Дигитална логика и системи

4

XOR.
Грејов код.
Карнови мапи (Karnaugh Mapping)
Quine-McCluskey Tabular Method

Доц. д-р Никола Рендевски

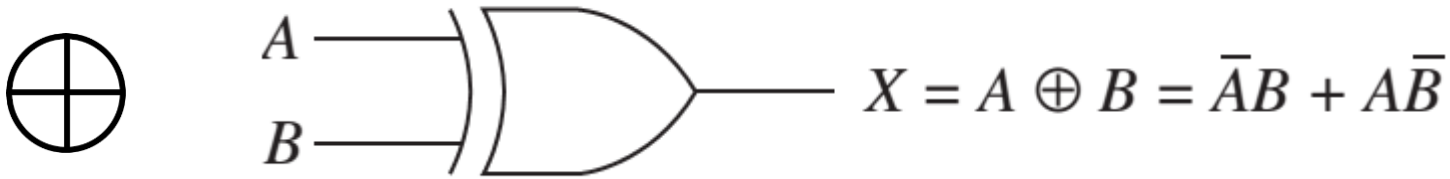
nikola.rendevski@fikt.edu.mk

летен семестар, 2017/2018
ФИКТ, УКЛО, Битола

Екслузивно или XOR (ЕксИли)

- **Видовме дека со употреба на дигитални кола во комбинаторна логика можеме да ја формираме било која логичка функција и да ја реализираме со прост дигитален систем
- Во продолжение ќе се запознаеме со уште една (две) логички функции
  - XOR
    - XNOR (Екслузивно Нили)
- XOR овозможува дополнителна дигитална функционалност која: дава 1 на излез доколку двата влезови се различни (1,0; 0,1)
  - Дава 0 на излез доколку се исти (0,0; 1,1)

Екслузивно Или - XOR (ЕксИли)



$X = \bar{A}B + A\bar{B}$

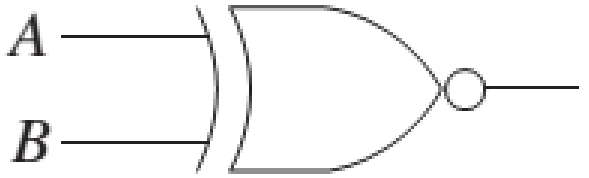
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

XOR

1 -> кога двата влеза имаат различни бинарни вредности

Симбол и таблица на вистинитост на XOR

XNOR (ЕксНили)



$$X = \overline{A \oplus B} = AB + \overline{A}\,\overline{B}$$

1 -> кога двата влеза се високо или ниско ниво. 0 - кога двата влеза се различни

$$X = AB + \overline{A}\,\overline{B}$$

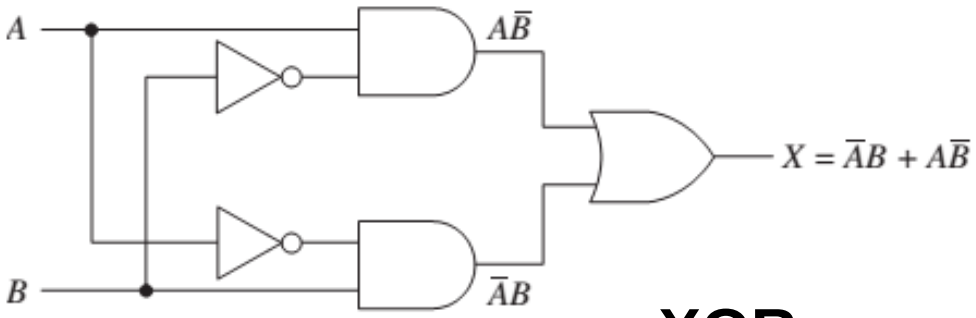
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

XNOR

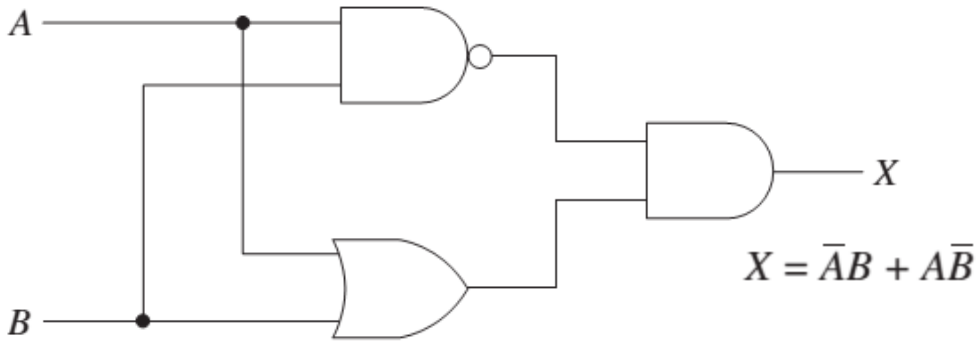
Симбол и таблица на вистинитост на XNOR

Реализации на XOR

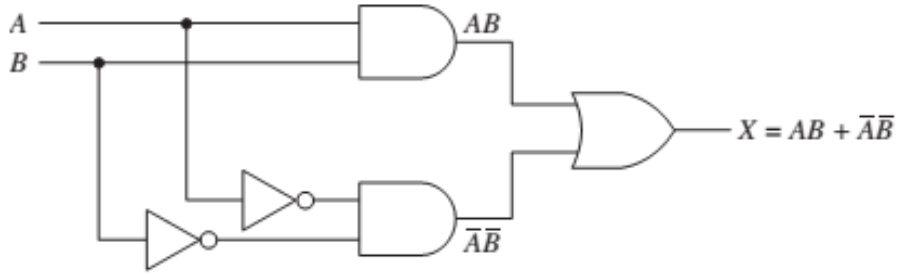
- Користејќи ги буловите закони, правила и Теоремата на Де Морган може XOR може да се изведи на неколку начини.
- Примери:



XOR

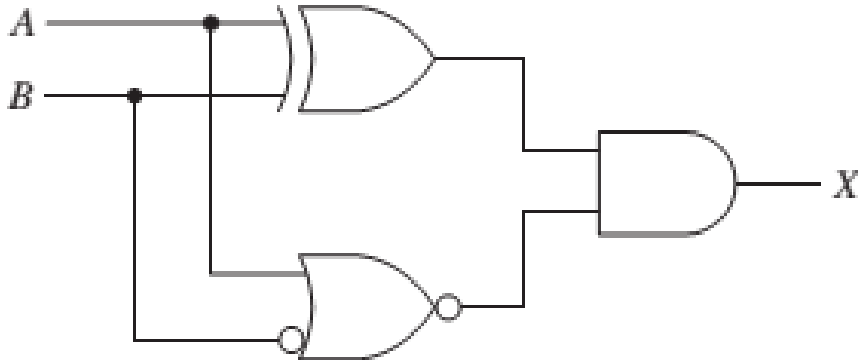


XNOR



Пример 1.

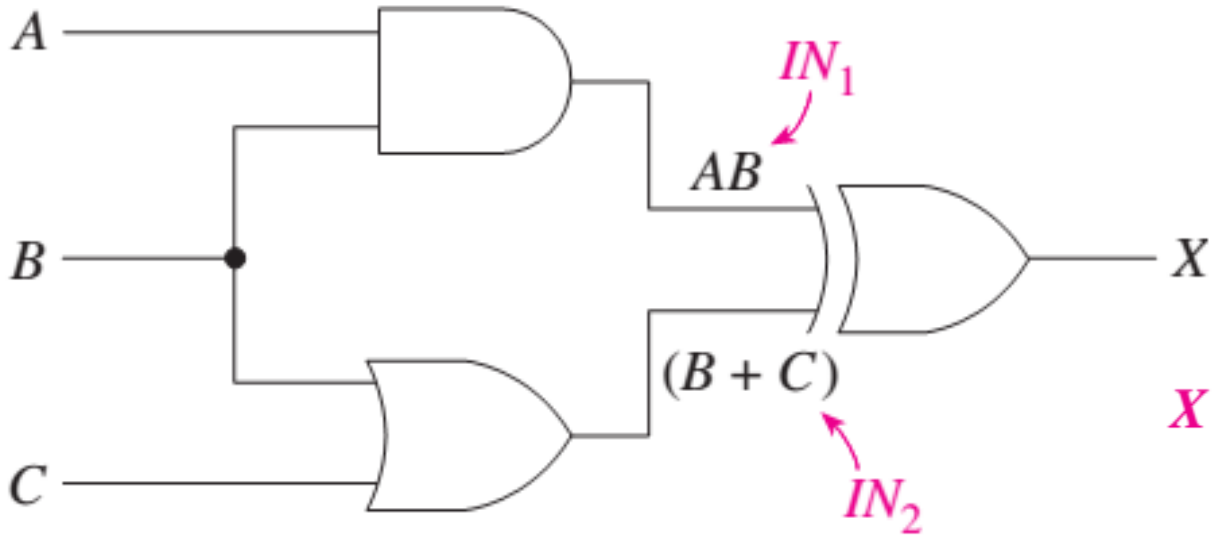
- Да се анализира излезот на колото на сликата



$$
\begin{aligned}
X &= (\overline{A}B + A\overline{B})\overline{A + \overline{B}} \\
&= (\overline{A}B + A\overline{B})\overline{A}\,\overline{\overline{B}} \\
&= \overline{A}B\overline{A}B + A\overline{B}\,\overline{A}B \\
&= \overline{A}B
\end{aligned}
$$

Пример 2.

- XOR со мултиваријабилен влез!



$$X = \overline{IN}_1 IN_2 + IN_1 \overline{IN}_2$$

$$\begin{aligned} X &= \overline{A}\overline{B}(B + C) + AB(\overline{B + C}) \\ &= (\overline{A} + \overline{B})(B + C) + AB\overline{B}\,\overline{C} \\ &= \overline{A}B + \overline{A}C + \overline{B}B + \overline{B}C \\ &= \overline{A}B + \overline{A}C + \overline{B}C \end{aligned}$$

Привила на XOR алгебра

$$X \oplus 0 = X$$

$$X \oplus 1 = \overline{X}$$

$$X \oplus X = 0$$

$$X \oplus \overline{X} = 1$$

$$\overline{X} \oplus \overline{X} = 0$$

$$\overline{X} \oplus \overline{X} = X \oplus X$$

$$A \oplus AB = A\overline{B}$$

$$A \oplus A\overline{B} = AB$$

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

$$(A \oplus B)\cdot(A \oplus C) = \overline{A}BC + A\overline{B}\,\overline{C}$$

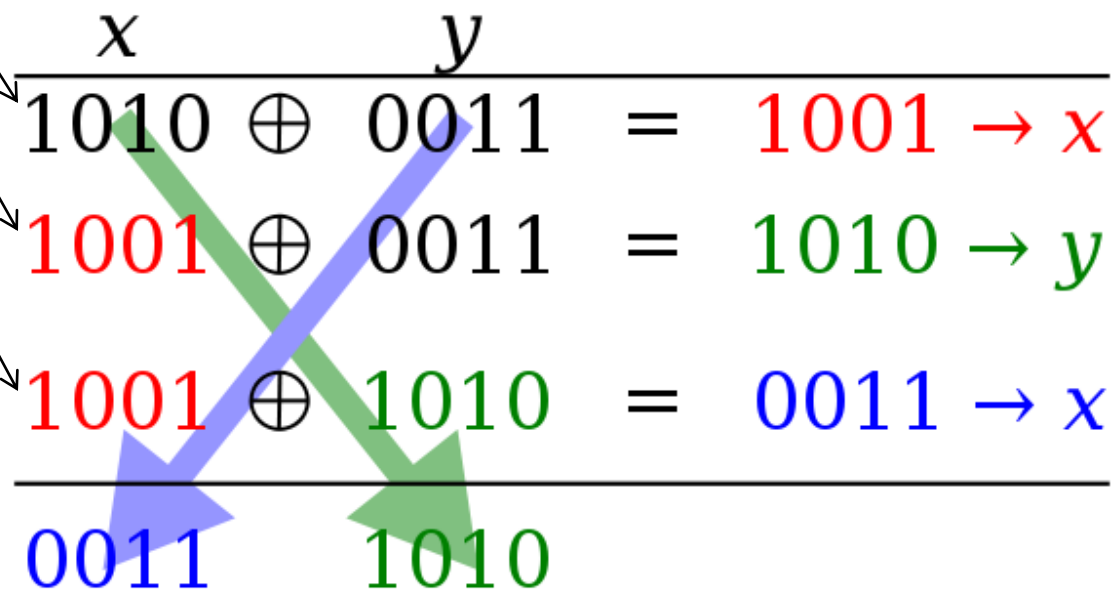
Примена на XOR

- XOR логиката има широка примена во дигиталните системи (архитектури) и компјутерските науки
- Примери:
  - Формирање на Грејов код (*ќе го разгледаме подетално во наредните слајдови)
  - **При проверка на парност
  - Во програмските јазици (JavaScript, C и C#) се користи XOR swap функција наместо да се користи temp променлива за менување на вредностите на две (Вредноста од A оди во B, Вредноста од B оди во A

