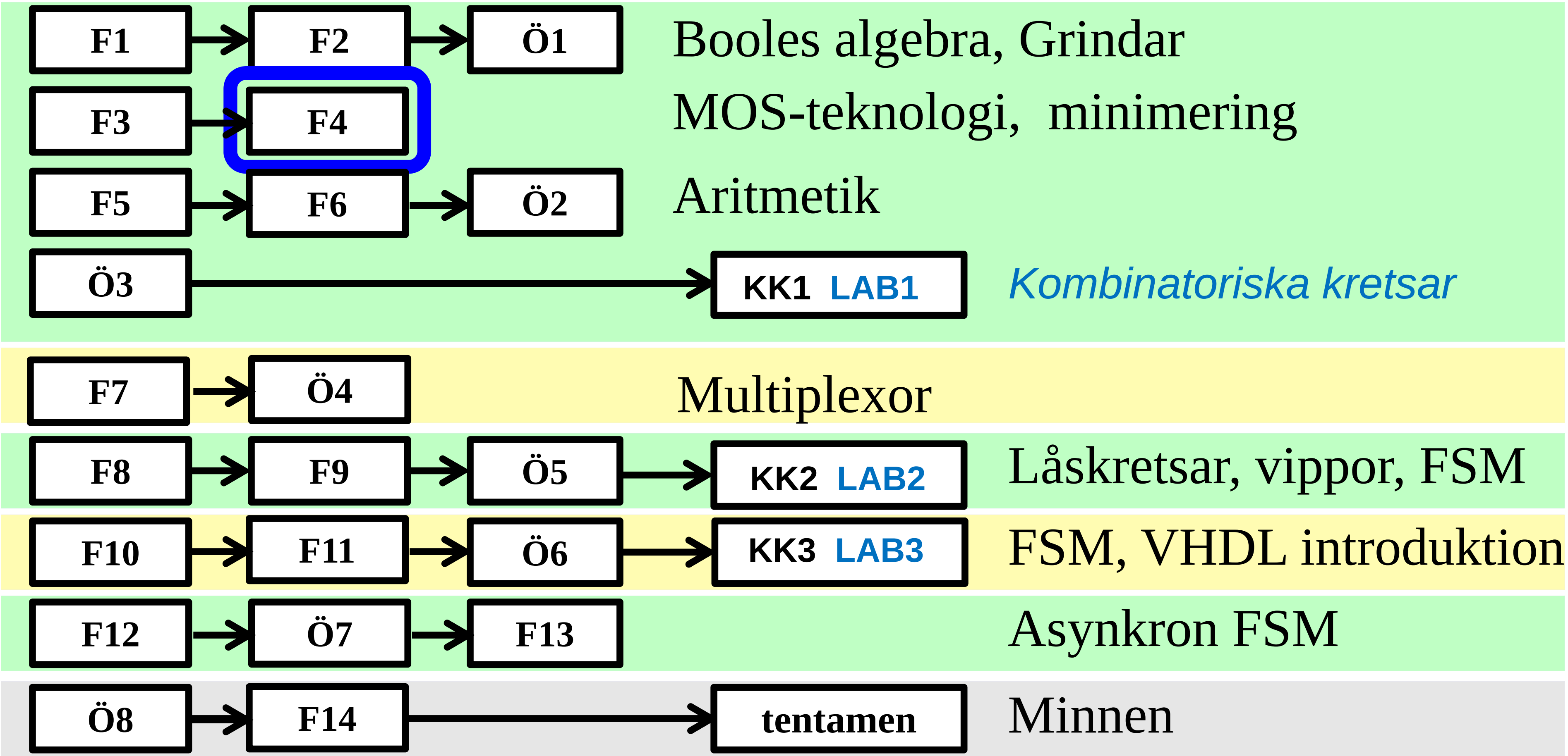
Digital Design IE1204

Föreläsningsbilder av William Sandqvist

F4 Karnaugh-diagrammet, två- och fler-nivå minimering

Carl-Mikael Zetterling
bellman@kth.se

IE1204 Digital Design



Föreläsningar och övningar bygger på varandra! Ta alltid igen det Du missat!
Läs på i förväg – delta i undervisningen – arbeta igenom materialet efteråt!

Detta har hänt i kursen …

Talsystem: Decimala, hexadecimala, oktala, binära

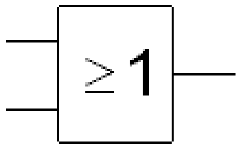
$$(175,5)_{10} = (AE.8)_{16} = (256.4)_8 = (10101110.1)_2$$

AND OR NOT XOR XNOR Sanningstabell, mintermer Maxtermer PS-form SP-form deMorgans lag Bubbelgrindar Fullständig logik NAND NOR

CMOS grindar, standardkretsar

Mintermer

OR



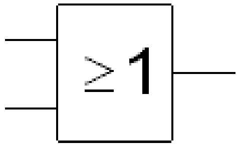
	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

En minterm är en produktterm som innehåller **alla** variabler och som anger den kombination av 1:or och 0:or som tillsammans gör att termen antar värdet 1.

SP-form med tre mintermer.

$$f = \sum m(1,2,3) = \bar{x}_1 x_0 + x_1 \bar{x}_0 + x_1 x_0$$

Minimering med boolesk algebra

OR 

$$f = \sum m(1,2,3) = \bar{x}_1 x_0 + x_1 \bar{x}_0 + x_1 x_0$$

	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

$$= \bar{x}_1 x_0 + x_1(\bar{x}_0 + x_0) =$$

$$= \bar{x}_1 x_0 + x_1(1 + vsh) =$$

$$= \bar{x}_1 x_0 + x_1(1 + x_0) =$$

$$= \bar{x}_1 x_0 + x_1 + x_1 x_0 =$$

$$= x_0(\bar{x}_1 + x_1) + x_1 =$$

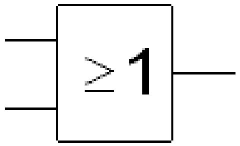
$$= \boxed{x_0 + x_1}$$

Som väntat!

Förenkling med boolesk algebra

Maxtermer

OR

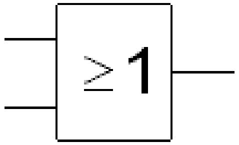


	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

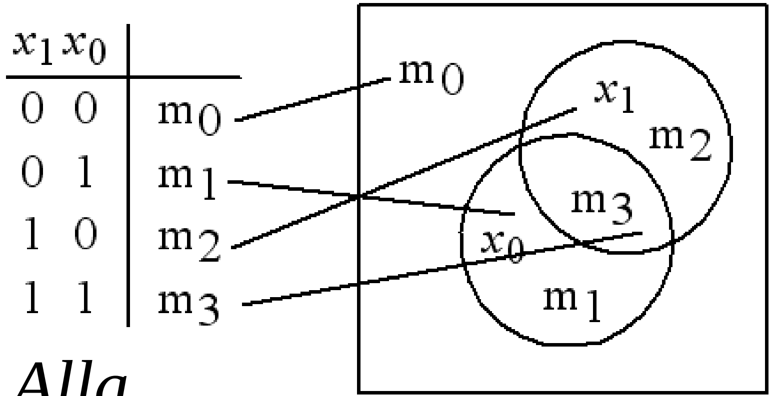
En maxterm är en summa-faktor som innehåller **alla** variabler och som anger den kombination av 1:or och 0:or som tillsammans gör att faktorn antar värdet 0.

$$f = \prod M(0) = \boxed{x_0 + x_1}$$

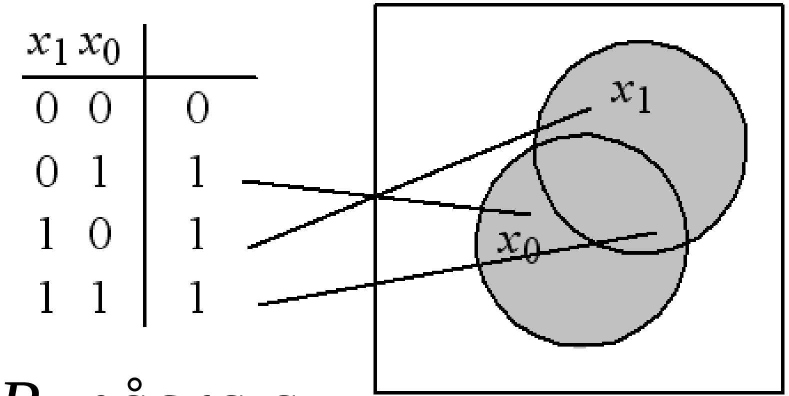
Denna gång fick vi det enkla uttrycket med en maxterm direkt!

OR 

Venn-diagram



Alla mintermer



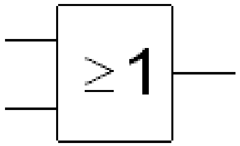
OR, några av mintermerna

$\boxed{\bar{x}_1 x_0 + x_1 \bar{x}_0 + x_1 x_0}$

$\boxed{(x_0 + x_1)}$

I ett Venn-diagram kan man se att en funktion kan uttryckas på en mängd olika sätt, men det är *inte lätt* att se vad som är optimalt. En annan fråga är också hur man kan rita Venn-diagram för mer än tre variabler ???

Grafisk minimeringsmetod

OR 

	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

$x_1 \backslash x_0$	0	1
0	0	1
1	1	1

$x_1 \backslash x_0$	0	1
0	m_0	m_1
1	m_2	m_3

$$\boxed{m_2 + m_3} = x_1\bar{x}_0 + x_1x_0 =$$

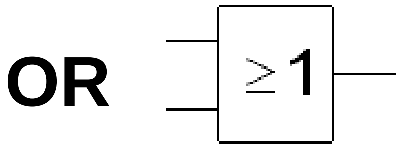
$$= x_1(\bar{x}_0 + x_0) = x_1$$

$$\boxed{m_1 + m_3} = \bar{x}_1x_0 + x_1x_0 =$$

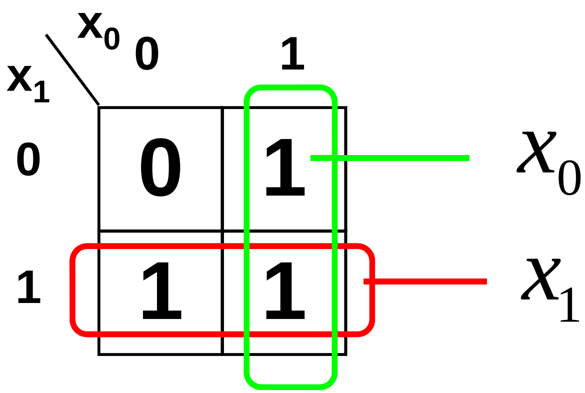
$$= x_0(\bar{x}_1 + x_1) = x_0$$

$$f = x_1 + x_0$$

Grafisk minimeringsmetod



	x_1	x_0	f
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1



$$f = x_1 + x_0$$

Gör ”hoptagningar” av par av rutor med 1:or (horisontellt eller vertikalt), behåll de variabler i produktermerna som är gemensamma.

Grindfunktioner på grafisk form

AND

$x_1 \backslash x_0$	0	1
0	0	0
1	0	1

$$x_1 \cdot x_0$$

OR

$x_1 \backslash x_0$	0	1
0	0	1
1	1	1

$$x_1 + x_0$$

XOR

$x_1 \backslash x_0$	0	1
0	0	1
1	1	0

$$\overline{x_1} x_0 + x_1 \overline{x_0}$$

NAND

$x_1 \backslash x_0$	0	1
0	1	1
1	1	0

$$\overline{x_1} + \overline{x_0} = \overline{\overline{\overline{x_1} + \overline{x_0}}} = \overline{x_1 \cdot x_0}$$

NOR

$x_1 \backslash x_0$	0	1
0	1	0
1	0	0

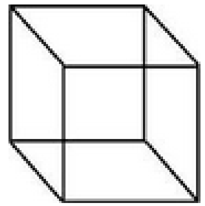
$$\overline{x_1} \cdot \overline{x_0} = \overline{\overline{\overline{x_1} \cdot \overline{x_0}}} = \overline{x_1 + x_0}$$

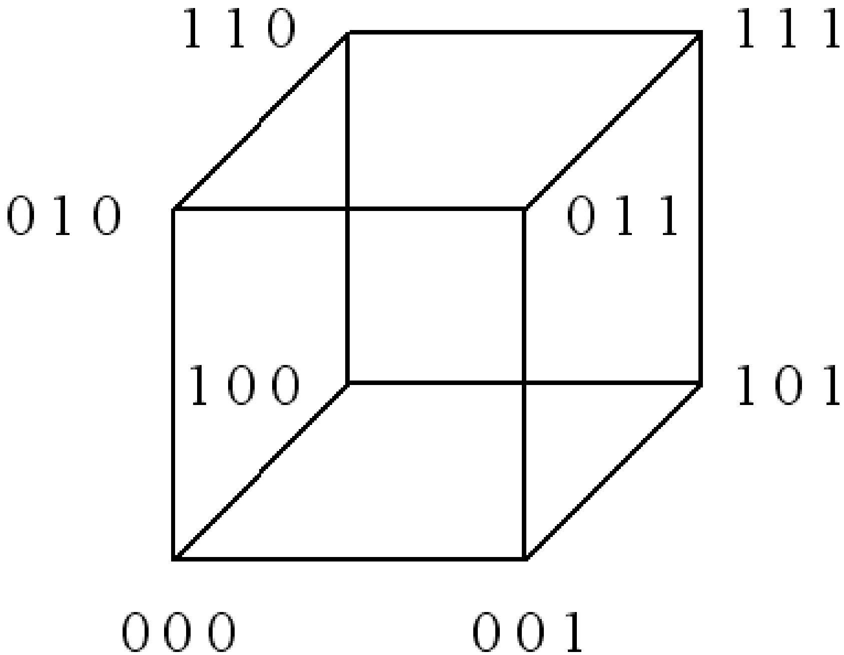
XNOR

$x_1 \backslash x_0$	0	1
0	1	0
1	0	1

$$\overline{x_1}\,\overline{x_0} + x_1 x_0$$

3D boolesk talrymd





Kubens hörn är kodade med Gray-kod.
För hörn som är "grannar" är det bara skillnad i en variabel.

