Logische algebra

De blokken combinatorische logica vormen een belangrijk deel van de digitale elektronica. In een blok combinatorische logica wordt van een aantal digitale ingangssignalen een aantal digitale uitgangssignalen gevormd. Het doel van dit hoofdstuk is een algemene werkwijze voor te stellen om dit op een systematische wijze te doen.
Hierbij maken we gebruik van de Booleaanse algebra. Deze laat ons toe de manier te vinden om de nodige logica op de meest eenvoudige wijze voor te stellen. De meest eenvoudige voorstelling van de logica leidt ook tot de beste hardware implementatie namelijk zowel de snelste implementatie als de implementatie die het minste oppervlakte verbruikt.
In de praktijk heeft zo'n combinatorisch blok meerdere ingangen en meerdere uitgangen. We kunnen dit probleem echter steeds opsplitsen in deelproblemen met dezelfde ingangen en slechts een uitgang.
Voor elk van deze realisaties kunnen we dan de implementatie neerschrijven onder de vorm van een Booleaanse vergelijking. Later in dit hoofdstuk zullen we dan zien hoe we gebruik kunnen maken van gemeenschappelijke delen in meerdere uitgangen om tot een eenvoudigere oplossing te komen.

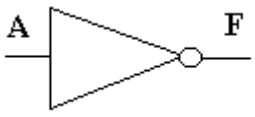
1. Wat zijn Booleaanse variabelen?

Zoals jullie reeds in de cursus C++ gezien hebben is een Booleaanse variabele een variabele die twee waardes kan aannemen, namelijk "True" of "False". De "True" implementeren we als "1" en de "False" implementeren we als "0". In de realisatie aan de hand van een elektronische schakeling komt deze "1" overeen met de voedingsspanning (5 Volt, 3.3 Volt of nog lager in de toekomst) en de "0" met de grond.

2. Bewerkingen op Booleaanse variabelen

2.1 Inversie

De inversie van een Booleaanse variabele is "0" als de Booleaanse variabele "1" is en in "1" als de Booleaanse variabele "0" is. De inversie van a wordt voorgesteld als $\overline{a}$. De hardware implementatie van de inversie gebeurt door een invertor. In een Booleaanse uitdrukking heeft de invertor steeds de hoogste prioriteit.



Figuur 1 Hardware implementatie van de Booleaanse vergelijking $F = \overline{A}$.

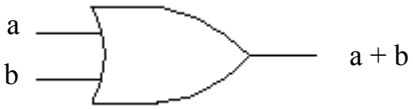
2.2 Product

Het product van Booleaanse variabelen is “0” als een (of meer) variabelen “0” is en is “1” als alle variabelen “1” zijn. De hardware implementatie van het product gebeurt door een AND poort.

Figuur 2 Hardware implementatie van de Booleaanse vergelijking a.b

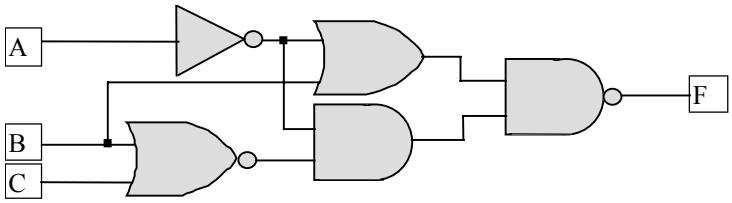
2.3 Som

De som van Booleaanse variabelen is “1” als een (of meer) variabelen “1” is en is “0” als alle variabelen “0” zijn. De hardware implementatie van de som gebeurt door een OR poort, zoals getoond wordt in Figuur 3. De som heeft in de Booleaanse algebra de laagste prioriteit. De prioriteitsregels kunnen natuurlijk steeds doorbroken worden door het plaatsen van haakjes.



Figuur 3 Hardware implementatie van de Booleaanse vergelijking a + b

Eender welke Booleaanse vergelijking kan dus onmiddellijk omgezet worden naar een hardware equivalent en omgekeerd. In Figuur 4 tonen we in een voorbeeld hoe een Booleaanse vergelijking onmiddellijk kan worden omgezet naar een hardware equivalent.



Figuur 4 Hardware implementatie van de Booleaanse vergelijking $F = \overline{(A+B)}\,\overline{(\overline{A}\ \overline{(B+C)})}$

3. Wetten en afgeleide regels

3.1 Wetten

Commutatieve wetten

$$a+b = b+a \quad (1.a)$$
$$a\,b = b\,a \quad (1.b)$$

De volgorde van Booleaanse variabelen is niet belangrijk. We kunnen bij een AND of een OR poort de variabelen omwisselen van plaats, het resultaat is hetzelfde.

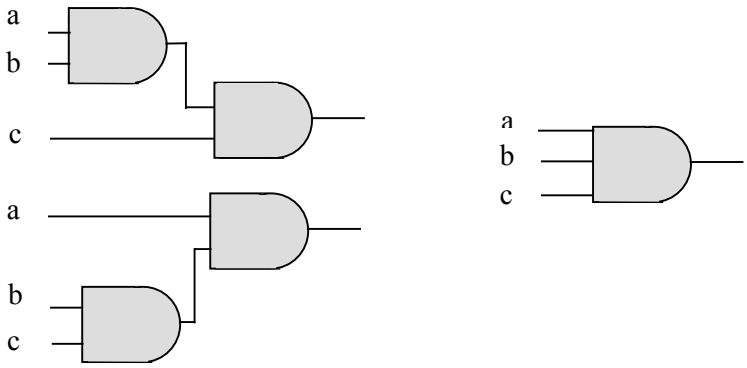
Figuur 5 Commutatieve met voor de Booleaanse optelling aan de hand van de hardware implementatie.

Figuur 6 Commutatieve wet voor de Booleaanse vermenigvuldiging aan de hand van de hardware implementatie.

Associatieve wetten

$$(a+b)+c = a+(b+c)=a+b+c \quad (2.a)$$
$$(a\,b)c = a\,(b\,c)= a\,b\,c \quad (2.b)$$

De plaats van de haakjes is onbelangrijk. Het komt er dus op neer dat je een AND met 3 ingangen kan vormen aan de hand van 2 ANDs met 2 ingangen. De volgorde waarin je de variabelen kiest is onbelangrijk. Hetzelfde geldt voor de OR poorten.



Figuur 7 Associatieve wet voor de Booleaanse vermenigvuldiging aan de hand van de hardware implementatie.

Distributiviteitsregels

$$a + b\,c = (a+b)\,(a+c) \quad (3.a)$$
$$a\,b + a\,c = a\,(b+c) \quad (3.b)$$

Zoals verder in dit hoofdstuk zal blijken zijn deze distributiviteitsregels belangrijk voor het bewerken van Booleaanse uitdrukkingen omdat ze een som van producten (linkerkant van de vergelijkingen) omzetten naar een product van sommen (rechterkant van de vergelijkingen) en omgekeerd. Deze omzettingen zijn belangrijk voor het herschrijven van Booleaanse uitdrukkingen.

Bewerking van een variabele met zijn inverse

$$a \cdot \bar{a} = 0 \quad (4.a)$$
$$a + \bar{a} = 1 \quad (4.b)$$

Het heeft dus geen zin een logische bewerking van een variabele met zijn inverse uit te voeren, het resultaat is al op voorhand gekend, en we kunnen de schakeling dus vereenvoudigen.

