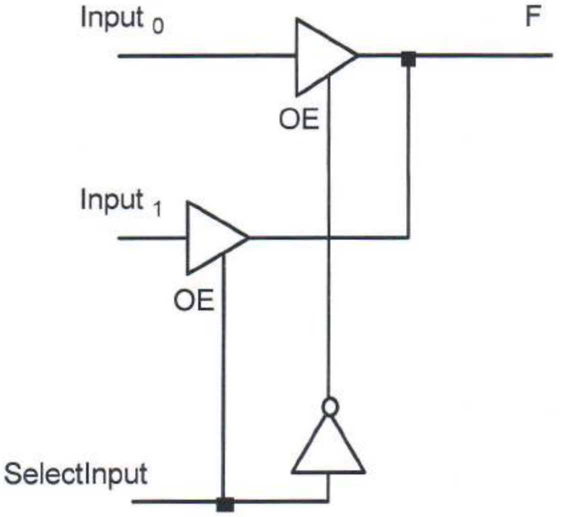
To omogoča, da lahko na isto izhodno linijo povežemo več vrat, če imajo *istočasno* le ena vrata omogočen svoj izhod.

Časovni potek za neinvertirajoči vmesnik



Tristanjski izhod in izhod z odprtim kolektorjem

Uporaba vrat s tristanjskim izhodom za implementacijo multipleksorja



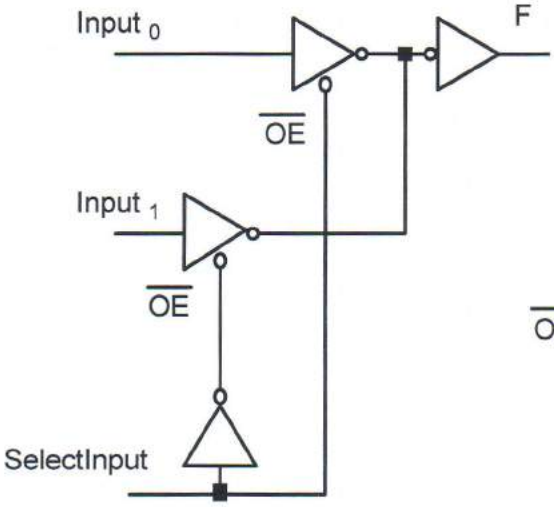
Če je SelectInput visok, je Input1 povezan na F.

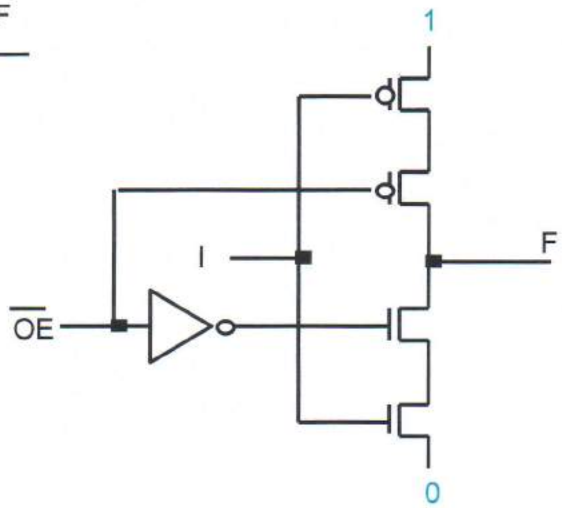
Če je SelectInput nizek, je Input0 povezan na F.

To je dejansko multipleksor 2:1.

Tristanjski izhod in izhod z odprtim kolektorjem

Alternativna organizacija tristanjskega izhoda



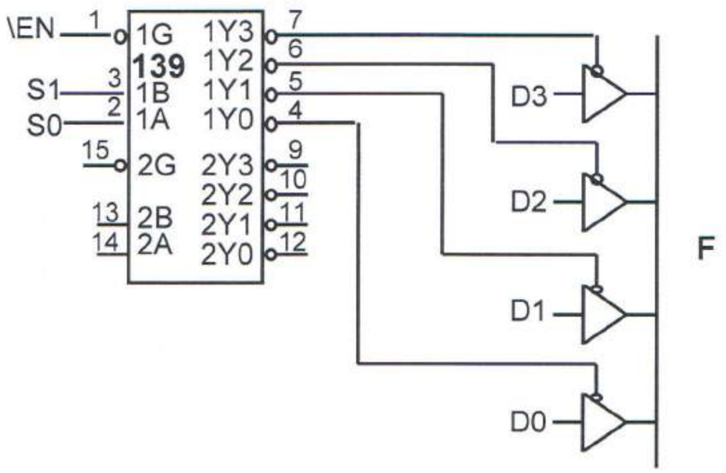


Aktivni nizki omogočitveni signal in invertirajoči tristanjski vmesniki.

Implementacija vrat s tristanjskim izhodom na nivoju tranzistorjev.

Tristanjski izhod in izhod z odprtim kolektorjem

Implementacija multipleksorja 4:1



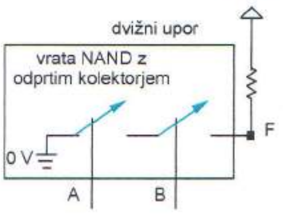
Dekodirnik + 4 vrata s tristanjskim izhodom

Tristanjski izhod in izhod z odprtim kolektorjem

Odprti kolektor je drugi način za povezavo več vrat na isto izhodno linijo.

Vrata lahko postavijo izhod z odprtim kolektorjem na nizek nivo, ne morejo pa ga aktivno povleči na visok nivo.

Na izhod vrat od zunaj priključimo dvižni upor na napetost logične "1".

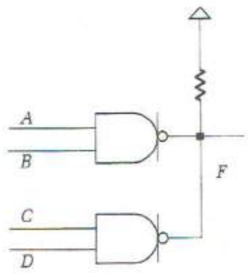


Vrata NAND z odprtim kolektorjem

Ožičeni AND:

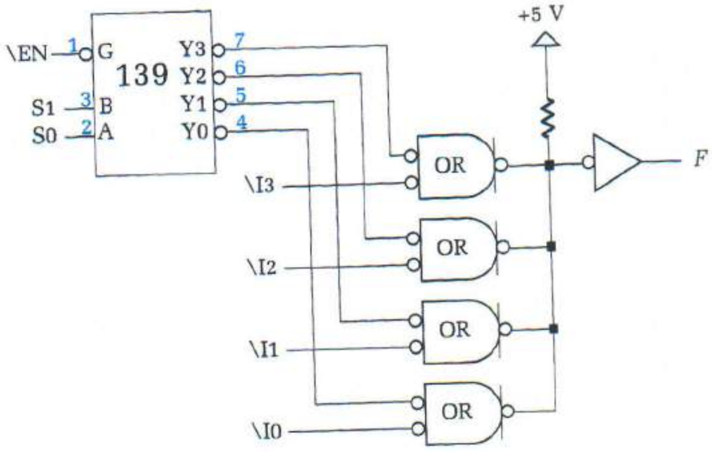
Če sta A in B "1", je izhod aktivno povlečen na "0", če sta C in D "1", je izhod aktivno povlečen na "0", če ena vrata vlečejo na "0" druga pa na "1", prevlada "0", če sta izhoda obeh vrat "1", izhod plava in ga dvižni upor povleče na "1".

Vidimo torej, da sta funkciji NAND z ožičenjem povezani konjunktivno.



Tristanjski izhod in izhod z odprtim kolektorjem

Implementacija multipleksorja4:1



Dekodirnik + 4 vrata z odprtimi kolektorji

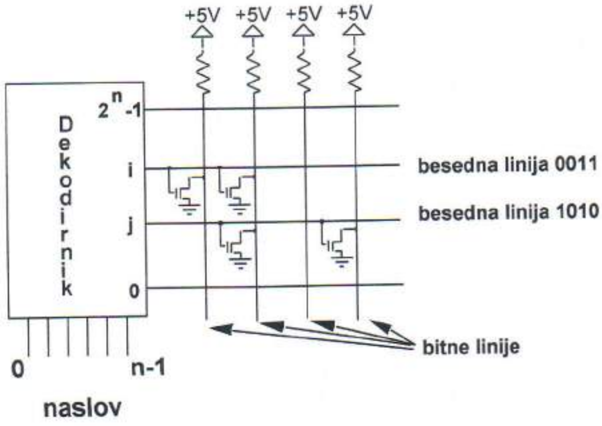
Bralni pomnilnik

Bralni pomnilnik (Read Only Memory-ROM) je dvodimenzionalno polje z vrednostmi "0" in "1".

Vrstica se imenuje "beseda", indeks vrstice se imenuje "naslov".

Širina vrstice se imenuje *bitna širina* ali *velikost besede*.

Naslov je vhod, izbrana beseda je izhod.



Interna organizacija

Bralni pomnilnik

Primer: Implementacija kombinacijske logike

F0 = A' B' C + A B' C' + A B' C

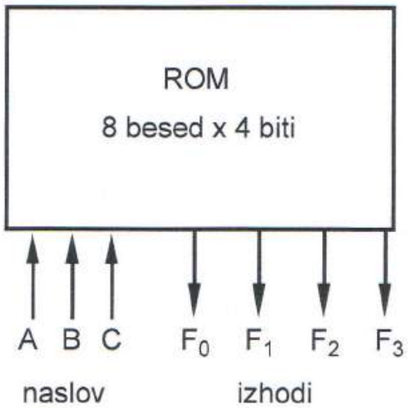
F1 = A' B' C + A' B C' + A B C

F2 = A' B' C' + A' B' C + A B' C'

F3 = A' B C + A B' C' + A B C'

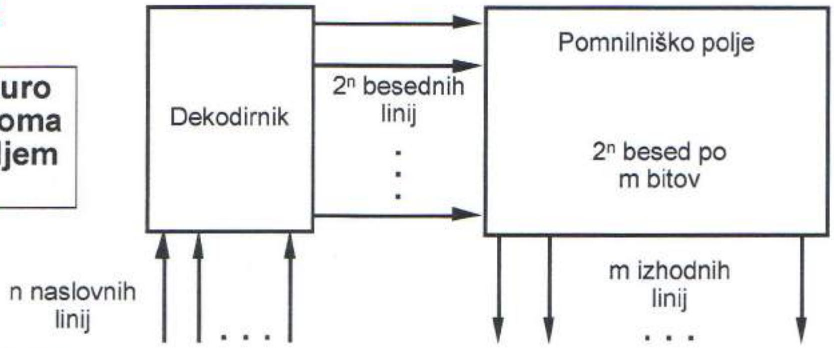
naslov → (first row A B C) — (first row F_0 … F_3) ← vsebina besede

A	B	C	F_0	F_1	F_2	F_3
0	0	0	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	1	0	0
0	1	1	0	0	0	1
1	0	0	1	0	1	1
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	1	0	0



Bralni pomnilnik

ROM ima strukturo PLA-ja s popolnoma dekodiranim poljem AND!



Uporabiti ROM ali PLA?

Uporaba ROM-a je primerna v naslednjih primerih:

(1) čas načrtovanja je kratek (ni treba minimizirati izhodnih funkcij),
(2) obstaja veliko edinstvenih produktnih členov,
(3) malo skupnih produktnih členov med izhodnimi funkcijami.

Problema z ROM-om: za vsak dodatni vhod se velikost ROM-a podvoji, ne moremo uporabiti vrednosti "don't care".

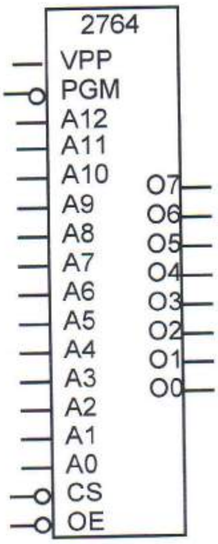
Uporaba PLA-ja je primerna v naslednjih primerih:

(1) na razpolago imamo načrtovalsko orodje kot npr. espresso,
(2) obstaja relativno malo edinstvenih produktnih členov,
(3) med izhodnimi funkcijami je mnogo skupnih produktnih členov.

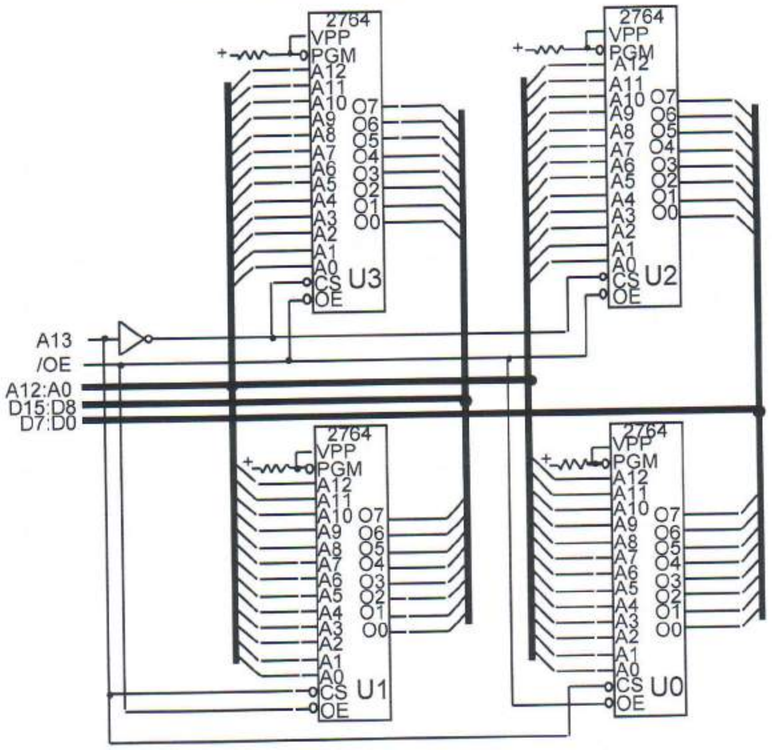
Problem s PAL-om: omejen fan-in v polju OR.

Bralni pomnilnik

EPROM 2764
8K x 8



Pomnilniški sistem 16K x 16



Primer načrtovanja kombinacijske logike

Splošen postopek načrtovanja

1. **Razumevanje problema**

   Kaj naj vezje dela?
   Zapišemo vhode (podatkovne in kontrolne) in izhode.
   Narišemo blokovni diagram ali kakšno drugo ilustracijo problema.

2. **Problem formuliramo v standardno predstavitev**

   pravilnostna tabela, Boolove enačbe, časovni diagram

3. **Izberemo način implementacije**

   dvonivojsko ali večnivojsko vezje iz diskretnih vrat, PAL, PLA, multipleksor, dekodirnik + vrata OR, ROM, gradnik AOI ali OAI

4. **Izvedemo postopke za implementacijo**

   K-diagrami, espresso, misII

Primer načrtovanja kombinacijske logike

Krmiljenje industrijskega procesa

Postavitev problema:

Na tekočem traku potujejo palice, ki se po dolžini razlikujejo za ±10%. Mehanična roka potiska palice, ki ne odstopajo za več kot ±5% od specificirane dolžine, na eno stran. Druga roka potiska predolge palice na drugo stran. Prekratke palice ostanejo na traku.

Uporabite tri svetlobne zapornice (vir svetlobe + fotocelica) kot senzorje in načrtajte kombinacijsko logiko za aktiviranje obeh mehaničnih rok.

Razumevanje problema:

Vhodi so trije senzorji, izhoda sta krmilna signala za mehanični roki.

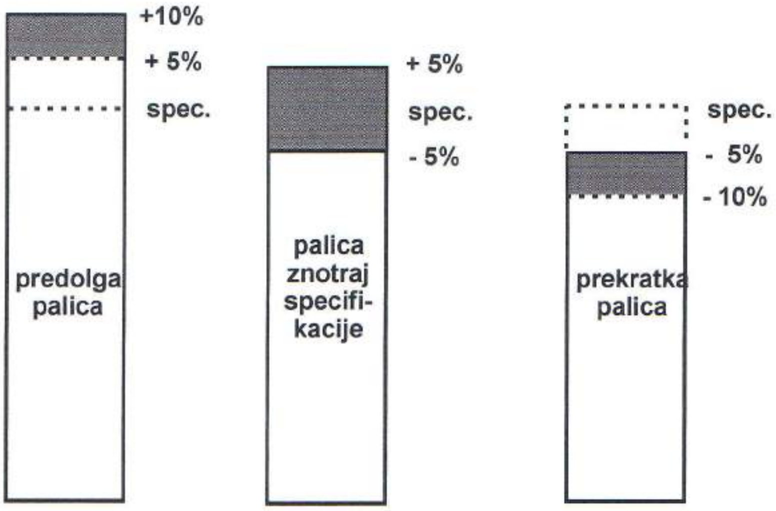
Predpostavimo, da senzor daje vrednost "1", ko se snop svetlobe prekine, sicer pa vrednost "0".

Imenujmo senzorje A, B in C.

Narišimo sliko!

Primer načrtovanja kombinacijske logike

Krmiljenje industrijskega procesa

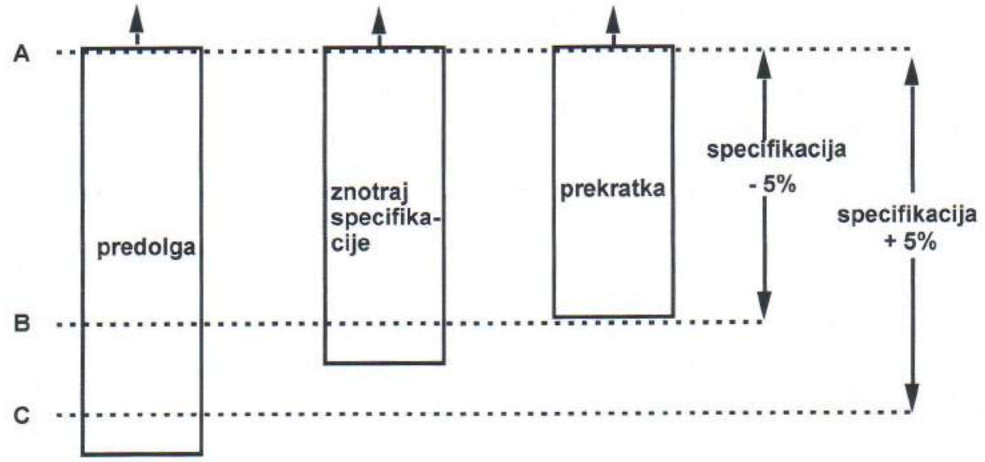


Kam naj postavimo svetlobne senzorje A, B in C, da bomo razlikovali te tri razrede?

Predpostavimo, da A detektira začetek palice na tekočem traku.

Primer načrtovanja kombinacijske logike

Krmiljenje industrijskega procesa



Razdalja med A in B je enaka specificirani dolžini palice - 5%.

Razdalja med A in C je enaka specificirani dolžini palice + 5%.

Primer načrtovanja kombinacijske logike

Krmiljenje industrijskega procesa

Problem zdaj zlahka formuliramo v obliki pravilnostne tabele.

A	B	C	funkcija
0	0	0	X
0	0	1	X
0	1	0	X
0	1	1	X
1	0	0	prekratka
1	0	1	X
1	1	0	znotraj specifikacije
1	1	1	predolga

Dejanska logika je zelo preprosta, zato jo lahko implementiramo kar z logičnimi vrati.

Pogoj "predolga" je predstavljen s funkcijo F=ABC.

Pogoj "znotraj specifikacije" je predstavljen s funkcijo G=ABC'.

Obe funkciji lahko implementiramo z dvoje 3-vhodnimi vrati AND in invertorjem.

Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

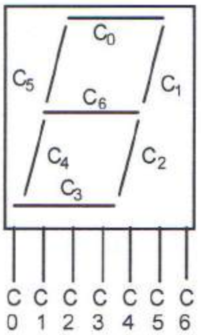
Razumevanje problema:

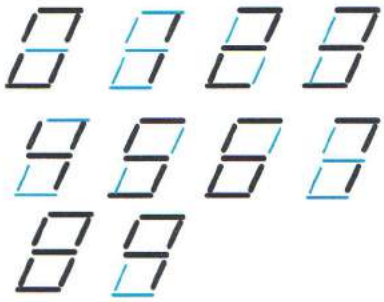
Vhod je 4-bitna BCD števka.

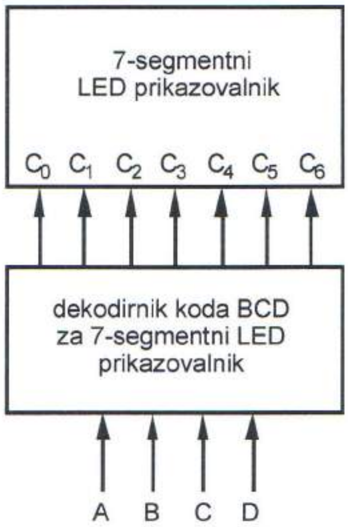
Izhodi so krmilni signali za LED prikazovalnik.

Imamo 4 vhode: A, B, C, in D.

Imamo 7 izhodov: C0 - C6.







Blokovni diagram

Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	X	X	X	X	X	X	X
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

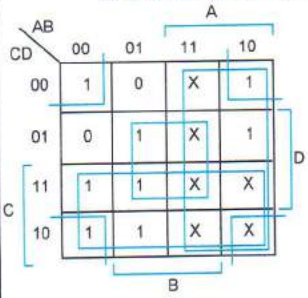
Problem formuliramo v obliki pravilnostne tabele.

**Izberemo način implementacije:
če izberemo ROM, vanj enostavno vpišemo to pravilnostno tabelo, vendar nedoločene vrednosti X v tabeli sugerirajo, da je lahko primernejša implementacija z vezjem PAL oz. PLA.**

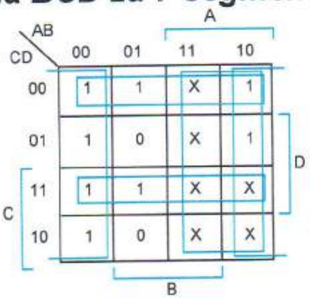
**Izvedemo proceduro za implementacijo:
ročna minimizacija s K-diagrami ali uporaba CAD orodja espresso.**

Primer načrtovanja kombinacijske logike

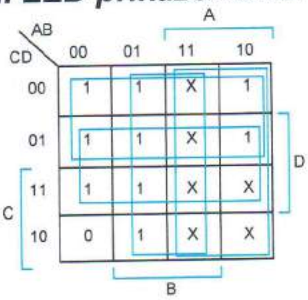
Dekodirnik koda BCD za 7-segmentni LED prikazovalnik



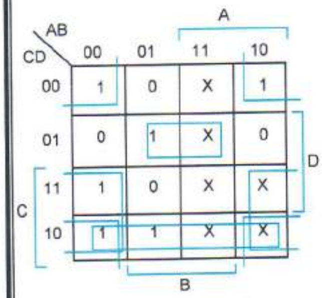
K-diagram za C_0



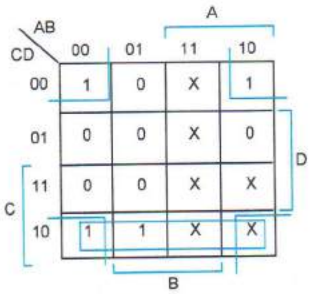
K-diagram za C_1



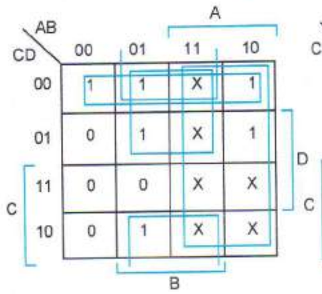
K-diagram za C_2



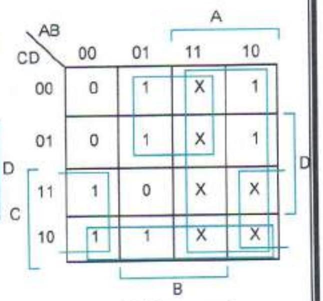
K-diagram za C_3



K-diagram za C_4



K-diagram za C_5



K-diagram za C_6

C0 = A + B D + C + B' D'
C1 = A + C' D' + C D + B'
C2 = A + B + C' + D

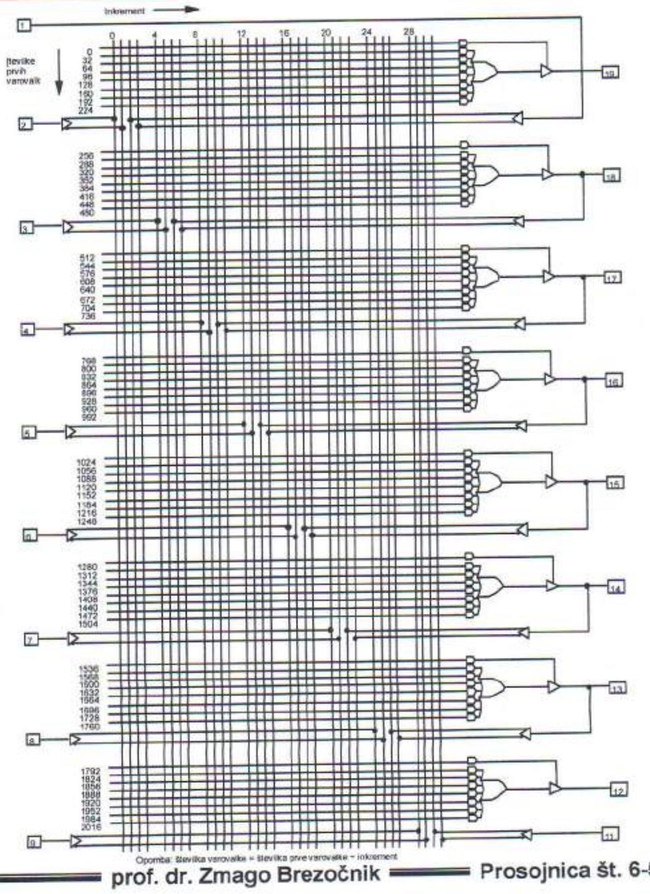
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

15 edinstvenih produktnih členov

Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

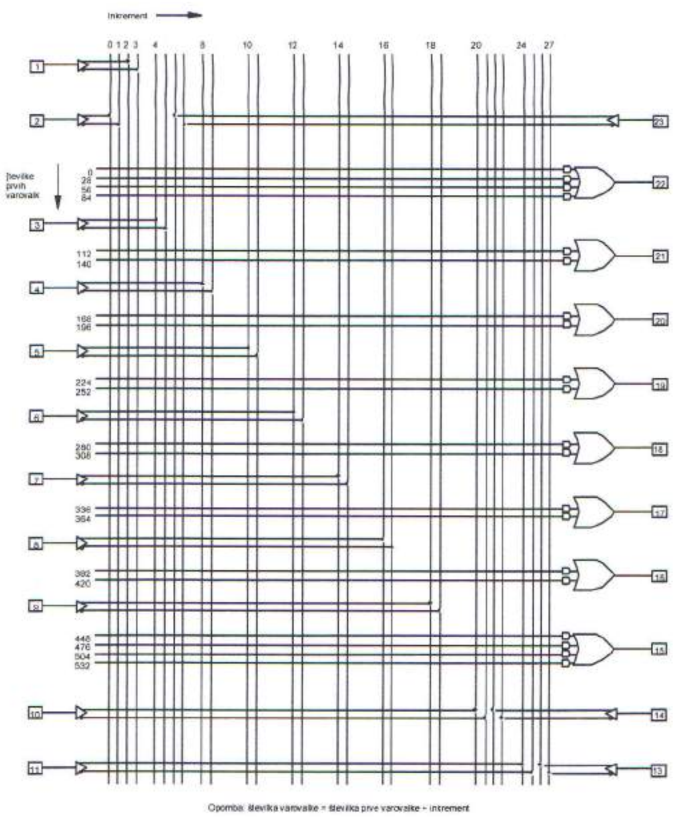
Dekodirnik lahko implementiramo s PAL-om 16H8.



Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

Dekodirnika ne moremo implementirati s PAL-om 14H8.



Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

```
.i 4
.o 7
.ilb a b c d
.ob c0 c1 c2 c3 c4 c5 c6
.p 16
0000 1111110
0001 0110000
0010 1101101
0011 1111001
0100 0110011
0101 1011011
0110 1011111
0111 1110000
1000 1111111
1001 1110011
1010 -------
1011 -------
1100 -------
1101 -------
1110 -------
1111 -------
.e
```

vhod v program espresso

```
.i 4
.o 7
.ilb a b c d
.ob c0 c1 c2 c3 c4 c5 c6
.p 9
-10- 0000001
-01- 0001001
-0-1 0110000
-101 1011010
--00 0110010
--11 1110000
-0-0 1101100
1--- 1000011
-110 1011111
.e
```

izhod iz programa espresso

C0 = B C' D + C D + B' D' + B C D' + A
C1 = B' D + C' D' + C D + B' D'
C2 = B' D + B C' D + C' D' + C D + B C D'
C3 = B C' D + B' C + B' D' + B C D'
C4 = B' D' + B C D'
C5 = B C' D + C' D' + A + B C D'
C6 = B' C + B C' + B C D' + A

9 edinstvenih produktnih členov

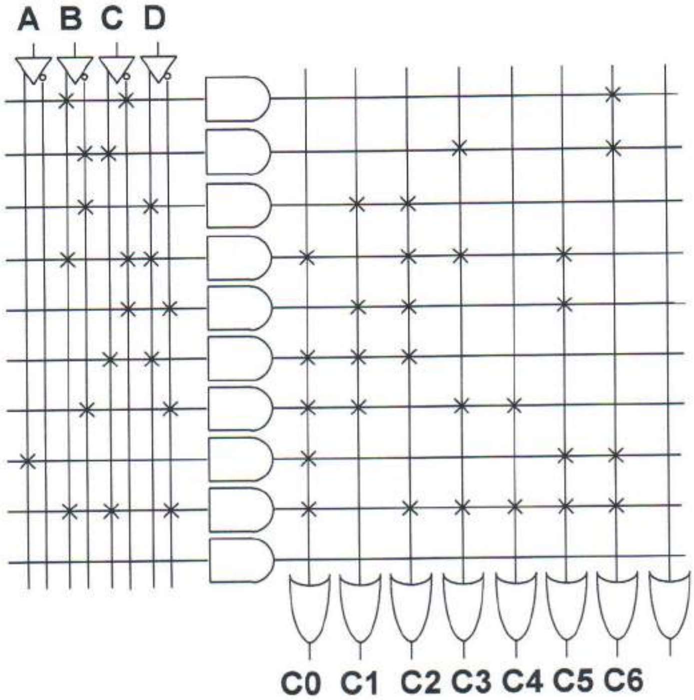
63 literalov, 18 vrat

Zaradi majhnega števila edinstvenih produktnih členov je primerna implementacija z vezjem PLA.

Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

Implementacija z vezjem PLA



Primer načrtovanja kombinacijske logike

Dekodirnik koda BCD za 7-segmentni LED prikazovalnik

Večnivojska implementacija

X = C' + D'

Y = B' C'

C0 = C3 + A' B X' + A D Y

C1 = Y + A' C5' + C' D' C6

C2 = C5 + A' B' D + A' C D

C3 = C4 + B D C5 + A' B' X'

C4 = D' Y + A' C D'

C5 = C' C4 + A Y + A' B X

C6 = A C4 + C C5 + C4' C5 + A' B' C

52 literalov, 33 vrat

Program misll ne zna izkoristiti nedoločenih vrednosti X. Vezje je precej počasnejše od dvonivojskega. Najbolj zakasnjen je izhod C1 (9 ali 10 zakasnitev posameznih vrat).

Primer načrtovanja kombinacijske logike

Logična funkcijska enota

Postavitev problema:

Načrtati moramo logično vezje, ki ima podatkovna vhoda A in B ter krmilne vhode C0, C1 in C2. Vezje naj bi implementiralo logično funkcijo F, ki je specificirana v spodnji funkcijski tabeli.

C0	C1	C2	F	komentar
0	0	0	1	vedno 1
0	0	1	$A + B$	logični OR
0	1	0	$\overline{AB}$	logični NAND
0	1	1	$A \oplus B$	logični XOR
1	0	0	$\overline{A \oplus B}$	logični XNOR
1	0	1	AB	logični AND
1	1	0	$\overline{A+B}$	logični NOR
1	1	1	0	vedno 0

3 krmilni vhodi: C0, C1, C2
2 podatkovna vhoda: A, B
1 izhod: F

F je kombinacijska logična funkcija spremenljivk C0, C1, C2, A in B.

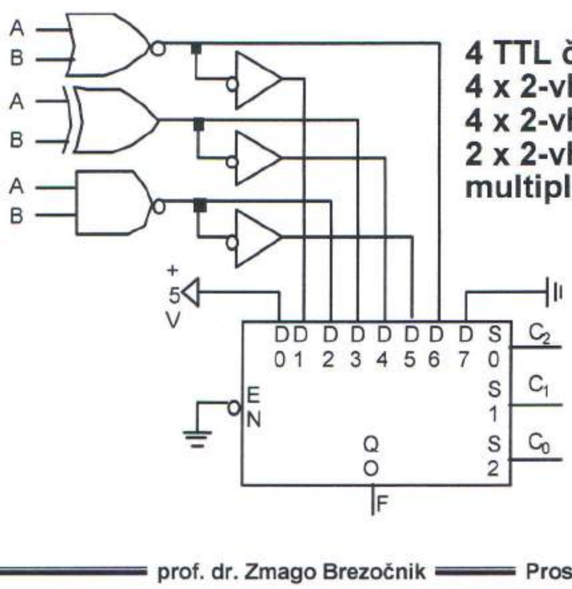
Primer načrtovanja kombinacijske logike

Logična funkcijska enota

C_0	C_1	C_2	A	B	F
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	1
0	0	1	0	0	0
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	0	0	1	0
1	0	0	1	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	0	1	0
1	0	1	1	0	0
1	0	1	1	1	1
1	1	0	0	0	1
1	1	0	0	1	0
1	1	0	1	0	0
1	1	0	1	1	0
1	1	1	0	0	0
1	1	1	0	1	0
1	1	1	1	0	0
1	1	1	1	1	0

Problem formuliramo s pravilnostno tabelo.

Izberemo način implementacije:
K-diagrami s 5 spremenljivkami,
espresso,
implementacija z multipleksorjem.



4 TTL čipi:
4 x 2-vhodna NAND
4 x 2-vhodna NOR
2 x 2-vhodna XOR
multipleksor 8:1

Primer načrtovanja kombinacijske logike

Logična funkcijska enota

Izvedemo proceduro za implementacijo

AB \ $C_0C_1C_2$	000	001	011	010	110	111	101	100
00	1	0	0	1	1	0	0	1
01	1	1	1	1	0	0	0	0
11	1	1	0	0	0	0	1	1
10	1	1	1	1	0	0	0	0

F = C2' A' B' + C0' A B' + C0' A' B + C1' A B

Za implementacijo potrebujemo štiri 3-vhodna vrata, ena 4-vhodna vrata in pet invertorjev. Vezje lahko implementiramo s tremi TTL čipi: 1 čip s tremi 3-vhodnimi vrati NAND, 1 čip z dvoje 4-vhodnimi vrati NAND in en čip s šestimi invertorji.
Mogoča je tudi alternativna implementacija z enim samim ROM-om 32x1.

Primer načrtovanja kombinacijske logike

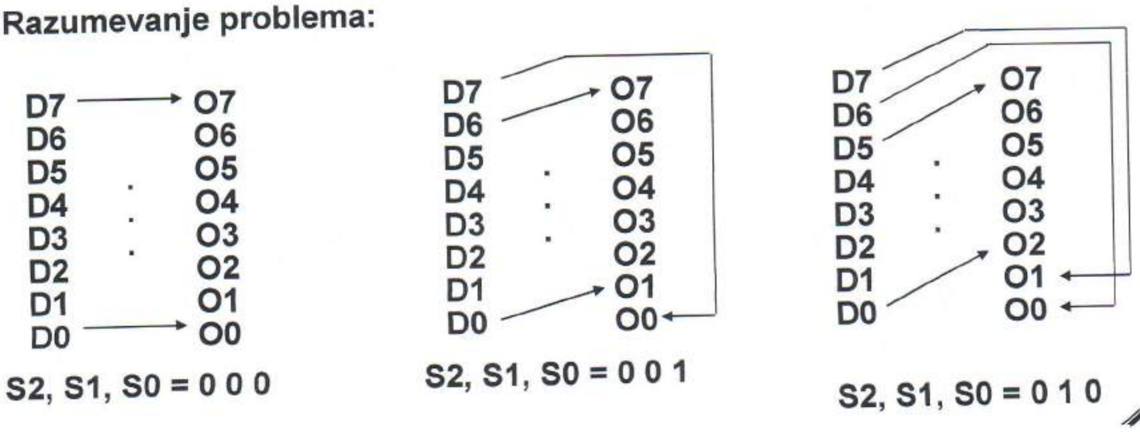
8-vhodni barrel pomikalnik

Specifikacija:

Podatkovni vhodi: D7, D6, …, D0
Podatkovni izhodi: O7, O6, …, O0
Krmilni vhodi: S2, S1, S0

Barrel pomikalnik pomakne vhod za specificirano število mest v levo.