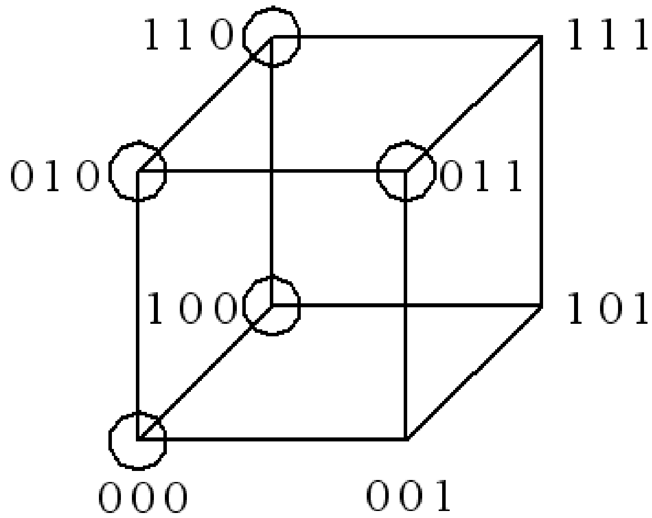
ÖH 3.4 kubrepresentation

$$f(x_2, x_1, x_0) = \sum m(0,2,3,4,6) =$$

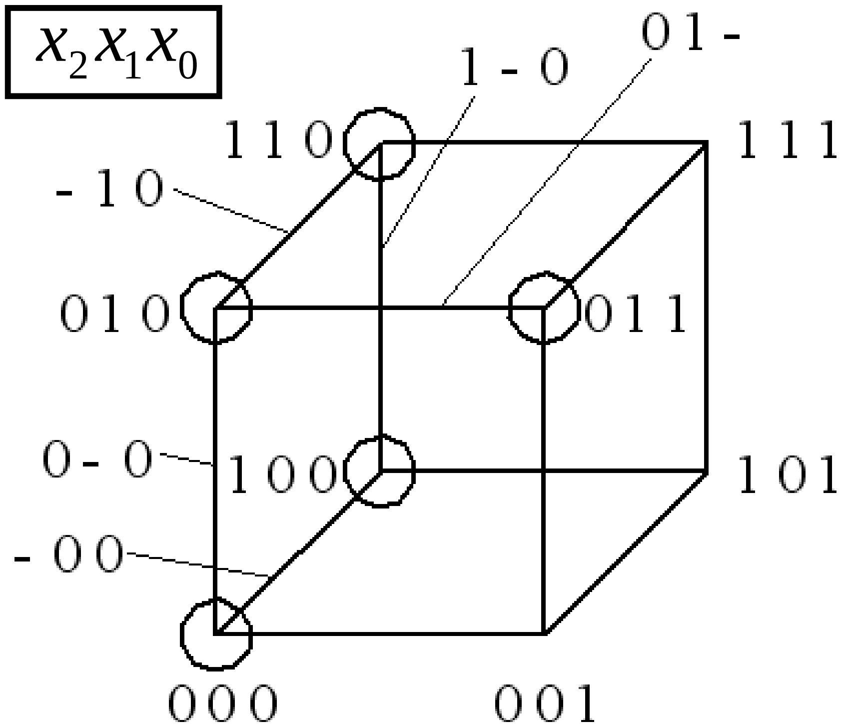
$$= \overline{x}_2\,\overline{x}_1\,\overline{x}_0 + \overline{x}_2\,\overline{x}_1\,x_0 + \overline{x}_2\,x_1\,x_0 + x_2\,\overline{x}_1\,\overline{x}_0 + x_2\,x_1\,\overline{x}_0$$

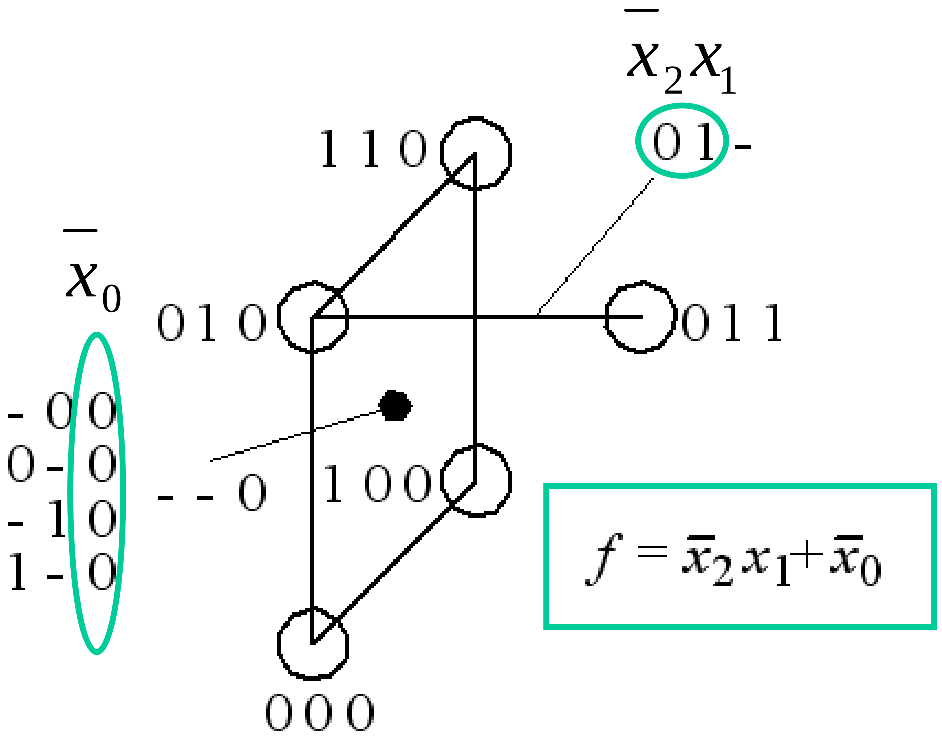
Så här representerar man en funktion av tre variabler, som en 3D kub med Gray-kodade hörn.

x_2	x_1	x_0		
0	0	0	1	$\bar{x}_2\bar{x}_1\bar{x}_0$
0	0	1	0	
0	1	0	1	$\bar{x}_2 x_1\bar{x}_0$
0	1	1	1	$\bar{x}_2 x_1 x_0$
1	0	0	1	$x_2\bar{x}_1\bar{x}_0$
1	0	1	0	
1	1	0	1	$x_2 x_1\bar{x}_0$
1	1	1	0	



ÖH 3.4 minimering med kub





En **yta** representeras av ***en*** variabel, en **kant** av en produktterm med ***två*** variabler, och ett **hörn** en minterm med ***tre*** variabler. Kub-metoden kan generaliseras till ***"Hyperkuber"*** med godtyckligt antal variabler.

Hyperkuber

**Ett hörn är en 0-dimension subspace,
en sida kallas för en 1-dimension subspace,
en yta kallas för en 2-dimension subspace,
en kub kallas för en 3-dimension subspace…**

Det finns minimeringsmetoder för hyperkuber, och de går att tillämpa för valfritt antal variabler! Metoderna med hyperkuber är lämpade för datoralgoritmer.

Graykoden är en speglad binärkod

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

0
1

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

0
1
———
1
0

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

00
01
11
10

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

00
01
11
10
10
11
01
00

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

000
001
011
010
110
111
101
100

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

000
001
011
010
110
111
101
100
100
101
111
110
010
011
001
000

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Graykoden är en speglad binärkod

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000
1000
1001
1011
1010
1110
...

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* "hyperhörnen" i "hyperkuber" med!

Graykoden är en speglad binärkod

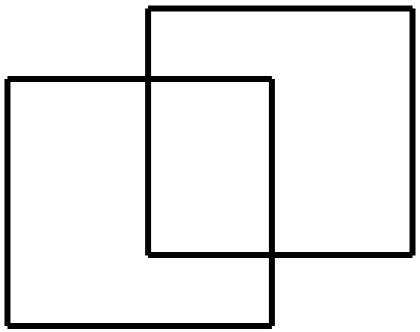
```
00000
00001
00011
00010
00110
00111
00101
00100
01100
01101
01111
01110
01010
01011
01001
01000
11000
11001
11011
11010
11110
...
```

Man kan lätt ta fram den Graykod med ett godtyckligt antal bitar som behövs för att *numrera* ”hyperhörnen” i ”hyperkuber” med!

Och så vidare …

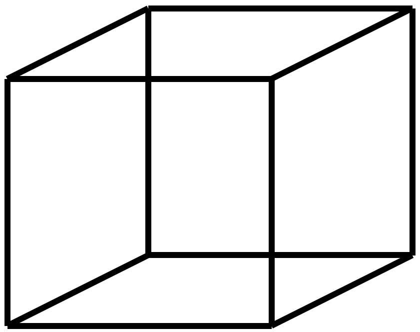
Hur ritar man en 3D-kub?

Man ritar två st 2D-kuber (= kvadrater), och sammanbinder sedan deras hörn.



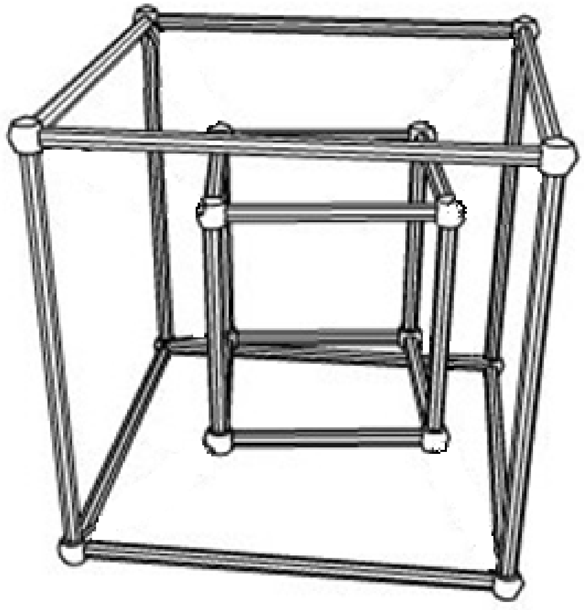
Hur ritar man en 3D-kub?

Man ritar två st 2D-kuber (= kvadrater), och sammanbinder sedan deras hörn.



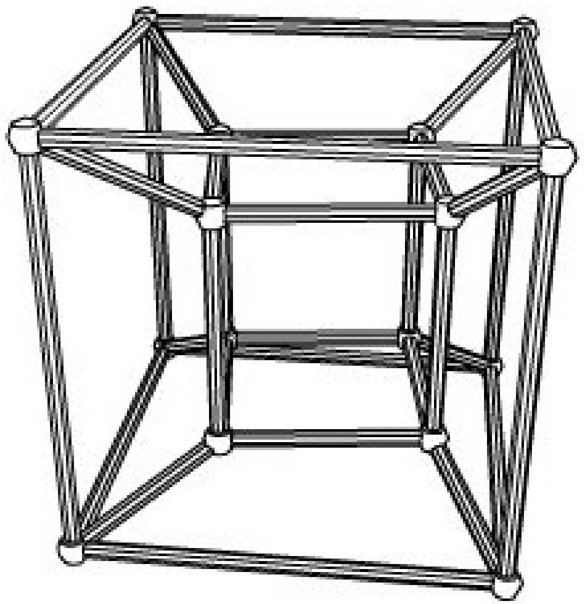
Hur ritar man en 4D-kub?

Man ritar *två* st 3D-kuber (kuber), och samman-binder sedan deras hörn.