Bewerking met constanten

$$a + 0 = a \quad (5.a)$$
$$a\ 1 = a \quad (5.b)$$

Ook hier kunnen we de schakeling dus vereenvoudigen.

We noteren dat deze wetten gegeven zijn in paren, die we telkens #.a en #.b genoemd hebben. Inderdaad, we kunnen door alle + met * om te wisselen en alle 0 met 1 om te wisselen overgaan van de ene wet naar de andere wet. Deze eigenschap van de Booleaanse algebra noemen we de dualiteit. Dit is een handig middel om de wetten te onthouden.

3.2 afgeleide regels

Op basis van de hierboven gegeven wetten kunnen we nog een aantal andere regels in de Booleaanse afleiden. Deze afgeleide regels laten ons toe Booleaanse vergelijkingen op een snelle manier te vereenvoudigen.

Bewerking met zichzelf

$$a + a = a \quad (6.a)$$
$$a\ a = a \quad (6.b)$$

Bewerking met constanten

$$a+1 = 1 \quad (7.a)$$
$$a\ 0 = 0 \quad (7.b)$$

Onder (5) waren reeds 2 bewerkingen met constanten gedefinieerd. Hier geven we dus de 2 overige mogelijkheden.

Absorptie met zichzelf

$$a + a\,b = a \quad (8.a)$$
$$a\,(a + b) = a \quad (8.b)$$

Dubbele inversie

$$\bar{\bar{a}} = a \quad (9)$$

Voor deze afgeleide regel is de duale regel terug de regel zelf. Vandaar dat we hier slechts één regel hebben.

Absorptie met complement

$$a + \bar{a} \cdot b = a + b \quad (10.a)$$
$$a\left(\bar{a} + b\right) = a \cdot b \quad (10.b)$$

Oefening:

Leidt al deze afgeleide regels af enkel op basis van de wetten gegeven in de vorige paragraaf.

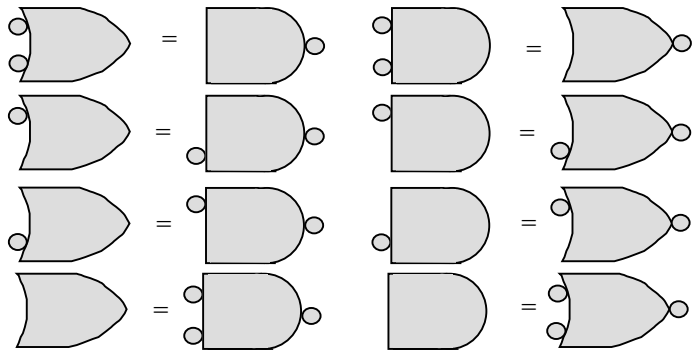
3.3 Wetten van De Morgan

Deze twee wetten zin veruit de meest belangrijke voor de elektronica. Ze laten toe om van een AND over te gaan naar een NOR [zie (11.a)] en van een OR over te gaan in een NAND [zie (11.b)]. Hierbij wordt telkens de polariteit van het ingangssignaal omgewisseld.

$$\overline{a + b} = \bar{a} \cdot \bar{b} \quad (11.a)$$

$$\overline{a \cdot b} = \bar{a} + \bar{b} \quad (11.b)$$

De hardware equivalenten van deze wetten zien er uit als in



Figuur 8 Hardware voorstellingen van de wetten van DE MORGAN. Op deze wijze kunnen inversies over logische poorten geschoven worden.

We zagen in het vorige hoofdstuk dat een CMOS schakeling met een NAND of NOR als basisvorm eenvoudiger en sneller te realiseren zijn dan in de vorm van ANDs en ORs. Deze wetten laten ons dus toe een inversie te bekomen in de laatste trap en alle andere inversie te verschuiven naar de ingangen.

Ook voor het vereenvoudigen van logische uitdrukking, waar we geen inversie intern in wensen, ook niet in de laatste trap, laten deze wetten ons toe inversies die in de uitdrukking aanwezig zijn, stapsgewijs te herwerken tot inversies aan de ingangen van de schakeling.

4. Logische vergelijkingen in SOP en POS vorm.

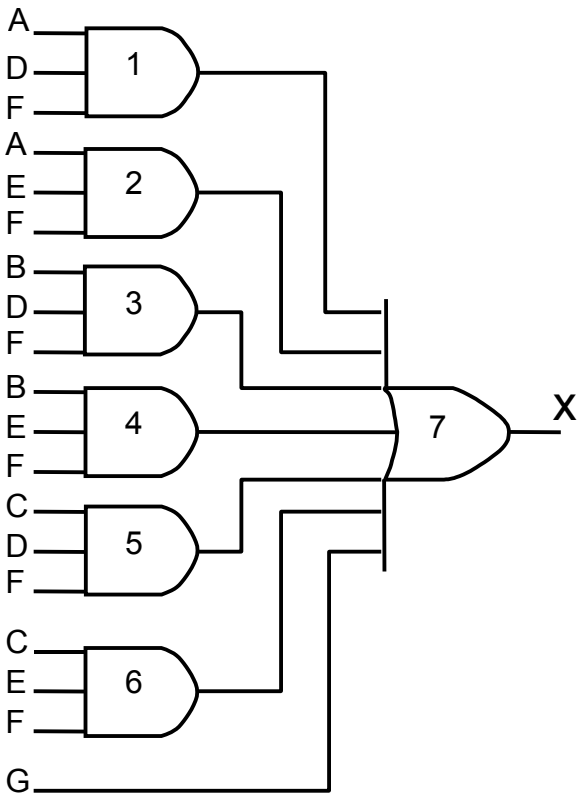
Om logische uitdrukkingen te vereenvoudigen is de vertrekbasis gewoonlijk het herschrijven van de uitdrukking in een som van producten (SOP) of een product van sommen (POS).

Om de uitdrukking te herwerken naar een van deze 2 vormen gaan we eerst de inversies herleiden tot inversies op de ingangsvariabelen. Dit doen we aan de hand van de wetten van DE MORGAN, zoals hierboven is aangegeven.

Op van een Booleaanse vergelijking een Sum Of Products (SOP) vorm te bekomen gebruiken we de distributiviteitsregels (3.a) en (3.b) systematisch tot alle haakjes verdwenen zijn.

$$(a+b)\,(a+c) = a + b\,c \qquad (3.a)$$
$$a\,(b+c) = a\,b + a\,c \qquad (3.b)$$

Op deze wijze wordt elke som die niet voorkomt op het hoogste niveau, stapsgewijs verschoven naar het hoogste niveau. Uiteindelijk zijn alle haakjes weg en blijft er maar een grote som over. Deze som is op het hoogste niveau, en komt dus overeen met één OR poort.



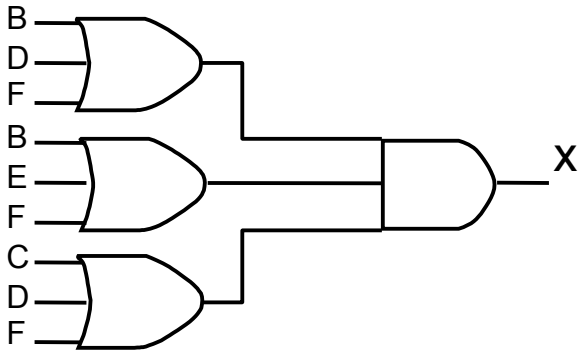
Figuur 9 *Hardware equivalent van de Sum Of Products (SOP) vorm voor de functie x* ***A D F + A E F + B D F + B E F + C D F + C E F + G***

Op van een Booleaanse vergelijking een Product Of Sums (POS) vorm te bekomen gebruiken we dezelfde distributiviteitsregels maar in de omgekeerde richting. Dit is dus zoals ze hierboven opgegeven zijn. Op deze wijze wordt elk product dat niet voorkomt op het hoogste niveau, dit is de laatste stap van de hardware implementatie, verschoven in de richting van het hoogste niveau.