Razumevanje problema:



Primer načrtovanja kombinacijske logike

8-vhodni barrel pomikalnik

Funkcijska tabela

S2	S1	S0	O7	O6	O5	O4	O3	O2	O1	O0
0	0	0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	D6	D5	D4	D3	D2	D1	D0	D7
0	1	0	D5	D4	D3	D2	D1	D0	D7	D6
0	1	1	D4	D3	D2	D1	D0	D7	D6	D5
1	0	0	D3	D2	D1	D0	D7	D6	D5	D4
1	0	1	D2	D1	D0	D7	D6	D5	D4	D3
1	1	0	D1	D0	D7	D6	D5	D4	D3	D2
1	1	1	D0	D7	D6	D5	D4	D3	D2	D1

Boolove enačbe

$$O7 = S2'\,S1'\,S0'\,D7 + S2'\,S1'\,S0\,D6 + \ldots + S2\,S1\,S0\,D0$$

$$O6 = S2'\,S1'\,S0'\,D6 + S2'\,S1'\,S0\,D5 + \ldots + S2\,S1\,S0\,D7$$

$$O5 = S2'\,S1'\,S0'\,D5 + S2'\,S1'\,S0\,D4 + \ldots + S2\,S1\,S0\,D6$$

$$O4 = S2'\,S1'\,S0'\,D4 + S2'\,S1'\,S0\,D3 + \ldots + S2\,S1\,S0\,D5$$

$$O3 = S2'\,S1'\,S0'\,D3 + S2'\,S1'\,S0\,D2 + \ldots + S2\,S1\,S0\,D4$$

$$O2 = S2'\,S1'\,S0'\,D2 + S2'\,S1'\,S0\,D1 + \ldots + S2\,S1\,S0\,D3$$

$$O1 = S2'\,S1'\,S0'\,D1 + S2'\,S1'\,S0\,D0 + \ldots + S2\,S1\,S0\,D2$$

$$O0 = S2'\,S1'\,S0'\,D0 + S2'\,S1'\,S0\,D7 + \ldots + S2\,S1\,S0\,D1$$

Primer načrtovanja kombinacijske logike

8-vhodni barrel pomikalnik

Alternativni načini implementacije:

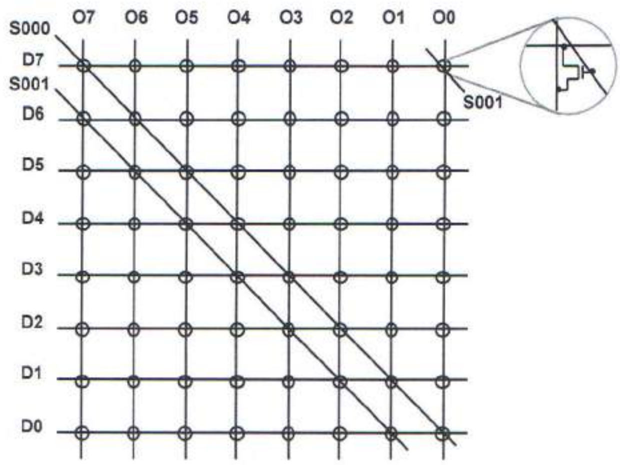
diskretna vrata (40 čipov: 32 za 4-vhodna in 8 za 8-vhodna vrata) ali

8 multipleksorjev 8:1 ali

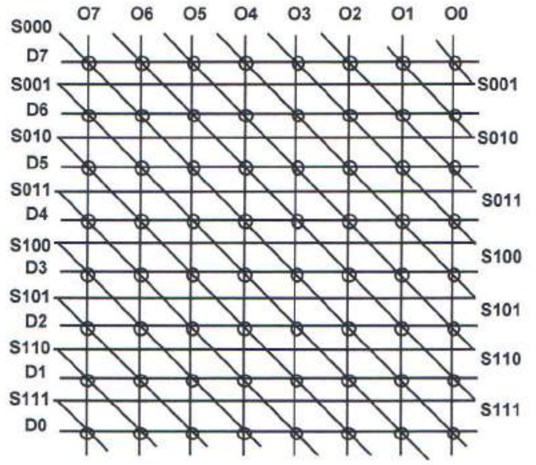
ROM s kapaciteto 2048 (2^{11}) 8-bitnih besed ali

PAL z 11 vhodi, 8 izhodi in 8 produktnimi členi na izhod ali

64 transistorskih stikal + dekodirnik 3:8



Mreža stikal



Popolnoma povezana mreža stikal

Povzetek poglavja

Predstavili smo:

- **Logične gradnike srednje stopnje integracije (MSI):**
  - **Vezja PAL in PLA**
  - **Multipleksor**
  - **Demultipleksorje in dekodirnike**
  - **Bralne pomnilnike**
  - **Tristanjske izhode in izhode z odprtim kolektorjem**

- **Probleme načrtovanja kombinacijske logike:**
  - **Razumevanje problema**
  - **Formulacija v standardno predstavitev**
  - **Izbira tehnologije za implementacijo**
  - **Izvedba postopkov za implementacijo**

7. Sekvenčna preklopna vezja

Digitalna tehnika

prof. dr. Zmago Brezočnik

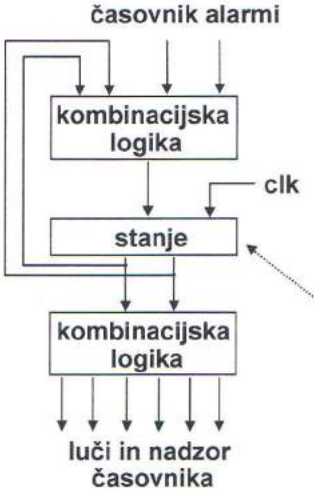
Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Vsebina poglavja

- *Sekvenčna preklopna vezja*
- *Flip-flopi*
- *Realizacija vezij z različnimi flip-flopi*
- *Model asinhronega sekvenčnega vezja*
- *Model sinhronega sekvenčnega vezja*
- *Analiza sinhronega sekvenčnega vezja*
- *Sinteza sinhronega sekvenčnega vezja*

Sekvenčna preklopna vezja

Vezja s povratno vezavo:
nekateri izhodi so tudi vhodi



Krmilnik križiščnih semaforjev je sekvenčno logično vezje.

Sekvenčna logika tvori osnovo za vgradnjo pomnilnika v vezja.

Ti pomnilniški elementi so osnovna sekvenčna vezja.

Sekvenčna preklopna vezja

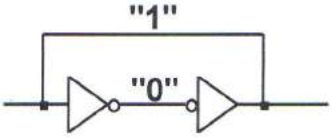
Osnovni pomnilniški elementi so zgrajeni iz zaporedno povezanih vrat.

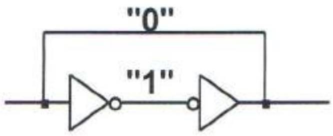
Najpreprostejša vrata: invertor.

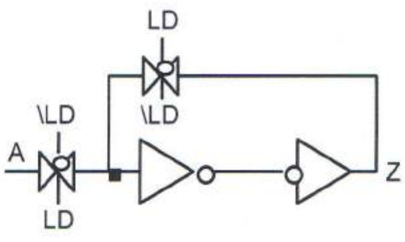
Osnova za načrtovanje statičnega RAM pomnilnika.

Mogoča je tudi izvedba z navzkrižno povezanimi vrati NOR ali NAND.

Zaporedna vezava invertorjev: statična pomnilniška celica.



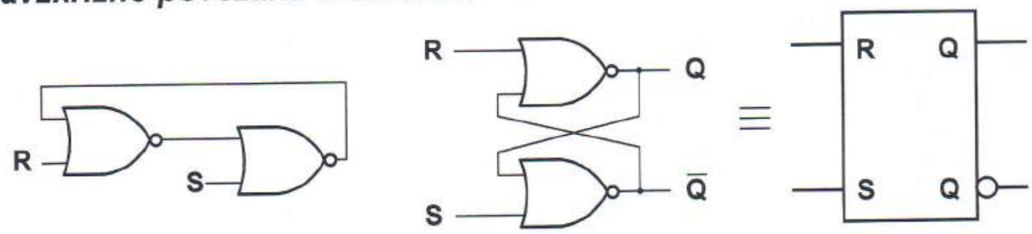




Prekinitev povratne vezave za naložitev nove vrednosti v celico.

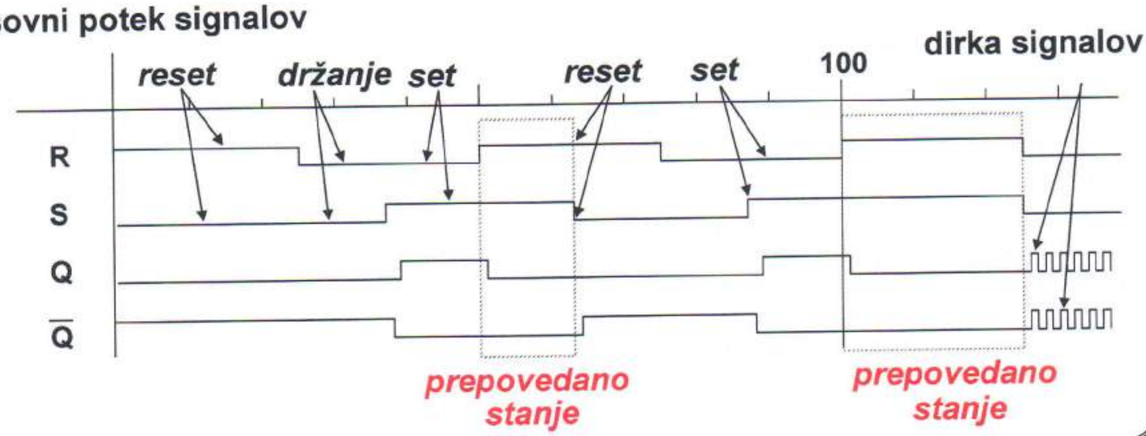
Sekvenčna preklopna vezja

Navzkrižno povezana vrata NOR



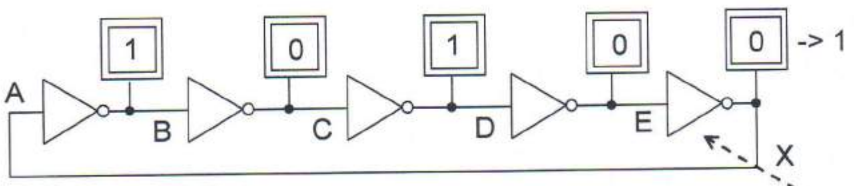
Obnašanje enako kot pri zaporedni vezavi invertorjev z možnostjo postavljanja izhoda na 0 (reset) ali 1 (set).

Časovni potek signalov



Sekvenčna preklopna vezja

Veriga invertorjev



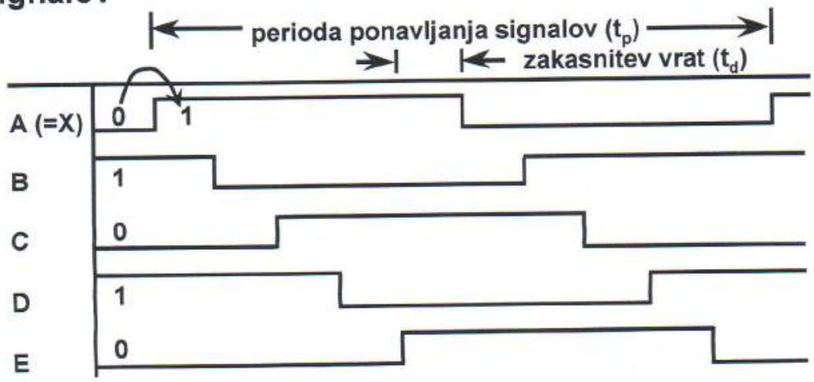
Liho število invertorjev vodi do krožne oscilacije. Posnetek je narejen tik pred spremembo vrednosti na izhodu zadnjega invertorja.

postavljanje izhoda tega invertorja

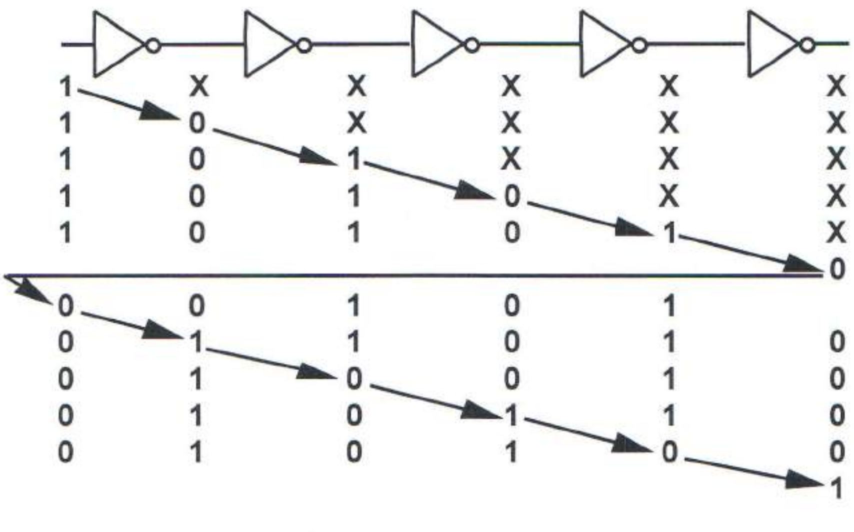
Časovni potek signalov

$t_p = 2 * n * t_d$

n = št. invertorjev



Sekvenčna preklopna vezja



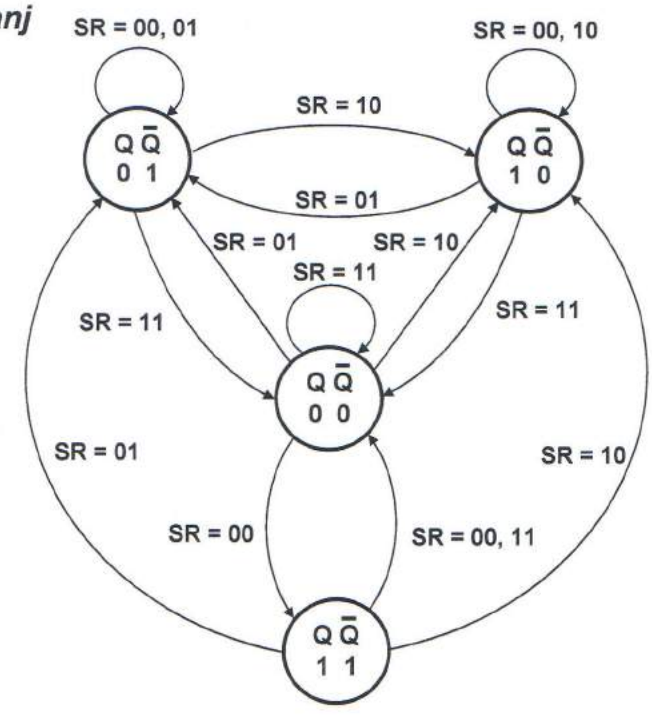
širjenje signalov skozi verigo invertorjev

Sekvenčna preklopna vezja

Teoretični diagram prehajanja stanj za SR zadrževalnik

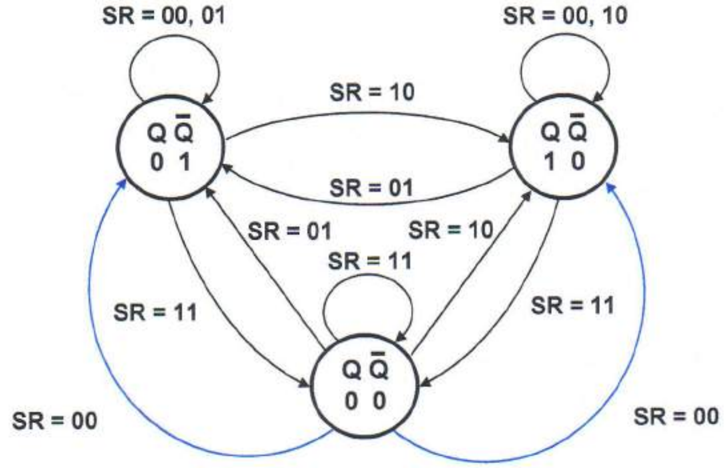
S	R	Q
0	0	držanje
0	1	0
1	0	1
1	1	nestabilno

Funkcionalna pravilnostna tabela za SR zadrževalnik



Sekvenčna preklopna vezja

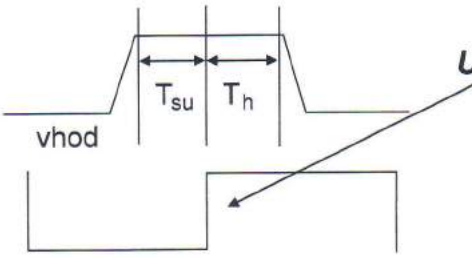
Dejansko obnašanje SR zadrževalnika



Zelo težko je zaznati SR zadrževalnik v stanju 11.
V odvisnosti od zakasnitev v logičnih vratih dirka signalov pripelje zadrževalnik bodisi v stanje 01 bodisi v stanje 10.

Sekvenčna preklopna vezja

Ura, nastavitveni čas, čas držanja



Ura (Clock):

Periodični urin dogodek, ki povzroči spremembo stanja v pomnilniških elementih. Dogodek je lahko pozitivna *fronta*, negativna *fronta*, visok *nivo* ali nizek *nivo* urinega signala.

Nastavitveni čas (Setup Time – T_{su})

Je minimalni časovni interval pred nastopom urinega dogodka, znotraj katerega mora biti vhod stabilen

Čas držanja (Hold Time – T_h)

Je minimalni časovni interval po nastopu urinega dogodka, znotraj katerega mora biti vhod stabilen.

Obstaja časovno okno okrog urinega dogodka, znotraj katerega mora vhod ostati stabilen, da zagotovimo pravilno delovanje pomnilniškega elementa.

Sekvenčna preklopna vezja

SR zadrževalnik

naslednje stanje = f(S, R, sedanje stanje)

S(t)	R(t)	Q(t)	Q(t+Δ)	
0	0	0	0	} DRŽANJE
0	0	1	1	
0	1	0	0	} BRISANJE
0	1	1	0	
1	0	0	1	} POSTAVLJANJE
1	0	1	1	
1	1	0	X	} NEDOVOLJENO
1	1	1	X	

K-diagram za Q(t+Δ) :

Q(t) \ SR	00	01	11	10
0	0	0	X	1
1	1	0	X	1

(S: stolpca 11, 10; R: stolpca 01, 11)

Karakteristična enačba:

$$Q(t+\Delta) = S(t) + \overline{R(t)}Q(t)$$

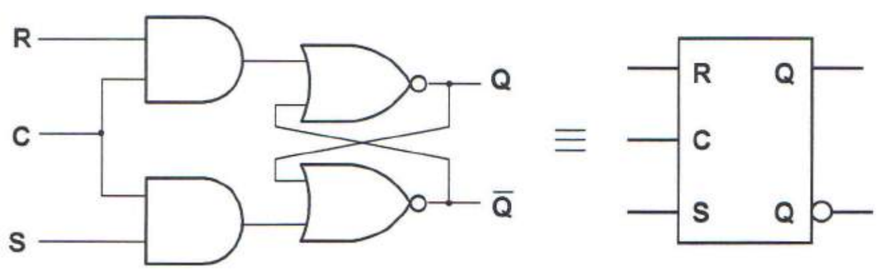
oz.

$$Q^{+} = S + \overline{R}Q$$

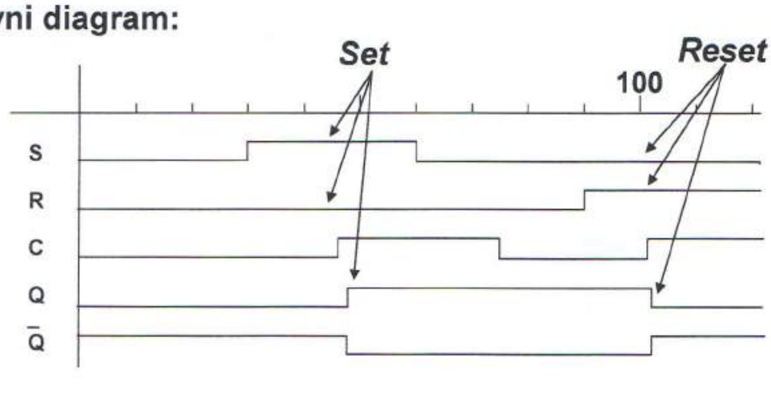
Sekvenčna preklopna vezja

SR zadrževalnik z omogočitvenim vhodom

Shema:



Časovni diagram:



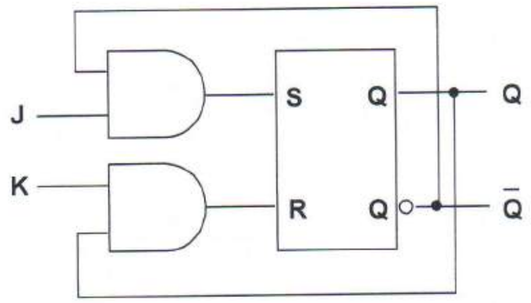
Sekvenčna preklopna vezja

JK zadrževalnik

Kako izločiti prepovedano stanje?

Ideja: uporabimo povratno vezavo iz izhodov in s tem zagotovimo, da S in R nikoli nista oba na 1.

Če sta J in K oba na 1, se Q obrne.