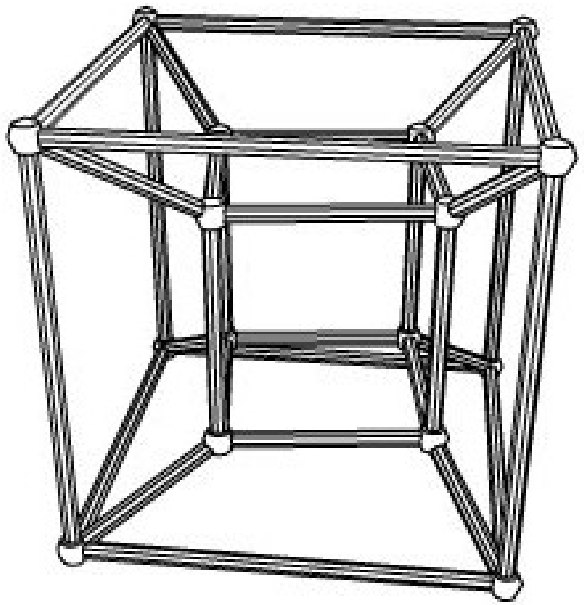
Hur ritar man en 4D-kub?

Man ritar *två* st 3D-kuber (kuber), och samman-binder sedan deras hörn.

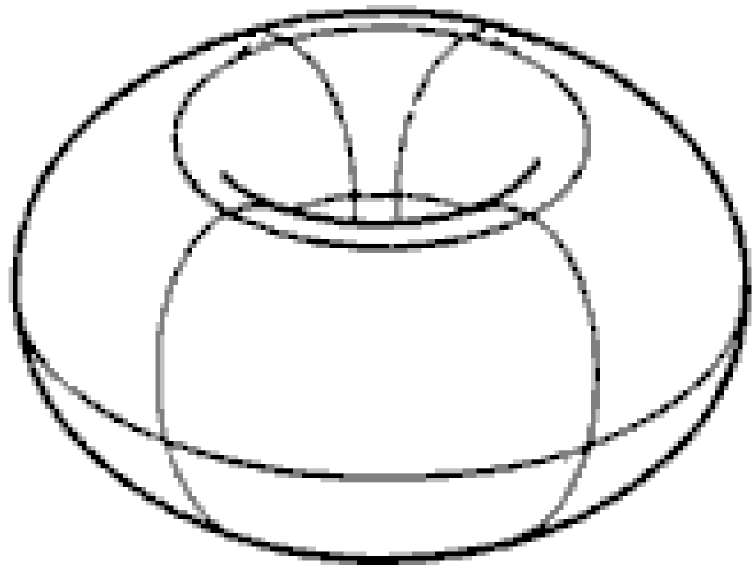


4D-kuben i Paris

4D-Donut



≡

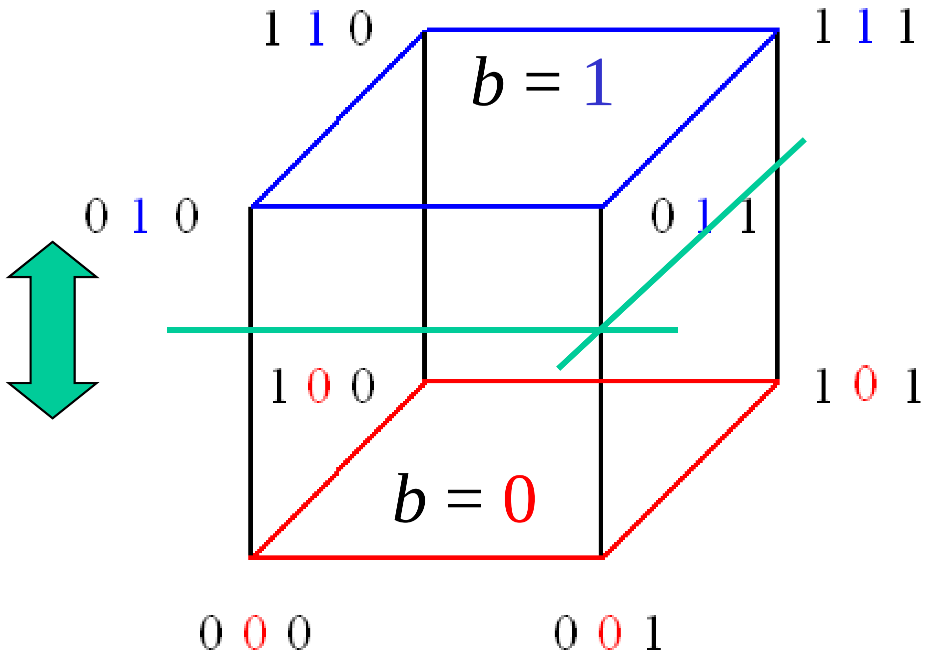


4D-hyperkuben kan också representeras med en toroid, en donut.

4D-kuber i USA

3D-Kub $\Rightarrow$ 2D-diagram

$a\ b\ c$



Gray-kod $\rightarrow$ spegla

$a \backslash bc$	00	01	11	10
0	000	001	011	010
1	100	101	111	110

Karnaugh-diagrammet

Grafisk metod för minimering ”med papper och penna” av mindre booleska funktioner
(för upp till sex variabler)

Maurice Karnaugh

(The Map for Synthesis of Combinational Logic Circuits, AIEE, Nov. 1953)

En funktion av fyra variabler a b c d

Sanningstabellen med 11 st 1:or och 5 st 0:or. Funktionen kan ut-tryckas på SP-form med 11 st mintermer eller på PS-form med 5 st maxtermer.

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

$$f(a,b,c,d) = \sum(0,1,2,3,4,5,6,7,8,10,13)$$

$$f = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}\bar{c}d + \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + \bar{a}bcd + a\bar{b}\bar{c}\bar{d} + a\bar{b}c\bar{d} + ab\bar{c}d$$

$$f(a,b,c,d) = \prod(9,11,12,14,15)$$

$$f = (\bar{a}+b+c+\bar{d})\cdot(\bar{a}+b+\bar{c}+\bar{d})\cdot(\bar{a}+\bar{b}+c+d)\cdot(\bar{a}+\bar{b}+\bar{c}+d)\cdot(\bar{a}+\bar{b}+\bar{c}+\bar{d})$$

4D-Kub ⇒ 2D-diagram

Karnaughdiagrammet är sanningstabellen men med en annan ordning. Lägg märke till numreringen!

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

ab \ cd	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

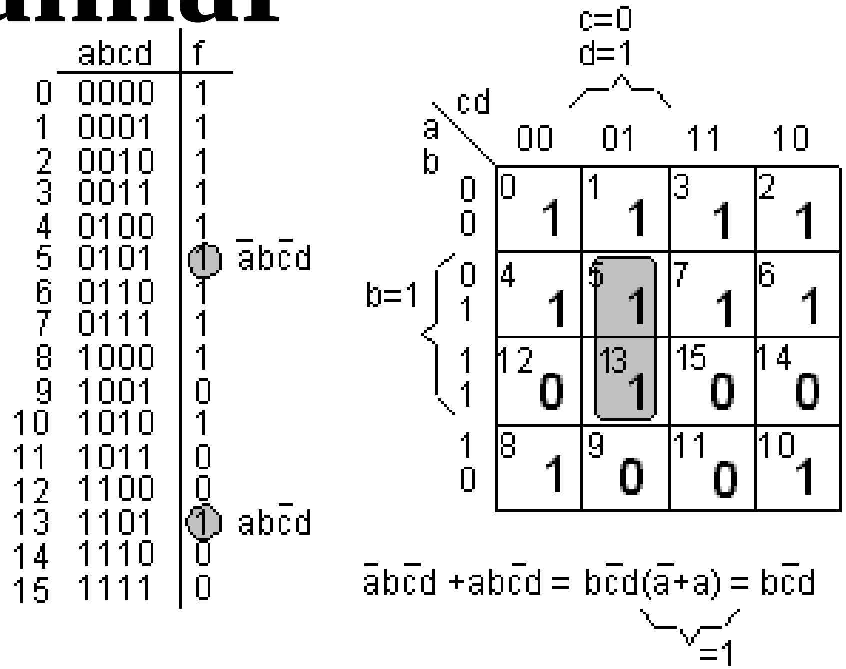
Rutorna är ordnade så att endast en bit ändras mellan två vertikala eller horisontella rutor.

Denna ordning kallas för Gray-kod.

$$b\bar{c}d$$

Två ”grannar”

Rutorna "5" och "13" är "grannar" i Karnaughdiagrammet.
De svarar mot *två* mintermer med *fyra* variabler, och i figuren visas hur de med Booles algebra, kan reduceras till *en* term med *tre* variabler.



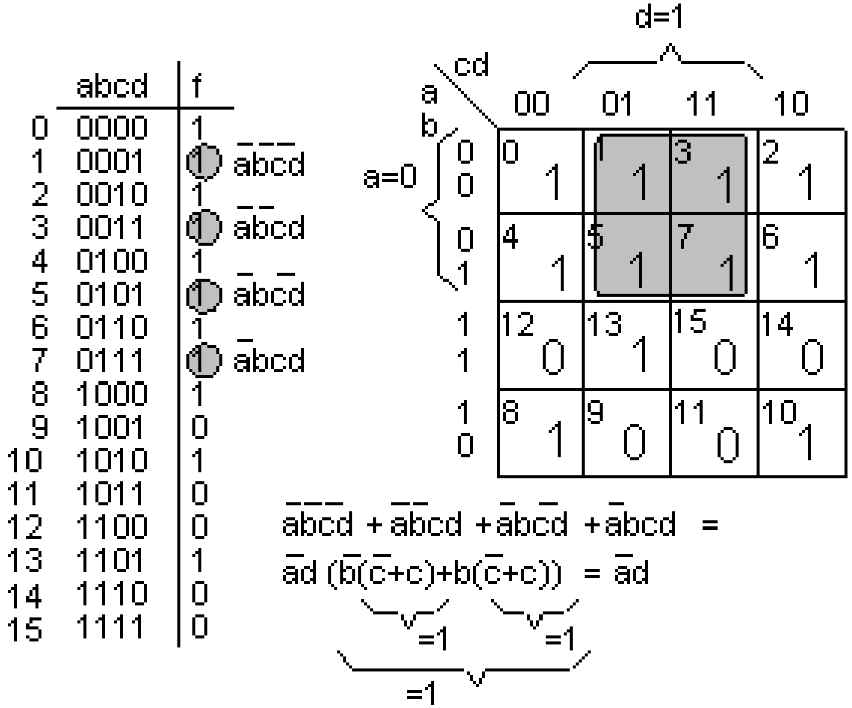
Det de två rutorna har gemensamt är att b=1, c=0 och d=1, och den reducerade termen uttrycker precis detta.

Överallt i Karnaughdiagrammet där man hittar två ettor som är "grannar" (vertikalt eller horisontellt) kan man reducera de min-termerna till *det som är gemensamt* för de två rutorna.

Detta kallas för en *hoptagning*.

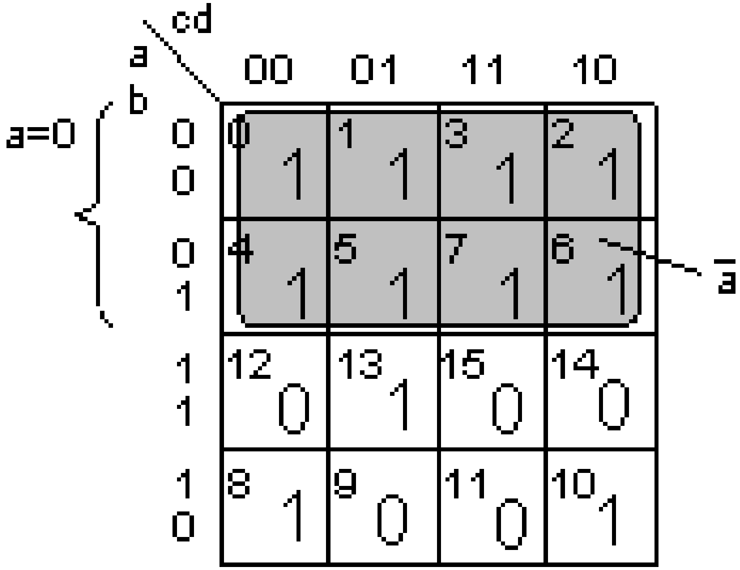
Fyra ”grannar” $\overline{a}\,d$

Rutorna "1" "3" "5" "7" är en grupp av fyra rutor med ettor som ligger som "grannar" till varandra. Även här går de fyra mintermerna att reducera till en term som uttrycker det som är gemensamt för rutorna, nämligen att a=0 och d=1.



Överallt i Karnaughdiagrammet där man hittar sådana grupper av fyra ettor kan man göra sådana förenklingar, *hoptagningar*.