XOR Swap алгоритам

- Bitwise swap (бит по бит на иста позиција)
- За две променливи X И Y
- 3 Чекори:

1. **X := X ⊕ Y**
2. **Y := Y ⊕ X**
3. **X := X ⊕ Y**

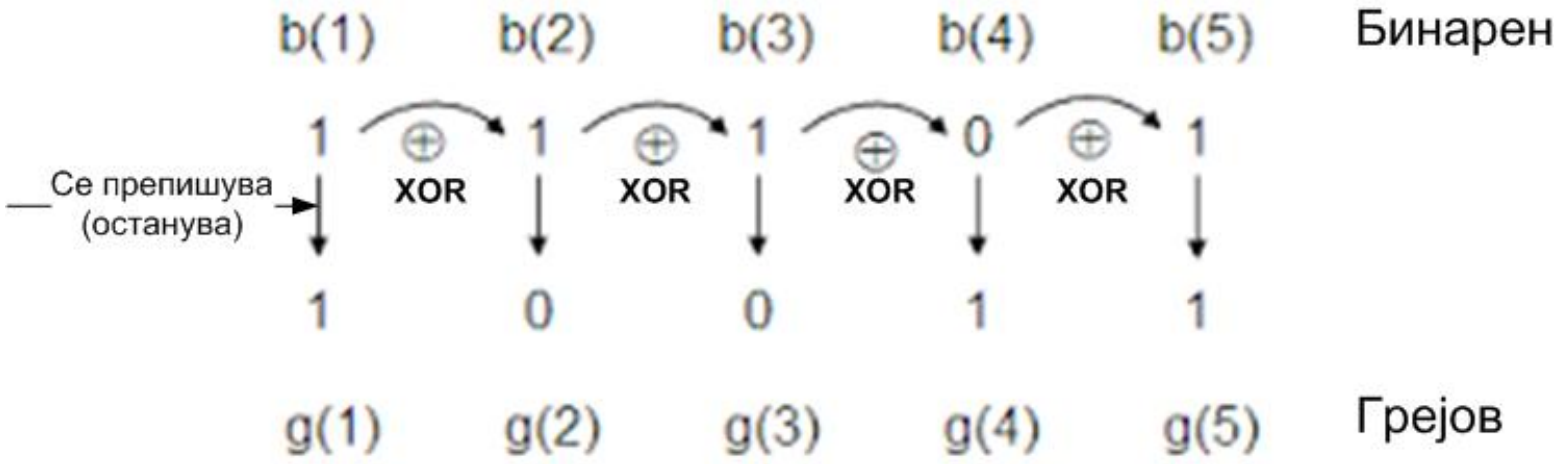


Грејов код (Gray Code)

- Во теоријата на информации главна цел при дизајнот на КОД е “Minimum Error Code”
- Код кој ги поддржува техниките за детекција и корекција
- Грејовиот код често се нарекува и Рефлектиран Бинарен Код или Цикличен пермутациски код
- Идејата е следна: Менувањето на секоја следна состојба да се реализира со менување на нивото само на едно битово место. Ова би гарантирало дека две соседни состојби ќе се разликуваат само во 1 бит. Тоа не е така кај бинарниот или BCD кодот.
- Пример 011 = 3, 100=4, Преминот од 3 во 4 бара промена на 3 бита. Битовото растојание помеѓу 011 и 100 е 3. На сите позиции има различни вредности на битовите! Ваквиот пристап е непрактичен.
- Грејовиот код обезбедува претходната и следната состојба да се разликуваат само во 1 бит

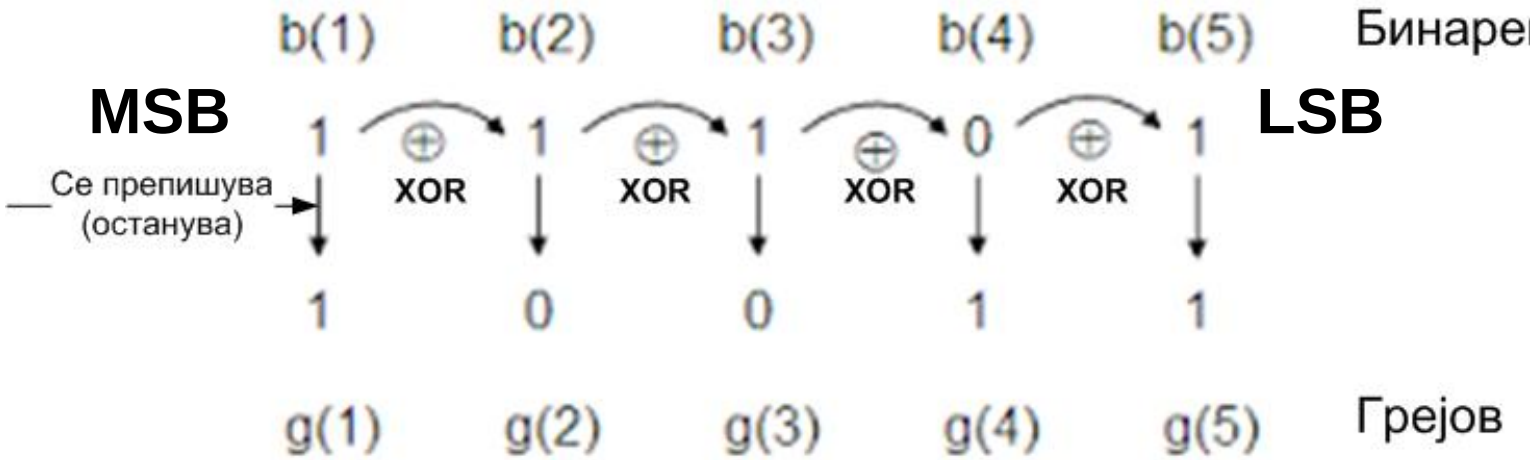
Кодирање на Бинарна секвенца во Грејов код

- MSB (Most Significant) битот од битовата низа останува (се препишува)
- Секоја следна позиција е **XOR** бинарна операција на тековниот со следниот дигит во насока од MSB кон LSB



Кодирање на бинарна секвенца во Грејов код

- Пример: Бинарна секвенца 11101



- Грејов код: 10011

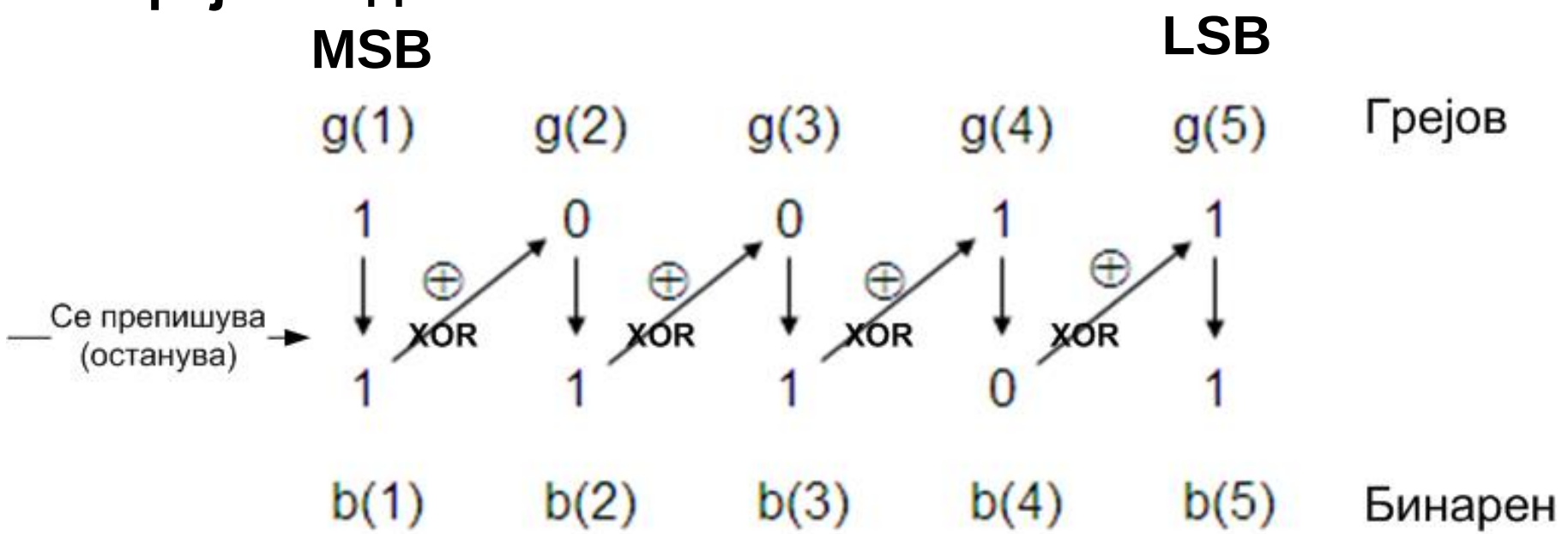
- Да се изврши кодирање на бинарната секвенца 1110001110 во Грејов код

Декодирање од Грејов код во Бинарен

- Слична процедура
- Реверзибилна (секогаш може да се врати оригиналната бинарна секвенца од Грејов во Бинарен)
- Првиот (MSB) дигит се препишува
- Секој следен е **XOR**, но сега на тековно декодираниот дигит со следниот дигит во грејовата секвенца
- На овој начин се обезбедува реверзибилноста (обратна операција на XOR)

Декодирање од Грејов во назад во Бинарен

Грејов код: 10011



Битова секвенца 11101 – оригиналната (вратена од Грејовиот код)

Декаден→Бинарен→Грејов

Декаден	Бинарен	Грејов
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Карнови мапи (Karnaugh Mapping)

- Со користење на правилата, законите и Теоремата на Де Морган извршивме редукција – минимизација на логичките кола за реализација на одредена логичка функција
- Логичката функција ја доведовме во форма SOP (Сума од производи)
- Карновите мапи се уште една техника која обезбедува упростување и редукција на логичките кола
- Како и да е: Функцијата треба да се доведи во SOP (POS) форма, но од таму без користење на Булова алгебра, со систематски пристап се овозможува со сигурност добивање на најпростата дигитална конфигурација.
- Карновата мапа има слична форма на таблицата на вистинитост (дводимензионална таблица на вистинитост)
- Можат да се искористат за упростување на дигитални функции со 2,3,4,5 и 6 варијабли (влезови)
- Ние ќе реализираме со 2,3, и 4 варијабли (испит)

