DIGITALNA TEHNIKA 2017/2018 (nazadnje spremenjeno 26.10.2017)

SEMINARSKE VAJE – 15 ur

1. Številski sistemi (vadite sami doma)
2. Boolova algebra, Huntingtonovi postulati (27. 10. 2017)
3. Kanonična oblika Boolovih funkcij, PDNO, PKNO (27. 10. 2017)
4. Minimizacija Boolovih funkcij, MDNO, MKNO (3. 11. 2017)
5. Kombinacijska vezja s Shefferjevimi in Peircovimi elementi (7. 11. 2017)
6. Priprave na prvi kolokvij
7. Računalniško modeliranje in simulacija vezij (1. 12. 2017)
8. Multipleksor (4. 12. 2017)
9. Pomnilne celice, analiza sekvenčnih vezij (4. 1. 2018)
10. Sinteza sekvenčnih vezij (4. 1. 2018)
11. Mealyjevi in Moorovi avtomati, izvedba s sekvenčnim vezjem (9. 1. 2018)
12. Pretvorba avtomatov, minimizacija avtomatov (16. 1. 2018)
13. Priprave na drugi kolokvij

Dodatna poglavja, ki jih letos ne bomo obravnavali

D1. Računalniška predstavitev in obdelava Boolovih funkcij
D2. Statični hazardi

LABORATORIJSKE VAJE – 15 ur
(v oklepaju je naveden okvirni termin)

1. Enostavna kombinacijska vezja s Shefferjevimi in Peircovimi elementi (november)
2. Zahtevnejša kombinacijska vezja s Shefferjevimi in Peircovimi elementi (november)
3. Kombinacijska vezja z multipleksorji (december)
4. Sekvenčna vezja (januar)
5. Izvedba avtomatov s sekvenčnim vezjem (januar)

OPOMBA K GRADIVU

Pričujoče gradivo predstavlja asistentove zapiske za vaje in ni učbenik. Uporabljajte knjige, ki jih profesor navede kot vire za učenje.

Gradivo je prosto dostopno vsem, tako da če koga zanima, mu dajte povezavo na mojo stran. Gradivo bom še dopolnjeval, zato naj stare različice ne krožijo naokoli.

Pričujoče gradivo je v beta verziji, zato so v njem možne napake. Če kakšno najdete, me prosim obvestite na robert.meolic@um.si. Hvala.

Robert Meolic

0. Uvod

Nemški matematik **Gottfried Wilhelm Leibniz** (1646 – 1716) je bil eden prvih, ki je razmišljal binarnem sistemu. A preteči sta morali še dve stoletji, da je ta ideja postala uporabna.

Anglež **George Boole** je živel v 19. stoletju (1815 – 1864) in je bil pravi mali genij: pri 16-ih letih je poleg angleščine tekoče govoril nemško, italijansko in francosko in je že postal asistent na šoli. Ko je bil star 20 let, je ustanovil svojo šolo. Ko je bil star 24 let je izdal svoj prvi znanstveni članek v matematični reviji na univerzi v Cambridgeu. Leta 1847 je napisal razpravo, v kateri je logiko predstavil kot del matematike in si s tem pridobil svetovni sloves. Njegovo delo je pohvalil tudi 10 let starejši anglež **Augustus de Morgan**, ki je bil v tistem času eden najpomembnejših znanstvenikov na področju logike in je istočasno napisal svojo razpravo, ki pa ni bila tako dobra. Glavna Boolova ideja je bila v tem, da je kot osnovo logike vzel dvojiški sistem in operatorje AND, OR in NOT, kar se uporablja še danes. Svoje delo je natančneje opisal v knjigi leta 1854. Kljub vsem priznanjem, pa drugi raziskovalci tistega časa niso videli uporabne vrednosti njegove ideje. Šele 12 let pozneje, ko je bil Boole žal že mrtev, je 25 let mlajši američan **Charles Sanders Peirce** začel nadaljevati njegovo delo. Prvi sistem aksiomov za Boolovo algebro je predstavil američan **Edward Vermilye Huntington** leta 1904.

Kljub temu, da je že v Boolovem času anglež **Charles Babbage** hotel narediti prvi računalnik, je šele leta 1937 američan **Claude Shannon** (1916 – 2001) uporabil Boolovo algebro za načrtovanje električnih/elektronskih vezij in računalnikov. Američana **John Vincent Atanasoff** (1905 – 1995) in **Clifford Edward Berry** (1918 – 1963) sta leta 1942 zgradila prvi stroj, ki je bil podoben današnjim računalnikom, a ga nisi mogel programirati. Njuno idejo sta nadgradila američana **John Mauchly** (1907 – 1980) in **John Adam Presper Eckert Jr.** (1919 – 1995), ki sta 14. februarja 1946 na University of Pennsylvania predstavila ENIAC (Electronic Numerical Integrator And Computer). ENIAC velja za prvi računalnik na svetu, velik je bil 2.6 x 0.9 x 26 m, tehtal je 27 ton in porabil 150 kW elektrike. Zmogel je 5000 instrukcij na sekundo in je zaradi vsakodnevnih okvar deloval le približno pol časa.

Potem, ko se je Boolova algebra izkazala kot odlična osnova za načrtovanja računalnikov, so se razikovalci posvetili predvsem problemu kompleksnosti načrtovanja. Ročno metodo minimizacije Boolove funkcije s pomočjo diagramov si je leta 1952 izmislil američan **Edward W. Veitch**. Leta 1953 je američan **Maurice Karnaugh** njegovo metodo nekoliko preoblikoval in njegova različica se danes v praksi najpogosteje uporablja. Leta 1956 sta američana **Willard Van Orman Quine** in **Edward J. McCluskey** predstavila prvi računalniški algoritem za minimizacijo Boolovih funkcij. Leta 1978 je američan **Sheldon B. Akers** objavil odmeven članek o uporabi podatkovne strukture BDD (binarni odločitveni grafi), s katerimi lahko Boolove funkcije v računalniku predstavimo mnogo bolje kot s tabelami. Leta 1990 so američani **Karl S. Brace**, **Richard L. Rudell** in **Randal E. Bryant** objavili prve učinkovite algoritme za BDD-je.

1. Številski sistemi

Obravnavali bomo pretvorbo med desetiškim, dvojiškim in šestnajstiškim sistemom.

VAJA 1.1
Pretvori $+52_{(10)}$ in $-52_{(10)}$ v dvojiški in šestnajstiški sistem.

$+52_{(10)} = 00110100_{(2)} = 34_{(16)}$
$-52_{(10)} = 11001100_{(2)} = CC_{(16)}$

VAJA 1.2
Predpostavimo osembitna predznačena števila in uporabo dvojiškega komplementa. Določite predstavitev desetiškega števila -26 tako, da zapišete desetiški števili -128 in +102 in ju nato seštejete.

Rešite sami.

VAJA 1.3
Zapiši mešano število $52{,}812_{(10)}$ v dvojiškem sistemu tako, da bo celi del predstavljen z 8 biti, ulomljeni del pa s 4 biti.

$52{,}812_{(10)} = 00110100{,}1101_{(2)}$

VAJA 1.4
Zapiši števila +1, +2, +2,5 in +331,3 v dvojiškem sistemu po standardu IEEE 754.

Ta naloga zahteva, da realno število predstavimo s plavajočo vejico. Po standardu IEEE 754 je zapis dolg 32 bitov in je sestavljen iz treh delov:

- prvi bit je **predznak**: 0 pomeni plus, 1 pomeni minus,
- sledi 8 bitov, ki pomenijo **eksponent** povečan za 127,
- sledi 23 bitov, ki pomenijo **mantiso**, to je ulomljeni del (celi del je vedno 1).

`+1 =` $0\ 01111111\ 00000000000000000000000\ (1 = 1{,}0 * 2^{0})$

`+2 =` $0\ 10000000\ 00000000000000000000000\ (2 = 1{,}0 * 2^{1})$

`+2,5 =` $0\ 10000000\ 01000000000000000000000\ (2{,}5 = 1{,}01 * 2^{1})$

`+331,3 =` $0\ 10000111\ 01001011010011001100110\ (331{,}3 = 1{,}01001... * 2^{8})$

Če zadnje število zaokrožimo na 16 mest v mantisi, dobimo:

`+331,3 =` $0\ 10000111\ 01001011010011010000000\ (= 331{,}30078_{10})$

Zaokroževanje v binarnem svetu je vedno navzgor, torej `000=00`, `001=01`, `010=01`, `011=10`, `100=10`, `101=11` in `110=11`. Število `111` se zaokroži s prenosom naprej, podobno kot se v desetiškem svetu število 99,6 zaokroži na 100. V zgornjem primeru smo `01100110` zaokrožili na `10000000`.

2. Boolova algebra, Huntingtonovi postulati

Moderna matematika temelji na aksiomih. Prvi sistem aksiomov za Boolovo algebro je predstavil američan **Edward Vermilye Huntington** leta 1904. Pozneje so našli še veliko drugih načinov aksiomatizacije Boolove algebre, enega zelo pomembnega je leta 1933 predstavil sam Huntington. A njegov prvi sistem iz leta 1904 je med vsemi najbolj razumljiv in enostaven, zato se še danes uporablja za učenje Boolove algebre.

Huntingtonovi postulati (1904)

P1a) $\forall a,b \in B: a+b \in B$
P1b) $\forall a,b \in B: a \cdot b \in B$
P2a) $\exists 0 \in B \ \forall a \in B: a+0=a$
P2b) $\exists 1 \in B \ \forall a \in B: a \cdot 1=a$
P3a) $\forall a,b \in B: a+b=b+a$
P3b) $\forall a,b \in B: a \cdot b=b \cdot a$
P4a) $\forall a,b,c \in B: a+bc=(a+b)(a+c)$
P4b) $\forall a,b,c \in B: a \cdot (b+c)=a \cdot b+a \cdot c$
P5a) $\forall a \in B \ \exists a' \in B: a+a'=1$
P5b) $\forall a \in B \ \exists a' \in B: a \cdot a'=0$

Vse druge zakonitosti, ki veljajo v Boolovi algebri, bomo imenovali izreki. Tukaj je nekaj primerov izrekov:

- absorbcija (dokaz v skripti): $a+ab=a$, $a \cdot (a+b)=a$
- sosednost (dokaz v skripti): $a \cdot b+a \cdot b'=a$, $(a+b) \cdot (a+b')=a$
- asociativnost (dokaz poiščite na spletu): $a+(b+c)=(a+b)+c$, $a \cdot (b \cdot c)=(a \cdot b) \cdot c$
- DeMorganov zakon (dokaz poiščite na spletu): $(a+b)'=(a' \cdot b')'$, $(a \cdot b)'=(a'+b')'$

VAJA 2.1

S pomočjo Huntingtonovih postulatov dokaži idempotenco ($a+a=a$, $a \cdot a=a$) !

$a+a$ = (P2b)
$(a+a) \cdot 1$ = (P5a)
$(a+a) \cdot (a+a')$ = (P4a)
$a+a \cdot a'$ = (P5b)
$a+0$ = (P2a)
a

$a \cdot a$ = (P2a)
$a \cdot a+0$ = (P5b)
$a \cdot a \cdot +a \cdot a'$ = (P4b)
$a \cdot (a+a')$ = (P5a)
$a \cdot 1$ = (P2b)
a

VAJA 2.2

S pomočjo Huntingtonovih postulatov dokaži identiteto ($a+1=1$, $a \cdot 0=0$) !

$a+1$ = (P5a)
$a+(a+a')$ = (asociativnost)
$(a+a)+a'$ = (idempotenca)
$a+a'$ = (P5a)
1

Tukaj pa je še dokaz za dualno obliko.

$a \cdot 0$ = (P5b)
$a \cdot (a \cdot a')$ = (asociativnost)
$(a \cdot a) \cdot a'$ = (idempotenca)
$a \cdot a'$ = (P5a)
0

VAJA 2.3
S pomočjo Huntingtonovih postulatov dokaži edinstvenost nasprotnega elementa!

Predpostavimo, da sta a_1' in a_2' dva poljubna nasprotna elementa od a. Glede na postulate ob tej predpostavki velja: $a+a_1' = 1$ (P5a), $a+a_2' = 1$ (P5a), $a \cdot a_1' = 0$ (P5b) in $a \cdot a_2' = 0$ (P5b).

a_2' = (P2b)
$a_2' \cdot 1$ = (P5a)
$a_2' \cdot (a+a_1')$ = (P4b)
$a_2' \cdot a + a_2' \cdot a_1'$ = (P3b)
$a \cdot a_2' + a_2' \cdot a_1'$ = (P5b)
$0 + a_2' \cdot a_1'$ = (P3b)
$0 + a_1' \cdot a_2'$ = (P5b)
$a \cdot a_1' + a_1' \cdot a_2'$ = (P3b)
$a_1' \cdot a + a_1' \cdot a_2'$ = (P4b)
$a_1' \cdot (a+a_2')$ = (P5a)
$a_1' \cdot 1$ = (P2b)
a_1'

Dokazali smo, da če sta a_1' in a_2' oba nasprotna elementa od a, potem sta med seboj enaka. Torej ne obstajata dva različna nasprotna elementa od a.

VAJA 2.4
S pomočjo Huntingtonovih postulatov dokaži involucijo ((a')'=a)!

$a'+a$ = (P3a)
$a+a'$ = (P5a)
1

$a' \cdot a$ = (P3b)
$a \cdot a'$ = (P5b)
0

Če velja $a'+a=1$ in $a' \cdot a=0$, potem je a nasportni element od a'.
(a')' je glede na zapis tudi nasprotni element od a'.
Ker velja edinstvenost nasprotnega elementa, torej velja (a')'=a.

VAJA 2.5
S pomočjo Huntingtonovih postulatov in izrekov, za katere poznaš ime, dokaži, da velja $(a \cdot b') + (a' \cdot b) = (a+b) \cdot (a'+b')$. - *Rešite sami.*

3. Kanonična oblika Boolovih funkcij, PDNO, PKNO

Popolna disjunktivna normalna oblika (PDNO) je vsota produktov, zato jo označujemo tudi s kratico SOP (sum of products). Vendar pa produkti ne morejo biti poljubni, dovoljeni so le mintermi – to so taki produkti, ki vsebujejo vse spremenljivke. PDNO na kratko zapišemo kot Σ(...).

Popolna konjunktivna normalna oblika (PKNO) je produkt vsot, zato jo označujemo tudi s kratico POS (product of sums). Vendar pa vsote ne morejo biti poljubne, dovoljeni so le makstermi – to so take vsote, ki vsebujejo vse spremenljivke. PKNO na kratko zapišemo kot Π(...).

PDNO in PKNO sta kanonični obliki Boolovih funkcij. Če se dogovorimo za vrstni red spremenljivk, potem za vsako Boolovo funkcijo obstaja natanko ena PDNO in natanko ena PKNO.

Za funkcijo F(a,b,c) velja:

$m_0 = a'\cdot b'\cdot c'$	$m_4 = a\cdot b'\cdot c'$	$M_0 = a+b+c$	$M_4 = a'+b+c$
$m_1 = a'\cdot b'\cdot c$	$m_5 = a\cdot b'\cdot c$	$M_1 = a+b+c'$	$M_5 = a'+b+c'$
$m_2 = a'\cdot b\cdot c'$	$m_6 = a\cdot b\cdot c'$	$M_2 = a+b'+c$	$M_6 = a'+b'+c$
$m_3 = a'\cdot b\cdot c$	$m_7 = a\cdot b\cdot c$	$M_3 = a+b'+c'$	$M_7 = a'+b'+c'$

V splošnem velja $m_i(...) = (M_i(...))'$. Na primer: $m_3(a,b,c) = a'\cdot b\cdot c = (M_3(a,b,c))' = (a+b'+c')'$

PDNO pretvorimo v PKNO tako, da naštejemo vse tiste maksterme M_i, za katere velja, da minterm m_i ne nastopa v PDNO.

PKNO pretvorimo v PDNO tako, da naštejemo vse tiste minterme m_i, za katere velja, da maksterm M_i ne nastopa v PKNO.

VAJA 3.1
Dani sta funkciji F(A,B,C)=A+B·C in G(A,B,C)=(B+C)'. Zapiši ju v PDNO in PKNO. Nato zapiši v PDNO in PKNO tudi funkcije F+G, F·G in F≡G.

Najprej pretvorimo v disjunktivno normalno obliko in zapišemo pravilnostno tabelo.

F(A,B,C) = A+B·C
G(A,B,C) = (B+C)' = B'·C'

A	B	C	F	G	F + G	F · G	F ≡ G
0	0	0	0	1	1	0	0
0	0	1	0	0	0	0	1
0	1	0	0	0	0	0	1
0	1	1	1	0	1	0	0
1	0	0	1	1	1	1	1
1	0	1	1	0	1	0	0
1	1	0	1	0	1	0	0
1	1	1	1	0	1	0	0

Dobimo:

$F(A,B,C) = \Sigma(3,4,5,6,7) = \Pi(0,1,2)$
$G(A,B,C) = \Sigma(0,4) = \Pi(1,2,3,5,6,7)$
$F(A,B,C) + G(A,B,C) = \Sigma(0,3,4,5,6,7) = \Pi(1,2)$
$F(A,B,C) \cdot G(A,B,C) = \Sigma(4) = \Pi(0,1,2,3,5,6,7)$
$F(A,B,C) \equiv G(A,B,C) = \Sigma(1,2,4) = \Pi(0,3,5,6,7)$
Do rezultata pa lahko pridemo tudi brez pisanja pravilnostne tabele. Označimo množico mintermov v PDNO poljubne funkcije F kot F_m, množico makstermov v PKNO te funkcije pa kot F_M. Potem velja:

$(F')_m = F_M$	$(F+G)_m = F_m \cup G_m$	$(F \cdot G)_m = F_m \cap G_m$
$(F')_M = F_m$	$(F+G)_M = F_M \cap G_M$	$(F \cdot G)_M = F_M \cup G_M$

VAJA 3.2
Dana je funkcija F = (A+B' +C) ⎓ (A+B)'. Zapišite jo v PDNO in PKNO. - *Rešite sami.*

VAJA 3.3
Dani sta funkciji F=A'+B in G=A·B·C. Funkcije F → G, F|G in F↓G zapiši v PDNO in PKNO. - *Rešite sami.*

4. Minimizacija Boolovih funkcij, MDNO, MKNO

Minimalna disjunktivna normalna oblika (MDNO) je dvonivojska oblika Boolove funkcije (vsota produktov), pri kateri je uporabljeno najmanjše možno število produktov, produkti pa vsebujejo najmanjše možno število spremenljivk.