J(t)	K(t)	Q(t)	Q(t+Δ)	
0	0	0	0	} DRŽANJE
0	0	1	1	
0	1	0	0	} BRISANJE
0	1	1	0	
1	0	0	1	} POSTAVLJANJE
1	0	1	1	
1	1	0	1	} OBRAČANJE
1	1	1	0	

Karakteristična enačba:

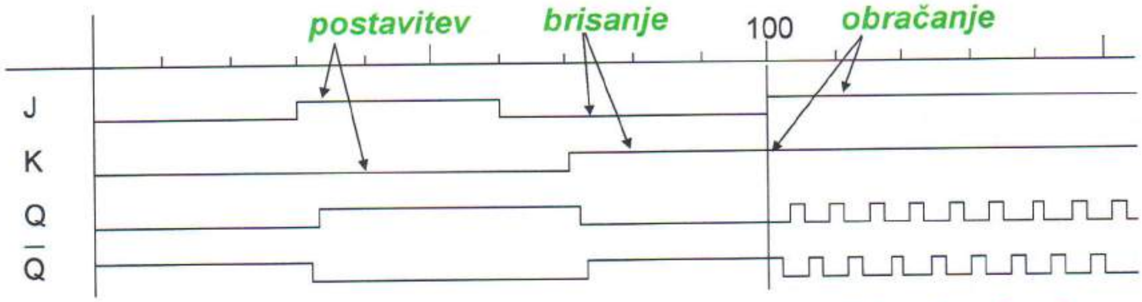
$$Q(t+\Delta) = J(t)\overline{Q}(t) + \overline{K(t)}Q(t)$$

oz.

$$Q^{+} = J\,\overline{Q} + \overline{K}\,Q$$

Sekvenčna preklopna vezja

JK zadrževalnik: tekma signalov



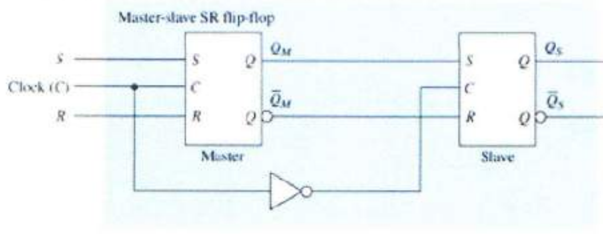
dirka signalov!

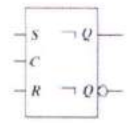
Želimo samo eno spremembo stanja, ne pa oscilacij.

Rešitev: flip-flop "master/slave"

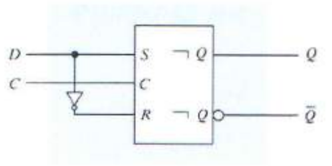
Flip-flopi

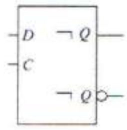
SR flip-flop "master/slave"



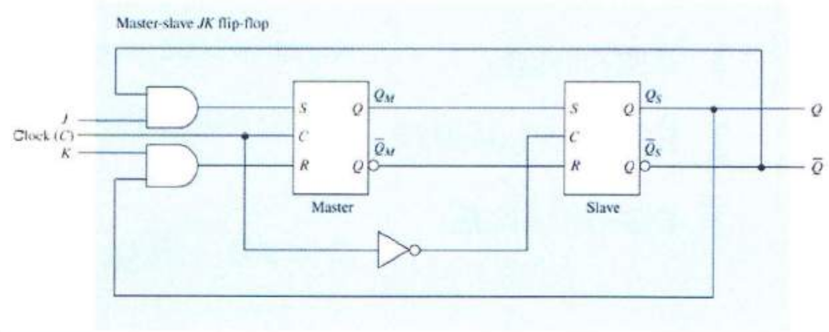


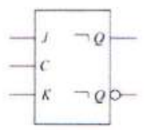
D flip-flop "master/slave"





JK flip-flop "master/slave"



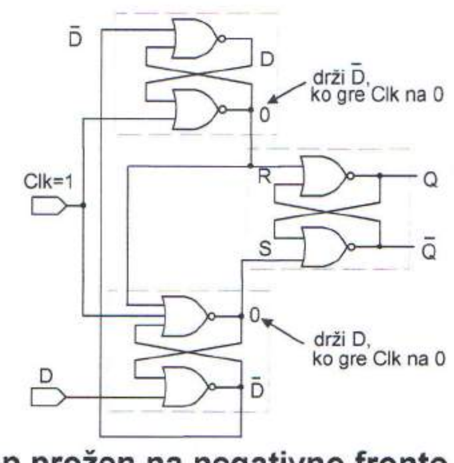


Flip-flopi

Flip-flopi proženi na fronto

Ujetje enice: konica **0-1-0 na vhodih J ali K vodi do spremembe stanja! Prisiljeni smo načrtati logiko brez hazardov.**

Rešitev: **logika za proženje na fronto**



D flip-flop prožen na negativno fronto, ko je Clk=1.

D flip-flop, prožen na negativno fronto

4-5 zakasnitev vrat

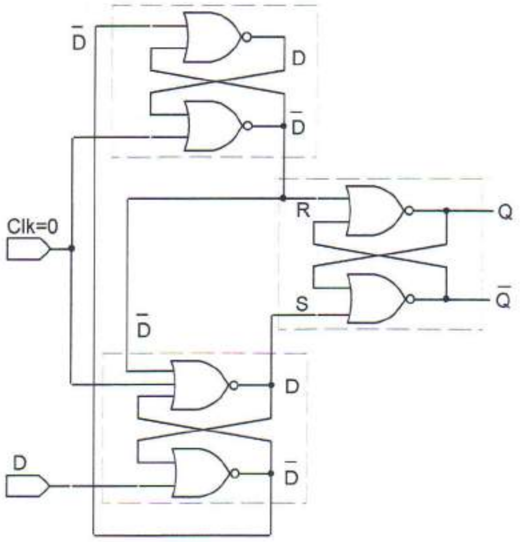
potrebni nastavitveni in držalni časi za uspešno zadržanje vhoda

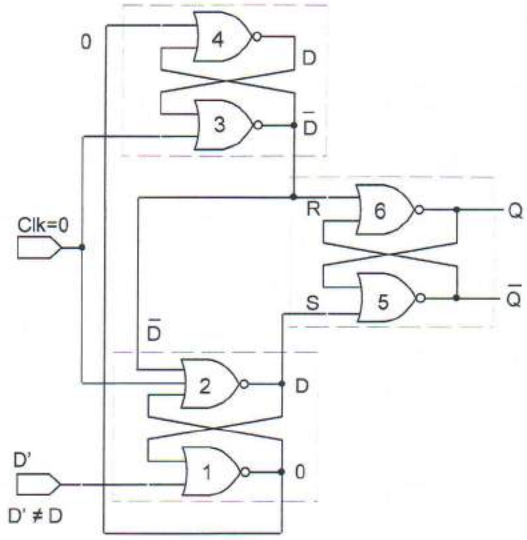
karakteristična enačba:

$Q^+ = D$

Flip-flopi

Flip-flopi proženi na fronto: analiza obnašanja po korakih





D flip-flop prožen na negativno fronto, ko gre Clk iz 1 na 0 – podatek se zadrži.

D flip-flop prožen na negativno fronto, ko je Clk=0 – podatek se obdrži, četudi D spremeni vrednost.

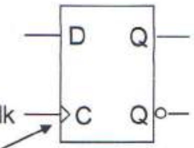
Flip-flopi

Vhodno-izhodno obnašanje zadrževalnikov in flip-flopov

Tip pomn. celice	Kdaj se vzorčijo vhodi?	Kdaj se spremeni izhod?
zadrževalnik (brez ure)	nenehno	po preteku zakasnitve razširjanja od vhodne spremembe
zadrževalnik prožen na nivo	ob visokem nivoju ure (T_{su} in T_h okrog negativne fronte ure)	po preteku zakasnitve razširjanja od vhodne spremembe
flip-flop prožen na pozitivno fronto	ob prehodu ure z 0 na 1 (T_{su} in T_h okrog pozitivne fronte ure)	po preteku zakasnitve razširjanja od pozitivne fronte ure
flip-flop prožen na negativno fronto	ob prehodu ure z 1 na 0 (T_{su} in T_h okrog negativne fronte ure)	po preteku zakasnitve razširjanja od negativne fronte ure
flip-flop "master/slave"	ob prehodu ure z 1 na 0 (T_{su} in T_h okrog negativne fronte ure)	po preteku zakasnitve razširjanja od negativne fronte ure

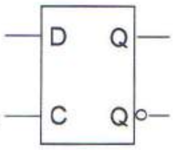
Flip-flopi

7474



Flip-flop prožen
na pozitivno fronto

7476



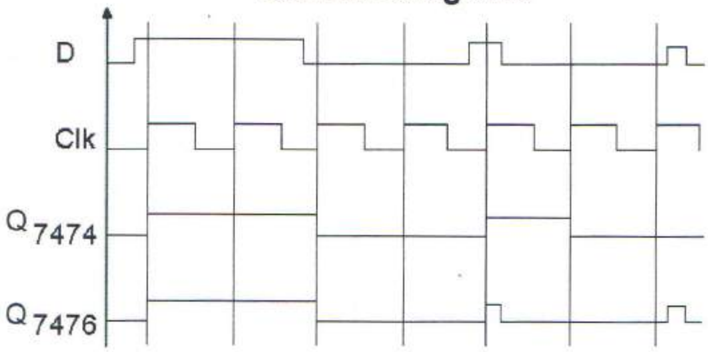
Zadrževalnik prožen
na nivo

Proženje na negativno fronto bi bilo označeno s krožcem.

__*Flip-flopi*__ vzorčijo vhode ob aktivni fronti ure.

__*Zadrževalniki*__ vzorčijo vhode, dokler je urin signal na visokem nivoju.

Časovni diagram:



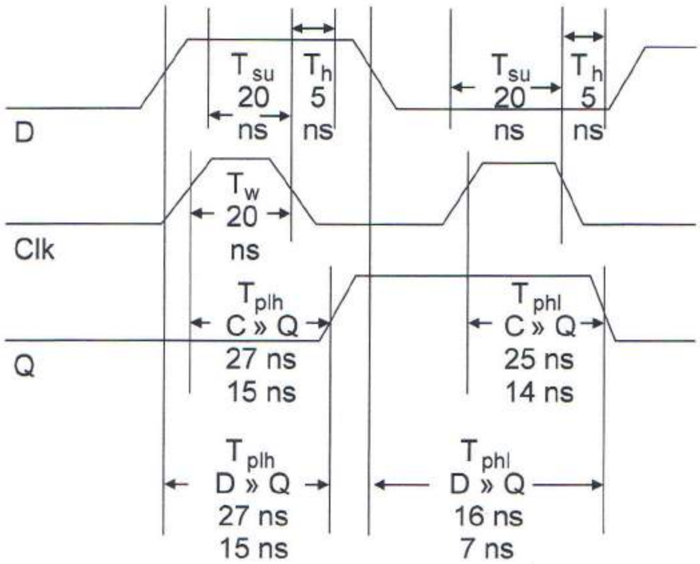
Obnašanji sta enaki, razen če se vhod spremeni, medtem ko je urin signal visok.

Flip-flopi

Tipične časovne specifikacije

74LS76: D zadrževalnik

- **Tsu: nastavitveni čas**
- **Th: čas držanja**
- **Tw: minimalna širina ure**
- **Tplh, Tphl: zakasnitve razširjanja (nizko na visoko, visoko na nizko, maksimalno, tipično)**



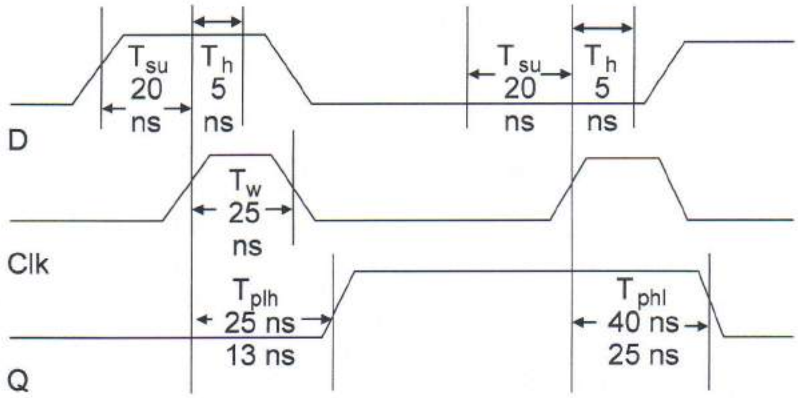
Trenutek zadržanja je ob negativni fronti urinega signala.

Flip-flopi

Tipične časovne specifikacije

74LS74: D flip-flop prožen na pozitivno fronto

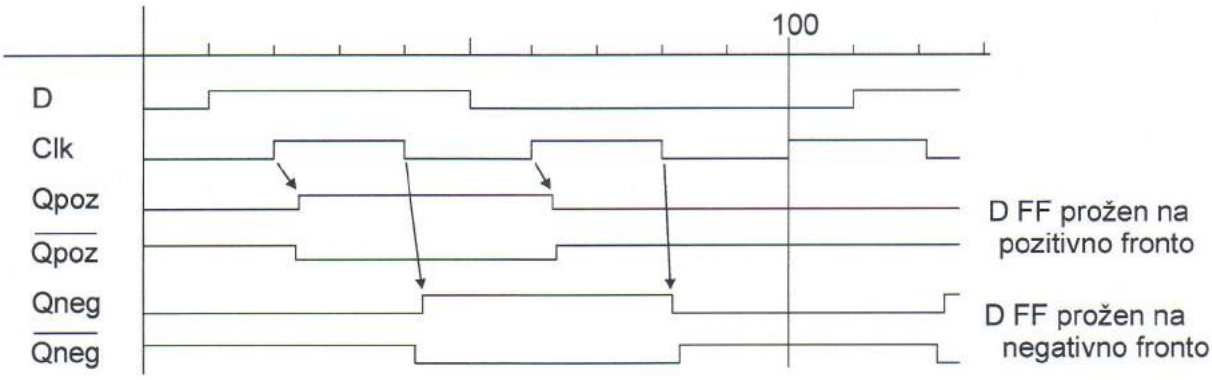
- **T_{su}: nastavitveni čas**
- **T_h: čas držanja**
- **T_w: minimalna širina ure**
- **T_{plh}, T_{phl}: zakasnitve razširjanja (nizko na visoko, visoko na nizko, maksimalno, tipično)**



Vse meritve so napravljene od pozitivne fronte urinega signala.

Flip-flopi

Primerjava med proženjem na pozitivno in proženjem na negativno fronto



Proženje na pozitivno fronto	***Proženje na negativno fronto***
Vhod je vzorčen na pozitivno fronto, izhod se spremeni po pozitivni fronti.	**Vhod je vzorčen na negativno fronto, izhod se spremeni po negativni fronti.**

T flip-flop

T flip-flop dobimo iz JK flip-flopa, če povežemo vhoda J in K ter tako dobljeni vhod poimenujemo T (toggle).

Flip-flopi

SR flip-flop

Simbol

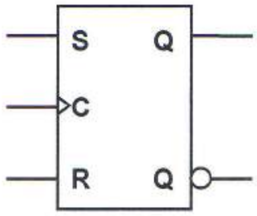
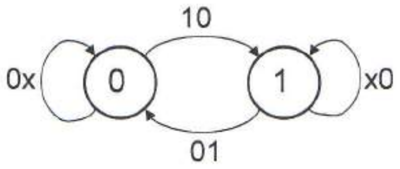


Tabela prehajanja stanj

sedanje stanje Q	naslednje stanje Q^+ sedanja vhoda SR = 00	01	11	10
0	0	0	x	1
1	1	0	x	1

Diagram prehajanja stanj:



Karakteristična enačba:

$$Q^+ = S + \overline{R}Q,\ SR = 0$$

Digitalna tehnika — *Sekvenčna preklopna vezja*

Flip-flopi

JK flip-flop

Simbol

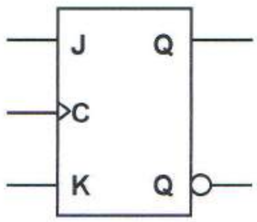
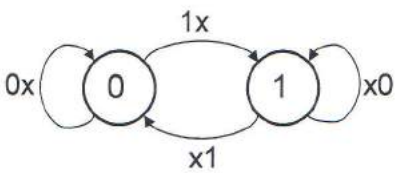


Tabela prehajanja stanj

sedanje stanje Q	naslednje stanje Q^+ sedanja vhoda JK = 00	01	11	10
0	0	0	1	1
1	1	0	0	1

Diagram prehajanja stanj:



Karakteristična enačba:

$$Q^+ = J\overline{Q} + \overline{K}Q$$

Flip-flopi

D flip-flop

Simbol

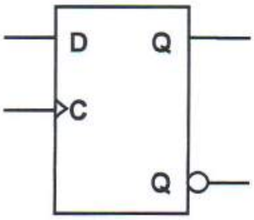
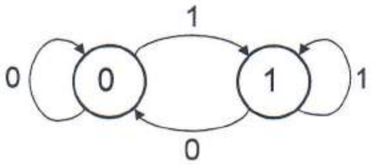


Tabela prehajanja stanj

sedanje stanje Q	naslednje stanje Q^+ sedanji vhod D = 0	1
0	0	1
1	0	1

Diagram prehajanja stanj:



Karakteristična enačba:

$$Q^+ = D$$

Digitalna tehnika — Sekvenčna preklopna vezja

Flip-flopi

T flip-flop

Simbol

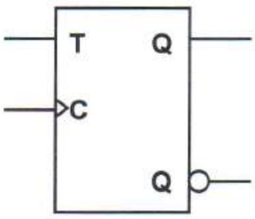
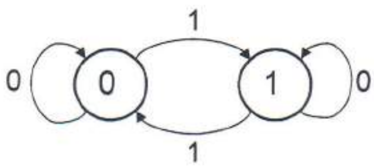


Tabela prehajanja stanj

sedanje stanje Q	naslednje stanje Q^+ sedanji vhod T = 0	1
0	0	1
1	1	0

Diagram prehajanja stanj:



Karakteristična enačba:

$$Q^+ = \overline{T}Q + T\overline{Q}$$

Splošna karakteristična enačba in vzbujevalna tabela

Splošna karakteristična enačba

Splošno karakteristično enačbo lahko zapišemo kot $Q^+ = g_1Q + g_2\overline{Q}$, kjer sta g_1 in g_2 funkciji, odvisni od primarnih vhodov.

Vzbujevalna tabela

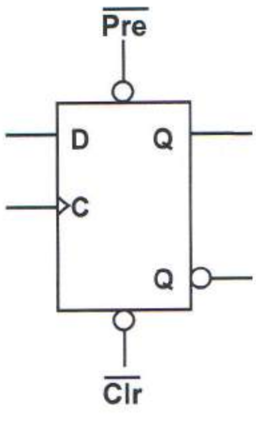
Za vsako pomnilno celico lahko zapišemo *vzbujevalno tabelo*, ki pove, kakšne logične vrednosti moramo postaviti na vhode pomnilnih celic, da bomo ob urini fronti dosegli želen prehod.

želen prehod $Q \rightarrow Q^+$		vhodi flip-flopov			
		SR	D	JK	T
0	0	0x	0	0x	0
0	1	10	1	1x	1
1	0	01	0	x1	1
1	1	x0	1	x0	0

Asinhroni vhodi za brisanje in postavljanje

Pomnilne celice imajo ponavadi še dva dodatna kontrolna vhoda, namenjena za inicializacijo stanja. Aktivni signal na vhodu za brisanje ($\overline{Clr}$) prisili pomnilno celico v stanje Q=0, na vhodu za postavljanje ($\overline{Pre}$) pa v stanje Q=1. Kontrolna vhoda za inicializacijo sta *asinhrona*, saj učinkujeta takoj in v ničemer nista odvisna od ure.

D flip-flop z asinhronima vhodoma za brisanje in postavljanje



Ob sočasnem aktiviranju obeh asinhronih vhodov ponavadi prevlada brisanje.

Realizacija vezij z različnimi flip-flopi

Izbira flip-flopa

SR zadrževalnik z omogočitvenim vhodom:
- Njegova uporaba ni priporočljiva (pogoj SR = 0)!
- Kljub temu je temeljni gradnik drugih tipov flip-flopov.

J-K flip-flop:
- Je vsestranski pomnilniški gradnik.
- Uporabimo ga lahko za izvedbo D in T flip-flopa.
- Pogosto zahteva najmanjši obseg logike za izvedbo funkcije naslednjega stanja.
- Ima dva vhoda, kar povečuje kompleksnost povezovanja.