Карнови мапи (Karnaugh Mapping)

- Решавањето на Карнови мапи со 5 и 6 варијабли рачно е покомплексно и подложно на грешки (потребна е перфектна визуелна точност при решавањето)
- Се препорачува да се искористат малку понапредни компјутерски техники за нивно решавање при реален дизајн за повисоките димензии на КМ (Ќе користиме Logic Friday)
- *Информативно ќе го покажеме (на табла со конкретен пример) за КМ со 5 и 6 варијабли
- Целта е да се разбери суштината и применливоста на оваа метода за редукција (минимизација, оптимизација) која бара и визуелна перцепција
- Карновите мапи се дводимензионални таблици (матрици) чии ќелии ги содржат вредностите на излезот за различни комбинации на влезовите
- Можат да се применат кај системи со 2,3,4,5 и 6 варијабли (влезни)
- Бројот на ќелии на КМ е 2^{Br}, каде Br е бројот на варијабли
  - **За 2 варијабли – 4 ќелии (2x2), За 3 варијабли – 8 ќелии (2x4, 4x2),**

Карнови мапи (Karnaugh Mapping)

- Редиците и колоните претставуваат комбинации од влезните променливи
  - Треба да се внимава, да се запази **грејовата** репрезентација на комбинациите, т.е секоја соседна комбинација да се разликува само во 1 променлива (00,01,11,10)
- Функцијата се доведува до SOP
- Не е потребно понатамошна оптимизација со примена на Буловата алгебра доколку имаме SOP презентација на функцијата
- Ни треба:
  - Табела на вистинитост, или
  - Каноничка форма на функцијата
- Каноничка форма

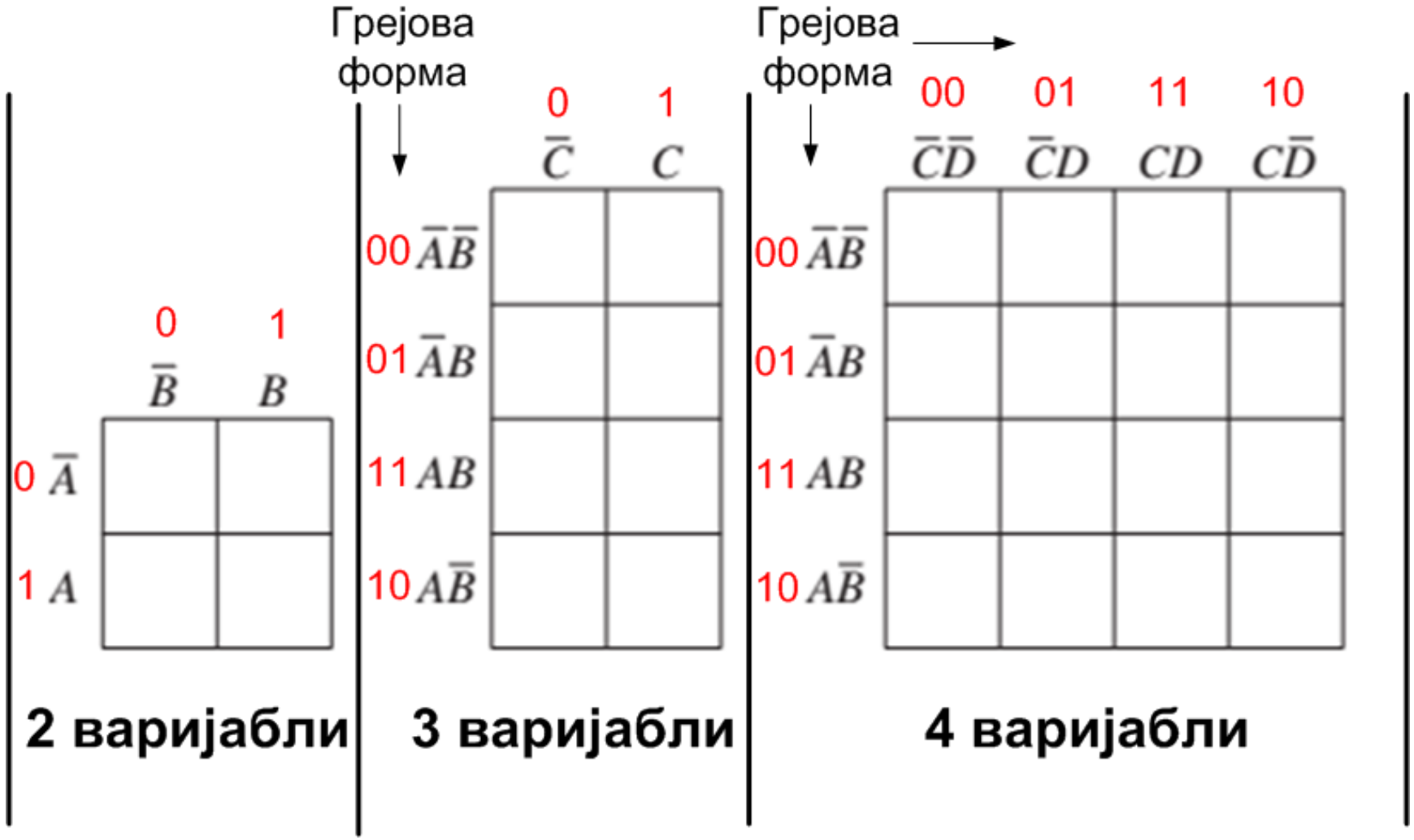
Редниот број на комбинацијата во табелата на вистинитост дава 1 (се брои од 0 до N-1)

Минтерми (minterms) SOP →

$$F_1(x, y, z) = \Sigma(3, 5, 6, 7)$$

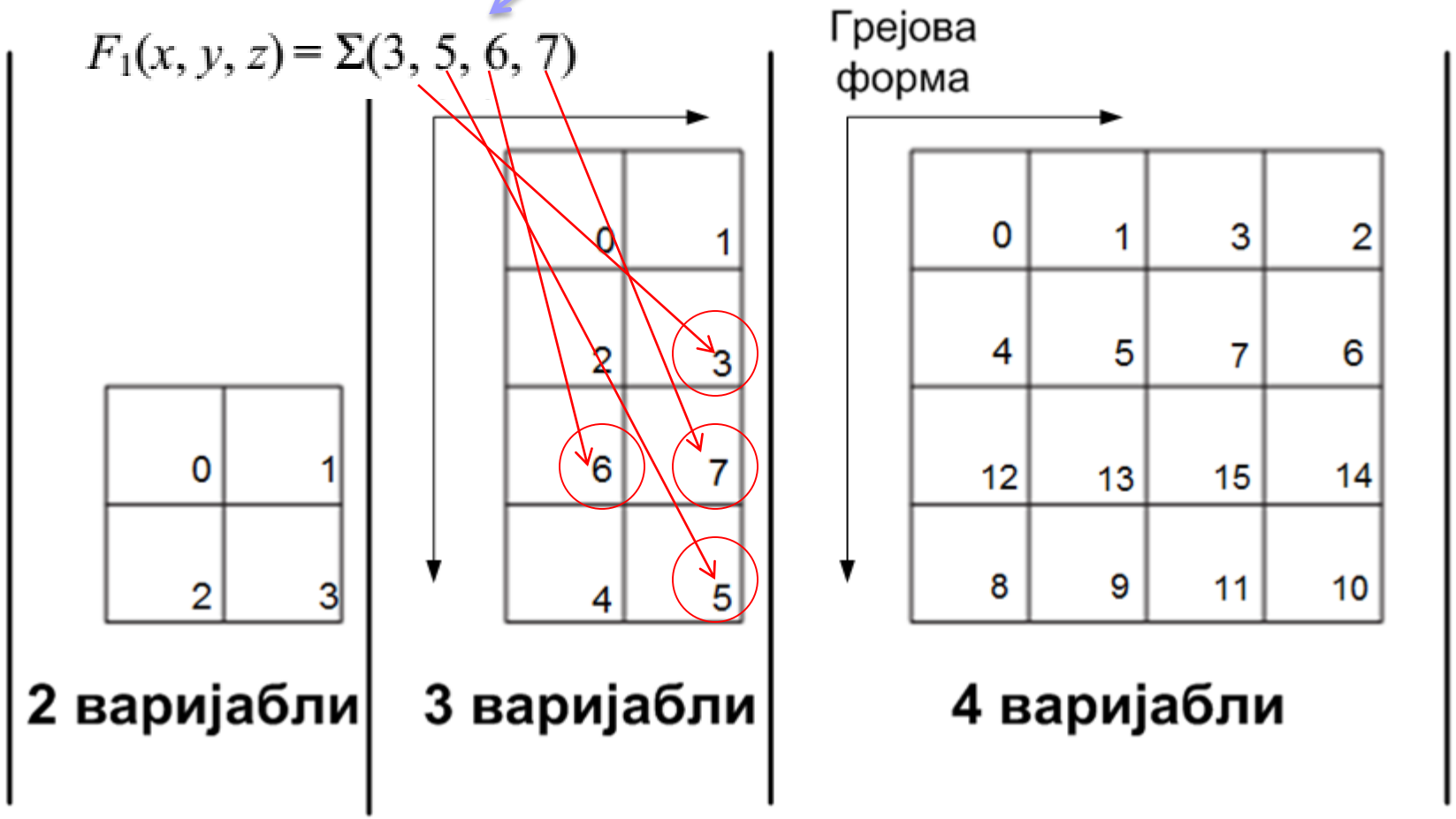
$$F_2(x, y, z) = \Pi(0, 1, 2, 4)$$ → **Maxterms (макстерми – POS) – со нулите**

Карнови мапи (Karnaugh Mapping)



Карнови мапи (Karnaugh Mapping)

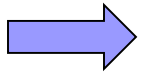
Позиции на кои има единици (1) во таблицата на вистинистост



Како се работи со КМ?

- Ќе образложиме со конкретни примери:
- Реализираме SOP
- Ги внесуваме 1 на соодветните локации во Карновата мапа според SOP или Каноничката форма
- Заокружуваме: осумки, четвроки, составени од соседни места (квадрати) од Карновата Мапа каде има единици и тоа
  - Хоризонтално
  - Вертикално
  - Квадратно (squares) (2 хориз. или вертикални соседи кои креираат 4-ка, 8-ка...)
  - Позициите на рабовите горе-долу, и лево-десно се соседни позиции и можат да образуваат “encircling” т.е заокружување
  - По “ќошовите” ((нај)лево-горе, (нај)десно-горе, (нај)лево долу, (нај)десно долу)
  - **НЕ** дијагонално!!!

Стратегијата најпрво да се заокружи најголемата можна група на соседни квадрати не дава секогаш добри резултати!!! Внимавајте на испит!



Карнови мапи

- Секој квадрат е еден минтерм/макстерм на функцијата која треба да се минимизира
- Карновата мапа е визуелен дијаграм на сите можни начини на кои функцијата може да биде изразена во стандардна форма
- Со препознавање на различни облици во КМ можеме да изведеме алтернативни алгебарски изрази за истата функција од кои понатаму ќе се избере наједноставниот
- Наједноставниот алгебарски израз е со минимален број на членови и литерали (променливи) во секој член
- Овој израз дава минимален број на логички кола и минимален број на влезови во дигиталната логика
- **ВАЖНО!!!** Наједноставниот израз не мора да биде уникатен: Постои можност во некои случаи да има два или повеќе изрази кои го задоволуваат критериумот за минималност
- Во тој случај секое од решенијата ЗАДОВОЛУВА!
- *Да разбереме што е примарна а што есенцијална импликанта е клучно и за овој но и за Quine-McCluskey табуларниот метод.