Minimalna konjunktivna normalna oblika (MKNO) je dvonivojska oblika Boolove funkcije (produkt vsot), pri kateri je uporabljeno najmanjše možno število vsot, voste pa vsebujejo najmanjše možno število spremenljivk.

Iskanje MDNO in MKNO imenujemo ***minimizacija Boolove funkcije***. Minimizacija Boolovih funkcij z dvema, tremi ali štirimi spremenljivkami lahko dokaj enostavno določimo ročno. Z malo več truda lahko na podoben način minimiziramo tudi funkcije s petimi ali celo šestimi spremenljivkami, pri večjih funkcijah pa je potrebno uporabiti računalnik.

Ročno metodo minimizacije Boolove funkcije s pomočjo diagramov si je leta 1952 izmislil američan **Edward W. Veitch**. Leta 1953 je američan **Maurice Karnaugh** njegovo metodo nekoliko preoblikoval in njegova različica se danes v praksi najpogosteje uporablja.

Veitchev diagram za Boolove funkcije z dvema, tremi in štirimi spremenljivkami (številke v diagramu označujejo minterme)

3	1
2	0

6	7	3	2
4	5	1	0

12	14	6	4
13	15	7	5
9	11	3	1
8	10	2	0

Karnaugh-jev diagram za Boolove funkcije z dvema, tremi in štirimi spremenljivkami (številke v diagramu označujejo minterme)

0	2
1	3

0	2	6	4
1	3	7	5

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

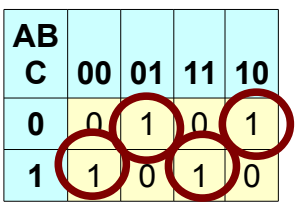
Minimizacija Boolovih funkcij poteka po naslednjih pravilih:

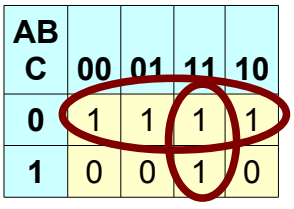
- V Veitchev oz. Karnaugh-jev diagram vpiši enice na tista mesta, ki pripadajo mintermom v dani Boolovi funkciji, druga polja pustiš prazna ali pa vpišeš ničle.
- Obkroži enice v diagramu s čim manjšim številom (prvi kriterij) čim večjih krogov.
- Krogi lahko obsegajo 1, 2, 4, 8, 16, … polj (potence števila 2).
- Krogi lahko segajo čez rob in se lahko med seboj prekrivajo.
- Obkrožiti moraš vsa polja z enicami in nobeno prazno polje (polje z ničlo).
- Ko uspešno izvedeš kroženje, iz diagrama razberi minimalno obliko.
- Nekatere Boolove funkcije imajo več različnih enako dobrih minimalnih oblik!

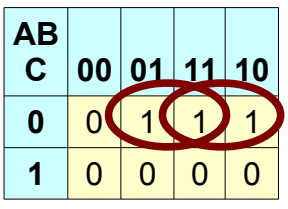
Opisan postopek minimizacije lahko učinkovito uporabimo tudi za iskanje najmanjše Boolove funkcije iz družine funkcij, ki je podana s pomočjo množice DCS (don't care set).

VAJA 4.1
Dane so funkcije F=(A⊕B)⊕C, G=(A|B)|C in H=(A↓B)↓C. Zapiši jih v MDNO in MKNO.

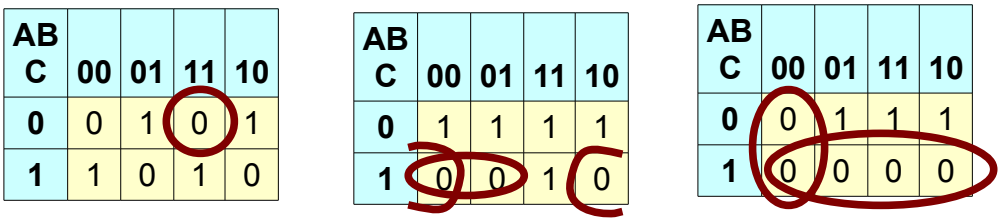
A	B	C	A ⁞⁞ B	F	A|B	G	A↓B	H
0	0	0	0	0	1	1	1	0
0	0	1	0	1	1	0	1	0
0	1	0	1	1	1	1	0	1
0	1	1	1	0	1	0	0	0
1	0	0	1	1	1	1	0	1
1	0	1	1	0	1	0	0	0
1	1	0	0	0	0	1	0	1
1	1	1	0	1	0	1	0	0







F = A'·B'·C + A'·B·C' + A·B·C + A·B'·C'
G = C' + A·B
H = B·C' + A·C'



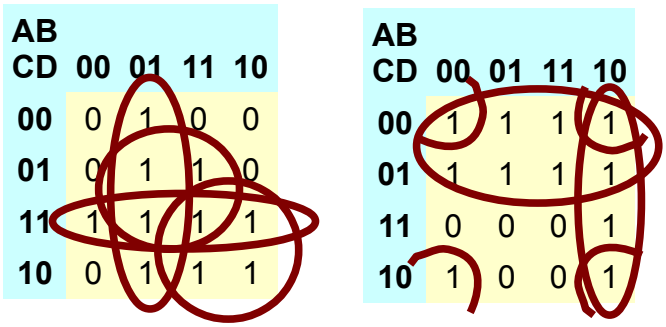
AB \ C	00	01	11	10
0	0	1	0	1
1	1	0	1	0

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

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

F = (A+B+C) · (A+B'+C') · (A'+B'+C) · (A'+B+C')
G = (A+C') · (B+C')
H = C' · (A+B)

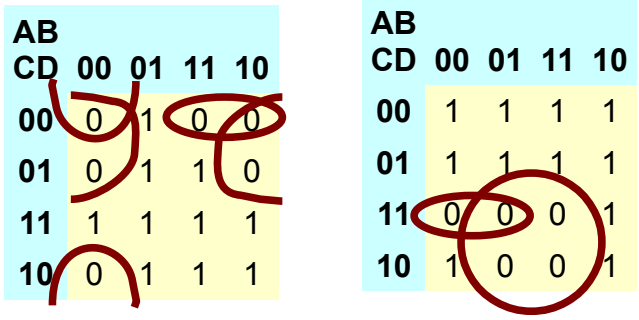
VAJA 4.2
Dani sta funkciji F(A,B,C,D) = Σ(3,4,5,6,7,10,11,13,14,15) in G(A,B,C,D) = Π(3,6,7,14,15). Zapiši ju v MDNO in MKNO.



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

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

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



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

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

F = (B+C) · (A'+C+D) · (A+B+D)
G = (B'+C') · (A+C'+D')

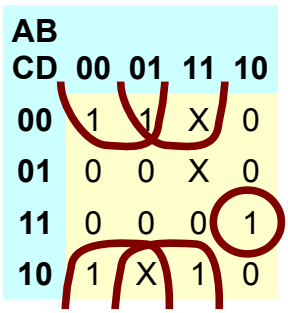
VAJA 4.3
Dane so funkcije F1(A,B,C,D) = Σ(0,2,6,10,15), F2(A,B,C,D) = Σ(3,6,7,10,15) ter izpeljana funkcija F3(A,B,C,D) = F1·F2. Funkciji F1 in F2 zapiši v MDNO. Funkcijo F3 zapiši v MKNO.

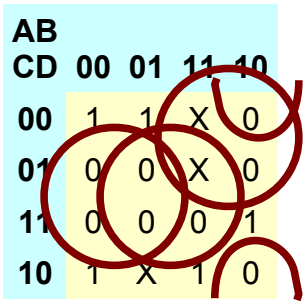
Rešite sami.

VAJA 4.4
Poišči MDNO in MKNO Boolove funkcije F(A,B,C,D) = A·(B|C) + B·(C|D)

Rešite sami.

VAJA 4.5
Poišči MDNO in MKNO najmanjše Boolove funkcije F(A,B,C,D), ki pripada družini funkcij Σ(0,2,4,11,14) + Δ(6,12,13).





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

5. Kombinacijska vezja s Shefferjevimi in Peircovimi elementi

Shefferjev element (vrata NAND) in Peircov element (vrata NOR) vsak za sebe tvorita funkcijsko poln sistem. To pomeni, da lahko samo z enim od njiju realiziramo vse Boolove funkcije. To lastnost je leta 1880 prvi odkril američan **Charles Sanders Peirce**, a je bilo njegovo delo objavljeno šele leta 1933. Vmes je leta 1913 to lastnost ponovno odkril američan poljskega rodu **Henry Maurice Sheffer.**

A'= A | A = A ↓ A
A + B = (A' · B')' = A' | B' = (A | A) | (B | B)
A · B = (A · B)'' = (A | B)' = (A | B) | (A | B)
A + B = (A + B)'' = (A ↓ B)' = (A ↓ B) ↓ (A ↓ B)
A · B = (A' + B')' = A' ↓ B' = (A ↓ A) ↓ (B ↓ B)

Realizacijo Boolove funkcije s Shefferjevimi elementi najlažje izvedemo tako, da funkcijo najprej zapišemo v MDNO. Realizacijo Boolove funkcije s Peircovimi elementi najlažje izvedemo tako, da funkcijo najprej zapišemo v MKNO.

A + B·C + D·E·F = A' | (B | C) | (D | E | F)
A · (B+C) · (D+E+F) = A' ↓ (B ↓ C) ↓ (D ↓ E ↓ F)

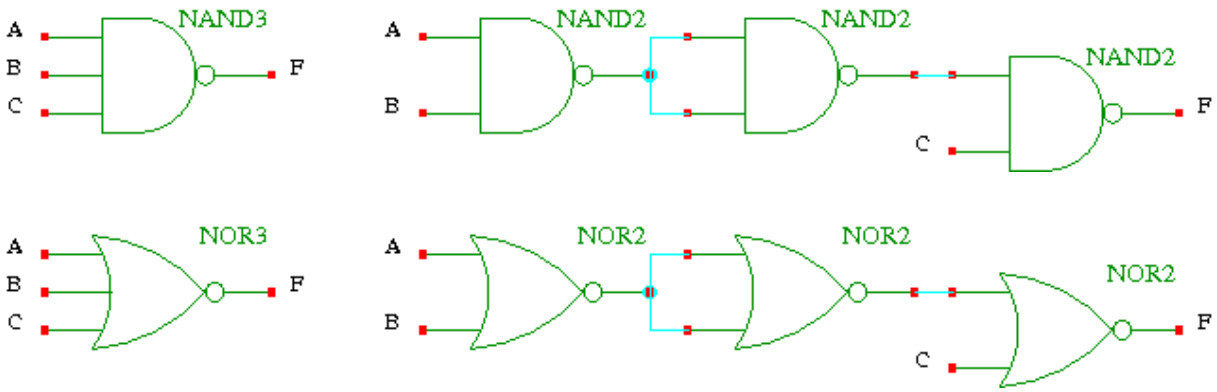
Če ima MDNO manj členov kot MKNO, se splača realizacijo s Peircovimi elementi izpeljati iz MDNO. Podobno velja za realizacijo s Shefferjevimi elementi, če ima MKNO manj členov.

A + B·C + D·E·F = (A ↓ (B' ↓ C') ↓ (D' ↓ E' ↓ F'))'
A · (B+C) · (D+E+F) = (A | (B' | C') | (D' | E' | F'))'

Tri-vhodni Shefferjev in Peircov element realiziramo z dvo-vhodnimi po naslednjem vzorcu:

(A | B | C) = (A | B)' | C
(A ↓ B ↓ C) = (A ↓ B)' ↓ C

V vezju to izgleda takole:



Ker so Shefferjevi in Peircovi elementi poleg svoje uporabnosti tudi enostavni za izvedbo z integriranimi vezji, na njih temelji precejšen del strojne opreme današnjih računalnikov in pripomočkov, npr. pomnilniki flash. Računalnik, ki je pomagal krmiliti raketo Apollo, s katero je na luno poletel prvi človek (Apollo Guidance Computer), je bil zgrajen iz tri-vhodnih Peircovih elementov – v prvi različici jih je bilo 4100, vsak v svojem ohišju.

VAJA 5.1
Samo z uporabo Shefferjevih in Peircovih elementov realizirajte konstanti 0 in 1.

Rešite sami.

VAJA 5.2
Z enim ali več tri-vhodnimi Shefferjevimi elementi realizirajte dvo-vhodni Shefferjev element.

Rešite sami.

VAJA 5.3
Z enim ali več dvo-vhodnimi Shefferjevimi elementi realizirajte štiri-vhodni Shefferjev element.

Rešite sami.

VAJA 5.4
Dane so Boolove funkcije F = A ↓ B, G = A ↓ B ↓ C = (A+B+C)' in H(A,B,C) = Σ(1,2,3,5,7). Realizirajte jih tako, da uporabite le dvo-vhodne Shefferjeve elemente.

A	B	C	A↓B	G	H
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	0	0
1	1	1	0	0	1

A B	0	1
0	1	0
1	0	0

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

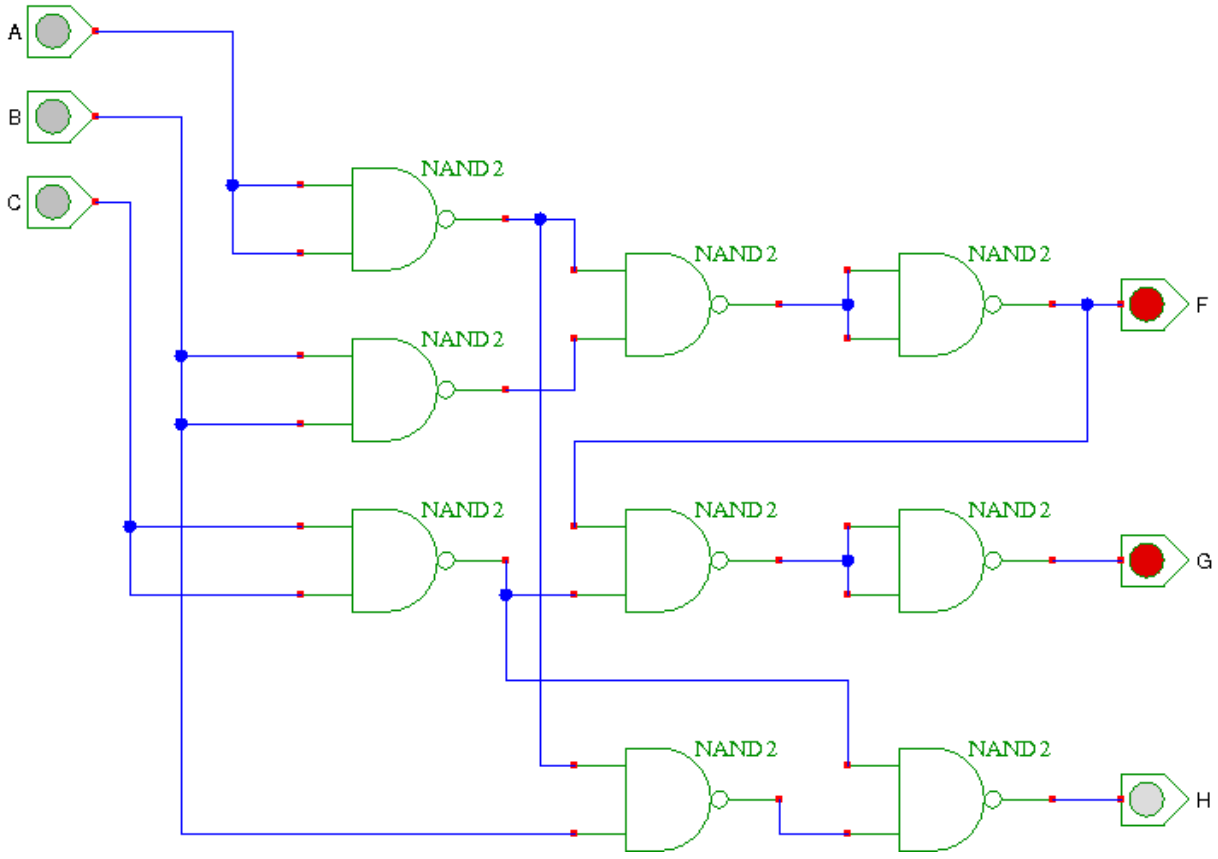
AB C	00	01	11	10
0	0	1	0	0
1	1	1	1	1

F = A'·B' = (A' | B')'

G = A'·B'·C' =
= (A' | B' | C')' =
= ((A' | B')' | C')'

H = C + A'·B = C' | (A' | B)

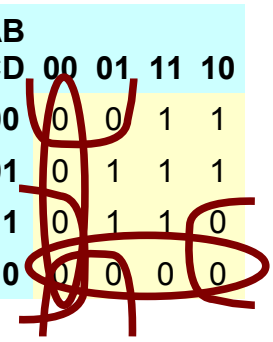
Pri realizaciji bomo vsa tri vezja združili.



VAJA 5.5

Funkcijo F(A,B,C,D) = Π(0,1,2,3,4,6,7,10,11,14) realiziraj najprej samo z dvo-vhodnimi Shefferjevimi elementi nato pa še samo z dvovhodnimi Peircovimi elementi.