Uiteindelijk komt dus daardoor elk product op het hoogste niveau, en is de enige AND poort in de schakeling een AND poort in de laatste stap.

$$a + b\,c = (a+b)\,(a+c) \qquad (3.a)$$
$$a\,b + a\,c = a\,(b+c) \qquad (3.b)$$

Dit toont aan dat, eender hoe ingewikkeld de logische vergelijking geformuleerd is, we de functie altijd in 2 niveaus kunnen realiseren. Hierbij nemen we wel aan dat we ook de inverse waarde van alle ingangssignalen ter beschikking hebben. Indien dit niet het geval is, moeten we indien nodig, nog de inversie van de ingangen berekenen. Dit leidt tot de realisatie van eender welke logische functie in maximaal 3 niveaus.



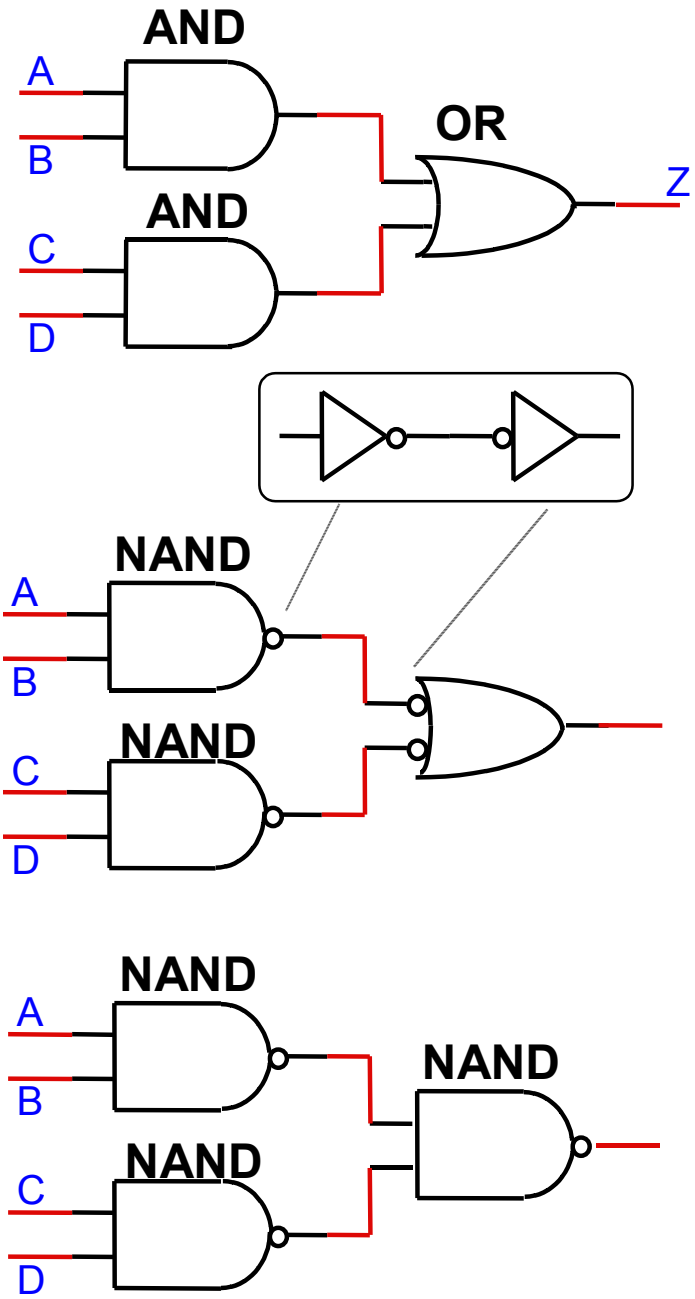
Figuur 10 Hardware equivalent van de Product Of Sums (SOP) voor de functie x, namelijk ***(B+D+F)(B+E+F)(C+D+F)***

In het geval van een SOP voeren we eerst een hele reeks AND bewerkingen uit, en van het resultaat nemen we de OR bewerking.
In het geval van een POS voeren we eerst een hele reeks OR bewerkingen uit, en van het resultaat nemen we de AND bewerking.

Deze beide methodes leiden wel tot oplossingen waarvan de termen (somtermen of producttermen) kunnen overlappen. Het is dus niet de meest eenvoudige oplossing, die kan gerealiseerd worden.
In wat verder volgt zullen we zoeken naar de eenvoudigste oplossing. Omdat de eenvoudigste oplossing ook de minste poorten gebruikt, en voor het aantal poorten het minste ingangen voorziet, zal de eenvoudigste oplossing ook de snelste zijn.

5. Voorstellen van logische vergelijkingen in uitsluitend NANDs of uitsluitend NORs

De wetten van De Morgan laten toe de AND-OR logica, die volgt uit een SOP vorm, om te zetten in een NAND-NAND vorm. Inderdaad, wanneer we de uitgang van de AND inverteren om deze te kunnen implementeren als een NAND, moeten we ook de ingangen van de OR inverteren om hetzelfde resultaat te bekomen. Vergelijking (11.b) laat toe om van een OR met geïnverteerde ingangen over te gaan naar een NAND. In de praktijk moeten we vaak nog een derde niveau toevoegen om alle functies te kunnen realiseren, omdat we vaak het omgekeerde van de ingang nodig hebben. We spreken dan van INV-NAND-NAND logica.



Figuur 11 Omzetten van AND-OR logica naar NAND-NAND logica

We kunnen ook op eenzelfde wijze van een POS-vorm overgaan naar INV-NOR-NOR logica aan de hand van vergelijking (11.a)

6. Reductie van logische vergelijkingen

Zoals we hierboven reeds besproken hebben, is het belangrijk om de logische vergelijkingen die we gebruiken te herleiden tot hun meest eenvoudige vorm. In deze sectie bespreken we een methode die ons dit toelaat. Deze methodes zijn niet enkel toepasbaar voor het ontwikkelen van elektronische schakelingen, maar ook voor het schrijven van software.

Om een schakeling met een minimum aan poorten op te bouwen wordt de logische vergelijking, die de schakeling voorstelt, best herleid tot haar eenvoudigste vorm. Deze reductie kan op verschillende manieren gebeuren, namelijk door gebruik te maken van:

a) formules uit de logische algebra;
b) Venn-diagrammen;
c) een Karnaugh-kaart;
d) de methode van Quine en McCluskey.

Het Venn-diagram is voor weinig veranderlijken duidelijk, maar omwille van de arceringen niet praktisch. De Karnaugh-kaart is een grafische methode die tot vijf veranderlijken snel tot een minimale logische vergelijking leidt. Voor meer dan vijf veranderlijken wordt de Karnaugh-kaart te omslachtig. De methode van Quine en McCluskey tenslotte is een tabellenmethode, eerder geschikt als algoritme voor een computerprogramma dat de eenvoudigste vorm van de logische vergelijking moet opzoeken.

6.1 Voorstellingswijze van logische functies

Om logische vergelijkingen systematisch aan te pakken kunnen zij in twee karakteristieke **standaardvormen** of **normaalvormen** geschreven worden, namelijk:

a) als **standaardsom** van producten;
b) als **standaardproduct** van sommen.