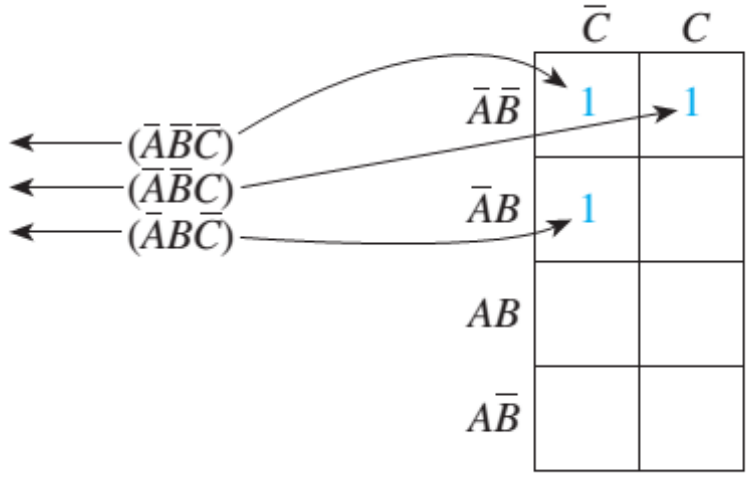
КМ: примарни и есенцијални импликанти

- Да претпоставиме SoP:
- Секој член во SoP се нарекува импликанта
- Примарна импликанта (Prime Implicant - PI) е импликанта која не е целосно содржана во друго заокружување
- Есенцијална импликанта (Essential Implicant - EI) е примарна импликанта која покрива барем едно поле (минтерм) кое не може да се покрие со друго заокружување – со други зборови мора да постои за минтермот да биде земен во предвид при минимизацијата.
- На почеток на минимизацијата со КМ ги детектираме и заокружуваме есенцијалните импликанти. Одбираме најголемо можно заокружување (1,2,4,8...)
- Потоа: бараме двојки,четворки, осумки итн., кои единствено можат да се комбинираат во соодветна група на тој начин (на единствен начин формираат двојки, четворки, осумки итн).

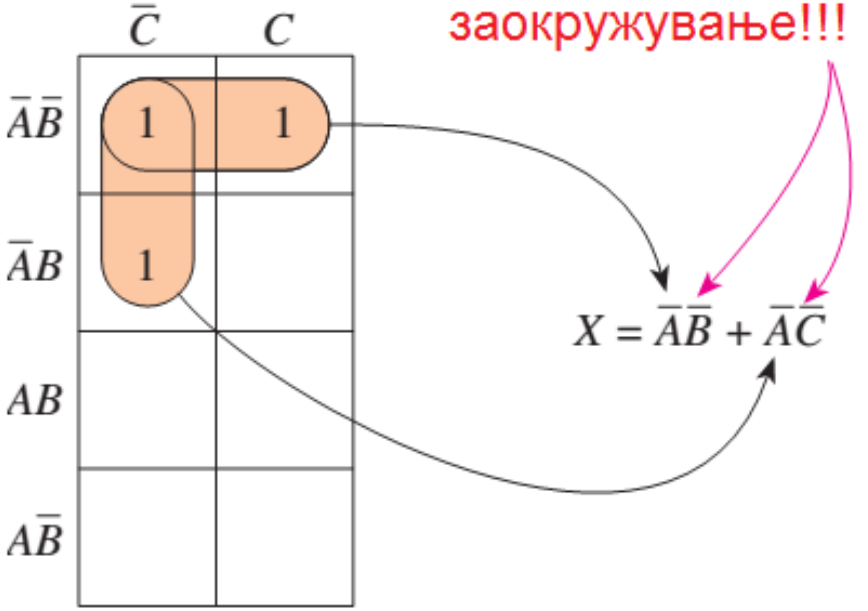
Пример:

$$X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C}$$

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



Варијаблите кои останаа константни во секое заокружување!!!



Доказ:

$$X = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}B\overline{C}$$

$$X = \overline{A}\overline{B}(C + \overline{C}) + \overline{A}B\overline{C}$$

↓

=1 (Според правило 8)

$$X = \overline{A}\overline{B} + \overline{A}B\overline{C}$$

$$X = \overline{A}(\overline{B} + B\overline{C})$$

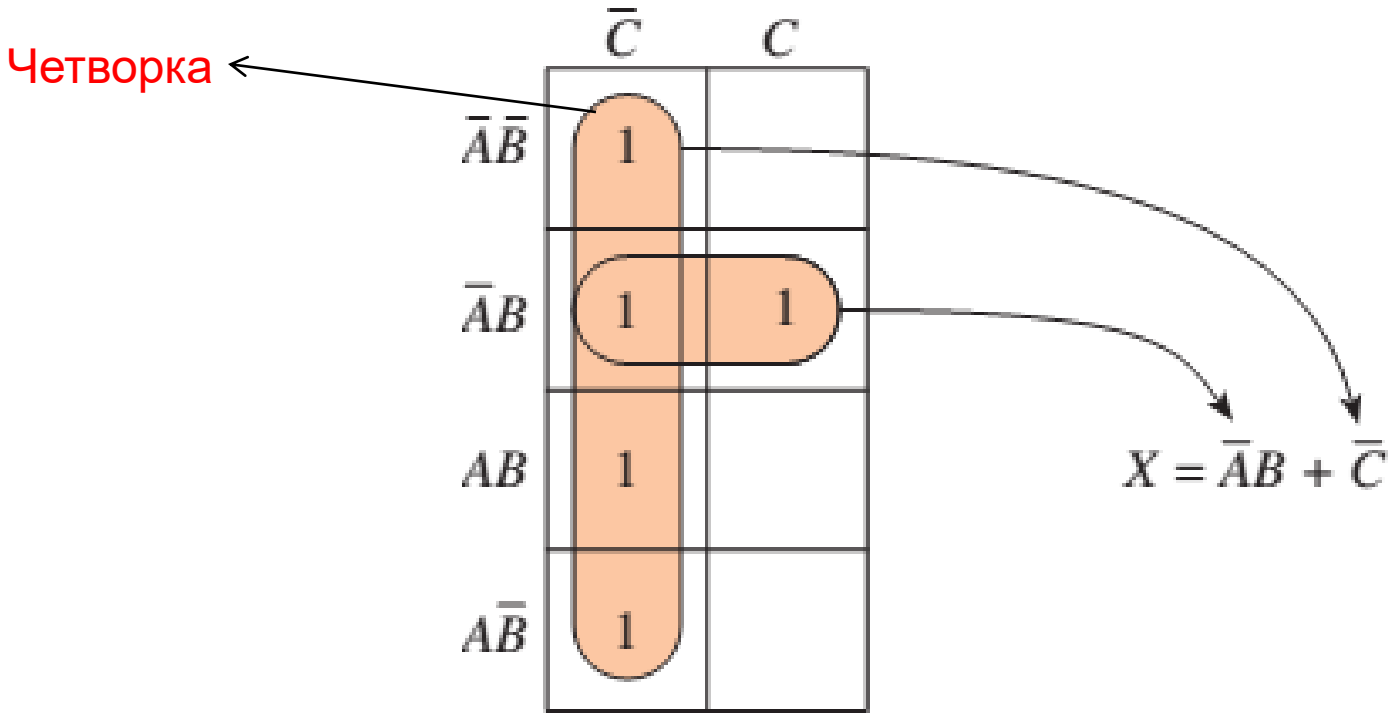
Правило 10b!!!

$$(\overline{B} + B\overline{C}) = (\overline{B} + \overline{C})$$

$$X = \overline{A}\overline{B} + \overline{A}\overline{C}$$ ✓

Пример:

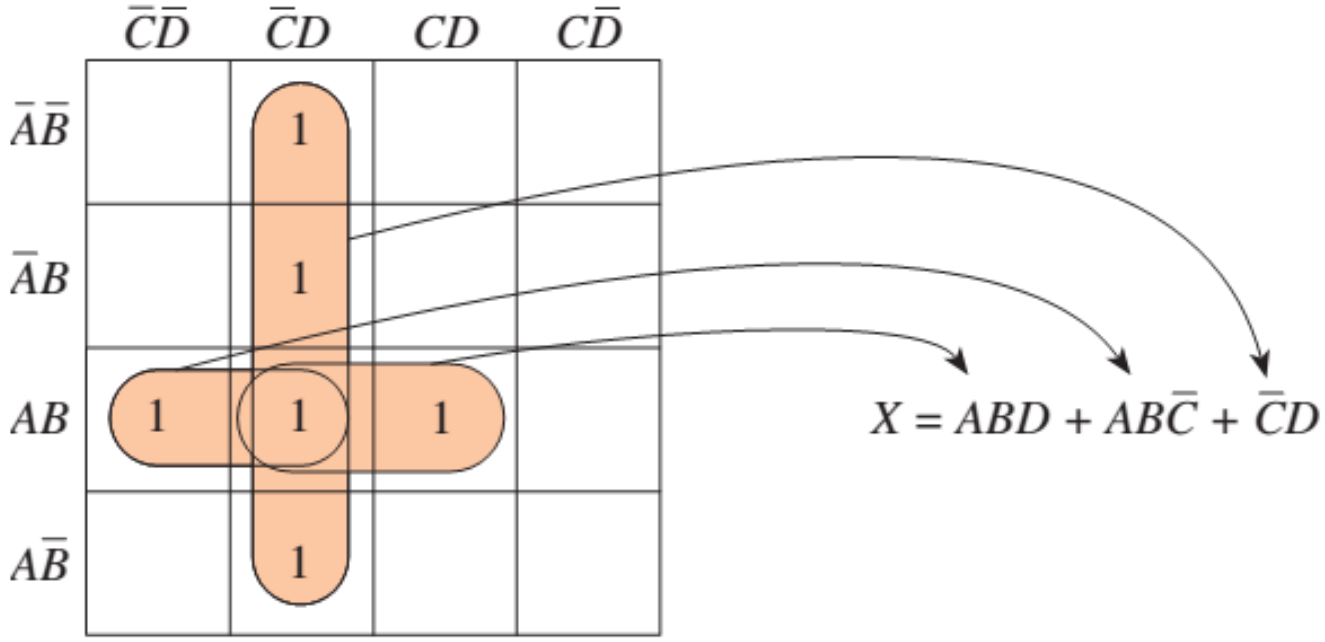
$$X = \overline{A}B + \overline{A}\,\overline{B}\,\overline{C} + AB\overline{C} + A\overline{B}\,\overline{C}$$



Да забележиме дека во членот $\overline{A}B$ нема варијабла С. Ова значи дека $\overline{A}B$ ќе биде 1 и за C и за $\overline{C}$

Пример:

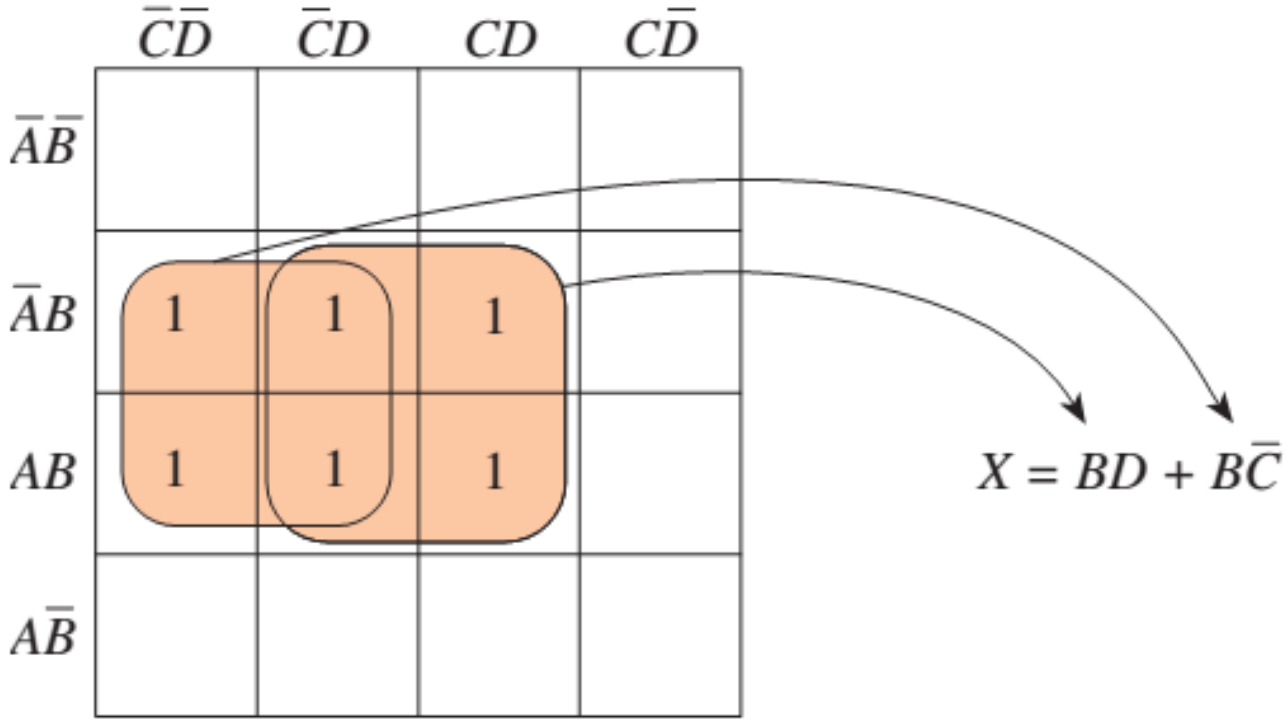
$$X = \overline{A}B\overline{C}D + A\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}\,\overline{C}D + AB\overline{C}D + AB\overline{C}\,\overline{D} + ABCD$$



Ова е случај со 4 варијабли, поради тоа бројот на ќелии во Карновата мапа е 16.

Пример:

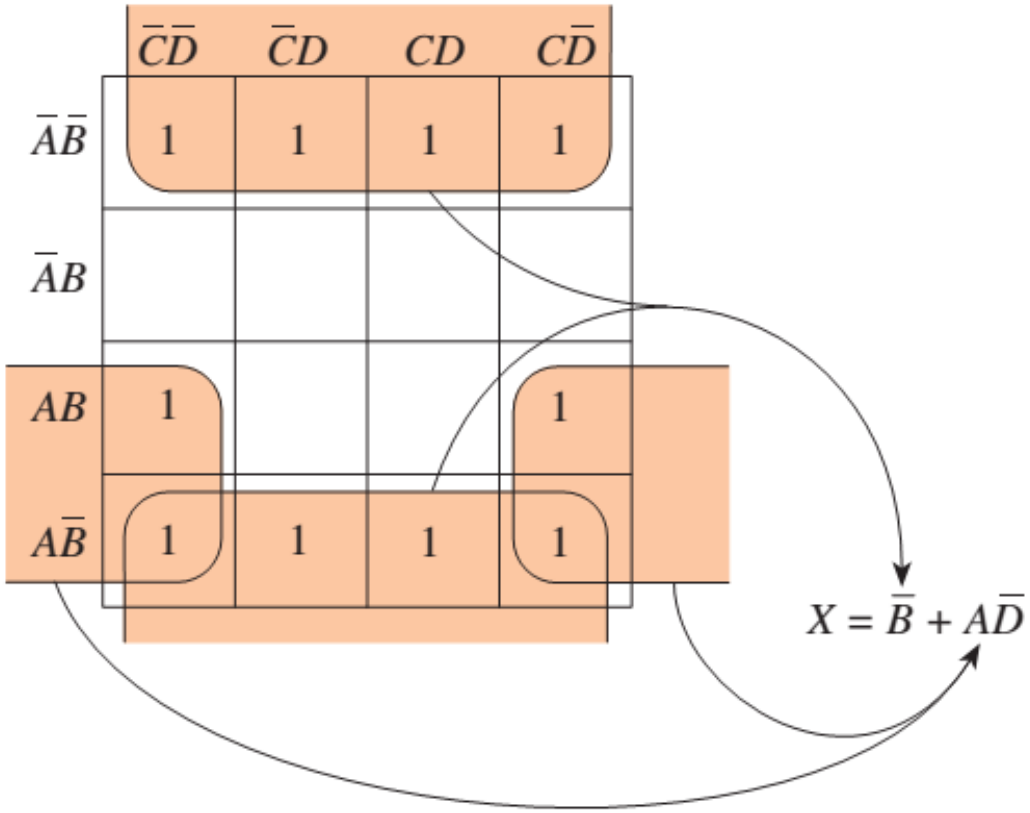
$$X = B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}D + \overline{A}BCD + ABCD$$



Случај со детектирани 2 квадратни четворки

Пример:

$$X = \overline{A}\,\overline{B}\,\overline{C} + A\overline{C}\,\overline{D} + A\overline{B} + ABC\overline{D} + \overline{A}\,\overline{B}C$$



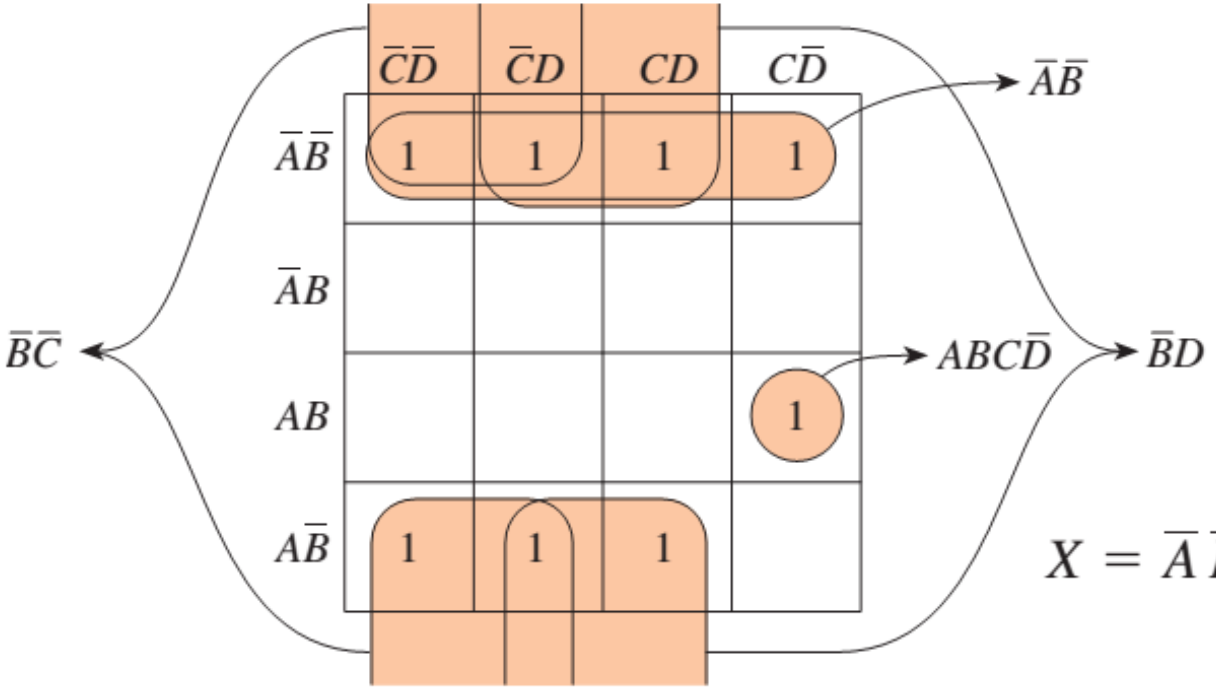
Случај со детектирани осумка и четворка по рабовите на мапата!

Пример:

$$X = \overline{B}(CD + \overline{C}) + C\overline{D}(\overline{A + B} + AB)$$

$$X = \overline{B}CD + \overline{B}\,\overline{C} + C\overline{D}(\overline{A}\,\overline{B} + AB)$$

$$= \overline{B}CD + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}C\overline{D} + ABC\overline{D}$$

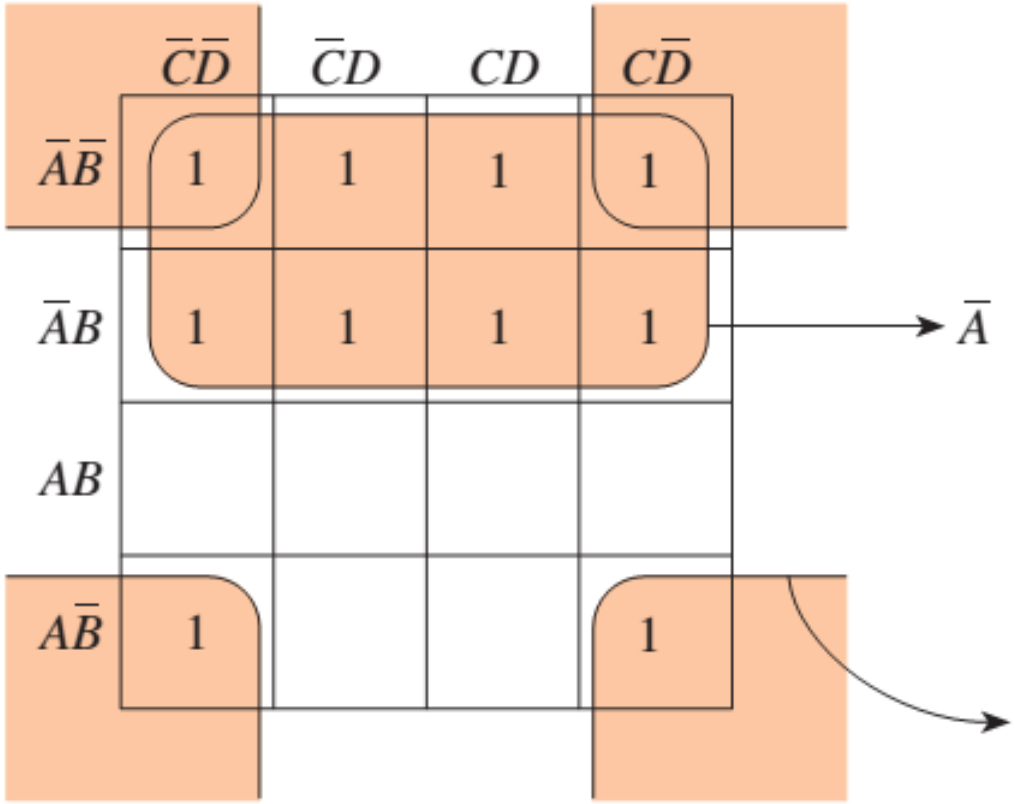


$$X = \overline{A}\,\overline{B} + \overline{B}\,\overline{C} + \overline{B}D + ABC\overline{D}$$

Случај хоризонтално, по рабовите на мапата и заокружување со 1 елемент!