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



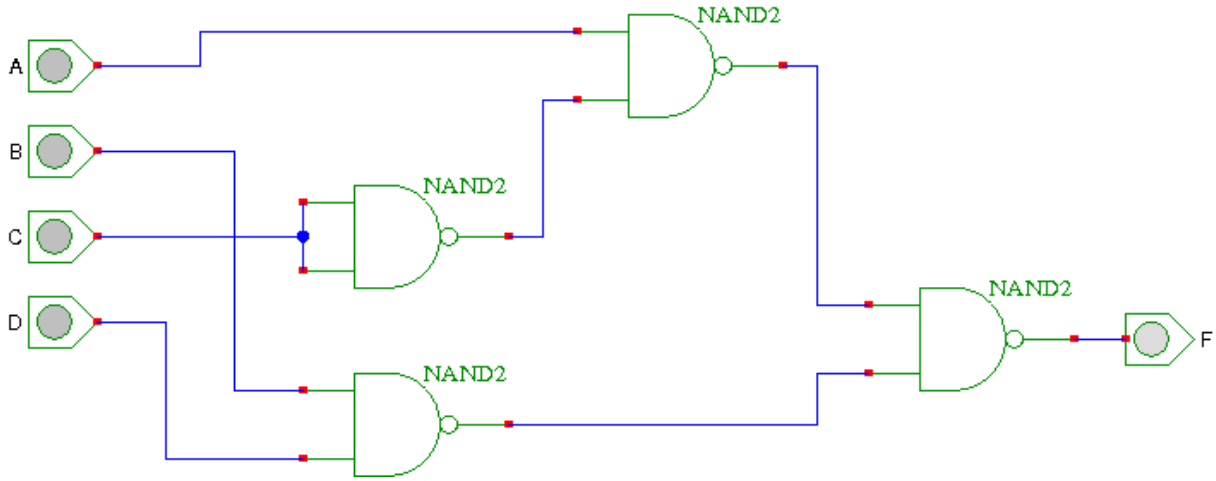
F = A·C' + B·D = (A | C') | (B | D)

F = (A+B) · (B+C') · (C'+D) · (A+D) =
= (A↓B) ↓ (B↓C') ↓ (C'↓D) ↓ (A↓D) =
= ((A↓B) ↓ (B↓C'))' ↓ ((C'↓D) ↓ (A↓D))'

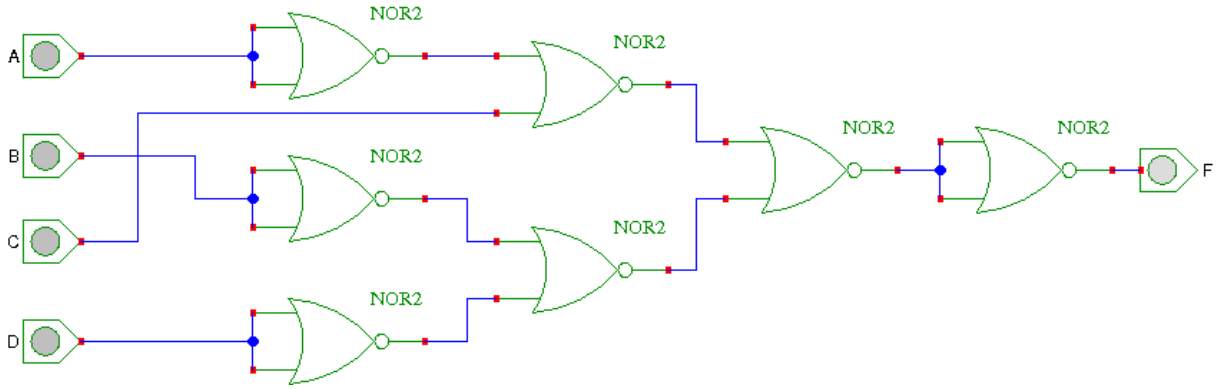
Ker ima MDNO bistveno manj členov, je boljša naslednja realizacija s Peircovimi elementi:

F = A·C' + B·D =
= (A·C' ↓ B·D)' =
= ((A'↓C) ↓ (B'↓D'))'

Tukaj je realizacija z dvo-vhodnimi Shefferjevimi elementi.



Tukaj je realizacija z dvo-vhodnimi Peircovimi elementi.



6. Priprave na prvi kolokvij

Snov prvega kolokvija je vse, kar je v tem gradivu v poglavjih 1-5.

7. Računalniško modeliranje in simulacija vezij

Programe za modeliranje in simulacijo vezij lahko razdelimo v dve skupini:

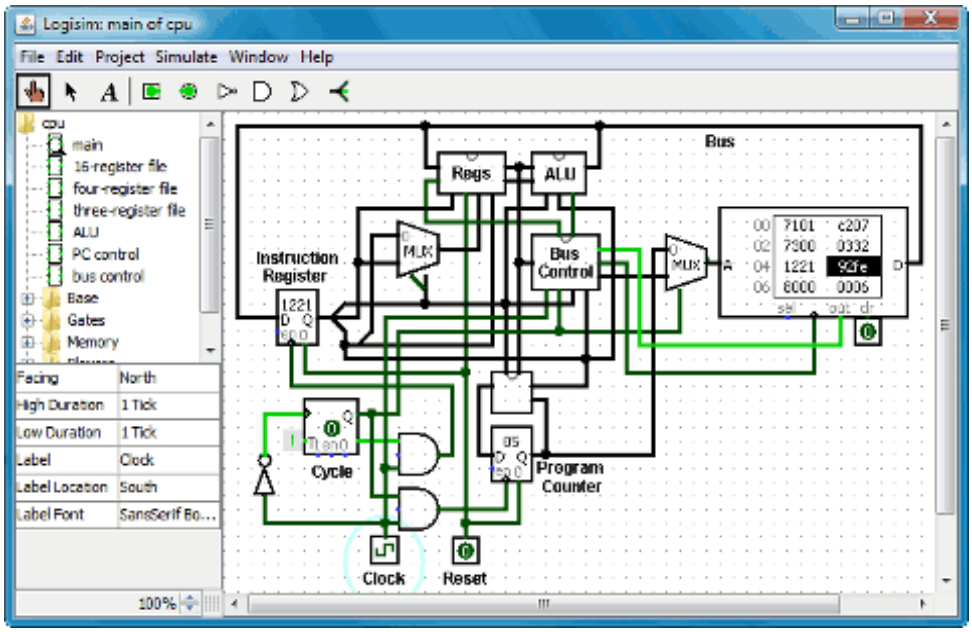
1. programi namenjeni izobraževanju
2. programi namenjeni sintezi realnih vezij

Izobraževalni programi ponujajo grafični urejevalnik, s katerim sestavimo vezje na logičnem nivoju. Torej nam ni treba skrbeti za to, kako se bo vezje napajalo, s kakšno napetostjo ter kje bo masa. Na voljo imamo osnovna vrata in elemente, kot vhod uporabimo stikala, za izhod pa LED diode. Boljši programi omogočajo tudi kompleksnejše vhode in izhode kot sta npr. šestnajstiška tipkovnica in 7-segmentni prikazovalnik. Pomemben del teh programov je simulator, ki mora znati upoštevati tudi zakasnitve elementov. Boljši programi znajo prikazati časovne diagrame za označene signale.

Programi namenjeni sintezi realnih vezij so mnogo bolj kompleksni. V vezju uporabljamo gradnike, ki označujejo realne čipe. Simulator je pogosto dodatni modul, simulacija pa običajno poteka na analognem nivoju in ni interaktivna. Glavni namen teh orodij je optimizirati razporeditev gradnikov in povezav ter pripraviti takšne datoteke, ki jih razumejo naprave za izdelavo vezij.

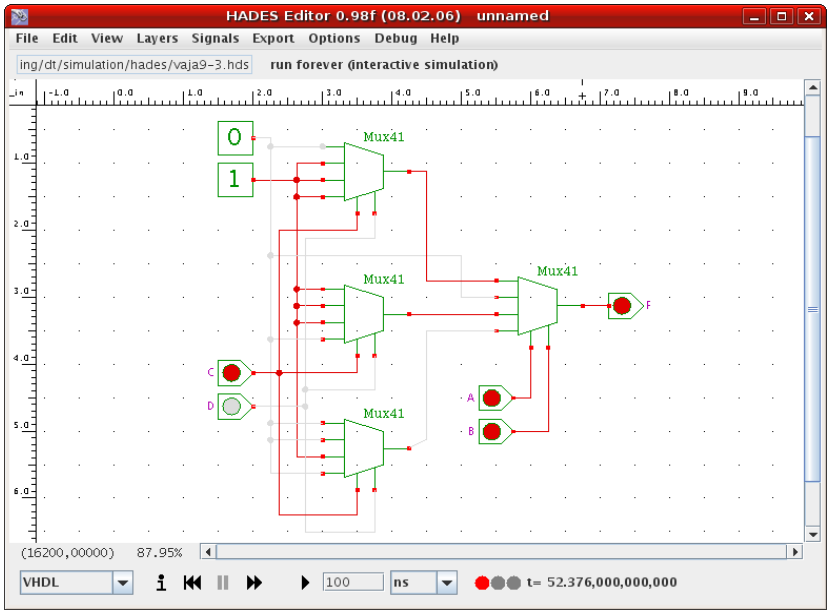
Logisim

Med mnogimi prostimi izobraževalnimi programi, ki jih najdemo na internetu, je v zadnjem času najboljše vzdrževan program Logisim. Je v celoti napisan v Javi, zato lahko z njim delamo na različnih operacijskih sistemih.



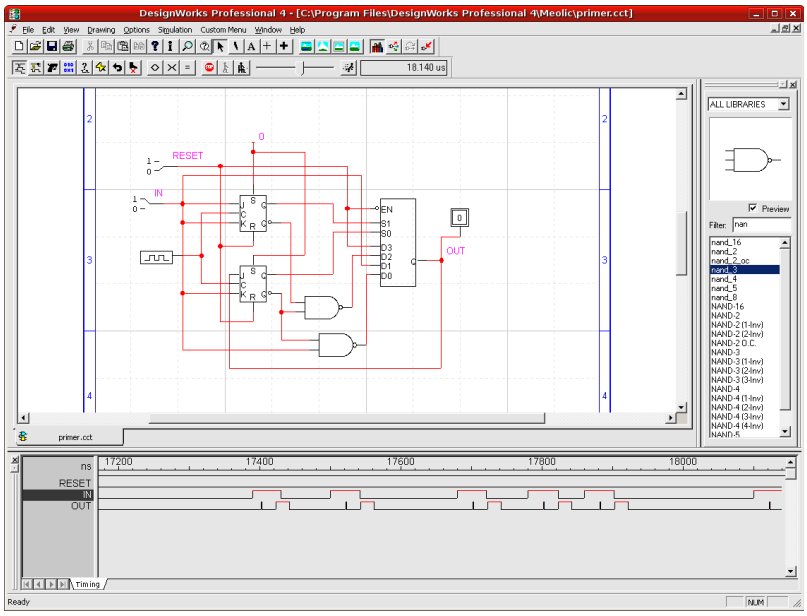
Hades

Prostodostopno je tudi orodje Hades (The Hamburg Design System), ki ga razvijajo ter uporabljajo na Univerzi v Hamburgu v Nemčiji. Gre za licenčno orodje, ki pa se lahko (zaenkrat) prosto uporablja za izobraževalne namene. Hades je v celoti napisa v Javi in ga lahko uporabljamo kot Java program (application) ali pa Java programček (applet).



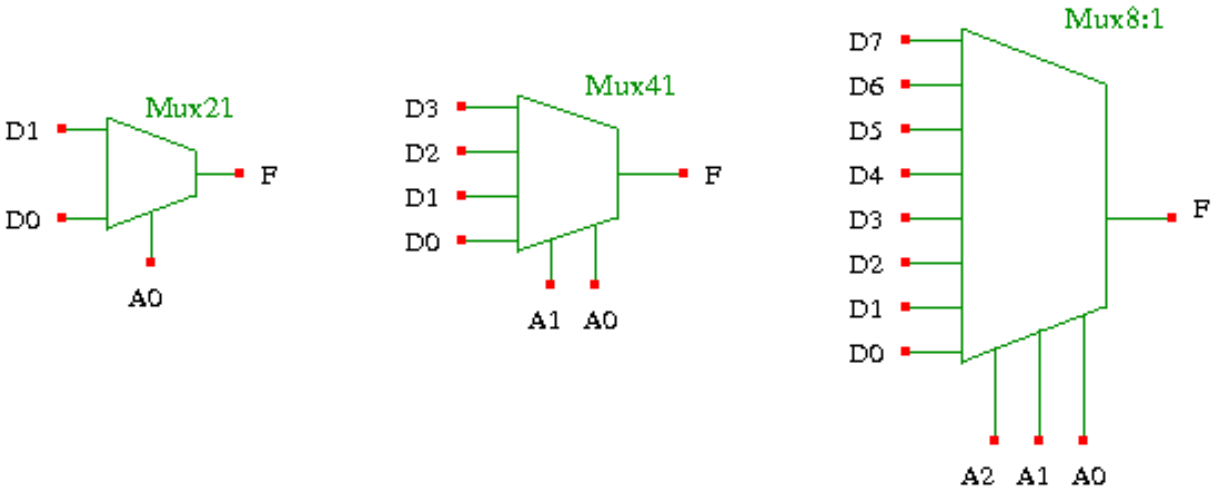
DesignWorks in LogicWorks

Med komercialnimi programi, ki so uporabni v izobraževalne namene, je zelo dodelano orodje DesigWorks. V polni različici (DesignWorks Professional, urejevalnik + simulator = $790) ima program precej funkcionalnosti, ki so značilne za programe namenjene sintezi realnih vezij. Omejena različica, ki je osredotočena predvsem na simulacijo, se imenuje LogicWorks ($90) in se prodaja le skupaj z knjigo, ki je neke vrste učbenik. DesignWorks (in tudi LogicWorks) sta programa za računalnike z operacijskim sistemom MS Windows, na GNU/Linuxu pa ju lahko poženemo s pomočjo pripomočka Wine.

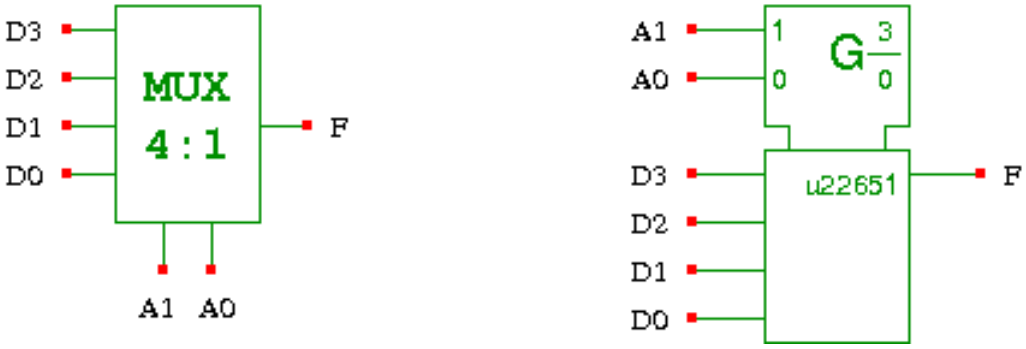


8. Multipleksor

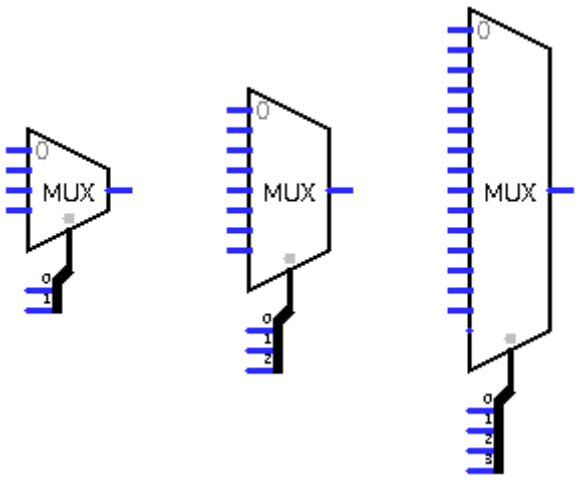
Multipleksorji so strukturalno vezje, s katerim lahko realiziramo poljubno Boolovo funkcijo. Po delovanju spominjajo na preklopno stikalo. Ločimo jih glede na število naslovnih vhodov. Multipleksor z n podatkovnimi vhodi ima 2^n podatkovnih vhodov. Multipleksorji imajo en sam izhod. Oznaka multipleksorja je sestavljena iz število podatkovnih vhodov in število izhodov (ki je vedno enako 1), torej MUX 2:1, MUX 4:1, MUX 8:1 itd.



Različna orodja uporabljajo različne simbole. Multipleksor MUX 4:1 lahko npr. najdemo narisan tudi v naslednjih oblikah:



Pri orodju Logisim (glej v nadaljevanju) vsakemu multipleksorju dodamo gradnik imenovan Splitter, da ločimo posamezne naslovne vhode med seboj. Rezultat je takšen:

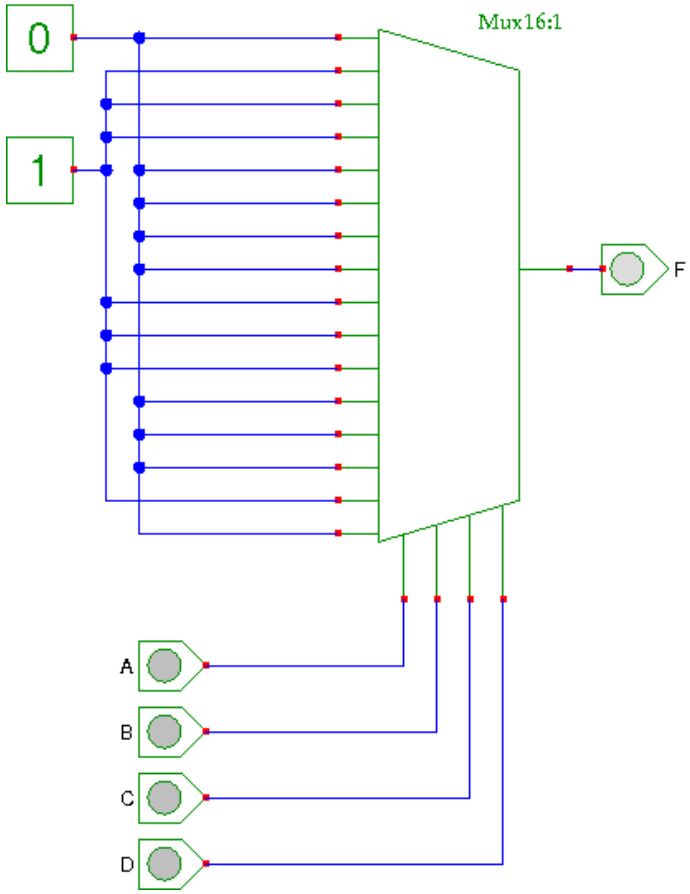


VAJA 8.1

Boolovo funkcijo F(A,B,C,D) = Σ(1,5,6,7,12,13,14) realiziraj z multipleksorjem MUX 16:1 tako, da bodo na podatkovnih vhodih le konstante 0 ali 1.