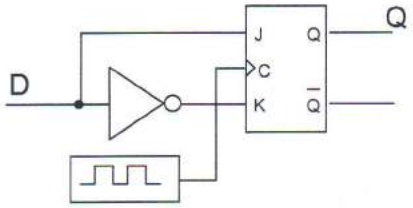
D flip-flop:
- Minimizira število povezav, ima prednost v VLSI tehnologijah.
- Zaradi najpreprostejše karakteristične enačbe je z njim najlažje načrtovati.
- Je najboljša izbira za pomnilne registre.

T flip-flop:
- So redko razpoložljivi v čipih, izvedemo jih z JK flip-flopi.
- Ponavadi so najboljša izbira za izvedbo števnikov.

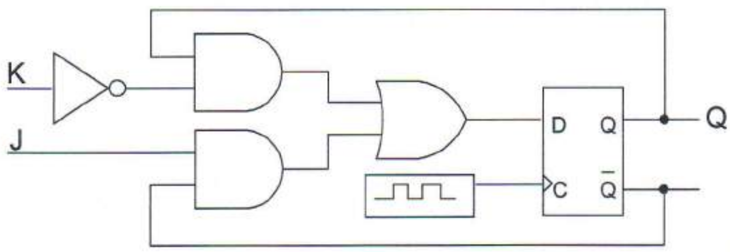
Asinhrona vhoda Preset and Clear sta zelo zaželena!

Realizacija vezij z različnimi flip-flopi

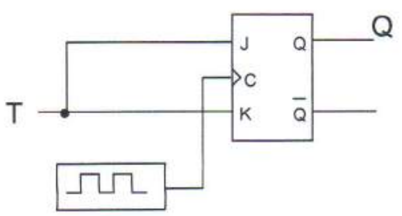
Izvedba enega tipa flip-flopa z drugim



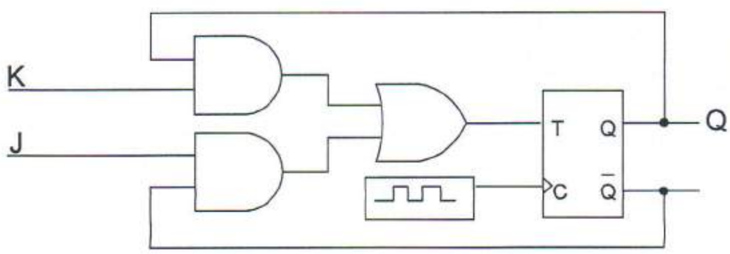
D FF izveden z JK FF



JK FF izveden z D FF



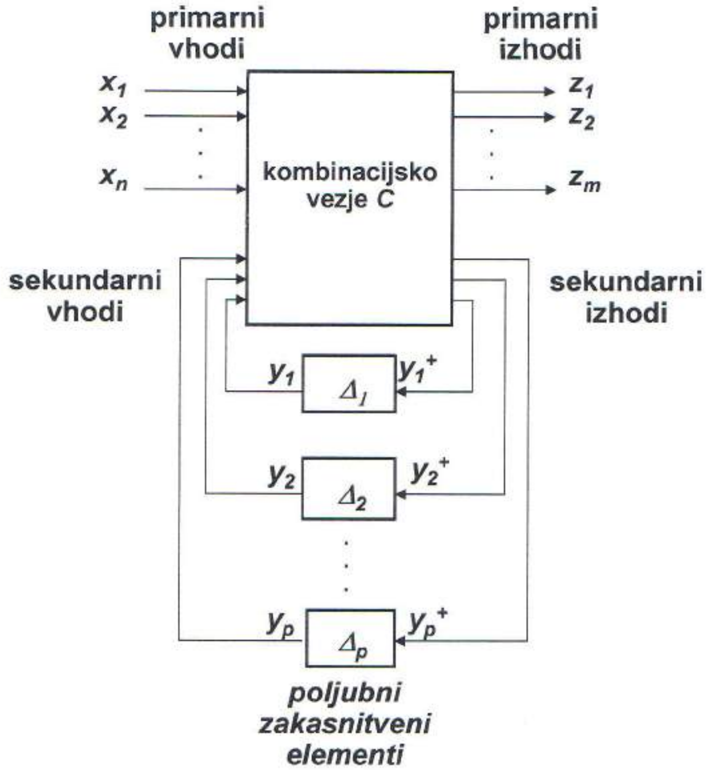
T FF izveden z JK FF



JK FF izveden s T FF

Model asinhronega sekvenčnega vezja

Zakasnitveni elementi imajo različne dolge zakasnitvene čase $\Delta_1, \Delta_2, \ldots, \Delta_p$, kar povzroči, da se signali razširjajo skozi vezje z različnimi hitrostmi in lahko medsebojno učinkujejo vnaprej nepredvidljivo in na neželene načine (kritične tekme signalov).

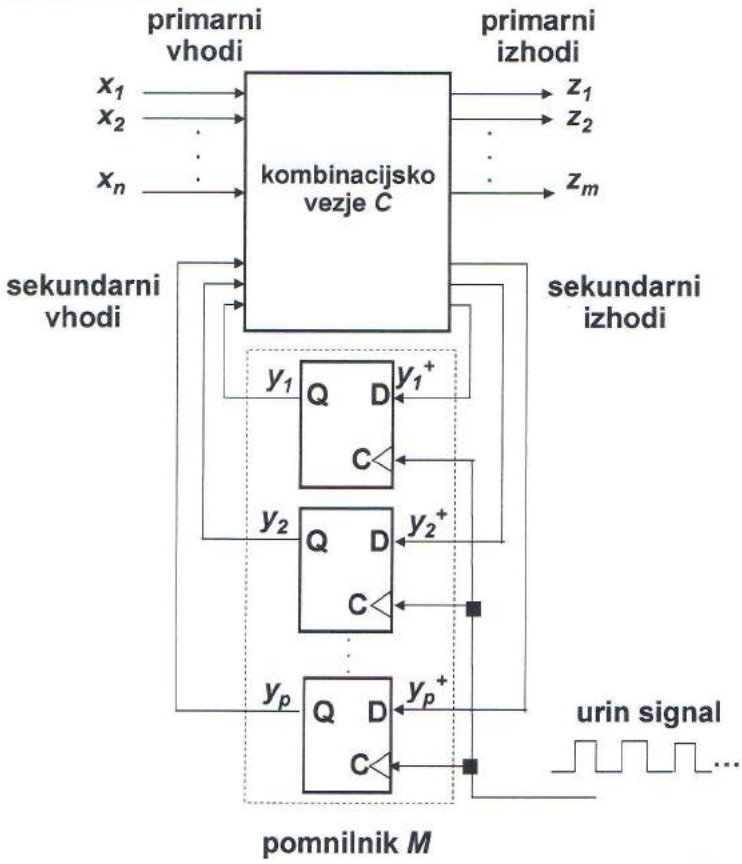


Model sinhronega sekvenčnega vezja

Splošni model sinhronega sekvenčnega vezja dobimo, če poljubne zakasnitvene elemente zamenjamo z D flip-flopi, ki so proženi z istim urinim signalom.

p-terica sekundarnih vhodnih signalov predstavlja vso informacijo, ki si jo je vezje zapomnilo od prejšnjih aktivnosti.

Kombinacijska logika C generira sekundarne izhodne signale, ki predstavljajo naslednje stanje sekvenčnega vezja.



Model sinhronega sekvenčnega vezja

Predstavitev modela z D flip-flopi je dovolj splošna, saj lahko vsak flip-flop drugega tipa pretvorimo v flip-flop tipa D. Če bi v modelu za pomnilnik izbrali JK flip-flope, bi bili sekundarni izhodi (naslednje stanje) definirani posredno s p pari signalov J in K, ki jih generira C.

S p flip-flopi lahko predstavimo 2^p različnih stanj, vsakemu je pridružen svoj vzorec spremenljivk stanja. Da bi realizirali sekvenčno vezje s P stanji, potrebujemo

$p = \lceil \log_2 P \rceil$ flip-flopov.

Sekundarni izhodi (naslednje stanje) so odvisni od primarnih in sekundarnih vhodov. Primarni izhodi so lahko odvisni od primarnih in sekundarnih vhodov – dobimo sinhrona sekvenčna vezja tipa Mealy, če pa so odvisni le od sekundarnih vhodov, dobimo sinhrona sekvenčna vezja tipa Moore.

Analiza sinhronih sekvenčnih vezij

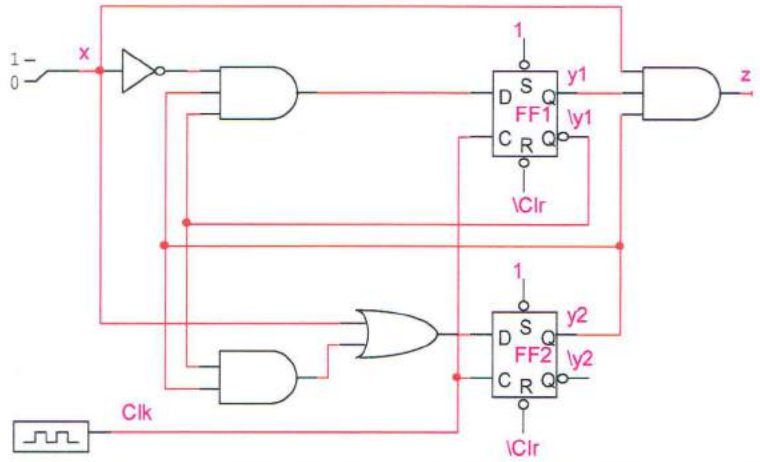
Problem: Podana je struktura sinhronega sekvenčnega vezja. Ugotoviti želimo, kako se vezje obnaša. Analizo izvajamo po naslednjih korakih.

1. Vsaki pomnilni celici v sinhronem sekvenčnem vezju priredimo spremenljivko stanja.
2. Za vsako pomnilno celico zapišemo vhodne vzbujevalne enačbe. Zapišemo tudi vse izhodne enačbe.
3. Vhodne vzbujevalne enačbe vstavimo v karakteristične enačbe uporabljenih pomnilnih celic, da dobimo enačbe za naslednje stanje.
4. Iz enačb za naslednje stanje in iz izhodnih enačb sestavimo tabelo prehajanja stanj, diagram prehajanja stanj ali časovni diagram, da prikažemo obnašanje vezja.

Analiza sinhronih sekvenčnih vezij

Primer: **Analizirajte naslednje sinhrono sekvenčno vezje**

1. korak: Vsakemu **flip-flopu v vezju priredimo spremenljivko stanja.**



2. korak: **Zapišemo vzbujevalno enačbo za vsak flip-flop in izhodno enačbo.**

$D_1 = x'y_1'y_2$ $\quad D_2 = x + y_1'y_2$ $\quad z = xy_1y_2 \Longrightarrow$ **Vezje je tipa Mealy.**

3. korak: **Vzbujevalno enačbo vsakega flip-flopa vstavimo v karakteristično enačbo za D flip-flop in dobimo enačbi za naslednje stanje.**

$y_1^+ = D_1 = x'y_1'y_2$ $\quad y_2^+ = D_2 = x + y_1'y_2$

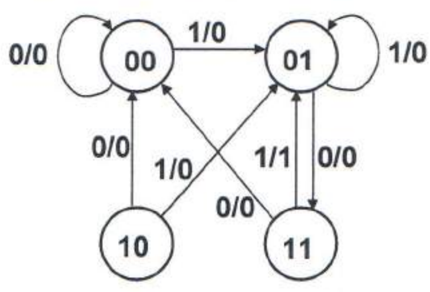
Analiza sinhronih sekvenčnih vezij

4. korak: **Tvorimo tabelo in diagram prehajanja stanj ter primer časovnega poteka.**

Tabela prehajanja stanj:

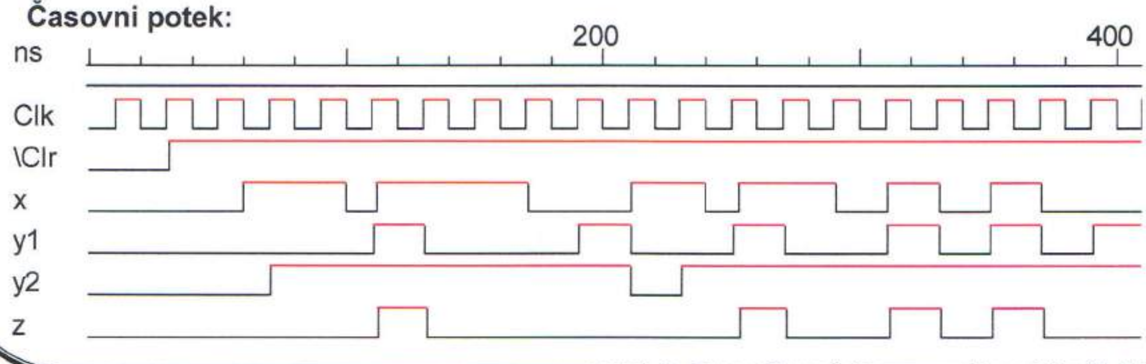
sedanje stanje y_1y_2	nasl. stanje $y_1^+y_2^+$/ sedanji izhod z; sedanji vhod x = 0	x = 1
00	00/0	01/0
01	11/0	01/0
11	00/0	01/1
10	00/0	01/0

Diagram prehajanja stanj:



Vezje odkriva vhodno zaporedje 101.

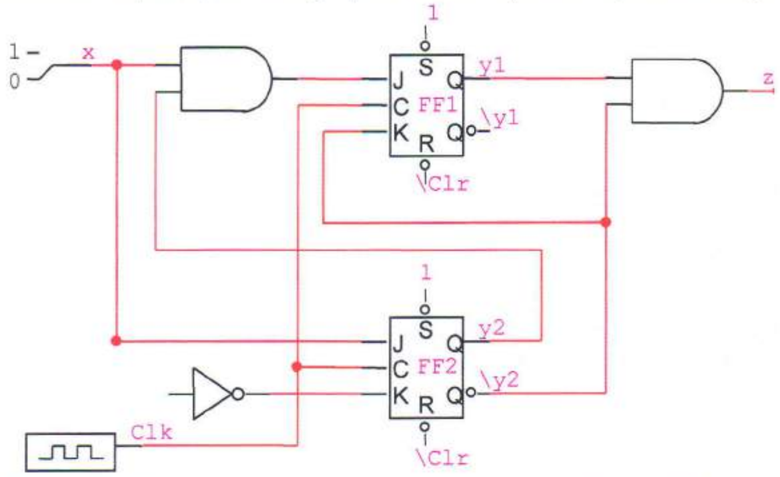
Časovni potek:



Analiza sinhronih sekvenčnih vezij

Primer: **Analizirajte naslednje sinhrono sekvenčno vezje.**

1. korak: Vsakemu **flip-flopu v vezju priredimo spremenljivko stanja.**



2. korak: Zapišemo vzbujevalno enačbo za vsak flip-flop in izhodno enačbo.

$J_1 = xy_2$ $\quad$ $J_2 = x$ $\quad$ $z = y_1y_2' \Longrightarrow$ **Vezje je tipa Moore.**

$K_1 = y_2'$ $\quad$ $K_2 = x'$

3. korak: Vzbujevalno enačbo vsakega flip-flopa vstavimo v karakteristično enačbo za JK flip-flop in dobimo enačbi za naslednje stanje.

$y_1^+ = J_1y_1' + K_1'y_1 = xy_1'y_2 + y_1y_2$ $\qquad$ $y_2^+ = J_2y_2' + K_2'y_2 = xy_2' + xy_2 = x$

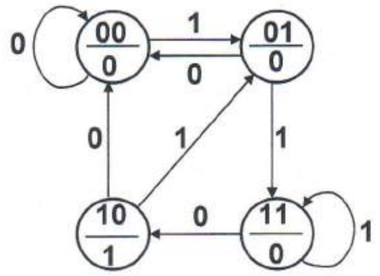
Analiza sinhronih sekvenčnih vezij

4. korak: **Tvorimo tabelo in diagram prehajanja stanj ter primer časovnega poteka.**

Tabela prehajanja stanj:

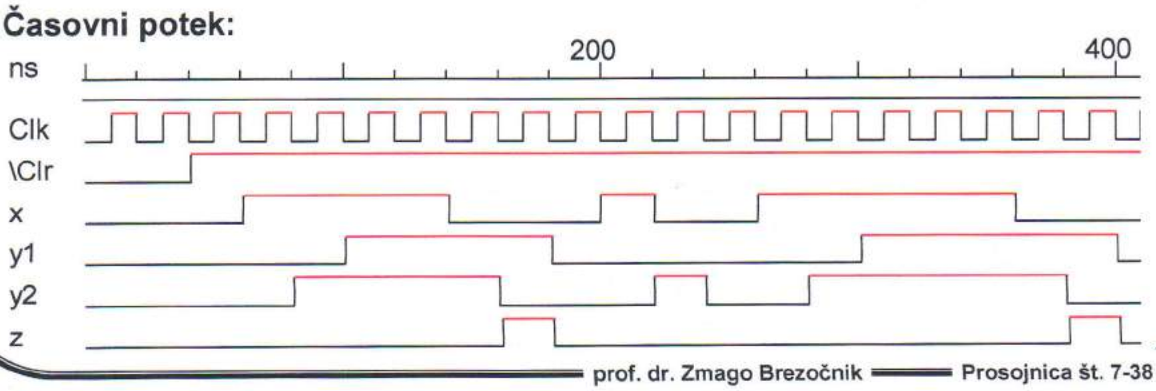
sedanje stanje y_1y_2	naslednje stanje $y_1^+y_2^+$ sedanji vhod x = 0	1	sedanji izhod z
00	00	01	0
01	00	11	0
11	10	11	0
10	00	01	1

Diagram prehajanja stanj:



Vezje odkriva vhodno zaporedje 110.