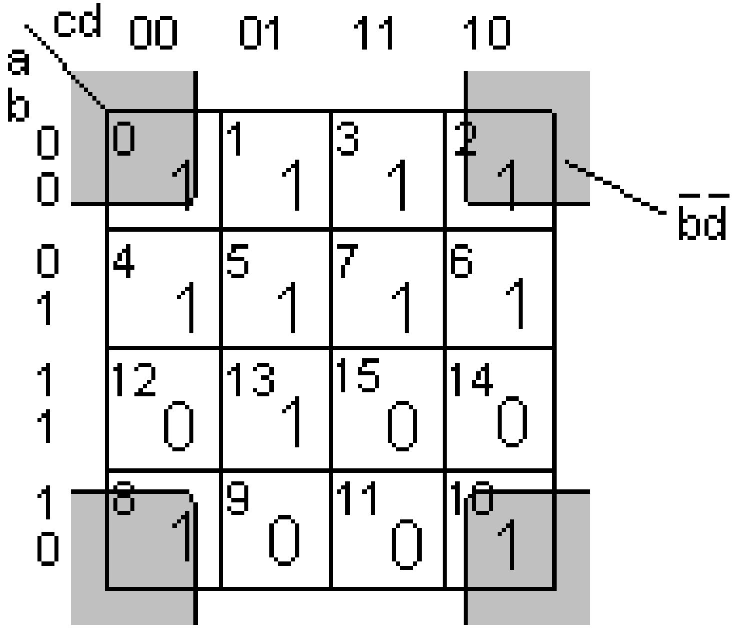
Åtta ”grannar” $\overline{a}$

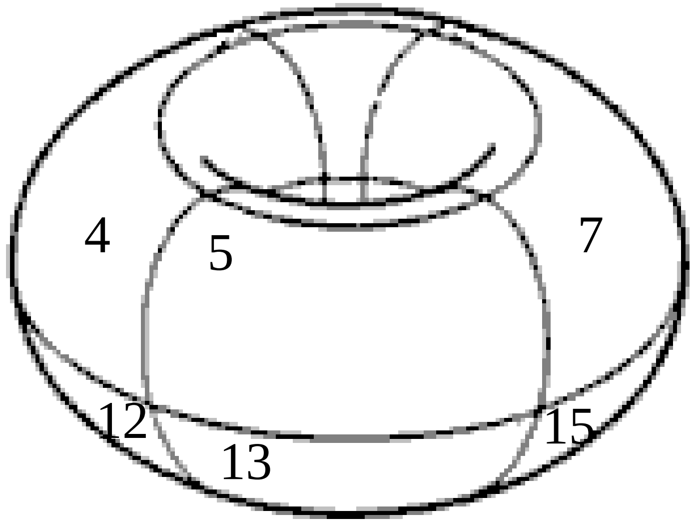


Alla grupper av 2, 4, 8, (... 2^N dvs. med jämna 2-potenser) rutor, som innehåller ettor kan reduceras till en term, med "det som är gemensamt", en *hoptagning*.

Karnaugh - toroid

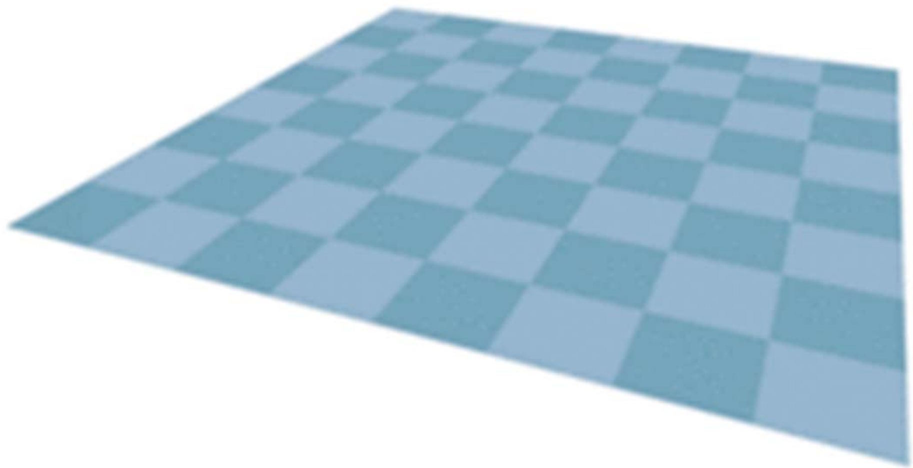
$\bar{b}\,\bar{d}$





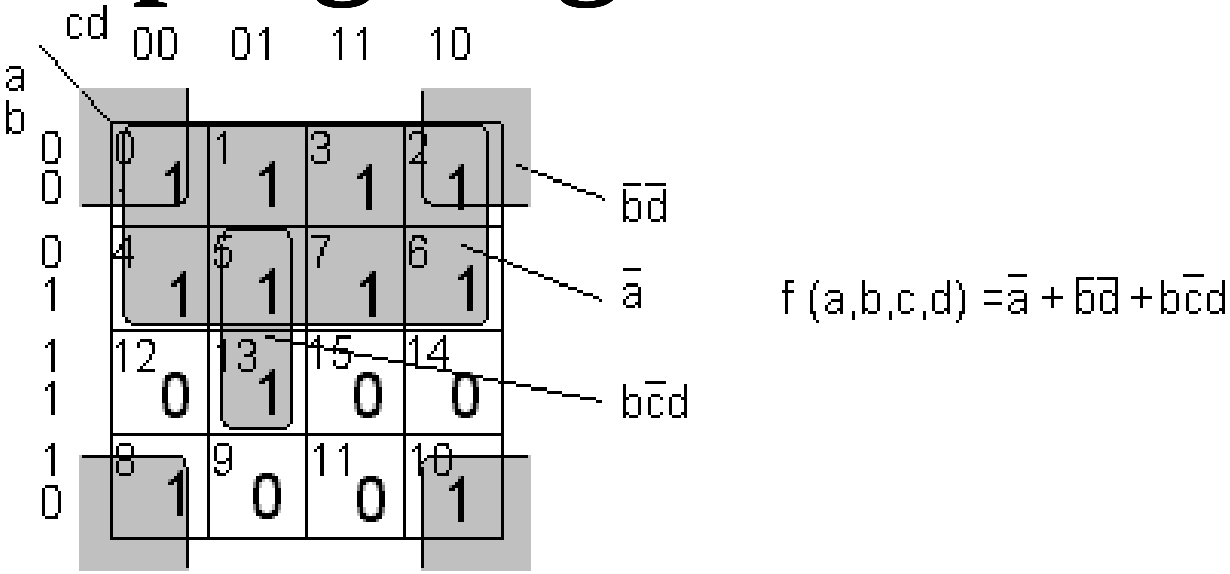
Egentligen bör man avbilda Karnaughdiagrammet på en toroid (en donut). Når man en kant, så börjar diagrammet om från den motsatta sidan! Ruta 0 är således "granne" med ruta 2, men även "granne" med ruta 8 som är granne med ruta 10. De fyra ettorna i hörnen har b=0 och d=0 gemensamt och kan därför bilda en hoptagning.

Karnaugh - toroid



Bästa hoptagningar?

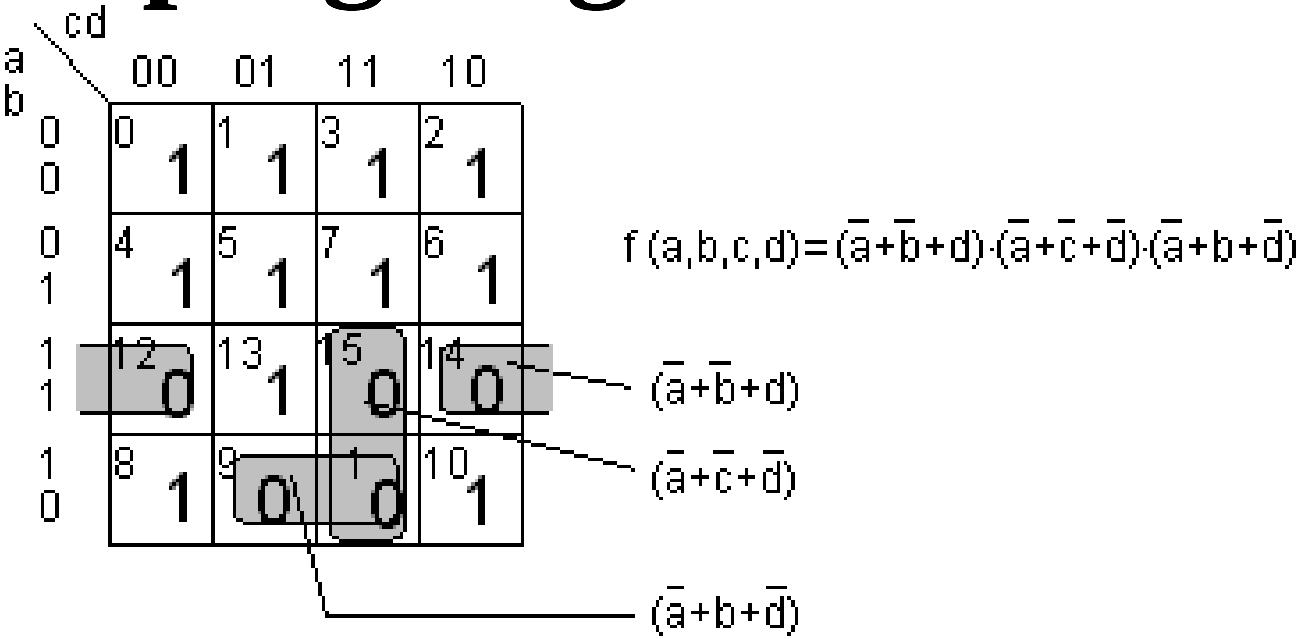
Man söker efter så stora hoptagningar som möjligt. I exemplet finns det en hoptagning med åtta ettor (rutorna 0,1,3,2,4,5,7,6). Hörnen (0,2,8,10) är en hoptagning av fyra ettor.



Två av rutorna (0,10) har redan tagits med i den första hoptagningen, men inget hindrar att en ruta bir medtagen flera gånger.
Alla ettor måste med i funktionen, antingen i en hoptagning, eller som en minterm. Ettan i ruta 13 kan bilda en hoptagning med ettan i ruta 5, någon större hoptagning finns tyvärr inte för denna etta.

$$\boxed{f(a,b,c,d) = \bar{a} + \bar{b}\,\bar{d} + b\,\bar{c}\,d}$$

Hoptagningar av 0:or



Karnaughdiagrammet är också användbart för hoptagning av 0:or. Hoptagningarna kan omfatta samma antal rutor som i fallet med hoptagning av 1:or. I detta exempel kan 0:orna tas ihop i par med sina "grannar". Maxtermerna förenklas till *det som är gemensamt* för rutorna.

$$f(a,b,c,d)=(\overline{a}+\overline{b}+d)(\overline{a}+\overline{c}+\overline{d})(\overline{a}+b+\overline{d})$$

Andra variabelantal

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

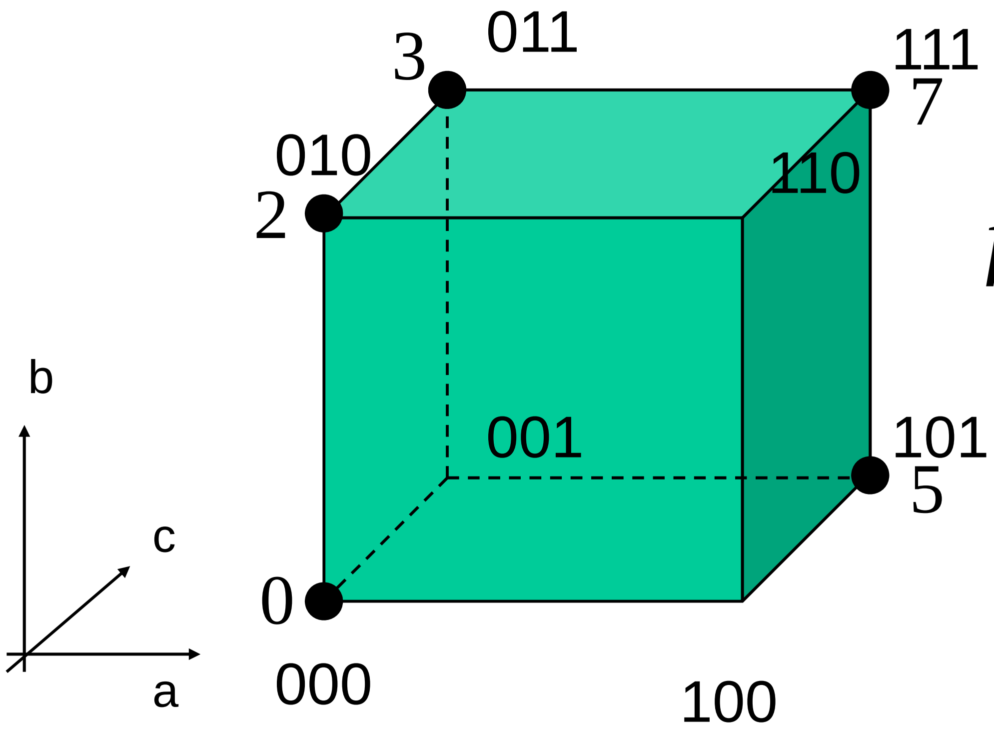
c \ ba	00	01	11	10
0	0	1	3	2
1	4	5	7	6

	b	a
0	0	0
1	0	1
2	1	0
3	1	1

b \ a	0	1
0	0	1
1	2	3

Karnaughdiagram med tre och två variabler är också användbara.