Rešitev te naloge je enostavna. Na naslovne vhode pripeljemo spremenljivke, na podatkovne pa vrednosti iz pravilnostne tabele (od spodaj navzgor).

A	B	C	D	F
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**0**
0	1	0	1	**1**
0	1	1	0	**1**
0	1	1	1	**1**
1	0	0	0	**0**
1	0	0	1	**0**
1	0	1	0	**0**
1	0	1	1	**0**
1	1	0	0	**1**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

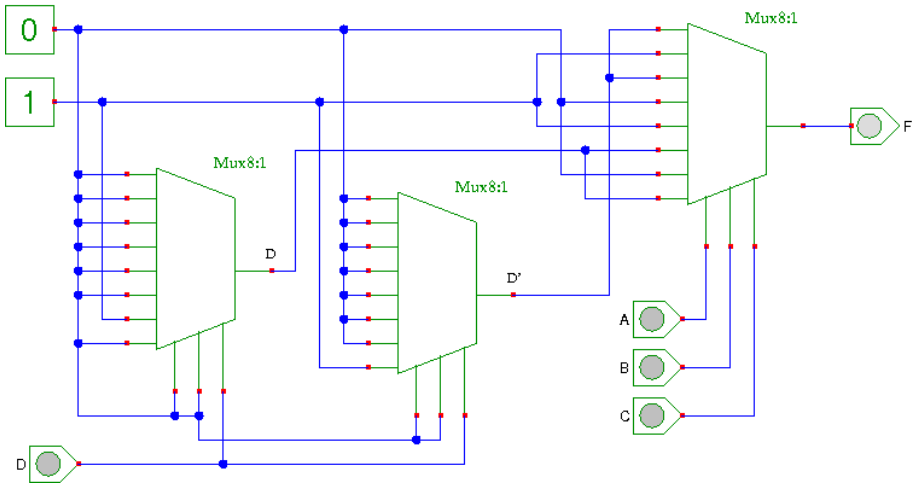


VAJA 8.2

Boolovo funkcijo F(A,B,C,D) = Σ(1,5,6,7,12,13,14) realiziraj z enim ali več multipleksorjev MUX 8:1 tako, da bodo na podatkovnih vhodih le konstante 0 ali 1.

V tem primeru izberemo tri spremenljivke, ki bodo na skrajno desnem multipleksorju (najlažje je, če izberemo spremenljivke A, B in C). Nato iz pravilnostne tabele razberemo funkcijske ostanke. Dobimo:

F(0,0,0,D) = D
F(0,0,1,D) = 0
F(0,1,0,D) = D
F(0,1,1,D) = 1
F(1,0,0,D) = 0
F(1,0,1,D) = 0
F(1,1,0,D) = 1
F(1,1,1,D) = D'



Vidimo lahko, da je rešitev zelo slaba, saj smo potrebovali 3 multipleksorje, vendar pa dva od teh skrbita le za izračun preprostih funkcij D in D'. Morda se lahko odvečnim multipleksorjem izognemo, če na skrajno desnem multipleksorju izberemo drugačen nabor spremenljivk. Na voljo imamo 4 možnosti.

F(A,0,0,0) = 0 F(A,0,0,1) = A' F(A,0,1,0) = 0 F(A,0,1,1) = 0 F(A,1,0,0) = A F(A,1,0,1) = 1 F(A,1,1,0) = 1 F(A,1,1,1) = A'	F(0,B,0,0) = 0 F(0,B,0,1) = 1 F(0,B,1,0) = B F(0,B,1,1) = B F(1,B,0,0) = B F(1,B,0,1) = B F(1,B,1,0) = B F(1,B,1,1) = 0	F(0,0,C,0) = 0 F(0,0,C,1) = C' F(0,1,C,0) = C F(0,1,C,1) = 1 F(1,0,C,0) = 0 F(1,0,C,1) = 0 F(1,1,C,0) = 1 F(1,1,C,1) = C'	F(0,0,0,D) = D F(0,0,1,D) = 0 F(0,1,0,D) = D F(0,1,1,D) = 1 F(1,0,0,D) = 0 F(1,0,1,D) = 0 F(1,1,0,D) = 1 F(1,1,1,D) = D'

Ugotovimo lahko, da je rešitev, pri kateri na desnem multipleksorju izberemo spremenljivke A,C in D boljša od ostalih, saj ne potrebujemo B' in bomo imeli en multipleksor manj.

Vezje za optimalno rešitev narišite sami.

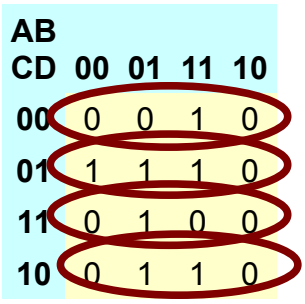
VAJA 8.3

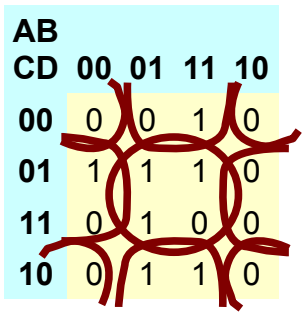
Boolovo funkcijo F(A,B,C,D) = Σ(1,5,6,7,12,13,14) realiziraj z enim ali več multipleksorjev MUX 4:1 tako, da bodo na podatkovnih vhodih le konstante 0 ali 1.

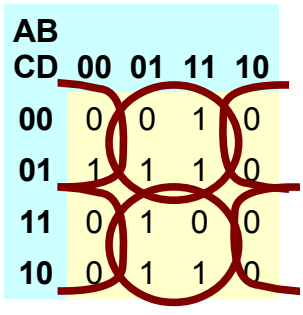
Izbrati moramo dve spremenljivki, ki bosta na skrajno desnem multipleksorju. Lotimo se lahko tako, da izberemo spremenljivki A, B, ali pa med vsemi možnostmi poiščemo najboljšo. V tem primeru je 6 različnih možnosti, ki nam dajo naslednjo tabelo:

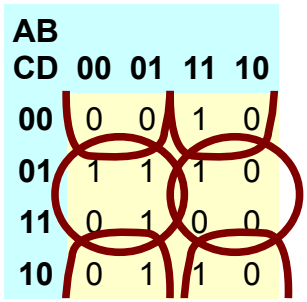
F(A,B,0,0) = A·B F(A,B,0,1) = A'+B F(A,B,1,0) = B F(A,B,1,1) = A'·B	F(A,0,C,0) = 0 F(A,0,C,1) = A'·C' F(A,1,C,0) = A+C F(A,1,C,1) = A'+C'	F(A,0,0,D) = A'·D F(A,0,1,D) = 0 F(A,1,0,D) = A+D F(A,1,1,D) = A'+D'
F(0,B,C,0) = B·C F(0,B,C,1) = B+C' F(1,B,C,0) = B F(1,B,C,1) = B·C'	F(0,B,0,D) = D F(0,B,1,D) = B F(1,B,0,D) = B F(1,B,1,D) = B·D'	F(0,0,C,D) = C'·D F(0,1,C,D) = C+D F(1,0,C,D) = 0 F(1,1,C,D) = C'+D'

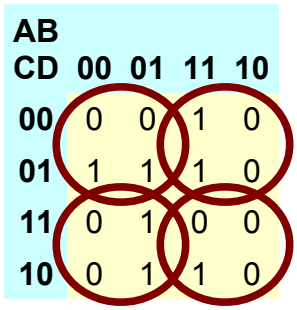
Ugotovimo lahko, da pari spremenljivk (B,D), (B,C), (A,C) in (A,B) na desnem multipleksorju zahtevajo po 3 dodatne multipleksorje, medtem ko para (C,D) in (A,D) na desnem multipleksorju zahtevata po 4 dodatne multipleksorje. Pri iskanju optimalne razporeditve vhodov si lahko pomagamo tudi s Karnaugh-jevim diagramom.

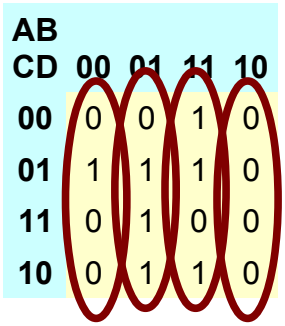




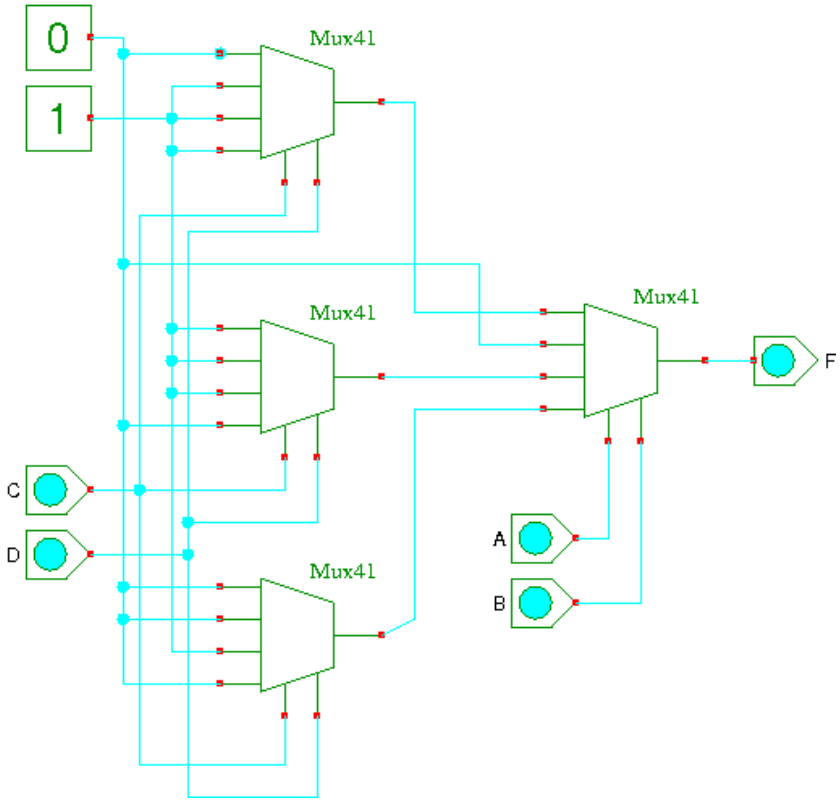








Na osnovi izračuna se odločimo, da na desnem multipleksorju izberemo A in B, ker pri tej rešitvi najlažje vidimo, da se nismo zmotili – vrednosti iz pravilnostne tabele so naštete od spodaj navzgor.



VAJA 8.4

Dane so Boolove funkcije F=A|B, G=(A·B·C·D)' in H=ITE(A,B,C)=(A·B)+(A'·C). Realizirajte jih z enim ali več multipleksorjev MUX 4:1 tako, da so na podatkovnih vhodih le konstante.

Rešite sami.

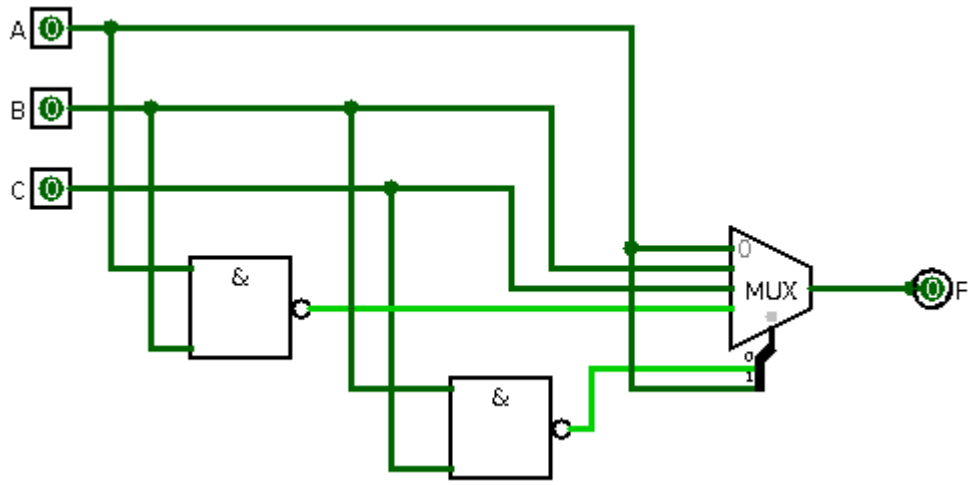
VAJA 8.5

Imamo en multipleksor MUX 8:1 in en multipleksor MUX 4:1. S tema dvema elementoma želimo realizirati funkcijo F(A,B,C,D)=Σ(2,6,7,10,11,14) tako, da bosta na podatkovnih vhodih multipleksorjev le konstanti 0 in 1. Nimamo nobenih drugih vrat, niti invertorjev! Poiščite pravilno realizacijo ali pa pokažite, da se ta funkcija na ta način ne da realizirati.

Rešite sami.

VAJA 8.6

Zapišite MDNO za logično funkcijo F, ki je realizirana s prikazanim vezjem.

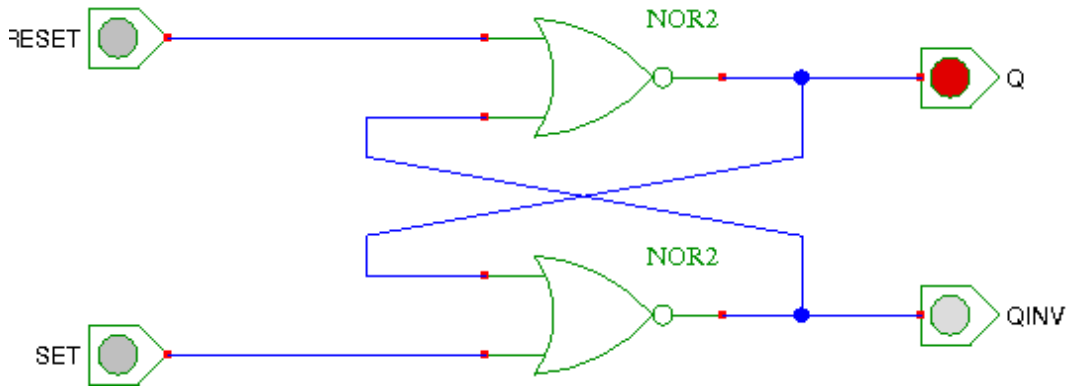


Rešite sami.

9. Pomnilne celice, analiza sekvenčnih vezij

Če v vezju tvorimo zanke tako, da vrednost izhoda iz nekih vrat pripeljemo nazaj na vhode teh istih vrat, vnesemo v vezje nestabilnost. Nekatera taka vezja so trajno nestabilna in se njihovi izhodi neprestano spreminjajo iz 0 v 1 ter nazaj – pravimo, da oscilirajo. Nekatera taka vezja po nekaj oscilacijah dosežejo stabilno vrednost iz katere jih ni več mogoče premakniti s spreminjanjem vhodov. Nekatera vezja pa imajo več različnih stabilnih vrednosti (stanj) med katerimi lahko preklapljamo z ustreznimi kombinacijami vhodov. Slednja vezja so praktično zelo uporabna, saj imajo tako imenovani spominski efekt – glede na zgodovino vhodov se postavijo v določeno stabilno stanje in tam ostanejo dokler spet ustrezno ne spremenimo vhodov.

Vezja s spominskim efektom imenujemo sekvenčna vezja. Naj še enkrat poudarimo, da spominskega efekta brez zank v vezju ni mogoče doseči. Najpreprostejše sekvenčno vezje ima samo en izhod in ga imenujemo zadrževalnik RS (v angleščini „RS latch“). Prikazan je na spodnji sliki.

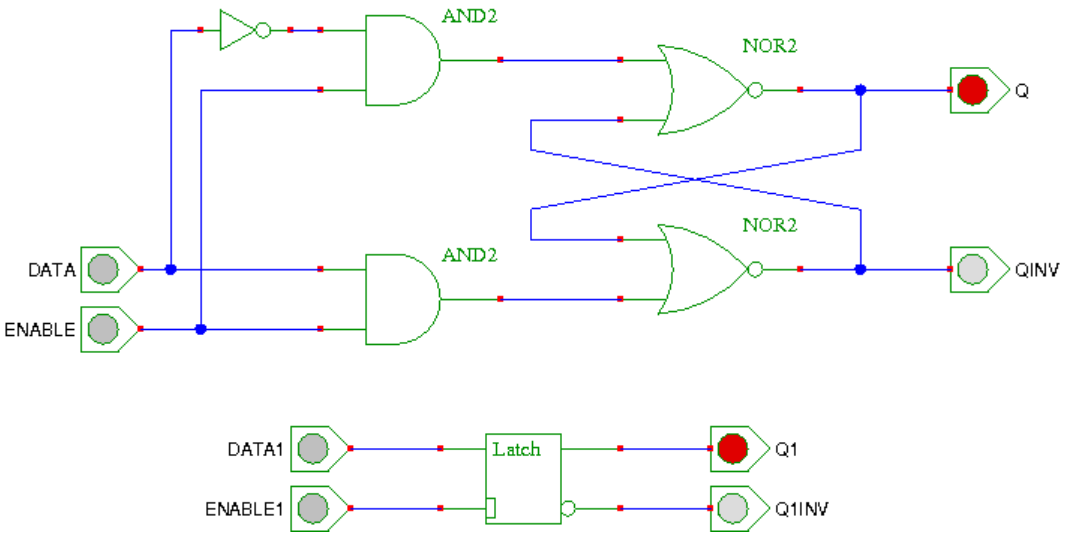


Dokler sta vhoda SET in RESET oba na nič, je vezje stabilno in se vrednost izhoda Q ne spreminja. Če se vhod RESET postavi na ena, se vrednost izhoda Q ne glede na prejšnjo vrednost postavi na nič in takšno stanje ostane tudi potem, ko vrednost vhoda RESET spremenimo nazaj na nič. Če se vhod SET postavi na ena, pa se vrednost izhoda Q ne glede na prejšnjo vrednost postavi na ena in takšno stanje ostane tudi potem, ko vrednost vhoda SET spremenimo nazaj na nič.