Časovni potek:



Sinteza sinhronih sekvenčnih vezij

Problem: **Podano je zahtevano obnašanje sinhronega sekvenčnega vezja. Izpeljati želimo čimboj ekonomično strukturo (vezje) s takim obnašanjem. Sintezo izvajamo po naslednjih korakih.**

1. **Iz besednega opisa problema sestavimo diagram ali tabelo prehajanja stanj, ki formalno specificira obnašanje vezja.**
2. **Iz sestavljenega diagrama ali tabele prehajanja stanj odstranimo vsa morebitna redundantna stanja. (Postopek za odkrivanje in odpravljanje redundantnih stanj bo obravnavan v naslednjem poglavju. Do takrat bomo obravnavali primere brez redundantnih stanj.)**
3. **Stanja zakodiramo. *P* stanj zakodiramo s *p* spremenljivkami stanj (y_1, y_2, ..., y_p), kjer je $2^p \geq P$. Dobimo zakodiran diagram ali zakodirano tabelo prehajanja stanj.**
4. **Izberemo tip pomnilne celice. Izpeljemo funkcije za primarne izhode in sekundarne izhode (vzbujevalne enačbe).**
5. **Vezje narišemo.**

Sinteza sinhronih sekvenčnih vezij

Primer: **Potrebujemo sinhrono sekvenčno vezje z vhodom *x* in izhodom *z*, ki generira izhod *z* = 1, ko se na vhodu odkrije zaporedje 0101. V vseh drugih časih je *z* = 0. Npr., za vhodno zaporedje 010101 je izhodno zaporedje 000101. Za pomnilne celice uporabimo JK flip-flope.**

1. korak: **Narišemo diagram in tabelo prehajanja stanj.**

Diagram prehajanja stanj:

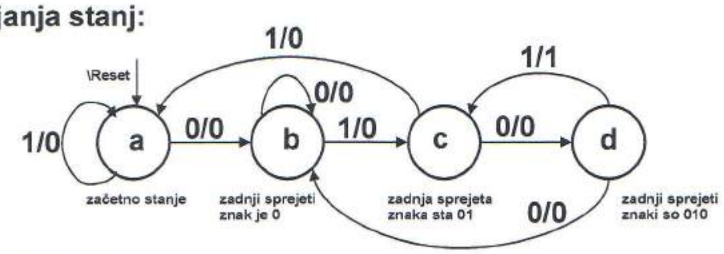


Tabela prehajanja stanj:

sedanje stanje	naslednje stanje / sedanji izhod z, sedanji vhod *x* = 0	1
a	b/0	a/0
b	b/0	c/0
c	d/0	a/0
d	b/0	c/1

Analiza sinhronih sekvenčnih vezij

2. korak: **Ker redundantnih stanj ni, gremo na točko 3.**

3. korak: **Za zakodiranje štirih stanj potrebujemo dve spremenljivki stanja. Označimo ju z y_1 in y_2.**

Izberemo kode za stanja.

ime stanja	koda stanja y_1y_2
a	00
b	01
c	11
d	10

Dobimo zakodirano tabelo stanj.

sedanje stanje y_1y_2	nasl. stanje $y_1^+y_2^+$/ sedanji izhod z, sedanji vhod $x=$ 0	1
00	01/0	00/0
01	01/0	11/0
11	10/0	00/0
10	01/0	11/1

Sinteza sinhronih sekvenčnih vezij

4. korak: **Izberemo JK flip-flope. Narišemo vzbujevalno tabelo.**

primarni vhod x	sedanje stanje y_1y_2	naslednje stanje $y_1^+y_2^+$	sekundarni izhodi $J_1K_1J_2K_2$	primarni izhod z
0	0 0	0 1	0 x 1 x	0
0	0 1	0 1	0 x x 0	0
0	1 0	0 1	x 1 1 x	0
0	1 1	1 0	x 0 x 1	0
1	0 0	0 0	0 x 0 x	0
1	0 1	1 1	1 x x 0	0
1	1 0	1 1	x 0 1 x	1
1	1 1	0 0	x 1 x 1	0

Sinteza sinhronih sekvenčnih vezij

S počjo Karnaughovih diagramov izpeljemo minimalne oblike funkcij za sekundarne izhode in primarni izhod.

J_1:

y_1y_2 \ x	0	1
00	0	0
01	0	1
11	x	x
10	x	x

K_1:

y_1y_2 \ x	0	1
00	x	x
01	x	x
11	0	1
10	1	0

J_2:

y_1y_2 \ x	0	1
00	1	0
01	x	x
11	x	x
10	1	1

K_2:

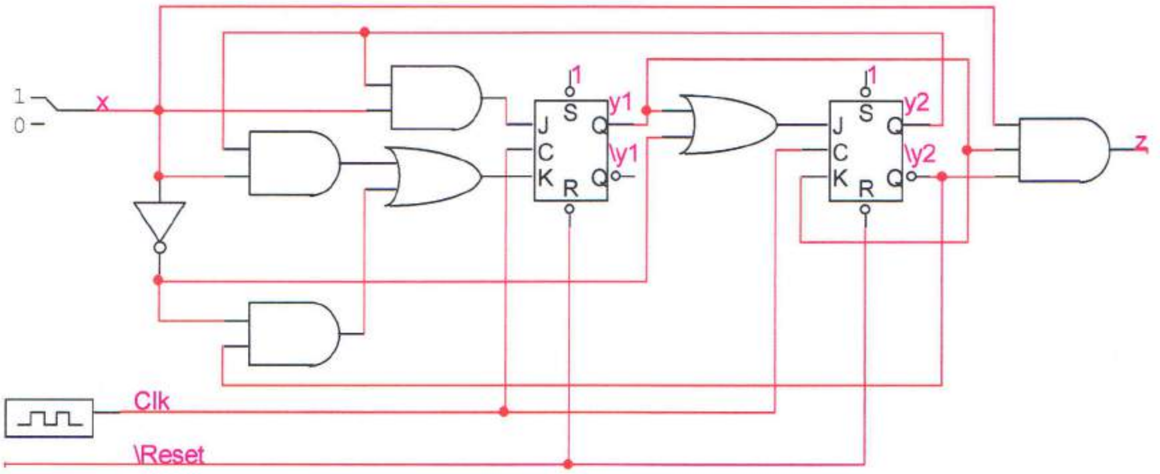
y_1y_2 \ x	0	1
00	x	x
01	0	0
11	1	1
10	x	x

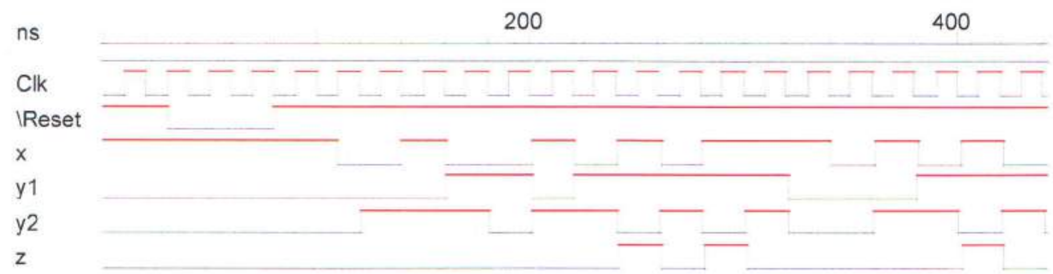
$J_1 = xy_2$ $\quad K_1 = x'y_2' + xy_2$ $\quad J_2 = x' + y_1$ $\quad K_2 = y_1$

$z = xy_1y_2'$

Sinteza sinhronih sekvenčnih vezij

5. korak: **Vezje narišemo. Na koncu pravilnost delovanja preverimo s simulacijo.**





Sinteza sinhronih sekvenčnih vezij

Še o zakodiranju stanj

Obstaja mnogo načinov za zakodiranje stanj z binarnimi kodami. Koliko? S p spremenljivkami stanj ($y_1, y_2, \ldots, y_p$) lahko dobimo 2^p različnih kod. Zakodirati moramo P različnih stanj. Izmed 2^p možnih kod lahko izberemo P potrebnih na

$$\binom{2^p}{P} = \frac{2^p!}{(2^p - P)!P!} \quad \text{načinov.}$$

Izbrano množico kod lahko permutiramo na $P!$ načinov, tako da dobimo

$$\binom{2^p}{P}P! = \frac{2^p!\,P!}{(2^p - P)!P!} = \frac{2^p!}{(2^p - P)!} \quad \text{možnih zakodiranj.}$$

Na primer, za $P = 9$ in $p = 4$ obstaja $\frac{2^4!}{(2^4 - 9)!}$ = 4.151.347.200 različnih zakodiranj.

Povzetek poglavja

Predstavili smo:

- **Osnovne gradnike vezij s pomnjenjem: zadrževalnike in flip-flope**
- **Osnovne zadrževalnike brez omogočitvenega vhoda, zadrževalnike z omogočitvenim vhodom, master-slave flip-flope, flip-flope prožene na fronto (SR, JK, D, T)**
- **Realizacijo vezij z različnimi flip-flopi**
- **Model sinhronega sekvenčnega vezja**
- **Postopek analize sinhronega sekvenčnega vezja**
- **Postopek sinteze sinhronega sekvenčnega vezja**

8. Končni avtomati

Digitalna tehnika

prof. dr. Zmago Brezočnik

Univerza v Mariboru
Fakulteta za elektrotehniko, računalništvo in informatiko

Vsebina poglavja

- *Končni avtomati – osnovne definicije*
- *Ekvivalenca stanj in minimizacija avtomata*
- *Pretvorba avtomatov iz tipa Moore v tip Mealy in obratno*
- *Računanje s particijami*
- *Serijska dekompozicija končnih avtomatov*
- *Paralelna dekompozicija končnih avtomatov*
- *Iskanje particij s substitucijsko značilnostjo*

Avtomati

- *Avtomat* je skupek množic in relacij med njimi.
- Avtomate obravnava *teorija o avtomatih* (Automata Theory).
- Avtomati se uporabljajo na najrazličnejših področjih: v digitalnih sistemih, računalništvu, matematiki, avtomatiki, robotiki, jezikoslovju, družbenih vedah itd.
- Realizaciji avtomata lahko rečemo informacijski stroj (ali kar stroj). Sekvenčna vezja, ki smo jih obdelali v prejšnjem poglavju, ustrezajo informacijskemu stroju ali realizaciji avtomata.
- Glavni razvijalci teorije o avtomatih so bili: David A. Huffman, Edward F. Moore, George H. Mealy, Alan M. Turing in John von Neumann v letih od 1936 do 1960.
- Od vseh avtomatov so najbolj omejeni a najbolj razširjeni *končni avtomati*. Jeziki, ki jih razpoznavajo končni avtomati, so natanko *regularni jeziki*.
- Najbolj splošen je *Turingov stroj* oz. Turingov avtomat, ki predstavlja teoretični model računanja z neomejenim pomnilnikom.
- Po splošnosti se med obe skrajnosti umeščajo t. im. *skladovni* avtomati. To so končni avtomati, ki lahko uporabljajo sklad za zapisovanje podatkov. Jeziki, ki jih razpoznavajo skladovni avtomati, so natanko *kontekstno neodvisni jeziki*.
- Zanimali nas bodo samo *deterministični avtomati*, pri katerih lahko izid prehoda iz nekega stanja v drugo stanje ob danem vhodnem simbolu zmeraj napovemo.

Končni avtomati – osnovne definicije

Definicija. Končni avtomat tipa Mealy A_{ME} je šesterica

$$A_{ME} = (I, S, O, \delta, \lambda, s_0),$$

kjer je

I končna in neprazna množica vhodnih črk – vhodna abeceda,
S končna in neprazna množica notranjih črk (stanj) avtomata – notranja abeceda,
O končna in neprazna množica izhodnih črk – izhodna abeceda,
$\delta: I \times S \rightarrow S$ funkcija prehajanja stanj,
$\lambda: I \times S \rightarrow O$ izhodna funkcija,
$s_0 \in S$ *začetno stanje, ki definira začetek delovanja avtomata.*■

Funkcijo prehajanja stanj in izhodno funkcijo lahko izrazimo s

$$s^+ = \delta(i,s), \qquad o = \lambda(i,s), \qquad i \in I, s \in S, s^+ \in S, o \in O,$$

kjer je

i sedanja vhodna črka,
s sedanja notranja črka (sedanje stanje),
s^+ naslednja notranja črka (naslednje stanje) in
o sedanja izhodna črka avtomata.

Končni avtomati – osnovne definicije

Definicija. **Končni avtomat tipa Moore A_{MO} je šesterica**

$$A_{MO} = (I, S, O, \delta, \lambda, s_0),$$

kjer je

- **I končna in neprazna množica vhodnih črk – vhodna abeceda,**
- **S končna in neprazna množica notranjih črk (stanj) avtomata – notranja abeceda,**
- **O končna in neprazna množica izhodnih črk – izhodna abeceda,**
- **$\delta: I \times S \rightarrow S$ funkcija prehajanja stanj,**
- **$\lambda: S \rightarrow O$ izhodna funkcija,**
- **$s_0 \in S$ *začetno stanje, ki definira začetek delovanja avtomata.*■**

Funkcijo prehajanja stanj in izhodno funkcijo lahko izrazimo s

$$s^+ = \delta(i,s), \qquad o = \lambda(s), \qquad i \in I,\ s \in S,\ s^+ \in S,\ o \in O,$$

kjer je

- **i sedanja vhodna črka,**
- **s sedanja notranja črka (sedanje stanje),**
- **s^+ naslednja notranja črka (naslednje stanje) in**
- **o sedanja izhodna črka avtomata.**

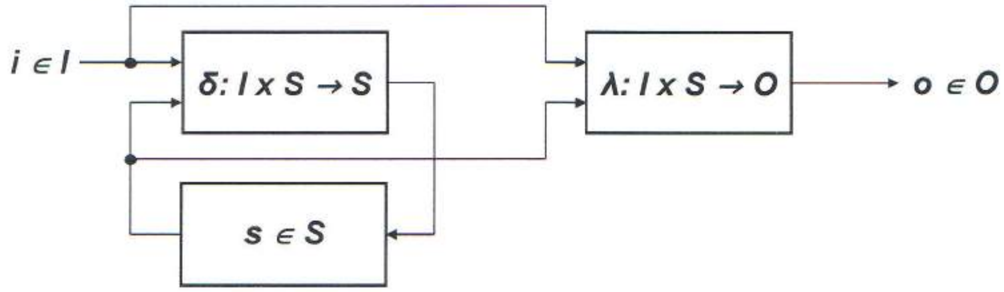
Končni avtomati – osnovne definicije

- **Končni avtomat tipa Moore ima v primerjavi z avtomatom tipa Mealy enostavnejšo izhodno funkcijo. Izhodna črka Moorovega avtomata je odvisna samo od notranje črke, torej od stanja, v katerem se avtomat nahaja, nič pa od vhodne črke.**
- **Če primerjamo definicijo končnega avtomata z modelom sinhronega sekvenčnega vezja, hitro ugotovimo, da predstavlja sinhrono sekvenčno vezje možno realizacijo nekega avtomata. Funkcija prehajanja stanj δ se realizira z vzbujevalnimi enačbami za vhode pomnilnih celic, izhodna funkcija λ pa na primarnih izhodih kombinacijske logike C.**
- **V definiciji obeh avtomatov je δ *časovna* preklopna funkcija, saj iz sedanjega stanja in sedanje vhodne črke izračuna <u>naslednje</u> stanje, λ pa je navadna preklopna funkcija, saj iz sedanjega stanja (in v primeru Mealyjevega avtomata tudi sedanje vhodne črke) izračuna <u>sedanjo</u> izhodno črko.**
- **Obnašanja Mealyjevih in Moorovih avtomatov ponavadi ne opisujemo eksplicitno s funkcijskima izrazoma za s^+ in o, pač pa s tabelami prehajanja stanj ali z diagrami prehajanja stanj na enak način, kot smo opisovali obnašanje sekvenčnih vezij tipa Mealy in Moore.**
- **Vsaka abeceda končnega avtomata (I, S in O) mora biti končna (po tem pridevniku se tudi imenujejo) in neprazna. Če bi imela, na primer, abeceda S neskončno mnogo črk, bi zahtevala realizacija avtomata neskončno velik pomnilnik, če pa bi bila prazna, bi se avtomat degeneriral v navadno kombinacijsko vezje.**

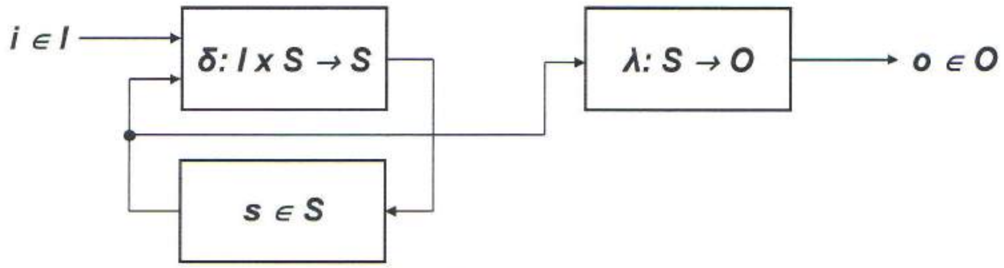
Končni avtomati – osnovne definicije

Mealyjev in Moorov avtomat si lahko ponazorimo z blokovno shemo.

Mealyjev avtomat



Moorov avtomat



Omenimo že zdaj, da lahko vsak Mealyjev avtomat pretvorimo v njemu ekvivalenten Moorov avtomat in obratno.

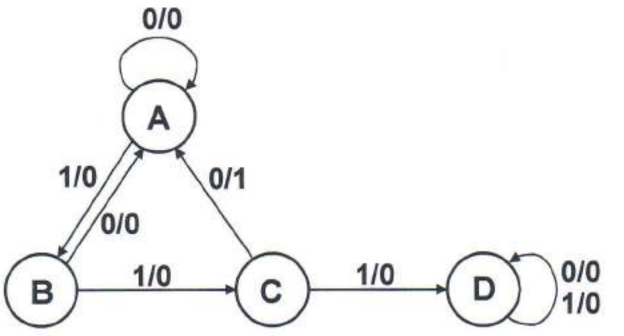
Končni avtomati – osnovne definicije

- ***Niz*** **(ali *beseda*) nad končno in neprazno abecedo je končno zaporedje (lahko tudi prazno) črk iz te abecede. Množica vseh nizov nad abecedo Σ poljubne dolžine je t. im. *Kleenejeva ovojnica* in jo označimo s Σ^*.**
- **Vhodne nize, v katerih si sledijo črke iz vhodne abecede *I*, označujemo z $\omega \in I^*$, izhodne nize, v katerih si sledijo črke iz izhodne abecede *O*, pa s $\varphi \in O^*$.**
- **Na končni avtomat lahko gledamo kot na *pretvornik* vhodnega niza v izhodnega. Končni avtomat lahko uporabimo za reševanje vseh tistih problemov, ki se dajo izraziti s pretvorbo nizov.**
- **Pomembna funkcija končnega avtomata je, da določi, ali je dani vhodni niz element neke vnaprej specificirane množice nizov. Avtomat izvrši to funkcijo tako, da sprejme tiste nize, ki so elementi množice in zavrne tiste, ki to niso. Avtomat, ki starta v svojem začetnem stanju, sprejme vhodni niz s postavitvijo izhoda na 1, ko sprejme zadnjo črko tega niza.**
- **Če velja $s^+ = \delta(i,s)$, *pravimo, da je stanje* s^+ *i-naslednik* stanja *s* (*pišemo* $s \xrightarrow{i} s^+$).**

  V splošnem, če vhodni niz ω *popelje avtomat iz stanja* s_i *v stanje* s_j, *pravimo, da je stanje* s_j *ω-naslednik* stanja s_i (*pišemo* $s_i \xrightarrow{\omega} s_j$).

Končni avtomati – osnovne definicije

Primer: **Imamo avtomat A s podanim diagramom prehajanja stanj.**



Abecede tega avtomata so naslednje množice:

I = {0,1} – vhodna abeceda,
***S = {A,B,C,D}* – notranja abeceda,**
***O = {0,1}* – izhodna abeceda.**

Predpostavimo, da je stanje A začetno stanje avtomata.
Iz diagrama vidimo na primer naslednje:

- **da je stanje B 1-naslednik stanja A, ker iz stanja A pridemo v stanje B, če damo na vhod črko 1,**
- **da je stanje D 111-naslednik stanja A, saj iz stanja A pridemo v stanje D, če na vhod avtomata damo vhodni niz 111,**
- **da avtomat vhodni niz ω = 110 pretvori v izhodni niz φ = 001,**
- **avtomat sprejme vse vhodne nize, ki se končajo s črko 0, pred katero sta dve črki 1 (npr. avtomat sprejme niza 110 in 0110, zavrne pa niz 01100).**

Končni avtomati – osnovne definicije

- **Neko stanje v avtomatu se imenuje *končno*, če**

  1. **iz njega ne vodi nobena povezava v kakšno drugo stanje (govorimo o *ponornem stanju*) ali**
  2. **vanj ne vodi nobena povezava iz kakšnega drugega stanja (govorimo o *izvornem stanju*).**

  Pojma izvorno stanje ne smemo zamenjati s pojmom začetno stanje, ki je stanje, v katerem avtomat začne delovati.