De termen uit deze vergelijkingen zijn producten of sommen van de veranderlijken. Bevat een term alle veranderlijken, dan spreekt men van een **normaalterm**. Een normaalterm onder de vorm van een product is een **minterm**. Een normaalterm onder de vorm van een som is een **maxterm.**

STANDAARDSOM VAN PRODUCTEN

De logische vergelijking is onder de vorm van een som van producten geschreven. In iedere term komt elke veranderlijke éénmaal voor. Indien het aantal veranderlijken n bedraagt zijn er 2^n mogelijke mintermen.

Wij bestuderen als voorbeeld de gedragstafel van een functie met drie veranderlijken en leiden er de normaalvorm uit af.

	A	B	C	Minterm	f
0	0	0	0	$m_0 = \overline{A}.\overline{B}.\overline{C}$	0
1	0	0	1	$m_1 = \overline{A}.\overline{B}.C$	0
2	0	1	0	$m_2 = \overline{A}.B.\overline{C}$	1
3	0	1	1	$m_3 = \overline{A}.B.C$	0
4	1	0	0	$m_4 = A.\overline{B}.\overline{C}$	1
5	1	0	1	$m_5 = A.\overline{B}.C$	0
6	1	1	0	$m_6 = A.B.\overline{C}$	1
7	1	1	1	$m_7 = A.B.C$	0

De logische functie f(A,B,C) kan geschreven worden als een som van dié mintermen, die de functie gelijk aan 1 maken. Aldus is

$$f(A,B,C) = \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C}$$
$$= m_2 + m_4 + m_6$$
$$= m(2,4,6)$$

hetgeen de **normaalvorm** van de functie voorstelt.

Een onvolledige normaalvorm kan volledig gemaakt worden, zoals het volgende voorbeeld laat zien.

$$f(A,B,C) = \overline{B}.C + A.\overline{C} + B.\overline{C}$$
$$= (A + \overline{A}).\overline{B}.C + (B + \overline{B}).A.\overline{C} + (A + \overline{A}).B.\overline{C}$$
$$= A.\overline{B}.C + \overline{A}.\overline{B}.C + A.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C} + \overline{A}.B.\overline{C}$$

We gebruiken hiervoor dus achtereenvolgens de wet (5.b), de wet (4.b) en de wet (3.b) We doen dit achtereenvolgens voor elke variabele die ontbreekt in elke productterm. Een product waar één variabele in ontbreekt geeft dus aanleiding tot 2 producten, en product waar 2 variabelen in ontbreken geeft aanleiding tot 4 producten, en een product waar 3 variabelen in ontbreken geeft aanleiding tot 8 producten. Nadat we dit uitgevoerd hebben kan het zijn dat we ontdekken dat we gelijke mintermen bekomen. Deze kunnen we natuurlijk vereenvoudigen aan de hand van de afgeleide regel (6.a)

STANDAARDPRODUCT VAN SOMMEN

De logische vergelijking is onder de vorm van een product van sommen geschreven. In iedere term komt elke veranderlijke éénmaal voor. Indien het aantal veranderlijken n bedraagt zijn er 2^n mogelijke maxtermen.
Wij bestuderen de gedragstafel van de functie die als voorbeeld diende voor de standaardsom van producten en leiden er de normaalvorm uit af.

	A	B	C	maxterm	f
0	0	0	0	$M_0 = A + B + C$	0
1	0	0	1	$M_1 = A + B + \overline{C}$	0
2	0	1	0	$M_2 = A + \overline{B} + C$	1
3	0	1	1	$M_3 = A + \overline{B} + \overline{C}$	0
4	1	0	0	$M_4 = \overline{A} + B + C$	1
5	1	0	1	$M_5 = \overline{A} + B + \overline{C}$	0
6	1	1	0	$M_6 = \overline{A} + \overline{B} + C$	1
7	1	1	1	$M_7 = \overline{A} + \overline{B} + \overline{C}$	0

De logische functie f(A,B,C) kan geschreven worden als het product van dié maxtermen, die de functie gelijk aan **nul** maken. Aldus is

$$f(A,B,C) = (A+B+C).(A+B+\overline{C}).(A+\overline{B}+\overline{C}).(\overline{A}+B+\overline{C}).(\overline{A}+\overline{B}+\overline{C})$$
$$= M_0 . M_1 . M_3 . M_5 . M_7$$
$$= \prod M(0,1,3,5,7)$$

hetgeen de **normaalvorm** van de functie voorstelt.
Het volgende voorbeeld laat zien hoe een onvolledige normaalvorm volledig kan gemaakt worden.

$$f(A,B,C) = (\overline{B}+\overline{C}).(A+B+C)$$
$$= (A.\overline{A}+\overline{B}+\overline{C}).(A+B+C)$$
$$= (A+\overline{B}+\overline{C}).(\overline{A}+\overline{B}+\overline{C}).(A+B+C)$$

We passen hiervoor de wet (5.a), de wet (4.a) en de wet (3.a) toe. We doen dit achtereenvolgens voor elke variabele die ontbreekt in elke som. Een som waar een variabele in ontbreekt geeft dus aanleiding tot 2 sommen, en som waar 2 variabelen in ontbreken geeft aanleiding tot 4 sommen, en een som waar 3 variabelen in ontbreken geeft aanleiding tot 8 sommen. Nadat we dit uitgevoerd hebben kan het zijn dat we ontdekken dat we gelijke maxtermen bekomen. Deze kunnen we natuurlijk vereenvoudigen aan de hand van de afgeleide regel (6.b)

De twee normaalvormen (standaardsom en standaardproduct) zijn complementair, hetgeen als gevolg heeft dat wanneer de ene vorm bekend is de andere er uit afgeleid kan worden. Zo stellen de twee voorbeelden dezelfde functie voor en is

$$f(A,B,C) = \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.B.\overline{C}$$
$$= (A+B+C).(A+B+\overline{C}).(A+\overline{B}+\overline{C}).(\overline{A}+B+\overline{C}).(\overline{A}+\overline{B}+\overline{C})$$
$$= \quad m(2,4,6) = \prod M(0,1,3,5,7)$$

Om een functie te vereenvoudigen bij middel van de methoden van Karnaugh of Quine en McCluskey moet ofwel een der normaalvormen ofwel de gedragstafel gegeven zijn.

6.2 KARNAUGH-KAART

De Karnaugh-kaart of het Karnaugh-diagram brengt de gedragstafel in beeld. Iedere mogelijke minterm wordt voorgesteld door een vierkant waarin de logische waarde 0 of 1 van de functie is weergegeven. De vierkantjes zijn gerangschikt volgens rijen en kolommen op een zodanige wijze, dat twee naburige vierkantjes mintermen voorstellen die slechts in één bit verschillen.
De Karnaugh-kaart is een ideaal middel om functies tot maximaal vijf veranderlijken te vereenvoudigen.

KARNAUGH-KAART MET 2 VERANDERLIJKEN

De gedragstafel van een functie met 2 veranderlijken telt $2^2 = 4$ rijen. De overeenstemmende Karnaugh-kaart bestaat dan ook uit 4 vierkantjes, gerangschikt volgens 2 rijen en 2 kolommen. Verschillende voorstellingswijzen om de mintermen aan te geven zijn mogelijk. De tweede schikking is echter de eenvoudigste. In de kolom of rij, die onder of naast een veranderlijke staat, heeft deze veranderlijke de logische waarde 1.