Zadrževalnik RS lahko zgradimo tudi z dvema Shefferjevima elementoma, le da potem vhoda vplivata na vezje (sta aktivna) takrat, ko je njuna vrednost enaka nič.

Vezje na sliki ima dva izhoda, ki pa med seboj nista neodvisna. Izhod QINV je vedno negirana vrednost izhoda Q (izjema se zgodi, če oba vhoda postavimo hkrati na ena, kar pa proglasimo za prepovedano kombinacijo). Izhod QINV je v praksi lahko koristen, v teoriji pa se z njim ne rabimo ukvarjati.

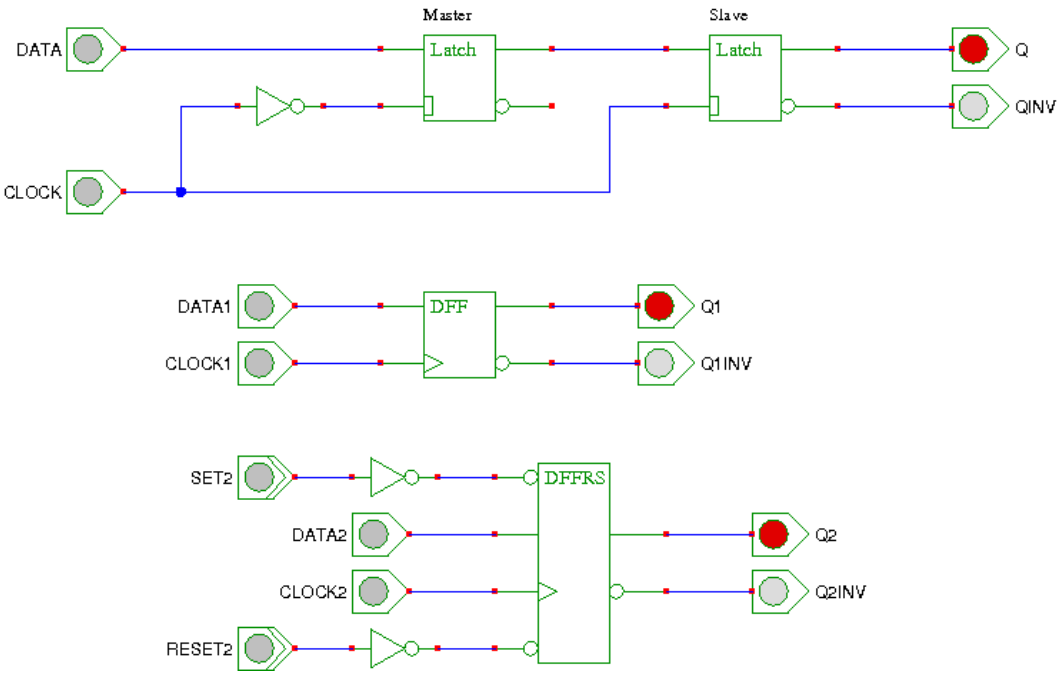
Če dodamo še nekaj vrat, dobimo zadrževalnik D. Na izhod tega elementa vplivamo samo z enim vhodom, ki ga označimo z DATA. Ima pa vezje še dodatni vhod ENABLE, ko je ta postavljen na nič, spremembe vhoda ne vplivajo na vrednost izhoda.



Zadrževalniki sami po sebi ne zagotavljajo sinhronega delovanja. To pomeni, da če imamo v vezju več zadrževalnikov, vsak spremeni svoj izhod takoj, ko njegov vhod DATA to zahteva. Če se signali DATA1, DATA2, … (kolikor jih pač imamo) ne spremenijo istočasno, se tudi spremembe izhodov ne bodo zgodile istočasno. Zaradi te pomanjkljivosti vpeljemo pomnilne celice.

Flip-flop D oz. lepše rečeno pomnilna celica D je element, ki je po svoji funkciji podoben zadrževalniku D, le da njegov izhod reagira na spremembo vhoda le v določenem trenutku. To je lahko takrat, ko se vhod ENABLE spremeni iz nič v ena (pozitivna fronta) ali pa takrat, ko se vhod ENABLE spremeni iz ena v nič (negativna fronta). Na vhod ENABLE v praksi pošljemo enakomerno zaporedje pozitivnih in negativnih front. To zaporedje tvori oscilator, ki ga imenujemo ura, vhod ENABLE pa zato označimo kot CLOCK.

Pomnilno celico D lahko zgradimo na različne načine, najenostavnejša je tako imenovana „Master-Slave“ izvedba, pri kateri vežemo dva zadrževalnika D zaporedoma.



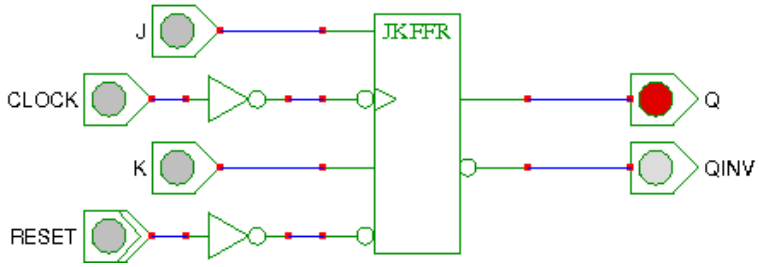
Delovanje pomnilne celice D opišemo s karakteristično tabelo, v kateri podamo naslednjo vrednost izhoda Q glede na trenutno vrednost izhoda Q in vrednost vhoda DATA, ki ga označimo na kratko z D.

Q (trenutna vrednost)	**D (vrednost DATA)**	**Q+ (naslednja vrednost)**
0	0	0
0	1	1
1	0	0
1	1	1

Pri analizi sekvenčnega vezja nam bolj kot karakteristična tabela pride prav karakteristična enačba, ki jo dobimo iz te tabele:

Q+ = D

Poleg pomnilne celice D se v praksi pogosto uporabljata tudi nekoliko kompleksnejša pomnilna celica JK, ki ima dva vhoda.



Karakteristična tabela za pomnilno celico JK je naslednja:

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

Karakteristična enačba za pomnilno celico JK je naslednja:

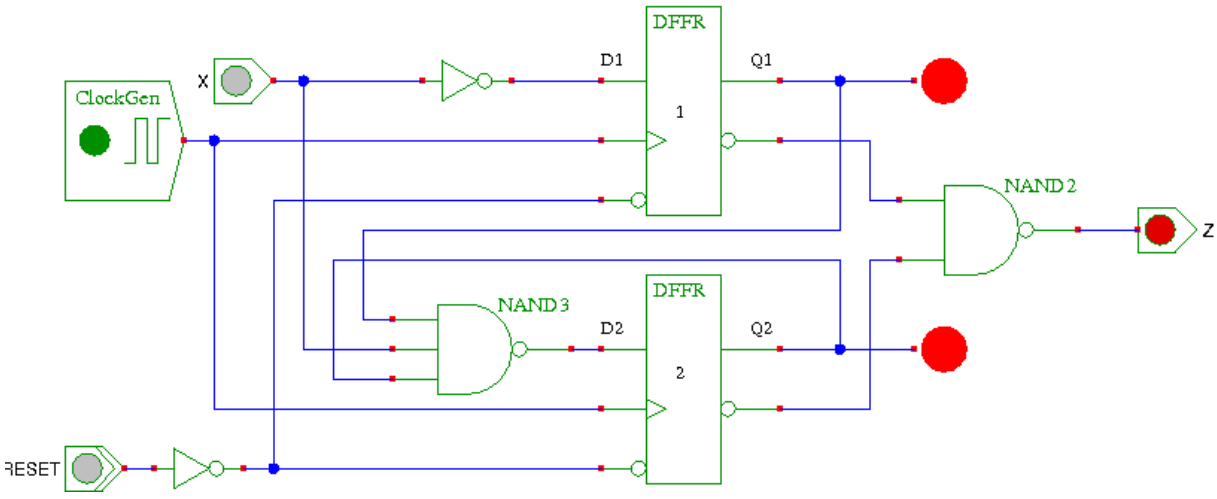
Q+ = J * Q' + K' * Q

Analiza sekvenčnega vezja poteka v splošnem poteka v naslednjih korakih:

1. oštevilčimo pomnilne celice
2. zapišemo enačbe vhodov
3. s pomočjo karakterističnih enačb določimo enačbe naslednjega stanja
4. določimo enačbe morebitnih izhodov
5. napišemo tabelo prehajanja stanj
6. narišemo diagram prehajanja stanj

VAJA 9.1

Analiziraj sekvenčno vezje podano na spodnji sliki. Po inicializaciji pomnilnih celic s signalom RESET (začetno stanje) sta izhoda Q1 in Q2 oba na nič.



Pomnilne celice so že oštevilčene.

Zapišemo enačbe vhodov:

D1 = X'

D2 = Q1 | X | Q2

S pomočjo karakterističnih enačb določimo enačbe naslednjega stanja:

Q1+ = D1 = X'

Q2+ = D2 = Q1 | X | Q2

Določimo enačbe izhodov:

Z = Q1' | Q2' = Q1 + Q2

Napišemo tabelo prehajanja stanj. Najprej vpišemo enačbi naslednjega stanja in enačbo izhoda v pravilnostno tabelo, nato poimenujemo stanja in na koncu pravilnostno tabelo spremenimo v tabelo prehajanja stanj.

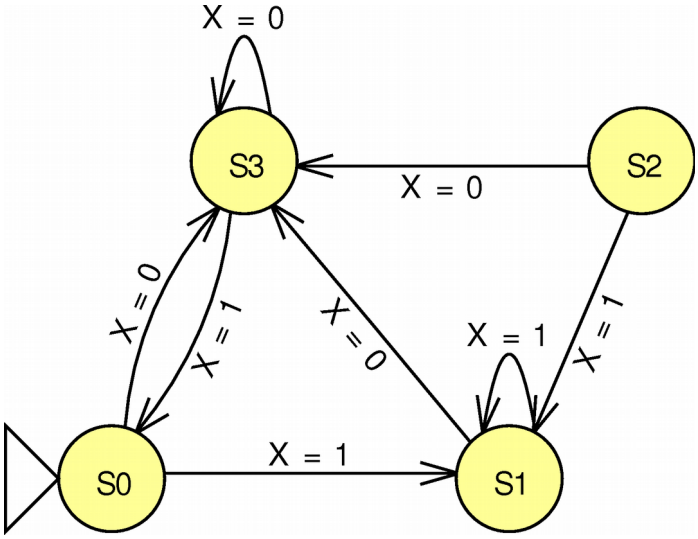
Q1	Q2	X	Q1+	Q2+	Z
0	0	0	1	1	0
0	0	1	0	1	0
0	1	0	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	0	1

Stanja poimenujmo po vrsti, torej 00 = S0, 01 = S1, 10 = S2 in 11 = S3.

Dobimo naslednjo tabelo prehajanja stanj:

Sedanje stanje	Vhod X	Naslednje stanje	Izhod Z
S0	0	S3	0
S0	1	S1	0
S1	0	S3	1
S1	1	S1	1
S2	0	S3	1
S2	1	S1	1
S3	0	S3	1
S3	1	S0	1

Narišemo diagram prehajanja stanj. Ta diagram ne prikazuje obnašanja izhodov.



10. Sinteza sekvenčnih vezij

Sinteza sekvenčnega vezja poteka v obratnem vrstnem redu kot analiza:

1. napišemo tabelo prehajanja stanj, če je to potrebno
2. napišemo pripadajočo pravilnostno tabelo
3. s pomočjo vzbujevalnih tabel določimo vrednosti vhodov
4. zapišemo enačbe vhodov
5. pretvorimo dobljene enačbe v gradnike, ki so na voljo
6. narišemo vezje

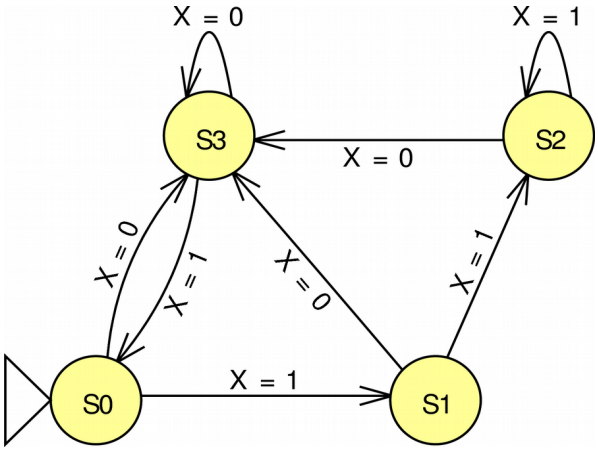
Vzbujevalne tabele, ki jih potrebujemo pri sintezi sekvenčnega vezja, dobimo s preoblikovanjem karakterističnih tabel za posamezne pomnilne celice.

Q (trenutna vrednost)	**Q+ (naslednja vrednost)**	**D**	**JK**
0	0	0	0X
0	1	1	1X
1	0	0	X1
1	1	1	X0

Prvo vrstico vzbujevalne tabelo preberemo takole. Če je trenutna vrednost izhoda Q enaka nič in želimo, da bo naslednja vrednost izhoda tudi enaka nič, potem moramo pri pomnilni celici D na vhod pripeljati nič, pri pomnilni celici JK pa moramo na vhod J pripeljati nič, na vhodu K pa je lahko karkoli. Podobno preberemo ostale vrstice.

VAJA 10.1

Samo z uporabo pomnilnih celic D in dvo-vhodnih Shefferjevih elementov tvori sekvenčno vezje, ki ima en vhod in se obnaša v skladu s podanim diagramom prehajanja stanj. Stanja kodirajte po vrsti, torej S = 00, S1 = 01, S2 = 10 in S3 = 11. Vezje naj ima dva izhoda, na katerih lahko odčitamo kodo trenutnega stanja.



Zapišemo tabelo prehajanja stanj:

Sedanje stanje	Vhod X	Naslednje stanje
S0	0	S3
S0	1	S1
S1	0	S3
S1	1	S2
S2	0	S3
S2	1	S2
S3	0	S3
S3	1	S0

Če upoštevamo podano kodiranje stanj, dobimo naslednjo pravilnostno tabelo:

Q1	Q2	X	Q1+	Q2+
0	0	0	1	1
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Sedaj v tej tabeli dodamo stolpca, ki ponazarjata vrednosti vhodov obeh pomnilnih celic. Ker gre za pomnilni celici D ta korak pravzaprav niti ni potreben, kajti stolpca D1 in D2 sta enaka stolpcema Q1+ oz. Q2+.

Q1	Q2	X	Q1+	Q2+	D1	D2
0	0	0	1	1	1	1
0	0	1	0	1	0	1
0	1	0	1	1	1	1
0	1	1	1	0	1	0
1	0	0	1	1	1	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0

Stolpca D1 in D2 vnesemo v Karnaugh-jeva diagrama, da določimo minimalno obliko.

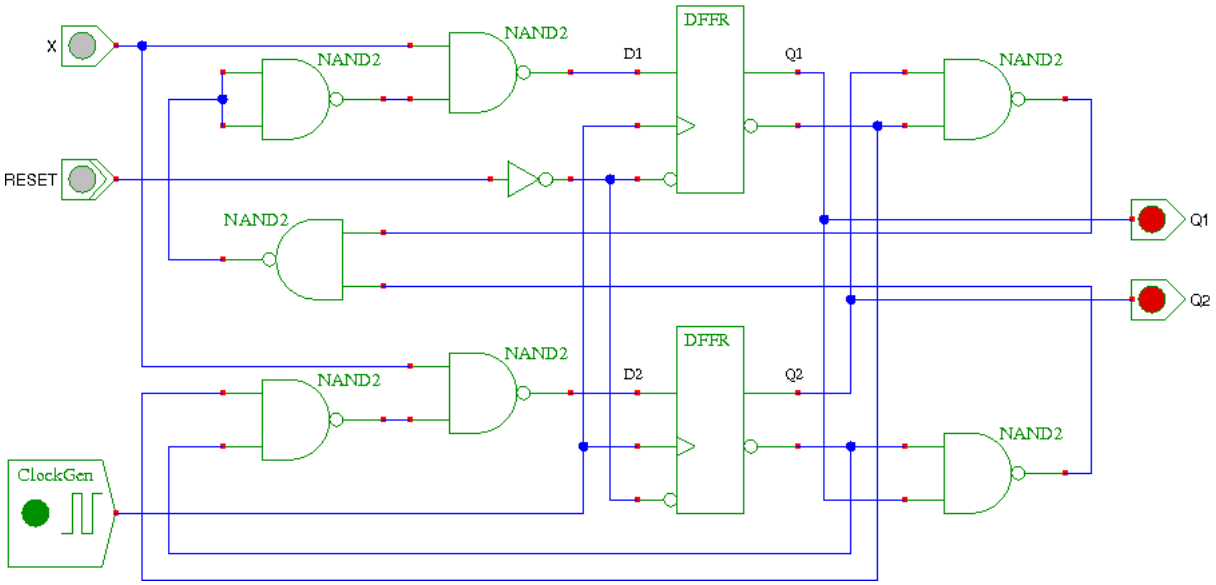
Q1,Q2 X	00	01	11	10
0	1	1	1	1
1	0	1	0	1

Q1,Q2 X	00	01	11	10
0	1	1	1	1
1	1	0	0	0

Iz Karnaugh-jevih diagramov razberemo enačbi vhodov in ju spremenimo tako, da bosta funkciji izvedeni z dvo-vhodnimi Shefferjevimi elementi.