- **Če za vsak par stanj *si*, s_j avtomata A obstaja vhodni niz, ki popelje avtomat iz stanja s_i v stanje s_j, pravimo, da je A *krepko povezan*.**

- ***Primer:***
  - **V avtomatu A je stanje D ponorno stanje, saj ga avtomat ne more zapustiti, če se enkrat znajde v njem.**
  - **V avtomatu A ni nobenega izvornega stanja, saj v vsako stanje vodi vsaj ena povezava iz kakšnega drugega stanja.**

Ekvivalenca stanj in minimizacija avtomata

- Pri tvorbi diagrama (ali tabele) prehajanja stanj končnega avtomata se pogosto zgodi, da vsebuje diagram (ali tabela) redundantna stanja. Število pomnilnih celic, ki jih potrebujemo za realizacijo avtomata, je neposredno odvisno od števila stanj. Spomnimo se, da potrebujemo za avtomat s P stanji $p = \lceil \log_2 P \rceil$ spremenljivk stanja. Zaradi tega minimizacija števila stanj v mnogih primerih zmanjša kompleksnost in ceno realizacije.

- *k-ekvivalenca*
  Stanji s_i in s_j avtomata A sta *razločljivi*, če in samo če obstaja vsaj en končen vhodni niz, ki povzroči različna izhodna niza, v odvisnosti od tega, ali je začetno stanje s_i ali s_j.

  Niz, ki razloči stanji s_i in s_j, se imenuje *razločevalni niz* para (s_i, s_j). Če za *par* (s_i, s_j) obstaja razločevalni niz dolžine k, sta stanji v (s_i, s_j) *k-razločljivi.*

  Stanji, ki nista k-razločljivi, sta *k-ekvivalentni.*

  Stanji, ki sta k-ekvivalentni, sta tudi r-ekvivalentni za vse $r<k$.

  Stanji, ki sta k-ekvivalentni za vsak k, sta *ekvivalentni.*

Ekvivalenca stanj in minimizacija avtomata

- *Primer:* Za primer vzemimo par stanj (A,B) avtomata A_1.

Avtomat A_1

sedanje stanje	naslednje stanje/ sedanji izhod z — sedanji vhod x = 0	1
A	E/0	D/1
B	F/0	D/0
C	E/0	B/1
D	F/0	B/0
E	C/0	F/1
F	B/0	C/0

Stanji (A,B) sta 1-razločljivi, ker za i = 1 dobimo izhod 1, če je A_1 v začetnem stanju A, in izhod 0, če je A_1 v začetnem stanju B. Po drugi strani pa sta stanji (A,E) 3-razločljivi. Razločevalni niz je ω = 111, izhodna niza za začetni stanji A in E pa sta φ_A = 100 in φ_E = 101.

Definicija. Stanji s_i in s_j avtomata A sta *ekvivalentni*, če in samo če za vsak možen vhodni niz dobimo enak izhodni niz, ne glede na to, ali je začetno stanje s_i ali s_j. ■

Ekvivalenca stanj in minimizacija avtomata

Definicijo lahko posplošimo na primer, kjer je s_i možno začetno stanje v avtomatu A_1, medtem ko je s_j začetno stanje v avtomatu A_2, kjer imata oba avtomata isto vhodno abecedo.

Izrek. **Če sta stanji s_i in s_j avtomata A razločljivi, potem sta razločljivi z nizom dolžine *P-1* ali manj, kjer je *P* število stanj v A. ■**

Relacija ekvivalence je tranzitivna. Če velja $s_i = s_j$ in $s_j = s_k$, potem $s_i = s_k$. Iz tega sledi, da lahko množico stanj avtomata razdelimo v disjunktne podmnožice, imenovane *ekvivalenčni razredi*, tako da sta stanji v istem ekvivalenčnem razredu, če in samo če sta ekvivalentni, in sta v različnih razredih, če in samo če sta razločljivi.

Procedura za določitev množice ekvivalentnih stanj avtomata (to je ekvivalenčnih razredov) izhaja iz naslednje lastnosti. Če sta stanji s_i in s_j avtomata A ekvivalentni, sta njuna ω-naslednika, za vsak ω, tudi ekvivalentna.

Ekvivalenca stanj in minimizacija avtomata

Procedura minimizacije stanj

Opišimo proceduro za določitev množic ekvivalentnih stanj avtomata A. Želen rezultat je taka particija stanj avtomata A, da sta dve stanji v istem bloku, če in samo če sta ekvivalentni.

V prvem koraku razdelimo stanja avtomata A v bloke tako, da so vsa stanja v istem bloku 1-ekvivalentna. To izvedemo tako, da postavimo stanja, ki imajo identične izhodne črke pri vseh možnih vhodnih črkah v isti blok. Jasno je, da sta stanji, ki sta v različnih blokih, 1-razločljivi. Tako dobimo particijo π_1.

V naslednjem koraku zapišemo particijo π_2, katere bloki so množice stanj, ki so 2-ekvivalentna. V splošnem dobimo particijo π_{k+1} iz π_k tako, da postavimo v isti blok particije π_{k+1} tista stanja, ki so v istem bloku particije π_k *in* katerih i-nasledniki so za vsak $i \in I$ tudi v istem bloku particije π_k. Ta proces uvrsti v isti blok stanja, ki so (*k*+1)-ekvivalentna in v različni blok stanja, ki so (*k*+1)-razločljiva.

Če za neki *k* velja $\pi_{k+1} = \pi_k$, je procedura končana in π_k definira množice ekvivalentnih stanj avtomata. π_k je *ekvivalenčna particija*.

Izrek. Ekvivalenčna particija je edinstvena. ■

Ekvivalenca stanj in minimizacija avtomata

Primer: Poiščimo ekvivalenčno particijo avtomata A_1.

Avtomat A_1

sedanje stanje	naslednje stanje/ sedanji izhod z sedanji vhod x = 0	1
A	E/0	D/1
B	F/0	D/0
C	E/0	B/1
D	F/0	B/0
E	C/0	F/1
F	B/0	C/0

$\pi_1 = \{\overline{A,C,E}; \overline{B,D,F}\}$ – 1-ekvivalentna (izhod)

$\pi_2 = \{\overline{A,C,E}; \overline{B,D}; \overline{F}\}$ – 2-ekvivalentna

$\pi_3 = \{\overline{A,C}; \overline{E}; \overline{B,D}; \overline{F}\}$ – 3-ekvivalentna

$\pi_4 = \{\underset{a}{\overline{A,C}}; \underset{b}{\overline{E}}; \underset{c}{\overline{B,D}}; \underset{d}{\overline{F}}\}$ – 4-ekvivalentna

π_1 dobimo preprosto tako, da pregledamo tabelo in uvrstimo tista stanja, ki imajo pri vseh vhodnih črkah enake izhodne črke, v isti blok. Tako so stanja A, C in E v istem bloku, ker so njihove izhodne črke pri vhodni črki 0 vse enake 0, pri vhodni črki 1 pa vse enake 1. Zaradi podobnega razloga so stanja B, D in F uvrščena v drugi blok. π_1 tako vsebuje množici stanj, ki so 1-ekvivalentna.

Ekvivalenca stanj in minimizacija avtomata

Nadaljevanje primera

π_2 dobimo z razcepom blokov particije π_1 kadarkoli nasledniki stanj v bloku niso v istem bloku particije π_1. 0-naslednik bloka $\overline{A,C,E}$ je množica stanj {C,E}, 1-naslednik pa {B,D,F} in ker so stanja vsakega naslednika v skupnem bloku particije π_1, so stanja v $\overline{A,C,E}$ 2-ekvivalentna in zato $\overline{A,C,E}$ tvori blok v π_2. 1-naslednik bloka $\overline{B,D,F}$ je množica stanj {D,B,C}, toda ker ta stanja niso skupaj v bloku particije π_1, moramo blok $\overline{B,D,F}$ razdeliti na bloka $\overline{B,D}$ in $\overline{F}$. Na podoben način dobimo π_3 z razdelitvijo bloka $\overline{A,C,E}$ v π_2 na bloka $\overline{A,C}$ in $\overline{E}$, ker so 1-nasledniki stanj A, C in E stanja D, B in F, ki niso 2-ekvivalentna.

Poiščemo še π_4 in ugotovimo, da velja $\pi_4 = \pi_3$, zato je π_3 ekvivalenčna particija. Stanji A in C sta ekvivalentni, ekvivalentni sta tudi stanji B in D.

Ekvivalenca stanj in minimizacija avtomata

Ekvivalenca avtomatov

Izrek. Pravimo, da sta **avtomata A_1 in A_2 *ekvivalentna,* če in samo če za vsako stanje v A_1 obstaja ustrezno ekvivalentno stanje v A_2 in obratno. ■**

Ker je ekvivalenčna particija edinstvena, število blokov v ekvivalenčni particiji avtomata A definira *minimalno* število stanj, ki jih mora imeti katerikoli avtomat ekvivalenten avtomatu A. Avtomat, ki ne vsebuje ekvivalentnih stanj in je ekvivalenten A-ju, se imenuje *minimalna* ali *reducirana* oblika avtomata A.

Minimizacija avtomata A_1.
Bloke ekvivalenčne particije avtomata A_1 označimo z a, b, c in d in napišemo tabelo prehajanja stanj *minimalnega* ali *reduciranega* avtomata A_1, označenega z A_1*.

Ekvivalenčna particija avtomata A_1

$$\pi_3 = \{\underset{a}{\overline{A,C}};\ \underset{b}{\overline{E}};\ \underset{c}{\overline{B,D}};\ \underset{d}{\overline{F}}\}$$

Avtomat A_1*

sedanje stanje	naslednje stanje/ sedanji izhod z; sedanji vhod x = 0	1
a	b/0	c/1
b	a/0	d/1
c	d/0	c/0
d	c/0	a/0

Ekvivalenca stanj in minimizacija avtomata

Primer. **Dodatno ilustrirajmo postopek računanja ekvivalenčne particije na avtomatu A_2 in poiščimo njegovo minimalno obliko A_2*.**
Pri minimizaciji si lahko pomagamo tako, da v particiji π_k v blokih, ki vsebujejo več kot eno stanje, pod vsakim stanjem za vsako vhodno črko *i* $\in I$ napišemo zaporedno številko bloka v particiji π_k, v katerem se nahaja *i*-naslednik stanja. Stanja v istem bloku particije π_k, ki so označena z različnimi številkami, moramo v particiji π_{k+1} razdeliti v različne bloke.

Avtomat A_2

sedanje stanje	naslednje stanje/ sedanji izhod z; sedanji vhod x = 0	1
A	E/0	C/0
B	C/0	A/0
C	B/0	G/0
D	G/0	A/0
E	F/1	B/0
F	E/0	D/0
G	D/0	G/0

$$\pi_1 = \{\overline{\underset{2,1}{A}, \underset{1,1}{B}, \underset{1,1}{C}, \underset{1,1}{D}, \underset{2,1}{F}, \underset{1,1}{G}};\ \overline{E}\}$$

$$\pi_2 = \{\overline{\underset{3,2}{A}, \underset{3,2}{F}};\ \overline{\underset{2,1}{B}, \underset{2,2}{C}, \underset{2,1}{D}, \underset{2,2}{G}};\ \overline{E}\}$$

$$\pi_3 = \{\overline{\underset{4,3}{A}, \underset{4,2}{F}};\ \overline{\underset{3,1}{B}, \underset{3,1}{D}};\ \overline{\underset{2,3}{C}, \underset{2,3}{G}};\ \overline{E}\}$$

$$\pi_4 = \{\overline{A};\ \overline{F};\ \overline{\underset{4,1}{B}, \underset{4,1}{D}};\ \overline{\underset{3,4}{C}, \underset{3,4}{G}};\ \overline{E}\}$$

$$\pi_5 = \{\underset{a}{\overline{A}};\ \underset{b}{\overline{F}};\ \underset{c}{\overline{B, D}};\ \underset{d}{\overline{C, G}};\ \underset{e}{\overline{E}}\}$$

Avtomat A_2*

sedanje stanje	naslednje stanje/ sedanji izhod z; sedanji vhod x = 0	1
a	e/0	d/0
c	d/0	a/0
d	c/0	d/0
e	b/1	c/0
b	e/0	c/0

Ekvivalenca stanj in minimizacija avtomata

Nepopolno specificirani končni avtomati

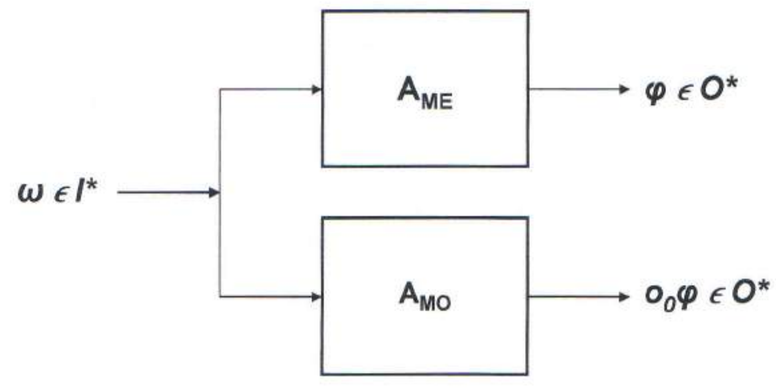
V praksi se pogosto dogaja, da različne kombinacije stanj in vhodov niso možne. V drugih primerih so prehodi stanj popolnoma definirani, toda za nekatere kombinacije stanj in vhodov vrednosti izhodov niso pomembne, zato jih ne specificiramo. Takim avtomatom pravimo *nepopolno specificirani* avtomati. Pri nepopolno specificiranih avtomatih je analogija k relaciji ekvivalence med stanji in avtomati, ki smo jo definirali pri popolno definiranih avtomatih, relacija *kompatibilnosti*.

Definicija. **Stanji s_i in s_j avtomata A sta *kompatibilni*, če in samo če je za vsak vhodni niz, ki ga pripeljemo tako na s_i kot na s_j, proizveden enak izhodni niz, *kadarkoli sta oba izhoda specificirana*, ne glede na to, ali je začetno stanje s_i ali s_j. ■**

Minimalne oblike nepopolno specificiranih avtomatov *niso* edinstvene.

Pretvorba avtomatov iz tipa Moore v Mealy in obratno

Končna avtomata istega tipa sta ekvivalentna, če proizvedeta pri enakih vhodnih nizih enaka izhodna niza. V primeru avtomatov različnih tipov (eden je tipa Mealy drugi pa tipa Moore) pa ekvivalenca ni tako stroga. Omenjena avtomata sta ekvivalentna, če imata pri enakih vhodnih nizih enake izhodne nize, pri tem pa prva črka (o_0) na izhodu Moorovega avtomata ne šteje (ni pomembna), saj se pojavi že takrat, ko na vhod še ni prišla nobena vhodna črka.



Pretvorba Moorovega avtomata v ekvivalenten Mealyjev avtomat

Pretvorba Moorovega avtomata A_{MO} v ekvivalenten Mealyjev avtomat A_{ME} je preprostejša, saj pretvarjamo manj splošen avtomat v bolj splošnega. Stanja in prehodi v A_{ME} so enaki onim v A_{MO}. Kadar ima stanje *s* v A_{MO} izhodno črko *o*, je vsak prehod v A_{ME} *v stanje s* označen z izhodno črko *o*. Izpeljava je naslednja:

Če je podan Moorov avtomat $A_{MO} = (I, S, O, \delta, \lambda_{MO}, s_0)$,

je njemu ekvivalenten Mealyjev avtomat $A_{ME} = (I, S, O, \delta, \lambda_{ME}, s_0)$,

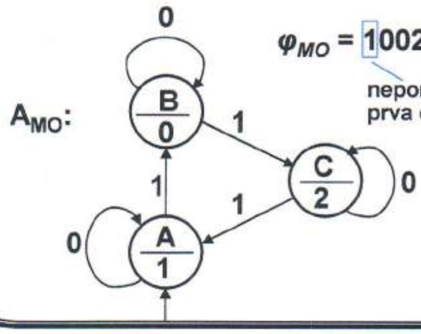
kjer je $\lambda_{ME}(i,s) = \lambda_{MO}(\delta(i,s)), \forall i \in I, s \in S$.

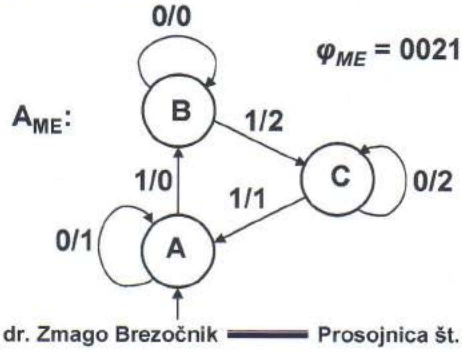
Primer: **Pretvorite narisan Moorov avtomat v Mealyjevega. Izračunajte izhodna niza φ_{MO} in φ_{ME}, če je vhodni niz $\omega = 10110$.**

$\varphi_{MO} = \boxed{1}00211$

nepomembna prva črka

$\varphi_{ME} = 00211$





Pretvorba Mealyjevega avtomata v ekvivalenten Moorov avtomat

Pretvorba Mealyjevega avtomata A_{ME} v ekvivalenten Moorov avtomat A_{MO} je nekoliko zahtevnejša, saj pretvarjamo bolj splošen avtomat v manj splošnega. Zgornje izpeljave ne moremo kar preprosto obrniti, ker lahko A_{ME} vsebuje stanje *s*, katerega vhodni prehodi so označeni z več kot eno izhodno črko. Da bi obšli to težavo, označimo stanja v avtomatu A_{MO} z množico vseh parov stanje-izhod $S_{ME} \times O$ v avtomatu A_{ME}. Avtomat A_{MO} bo vstopil v stanje (*s*,*o*), kadarkoli A_{ME} vstopi v stanje *s* in generira izhodno črko *o*. Izpeljava je naslednja:

Če je podan Mealyjev avtomat $A_{ME} = (I, S_{ME}, O, \delta_{ME}, \lambda_{ME}, s_0)$,

je njemu ekvivalenten Moorov avtomat $A_{MO} = (I, S_{MO}, O, \delta_{MO}, \lambda_{MO}, (s_0, o_0))$,

kjer je $S_{MO} = S_{ME} \times O$. Funkciji δ_{MO} in λ_{MO} sta definirani na naslednji način.

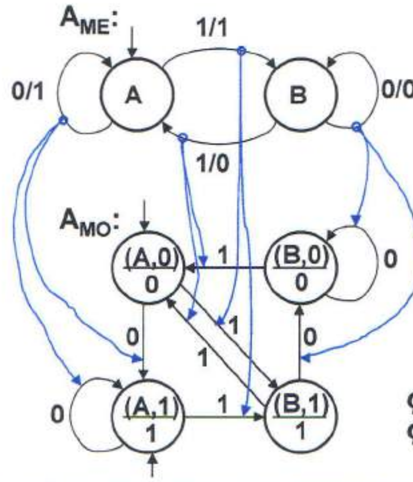
Pretvorba Mealyjevega avtomata v ekvivalenten Moorov avtomat

Prehod $s \xrightarrow{i/o} s'$ **v** A_{ME} **se preslika v prehode** $\frac{(s,o')}{o'} \xrightarrow{i} \frac{(s',o)}{o}$ **v** A_{MO}
za $\forall o' \in O$.

Prehod $s_0 \xrightarrow{i/o} s'$ **v** A_{ME} **se preslika v prehode** $\frac{(s_0,o_0)}{o_0} \xrightarrow{i} \frac{(s',o)}{o}$ **v** A_{MO},
kjer je $o_0 \in O$ **poljubno izbrana izhodna črka za začetno stanje avtomata** A_{MO}.

Primer: **Pretvorite narisan Mealyjev avtomat v Moorovega. Izračunajte izhodni niz za** A_{ME} **in** A_{MO}**, če je vhodni niz** $\omega = 10110$.