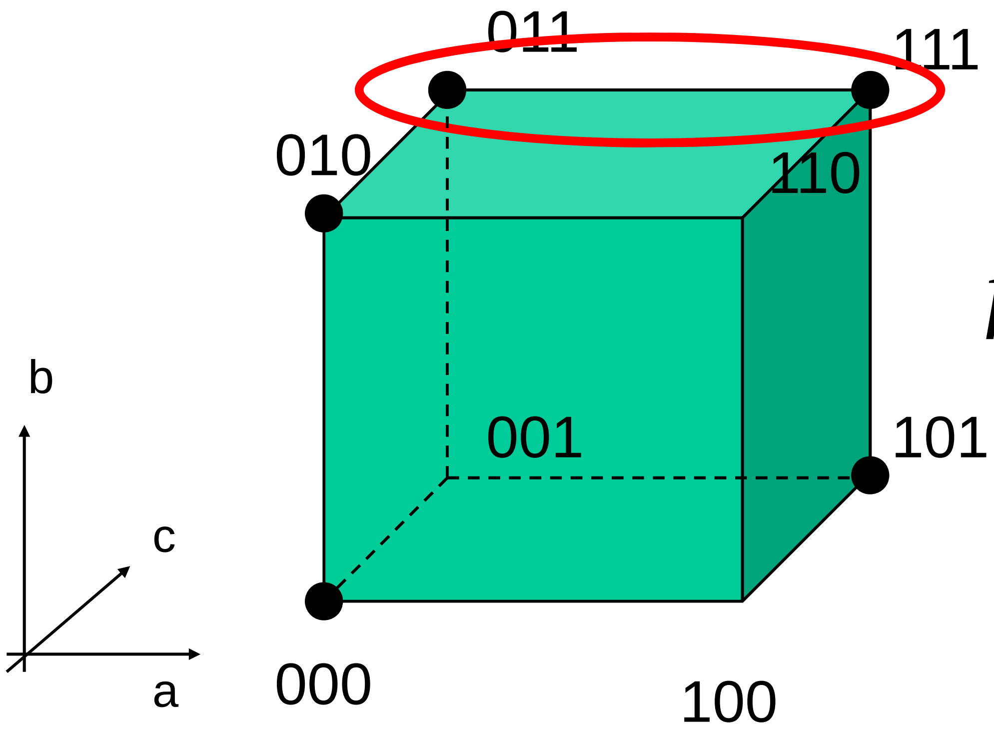
Minimeringsexempel



$$f(a,b,c)=\sum m(0,2,3,5,7)$$

a \ bc	00	01	11	10
0	0: 1	1:	3: 1	2: 1
1	4:	5: 1	7: 1	6:

Implikanter



Ringa in min-termer som ligger bredvid varandra

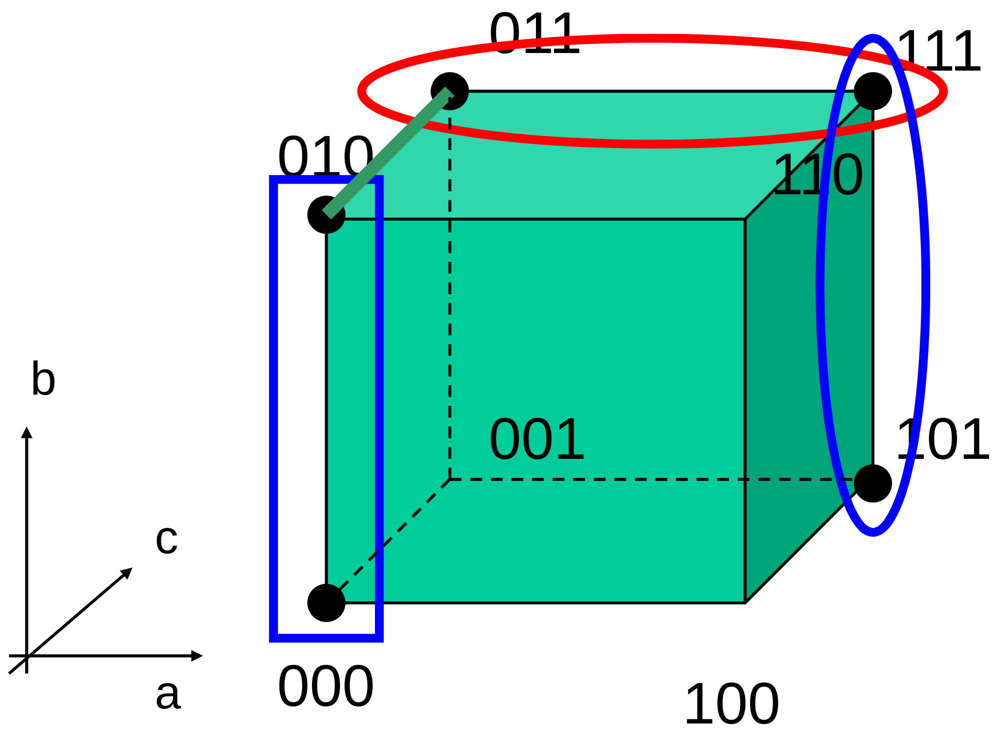
$$f(a,b,c)=\sum m(0,2,3,5,7)$$

a \ bc	00	01	11	10
0	1		1	1
1		1	1	

Lite implikant-terminologi

- **Implikant** - en inringning av min-termer
- **Prim-implikant** - en inringning av min-termer som inte kan göras större.
- **Essentiell** prim-implikant - en maximal inringning av min-termer som måste vara med för att funktionen skall täckas.
- **Redundant** prim-implikant – en maximal inringning av min-termer som inte nödvändigtvis måste vara med för att funktionen skall täckas.

Implikanter



Redundanta implikanter - bägge är inte nödvändiga (en måste vara med) för att täcka funktionen.

a \ bc	00	01	11	10
0	1		1	1
1		1	1	

$$f = \bar{a}\,\bar{c} + bc + ac$$

$$f = \bar{a}\,\bar{c} + \bar{a}b + ac$$

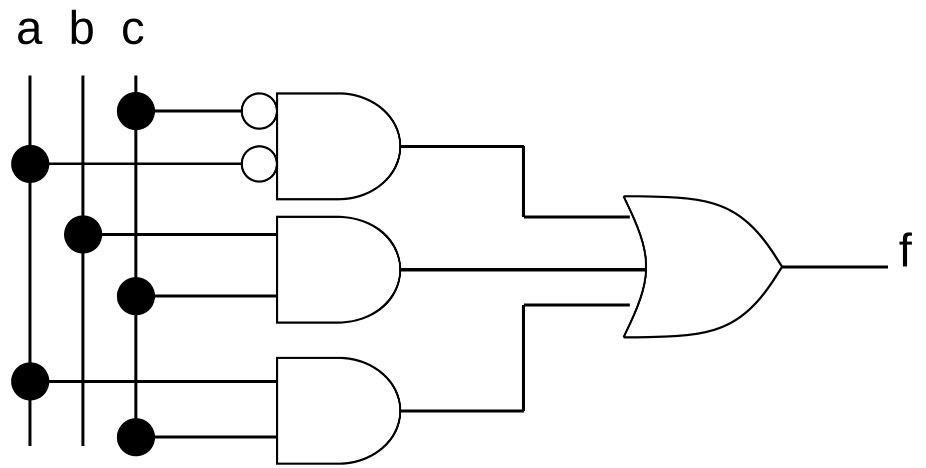
$$f(a,b,c) = \sum m(0,2,3,5,7)$$

Två-nivå minimering

Minimal Summa-Produkt Implementation

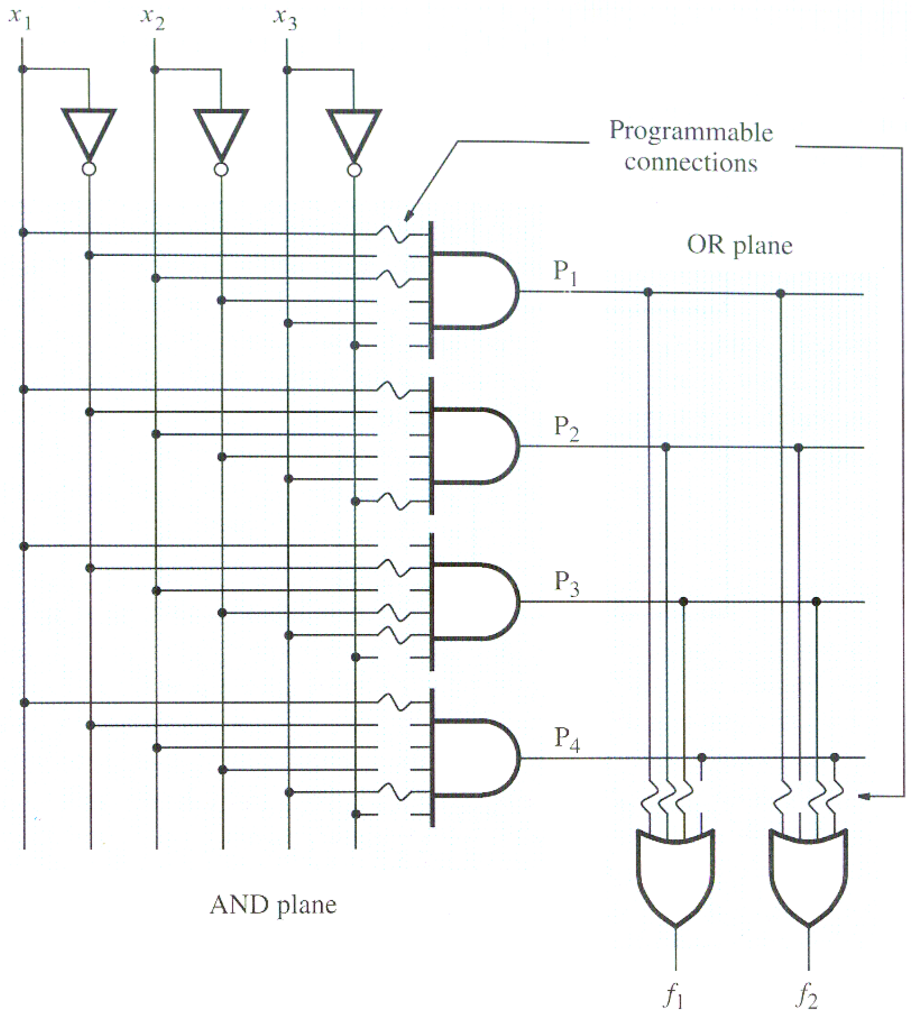
$$f(a,b,c) = \sum m(0,2,3,5,7)$$

a \ bc	00	01	11	10
0	1		1	1
1		1	1	



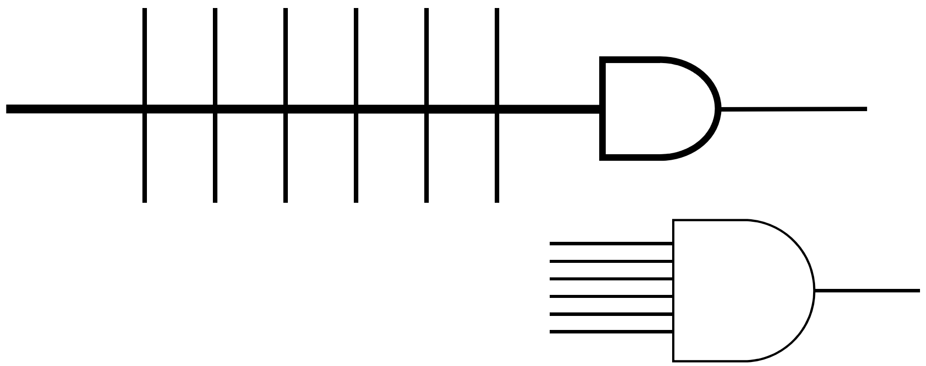
$$f = \bar{a}\,\bar{c} + bc + ac$$

Programmable Logic Array (PLA)

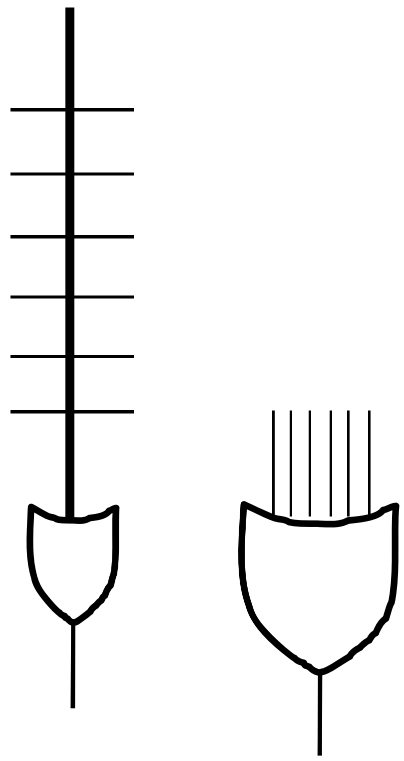


Programerbar AND och OR – matris. Programmeringen består av att man ”bränner av” anslutningar man inte vill ha kvar.
(nu föråldrad teknik)