A	B	mintermen
0	0	m_0
0	1	m_1
1	0	m_2
1	1	m_3

B\A	0	1
0	$m_0 = \overline{A}.\overline{B}$	$m_2 = A.\overline{B}$
1	$m_1 = \overline{A}.B$	$m_3 = A.B$

Wij nemen als voorbeeld de functie $f(A,B)=\overline{A}.\overline{B}+A.\overline{B}+A.B$

A	B	f(A,B)
0	0	1
0	1	0
1	0	1
1	1	1

B \ A	0	1
0	1	1
1	0	1

	A	
	1	1
B	0	1

Twee naburige vakjes verschillen slechts in één variabele of twee naast elkaar liggende mintermen verschillen slechts in één bit en kunnen dus samengenomen worden, hetgeen op de volgende regel steunt

$$\overline{A}.\overline{B}+A.\overline{B}=(\overline{A}+A).\overline{B}=\overline{B}$$

De veranderlijke die verschilt mag dus wegvallen. De functie van het voorbeeld herleidt zich aldus tot

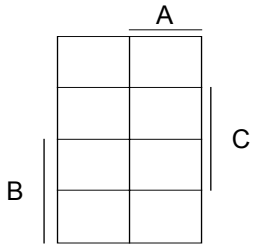
$$f(A,B)=\overline{B}+A$$

Algemeen zal men om een logische vergelijking te vereenvoudigen in de Karnaugh-kaart **zo weinig mogelijk** en **zo groot mogelijke lussen** rond de enen vormen. Deze lussen bevatten een aantal vakjes gelijk aan een macht van 2 (bv. 2, 4 of 8 vakjes). De lussen mogen elkaar overlappen, hetgeen betekent dat dezelfde minterm meerdere keren mag gebruikt worden.

KARNAUGH-KAART MET 3 VERANDERLIJKEN

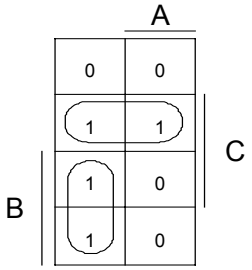
De gedragstafel van een functie met drie veranderlijken telt 8 mogelijke combinaties. De overeenstemmende Karnaugh-kaart is dan ook opgebouwd uit 8 vakjes. Zij zijn zodanig gerangschikt dat 2 horizontaal of vertikaal (niet diagonaal) aangrenzende vakjes slechts in één veranderlijke verschillen. De vakjes van de bovenste en de onderste rij worden eveneens als aangrenzend beschouwd.
We vinden hier dus de principes van de Gray-code terug

BC\A	0	1
00	$m_0 = \overline{A}.\overline{B}.\overline{C}$	$m_4 = A.\overline{B}.\overline{C}$
01	$m_1 = \overline{A}.\overline{B}.C$	$m_5 = A.\overline{B}.C$
11	$m_3 = \overline{A}.B.C$	$m_7 = A.B.C$
10	$m_2 = \overline{A}.B.\overline{C}$	$m_6 = A.B.\overline{C}$



Vertrekkend van de onderstaande gedragstafel tekenen wij de Karnaugh-kaart en nemen al de enen op in zo groot mogelijke lussen.

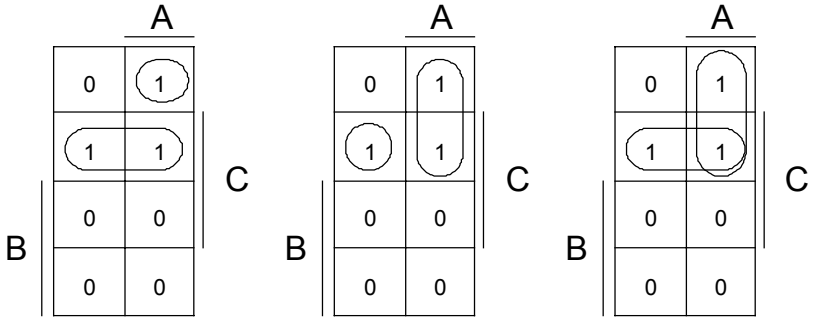
A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



De logische vergelijking die met de gedragstafel overeenkomt wordt aldus gereduceerd tot

$$f(A,B,C)=\overline{B}.C+\overline{A}.B$$

Het belang van elkaar overlappende lussen zal uit het onderstaande voorbeeld blijken.



De voorgestelde logische functie is

$$f(A,B,C)=A.\overline{B}.\overline{C}+\overline{A}.\overline{B}.C+A.\overline{B}.C$$

Zij wordt respectievelijk herleid tot

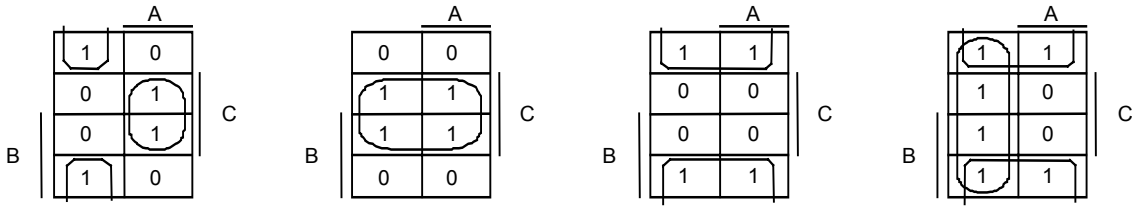
$$f(A,B,C)=\overline{B}.C+A.\overline{B}.\overline{C} \qquad \text{(eerste kaart)}$$

$$f(A,B,C)=A.\overline{B}+\overline{A}.\overline{B}.C \qquad \text{(tweede kaart)}$$

$$f(A,B,C)=A.\overline{B}+\overline{B}.C \qquad \text{(derde kaart)}$$

Op de derde kaart overlappen de lussen elkaar; zij zijn groter dan in de twee voorgaande gevallen. De derde kaart geeft de eenvoudigste oplossing, aangezien voor de implementatie eenvoudiger poorten kunnen gebruikt worden (met slechts twee ingangen).

Wij illustreren de werkwijze met nog enkele voorbeelden.



$f = \overline{A}.\overline{C} + A.C$
$= \overline{A \oplus C}$

$f = C$

$f = \overline{C}$

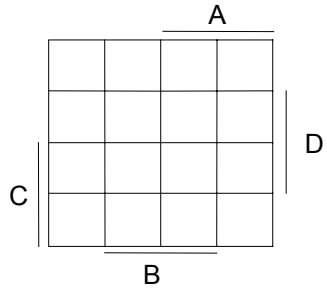
$f = \overline{A} + \overline{C}$

KARNAUGH-KAART MET 4 VERANDERLIJKEN

De gedragstafel van een functie met vier veranderlijken telt 16 mogelijke combinaties. De overeenstemmende Karnaugh-kaart is dan ook opgebouwd uit 16 vakjes.