D1 = X'+ Q1' * Q2 + Q1 * Q2' = X | ((Q1' | Q2) | (Q1 | Q2'))'
D2 = X' + Q1' * Q2' = X | (Q1' | Q2')

Sedaj lahko narišemo vezje, v katerem bomo uporabili pomnilne celice D z dodatnim vhodom RESET.



VAJA 10.2

Kako bi vezje iz prejšnje naloge izgledalo, če bi ga tvorili samo z uporabo pomnilnih celic JK in dvo-vhodnih Shefferjevih elementov?

Postopek sinteze se začne enako kot prej, le da v pravilnostno tabelo sedaj dodamo štiri stolpce, ki ponazarjajo vrednosti vhodov obeh pomnilnih celic.

Q1	Q2	X	Q1+	Q2+	J1	K1	J2	K2
0	0	0	1	1	1	X	1	X
0	0	1	0	1	0	X	1	X
0	1	0	1	1	1	X	X	0
0	1	1	1	0	1	X	X	1
1	0	0	1	1	X	0	1	X
1	0	1	1	0	X	0	0	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Stolpce J1, K1, J2 in K2 vnesemo v Karnaugh-jeve diagrame, da določimo minimalno obliko.

Q1,Q2 X	00	01	11	10
0	1	1	X	X
1	0	1	X	X

Q1,Q2 X	00	01	11	10
0	X	X	0	0
1	X	X	1	0

Q1,Q2 X	00	01	11	10
0	1	X	X	1
1	1	X	X	0

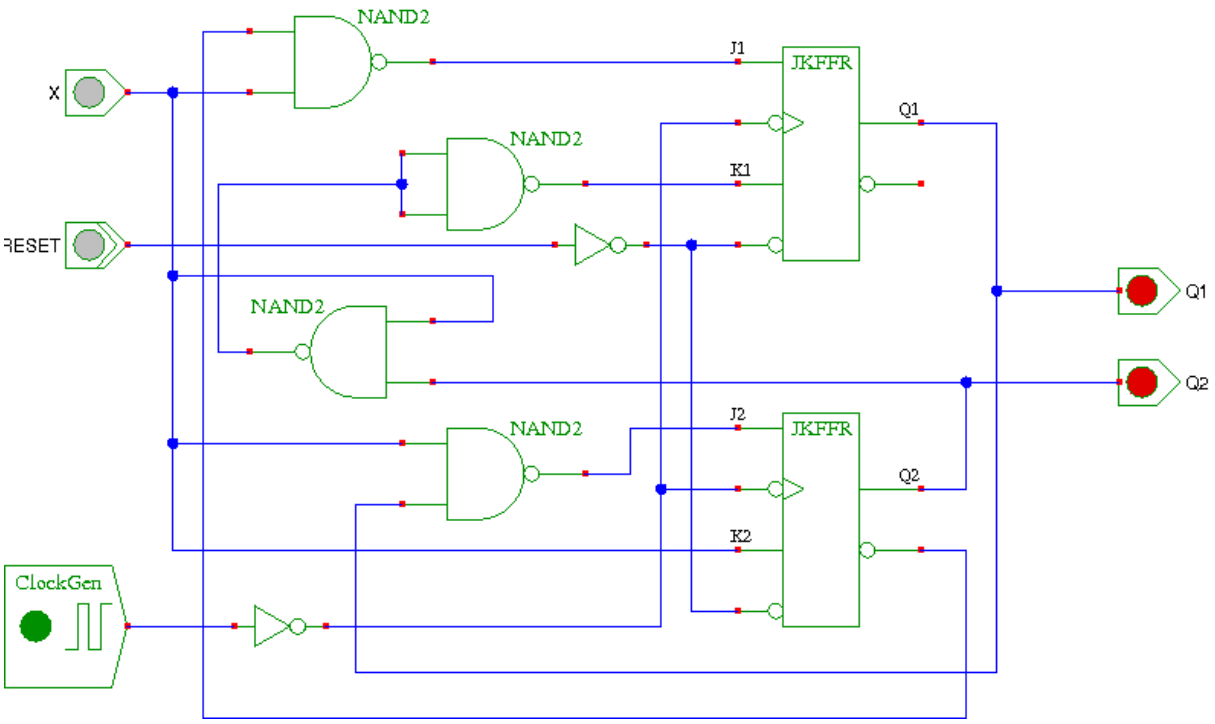
Q1,Q2 X	00	01	11	10
0	X	0	0	X
1	X	1	1	X

Iz Karnaugh-jevih diagramov razberemo enačbe vhodov in jih spremenimo tako, da bodo funkcije izvedene z dvo-vhodnimi Shefferjevimi elementi.

J1 = X' + Q2 = X | Q2'
K1 = X * Q2 = (X | Q2)'
J2 = X' + Q1' = X | Q1
K2 = X

Ko uporabimo pomnilne celice JK dobimo preprostejše funkcije, so pa štiri namesto dveh.

Sedaj lahko narišemo vezje, v katerem bomo uporabili pomnilne celice JK z dodatnim vhodom RESET.



11. Mealyjevi in Moorovi avtomati, izvedba s sekvenčnim vezjem

Diagram prehajanja stanj je dobra ponazoritev delovanja sekvenčnega vezja le, če izhodi iz vezja niso kompleksne funkcije. V diagramu prehajanja stanj namreč izhodi sploh niso prikazani. To ni problem, če so npr. izhodi enaki kodi trenutnega stanja (npr. pri števniku) – v tem primeru so izhodni signali kar izhodi pomnilnih celic.

Pri bolj kompleksnih izhodih pa sekvenčno vezje namesto z diagramom predstavimo v obliki Mealyjevega ali pa Moorovega avtomata. Za nekatere naloge je bolj primeren eden, za nekatere naloge pa drugi. V vsakem primeru lahko Mealyjev avtomat pretvorimo v Moorovega in obratno. Po drugi strani pa se lahko tudi zgodi, da avtomat, ki si ga zamislimo, ni minimalni (to pomeni, da bi lahko imel manj stanj).

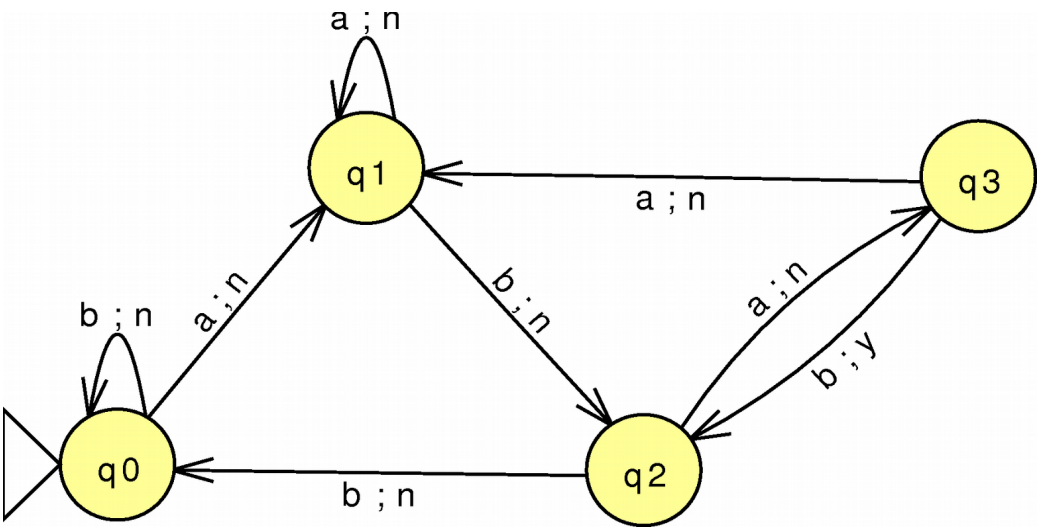
V splošnem velja tudi tole:

- Minimalni Mealyjev avtomat ima vedno manj ali enako stanj kot ekvivalentni minimalni Moorov avtomat,
- Moorov avtomat običajno vodi do manjšega vezja kot Mealyjev avtomat z enakim številom stanj.

VAJA 11.1

Tvori sekvenčno vezje za detektor sekvence, ki na vhodu dobi zaporedje simbolov „a“ in „b“, na izhodu pa generira zaporedje simbolov „n“ in „y“. Detektor sekvence naj reagira na vhodno zaporedje „abab“. Na izhodu naj bo „y“ le takrat, ko je na vhodu zaznano podano zaporedje, drugače pa mora biti na izhodu „n“.

Glede na besedilo najprej tvorimo ustrezni Mealyjev avtomat.



S tabelo ta avtomat predstavimo takole:

Trenutno stanje	Vhod „a“	Vhod „b“
q0	„n“ / q1	„n“ / q0
q1	„n“ / q1	„n“ / q2
q2	„n“ / q3	„n“ / q1
q3	„n“ / q1	„y“ / q2

Da bi Mealyjev avtomat lahko izvedli z vezjem, moramo kodirati stanja, vhode in izhode.

- Imamo štiri stanja, dovolj bo dvo-bitna koda, izberimo q0=00, q1=01, q2=10, q3=11.
- Imamo dva vhoda, dovolj bo eno-bitna koda, izberimo a=0, b=1.
- Imamo dva izhoda, dovolj bo eno-bitna koda, izberimo n=0, y=1.

Po kodiranju lahko zapišemo tabelo prehajanja stanj. Stanja označimo z Q1 in Q2, vhod z X in izhod z Z.

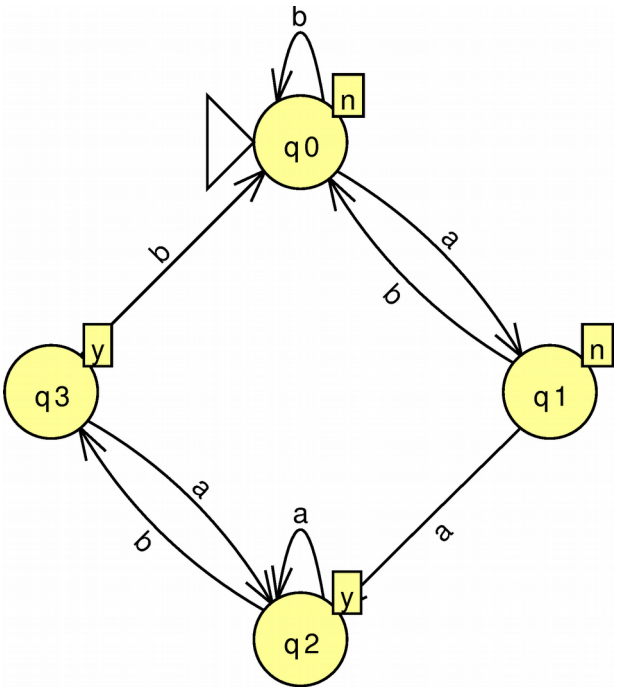
Q1	Q2	X	Q1+	Q2+	Z
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	0	1

Od tod naprej nalogo rešujemo tako, kot je bilo prikazano v poglavju o sintezi sekvenčnih vezij.

VAJA 11.2

Tudi ta naloga zahteva tvorjenje sekvenčnega vezja, ki deluje kot detektor sekvence. Na vhodu naj dobi zaporedje simbolov „a“ in „b“, na izhodu pa tvori zaporedje simbolov „n“ in „y“. Detektor sekvence tokrat reagira na zaporedje „aa“. Ko se to zaporedje pojavi, se izhod postavi na „y“ in tak ostane do takrat, ko se na vhodu pojavi zaporedje „bb“.

Tokrat bomo tvorili Moorov avtomat.



S tabelo ta avtomat predstavimo takole:

Trenutno stanje	Vhod „a“	Vhod „b“	Izhod
q0	q1	q0	„n“
q1	q2	q0	„n“
q2	q2	q3	„y“
q3	q2	q0	„y“

Da bi Moorov avtomat lahko izvedli z vezjem, moramo kodirati stanja, vhode in izhode.

- Imamo štiri stanja, dovolj bo dvo-bitna koda, izberimo q0=00, q1=01, q2=10, q3=11.
- Imamo dva vhoda, dovolj bo eno-bitna koda, izberimo a=0, b=1.
- Imamo dva izhoda, dovolj bo eno-bitna koda, izberimo n=0, y=1.

Po kodiranju lahko zapišemo tabelo prehajanja stanj. Stanja označimo z Q1 in Q2, vhod z X in izhod z Z.

Q1	Q2	X	Q1+	Q2+	Z
0	0	0	0	1	0
0	0	1	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0

1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	0	0	1

Od tod naprej nalogo spet rešujemo tako, kot je bilo prikazano v poglavju o sintezi sekvenčnih vezij.

12. Pretvorba avtomatov, minimizacija avtomatov

Gradivo še ni na voljo v elektronski obliki.

13. Priprave na drugi kolokvij

Snov drugega kolokvija so multipleksor, sekvenčna vezja in avtomati.

Dodatno poglavje 1: Računalniška predstavitev in obdelava Boolovih funkcij

Boolove funkcije računalniški programi interno predstavijo z eno od kanoničnih oblik. Primerni sta npr. PDNO in PKNO. Leta 1990 so američani **Karl S. Brace**, **Richard L. Rudell** in **Randal E. Bryant** objavili zelo odmeven članek "Efficient Implementation of a BDD Package", v katerem so opisali uporabo ***binarnih odločitvenih grafov (Binary Decision Diagrams, BDD)*** za učinkovito predstavitev in obdelavo Boolovih funkcij. O podobni strukturi je sicer že leta 1959 pisal američan **C. Y. Lee** ter nato leta 1976 rus **R. Ubar** in američan **Raymond T. Boute**. Z današnjim imenom je BDD-je leta 1978 poimenoval američan **Sheldon B. Akers**. Letnice in imena raziskovalcev smo tukaj navedli predvsem zato, da poudarimo, kako dandanes pomembna odkritja niso slučajne domislice „laikov“ in da ne nastanejo čez noč. Randal E. Bryant, ki se je med omenjenimi najbolj aktivno trudil za uveljavitev BDD-jev, je npr. ugledni profesor na Univerzi Carnegie Mellon v Pittsburghu v ZDA.

Binarni odločitveni graf

BDD-ji so rekurzivna predstavitev Boolovih funkcij. Temeljijo na Shannonovem razčlenitvenem zakonu:

F(x1,x2,...,xi,...xn) = xi * F(x1,x2,...,1,...xn) + xi' * F(x1,x2,...,0,...xn)
= ITE(xi, F(x1,x2,...,1,...xn), F(x1,x2,...,0,...xn))

Zgornji zapis je hkrati definicija tri-členega operatorja ITE.
Če v funkciji F izberemo vrstni red spremenljivk in jo nato rekurzivno razčlenimo po vseh spremenljivkah, dobimo kanonično obliko predstavitve. Če dobljeni izraz brez redundantnih stanj predstavimo v obliki binarnega acikličnega grafa,dobimo BDD za funkcijo F. Bolj natančno, dobimo ROBDD (Reduced Ordered BDD), ki je danes ena najbolj popularnih oblik predstavitve Boolove funkcije v računalniku.

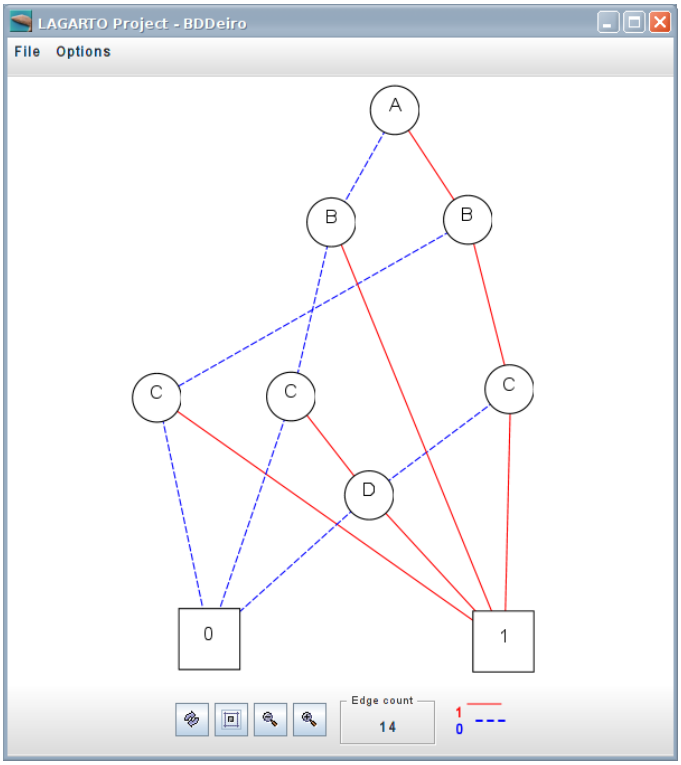
VAJA D1.1

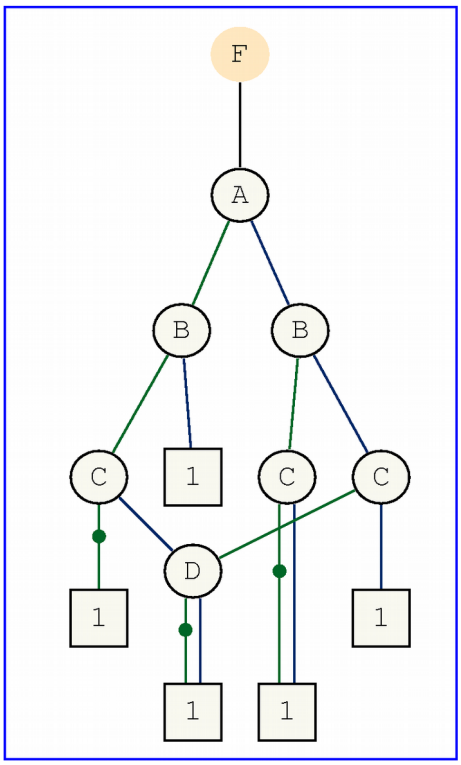
Boolovo funkcijo F = A'·B + B·D + A·C + C·D predstavi z BDD-jem.