Programmable Logic Array (PLA)



Grind-matriserna har så många ingångar att man brukar rita grindarna med ett ”förenklat ritsätt”

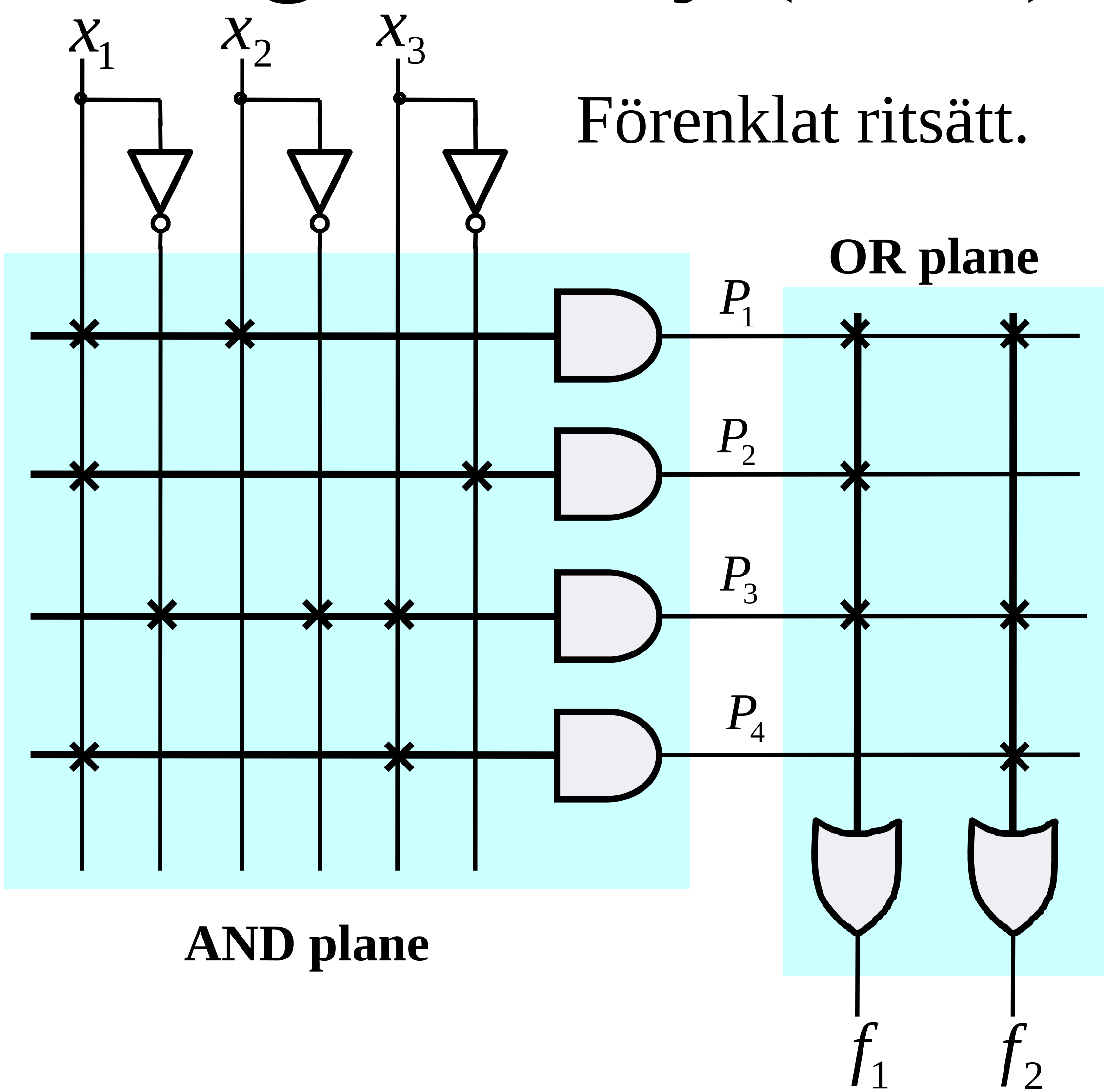


Programmable Logic Array (PLA)

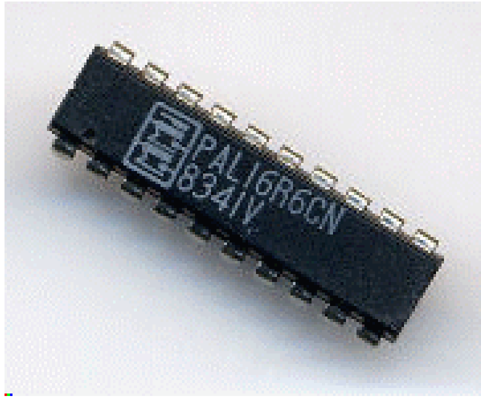
PLA. Både **AND**- och **OR**-matriserna är programmer-bara.

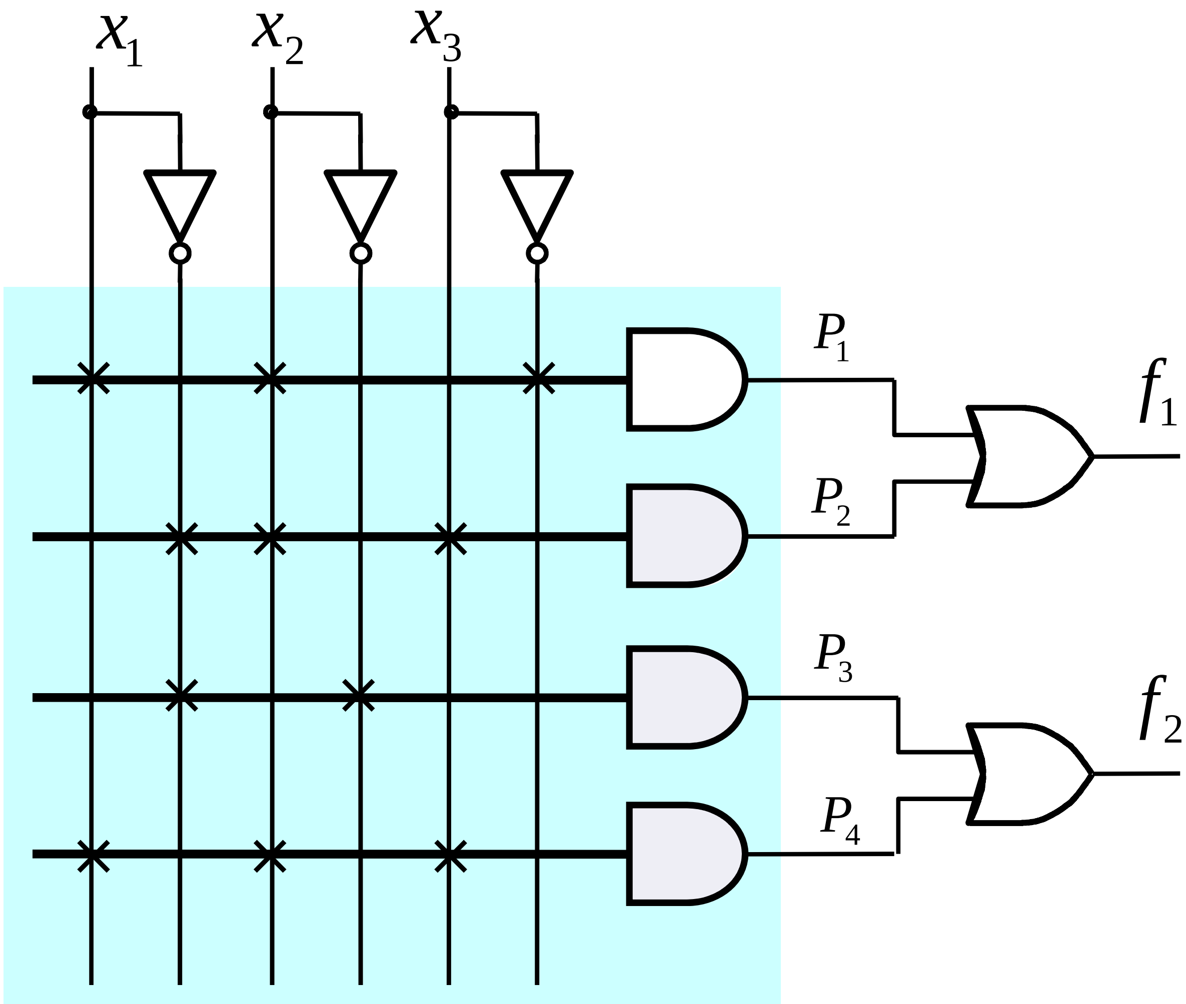
(en kostsam flexibilitet)

Förenklat ritsätt.



Programmable Array Logic (PAL)

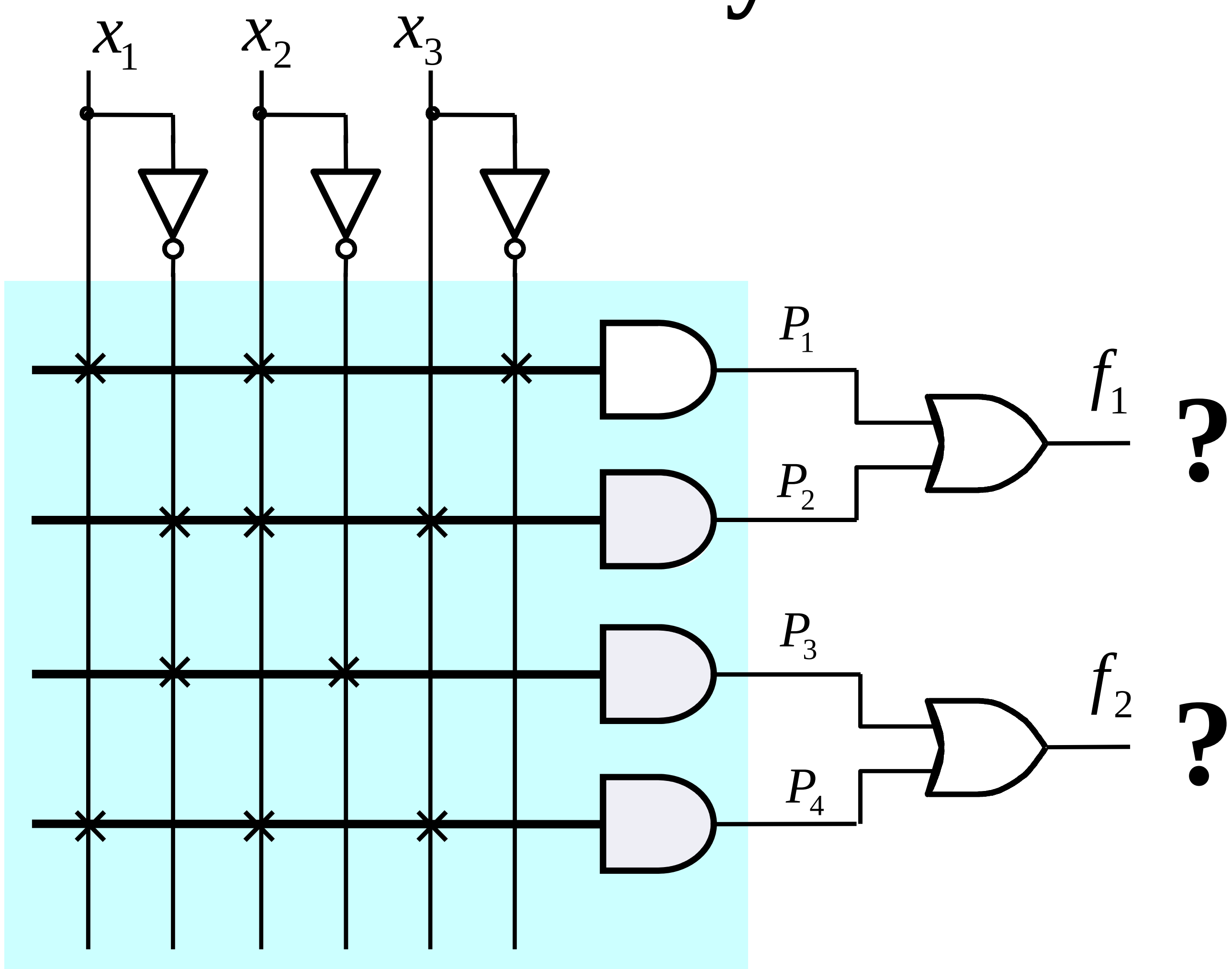




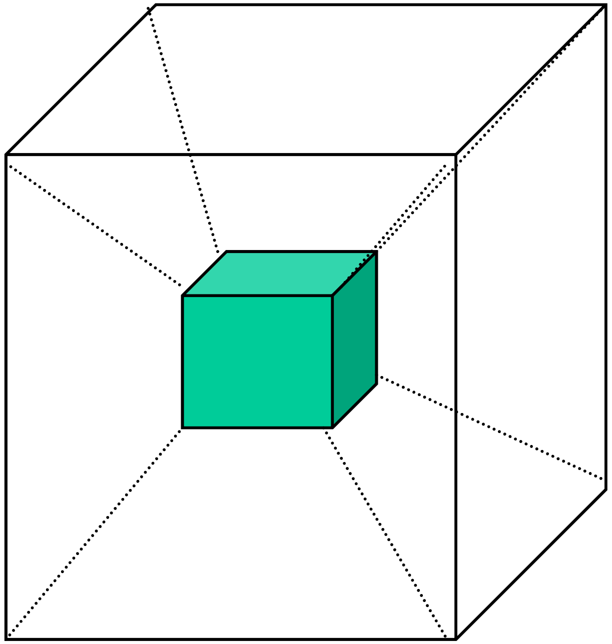
PAL. Bara **AND**-matrisen är programmerbar.