Пример:

$$X = \overline{A}\,\overline{D} + A\overline{B}\,\overline{D} + \overline{A}\,\overline{C}D + \overline{A}CD$$

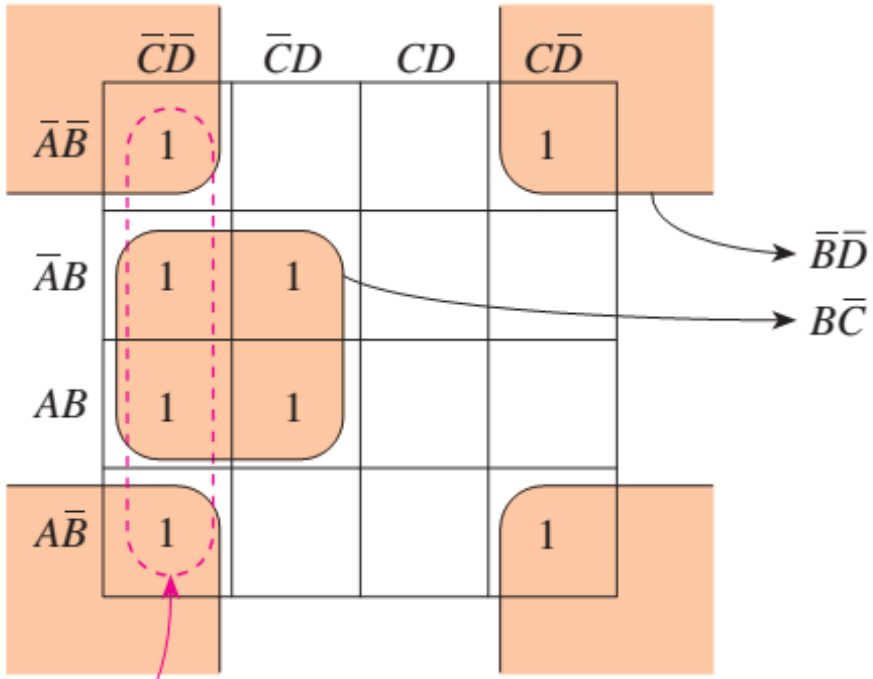


$$X = \overline{A} + \overline{B}\,\overline{D}$$

4 – ќошиња (агли)

Пример:

$$X = \overline{A}\,\overline{B}\,\overline{D} + A\overline{C}\,\overline{D} + \overline{A}B\overline{C} + AB\overline{C}D + A\overline{B}C\overline{D}$$



$$X = \overline{B}\,\overline{D} + B\overline{C}$$

Треба да се внимава да се избегни редундантност (случај кога сите елементи од една импликанта се членови во други импликанти.

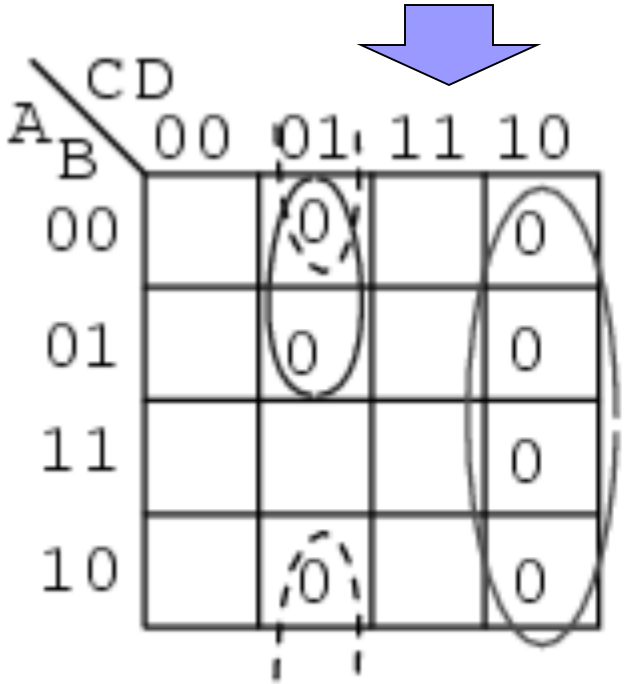
Да ги следиме препорачаните чекори, прво да ги заокружиме есенцијалните примарни импликанти во најголеми можни групи, а потоа за незаокружените да бараме кои од нив влегуваат во најголеми групи на единствен начин. Да не се заокружува веднаш најголемата и прво-воочената група!

SoP или PoS пристап?

- Кога функцијата ја претставуваме со SoP (сума од производи) ги земаме комбинациите од варијаблите (A,B,C,D…) во кои функцијата е 1. Во Карновата мапа впишуваме единици на соодветните локации кои одговараат на комбинацијата од варијаблите
- Кога функцијата ја претставиме како PoS (производ од суми) ги земаме комбинациите каде функцијата дава 0. Во Карновата мапа впишуваме нули
- Прашање: Кој пристап треба да го употребиме? Одговорот на ова прашање зависи од технологијата во која дигиталниот дизајн ќе биде имплементиран.
- Во следните примери ќе дискутираме за овој избор.

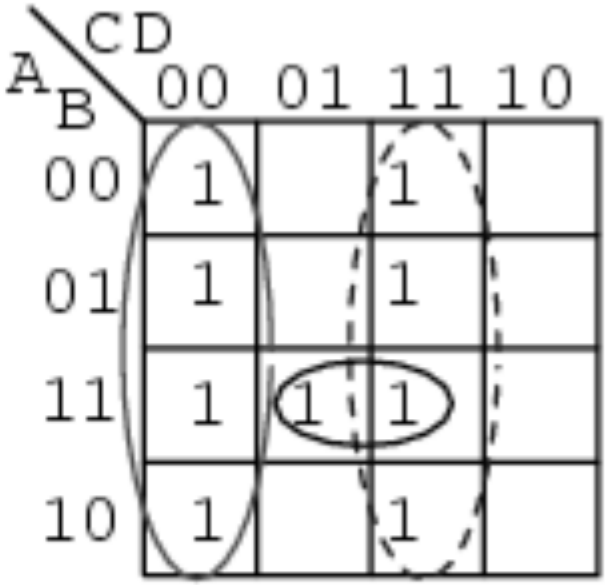
SoP (минтерми) и PoS (макстерми)

PoS $Y = (A+B+C+\overline{D})(A+B+\overline{C}+D)(A+\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$
$(\overline{A}+\overline{B}+\overline{C}+D)(\overline{A}+B+C+\overline{D})(\overline{A}+B+\overline{C}+D)$



PoS решение:

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



$Y = \overline{C}\,\overline{D} + CD + ABD$

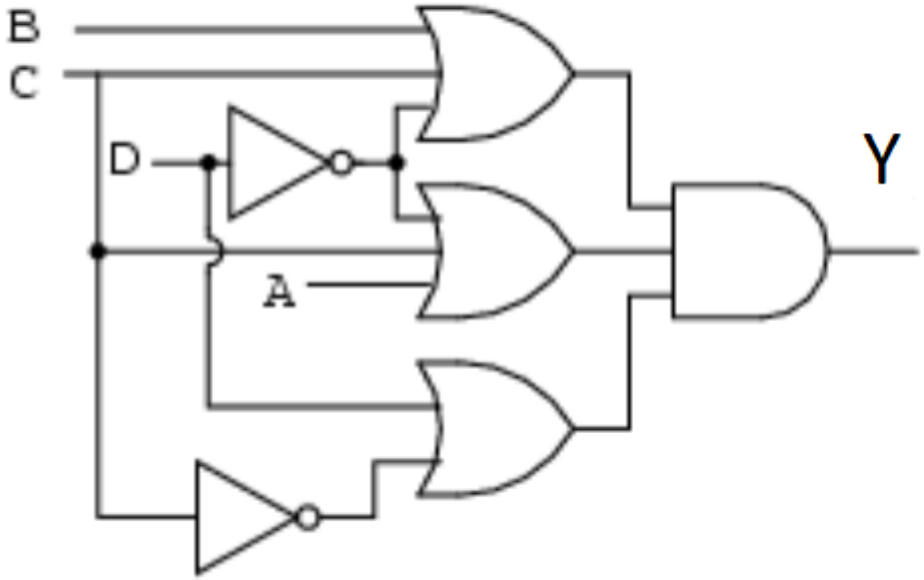
SoP решение:

SoP или PoS, што е „подобра“ опција?

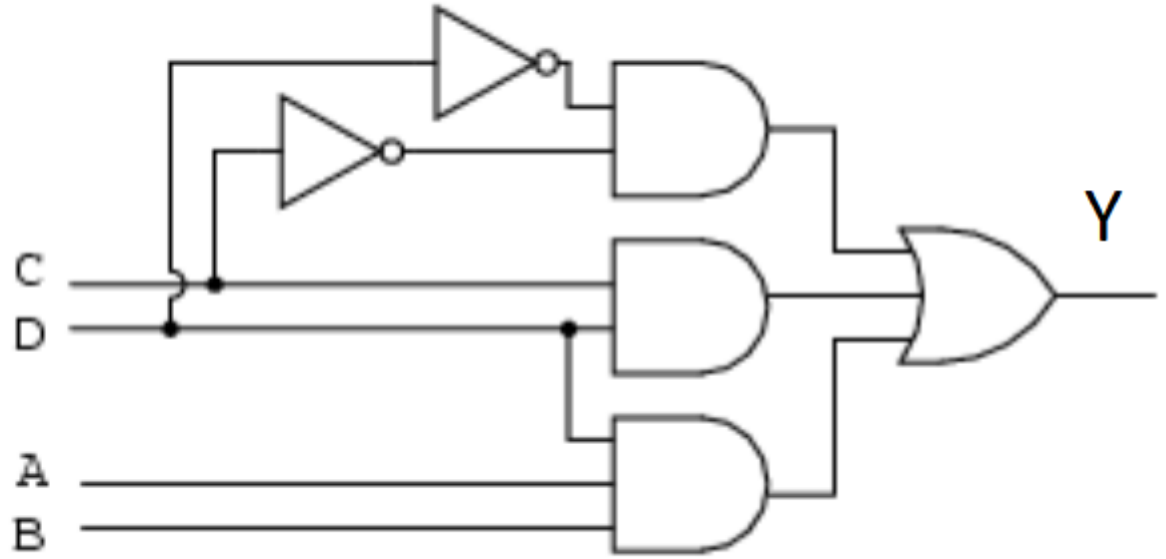
- Кое од решенијата од претходниот пример е попросто?
- PoS решението се реализира со 3 ИЛИ порти и една И порта
- SoP решението се реализира со 3 И порти и една ИЛИ порта
- Двете решенија се реализираат со ист број на порти
- Но ако погледнеме подобро, броејќи го бројот на влезовите во портите, за PoS треба 8 влеза, а за SoP треба 7 влеза
- Во однос на минимална цена, SoP решението е попросто?!?..
- Технички гледано тоа е точно, но е малку применливо во реалниот дизајн.
- Подоброто решение зависи од комплексноста и фамилијата на логички интегрирани кола кои се користат
- SoP пристапот е најчесто подобра опција при користење на т.н TTL логика (фамилија на кола), бидејќи таму NAND колата се основна градбена клетка и затоа SoP пристапот е подобар
- Од друга страна PoS е подобар пристап кај т.н. CMOS логичка фамилија, бидејќи достапни се NOR порти со различен број на влезови.

SoP или PoS, што е „подобра“ опција?