Izberemo vrstni red spremenljivk A,B,C,D in rekurzivno razčlenimo funkcijo.

F = A'·B + B·D + A·C + C·D =
= ITE(A, B·D + C + C·D, B + B·D + C·D) =
= ITE(A, ITE(B, D + C + C·D, C + C·D), ITE(B, 1, C·D)) =
= ITE(A, ITE(B, ITE(C,1,D), ITE(C,1 0)), ITE(B, 1, ITE(C,D,0))) =
= ITE(A, ITE(B, ITE(C, 1, ITE(D,1,0)), ITE(C,1,0)), ITE(B, 1, ITE(C, ITE(D,1,0), 0)))

V nadaljevanju je na levi sliki prikazan ROBDD, ki ga iz te formule dobimo tako, da za vsak člen ITE(x,T,E) narišemo vozlišče, v katerem je spremenljivka x, formulo T rekurzivno narišemo kot desnega naslednika tega vozlišča, formulo E pa rekurzivno narišemo kot levega naslednika tega vozlišča. Na desni sliki spodaj je prikazana optimizacija tega BDD-ja s pomočjo označenih povezav. Oznaka na povezavi pomeni negacijo. Oznak ne smemo poljubno uporabljati, ker bi s tem pokvarili kanoničnost predstavitve. Podrobnejšo razlago najdete v članku iz leta 1990, ki je bil omenjen v uvodu.





Na razvoj orodij za računalniško obdelavo logičnih funkcij so pomebno vplivali projekti na Univerzi v Berkeleyu v začetku 90-ih let 20. stoletja. Izpostavimo lahko program za minimizacijo Boolovih funkcij Espresso in dokaj obsežen programski paket za sintezo sekvenčnih vezij SIS.

Formati za zapis Boolovih funkcij s tekstovno datoteko

Boolovo funkcijo lahko s tekstovno datoteko predstavimo na različne načine. Najpreprosteje je, če jo zapišemo v obliki Boolove formule. Uporabimo lahko infiksno ali pa prefiksno obliko. Izbrati je potrebno oznake za operatorje.

Primer infiksne oblika je format EQN iz orodja SIS. Pri njem uporabljamo znak `!` ali `'` za negacijo, znak `*` za konjunkcijo, znak `+` za disjunkcijo in znak `^` za ekskluzivno vsoto. Vsako formulo zaključimo s podpičjem. Tukaj je primer dveh formul:

```
f = c + !a * b ;
g = !(f + d);
```

Primer prefiksne oblike je format, ki ga uporablja orodje BDD Scout. Operatorje označimo z oznakami NOT, AND, OR in EXOR. Zgornji formuli bi zapisali takole:

```
f = (OR c (AND (NOT a) b))
g = (NOT (or f d))
```

Orodja razvita na Univerzi v Berkeleyu so vpeljala format PLA (Programmable Logic Array) in format BLIF (Berkeley Logic Interchange Format).

Pri formatu PLA najprej podamo število spremenljivk, nato število funkcij, nato pa še imena spremenljivk in imena funkcij. Sledi vrstica, ki pomeni število vrstic v podani pravilnostni tabeli in nato pravilnostna tabela. Za vse funkcije uporabimo isto tabelo.

```
.i 4
.o 2
.ilb a b c d
.ob f g
.p 16
0000 01
0001 00
0010 10
0011 10
0100 10
0101 10
0110 10
0111 10
1000 01
1001 00
1010 10
1011 10
1100 01
1101 00
1110 10
1111 10
.e
```

Opis pravilnostne tabele lahko skrčimo tako, da združimo vrstice. Za podan primer nam orodje Espresso izračuna naslednjo optimalno obliko:

```
.i 4
.o 2
.ilb a b c d
.ob f g
.p 4
-000 01
1-00 01
01-- 10
--1- 10
.e
```

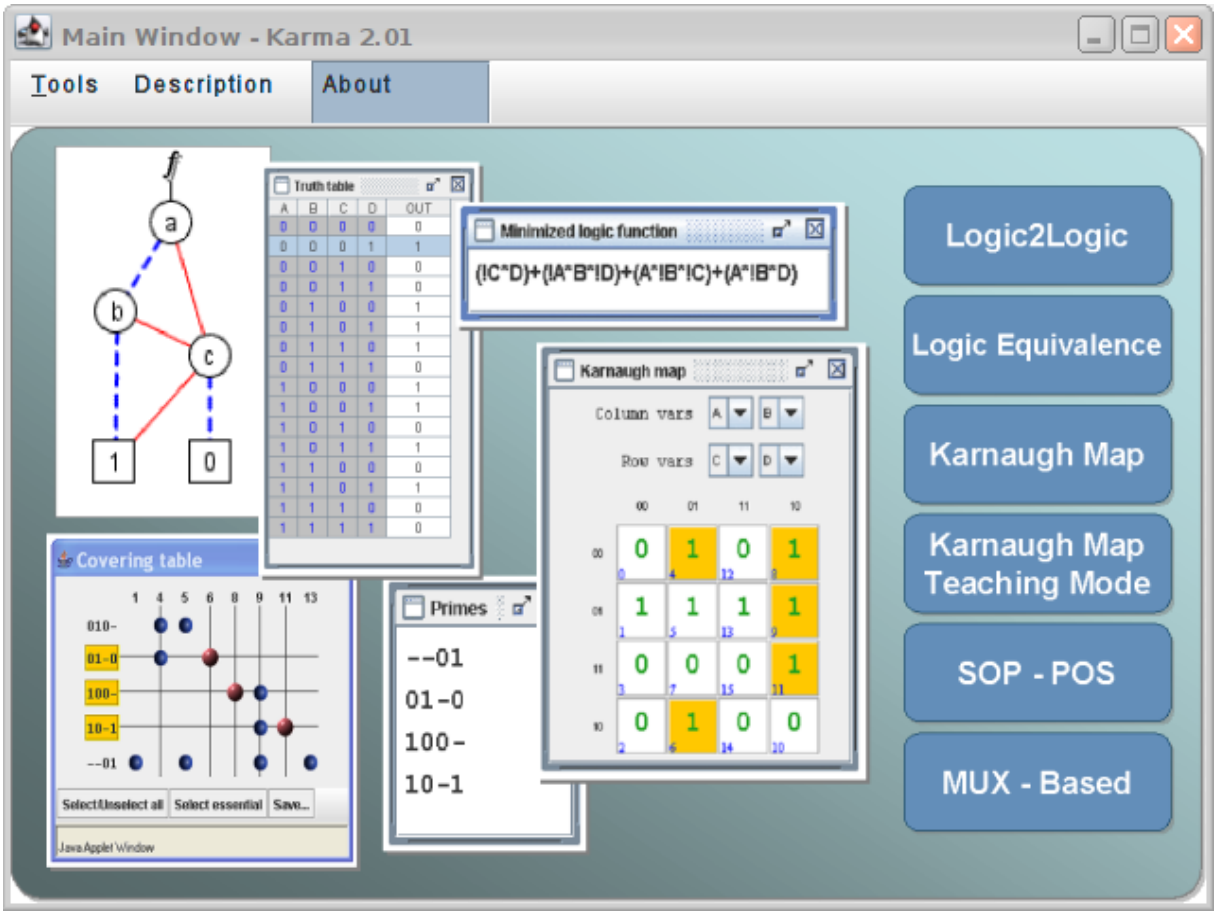
Format BLIF je na prvi pogled podoben. Najprej podamo ime modela, naštejemo spremenljivke in imena funkcij (istočasno lahko podamo več funkcij). Nato za vsako funkcijo podamo spremenljivke, od katerih je odvisna in njeno ime. Sledijo vrstice z oznakami produktov iz MDNO (pravzaprav ni nujno, da je minimalna) tako, da oznaka vsebuje „1“ za spremenljivke v pozitivni obliki, „0“ za spremenljivke v negirani obliki in „-“ za spremenljivke, ki ne nastopajo v produktu. Na koncu vrstice napišemo enico. Za razliko od formata PLA tukaj vsako funkcijo podamo posebej, lahko pa rezultat ene funkcije uporabimo kot vhod v drugo funkcijo.

Ker velja !(f + d) = !f * !d formuli iz zgornjega primera opišemo takole:

```
.model primer
.inputs a b c d
.outputs f g
.names a b c f
--1 1
01- 1
.names f d g
00 1
.end
```

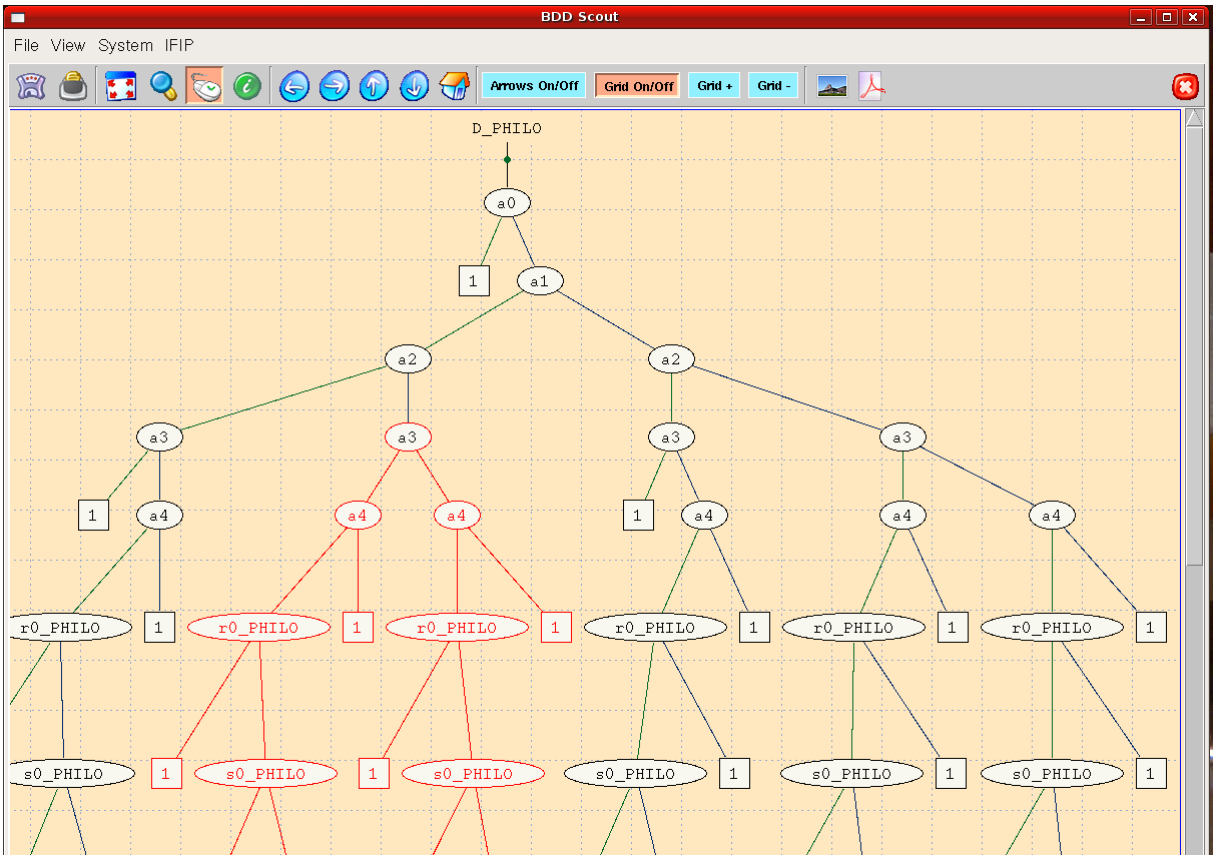
KARMA

Karma je program za obdelavo Boolovih funkcij, ki ga razvijajo na Univerzi Rio Grande do Sul v Braziliji. Program je napisan v Javi in ima licenco GPL. Program je v razvoju in v trenutni verziji 3.1 so nekatere funkcije še pomanjkljivo izvedene oz. celo manjkajo. Karma je pravzaprav zbirka orodij, med katerimi je še posebej zanimivo Karma Teaching Mode. To orodje vsebuje razne vaje, s katerimi se učimo minimizacije Boolove funkcije s pomočjo Karnaugh-jevih diagramov. Razen tega orodja zna Karma pretvarjati med raznimi formati za zapis Boolovih funkcij (BLIF, …), poiskati MDNO in MKNO za dano Boolovo funkcijo (zna upoštevati tudi DCS), prikazati potek Quine-McCluskey-evega algoritma za minimizacijo dane Boolove funkcije in prikazati BDD za dano Boolovo funkcijo.



BDD Scout

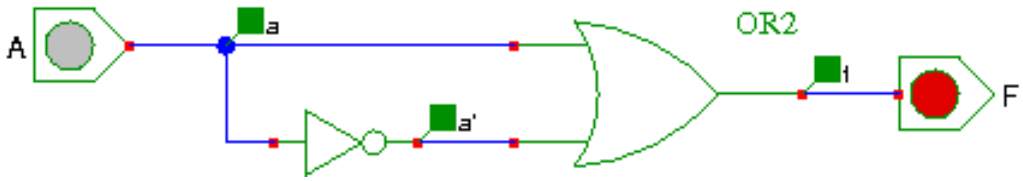
BDD Scout je program za obdelavo Boolovih funkcij, ki ga razvijamo na FERI na Univerzi v Mariboru. V trenutni verziji 1.0 je njegova funkcionalnost omejena le na prikaz BDD-jev za dano Boolovo funkcijo. Prav tako je omejen nabor vhodnih formatov, saj zna program prebrati le datoteke v formatu IFIP. BDD Scout je napisan v C-ju, grafični vmesnik pa je izveden kot Tcl/Tk skripta. Ima licenco GPL in deluje na sistemih GNU/Linux, MS Windows ter drugih.



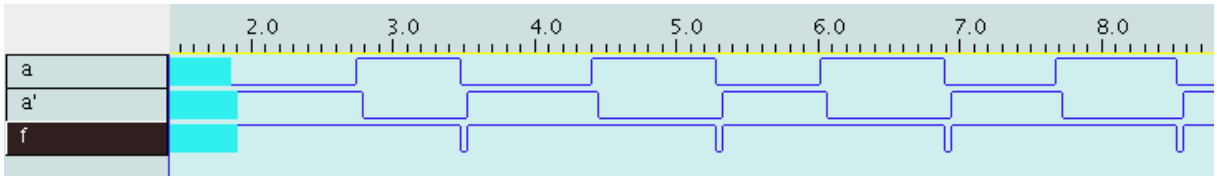
Dodatno poglavje 2: Statični hazardi

Statični hazardi so lastnost posamezne izvedbe logične funkcije.

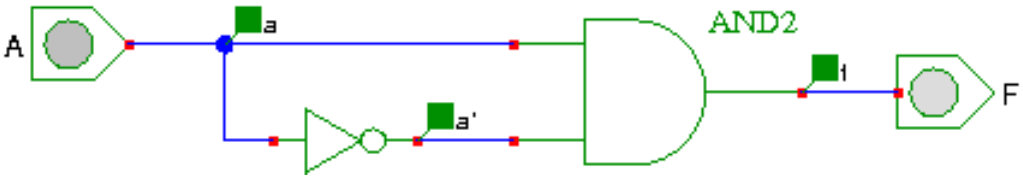
V Boolovi algebri velja (A+A')=1. Torej mora biti na izhodu naslednjega vezja vseskozi logična enica.



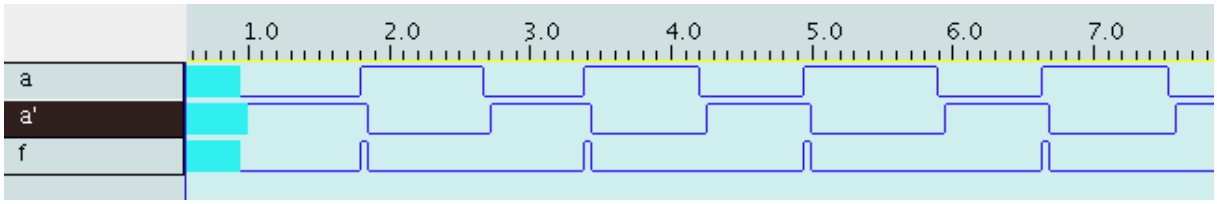
A simulacija pokaže, da v trenutkih, ko se vrednost A spremeni iz 1 v 0, izhod F zaniha iz 1 v 0 in takoj spet nazaj. To se zgodi le, če invertorju nastavimo neko zakasnitev. V praksi imajo vsa vrata zakasnitve, ki pa jih pri simulaciji ni potrebno upoštevati, če nas zanima le logična pravilnost vezja. Prikazani pojav imenujemo ***statični hazard-1***, saj bi vrednost morala biti vseskozi logična enica.



Če opazujemo vrata AND, ugotovimo podoben pojav. Ker v Boolovi algebri velja (A·A')=0, bi na izhodu naslednjega vezje morala biti vseskozi logična ničla. Vendar pa simulacija pokaže, da vrednost izhoda ob prehodu vrednosti A iz 0 v 1 zaradi zakasnitve invertorja za trenutek zaniha.