(en ekonomisk kompromisslösning)

PAL, vilka funktioner döljer sig bakom kryssen?



Sub-kub vid fyra varibler

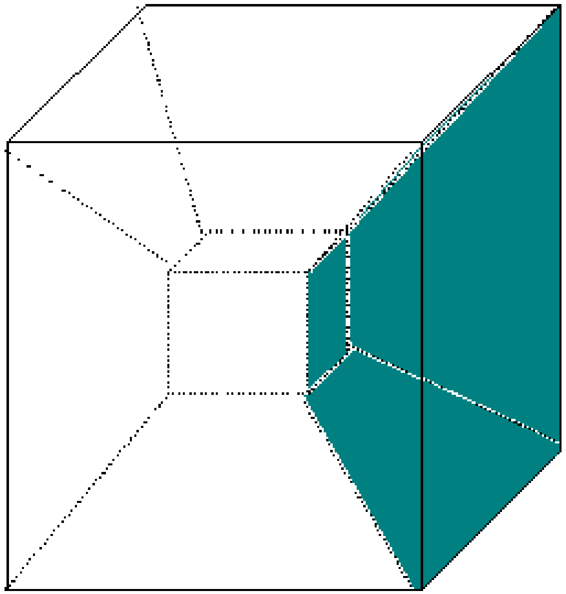


x_3x_2 \ x_1x_0	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

Vi ringar alltid in en hel sub-kub (så stor som möjligt)!!!

$$f = x_3$$

Sub-kub vid fyra varibler



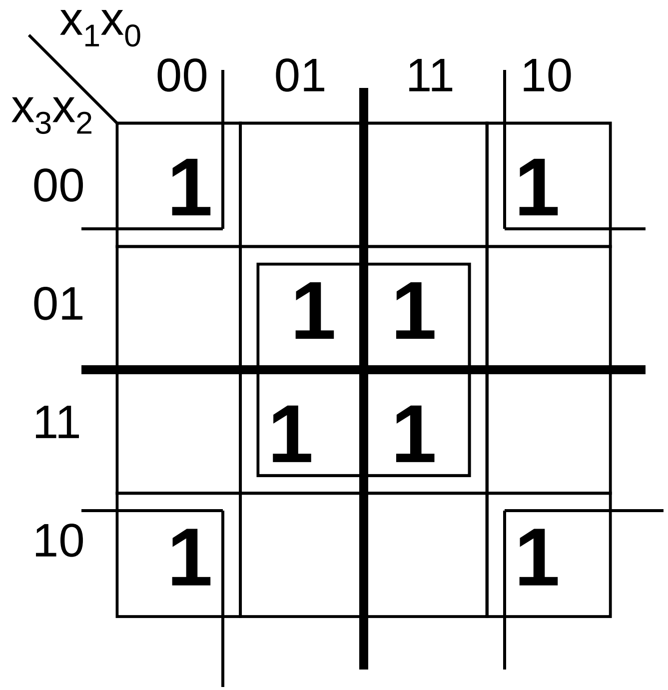
x_3x_2 \ x_1x_0	00	01	11	10
00		1	1	
01		1	1	
11		1	1	
10		1	1	

Vi ringar alltid in en hel sub-kub (så stor som möjligt)!!!

$$f = x_0$$

XOR kan vara till hjälp

Om två fyra-inringningar *inte* kan bilda en åtta-inringning kan kanske XOR/XNOR-funktionen vara till hjälp.



Detta under förutsättningen att det finns en särskilt effektiv implementering av XOR-funktionen.

$$f = \bar{x}_2 \bar{x}_0 + x_2 x_0 = \overline{x_2 \oplus x_0}$$

4D Mintermernas ordning ...

$$f(x_3, x_2, x_1, x_0) = \sum m(0,3,7,14)$$

MSB

LSB

x_3x_2 \ x_1x_0	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

5D fem variabler

x_4

$x_4 = 0$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			**1**	
11				
10				

$x_4 = 1$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			**1**	
11				
10				

Samma i båda diagrammen, oberoende av x_4. $\bar{x}_3 x_2 x_1 x_0$

Spegling med 5 variabler

Spegla!

x_3x_2 \ $x_4x_1x_0$	**0**00	**0**01	**0**11	**0**10	**1**10	**1**11	**1**01	**1**00
00								
01			1			1		
11								
10								

$$\bar{x}_3 x_2 x_1 x_0$$

och med en annan ordning …

x_4x_3 \ $x_2x_1x_0$	000	001	011	010	110	111	101	100
00						1		
01								
11								
10						1		

$$\overline{x}_3 x_2 x_1 x_0$$

Karnaugh-diagram med 6 variabler

x_4

x_5

$x_5 = 0$, $x_4 = 0$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			1	
11				
10				

$x_5 = 0$, $x_4 = 1$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			1	
11				
10				

$x_5 = 1$, $x_4 = 0$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			1	
11				
10				

$x_5 = 1$, $x_4 = 1$

x_3x_2 \ x_1x_0	00	01	11	10
00				
01			1	
11				
10				

$$\bar{x}_3 x_2 x_1 x_0$$

Oberoende av x_5 och x_4.

Spegling med 6 variabler

OBS!

$x_5x_3x_2$ \ $x_4x_1x_0$	000	001	011	010	110	111	101	100
000								
001			1			1		
011								
010								
110								
111								
101			1			1		
100								

$$\bar{x}_3 x_2 x_1 x_0$$

och med en annan ordning …

$x_5x_4x_3$ \ $x_2x_1x_0$	000	001	011	010	110	111	101	100
000						1		
001								
011								
010						1		
110						1		
111								
101								
100						1		

$$\overline{x}_3 x_2 x_1 x_0$$

Hoptagningar med 0:or

x_3x_2 \ x_1x_0	00	01	11	10
00	1	1	1	1
01	1	0	1	1
11	1	1	1	1
10	1	1	1	1

0:an som minterm – helt fel!

$$\overline{f} = \overline{x_3} x_2 \overline{x_1} x_0$$

invertera, så blir det rätt!

$$f = \overline{\overline{f}} = \overline{\overline{x_3} x_2 \overline{x_1} x_0} =$$

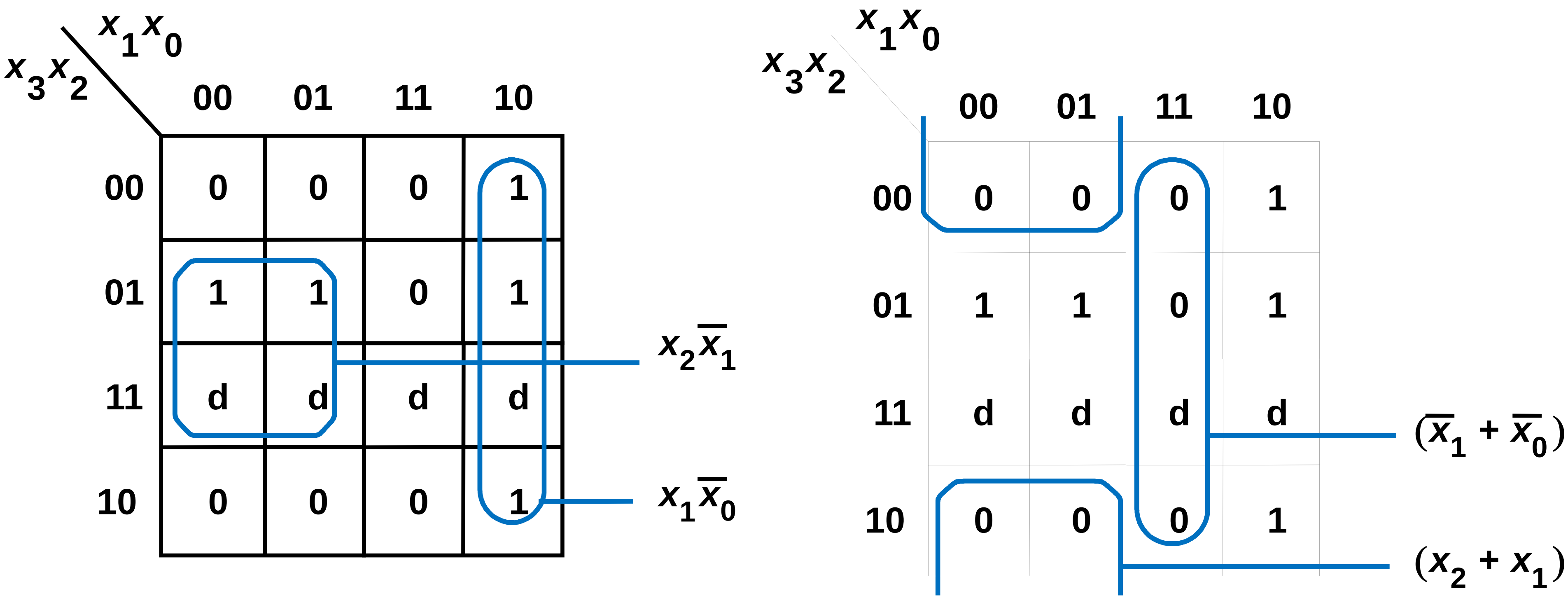
$$= \{dM\} = (x_3 + \overline{x_2} + x_1 + \overline{x_0})$$

Ringa in nollorna om dom är färre än ettorna!!!

Don’t care

- Ofta kan man förenkla specifikationen för den logiska funktionen eftersom man vet att vissa kombinationer kan aldrig förekomma
- För dessa kombinationer använder vi värdet ”don’t care”
- Det finns olika symboler för ”don’t care” i bruk
  - ’d’, ’D’, ’-’, ’Φ’, ’x’