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



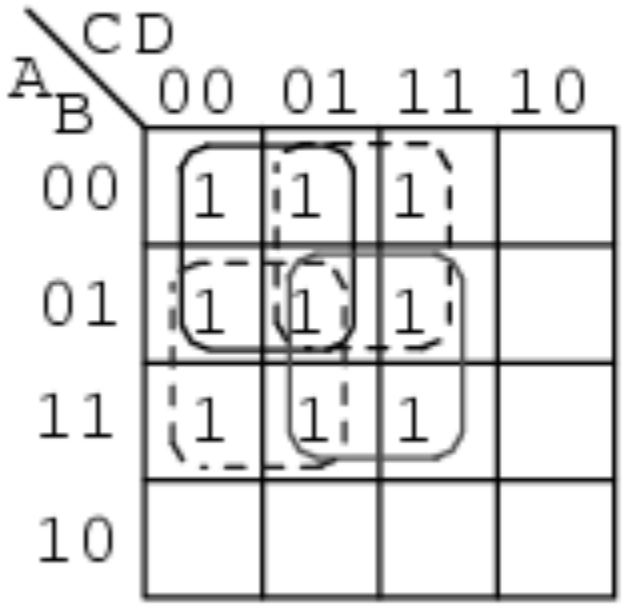
$Y = \overline{C}\,\overline{D} + CD + ABD$

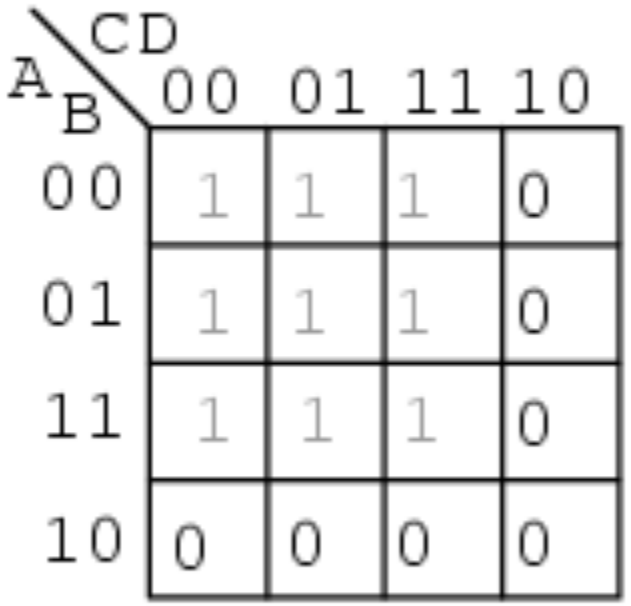


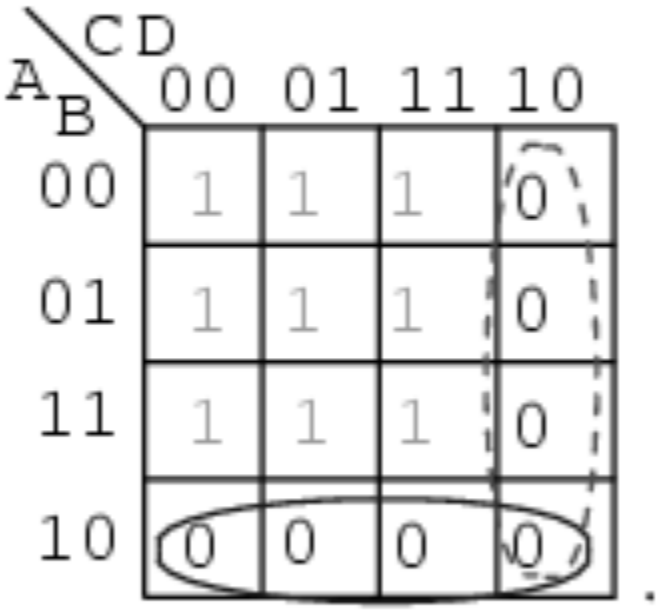
SoP или PoS, што е „подобра“ опција?

- Пример за PoS и SoP пристап

$$Out = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,D$$

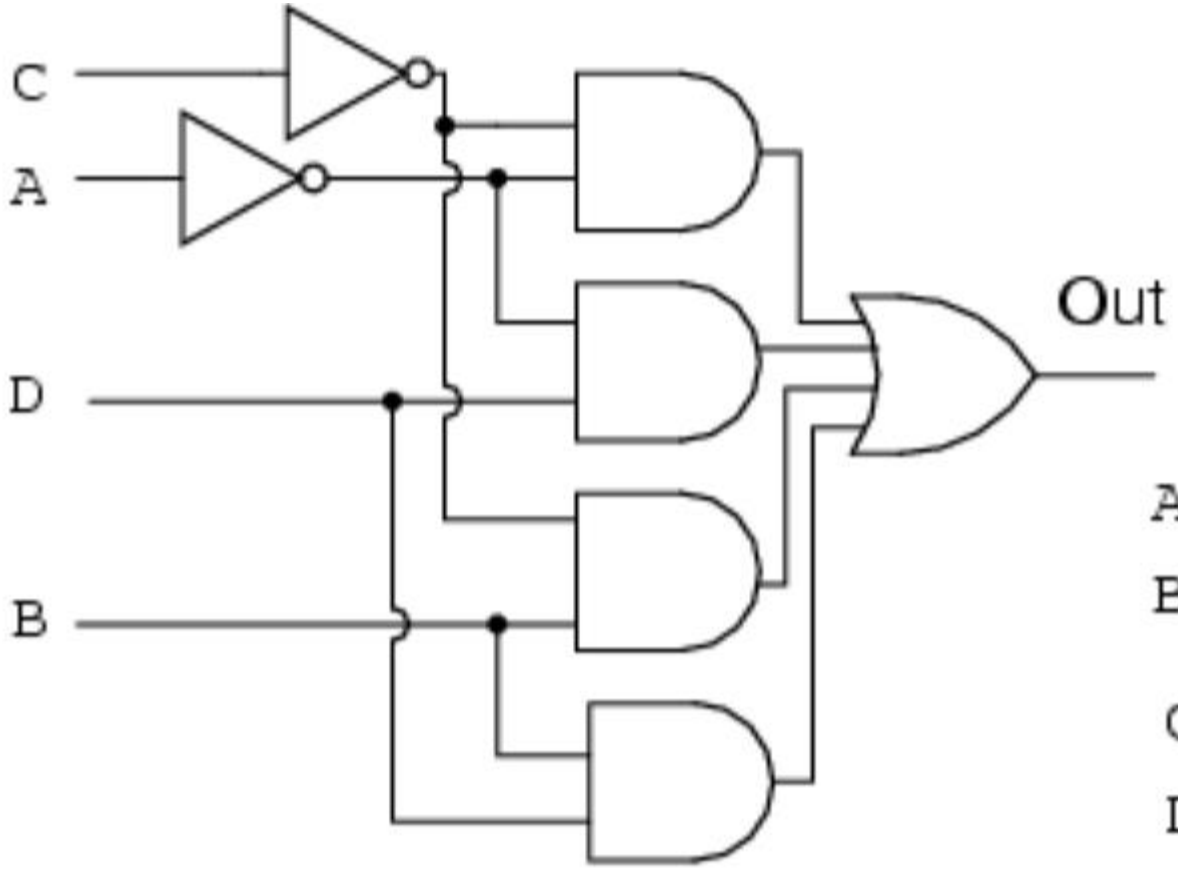






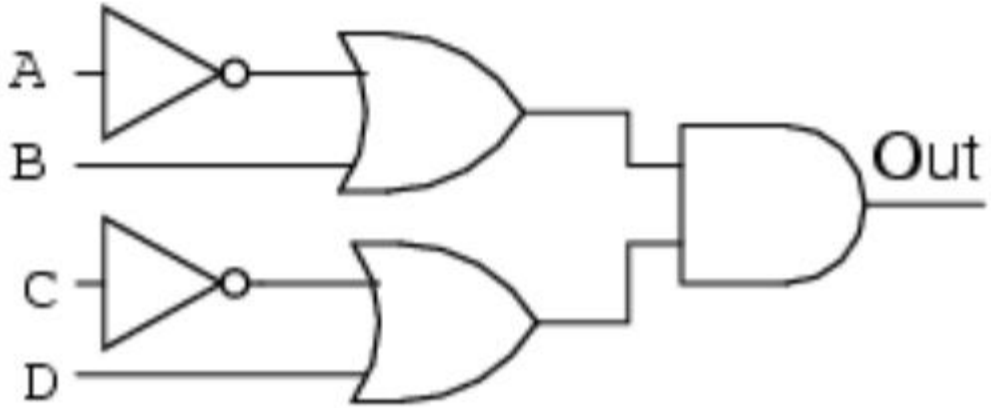
$$Out = \overline{A}\,\overline{C} + \overline{A}\,D + B\,\overline{C} + B\,D$$

$$Out = (\overline{A} + B)(\overline{C} + D)$$



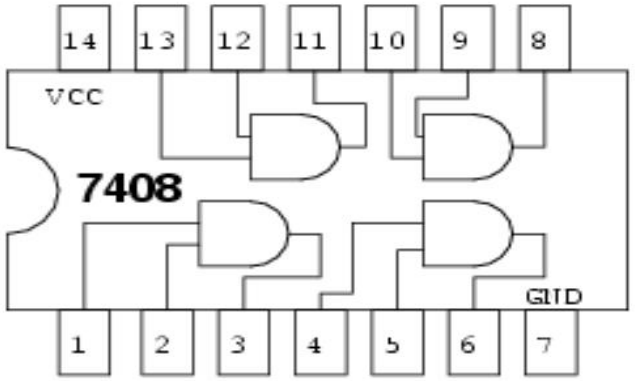
$$Out = \overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$$

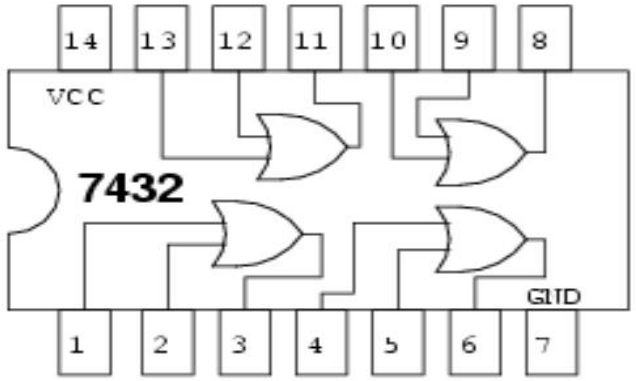
SoP

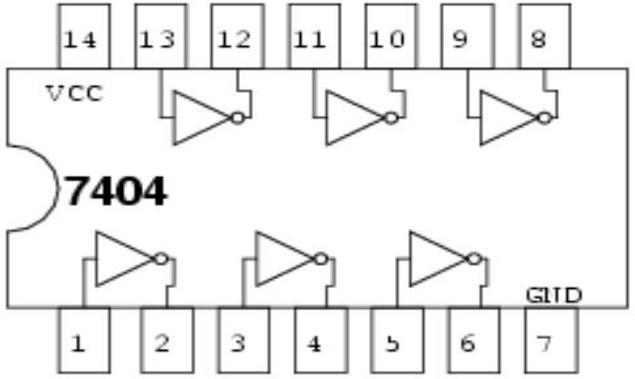


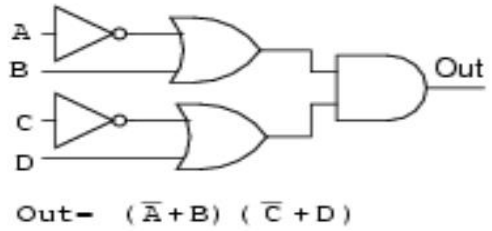
$$Out = (\overline{A}+B)(\overline{C}+D)$$

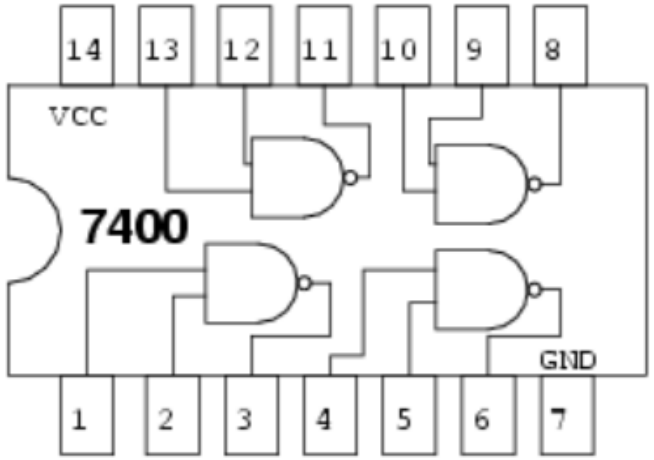
PoS

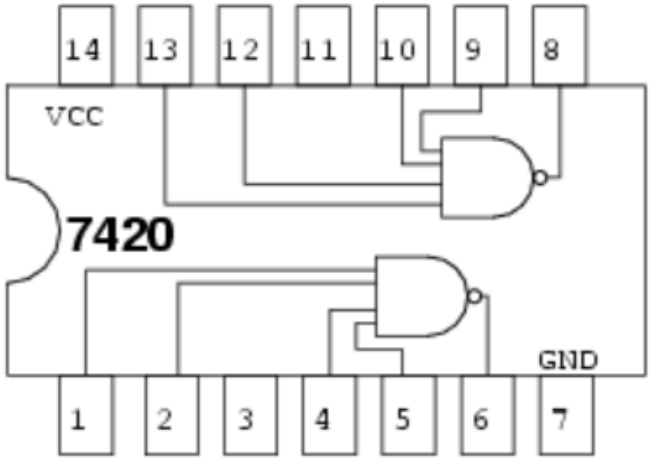












Во практичниот дизајн на дигитални кола, постојат стандардизирани чипови кои се составени од логички порти. Пиновите на интегралните кола се влезови во логичките порти од кои е составен чипот.