AB / CD	00	01	11	10
00	$m_0 = \overline{A}.\overline{B}.\overline{C}.\overline{D}$	$m_4 = \overline{A}.B.\overline{C}.\overline{D}$	$m_{12} = A.B.\overline{C}.\overline{D}$	$m_8 = A.\overline{B}.\overline{C}.\overline{D}$
01	$m_1 = \overline{A}.\overline{B}.\overline{C}.D$	$m_5 = \overline{A}.B.\overline{C}.D$	$m_{13} = A.B.\overline{C}.D$	$m_9 = A.\overline{B}.\overline{C}.D$
11	$m_3 = \overline{A}.\overline{B}.C.D$	$m_7 = \overline{A}.B.C.D$	$m_{15} = A.B.C.D$	$m_{11} = A.\overline{B}.C.D$
10	$m_2 = \overline{A}.\overline{B}.C.\overline{D}$	$m_6 = \overline{A}.B.C.\overline{D}$	$m_{14} = A.B.C.\overline{D}$	$m_{10} = A.\overline{B}.C.\overline{D}$

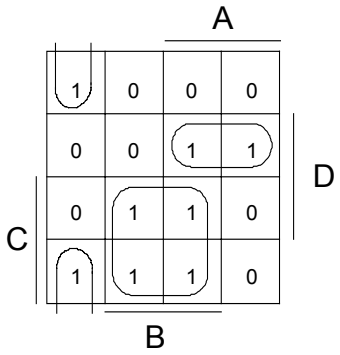
Zowel de bovenste en de onderste rij als de linker en de rechterkolom worden hier als aangrenzend verondersteld.
De onderstaande schikking van de Karnaugh-kaart verdient om haar eenvoud echter de voorkeur. De veranderlijken bij de verschillende rijen en kolommen mogen gerust van plaats verwisseld worden, zonder dat afbreuk gedaan wordt aan het principe dat twee aangrenzende vakjes slechts in één veranderlijke verschillen.



Een aantal voorbeelden illustreert de manier waarop lussen kunnen gevormd worden.

Voorbeeld 1

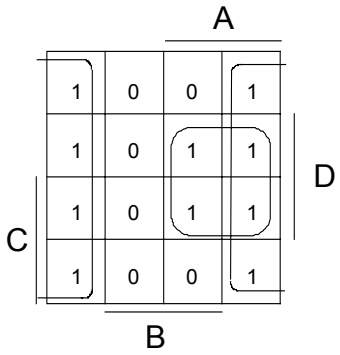
f(A,B,C,D) = Σm(0,2,6,7,9,13,14,15)



$$f(A,B,C,D) = B.C + A.\overline{C}.D + \overline{A}.\overline{B}.\overline{D}$$

Voorbeeld 2

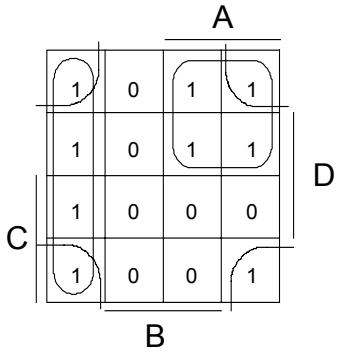
f(A,B,C,D) = Σm(0,1,2,3,8,9,10,11,13,15)



$$f(A,B,C,D) = \overline{B} + A.D$$

Voorbeeld 3

f(A,B,C,D) = Σm(0,1,2,3,8,9,10,12,13)



$$f(A,B,C,D) = \overline{A}.\overline{B} + A.\overline{C} + \overline{B}.\overline{D}$$

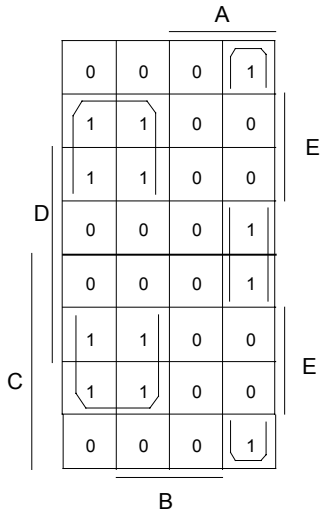
KARNAUGH-KAART MET 5 VERANDERLIJKEN

De gedragstafel van een functie met vijf veranderlijken telt 32 mogelijke combinaties. De overeenstemmende Karnaugh-kaart is dan ook opgebouwd uit 32 vakjes. De onderste helft van de kaart is het spiegelbeeld van de bovenste voor wat de veranderlijken A, B, D en E betreft. In de bovenste helft echter is de veranderlijke C=0, terwijl in de onderste helft C=1. Dit betekent dat spiegeltermen in dezelfde lus mogen samengenomen worden, aangezien zij slechts in één veranderlijke, namelijk C, verschillend zijn.

AB / CDE	00	01	11	10
000	$m_0 = \overline{A}.\overline{B}.\overline{C}.\overline{D}.\overline{E}$	$m_8 = \overline{A}.B.\overline{C}.\overline{D}.\overline{E}$	$m_{24} = A.B.\overline{C}.\overline{D}.\overline{E}$	$m_{16} = A.\overline{B}.\overline{C}.\overline{D}.\overline{E}$
001	$m_1 = \overline{A}.\overline{B}.\overline{C}.\overline{D}.E$	$m_9 = \overline{A}.B.\overline{C}.\overline{D}.E$	$m_{25} = A.B.\overline{C}.\overline{D}.E$	$m_{17} = A.\overline{B}.\overline{C}.\overline{D}.E$
011	$m_3 = \overline{A}.\overline{B}.\overline{C}.D.E$	$m_{11} = \overline{A}.B.\overline{C}.D.E$	$m_{27} = A.B.\overline{C}.D.E$	$m_{19} = A.\overline{B}.\overline{C}.D.E$
010	$m_2 = \overline{A}.\overline{B}.\overline{C}.D.\overline{E}$	$m_{10} = \overline{A}.B.\overline{C}.D.\overline{E}$	$m_{26} = A.B.\overline{C}.D.\overline{E}$	$m_{18} = A.\overline{B}.\overline{C}.D.\overline{E}$
110	$m_6 = \overline{A}.\overline{B}.C.D.\overline{E}$	$m_{14} = \overline{A}.B.C.D.\overline{E}$	$m_{30} = A.B.C.D.\overline{E}$	$m_{22} = A.\overline{B}.C.D.\overline{E}$
111	$m_7 = \overline{A}.\overline{B}.C.D.E$	$m_{15} = \overline{A}.B.C.D.E$	$m_{31} = A.B.C.D.E$	$m_{23} = A.\overline{B}.C.D.E$
101	$m_5 = \overline{A}.\overline{B}.C.\overline{D}.E$	$m_{13} = \overline{A}.B.C.\overline{D}.E$	$m_{29} = A.B.C.\overline{D}.E$	$m_{21} = A.\overline{B}.C.\overline{D}.E$
100	$m_4 = \overline{A}.\overline{B}.C.\overline{D}.\overline{E}$	$m_{12} = \overline{A}.B.C.\overline{D}.\overline{E}$	$m_{28} = A.B.C.\overline{D}.\overline{E}$	$m_{20} = A.\overline{B}.C.\overline{D}.\overline{E}$

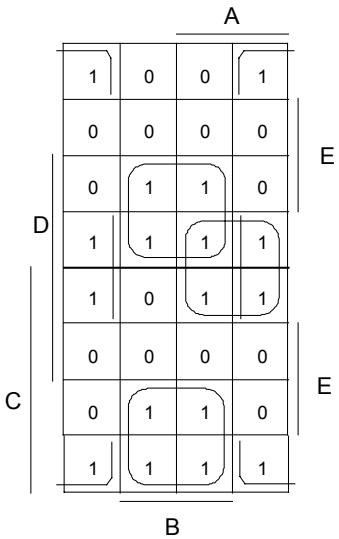
Met een tweetal voorbeelden illustreren we de manier waarop lussen kunnen gevormd worden.