Ofullständig funktionsspecifikation

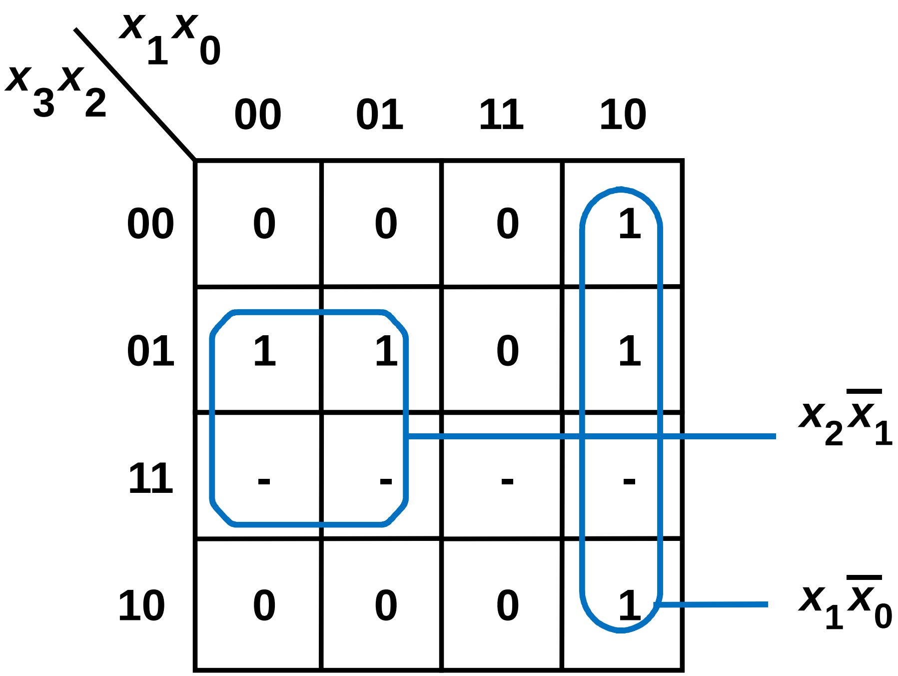


(a) SOP implementation **(b) POS implementation**

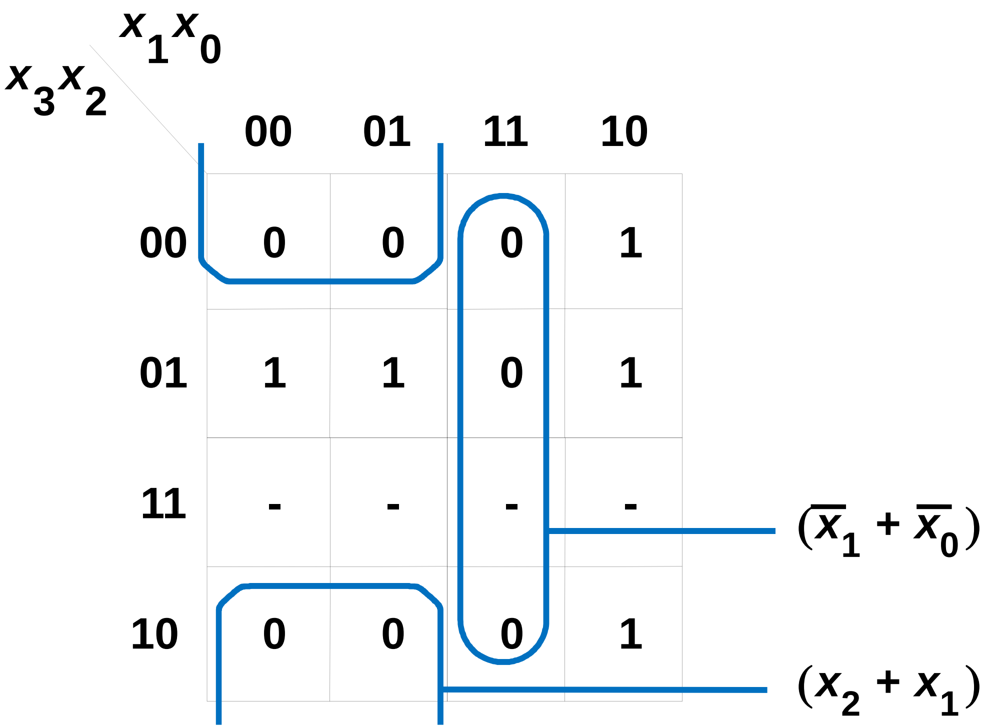
Två implementeringar av funktionen

$f\,(x_3,\ldots,x_0) = \Sigma\, m(2, 4, 5, 6, 10) + D(12, 13, 14, 15).$

Annan notation (-) …



(a) SOP implementation



(b) POS implementation

Två implementeringar av funktionen
$f(x_3, \ldots, x_0) = \Sigma\, m(2, 4, 5, 6, 10) + D(12, 13, 14, 15)$.

Funktioner med flera utgångar

f_0

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	1	1
01	0	0	0	1
11	0	0	0	1
10	1	0	1	1

f_1

x_3x_2 \ x_1x_0	00	01	11	10
00	1	0	1	1
01	0	0	0	1
11	1	0	0	1
10	1	0	1	1

$$f_1 = f_0 + \boxed{x_3 \overline{x_0}}$$

Olika utgångar kan dela prim-implikanter!!!

Fler-nivå minimering

Behöver man fler nivåer än **två**?

Man kan ju realisera alla kombinatoriska kretsar med ***två*** nivåer (**AND-OR**, **OR-AND**)

– Antagandet är då att alla ingångar finns också i inverterade form (som i **PAL**, **PLA**)

Varför fler-nivå logik?

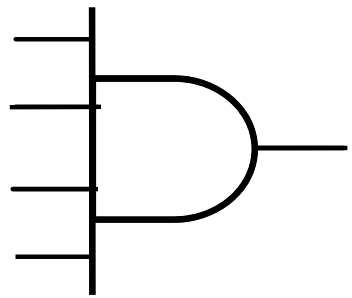
- Antalet ingångar i en krets kan vara begränsad
- Hög fan-in leder till långa fördröjningar
- Det kan vara mer kostnadseffektiv att använda en implementering med fler nivåer

Två strategier för fler-nivå logik

1 Faktorisering

2 Funktionell dekomposition

Faktorisering



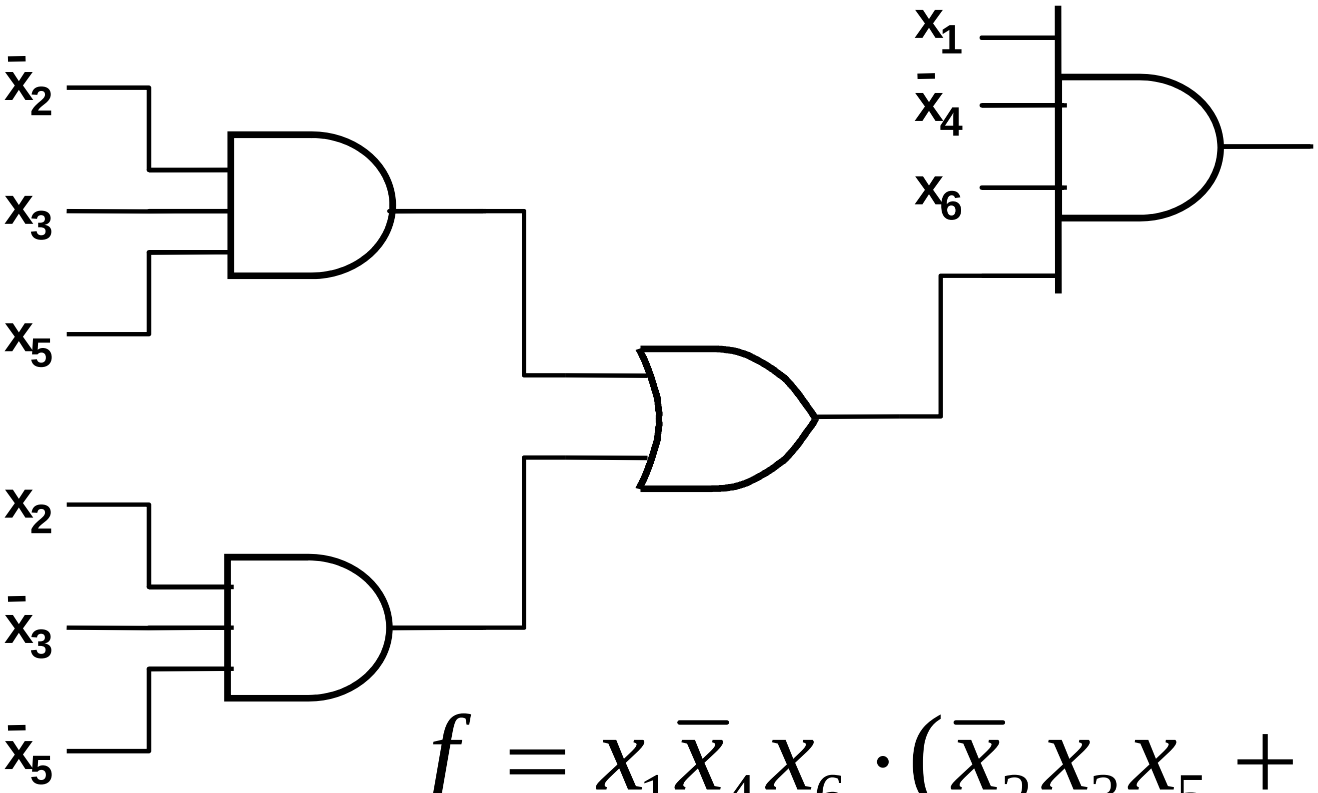
$$f = x_1\bar{x}_2x_3\bar{x}_4x_5x_6 + x_1x_2\bar{x}_3\bar{x}_4\bar{x}_5x_6$$

Vi har bara AND med som *mest* 4 ingångar?

Bryt ut $x_1\bar{x}_4x_6$

$$f = x_1\bar{x}_4x_6 \cdot (\bar{x}_2x_3x_5 + x_2\bar{x}_3\bar{x}_5)$$

Faktorisering



$$f = x_1 \bar{x}_4 x_6 \cdot (\bar{x}_2 x_3 x_5 + x_2 \bar{x}_3 \bar{x}_5)$$

Funktionell dekomposition

- Man kan ofta reducerar komplexiteten av en logisk funktion genom att återanvända funktioner flera gånger
- För implementeringen betyder det att man återanvänder en ”krets” vid flera ställen i sin konstruktion

Funktionell dekomposition

$$f = \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_4 + \bar{x}_1 \bar{x}_2 x_4$$

Funktioner kan ses som sammansatta av underfunktioner.

Funktionell dekomposition

Bryt ut x_3 respektive x_4

$$f = \bar{x}_1 x_2 x_3 + x_1 \bar{x}_2 x_3 + x_1 x_2 x_4 + \bar{x}_1 \bar{x}_2 x_4$$

$$= (\underbrace{\boxed{\bar{x}_1 x_2 + x_1 \bar{x}_2}}_{\text{XOR}\; = g}) x_3 + (\underbrace{\boxed{x_1 x_2 + \bar{x}_1 \bar{x}_2}}_{\text{XNOR}\; = \bar{g}}) x_4$$

$$f = g x_3 + \bar{g} x_4$$

Hade Du kommit på det? …

(XOR och XNOR)

$$XOR = \overline{XNOR}$$

XOR

$x_1 \backslash x_2$	0	1
0	0	1
1	1	0

$$\overline{x_1}x_2 + x_1\overline{x_2}$$

XNOR

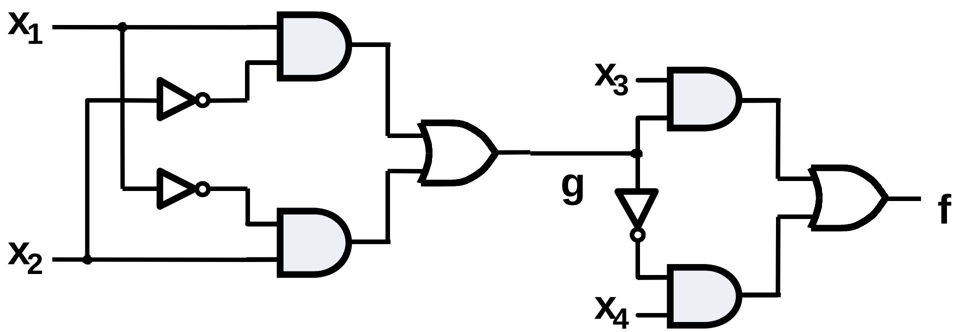
$x_1 \backslash x_2$	0	1
0	1	0
1	0	1

$$\overline{x_1}\,\overline{x_2} + x_1x_2$$

Med en effektiv implementering av XOR-grindar (få transistorer) så kan man se lösningar i Karnaughdiagrammet även när det *inte* går att göra några hoptagningar!

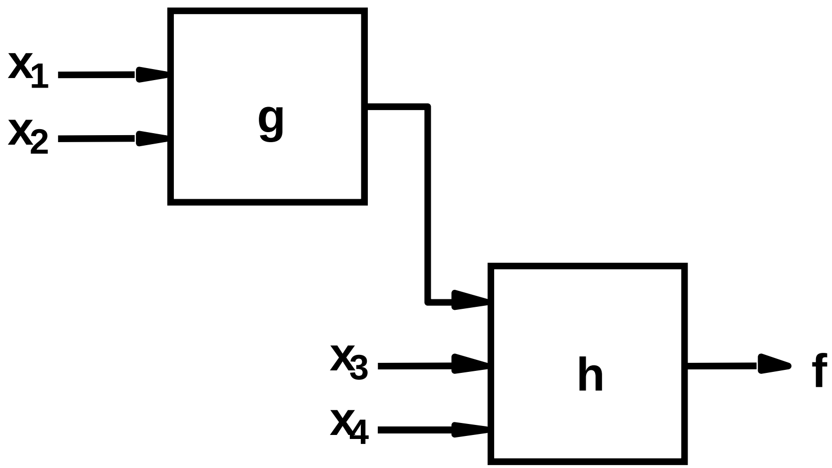
Funktionell dekomposition

Implementeringen



$$g = \bar{x}_1 x_2 + x_1 \bar{x}_2$$

$$f = g x_3 + \bar{g} x_4$$



Algoritmer för minimering

- Karnaugh-minimering ger en bra inblick hur man kan minimera logiska funktioner
- Men för att minimera komplexa funktioner med hjälp av datorstöd finns det bättre algoritmer
- Kapitel 4.9 och 4.10 i Brown/Vranesic ger en introduktion i minimeringsalgoritmer (för den intresserade studenten)

Sammanfattning

- Karnaugh-diagrammet är ett bra verktyg för att minimera logiska funktioner med få variabler
- Det finns algoritmer för både två-nivå och fler-nivås-minimering