Дизајнот може да биде воден од ограничувања од тоа што е достапно, или дизајнерот има слобода и обврска да обезбеди минимален број на употребени чипови со цел да ја намали цената на реализацијата, но и потрошувачката на енергија на дигиталната логичка реализација.

Познавањето на функцијата на сите логички порти, правилата за конверзии од една во друга логика е клучна при дизајнот на оптимална логика

Секако дека постојат софтверски алатки кои го поедноставуваат дизајнот. Употребата на FPGA чиповите за синтеза на логика зема се поголем замав! (Ќе се работи на вежби – VHDL со FPGA

Минимизација на логички функции кои не се целосно зададени (Don’t Care)

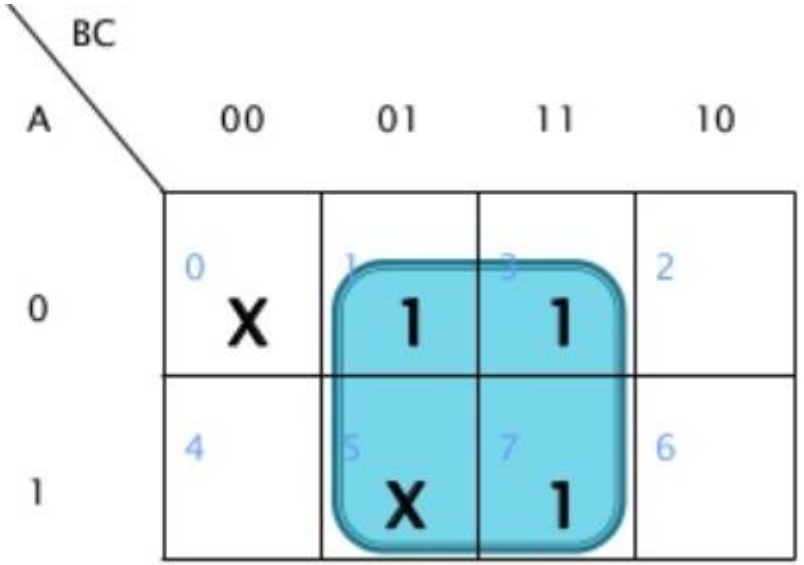
- Во сите примери кои ги разгледавме до сега, познато беше за кои комбинации на влезовите функцијата е 1, а за кои 0. Врз основа на тоа можеме да ги прикажиме како PoS или SoP и понатаму да ги анализираме и да дискутираме кој пристап е подобар според условите кои се поставени пред дизајнерот
- Во пракса, некогаш се појавуваат и случаи во кои функцијата не е комплетно зададена, т.е. постојат комбинации од влезните променливи за кои излезот на функцијата е небитен.
- Во ваков случај, овие комбинации можеме да ги третираме и како 1 и како 0, ако од тоа имаме некаква добивка при минимизацијата, или едноставно да ги изоставиме ако не можат да придонесат за минимизација/редукција
- Ваквите минтерми/макстерми при оптимизација со Карнови мапи во литературата се обележуваат различно: X,x,*,d,b, итн.
- Овие минтерми/макстерми можат да се комбинираат со дефинираните, но никако во заокружувања составени само од Don’t Care минтерми/макстерми

Минимизација на логички функции кои не се целосно зададени (Don’t Care)

- Во случајот Don’t Care минтермот 5 (101), може да се земе како 1 при што се формира четворка од која произлегува дека F=C. Don’t Care за 0 (000) се игнорира.
- На вежби – пример од пракса со нецелосно зададена функција со повеќе променливи.

$$F = \sum m(1, 3, 7) + \sum d(0, 5)$$

F=C



Квајн-Мек Класки (Quine-McCluskey) табличен метод за оптимизација

- Quine-McCluskey методот е егзактен алгоритам кој овозможува минимизирање на булова функција во SoP формат
- Овозможува минимизација со повеќе променливи и може да се имплементира во програмски код за комплексни функции со голем број влезови.
- Алгоритмот се реализира во 4 чекори:
  1) Генерирање примарни импликанти
  2) Конструкција на табела на примарни импликанти
  3) Редукција на табелата на примарни импликанти
     - Отстранување на есенцијални импликанти
     - Доминација на редица
     - Доминација на колона
  4) Решавање на табелата на примарни импликанти
- Ќе го објасниме преку конкретен пример со 4 променливи со цел полесно да го разбереме

Quine-McCluskey

$$F(A,B,C,D) = \Sigma m(0,2,5,6,7,8,10,12,13,14,15)$$

- Чекор 1. Генерирај примарни импликанти
  - Креирај табела на минтерми групирани во групи според бројот на единици

минтерми

Колона 1

0	0000	0 единици
2	0010	една единица
8	1000	
5	0101	две единици
6	0110	
10	1010	
12	1100	
7	0111	три единици
13	1101	
14	1110	
15	1111	четири единици

Quine-McCluskey

- Чекор 1. Генерирај примарни импликанти
  - Секоја комбинација од групата се споредува со секоја од следната група за да се најде пар кој се разликува само на една позиција. Од таквата споредба се креира втора колона, а местото на разликување се обележува со __
  - Секоја комбинација од групите која влегла во комбинација со друга се маркира со ✓

колона 1			колона 2	
0	0000	√	(0,2)	00-0
2	0010	√	(0,8)	-000
8	1000	√	(2,6)	0-10
5	0101	√	(2,10)	-010
6	0110	√	(8,10)	10-0
10	1010	√	(8,12)	1-00
12	1100	√	(5,7)	01-1
7	0111	√	(5,13)	-101
13	1101	√	(6,7)	011-
14	1110	√	(6,14)	-110
15	1111	√	(10,14)	1-10
			(12,13)	110-
			(12,14)	11-0
			(7,15)	-111
			(13,15)	11-1
			(14,15)	111-

Quine-McCluskey

- Чекор 1. Генерирај примарни импликанти
  - Секоја комбинација од групата се споредува со секоја од следната група за да се најде пар кој се разликува само на една позиција. Од таквата споредба се креира втора колона, а местото на разликување се обележува со __
  - Секоја комбинација од групите која влегла во комбинација со друга се маркира со ✓

колона 1			колона 2	
0	0000	√	(0,2)	00-0
2	0010	√	(0,8)	-000
8	1000	√	(2,6)	0-10
5	0101	√	(2,10)	-010
6	0110	√	(8,10)	10-0
10	1010	√	(8,12)	1-00
12	1100	√	(5,7)	01-1
7	0111	√	(5,13)	-101
13	1101	√	(6,7)	011-
14	1110	√	(6,14)	-110
15	1111	√	(10,14)	1-10
			(12,13)	110-
			(12,14)	11-0
			(7,15)	-111
			(13,15)	11-1
			(14,15)	111-

Quine-McCluskey

- Чекор 1. Генерирај примарни импликанти
  - Од колоната 2, од секоја група, повторно бараме парови од претходните и следната група кои се разликуваат на една позиција, и таа позиција пак ја обележуваме со _
  - Повторно се маркираат сите комбинации кои влегле во пар со друг.
  - При појава на дупликат комбинација, дупликатот се изостава/прецртува
  - Примарни импликанти кои останаа се: (0,2,8,10), (2,6,10,14), (5,7,13,15), (6,7,14,15), (8,10,12,14) и (12,13,14,15); или *B'D'*, *CD'*, *BD*, *BC*, *AD' u AB*

Колона 1			Колона 2			Колона 3	
0	0000	√	(0,2)	00-0	√	(0,2,8,10)	-0-0
2	0010	√	(0,8)	-000	√	(0,8,2,10)	-0-0
8	1000	√	(2,6)	0-10	√	(2,6,10,14)	–10
5	0101	√	(2,10)	-010	√	(2,10,6,14)	–10
6	0110	√	(8,10)	10-0	√	(8,10,12,14)	1–0
10	1010	√	(8,12)	1-00	√	(8,12,10,14)	1–0
12	1100	√	(5,7)	01-1	√	(5,7,13,15)	-1-1
7	0111	√	(5,13)	-101	√	(5,13,7,15)	-1-1
13	1101	√	(6,7)	011-	√	(6,7,14,15)	-11-
14	1110	√	(6,14)	-110	√	(6,14,7,15)	-11-
15	1111	√	(10,14)	1-10	√	(12,13,14,15)	11–
			(12,13)	110-	√	(12,14,13,15)	11–
			(12,14)	11-0	√		
			(7,15)	-111	√		
			(13,15)	11-1	√		
			(14,15)	111-	√		

Quine-McCluskey

- Чекор 2. Генерирај таблица на примарни импликанти

	$B'D'$ (0,2,8,10)	CD' (2,6,10,14)	BD (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
0	X					
2	X	X				
5			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

Quine-McCluskey

- Чекор 3. Редукција на табелата на примарни импликанти
- Итерација 1: Отстранување на есенцијални импликанти

	$B'D'(*)$ (0,2,8,10)	CD' (2,6,10,14)	$BD(*)$ (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
(○)0	X					
2	X	X				
(○)5			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

Колоната која соодветствува на редица во која има само еден X, е есенцијална импликанта. Колоните B'D' и BD ќе ги додадеме на крајното решение. Сите редици кои имаат X во колоните на примарните импликанти ќе ги изоставиме, т.е. прецртаме и нема да ги земеме во следниот чекор.

Quine-McCluskey

- Чекор 3. Редукција на табелата на примарни импликанти
- Доминација на редица

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X
14	X	X	X	X

Редицата 14 доминира врз редиците 12 и 6 бидејќи содржи X на сите места каде и 6 и 12 имаат X. Доколку откриеме доминантна редица ја прецртуваме и не ја земаме во предвид во следниот чекор.

Quine-McCluskey

- Чекор 3. Редукција на табелата на примарни импликанти
- Доминација на колона

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X

Колоната CD’ доминира врз колоната BC, има X во сите редици каде и BC има X. Истото важи и за AD’ и AB. Овде имаме случај на взаемна доминантност (ко-доминантност). Доминирачката колона се елиминира, т.е. прецртува. Ако имаме ко-доминантни колони, тогаш една прецртуваме (но никако не двете).

Quine-McCluskey

- Чекор 3. Редукција на табелата на примарни импликанти
- Итерација 2. Откривање на секундарните есенцијални импликанти (**)

	$CD'(**)$	$AD'(**)$
	$(2,6,10,14)$	$(8,10,12,14)$
$(\circ)6$	X	
$(\circ)12$		X

CD’ + AD’

Овие импликанти се прецртуваат и влегуваат во крајното решение. Со тоа во случајот алгоритмот е завршен

$$F = B'D' + BD + CD' + AD'$$