Voorbeeld 1 f(A,B,C,D,E) = Σm(1,3,5,7,9,11,13,15,16,18,20,22)



$f(A,B,C,D,E) = \overline{A}.E + A.\overline{B}.\overline{E}$

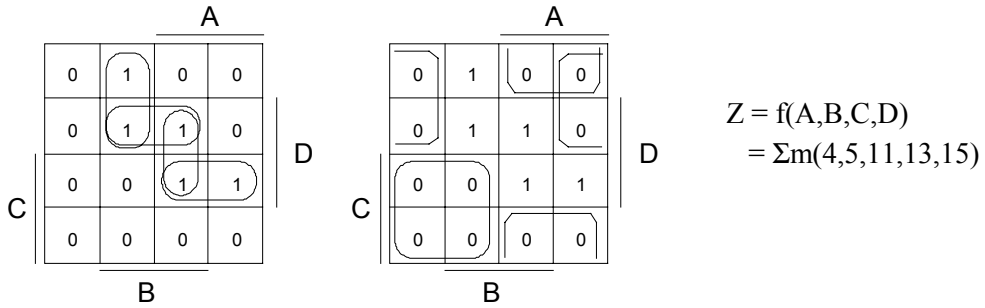
Voorbeeld 2 f(A,B,C,D,E) = Σm(0,2,4,6,10,11,12,13,16,18,20,22,26,27,28,29,30)



$f(A,B,C,D,E) = A.D.\overline{E} + B.C.\overline{D} + B.\overline{C}.D + \overline{B}.\overline{E}$

KARNAUGH-KAART MET MAX-TERMEN

Het kan soms eenvoudiger zijn in de Karnaugh-kaart nullen samen te nemen om aldus minder lussen te bekomen. De termen die de functie nul maken zijn echter maxtermen M. Zij geven aanleiding tot een functie die voorgesteld wordt door een product van sommen. Het volgende voorbeeld zal dit verduidelijken.



$Z = f(A,B,C,D)$
$= \Sigma m(4,5,11,13,15)$

Het samen nemen van nullen levert, geschreven onder de somvorm, de inverse op van de gezochte functie, namelijk

$$\overline{Z} = \overline{B}.\overline{C} + A.\overline{D} + \overline{A}.C$$
$$= \overline{\overline{B} + \overline{C}} + \overline{\overline{\overline{A}} + D} + \overline{A + \overline{C}}$$

Na inversie en toepassing van De Morgan krijgen wij

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

hetgeen de functie als een product van sommen of maxtermen voorstelt. De maxtermen zijn samengesteld uit de complementen van de veranderlijken die binnen de lussen vallen.

Het vormen van lussen met enen levert in het geval van het bovenstaande voorbeeld voor de functie een som van mintermen, zodat

$$Z = \overline{A}.B.\overline{C} + B.\overline{C}.D + A.C.D$$

De beide oplossingen zijn algebraïsch **niet** (steeds) gelijk aan elkaar. Nochtans leveren zij combinatorisch hetzelfde resultaat. De keuze tussen de vereenvoudiging met mintermen of met maxtermen hangt in hoge mate af van de poorten die voor de implementatie zullen gebruikt worden. Zo leent de oplossing met mintermen zich eerder voor verwezenlijking met NAND-poorten, terwijl de oplossing met maxtermen eerder geschikt is voor gebruik van NOR-poorten.

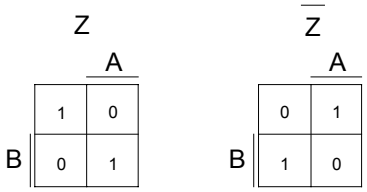
Opmerking

Een minterm dankt zijn naam aan het feit dat hij in de Karnaugh-kaart de kleinste lus van enen voorstelt. Een maxterm daarentegen is volgens De Morgan het complement van een minterm en stelt dus een maximum aan termen, namelijk alle andere mintermen voor.

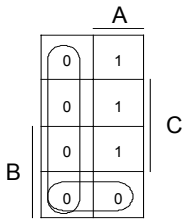
COMPLEMENTERING

Om het complement van een functie (de inverse functie) te bekomen zou men in de Karnaugh-kaart alle enen door nullen en vice-versa kunnen vervangen, zoals in het volgende eenvoudig voorbeeld is gebeurd.

A	B	Z	$\overline{Z}$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	0

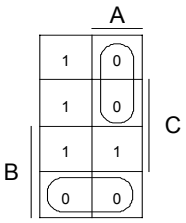


Men kan echter ook op de eerste kaart blijven verder werken, gewoon de nullen samennemen en de vereenvoudiging doorvoeren. De bekomen som van producten stelt dan wel de inverse of complementaire functie voor. Wij illustreren dit aan de hand van een paar voorbeelden.



$$\overline{Z} = \overline{A} + B.\overline{C}$$

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



$$\overline{Z} = A.\overline{B} + B.\overline{C}$$

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

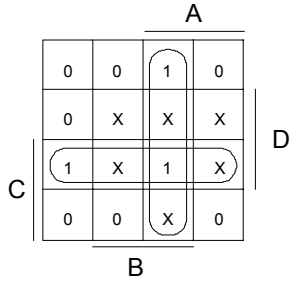
DON'T CARE

Soms is de functiewaarde voor bepaalde combinaties van de ingangsveranderlijken onbelangrijk of kunnen bepaalde combinaties zich zelfs niet voordoen. In deze gevallen mogen in de Karnaugh-kaart de overeenkomstige vakjes zowel met een 0 als met een 1 opgevuld worden. Wij verkiezen echter het teken X en noemen dit een "don't care". Nadien kan dan X als een 0 of een 1 geïnterpreteerd worden om ons toe te laten zo weinig mogelijk en zo groot mogelijke lussen te vormen.

In het onderstaande voorbeeld worden de X-tekens als een 1 beschouwd binnen de getekende lussen. Daarbuiten doen zij dienst als nullen.

$$f(A,B,C,D) = \Sigma m(3,12,15) \rightarrow 1$$

$$= \pi M(0,1,2,4,6,8,10) \rightarrow 0$$



$$f(A,B,C,D) = A.B + C.D$$

6.3 QUINE EN MCCLUSKEY

Met meer dan vijf veranderlijken is de methode van de Karnaugh-kaart niet meer praktisch.
De methode van Quine en Mc-Cluskey dringt zich dan op. Zij is, in tegenstelling tot deze van Karnaugh, een tabellenmethode die uit onmisbare termen een vereenvoudigde functie afleidt. De grondgedachte, waarop de werkwijze steunt, is dezelfde als bij Karnaugh; termen die slechts in één veranderlijke verschillen worden samengenomen.

De methode van Quine en McCluskey volgt een welbepaald algoritme, zodat de methode kan geprogrammeerd worden. Het spreekt vanzelf dat bij gebruik van een computer het aantal veranderlijken geen rol meer speelt. De methode wordt normaal enkel toegepast worden op vergelijkingen die onder de vorm van een som van producten gegeven zijn.

Het algoritme omvat drie delen:

a) herschikken van de gedragstafel
b) opzoeken van onmisbare termen
c) opzoeken van absoluut onmisbare termen

Wij demonstreren nu de methode aan de hand van een voorbeeld. De te vereenvoudigen functie is

$f(A,B,C,D) = \Sigma m(2,3,4,5,9,10,11,13)$

De mintermen zijn:

2	0010	$\overline{A}.\overline{B}.C.\overline{D}$
3	0011	$\overline{A}.\overline{B}.C.D$
4	0100	$\overline{A}.B.\overline{C}.\overline{D}$
5	0101	$\overline{A}.B.\overline{C}.D$
9	1001	$A.\overline{B}.\overline{C}.D$
10	1010	$A.\overline{B}.C.\overline{D}$
11	1011	$A.\overline{B}.C.D$
13	1101	$A.B.\overline{C}.D$

Herschikken van de gedragstafel

De volgorde van de termen wordt zodanig gewijzigd dat termen met hetzelfde aantal enen gegroepeerd worden.