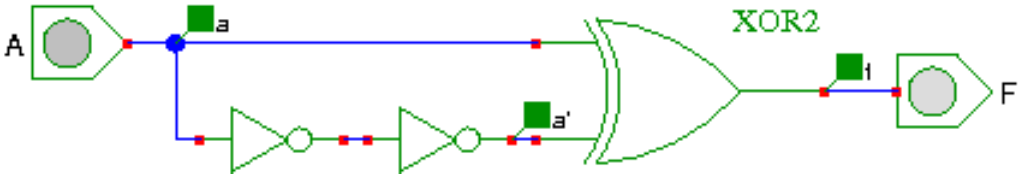
Prikazani pojav imenujemo ***statični hazard-0***, saj bi vrednost vseskozi morala biti logična ničla.

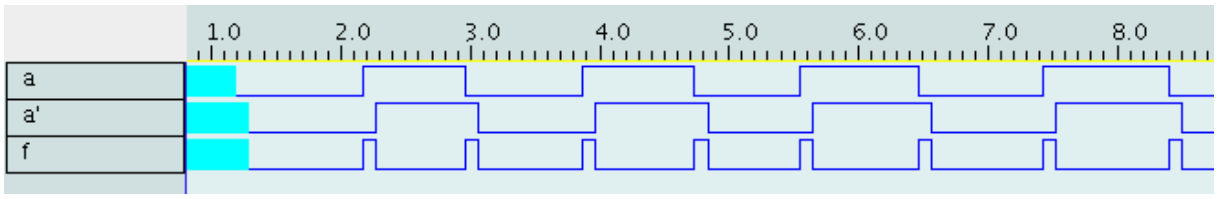


V splošnem poznamo več vrst hazardov, ki jih razvrščamo glede na vzrok ali pa posledico. Predpostavimo, da se lahko naenkrat (sočasno) spremeni le vrednost enega vhoda. Hazarde, ki se pojavljajo ob tej predpostavki, imenujemo ***logični hazardi***. Statični hazard-0 in statični hazard-1 torej spadata med logične hazarde. Poleg njiju se v kombinacijskih vezjih lahko pojavijo tudi nekoliko kompleksnejši logični hazardi imenovani dinamični hazardi.

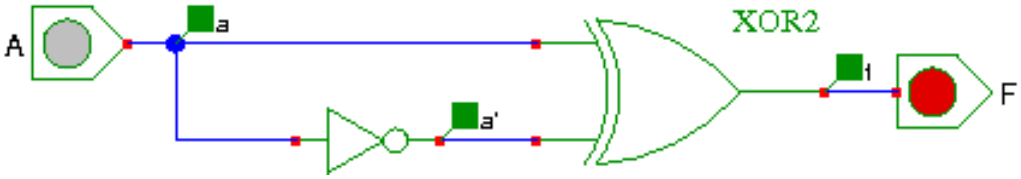
Na vratih OR se lahko pojavi le statični hazard-1, na vratih AND pa le statični hazard-0. Pri Shefferjevem elementu se tako, kot pri vratih AND, lahko pojavi le statični hazard-0. Pri Peircovem elementu se tako kot pri vratih OR lahko pojavi le statični hazard-1. Pri nekaterih drugih vratih pa se pojavljata celo obe vrsti statičnih hazardov. Kot primer navedimo vrata XOR.

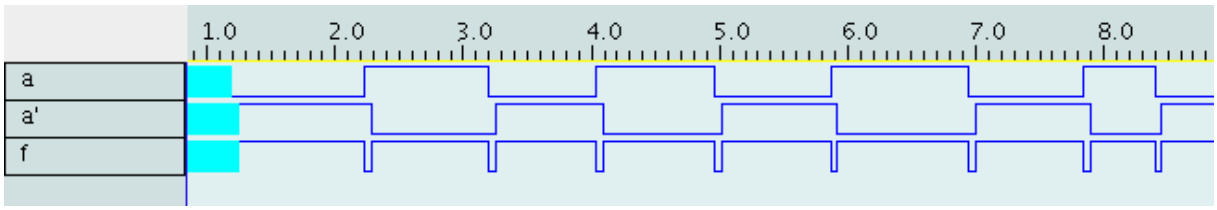
V Boolovi algebri velja A ⊕ A = 0 in A ⊕ A' = 1. Toda naslednji realizaciji teh dveh formul vodita do statičnih hazardov. V prvem primeru vezje vsebuje statični hazard-0.





V drugem primeru pa vezje vsebuje statični hazard-1.





V praksi imamo seveda več vhodov. V nadaljevanju je podan kratek opis dveh metod za odkrivanje hazardov. Opomba: če vam po prebranem opisu računalniške metode ne bo jasno kako deluje, nič hudega. Tukaj je opisana le za ilustracijo in je na vajah ne bomo uporabljali. Morate pa dobro razumeti računsko metodo za odkrivanje hazardov.

Računalniška metoda za odkrivanje hazardov

Izberemo en vhod v vezje, ki ga bomo spreminjali, vse ostale vhode pa postavimo na neko konstantno vrednost. Nato ugotovimo, če imamo v vezju kakšna vrata oz. kakšen element, katerega izhod zaniha ob spreminjanju izbranega vhoda. Pri tem ni nujno izvajati simulacije, saj npr. pri vratih AND z dvema vhodoma, možnost hazarda prepoznamo po tem, da je en vhod vrat enak izbranemu vhodu v vezje, drugi vhod pa ravno nasprotna vrednost. Problem računalniške metode je v tem, da je res primerna le za računalnik. Če imamo dva vhoda, moramo namreč pregledati $2*2^1 = 4$ kombinacije, pri treh vhodih $3*2^2 = 12$ kombinacij, pri štirih vhodih pa že $4*2^3 = 32$ kombinacij.

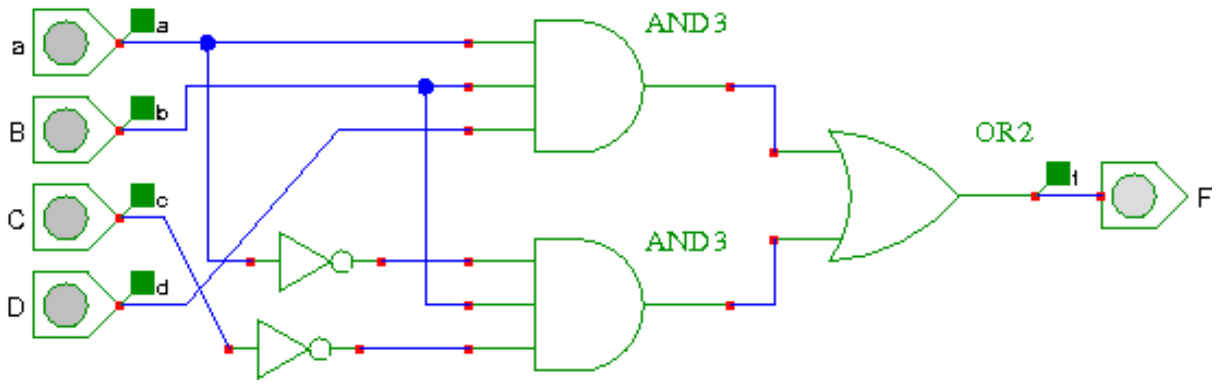
Računska metoda za odkrivanje hazardov

1. Zapiši logično funkcijo F, ki natančno predstavlja dano strukturo vezja.
2. S pomočjo postulatov pretvori funkcijo F v dvonivojsko formulo oblike vsota produktov, pri pretvorbi pa ne smeš uporabiti postulata P5b.
3. Na Karnaugh-jevem diagramu za funkcijo F s krogi označi vse produkte, ki nastopajo v dobljeni formuli (krožimo enice).
4. Če obstajata dva kroga, ki se dotikata po ploskvi in ne obstaja tretji krog, ki prekriva oba, potem v vezju obstaja statični hazard-1.
5. Ko poiščemo vse hazarde-1, nadaljujemo tako, da s pomočjo postulatov in izrekov pretvorimo funkcijo F v dvonivojsko formulo oblike produkt vsot, pri pretvorbi ne smemo uporabiti postulata P5a.
6. Na Karnaugh-jevem diagramu za funkcijo F potem s krogi označimo vse vsote, ki nastopajo v dobljeni formuli (krožimo ničle).
7. Če sedaj obstajata dva kroga, ki se dotikata po ploskvi in ne obstaja tretji krog, ki prekriva oba, potem v vezju obstaja statični hazard-0.
8. Če se v dvonivojski formuli oblike vsota produktov ne pojavijo členi, ki so glede na postulat P5b enaki 0, potem iskanja statičnih hazardov-0 ni treba izvesti, ker ne obstajajo.
9. Če ima vezje več izhodov, poiščemo statične hazarde za vsakega od izhodov posebej.

Pri pretvorbi v dvonivojsko formulo oblike uporabimo:
A | B = A' + B'
A ↓ B = A'·B'
A → B = A' + B
A ⩛ B = A·B' + A'·B = (A+B) · (A'+B')

Ni nujno, da se v realnem vezju pojavljajo vsi statični hazardi, ki smo jih izračunali z računsko metodo. Pojav hazarda je namreč odvisen od zakasnitev uporabljenih vrat in elementov. Možno je, da se ob določeni razporeditvi zakasnitev ne pojavi nobeden od izračunanih hazardov, pri drugačni razporeditvi zakasnitev pa vsi izračunani. Ne moremo pa samo s spreminjanjem zakasnitev dobiti takšnega hazarda, ki ga računska metoda ni napovedala.

VAJA D2.1
Poišči statične hazarde v danem vezju.



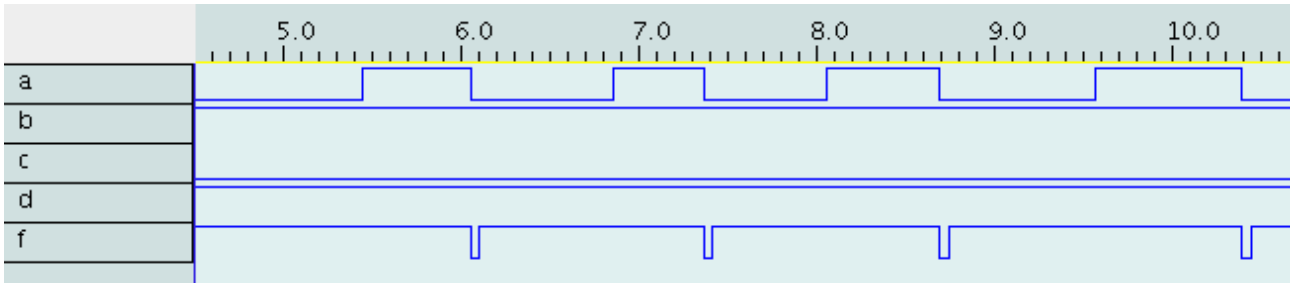
Najprej zapišemo logično funkcijo za izhod F tako, da natančno predstavimo strukturo danega vezja. Dobimo:

F = A·B·D + A·'B·C'

Ker je dobljena formula že v ustrezni dvonivojski obliki, preidemo kar na risanje krogov v Karnaugh-jevem diagramu.

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

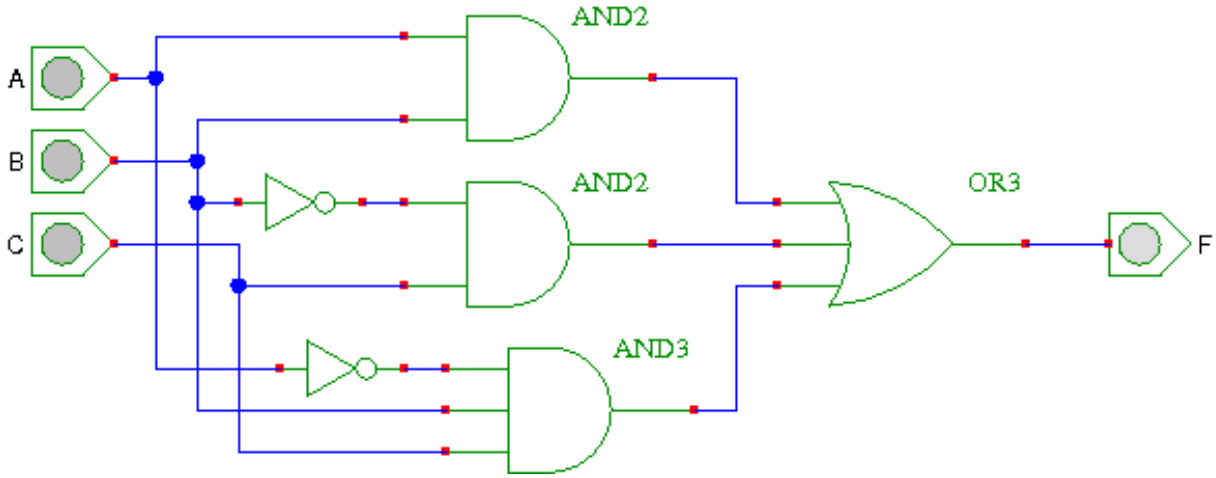
Iz slike lahko razberemo, da v vezju obstaja en statični hazard-1 in sicer lahko izhod zaniha. ko je B=1, C=0, D=1, A pa se spreminja. Ta hazard zaznamo v simulatorju, če zgornjemu invertorju nastavimo neko zakasnitev.



Ker ima podano vezje dvonivojsko strukturo oblike vsota produktov, ne vsebuje statičnih hazardov-0.

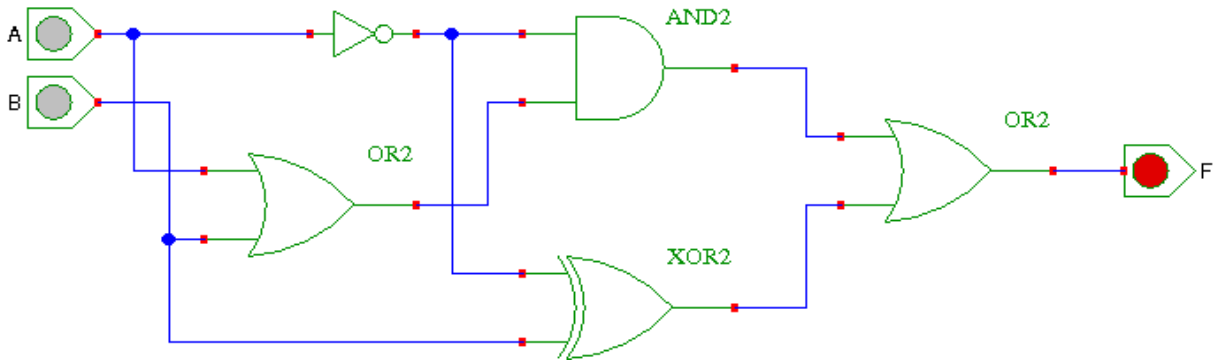
VAJA D2.2

Poišči statične hazarde v danem vezju.



Rešite sami.

VAJA D2.3
Poišči statične hazarde v danem vezju.



Najprej zapišemo logično funkcijo za izhod F tako, da natančno predstavimo strukturo danega vezja. Dobimo:

F = A'·(A+B) + (A' ⊻ B)

Funkcijo pretvorimo v dvonivojsko formulo oblike vsota produktov.

F = A'·(A+B) + (A' ⊻ B) = A'·A + A'·B + A'·B' + A·B

Prvi produkt je enak 0, vendar ga nismo odstranili, ker bi to zahtevalo uporabo postulata P5b. Obstoj tega člena kaže na možnost obstoja statičnega hazarda-0. Pri risanju krogov okoli enic takšnih členov ne upoštevamo.

A \ B	0	1
0	1	0
1	1	1

Iz slike lahko razberemo, da v vezju obstajata dva statična hazarda-1:
- ko je A=0 in se B spreminja
- ko je B=1 in se A spreminja

Sedaj funkcijo pretvorimo v dvonivojsko formulo oblike produkt vsot.

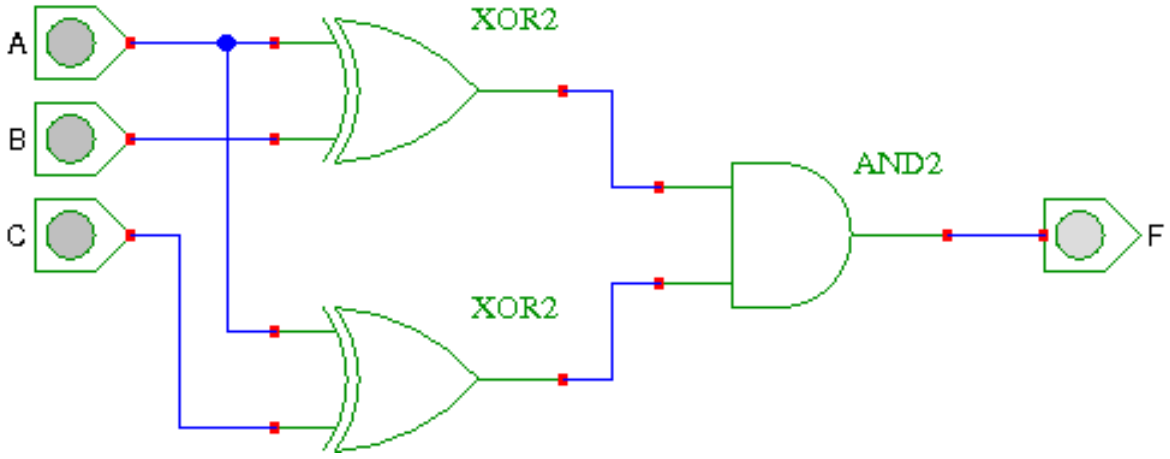
F = A'·(A+B) + (A' ⊻ B) = A'·(A+B) + (A'+B) · (A+B') =
= (A'·(A+B) + (A'+B)) · (A'·(A+B) + (A+B')) =
= (A'+A'+B) · (A+B+A'+B) · (A'+A+B') · (A+B+A+B') =
= (A'+B) · (A+A'+B) · (A+A'+B') · (A+B+B')

Ker imamo samo en krog, vezje gotovo ne vsebuje statičnih hazardov-0.

A B	0	1
0	1	0
1	1	1

VAJA D2.4

Poišči statične hazarde v danem vezju.



Rešite sami.