Začetno stanje v A_{MO} **je lahko (A,0) ali (A,1).**



$\varphi_{ME} = 10010$

nepomembna prva črka

$\varphi_{MO0} = \boxed{0}10010$
$\varphi_{MO1} = \boxed{1}10010$

Računanje s particijami

Vzemimo množico $S=\{s_1, s_2, \ldots, s_r\}$. **Iz elementov te množice je mogoče tvoriti** 2^r **različnih podmnožic. Če med njimi izberemo takšne podmnožice** $B_1, B_2, \ldots, B_q$, **da za njih velja**

$$B_1 \cup B_2 \cup \ldots \cup B_q = S$$
$$B_j \cap B_k = \emptyset \text{ za } \forall j,k \in \{1,2,\ldots,q\} \wedge j \neq k,$$

imenujemo množico $\pi = \{B_1, B_2, \ldots, B_q\}$ ***particija.*** **Elementi** B_i **te množice so bloki. Pri** q **blokih imamo** q**-bločno particijo.**

Definicija. **Če sta** π_1 **in** π_2 **particiji na množici S, je "*produkt*"** $\pi = \pi_1 \cdot \pi_2$ **particija na** S**, ki jo dobimo s presekom blokov iz** π_1 **s tistimi iz** π_2:

$$B = B_j \cap B_k,\ B \in \pi,\ B_j \in \pi_1,\ B_k \in \pi_2\ . \blacksquare$$

Definicija. **Če sta** π_1 **in** π_2 **particiji na množici S, je "vsota"** $\pi = \pi_1 + \pi_2$ **particija na** S**, v kateri so bloki najmanjše podmnožice množice** S**, ki vsebujejo vse bloke** B_j **in** B_k**, kjer velja**

$$B_j \cap B = B_j \text{ ali } B_j \cap B = \emptyset,$$
$$B_k \cap B = B_k \text{ ali } B_k \cap B = \emptyset,\ B \in \pi,\ B_j \in \pi_1,\ B_k \in \pi_2\ . \blacksquare$$

V vsoti π ***so torej združeni bloki particij*** π_1 ***in*** π_2, **ki imajo vsaj en skupni element.**

Računanje s particijami

Obstajata dve značilni particiji: *particija enote* π_E in *particija nič* π_0. π_E ima v enem bloku vse elemente množice *S*, π_0 pa ima za vsak element $s \in S$ *svoj blok.*

Če je π poljubna particija na množici *S*, *velja:*

$$\pi_0 \cdot \pi = \pi_0 \qquad \pi_E \cdot \pi = \pi$$
$$\pi_0 + \pi = \pi \qquad \pi_E + \pi = \pi_E$$

***Primer:* Dana je množica $S = \{1,2,3,4,5,6,7,8\}$ in particiji**

$\pi_1 = \{\overline{1};\overline{2,3,4};\overline{5,6,7};\overline{8}\}$,

$\pi_2 = \{\overline{1,2};\overline{3,4};\overline{5,7};\overline{6};\overline{8}\}$.

Izračunajmo particiji $\pi_a = \pi_1 \cdot \pi_2$ *in* $\pi_b = \pi_1 + \pi_2$. Zapišimo še π_E *in* π_0.

$\pi_a = \pi_1 \cdot \pi_2 = \{\overline{1};\overline{2};\overline{3,4};\overline{5,7};\overline{6};\overline{8}\}$ $\qquad$ $\pi_E = \{\overline{1,2,3,4,5,6,7,8}\}$

$\pi_b = \pi_1 + \pi_2 = \{\overline{1,2,3,4};\overline{5,6,7};\overline{8}\}$ $\qquad$ $\pi_0 = \{\overline{1};\overline{2};\overline{3};\overline{4};\overline{5};\overline{6};\overline{7};\overline{8}\}$

***Definicija.* Bodita π_1 in π_2 particiji na množici *S*. Particija π_1 *je vsebovana v* particiji π_2 (pišemo $\pi_1 \leq \pi_2$), če in samo če so vsa stanja v bloku π_1 tudi tudi skupaj v bloku π_2 (pravimo tudi, da je π_1 "manjša ali enaka" π_2). ■**

Računanje s particijami

***Definicija.* Particija $\pi = \{B_1, B_2, \ldots, B_q\}$ na množici stanj *S* končnega avtomata $A = (I, S, O, \delta, \lambda, s_0)$, ima *substitucijsko značilnost*, če se stanja v kateremkoli bloku B_j za vse vhodne črke $i \in I$ preslikajo v isti blok B_k:**

$$\delta(i,B_j) \in B_k, \qquad B_j, B_k \in \pi. \ ■$$

Sinonim za particijo s substitucijsko značilnostjo je *zaprta particija*.

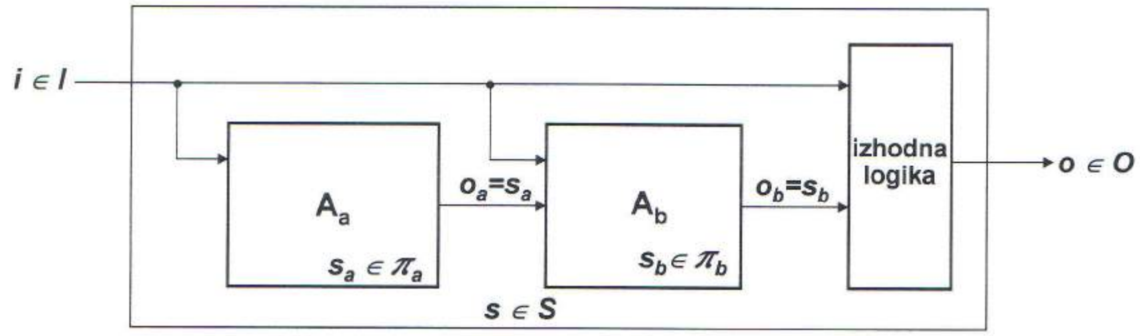
***Izrek.* Particija π s substitucijsko značilnostjo, ki ima za vsako stanje v bloku za vse vhodne črke enako izhodno črko, je ekvivalenčna particija, to pomeni, da so bloki v particiji π množice med seboj ekvivalentnih stanj. ■**

Sedaj se lotimo problema dekompozicije avtomata. Ideja je naslednja. Namesto da bi realizirali originalni obsežni avtomat, ga najprej dekomponiramo v manjše avtomate, vsakega posebej realiziramo in jih med seboj povežemo. Vsi delujejo sočasno in imajo navzven enako obnašanje kot originalni avtomat.

Osnovna tipa dekompozicije končnih avtomatov sta *serijska* in *paralelna* dekompozicija.

Serijska dekompozicija končnega avtomata

Serijska dekompozicija končnega avtomata A na avtomata A_a in A_b je možna le v primeru, če za množico stanj *S* avtomata A najdemo particiji *π_a in π_b, tako da velja $\pi_a \cdot \pi_b$* = π_0 in ima vsaj ena od partij π_a, π_b substitucijsko značilnost.



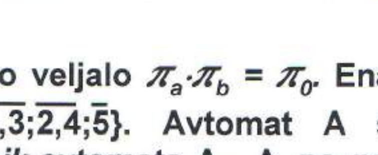
Primer. **Podan je Moorov avtomat A s tabelo prehajanja stanj. Ali obstaja pri particiji π_a = $\{\overline{1,2};\overline{3,4,5}\}$ serijska dekompozicija? Če obstaja, jo izvedite.**

Serijska dekompozicija končnega avtomata

Avtomat A:

sedanje stanje	naslednje stanje sedanji vhod i_1	naslednje stanje sedanji vhod i_2	izhod
1	5	3	o_2
2	3	4	o_0
3	1	5	o_1
4	2	3	o_2
5	1	4	o_2

Preverimo substitucijsko značilnost particije π_a.



Particija π_a ima substitucijsko značilnost, zato omogoča serijsko dekompozicijo.

Poiskati moramo tako particijo π_b, da bo veljalo $\pi_a \cdot \pi_b = \pi_0$. Ena izmed možnih takšnih particij je π_b = $\{\overline{1,3};\overline{2,4};\overline{5}\}$. Avtomat A serijsko dekomponiramo v A_a in A_b. A_a bo *prednik* avtomata A_b, A_b pa *naslednik* avtomata A_a. Avtomata A_a in A_b delata sočasno. Avtomat A_a dela čisto neodvisno od A_b. A_b je odvisen od A_a, saj je izhod A_a (enak je kar stanju) povezan na vhod avtomata A_b.

Notranja abeceda avtomata A_a je particija π_a,notranja abeceda drugega avtomata pa particija π_b. Označimo si bloke v avtomatih A_a in A_b:

Serijska dekompozicija končnega avtomata

A_a: $\pi_a = \{\overline{1,2};\overline{3,4,5}\} = \{A,B\}$ A_b: $\pi_b = \{\overline{1,3};\overline{2,4};\overline{5}\} = \{C,D,E\}$

Avtomat A_a:

sedanje stanje	naslednje stanje, sedanji vhod i_1	i_2
A	*B*	*B*
B	*A*	*B*

Avtomat A_b:

sedanje stanje	naslednje stanje/izhod, sedanji vhod Ai_1	Ai_2	Bi_1	Bi_2
C	E/o_2	C/o_2	C/o_1	E/o_1
D	C/o_0	D/o_0	D/o_2	C/o_2
E	–	–	C/o_2	D/o_2

Izhodna črka iz avtomata A_a je kar njena notranja črka, torej $o_a = s_a$.

V avtomat A_b vstopa tako vhodna črka (i_1 ali i_2) kot tudi stanje (*A* ali *B*) njegovega predhodnika – avtomata A_a. To je še zmeraj Moorov avtomat, saj izhodne črke $o \in \{o_0,o_1,o_2\}$ niso odvisne od vhodnih črk $i \in \{i_1,i_2\}$, ampak le od stanj avtomatov A_a in A_b. Poglejmo, kako za A_b izračunamo naslednje stanje v levem zgornjem delu tabele:

- vhod v avtomat A_b: Ai_1 (*A* je stanje avtomata A_a, i_1 je vhodna črka),
- stanje avtomata A_b: *C*,
- naslednje stanje: $\delta(i_1, A \cap C) = \delta(i_1,\{1,2\} \cap \{1,3\}) = \delta(i_1,\{1\}) = \{5\} = \{E\}$. Presek stanja avtomata A_a in stanja avtomata A_b je stanje originalnega avtomata A.

Serijska dekompozicija končnega avtomata

Do izhodne logike pridemo tako, da najprej tvorimo preseke med vsemi možnimi stanji avtomatov A_a in A_b in tako dobimo stanja originalnega avtomata A:

$A \cap C = \{1,2\} \cap \{1,3\} = \{1\}$ ... o_2
$A \cap D = \{1,2\} \cap \{2,4\} = \{2\}$ ... o_0
$A \cap E = \{1,2\} \cap \{5\} = \emptyset$
$B \cap C = \{3,4,5\} \cap \{1,3\} = \{3\}$ ... o_1
$B \cap D = \{3,4,5\} \cap \{2,4\} = \{4\}$ ... o_2
$B \cap E = \{3,4,5\} \cap \{5\} = \{5\}$ ... o_2

izhodne črke avtomata A

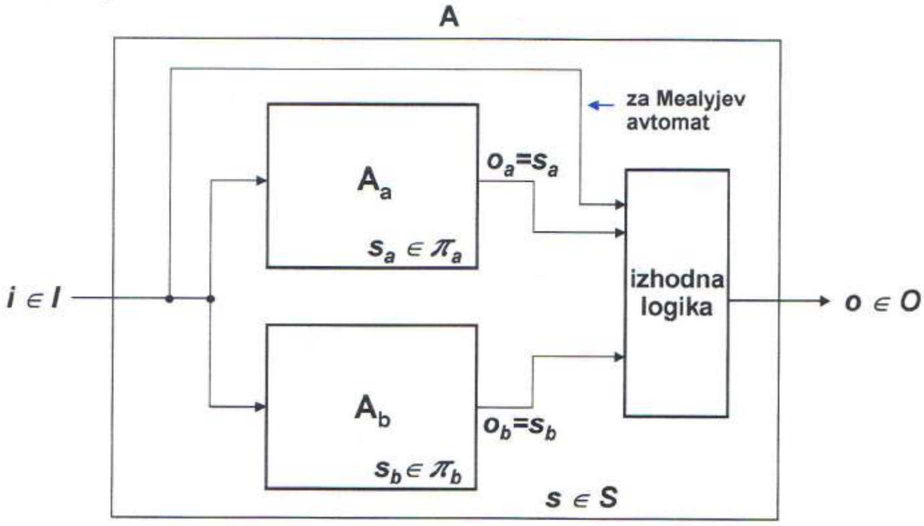
Izraze za izhodne črke dobimo na osnovi stanj avtomata A_a in A_b:

$o_0 = A \cap D$, $o_1 = B \cap C$, $o_2 = (A \cap C) \cup (B \cap D) \cup (B \cap E)$.

Če želimo avtomata A_a in A_b realizirati s sinhronim sekvenčnim vezjem, moramo zakodirati vhodne, notranje in izhodne črke. Izbrali bi lahko na primer naslednje kode:
$I = \{i_1,i_2\} = \{0,1\}$ – en vhod (npr. x),
$O = \{o_0,o_1,o_2\} = \{00,01,10\}$ – dva izhoda (npr. z_1 in z_2),
$\pi_a = \{A,B\} = \{0,1\}$ – ena spremenljivka stanja (npr. y_a),
$\pi_b = \{C,D,E\} = \{00,01,10\}$ – dve spremenljivki stanja (npr. y_b in y_c).

Paralelna dekompozicija končnega avtomata

Paralelna dekompozicija končnega avtomata A na avtomata A_a in A_b je možna le v primeru, če za množico stanj *S* avtomata A najdemo particiji π_a *in* π_b*, tako da velja* $\pi_a \cdot \pi_b = \pi_0$ in ima tako particija π_a *kot tudi particija* π_b substitucijsko značilnost.



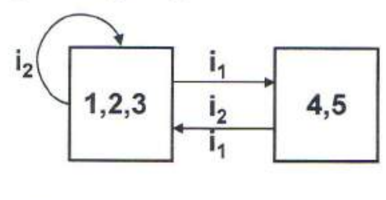
Primer. Podan je Moorov avtomat A s tabelo prehajanja stanj. Izvedite paralelno dekompozicijo avtomata pri particijah $\pi_a = \{\overline{1,2,3};\overline{4,5}\}$ in $\pi_b = \{\overline{1,4};\overline{2,5};\overline{3}\}$.

Paralelna dekompozicija končnega avtomata

Avtomat A:

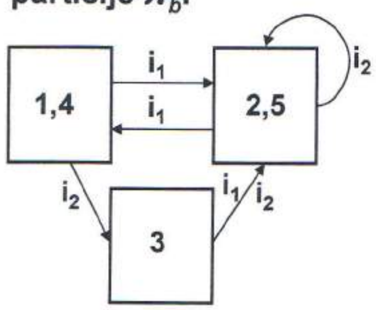
sedanje stanje	naslednje stanje, sedanji vhod i_1	naslednje stanje, sedanji vhod i_2	izhod
1	5	3	o_1
2	4	2	o_1
3	5	2	o_1
4	2	3	o_2
5	1	2	o_1

Preverimo substitucijsko značilnost particije π_a.



Particija π_a ima substitucijsko značilnost.

Preverimo še substitucijsko značilnost particije π_b.



Particija π_b ima substitucijsko značilnost.

Ker velja $\pi_a \cdot \pi_b = \pi_0$, je paralelna dekompozicija pri particijah π_a in π_b možna. Označimo si bloke v avtomatih A_a in A_b:

Paralelna dekompozicija končnega avtomata

A_a: $\pi_a = \{\overline{1,2,3};\overline{4,5}\} = \{A,B\}$

A_b: $\pi_b = \{\overline{1,4};\overline{2,5};\overline{3}\} = \{C,D,E\}$

Avtomat A_a:

sedanje stanje	naslednje stanje sedanji vhod i_1	 i_2
A	*B*	*A*
B	*A*	*A*

Avtomat A_b:

sedanje stanje	naslednje stanje sedanji vhod i_1	 i_2
C	*D*	*E*
D	*C*	*D*
E	*D*	*D*

Do izhodne logike pridemo tako, da najprej tvorimo preseke med vsemi možnimi stanji avtomatov A_a in A_b in tako dobimo stanja originalnega avtomata A:

$A \cap C = \{1,2,3\} \cap \{1,4\} = \{1\}$ ... o_1
$A \cap D = \{1,2,3\} \cap \{2,5\} = \{2\}$ ... o_1
$A \cap E = \{1,2,3\} \cap \{3\} = \{3\}$ ... o_1
$B \cap C = \{4,5\} \cap \{1,4\} = \{4\}$ ... o_2
$B \cap D = \{4,5\} \cap \{2,5\} = \{5\}$ ... o_1
$B \cap E = \{4,5\} \cap \{3\} = \emptyset$

izhodne črke avtomata A

Paralelna dekompozicija končnega avtomata

Izraze za izhodne črke dobimo na osnovi stanj (notranjih črk) avtomata A_a in A_b:

$o_1 = (A \cap C) \cup (A \cap D) \cup (A \cap E) \cup (B \cap D)$
$o_2 = B \cap C$

Če želimo avtomata A_a in A_b realizirati s sinhronim sekvenčnim vezjem, moramo zakodirati vhodne, notranje in izhodne črke. Izbrali bi lahko na primer naslednje kode:
$I = \{i_1,i_2\} = \{0,1\}$ – en vhod (npr. x),
$O = \{o_1,o_2\} = \{0,1\}$ – en izhod (npr. z),
$\pi_a = \{A,B\} = \{0,1\}$ – ena spremenljivka stanja (npr. y_a),
$\pi_b = \{C,D,E\} = \{00,01,10\}$ – dve spremenljivki stanja (npr. y_b in y_c).

Iskanje particij s substitucijsko značilnostjo

sedanje stanje	naslednje stanje		
	sedanji vhod		
	i_1	i_2	i_3
1	6	3	2
2	5	4	1
3	2	5	4
4	1	6	3
5	4	1	6
6	3	2	5

Za vsak par stanj s_i in s_j izračunamo najmanjšo particijo s substitucijsko značilnostjo, ki vsebuje stanji s_i in s_j v istem bloku – dobimo t. im. *osnovne* particije. Izračunamo tudi vse možne vsote osnovnih particij, ki so manjše od π_E. Tako dobimo vse netrivialne particije s substitucijsko značilnostjo.

i_1: (1,2) → (6,5) → (3,4) → (2,1)
i_2: (1,2) → (3,4) → (5,6) → (1,2)
i_3: (1,2) → (2,1)
$\pi_1 = \{\overline{1,2};\overline{3,4};\overline{5,6}\}$ √

i_1: (1,3) → (6,2) → (3,5) → (2,4) → (5,1) → (4,6) → (1,3)
i_2: (1,3) → (3,5) → (5,1) → (1,3)
i_3: (1,3) → (2,4) → (1,3)
$\pi_2 = \{\overline{1,3,5};\overline{2,4,6}\}$ √

i_1: (1,4) → (6,1) → (3,6) → (2,3) → (5,2) → (4,5) → (1,4)
i_2: (1,4) → (3,6) → (5,2) → (1,4)
i_3: (1,4) → (2,3) → (1,4)
$\pi_E = \{\overline{1,2,3,4,5,6}\}$

i_1: (1,5) → (6,4) → (3,1) → (2,6) → (5,3) → (4,2) → (1,5)
i_2: (1,5) → (3,1) → (5,3) → (1,5)
i_3: (1,5) → (2,6) → (1,5)
$\pi_2 = \{\overline{1,3,5};\overline{2,4,6}\}$

i_1: (1,6) → (6,3) → (3,2) → (2,5) → (5,4) → (4,1) → (1,6)
i_2: (1,6) → (3,2) → (5,4) → (1,6)
i_3: (1,6) → (2,5) → (1,6)
$\pi_E = \{\overline{1,2,3,4,5,6}\}$

...

Povzetek poglavja

Predstavili smo:

- Osnovne definicije končnih avtomatov
- Postopek minimizacije končnega avtomata
- Pretvorbo med končnimi avtomati različnih tipov
- Dekompozicijo končnih avtomatov