Groep 1	2	0010
	4	0100
---	---	---
Groep 2	3	0011
	5	0101
	9	1001
	10	1010
---	---	---
Groep 3	11	1011
	13	1101

Opzoeken van de onmisbare termen

Wij vergelijken nu termen uit verschillende groepen. Indien twee termen slechts in één veranderlijke verschillen worden zij vervangen door één term waarin het teken (-) de verdwenen veranderlijke vervangt. Samen met de niet samen te nemen termen krijgen wij aldus een nieuw stelsel termen, dat eveneens in groepen kan ingedeeld worden. Het bekomen stelsel leent zich dan opnieuw tot vereenvoudiging. De onmisbare termen zijn dan deze die uiteindelijk niet meer kunnen worden samengenomen. Het is aan te raden de gebruikte termen te merken. Wij hebben hier de onmisbare termen met het asterisk-teken (*) gemerkt.

```
  (2,3)          001-                    (2,3,10,11)        -01-   * e
  (2,10)         -010
  (4,5)          010-   * a
-------------------------------
 (3,11)          -011
 (5,13)          -101   * b
 (9,11)          10-1   * c
 (9,13)          1-01   * d
(10,11)          101-
```

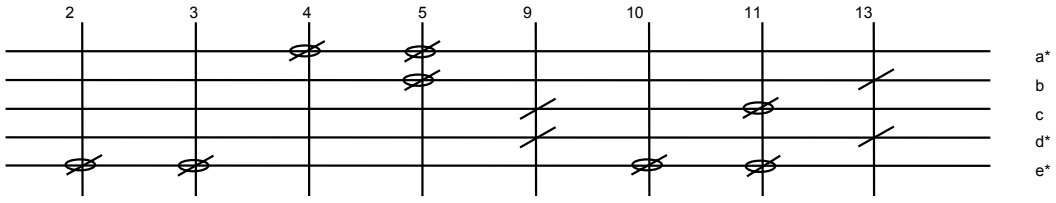
Voor het ogenblik mogen wij voor de functie reeds schrijven

$$f(A,B,C,D) = a+b+c+d+e = \overline{A}.B.\overline{C} + B.\overline{C}.D + A.\overline{B}.D + A.\overline{C}.D + \overline{B}.C$$

Deze uitdrukking kan misschien nog vereenvoudigd worden. Daarom gaan wij over tot de derde stap in het algoritme.

Opzoeken van de absoluut onmisbare termen

Wij construeren een rooster waarop is aangeduid met het slash-teken (/) welke mintermen in de onmisbare termen voorkomen.



Zo zien wij dat de termen 2, 3 en 4 enkel in a en e voorkomen. Dit maakt de termen a en e absoluut onmisbaar. Wij merken dan alle mintermen in a en e met het teken (O). Uiteindelijk blijven dan enkel de termen 9 en 13 over. Deze zitten beide in d, zodat ook d absoluut onmisbaar wordt.

De functie onder haar eenvoudigste vorm is aldus

$$f(A,B,C,D) = a+d+e = \overline{A}.B.\overline{C} + A.\overline{C}.D + \overline{B}.C$$

De vijf onmisbare termen zijn vervangen door drie absoluut onmisbare termen.

Bijkomend uitgewerkt voorbeeld

De functie van 5 inputs A, B, C, D, en E die waar wordt als de variabelen zijn zoals ze in de onderstaande tabel aangegeven zijn. Deze combinaties van bits drukken we ook in decimale vorm uit. (zie hoofdstuk 1 paragraaf 2.2)

EDCBA	decimaal
00000	0
00110	6
01000	8
01010	10
01100	12
01110	14
10001	17
10011	19
10100	20
10110	22
11001	25
11011	27
11100	28
11110	30

We groeperen nu de blokken per aantal enen

EDCBA	decimaal
00000	0
01000	8
00110	6
01010	10
01100	12
10001	17
10100	20
01110	14
10011	19
10110	22
11001	25
11100	28
11011	27
11110	30

Alles van 1 blok met alles van het volgende blok combineren en aftekenen met een * als het gelukt is.

EDCBA	decimaal		EDCBA	decimaal
00000	0	*	0-000	0,8
01000	8	*		
00110	6			
01010	10			
01100	12			
10001	17			
10100	20			
01110	14			
10011	19			
10110	22			
11001	25			
11100	28			
11011	27			
11110	30			

we doen zo verder

EDCBA	decimaal		EDCBA	decimaal
00000	0	*	0-000	0,8
01000	8	*	010-0	8,10
			01-00	8,12
00110	6	*		
01010	10	*	0-110	6,14
01100	12	*	-0110	6,22
10001	17	*	01-10	10,14
10100	20	*	011-0	12,14
			-1100	12,28
01110	14	*	100-1	17,19
10011	19	*	1-001	17,25
10110	22	*	101-0	20,22
11001	25	*	1-100	20,28
11100	28	*		
			-1110	14,30
11011	27	*	1-011	19,27
11110	30	*	1-110	22,30
			110-1	25,27
			111-0	28,30

Alle eerste niveaus blijken gecombineerd te kunnen worden. We gaan een niveau verder.

EDCBA	dec		EDCBA	dec		EDCBA	dec	
00000	0	*	0-000	0,8	f			
01000	8	*	010-0	8,10	*	01--0	8,10,12,14	a
			01-00	8,12	*			
00110	6	*						
01010	10	*	0-110	6,14	*	--110	6,14,22,30	b
01100	12	*	-0110	6,22	*	-11-0	12,14,28,30	c
10001	17	*	01-10	10,14	*	1-0-1	17,19,25,27	d
10100	20	*	011-0	12,14	*	1-1-0	20,22,28,30	e
			-1100	12,28	*			
01110	14	*	100-1	17,19	*			
10011	19	*	1-001	17,25	*			
10110	22	*	101-0	20,22	*			
11001	25	*	1-100	20,28	*			
11100	28	*						
			-1110	14,30	*			
11011	27	*	1-011	19,27	*			
11110	30	*	1-110	22,30	*			
			110-1	25,27	*			
			111-0	28,30	*			

Per term die we in kolom 3 kunnen vormen, kunnen we nu 4 termen in kolom 2 een * geven.

We zien dat enkel het eerste element van kolom 2 geen * gekregen heeft. We zien dat we niet verder meer kunnen samennemen. De overblijvende elementen noemen we van a tot f, we plaatsen ze in een tabel samen met hun decimale waarden.

	0	6	8	10	12	14	17	19	20	22	25	27	28	30
a *			X	X	X	X								
b *		X				X				X				X
c					X	X							X	X
d *							X	X			X	X		
e *									X	X			X	X
f *	X		X											
	X	X	X	X	X	X	X	X	X	X	X	X	X	X

We duiden eerst de essentiële implicanten aan, door te zoeken naar de kolomen waar er maar één enkele X staat. C is geen essentiële implicant omdat alle mintermen die door C gerealiseerd worden ook door een andere implicant gerealiseerd worden. We kunnen c dus weglaten.